Enriching Kademlia by Partitioning

João Monteiro, Pedro Ákos Costa, João Leitão
NOVA LINCS & DI/FCT/NOVA University of Lisbon,
Lisboa, Portugal
jmp.monteiro@campus.fct.unl.pt pah.costa@campus.fct.unl.pt
jc.leitao@fct.unl.pt

Alfonso de la Rocha
Protocol Labs,
alfonso@protocol.ai

Yiannis Psaras
Protocol Labs,
yiannis@protocol.ai

Abstract

Decentralizing the Web is becoming an increasingly interesting endeavor that aims at improving user security and privacy as well as providing guaranteed ownership of content. One such endeavor that pushes towards this reality, is Protocol Labs' Inter-Planetary File System (IPFS) network, that provides a decentralized large scale file system to support the decentralized Web. To achieve this, the IPFS network leverages the Kademlia DHT to route and store pointers to content stored by network members (i.e., peers). However, due to the large number of network peers, content, and accesses, the DHT routing needs to be efficient and quick to enable a decentralized web that is competitive.

In this paper, we present work in progress that aims at improving the Kademlia DHT performance through the manipulation of DHT identifiers by adding prefixes to identifiers. With this, we are able to bias the DHT topological organization towards locality (which can be either geographical or applicational), which creates partitions in the DHT and enables faster and more efficient query resolution on local content. We designed prototypes that implement our proposal, and performed a first evaluation of our work in an emulated network testbed composed of 5000 nodes. Our results show that our proposal can benefit the DHT look up on data with locality with minimal overhead.

1 Introduction

With the popularization of blockchain technology [28, 32] there has been an increased interested in peer-to-peer technology as a way to realize a novel decentralized web vision [1]. The decentralized web aims at decentralizing control from centralized infrastructures and entities (i.e., the cloud and its providers) towards end-users. Protocol Labs[1] has been push-

[1]`https://protocol.ai`

ing this endeavor with products such as IPFS [3] and FileCoin [4], that aim at building a large community of users on a large scale decentralized network that share content and build distributed applications.

In this paper, we are interested in the IPFS network in particular. IPFS is a community driven peer-to-peer distributed file system that aims to connect computing devices through a shared file system. Currently, IPFS hosts a multitude of content, that ranges from full web sites, such as Wikipedia, to images and other user public files. To support the operation of IPFS, a distributed hash table (DHT) – Kademlia [17] – is used to locate and store content pointers within the IPFS network. To this end, peers and content are encoded with an immutable identifier that Kademlia leverages to organize peers and store and find content.

The Kademlia DHT is widely popular, having been previously employed in BitTorrent [5] however, in the context of IPFS, content resolution (i.e., locating and retrieving content) can be extremely slow, sometimes even reaching latencies higher than $2,5$ hours [6]. The reasons for this phenomenon are still being investigated by the IPFS team and collaborators. Nevertheless, one possible reason can be due to a known challenge of peer-to-peer overlay networks – topology mismatch [14, 24]. Topology mismatch occurs when the topological organization of peers does not match the physical network topology, generating logical paths among peers that are suboptimal at the physical layer. This challenge can easily lead to higher resolution latencies in DHTs [23], as DHTs organize peers based on their identifiers, which are generated from a uniform distribution (e.g., a SHA-256 of the IP and Port of the peer) that does not encode any locality property (e.g., geo-location of the peer, applications that the peer runs, etc).

To address this challenge, in this paper we aim at presenting effective solutions that will incur in minimal operational overhead in modifications to IPFS. To this end, we present and evaluate a scheme to bias the DHT topology towards locality. In more detail, our scheme is rooted on the idea that by adding prefixes to the DHT identifier, one can organize peers that have the same prefix closely in the DHT, enabling faster resolutions for local content. The prefix can encode geographical regions, applications, or any other arbitrary proximity criteria among peers. Leveraging this insight, we present two solutions that partition the Kademlia DHT based on identifier prefixes. We present a first solution that creates virtual partitions in the DHT by adding the prefix to the identifier without further modifications to the Kademlia protocol. We name this scheme *Soft Partitioning*. The second solution creates physical partitions in the DHT, having nodes of each prefix participating in a segregated DHT. To enable communication among nodes across different prefix domains, our scheme leverages an additional protocol that stores and retrieves contact points for each known prefix (i.e., sub DHT). We name this scheme *Hard Partitioning*. Our preliminary evaluation shows that both schemes can achieve faster content resolution on data that presents locality (i.e., belongs to the same prefix/region/application) without adding significant overhead to the remainder of queries of the system.

The remainder of this paper is organized as follows: Section 2 discusses techniques to perform DHT topological optimizations; Section 3 presents our solutions in more detail; Section 4 details our experimental work, that leverages an emulated network testbed composed of 5000 peers; and finally, Section 5 concludes the paper with future directions.

2 Related Work

Distributed Hash Tables (DHTs) are crucial for the operation of large scale systems, specially for the ones that require users to find each other. A DHT, in short, provides a decentralized method of mapping an identifier (of a resource) to one or more nodes in the network. This resource is information that is stored by some peer(s) in the network. To enable this, DHT protocols build a structured overlay network on top of a logical key space that allows to find any key (i.e., identifier) by using application-level routing mechanisms. Examples of DHTs include, Chord [27] that operates with a consistent hashing [12] mechanism; Tapestry [30] and Pastry [26] that leverage a Plaxton Mesh [21] key space and routing system; SkipNet [10] that uses two different key spaces that are ordered lexicographically; and Kademlia [17], the protocol we are interested in this paper, that uses the XOR distance of keys for routing. However, in the operation of these protocols, keys are attributed based on a uniformly random distribution (e.g., the hash of an IP address), which does not encode any type of locality. Previous works have addressed this. These works can be divided in three categories: *i*) *Peer Selection*; *ii*) *Coordinate System Transformation*; and *iii*) *Identifier Manipulation.*

Peer Selection. To enable routing in a DHT, each peer keeps information about other peers with properties that usually encode logarithmic jumps in the DHT key space (i.e., each hop among peers should cut the distance to a target in half). However, the peers stored locally can be suboptimal due to physical network constraints, and thus lead to high latencies when routing requests. Because DHTs are designed to accommodate large numbers of peers in the system, there are multiple peers that can provide (close to) logarithmic jumps in the DHT. Peer selection means choosing a peer that optimizes this jump through some additional property. The most common property to optimize is latency among peers, which was studied in CAN [23] and Coral [7]. Both works present solutions to construct DHT topologies that can be optimized towards latency among peers, by favoring to store information about lower latency peers. However, to enable this, active latency measurements need to be done, which can highly influence the network traffic volume. Furthermore, and due to latency being a dynamic metric (i.e., it fluctuates over time), such techniques can generate unwanted instabilities and be difficult to tune.

Coordinate System Transformation. Alternatively, other works such as GeoPeer [2], NL-DHT [25], and Geodemlia [9], explore the transformation of an existing (physical) coordinate system (e.g., GPS, logical Cartesian space) into a logical key space that can be leveraged by a DHT protocol, that maintains the locality properties of the original coordinate system. In the case of GeoPeer, peers leverage the original coordinate system to organize themselves towards locality based on delaunay triangulations [15]. NL-DHT, on the other hand, propose the use of a modified Hilbert curve [18] method to transform a three dimensional space, that represents the location of a peer in a geographic space, into a single dimension DHT key space. Geodemlia follows a simpler approach, encoding the geographic space in a circle, and dividing the circle into regions. The management of these regions is similar to that of Kademlia. Unfortunately, these techniques can become highly limiting and complex, as only a single locality property can be encoded that depends on a physical global coordinate system.

Identifier Manipulation. Lastly, works such as Globase.KOM [13], LDHT [29], and the works presented in [31] and [11], focus on manipulating peer and resource identifiers (i.e., DHT keys). These works separate the key in two parts, a global and a local part. Usually, the global part is encoded by the most significant bits of the key, can vary in length, and is used to encode a geographical region. The local part of the key identifies the peer and is a unique identifier generated randomly. In the case of Globase.KOM, peers organize themselves in a hierarchical tree structure. In this solution, interior peers (of the tree) are representatives of the geographic region, and are identified by a key that encodes a global identifier and a local identifier. Leaf peers (of the tree) are only identified by a local identifier. In LDHT all peers are identified by a key with a global and a local part. The global part is based on the Autonomous System Network (ASN) of the peer, while the local part is based on the IP address of the peer. In [31] the authors propose to have the global part of the key encode multiple encompassing geographical regions (in a hierarchy). In [11] the authors propose the use of landmark nodes, that do not participate in the DHT, to compute the global part of keys. The techniques used in these solution share the same insights as the ones employed by our approach, however, these solutions mostly consider geographical information and strict key designs. Our mechanisms, although similar in concept, is more general and flexible, and thus easier to integrate in the IPFS system.

Our mechanisms group nodes that share the most significant bits of their identifiers by adding prefixes that encode some locality property (geographical or applicational). There are adjacent works that also group peers that share some property. This is the case of the work in [8] that proposes an algorithm for peers to self organize according to some order. Furthermore, the works in [16] and [19] build DHTs with groups of peers that encode the same interval in the DHT key space, mostly as a failover and load balancing mechanism.

3 Proposed Solution

Our proposals aim at improving the DHT resolution performance for queries with locality. Locality here can mean geographic, applicational, and/or logical proximity. To this end, we leverage ideas from previous work that performs identifier manipulation to bias the DHT organization towards locality that is encoded in the identifier of peers and resources. As such, we prepend to the keys used in the Kademlia DHT a prefix that encodes locality. Due to the natural operation of Kademlia, and the way peers are organized among themselves, by prepending locality identifiers, peers organize themselves in groups that represent each locality identifier. In the following, we present a brief description of the Kademlia protocol to support the description of our two proposed solutions that bias Kademlia for locality: *Soft Partitioning* and *Hard Partitioning*.

3.1 Kademlia

Kademlia operates over an m bit key space (in IPFS the value for m is the output length of a SHA-256), and uses the XOR distance metric to order keys. Each peer holds a routing table (named k-buckets) composed of multiple lists (or buckets), of fixed maximum size of k (the value for k used in IPFS is 20). Each bucket holds peers that can decreasingly cut the distance to a target in the DHT from the local peer. For example, bucket 0 holds peers that can perform jumps that cover (i.e., cut the distance to a target in) half of the key space, bucket 1 holds peers that can perform jumps that cover a quarter of the key space, and so on. Peers are initialized with a single bucket that is divided once it gets full. Only the bucket that covers the least amount of space (i.e., the last bucket) is divided. This means that a peer will always know more peers which are closer in the key space.

Kademlia employs a generic method to locate resources in the DHT, having optimizations for finding peers (FindNode operation) and stored values (FindValue operation). In short, the method works as follows: for any arbitrary key, the peer contacts the k peers that it knows of that are closest to the key. These peers will return the k peers that they known of that are closest to the key. The original peer continues to contact the k closest peers to the key gathered from the responses, until it has contacted all the closest peers or has found the resource that is mapped by the key. Storing values in the DHT is similar. The operation first finds the k closest peers to the to be stored key, and stores the value in those k peers. Kademlia can be configured with two additional parameters that control the parallelism of query resolution. These parameters are α, that controls how many parallel messages can be sent when locating a key, and β, that controls how many responses the protocol has to wait before performing the next round of messages for query resolution. In IPFS these values are parameterized as: $\alpha = 10$ and $\beta = 3$.

3.2 Sotf Partitioning

Our soft partitioning solution creates virtual partitions in the Kademlia DHT. This is a consequence of adding the locality prefix to peer and content identifiers and due to how the XOR distance metric in Kademlia operates. The addition of prefixes modify the XOR distance of any two keys that do not share a prefix, making keys that do not share the same prefix to always have higher distances between them than keys that share the prefix. This causes Kademlia's k-buckets to store more peers that share the prefix (as these are closer), hence creating virtual partitions in the DHT, as peers that share the same prefix will have more connections among them.

To store content with this scheme, the generated content identifier must also be prefixed. To preserve locality, the content is prefixed with the prefix of the peer that publishes the content. To find content, peers must also have knowledge of the content's prefix. This however, is a process that occurs out-of-band and is not a concern of the protocol.

3.3 Hard Partitioning

The hard partitioning solution takes the locality concept a step further by creating physical partitions in the Kademlia DHT, effectively dividing the original DHT into smaller disjoint DHTs, each encoded/indexed by a different locality prefix. This allows for smaller average number of hops to be needed to route queries with locality (i.e., queries for resources that share the prefix of the peer performing the query). For non-local queries (i.e., to resource that do not share the prefix of the peer performing the query) an external service to the DHT is used. We dubbed this service the *indexer service* that maps prefixes to a list of peers that have that prefix. This allows peers to find contact points for other prefixes in other DHTs. The indexer service can be implemented in a centralized way, having a single instance know all contact points, or in a decentralized way, where multiple instances know different contact points. The indexer service is updated by having peers contact the service periodically and with a configurable probability.

To avoid having peers constantly contacting the indexer service, peers maintain a small cache with contacts of frequently queried partitions (i.e., frequently contacted DHTs with different prefixes). As such, anytime a peer performs a query to a remote partition, it firsts tries to use the contacts it has in its cache. If these fail to respond, the peer contacts the indexer service for more contact points. Contacts are evicted from cache when they fail to respond or after configurable time-to-live (TTL) expires.

In this scheme, content is only stored in the local partition, as it is expected to be accessed mostly by peers in the local partition. However, if some content becomes overly popular, it can be replicated to other partitions. Nevertheless, in IPFS, we expect these to happen mostly by republishing the content in a different partition.

3.4 Discussion

In this work we study the impact of these two solutions over the base Kademlia protocol. Both solutions aim at improving Kademlia's query resolution for access patterns with locality. The soft partitioning solution is a solution that can be easily integrated with IPFS by simply changing peer and content identifiers to also encode locality properties and enabling peers and applications to leverage locality identifiers on the DHT. On the other hand, the hard partitioning solution requires a larger integration effort with IPFS, as the DHT protocol requires modifications and a new service (the indexer service) has to be developed and deployed. Nevertheless, the hard partitioning solutions provides an additional advantage by enabling the creation of jumps over large gaps in the key spaces through contaict points to remote partitions.

4 Evaluation

We have conducted an experimental evaluation of both our solutions. To this end, we have implemented simple prototypes in the Go programing language of both our solutions and the Kademlia protocol. The Kademlia protocol is implemented as per the description in the original paper, using UDP communication. We have additionally used code in libp2p [22] (that is also used to implement IPFS) that implements the XOR distance. Our solutions are implemented by extending the Kademlia protocol prototype.

We execute our prototypes in a network of 5000 peers with emulated latencies among them. This network was generated with the help of a graph analysis tool [20]. In this network, peers that are close by have very low latency among them, and some peers that are farther apart also have low latency among them. This network tries to emulate latencies experienced in the Internet, where peers are expected to have low latency links to other peers that are in the same ISP, and can occasionally have a low latency link to a peer outside their ISP. After this, we calculated partitions of different sizes (3, 10, and 100) on the generated network based on the proximity of peers in the network. Table 1 shows the properties of the generated network and the calculated partition sizes. Note that the average latency

Table 1: Average network latency and partition sizes.

Graph	Local (ms)	Remote (ms)	Partition Size		
			Avg	Min	Max
3 Partitions	*275.52*	*495.77*	*1667*	*1254*	*1519*
10 Partitions	*145.03*	*449.68*	*500*	*358*	*744*
100 Partitions	*43.82*	*418.26*	*50*	*26*	*102*

decreases with the higher number of partitions as the partitions are smaller and are more tighly packed in the network.

To effectively emulate the network we leverage Docker containers. We execute 100 Docker containers spread evenly across 20 servers in the Grid5000 platform [2], each with an Intel Xeon Gold 5220 CPU, with 18 cores, and 96 GiB of memory. Each container executes 50 independent instances of our prototypes. Each instance is identified by the IP of the container and the assigned UDP port. Latencies are emulated with the Linux `tc` tool with a rule for each pair of instances. In the following we discuss the performance evaluation with this network.

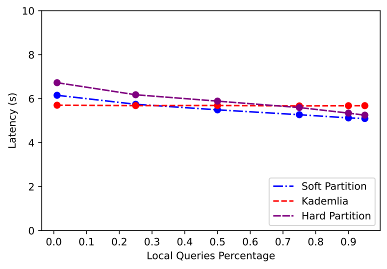
4.1 Performance Evaluation

Our evaluation is centered on the average latency of FindNode and FindValue operations. Furthermore, we configured all solutions with the following parameters: $k = 5$, $\alpha = 3$, $\beta = 2$. These values are substantially lower than those used in IPFS. This is due to the scale of our emulation being smaller than the IPFS network (otherwise most queries would be resolved on the first hops), and to operational limitations (to avoid overloading servers with too many messages). In our experiments for the hard partitioning solution, we use a single indexer service that is hosted by one of the peers of the network. The indexer service is configured to hold five contacts for each partition, and peers are configured to update their contact entry on the service every 20 seconds with a probability of 30%. Furthermore, hard partitioning peers also cache 3 peers per partition that have been used with a TTL of 60 seconds.

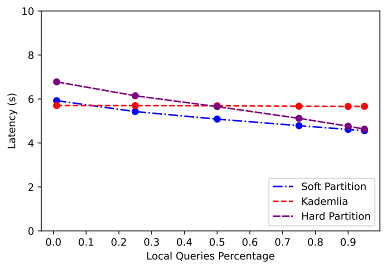
Experiments run for an average of 10 minutes. The first 2 minutes of the experiment are used as a grace period for all peers to join the network. In experiments with the FindNode operation, peers perform 5 minutes for queries of a our workload. The remainder of the time is used for queries to finish gracefully. In experiments with the FindValue operation, each peer stores five values in the DHT and waits for another minute grace period. This is followed by 5 minutes of queries of our workload, using the remainder of the time as before. We perform experiments with fault-free scenarios and faulty scenarios. In the following we present the results for each scenario. Results show the average of 3 repetitions of each experiment.

Fault-Free Scenario In the Fault-Free scenario, we configure the percentage of queries that are performed to the local partition of the peer. This is expressed through a probability over a random number. Once it is decided if the query is to be performed over the local

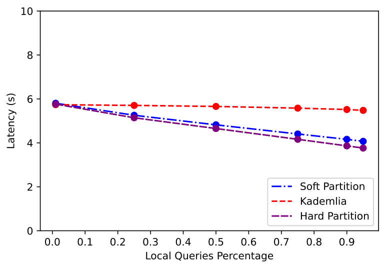
[2]https://www.grid5000.fr/



(a) 3 Partitions.

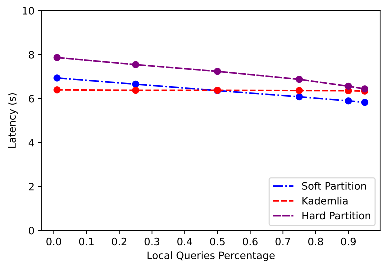


(b) 10 Partitions.

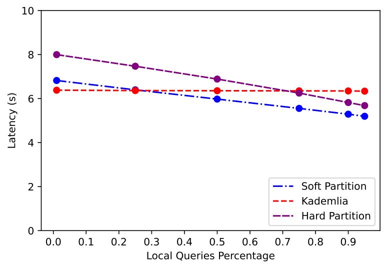


(c) 100 Partitions.

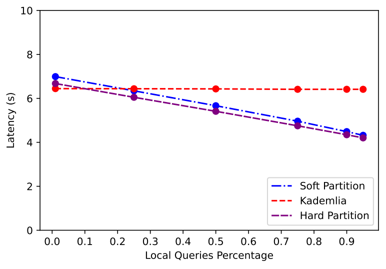
Figure 1: Fault-Free scenarios average FINDNODE latency.



(a) 3 Partitions.



(b) 10 Partitions.



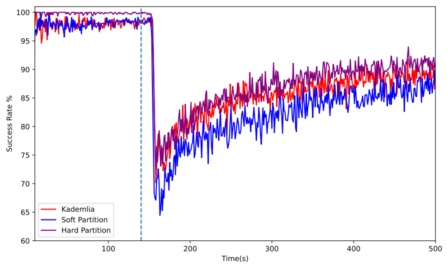
(c) 100 Partitions.

Figure 2: Fault-Free scenarios average FINDVALUE latency.

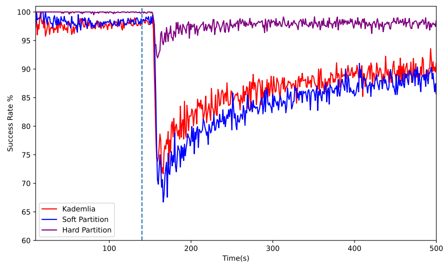
partition or not, a random existing key is chosen that matches the previous decision. Figure 1 shows the results for the FINDNODE operation for networks with 3, 10, and 100 partitions. Figure 2 shows the results for the FINDVALUE operation for networks with 3, 10, and 100 partitions. In our experiments we varied the percentage of locality in queries as follows: 1%, 25%, 50%, 75%, 90%, and 95% (this is represented in the x axis of figures, while the y axis represents the average latency of each operation).

The results presented in Figure 1 show that both solutions improve the average query resolution of Kademlia when the queries present locality accesses. As expected, query patterns with higher locality present higher improvements in our solutions. Furthermore, our solutions also benefit from higher number of partitions. This is due to creating smaller partitions and hence routing in these smaller partitions is faster. Another thing to note, is that the hard partition solution only presents a slight advantage over the soft partition solution with higher numbers of partitions. We believe this to be a consequence of the high latency of contacting the single instance of the indexer service. With more partitions, this effect is attenuated by the fact that partitions are very small (50 peers) and queries are resolved in fewer hops than in the soft partitioning solutions.

The results presented in Figure 2 are similar for all solutions, where the most significant difference is that all solutions experience higher latency. This is because the FINDVALUE operations, operates similarly to the FINDNODE operation, with the addition that peers perform an additional interaction to effectively fetch the stored content. These results show that our solutions has a minimal impact over the DHT operation, other than providing an advantage when queries have locality.



(a) 10 partitions.



(b) 100 partitions.

Figure 3: Faulty scenarios FindNode succaess rate with 30% instant failure.

Faulty Scenarios In these set of experiments, we fix the probability of locality to 50% of our workload and execute the FindNode operation in networks with 10 and 100 partitions. Furthermore, we generate a simultaneous peer fault of 30% of the network after a period of 2 minutes in the experiment (after the initial grace period). Figure 3 shows the results for these experiments. In these experiments we measure the success rate (in the y axis) over time (in the x axis) of queries.

These results show that all solutions, for low numbers of partitions (Fig. 3a) have similar fault tolerance, with the soft partition solution being the most affected by the failure of peers. With higher numbers of partitions (Fig. 3b), the hard partition solution has significantly more fault tolerance than the competing alternatives. This is due to the fact that the failure has less impact on the smaller DHTs as these smaller DHTs are more connected (i.e., peers have more connections among themselves), and that finding remote content (i.e., that is not on the local partition) is done primarily via the indexer service. It is also important to note that the operation of the hard partition is highly dependent on the indexer service, which as a single instance, is a single point of failure. However, the indexer service can be easily materialized by more than one (independent) instances with virtually no coordination among them.

5 Conclusion and Future Work

In this paper we have presented two solutions capable of improving the performance of Kademlia for DHT query patterns that present locality, without incurring in a strong operational overhead. Our soft partitioning solution is capable of influencing the routing table maintained by each peer to encode locality, and hence improve the DHT query routing on queries that exhibit locality. The hard partitioning solution on the other hand, presents interesting properties such as the fact of enabling jumping large portions of the DHT, effectively creating larger shortcuts, and hence further improving the DHT performance when a large number of partitions are present in the network.

The work presented in this paper is still work in-progress, and we aim at following these first findings to further improve these solutions. In particular, we aim at improving the soft partition solution with caching of peer contacts as employed by the hard partition solution, to enable larger shortcuts on the soft partition solution. Regarding the hard partition solution, the indexer service has a large design space worth exploring to remove the single point of failure and improve the latency required to contact the service. Furthermore, we plan to continue the evaluation of these solutions with different setups and by using data extracted from the operation of the IPFS network, to fully understand the impact that our solutions can have in a real system such as the IPFS network.

References

[1] Web 3.0 technology stack. `https://web3.foundation/about/`. Accessed July 2021.

[2] F. Araujo and L. Rodrigues. Geopeer: a location-aware peer-to-peer system. In *Third IEEE International Symposium on Network Computing and Applications, 2004. (NCA 2004). Proceedings.*, pages 39–46, 2004.

[3] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System. Technical Report Draft 3, 2014.

[4] Juan Benet and Nicola Greco. Filecoin: A Decentralized Storage Network. Technical report, 2017.

[5] Bram Cohen. The bittorrent protocol specification.

[6] ConsenSys. Consensys: Ipfs look up measurement. `https://github.com/ConsenSys/ipfs-lookup-measurement/`. Accessed February 2022.

[7] Michael J Freedman and David Mazieres. Sloppy hashing and self-organizing clusters. In *International Workshop on Peer-to-Peer Systems*. Springer, 2003.

[8] Vincent Gramoli, Ymir Vigfusson, Ken Birman, Anne-Marie Kermarrec, and Robbert van Renesse. Slicing distributed systems. *IEEE Transactions on Computers*, 58(11):1444–1455, 2009.

[9] Christian Gross, Dominik Stingl, Björn Richerzhagen, Andreas Hemel, Ralf Steinmetz, and David Hausheer. Geodemlia: A robust peer-to-peer overlay supporting location-based search. In *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*, pages 25–36, 2012.

[10] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A scalable overlay network with practical locality properties. In *4th USENIX Symposium on Internet Technologies and Systems (USITS 03)*, Seattle, WA, March 2003. USENIX Association.

[11] Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap. Decentralized and locality aware replication method for dht-based p2p storage systems. *Future Generation Computer Systems*, 84:32–46, 2018.

[12] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29 Annual ACM*

Symposium on Theory of Computing, STOC '97, page 654–663, New York, NY, USA, 1997. ACM.

[13] A. Kovacevic, N. Liebau, and R. Steinmetz. Globase.kom - a p2p overlay for fully retrievable location-based search. In *2007 7th International Conference on Peer-to-Peer Computing*, pages 87–96, Los Alamitos, CA, USA, sep 2007. IEEE Computer Society.

[14] João Leitão. *Topology Management for Unstructured Overlay Networks.* Phd thesis.

[15] J. Liebeherr, M. Nahas, and Weisheng Si. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, 2002.

[16] F. Maia, M. Matos, R. Vilaca, J. Pereira, R. Oliveira, and E. Riviere. Dataflasks: Epidemic store for massive scale systems. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems (SRDS)*, pages 79–88, Los Alamitos, CA, USA, oct 2014. IEEE Computer Society.

[17] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*. Springer, 2002.

[18] B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[19] J. Paiva, J. Leitão, and L. Rodrigues. Rollerchain: A dht for efficient replication. In *Procceedings of the 12th International Symposium on Network Computing and Applications*, pages 17–24, Aug 2013.

[20] Tiago P. Peixoto. graph-tool: Efficient network analysis. `https://graph-tool.skewed.de`. Accessed February 2022.

[21] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '97, page 311–320, New York, NY, USA, 1997. ACM.

[22] Protocol Labs. libp2p: A modular network stack. `https://libp2p.io`. Accessed February 2022.

[23] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, page 161–172, New York, NY, USA, 2001. ACM.

[24] Sylvia Ratnasamy, Ion Stoica, and Scott Shenker. Routing algorithms for dhts: Some open questions. In *International workshop on peer-to-peer systems*. Springer, 2002.

[25] Saurabh Ratti, Behnoosh Hariri, and Shervin Shirmohammadi. Nl-dht: A non-uniform locality sensitive dht architecture for massively multi-user virtual environment applications. In *2008 14th IEEE International Conference on Parallel and Distributed Systems*, pages 793–798, 2008.

[26] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, pages 329–350, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[27] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4), August 2001.

[28] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. Technical report, 2014.

[29] Weiyu Wu, Yang Chen, Xinyi Zhang, Xiaohui Shi, Lin Cong, Beixing Deng, and Xing Li. Ldht: Locality-aware distributed hash tables. In *2008 International Conference on Information Networking*, pages 1–5, 2008.

[30] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

[31] Shuheng Zhou, Gregory R Ganger, and Peter Alfons Steenkiste. Location-based node ids: Enabling explicit locality in dhts. Technical report, 2003.

[32] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, 2015.