An Empirical Study on the Performance of the Kademlia DHT Under High Node Churn

Yantao Luo
New York University
yl5929@nyu.edu

Yitao Zhou
New York University
yz4171@nyu.edu

***Abstract*—This paper presents an extensive simulation and analysis of Kademlia, a distributed hash table (DHT) protocol that supports decentralized peer-to-peer networks, under conditions of high node churn. Our primary findings suggest that Kademlia maintains stable operation regardless of fluctuations in the number of nodes, average node lifetime, or variations in the distance function. However, we observe an increased retrieval latency as the churn rate increases. The experimental setup is based on a widely-accepted Kademlia implementation in Python and uses a local computer network to mimic the conditions of a Kademlia environment for testing purposes. By changing various network parameters, we measure the impact on network performance and latency, offering a comprehensive understanding of Kademlia's behavior under different churn scenarios. Our analyses shed light on the key factors influencing Kademlia's performance in the face of node churn. Furthermore, we explore potential modifications, such as a hybrid approach that balances uptime and proximity-based node selection, that could potentially enhance Kademlia's robustness and efficiency under high churn rates.**

I. Introduction

Research into peer-to-peer (P2P) systems has became increasingly important due to the popularity of widely-deployed file-sharing services such as BitTorrent and IPFS. A particular area of interest lies in enhancing the algorithms employed in structured P2P overlay networks, also known as distributed hash tables (DHTs). These networks map a large identifier space onto the nodes in the system in a deterministic and distributed way, a process commonly referred to as routing or lookup. Typically, DHTs only require O(*log N*) overlay hops in a network of *N* nodes.

Kademlia, a prominent example of such a system, allows network nodes to locate and retrieve data from others without reliance on a central server. It accomplishes this by organizing nodes within a binary tree-like structure, thereby bounding its time complexity. Kademlia employs a distance metric to determine the proximity between nodes in the network. Nodes closer to a given key in the hash table are preferred for storing and retrieving data relevant to that key. Additionally, Kademlia uses a system of iterative lookups to quickly locate nodes containing data of interest, which reduces the number of messages required to find a specific node compared to similar algorithms like Chord.

In this paper, we explore the performance of Kademlia in dynamic environments characterized by node churn, where nodes continuously enter and exit the network. Our research primarily focuses on the success rate and the retrieval time. The success rate measures the likelihood of the *Get* operation retrieving the same result set by the *Set* operation. On the other hand, the retrieval time measures the average time taken to execute the *Get* function, which represents the average delay of retrieving the value of a corresponding key. These concepts are explained in greater detail in Section V. We investigated the influence of three variables on the success rate and retrieval time: the number of nodes in the network, the average node lifetime, and different distance functions.

Our findings demonstrate that Kademlia can effectively handle high churn rates, maintaining a high success rate. We also identify several factors that impact latency performance under churn, such as:

- Bootstrap nodes exiting the network, leading to lookup misses.
- The number of bootstrap nodes, which influences the number of neighbors to connect with during startup.
- The distance function, which can add lookup hops if not optimized with a complexity of O(*log N*).

Moreover, a significant part of our exploration is dedicated to investigating the impact of different distance functions on the performance of Kademlia in high churn rate scenarios. In traditional Kademlia networks, a simple XOR metric is used as the distance function, which is used for determining the proximity between nodes. However, in our research, we experimented with a hybrid distance function that combines the uptime of the nodes and the XOR distance. This approach is motivated by the hypothesis that prioritizing nodes based on their uptime could improve retrieval success rates, as older nodes are typically more stable and less likely to leave the network. The in-depth results and analysis of these experiments are presented later in this paper.

In Section II, we review the existing literature on DHT performance under high churn rates. In Section III, we introduce the main ideas behind our project. In Section IV, we outline the Kademlia network protocol upon which our network implementation is based. Section V details our network design for testing purposes and our experimental design. Finally, in Section VIII, we propose potential solutions that could effectively reduce latency while maintaining a high success rate. We also discuss the limitations and challenges encountered in our work.

II. Related works

Several studies have been conducted in the field of node churn, with an emphasis on proposing effective solutions to manage the challenges posed by this issue. Liben-Nowell et al. [1] presented a theoretical analysis of P2P networks, particularly focusing on the Chord system. They proposed a shift in performance metrics, arguing that the rate at which nodes in the network participate to maintain the system state is more informative compared to traditional run-time measures. Their work provides a general lower bound on this participation rate required for a network to remain connected and demonstrates that a modified version of Chord's maintenance rate is within a logarithmic factor of this optimum rate.

Rhea et al. [2], on the other hand, delved into the issue of high churn rates in distributed hash tables (DHTs). They contended that DHTs should ideally perform lookups quickly and consistently under churn rates comparable to those observed in deployed P2P systems such as Kazaa. However, their experiments on an emulated network revealed that current DHT implementations fall short in handling such high churn rates. In response, they explored three crucial factors impacting DHT performance under churn: reactive versus periodic failure recovery, message timeout calculation, and proximity neighbor selection. Through their analysis within the context of Bamboo, a mature DHT implementation, they demonstrated that careful attention to these factors enables Bamboo to operate effectively under high churn rates while utilizing less maintenance bandwidth than other DHT implementations.

Finally, the work of Castro et al. [3] detailed the optimization of an implementation of the Pastry protocol, MSPastry, to ensure consistent routing under churn with minimal overhead. Their contributions highlight the potential of structured P2P overlays in creating highly dependable and efficient distributed systems, even in environments characterized by high churn rates. Their techniques, evaluated through large-scale network simulation experiments, effectively dispel previous concerns regarding the performance and dependability of such overlays in realistic environments.

In summary, these studies underline the significance of understanding and managing node churn in P2P networks and DHTs. Their insights inform the direction of our research and provide a foundation for our proposed solutions.

III. Proposed Idea

In this study, we propose a multifaceted exploration of the performance of the Kademlia distributed hash table (DHT) protocol under varying network conditions. We focus on three key variables: network size, node lifetime, and the prioritization of uptime over distance in Kademlia's neighbor-finding algorithm.

Firstly, we investigate the impact of network size on Kademlia's performance. We aim to understand how increasing the number of nodes in the network affects the retrieval success rate and retrieval time, with node lifetime held constant.

Secondly, we turn our attention to node lifetime. By varying the lifetime of nodes in the network and keeping the number of nodes constant, we aim to examine the effects of churn (i.e., nodes joining and leaving the network) on Kademlia's performance. We hypothesize that a higher churn rate, implying shorter node lifetimes, might lead to increased retrieval times.

Lastly, we propose a modification to the Kademlia neighbor-finding algorithm. The original Kademlia design prioritizes nodes with the lowest XOR distance. However, we explore an alternative approach that prioritizes nodes based on their uptime, with the hypothesis that older nodes may be more reliable.

By conducting these investigations, we aim to gain a deeper understanding of the Kademlia protocol's performance and resilience under different network conditions. Furthermore, our findings could potentially guide the design of more robust and efficient distributed hash table systems.

IV. Kademlia implementation

We used an asynchronous Python implementation of the Kademlia distributed hash table on GitHub [4] as the underlying protocol for our experiments. It uses the asyncio library in Python 3 to provide asynchronous communication. The nodes within this setup communicate using Remote Procedure Call (RPC) over User Datagram Protocol (UDP).

The network in this implementation have *k*, the size of bucket, and the number of nodes looked up during node lookup set to 20. It also set α, the number of closest nodes that will be called during FIND_NODE RPC to 3, meaning that three closest nodes will return their *k* closest neighbors of the target.

This implementation provides *Get* and *Set* interfaces for key and value to spread in the network, which run underlying $FIND\_NODE$, $FIND\_VALUE$ and $STORE$ RPC as defined in the Kademlia paper [5]. The *Set* procedure initiates with a value type check, followed by the encoding of the key using the SHA1 digest. The system then identifies the *k* closest peers within the local routing table. Upon identifying these peers, the system requests their nearest node lists. Any node failing to respond is promptly removed from the local routing table. This results in a curated list of nearest nodes. The key-value pair is then stored in the nodes within a predefined distance from the initiating node.

The *Get* procedure mirrors the *Set* operation in terms of identifying peers within the local routing table. The network then attempts to locate the node containing the requisite key-value pair by querying all available nodes in the nearest list. In case the returned values from all nodes vary, a find common function is initiated to return the most common result. This robust design ensures accuracy and reliability in the retrieval of key-value pairs across the network.

V. Experimental Setup

Our experiments aim to evaluate the performance of the Kademlia protocol under various churn rates using a local computer network. We employ two computers for these experiments: a Macbook equipped with an Intel Core i5-8250U CPU and 8GB memory, running on macOS 12.6, is used for

the first two sets of experiments, namely for network size and node lifetime experiments. The third set of experiment in prioritizing node uptime vs distance is conducted on a MacBook with an Intel Core i7-5650U CPU and 8GB memory, also running on macOS 12.6.

Nodes in the network are associated with a specific port within an unused port range on our local IP. Each node communicates via RPC. When a new node connects to the network, we assign it a port and randomly select an existing node in the network as its bootstrap node. The only exception to this is the bootstrapping of the initial node, which has no neighbors to connect with during the setup phase.

Once a node has been bootstrapped, it is assigned a lifetime, following a normal distribution with a preset mean, μ. The standard deviation is set to be 10% of μ, introducing variability in node lifetimes. Upon reaching the end of its lifetime, the node will disconnect from the network. To maintain a constant number of nodes $NODE\_NUM$, a new node is introduced each time an existing node leaves.

Once the network is operational, a pair of testing nodes from outside the network join and assess the network's stability and performance under varying churn rates. Each testing node is assigned a designated port and bootstraps with a randomly selected existing node in the network. The number of nodes that the testing node will bootstrap with is varied, and this number is described as $bootstrap\_node$ in our demonstration. Testing involves conducting a *Set* operation, waiting for an interval, and then performing a *Get* operation (the specifics of these operations are described in Section IV). This sequence is repeated over several rounds, with a one-second interval between *Set* and *Get* operations to prevent immediate retrieval and to increase the chance of network churn impacting the operations.

We measure two primary metrics during testing: success rate and retrieval time (*get_time*). Success rate is measuring whether in the *Get* procedure will retrieve the same result as what has been set in the *Set* procedure. If the result from *Get* is the same as what was *Set* in the network, then we mark it as one success round. For get time, as the system runs on local network, the latency is minimal. Thus we defined the retrieval time as the time from the beginning of the *Get* procedure till the end of *Get* procedure. The reason of observing the long get time is from the delay in the RPC of UDP. If a node sends a RPC to another node that has already left the network, it must eventually timeout and attempt to connect with another neighbor. The default UDP response wait time in the Kademlia network is 5 seconds. These waiting periods accumulate and contribute to the average *get_time* for each round.

A. Network Size

In the first set of experiments, we aim to understand the impact of network size on Kademlia's performance. Here, we fix node lifetime to a normal distribution of 180 seconds and vary the number of nodes in the network from 10 to 500, running each test for 100 rounds. The number of nodes that a testing node bootstraps with is also varied to evaluate its effect on network performance.

B. Node Lifetime

In our second set of experiments, we study the effects of variable node lifetimes on Kademlia's performance. Here, we fix the number of nodes in the network to 50 and vary the lifetime of network nodes as a normal distribution with mean time from 60 to 1200 seconds. We also prepare an experiment without node churn to serve as the baseline, which returns full success rate and negligible get time. Two nodes are used for bootstrapping the testing nodes, as suggested by the first network size experiment. The results are obtained after 100 test rounds.

C. Prioritizing Uptime vs Distance

Our third experiment explores an alternative neighbor-finding algorithm for Kademlia. Instead of prioritizing nodes with the lowest XOR distance, as in the original Kademlia $find\_neighbors$ function, we propose an alternative approach that prioritizes nodes based on their uptime, with the hypothesis that older nodes may be more reliable. We wanted to investigate how this modified algorithm will impact retrieval success rate and retrieval time.

To test this hypothesis, we perform experiments in a network of 30 nodes where the node lifetime is evenly distributed across a wide range, simulating a network environment with a mix of nodes with short and long uptimes. There is a one-second delay between each set and get function calls. Each experiment has 100 set and get operations and is repeated three times to ensure consistent and representative results. We then measure the average retrieval success rate and average retrieval time under this modified algorithm and compare it with the original Kademlia design.

VI. Results and Analysis

A. Network Size

In our initial set of experiments, each testing node was assigned a single bootstrap node. Our findings revealed a considerable decline in Kademlia's performance as the number of nodes in the network increased, as illustrated by the blue line in Fig. 1. We attribute this behavior to two primary factors.

First, as we maintained a constant node lifetime throughout the network, an increase in the number of nodes directly translates to a higher churn rate, which is measured by $\frac{number\_of\_nodes}{lifetime}$. This increased churn rate destabilizes the network and causes content loss. Second, bootstrapping with a single node proves insufficient. Since we do not enforce a lock to keep the bootstrap node within the network, the bootstrap node may leave the network during the process. As the retrieval time increases due to extended response times caused by lookup misses, the likelihood of the bootstrap node leaving the network increases, resulting in a lower success rate. This scenario effectively mirrors real-world network behavior, wherein nodes freely join and leave the peer-to-peer network.

To address these shortcomings, we modified our experimental setup to include two bootstrap nodes for each testing node. We observed a substantial improvement in the success rate, as demonstrated by the red line in Fig. 1. The results indicate that having two bootstrap nodes sufficiently maintains high performance, regardless of the number of nodes in the network.

However, retrieval time increases as the number of nodes increases. This outcome is expected as a larger binary tree necessitates more hops for lookups, thereby increasing the probability of misses. As Fig. 2 illustrates, the average retrieval time rises with an increasing number of bootstrap nodes. This increase is due to a higher *Get* miss rate when involving more nodes in the *Get* process.

To interpret the peak in retrieval time when the number of nodes equals 250 and the subsequent decline as the node count grows, we theorize that a significantly larger network size introduces greater availability, reducing miss chances. Consequently, there should exist an optimal number of nodes that would yield the highest average retrieval time.

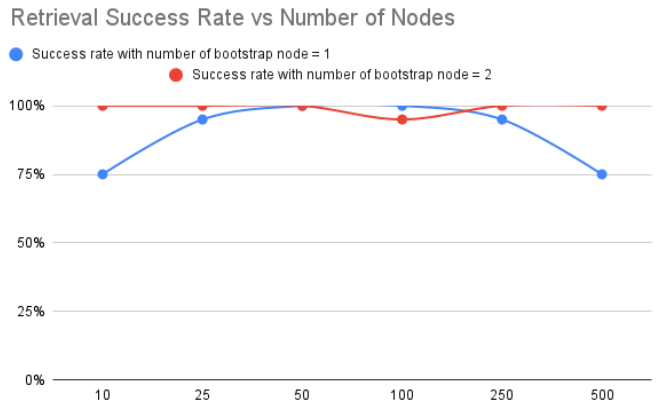

Fig. 1. Retrieval Success Rate vs Number of Nodes

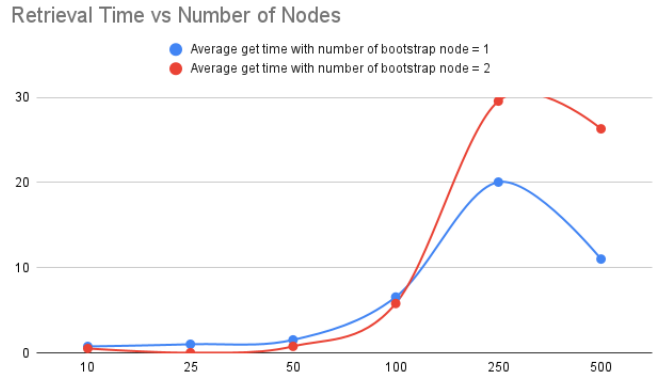

Fig. 2. Retrieval Time vs Number of Nodes

B. Node Lifetime

As shown in Fig. 3, the success rate appears to be negligibly affected by the average node lifetime. Minor fluctuations in the success rate are only evident when the node lifetime is relatively short. Combined with the results from the previous experiment, it can be concluded that the churn rate does not significantly impact the retrieval success rate, provided not all content copies are removed due to node departures between *Set* and *Get* operations.

Nevertheless, we noted a substantial increase in retrieval time when node lifetimes were shorter, as illustrated in Fig. 4. This finding can be attributed to an extremely high churn rate, where nodes continuously join and leave the network, thereby increasing the likelihood of lookup misses.

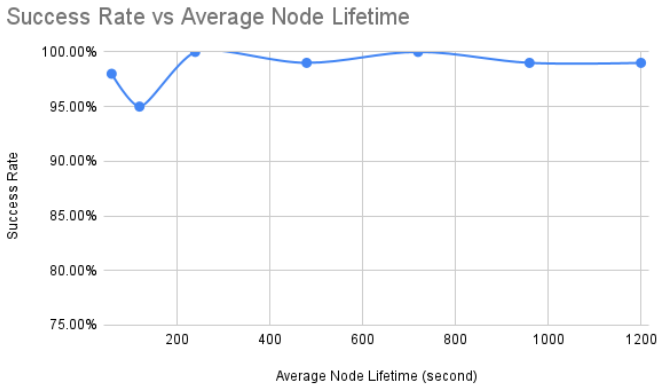

Fig. 3. Retrieval Success Rate vs Number of Nodes

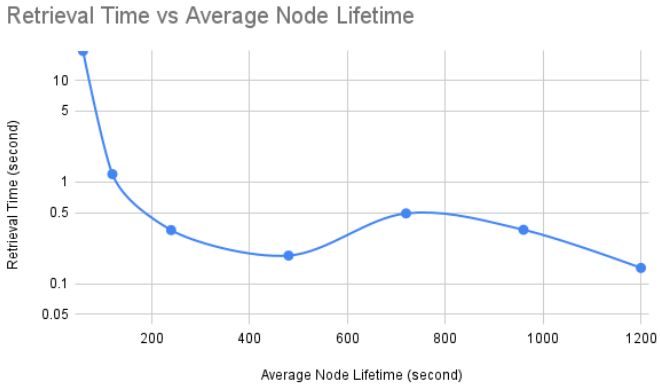

Fig. 4. Retrieval Time vs Average Node Lifetime

C. Prioritizing Uptime vs Distance

The results show that prioritizing nodes based on their uptime, rather than their distance in the key space, has an overall positive impact on retrieval success rates. This is likely due to the fact that older nodes with longer uptime are generally more available than newer nodes and less likely to leave the network. However, prioritizing uptime significantly increases the retrieval time (latency). In the traditional Kademlia protocol, nodes are selected based on their proximity in the key space, which typically leads to fewer hops and thus faster

retrieval times. By prioritizing older nodes, more hops might be added to each lookup operation, as these nodes could be distributed more widely in the key space.

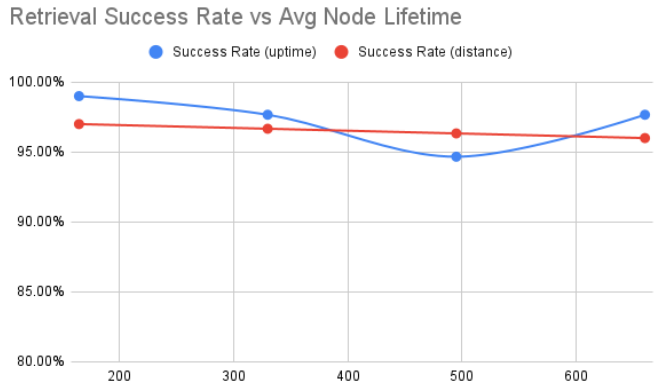

Fig. 5. Retrieval Success Rate vs Average Node Lifetime

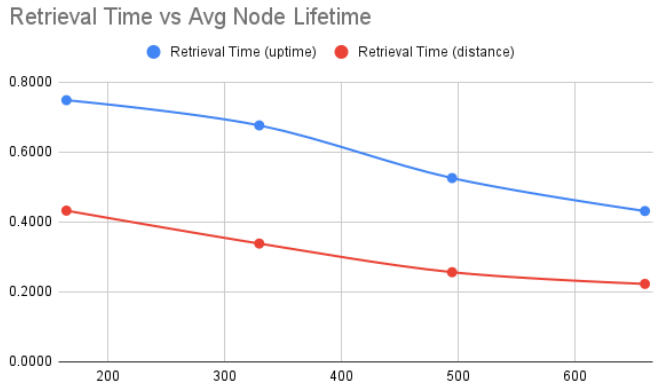

Fig. 6. Retrieval Time vs Average Node Lifetime

VII. A Hybrid Neighbor-finding Algorithm

Inspired by the experimental results discussed previously, we proposed and investigated a hybrid neighbor-finding algorithm for Kademlia. This algorithm aims to balance the benefits of both proximity-based and uptime-based node selection.

A. Implementation

Our hybrid approach involves calculating a distance threshold, which is set at half of the maximum key space. Nodes within this distance threshold are then sorted based on their uptime. This allows us to prioritize older nodes but only if they are within a certain distance of the key.

The updated $find\_neighbors$ function first collects a list of neighbors within the distance threshold, and then sorts them according to their uptime. If the number of these nodes is less than k, the function calls $find\_nearest\_neighbors$ to fill the remaining spots, ensuring the total number of neighbors returned is k.

B. Experiment

In addition to evaluating our hybrid approach using the previous experiment configurations, we also crafted a new experiment specifically designed to illuminate the distribution and storage of key-value pairs across the network. In this experiment, we constructed a network comprising 30 nodes, with lifetimes uniformly spanning a range from 120 to 1200 seconds. This setup serves to simulate a network environment that combines nodes with short and extended uptimes.

Within this network, we executed a sequence of 100 set and get operations, introducing a one-second delay between each operation to more accurately reflect real-world usage. To record relevant data, we implemented a monitoring program that logs the uptime and the number of key-value pairs each node holds every second.

Upon completion of all set and get operations, we conducted an analysis of the number of key-value pairs stored in each node for all three algorithms. The resulting data were then compiled and visualized in a histogram format, offering a clear and comprehensive view of the distribution and storage of key-value pairs within the network.

C. Evaluation

As anticipated, our hybrid approach strikes a balance between uptime-based and proximity-based node selection, displaying a modestly enhanced retrieval success rate compared to the original neighbor-finding algorithm. At the same time, it maintains a relatively low retrieval time, thereby affirming the efficiency of this approach.

Regarding the distribution and storage of key-value pairs, the results showcase varying patterns. The distance-based selection algorithm exhibits the most uniform distribution, with the majority of nodes storing approximately 50 to 65 key-value pairs. This uniformity suggests that the load is more evenly distributed across the network, which could potentially increase the resilience of the system to node churns.

The uptime-based selection algorithm, on the other hand, shows a significant concentration of data on nodes with longer uptimes. Most nodes in this setup store between 80 to 100 key-value pairs, representing a substantial portion of the data. Only a minority of nodes store less than 75 key-value pairs. While this distribution pattern effectively leverages the longevity of nodes, prioritizing data storage on nodes with longer uptimes, it somewhat defeats the purpose of a DHT, given that most nodes store the vast majority of the data.

Our hybrid algorithm demonstrates a more balanced distribution. About half of the nodes store between 85 to 95 key-value pairs, while the other half store between 60 to 70 key-value pairs. This distribution suggests that the hybrid approach can maintain a fair load distribution while prioritizing nodes with longer uptimes for key-value pair storage. It combines the advantages of both the uptime-based and distance-based node selection, potentially enhancing the overall robustness and efficiency of the Kademlia network.

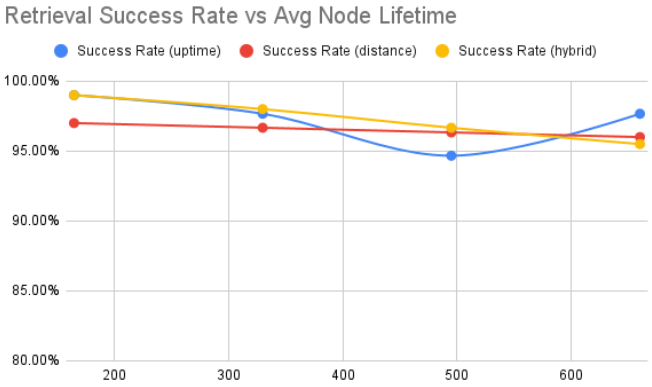

Fig. 7. Retrieval Success Rate vs Average Node Lifetime (Hybrid)

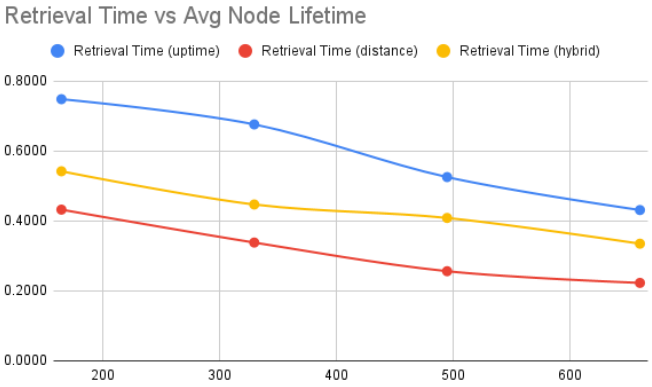

Fig. 8. Retrieval Time vs Average Node Lifetime (Hybrid)

VIII. Limitations and Future Work

While our results demonstrate robust success rates under high churn conditions, certain limitations and potential enhancements in our approach warrant discussion to further stabilize the network and minimize retrieval time.

In our current network design, a node departs the network after the expiration of its lifetime, initiating the creation of a new node. However, new node creation may be time-consuming due to potential lookup misses during the bootstrap process. Consequently, there may be instances where the total number of nodes does not equate to $NODE\_NUM$, as some other nodes may depart before the completion of the node creation process. One potential solution could be implementing a locking mechanism, which would require departing nodes to wait until the new node creation process is completed, thereby maintaining a stable network size. This approach, however, may inadvertently extend the lifetime of nodes beyond the preset value.

Moreover, as per the Kademlia paper [5] section 2.3, the original publisher of a key-value pair must republish it every 24 hours. As churn rate increases, the republishing interval could be dynamically adjusted to prevent loss of content

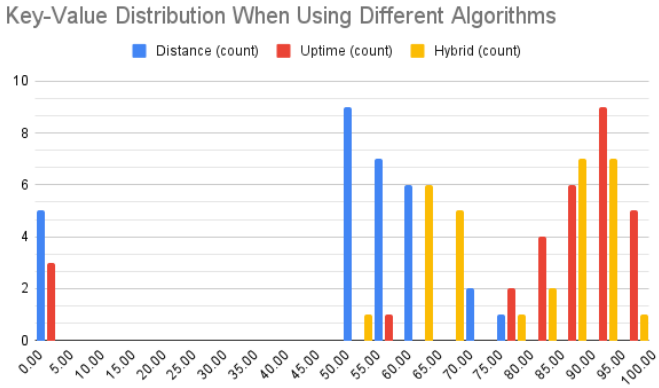

Fig. 9. Key-Value Pair Distribution Comparison Between Different Neighbor-finding Algorithms: Number of Nodes vs Number of Key-Value Pairs Stored.

within the network. Nodes could monitor the network churn rate by recording the lookup miss rate, thereby enabling the republishing interval to be proportionally adjusted to the churn rate. This modification could potentially extend the content's lifespan within the system.

Additionally, our preliminary results indicate that the hybrid approach potentially improves retrieval success rate by leveraging the stability of older nodes, while also retaining the benefits of proximity-based node selection. However, in our experiments, we only used a fixed distance threshold value, set at half of the maximum key space. This leaves open the question of how different distance thresholds might impact the performance of our hybrid approach. In future work, we aim to conduct a comprehensive study of varying distance thresholds. By assessing the performance of the Kademlia network under different distance thresholds, we expect to uncover more about the relationship between node proximity and network performance. The goal would be to identify an optimal distance threshold value or range, which could be dynamically adjusted according to specific network conditions, such as node density or churn rate.

IX. Conclusions

In this project, we conducted an in-depth exploration of the effects of churn rate on the performance of Kademlia peer-to-peer systems. We discovered that while Kademlia was able to provide a high success rate even when churn rate is high, this performance comes at the cost of extended retrieval times.

During our experimentation, we discovered that when the number of bootstrap nodes was limited to one, the success rate left much to be desired. Upon increasing the number of bootstrap nodes, however, we saw a significant improvement in the success rate. It is important to note, though, that as the number of nodes within the network increased, the retrieval time also saw an increase. This is likely due to the higher number of hops required for a lookup, which in turn increases the likelihood of lookup misses, causing longer retrieval times.

Interestingly, we found that varying node lifetimes had a minimal impact on the success rate. However, shorter node

lifetimes, which can be equated to high churn rates, resulted in extended retrieval times. This observation mirrors the trend we noted in our previous set of experiments, where an increase in churn rate was paralleled by an increase in the likelihood of lookup misses.

We additionally investigated the impact of prioritizing nodes based on uptime over distance within the key space. While this approach yielded a higher retrieval success rate, it also significantly increased retrieval times. Inspired by these results, we proposed and tested a novel hybrid approach, offering a middle ground between uptime-based and proximity-based node selection. This approach showcased a slight improvement in retrieval success rate compared to the original algorithm while also keeping retrieval times relatively low. This suggests that a balance between uptime and proximity-based approaches could yield better performance under certain conditions.

In conclusion, while Kademlia demonstrates resilience in the face of high churn rates, there are potential modifications, such as the hybrid approach, that could optimize its performance further. Future work in this area could lead to more robust and efficient peer-to-peer systems.

REFERENCES

[1] D. Liben-Nowell, H. Balakrishnan, and D. Karger. "Analysis of the evolution of peer-to-peer systems." In Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC '02). Association for Computing Machinery, 2002, pp. 233–242. https://doi.org/10.1145/571825.571863

[2] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT" in Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC '04), USA, 2004, pp. 10. doi: 10.1145/1015467.1015497.

[3] M. Castro, M. Costa and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays," International Conference on Dependable Systems and Networks, 2004, Florence, Italy, 2004, pp. 9-18, doi: 10.1109/DSN.2004.1311872.

[4] B. Muller, "Kademlia." GitHub, 2021. https://github.com/bmuller/kademlia (commit cf55e65)

[5] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01), Springer-Verlag, Berlin, Heidelberg, 2002, pp. 53-65.