IIUC Studies, Vol.-20, Issue-1, June 2023
DOI: https://doi.org/10.3329/iiucs.v20i1.69054

An effective and efficient protocol for propagating large files in Peer-to-Peer systems

MD Jiabul Hoque
Department of Computer and Communication Engineering
International Islamic University Chittagong (IIUC), Bangladesh

Abstract

Distributed Hash Tables (DHTs) are integral components in peer-to-peer systems, providing storage and lookup services for key-value pairs. While they have proven highly successful in managing and locating replicas of small files (typically under 1 MB), their efficiency in handling larger files diminishes. Factors like sluggish data senders and slow receivers further exacerbate the inefficiencies in peer-to-peer systems, causing delays in the file propagation process. To address these challenges, peer-to-peer systems require more efficient protocols and resources to expedite the handling of relatively large data files. This study introduces a novel, efficient, and effective mechanism to propagate substantial files within DHTs while balancing load and minimizing system overhead. To assess the proposed protocol's effectiveness, we evaluated using the PeerSim simulator. We analyzed two crucial metrics linked to our proposed system: overhead and propagation time. The outcomes of this research demonstrate a significant reduction in propagation time while system overhead remains minimal. Consequently, the proposed protocol can ensure seamless operation in real-world streaming applications.

Keywords DHTs, Overhead, Peer-to-peer systems, Propagation, Protocol, Replication

Paper type Research paper

1. Introduction

The advent of digital communication has witnessed significant growth in the development of peer-to-peer systems. Many applications, including Skype, WhatsApp, Spotify, Facebook Messenger, and many others, have emerged as exemplars of the capabilities of peer-to-peer systems (Tran, Nguyen, & Ha, 2016). These systems typically fall into two categories: unstructured and structured. Unstructured systems, while versatile, often compromise performance, as they do not employ pre-arranged peer organization for communication (Rodrigues & Druschel, 2010). Conversely, structured peer-to-peer systems utilize overlays to consolidate their peers, prioritizing performance enhancements. These purpose-built overlays expedite peer lookup and facilitate efficient data retrieval (Nasir, Muhammad, Bellavista, Lee, & Sajjad, 2020).

IIUC Studies
Vol.-20, Issue-1, June 2023
pp. 141-166

ISSN 2408-8544

Distributed Hash Tables (DHTs), serving a similar purpose as overlays, employ various topologies to organize their peers (Galuba & Girdzijauskas, 2009). For example, Kademlia employs a binary tree structure, while Chord arranges peers in a ring formation. Generally, DHTs excel in storing and propagating relatively small datasets, typically under 1 MB, operating through interfaces such as put, get, and remove (Hassanzadeh-Nazarabadi, Kupcu, & Ozkasap, 2021). DHTs initiate their operations by dispatching requests (e.g., put(value)) into the DHT ring. These requests undergo hashing and are directed toward a designated peer, the coordinator, whose identifier closely matches or equals the hashed value. The coordinator replicates the value among the DHTs (Franchi & Poggi, 2019). In scenarios where the size and number of replicas are substantial, and the peer bandwidth is limited or slow, the coordinator's workload becomes overwhelming, significantly impacting the overall system's performance. Consequently, DHTs are more suited for storing relatively small data files (Kaur, Gabrijelcic, & Klobucar, 2022).

The central objective of this thesis is to empower DHTs to efficiently store and utilize peer bandwidth for relatively large data files. While research has explored mutable DHTs and those with transactional consistency, more attention should be given to developing suitable protocols or solutions for propagating large data files within DHTs. The load-balancing capabilities of DHTs experience a sharp decline when handling a substantial increase in the size and number of stored files. Similarly, file propagation within DHT becomes time-consuming under low-bandwidth connections with the coordinator (Nakayama & Asaka, 2017).

This paper's motivation lies in designing a protocol that enables DHTs to swiftly store and propagate relatively large files while preserving load balancing and minimizing system overhead. Our proposed approach involves breaking large files into smaller pieces and facilitating collaborative exchange among interested peers. To evaluate the effectiveness of our new propagation protocol, we have implemented a simulation, emulating real-world events using a partial dataset from Wikipedia. We focus on two critical metrics during the evaluation: overhead and propagation time.

The remainder of this article is structured as follows: Section 2 provides an in-depth examination of peer-to-peer systems, Distributed Hash Tables (DHTs), and SpiderCast within the context of background analysis. Section 3 delineates the problems, emphasizing the formidable challenge of efficiently propagating large files within DHTs. Section 4 offers a distinctive resolution by introducing an innovative propagation protocol. Section 5 elucidates the experimental methodology, detailing the utilization of the PeerSim simulator alongside a Wikipedia dataset. Section 6 unveils the

outcomes of our experiments, incorporating lucid figures and comprehensive analyses. Finally, in Section 7, we conclude our study, summarizing its primary findings, addressing the challenges and potential remedies for propagation time and overhead management, and outlining future research directions and the broader implications of the proposed protocol.

2. Background

2.1 Peer-to-peer (P2P) systems

Peer-to-peer (P2P) systems have emerged as significant contributors to digital communication. These systems exhibit uniform functionality across all participating peers, encompassing routing, storage, processing, and more (Lamport, 2019). However, the relative architectural simplicity of P2P systems can pose a challenge for application developers seeking to construct complex applications, such as online video games, multimedia streaming services, and robust storage solutions (Stoica, Morris, Liben-Nowell, Karger, & Balakrishnan, 2003).

P2P systems offer several advantages, as outlined below (Hentschel, Hassanzadeh-Nazarabadi, Seraj, Shirley, & Lafrance, 2020):

- Minimal infrastructure requirements for operation.
- Independence from central servers, ensuring robustness.
- High scalability due to an equitable distribution of workloads among peers.
- Exceptional robustness, capable of handling churn and failures effectively.
- Accurate load balancing across all peers.
- Efficient mechanisms for handling failures, preventing disruptions in the event of Byzantine or computer crashes.

2.1.1 Types of P2P systems

P2P mechanisms encompass two primary variants: structured and unstructured peer-to-peer systems.

Structured P2P Systems: Within structured P2P systems, peers are organized systematically, avoiding the random allocation of values to peers. Such structured systems effectively utilize peer bandwidth, expedite data retrieval from peers, and eliminate unnecessary broadcast requests. Prominent overlays, such as Chord, Pastry, and Kademlia, arrange peers in structured P2P systems (Kwon & Ryu, 2004).

Unstructured P2P Systems: Conversely, unstructured P2P systems involve peers needing knowledge of overlays before or during network entry. This

paper does not delve into unstructured P2P systems (Kwon & Ryu, 2004).

2.1.2 DHTs

Distributed Hash Tables (DHTs) offer storage and lookup services for values and their associated keys. DHTs are widely adopted in distributed systems due to their demonstrated efficiency in load balancing, availability, low overhead, scalability, and user-friendly interfaces (Hassanzadeh-Nazarabadi, Kupcu, & Ozkasap, 2021). The proposed protocol for data propagation can be applied to both mutable and immutable DHTs, enabling the propagation of large data files while reducing latency and maintaining load balancing within the DHT.

2.2 SpiderCast

SpiderCast is a decentralized overlay that operates without reliance on a central server. This characteristic equips it with a robust failure-handling mechanism during distributed processing (Zhao, Huang, Stribling, Rhea, Joseph, & Kubiatowicz, 2004). This section provides insight into how SpiderCast maintains connected replicas of the same file within the network.

2.2.1 Modification of SpiderCast

Our technique for defining coverage differs from SpiderCast. In SpiderCast, the highest number of peers was k*h, where k was the required coverage, and h was the total quantity of entries (Raj & Rajesh, 2016). This method is impractical for DHTs with several values. Our method programs the maximum number of peers based on simulator characteristics. SpiderCast's availability is peers hosting an identical file 100% or partially. Another difference is when the overlay stops adding links. After covering K subjects, SpiderCast disconnects superfluous links. More neighbors than authorized start the disconnection process (Zahid, Abid, Shah, Naqvi, & Mehmood, 2018).

2.3 Replication

Peer-to-peer systems replicate files across several peers. This technique improves performance and availability, especially in dynamic contexts with rapid peer additions and removals. Replication improves system performance by reducing response times, improving load balancing, and scaling (Maymounkov & Mazières, 2002). However, communicating with replicas hosting the object takes much work. Replication approaches dominate this article. Web servers can route traffic to other copies to assist load balancing and system scaling as additional users join. However, fixing file copy maintenance is necessary for data replication (Tahir, Abid, & Shah, 2017).

2.3.1 Coordinators

Within a Chord ring, as illustrated in Figure 1, the coordinator plays a crucial role in generating new replicas by distributing documents among peers, aligning with the system's replication policy. This replication is necessary to address churn and preserve data availability, preventing data loss (Stein, Tucker, & Seltzer, 2002). Our solution benefits coordinators by reducing propagation load and latency.

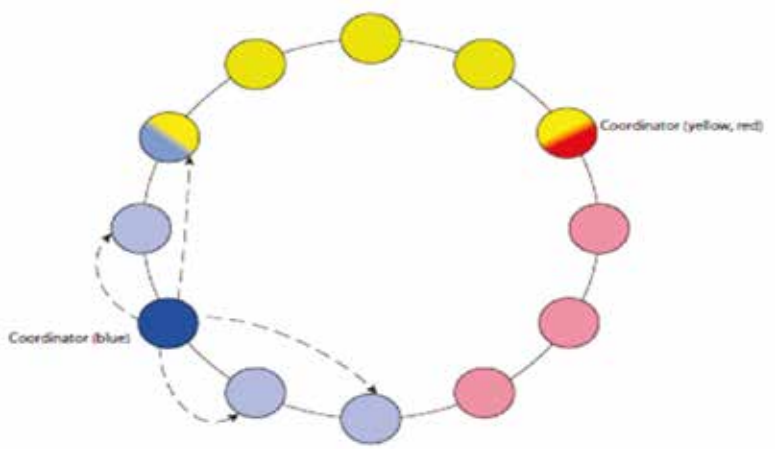

Figure 1
Coordinators are propagating files to peers in DHTs

2.3.2 Consistency

In the pursuit of achieving consistency, various information propagation strategies have been proposed and evolved. This section provides a concise overview of different concepts related to information propagation to replicas. It is important to note that the choice of strategy can vary depending on the specific system being employed, and these strategies can be adapted to ensure the desired level of consistency (Nguyen, Hoang, Hluchy, Vu, & Le, 2017).

Push Consistency

Push-based consistency techniques are server-based protocols. These protocols allow servers to update replicas. This method carefully orders updates received from a server, and pushes them to replicas. Push-based protocols ensure timely and consistent update distribution in situations that require high consistency (Sonbol, Ozkasap, Al-Oqily, & Aloqaily, 2020).

Pull Consistency

Client-based protocols, or pull-based procedures, entail replicas frequently

obtaining updates from a server to check changes. Unlike push-based protocols, replicas may easily retrieve updates, which might add overhead. As shown in Figure 2, replicas frequently seek data from primary replicas to obtain updates when most appropriate for their operating environment, even if it requires additional resource use (Sonbol, Ozkasap, Al-Oqily, & Aloqaily, 2020).

One notable drawback of these techniques is that they exhibit less uniformity than pushing mechanisms. This inherent characteristic implies a substantial delay between the initiation of an update and the moment when the updated material becomes accessible on replicas (Xie, Wang, & Wang, 2017).

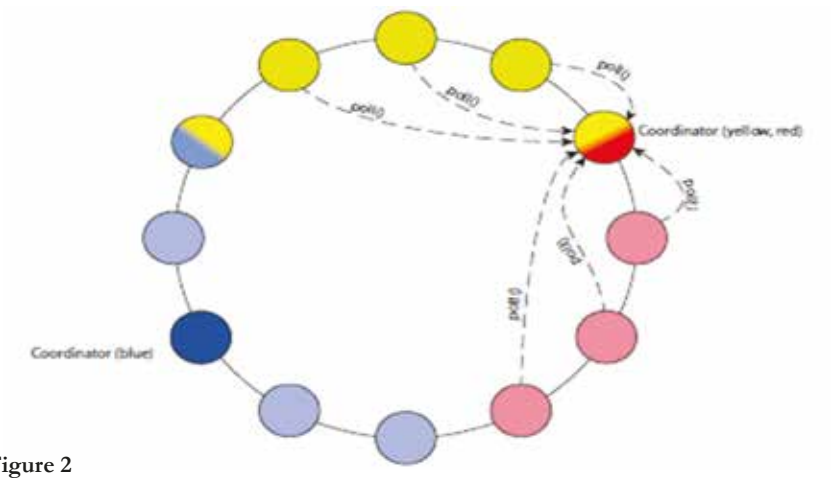

Figure 2
Updating process through pulling

3. Problem definition

In this section, we clearly define the problem at hand and outline the requirements that must be met to address this problem effectively.

3.1 Problem

Replication plays a vital role in enhancing data availability within peer-to-peer systems. However, replicating files becomes challenging when peers leave the system, potentially eliminating file replicas. Distributed Hash Tables (DHTs) offer mechanisms to create and distribute new replicas of relatively small files. However, there is a critical gap in propagating replicas of larger files (exceeding 1MB in size). Coordinators disseminating these more giant file replicas to interested peers encounter significant bandwidth constraints, resulting in slowed propagation of large files. Moreover, peers with low-bandwidth connections can further impede the coordinator, leading

to potential performance degradation across the entire DHT system (Gupta, Hada, & Sudhir, 2017).

The primary objective of this thesis is to develop a propagation scheme capable of accelerating the distribution of substantial files within DHTs. The implementation of our proposed propagation protocol offers several advantages:

1. It enhanced system performance through the efficient utilization of low bandwidth.
2. It efficiently replicates and rapidly propagates large files, particularly when videos or audio become viral in peer-to-peer systems.
3. It is compatible with mutable and immutable data stored within DHTs, enabling the swift propagation of updated replicas.

3.2 Requirements

The efficient and effective propagation of files within DHTs necessitates the fulfilment of both functional and non-functional requirements (Zhou, Dai, & Li, 2006):

Functional Requirements:

1. Ensure that each peer remaining within the system receives a file replica.
2. Our proposed protocol should exclusively involve peers interested in a particular file while excluding unrelated peers.

Non-Functional Requirements:

1. Preserve all essential properties offered by a typical DHT.
2. Maintain scalability, even as the number of files within the DHT increases, by enhancing load balancing through modified SpiderCast overlays and the propagation of partial replicas to establish connections between peers.
3. Uphold churn resilience without modifying the recovery mechanisms or failure detection processes of standard DHTs.
4. Reduce update and replica generation time to increase data availability.
5. Improve bandwidth utilization by deploying a modified push-pull protocol that allows peers to communicate efficiently, considering their available bandwidth.
6. Keep overhead to a minimum by monitoring control messages effectively.

4. Proposed solution

We previously discussed a conventional Master-Slave (MS) distribution strategy, where a coordinator was responsible for individually transferring file content to each peer. However, this approach needed to be revised, particularly in common peer-to-peer system scenarios involving low bandwidth connections. To overcome these limitations, we have introduce a novel protocol, which we will explain comprehensively along with its pseudo-code.

4.1 Proposed propagation protocol

The proposed protocol represents a modified push-pull protocol designed for rapidly propagating large files.

4.1.1 Algorithm for push

In this part, we consider that peer *p* has received the logs from node n, which has begun an iterative *push*. Periodically, push processes are called. The *push* iteration is started by the subsequent function. The state of peer *n* is updated in the *log* entries before pushing is initiated. Then local *logs* are added, and *neighbours* share them. The following function analyses nearby *log* files and distributes *logs* tailored to neighbors' interests.

Step 1: Start *push* iteration for nth time
Step 2: store ← *logs( )*
Step 3: Loop for all n *neighbors*
Step 4: while *(n.push)*
Step 5: Send *push* notification to *p*
Step 6: End while
Step 7: End Loop
Step 8: Stop

4.1.2 Algorithm for pop

We shall detail the *pull* procedures in this section. This protocol's major goal is to use every available slot, or to occupy every slot. A policy directs the sending of pull requests by transfer slots. The following function goes through every *file* hosted and determines which files are missing.

A *file* that is unfinished is one for which not all of the components have been sent. We iterate through all nearby hosts of the same file for each unfinished file. If a *transfer slot* becomes available, we use it to ask the neighbour for a piece. The pull processes are being started by peer n in our pseudo-code in order to ask peer *p* for a *piece*.

Step 1: Initialize the pull iteration for the nth time.

Step 2: Iterate over all files hosted by peer *n*.
Step 3: If the file is incomplete, proceed to the next steps.
Step 4: Iterate over all links associated with the file.
Step 5: Search for a free slot to use.
Step 6: Determine the specific piece to request based on a global piece policy.
Step 7: If the slot is idle, send a pull request, including the slot, piece, and file.
Step 8: End the slot check.
Step 9: End the loop for links.
Step 10: End the loop for files.
Step 11: End the pull iteration.
Step 12: Stop the process.

5. Experiment setup

5.1 Simulator

We employed the PeerSim simulator for our experiments due to its efficient memory utilization and effectiveness in simulating real-world scenarios (PeerSim P2P Simulator, 2009). We used customized PeerSim components to implement ou

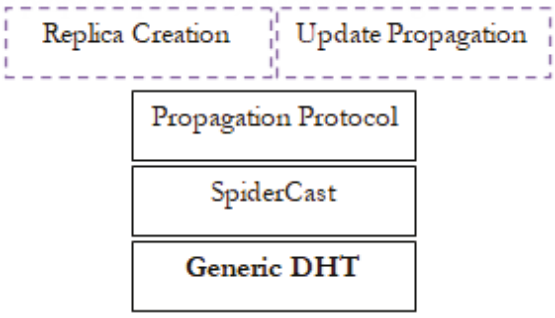

Figure 3
A simplified view of PeerSim components

5.1.1 PeerSim components

5.1.1.1 DHT

We simulated a DHT-lookup by returning pointers to peers using the DHT-lookup function. This method accelerated simulations without compromising precision. We created a large data transfer among the DHT-lookup origin and destination to mimic real-world bandwidth use. This optimization enabled more extended simulations that better captured real-world events (Ucar, Higuchi, & Altintas, 2019). Our approach and the standard master-slave protocol use replication strategies to propagate files

without DHT lookups. The revised parameters did not influence the amount of DHT lookups, but we changed the number of neighbors returned to account for more replicas.

5.1.1.2 Replication Policy

Our research focuses on developing an efficient file propagation protocol independently of the specific replication policy employed. Hence, our protocol can work in conjunction with any replication policy.

5.1.1.3 SpiderCast

The modifications to SpiderCast, as explained in Section 2.2, were successfully incorporated into our simulator. SpiderCast's performance will be assessed inferentially as part of our system.

5.1.1.4 Input Files

We categorize input files as Files, Peers, Events, and Parameters. These files are vital for the simulator, defining system files, peers, events, and parameter values necessary for execution.

5.1.1.5 Output Files

To avoid excessive log generation for uninteresting components, we turned off the logging of specific components. However, we focused on two significant output files:

Delay: The simulator records coordinators' activities, including delays, in log files for each file.

Consumption of Bandwidth: We developed a dedicated monitor to monitor the bandwidth usage associated with unique push and pull processes for each item. We store these logs as files upon completing the simulation.

5.2 Evaluation metrics

This study considers two crucial evaluation metrics: propagation time and overhead. These metrics allow us to assess our proposed protocol's performance compared to the standard master-slave protocol. Propagation time refers to the total time taken to disseminate a file from a coordinator to all peers within the DHT.

Control messages, that contribute significantly to overhead, were selected for evaluation, as outlined in Table I. Our protocol shares the same DHT, failure detection, and SpiderCast with the baseline MS protocol. Thus, the overhead associated with these three protocols does not impact the validity of our experiment. Table 1 reveals that our proposed protocol generates more pull

requests than its counterpart, resulting in higher overhead. However, measuring pull overhead is not warranted due to the negligible size of each pull message. Our proposed protocol efficiently propagates large files to all participating peers during propagation by maintaining necessary state information as bitmaps within the push header. Therefore, to measure the total overhead contributed by our protocol, we need to consider the size of the push message.

Table 1
Comparison between existing MS protocol and over proposed protocol

Overhead	Standard MS protocol	Our proposed protocol
Push	Control message exclusive of maps	Control message inclusive of maps
Pull	One large message	Single message for each piece
DHT	Unchanged	Unchanged
Failure Detection	Unchanged	Unchanged
SpiderCast	Unchanged	Unchanged

5.3 Dataset

We utilized a Wikipedia trace comprising images and pages to evaluate the performance of extensive file propagation. This dataset encompasses a mix of large and small files obtained by sampling HTTP requests to the server over several months (Urdaneta, 2011). The trace consists of two files: data on file types, sizes, and creation dates, and the other detailing updates or new creations. Note that the collection contains about 8,400 images that vary in size from 44 Bytes to 18 MB. Figure 4 shows the dataset's file size distribution. This section analyzes the crucial parameters in Section 5.5's sensitivity.

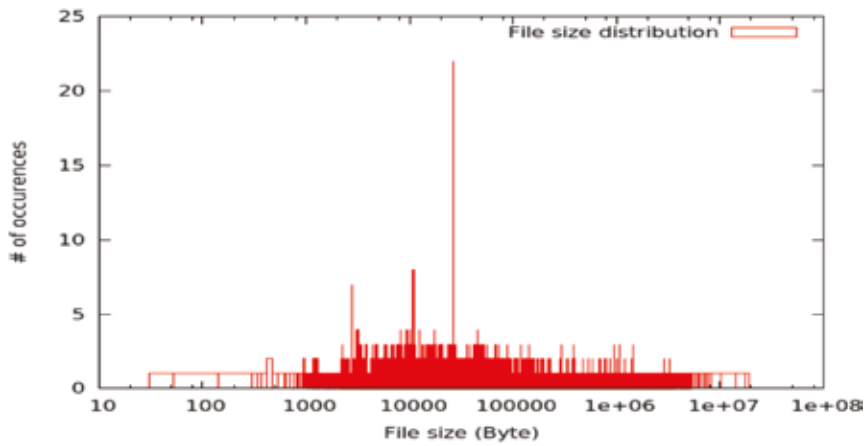

Figure 4
Distribution of files size in our dataset

5.4 Baseline

In our research, we compare P2PPP to two baselines.

5.4.1 Standard Master-Slave (MS)

The conventional master-slave protocol serves as our primary benchmark for comparison.

5.4.2 Lower-bound

The lower-bound baseline examines the minimum propagation time, where all peers utilize their entire bandwidth to propagate files. This benchmark is crucial for assessing collaborative propagation's performance, as it utilizes each peer's total bandwidth capacity. The lower-bound time for propagation depends on neighbor bandwidth (Taheri-Boshrooyeh, Hassanzadeh-Nazarabadi, & Ozkasap, 2020).

5.5 Important parameters and default values

To streamline our experimentation process and manage the multitude of parameters in the system, we assessed several parameters in isolation to study their impact on evaluation metrics. Table 2 outlines the most crucial variables for our system.

Table 2

Important evaluation criteria and their corresponding default values

System parameters	Default values
Number of active peers	10000
Peer-to-peer uniformity of bandwidth	Yes
Reversing a successful piece reception	Yes
Retort following piece reception failure	Yes
Iterative reactive pull after push	Yes
Timeout nodes	50s

Peer parameters	Default values
Peer bandwidth	30 KB/s
Number of transfer slots for peers	2
Transfer slot request policies	Random
Size of pieces in the system	3 KB
Frequency of push procedures	30s
Frequency of pull procedures	100s

Replication parameters	Default values
Peers returned via DHT-lookup	10
Number of replicas to create	7

SpiderCast parameters	Default values
Frequency of maintenance procedure	3 mins
Method of selecting neighbours	Greedy
Neighbour preference for each file	3
Maximum number of consecutive executions	15

Churn parameters	Default values
Availability skewness	None
Maximum peer availability	None
Mean offline time	None

6. Results and analysis

The primary objective of this study is to assess the speed at which large files propagate within DHTs using our proposed protocol compared to the existing master-slave protocol designed for the same purpose. Additionally, this experiment aims to quantify the reduction in overhead achieved by our proposed protocol. This section analyzes the crucial parameters in Section 5.5's sensitivity.

6.1 Default values

Our initial and critical evaluation metric focuses on the propagation delay introduced by our novel protocol within the DHT. Figure 5 presents the results obtained from ten iterations involving different seed sizes. The graph illustrates the mean propagation time based on default parameters across numerous experiments. The error bars within the graph represent the minimum and maximum delays observed when propagating seven copies of the same file within the DHT.

The propagation delay increases as file sizes grow. However, particular spikes are noticeable due to the inherent system for distributing events in the Master-Slave (MS) procedure. These spikes occur because files are introduced into the DHT with uneven timing, leading to irregular propagation patterns in the graph. An event trace was conducted randomly to mitigate this issue, introducing the duplicate files into the DHT randomized. Consequently, these spikes were eliminated from our dataset sourced from Wikipedia.

Figure 5 underscores the stability of our novel protocol concerning propagation delays. This stability can be attributed to the randomized selection of coordinators, which significantly alleviates the propagation delay by distributing a substantial portion of the load away from the coordinator.

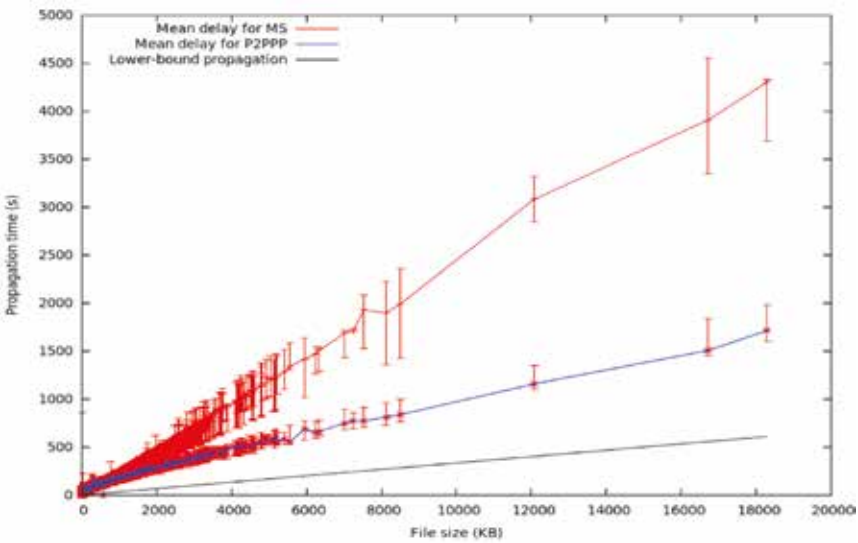

Figure 5
The propagation time for proposed default values

Our proposed protocol's evaluation of overhead introduced to the Distributed Hash Table (DHT) represents the second crucial metric in this research endeavor. Figure 6 comprehensively illustrates the bandwidth consumption attributed to our innovative protocol. During the propagation of a substantial file exceeding 18 MB, the most efficient utilization of push bandwidth recorded was slightly below 250 KB. Consequently, the overhead incurred during the propagation of the most extensive file via our protocol does not exceed 0.2%.

Notably, the graph exhibits some minor spikes, primarily attributable to increases in propagation time. An escalation in propagation delay necessitates transmitting a more significant number of push messages, which, in turn, contributes to the observed overhead.

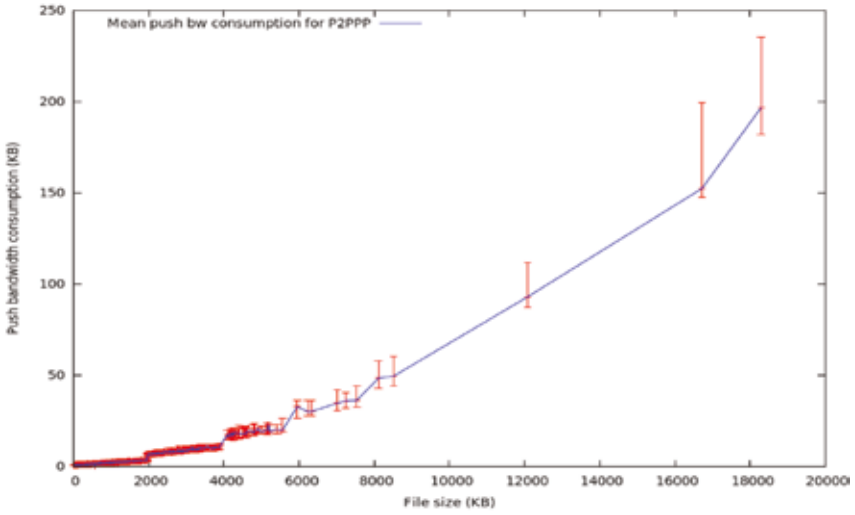

Figure 6
Push bandwidth consumed by our protocol

6.2 Transfer slots

Increasing the number of transfer slots during propagation boosts our protocol's parallelism. In this scenario, interested peers can concurrently execute multiple requests from their neighboring peers, reducing propagation time. Figure 7 visually represents the propagation time observed when propagating seven identical file copies, considering various available neighboring transfer slots, including 1, 2, 4, 6, and 8.

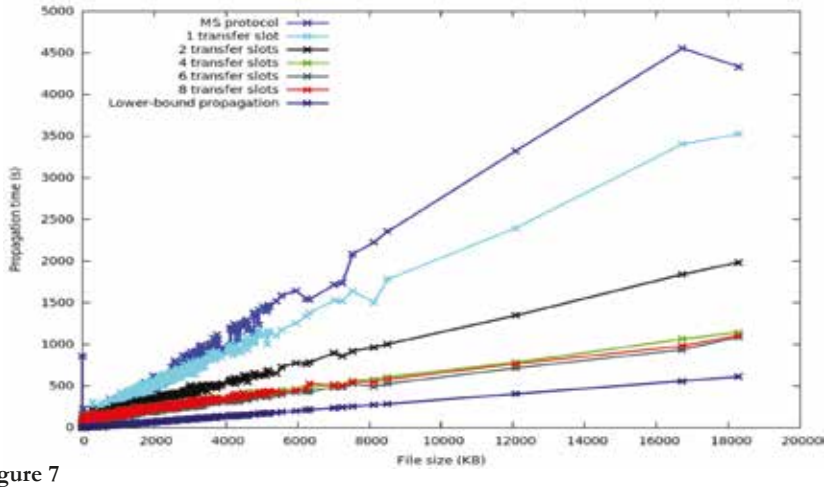

Figure 7
Transfer slots' impact on propagation time

The insights gleaned from Figure 7 reveal that a noticeable reduction in propagation time is achieved with an increasing number of available transfer slots. However, it is crucial to note that this enhanced parallelism does result in increased overhead attributed to the higher volume of request copies originating from a single neighbor. Intriguingly, the optimal balance between propagation time and overhead is struck when precisely six transfer slots are available from neighboring peers. Surprisingly, a significant reduction in propagation time is evident when each neighbor possesses only a single transfer slot.

In summary, our dataset demonstrates that the most efficient propagation time is achieved when six transfer slots are available, despite marginal disparities between 8 and 4 slots.

Figure 8 further elucidates the impact of different transfer slots on overhead. It becomes apparent that overhead diminishes as the number of transfer slots increases. This phenomenon can be attributed to the expedited propagation process, resulting in fewer push messages sent by participant nodes. Notably, the overhead consumption is minimized when six transfer slots are available in the dataset compared to other alternatives.

These outcomes are consistent with those observed in Figure 7, as propagation time and protocol overhead are closely intertwined. The highest level of overhead consumption, which remains below 350 KB, is recorded when a neighbor possesses only a single transfer slot. Overall, it is noteworthy that the overhead consumption does not exceed 0.3% in any scenario.

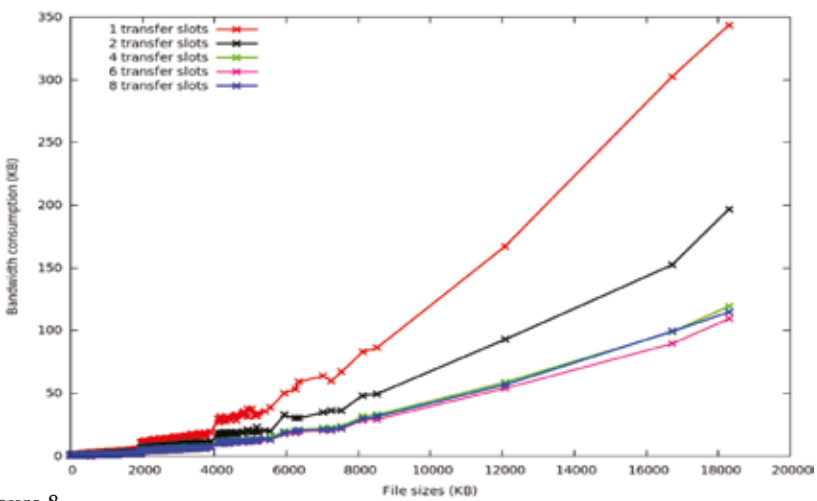

Figure 8

Effect of transfer slots on protocol overhead

6.3 Piece size

Here, we are going to examine how the alteration of component sizes affects propagation time and overhead usage. To do that default values will be used apart from changing the parameter of piece size. From figure 9, it can be seen that propagation time increased with the decrement of piece size. It is noted that negligible propagation time observed for over 100 KB piece size but highest propagation delay occured for 10 KB piece size.

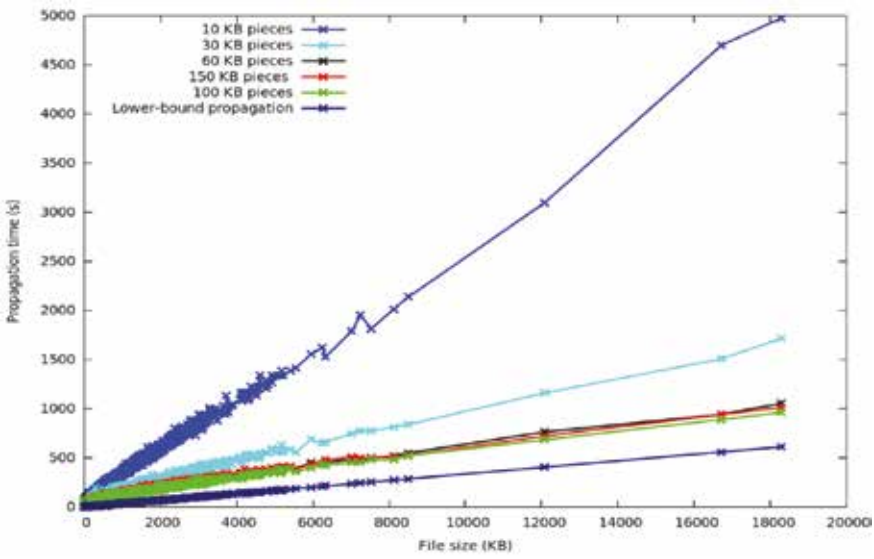

Figure 9

Propagation delay for different piece sizes

Following figure 10 illustrates consumption of overhead due to the push message of different piece sizes. The overhead is drastically increased to approximately 180 KB when having 30 KB piece size while less than 20 KB overhead has been consumed with the piece size of 150 KB. In relation to the volume of data propagated, less than 1.1% overhead has been consumed.

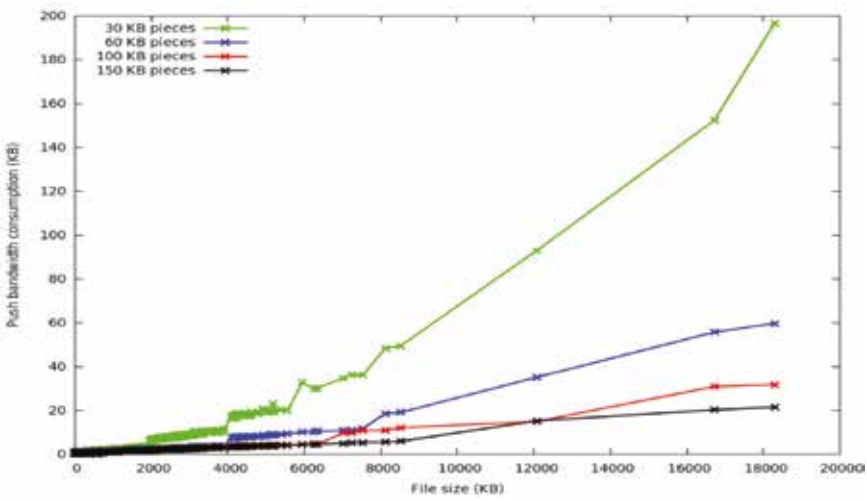

Figure 10
Median push bandwidth usage according to item size

6.4 Amount of replicas

Here, we only modify the number of created replicas and leave rest of the parameters remain unchanged. Following table 3 shows the values used for replication policy.

Table 3

Values the replication policy takes into account while selecting coordinators and replicas

Experiment No.	DHT Lookup Size	Replicas
1	7	5
2	10	7
3	13	10

It can be seen from the figure 11 that surprisingly, propagation time enlarges gradually with the increasing number of participants during propagation. For instance, 10 copies of file propagation have more spikes than propagating 5 copies. Figure 11 illustrates that in all 3 experiments with different replicas (5, 7 and 10), our proposed protocol performs well in term of holding lower propagation time than its counterpart MS protocol. The reason behind performing worse by MS protocol is that an entire file has to propagate to all the participating peers which lead to increasing propagation delay.

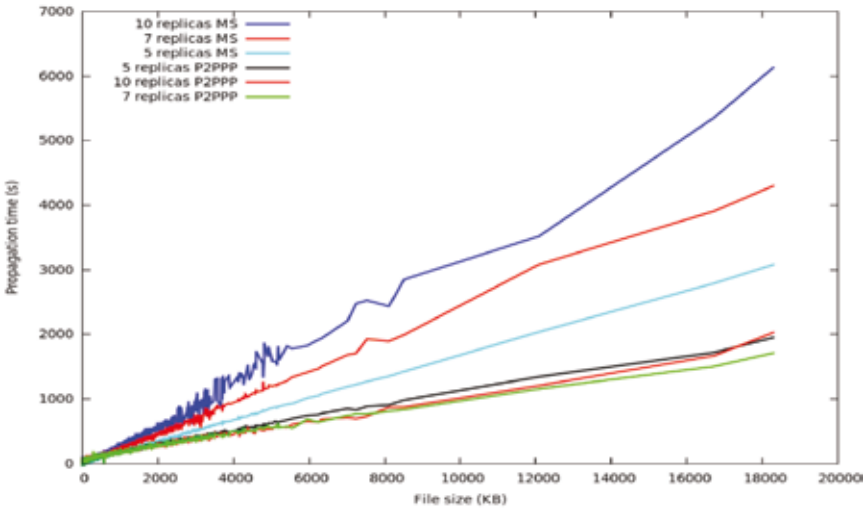

Figure 11
Propagation lag when 5, 7, and 10 clones are created

Figure 12 depicts the consumption of overhead for increasing number of participated peers. It can be seen from the figure that overhead consumption is proportional to the increasing number of replicas. For instance, highest amount of overhead was consumed for propagating 10 replicas where as negligible amount of overhead consumed for propagating 5 replicas within DHT. Overall, the consumption of overhead is slightly over 0.1% by our proposed protocol that is very negligible.

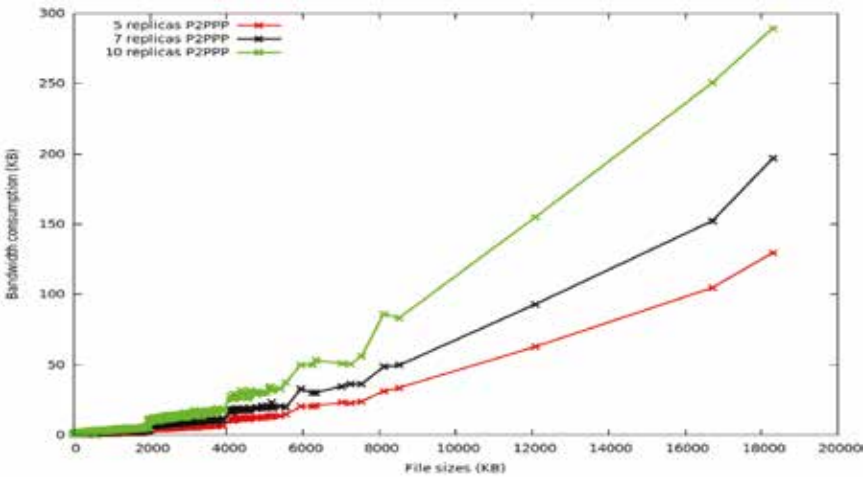

Figure 12
Overhead for the P2PPP protocol's growing participant population

6.5 Bandwidth

In order to get best possible result of propagation time and overhead caused by peer bandwidth, number of transfer slots and piece sizes should be modified from the default value mentioned in section 5.5. Table 4 shows the modified values of transfer slots and piece sizes during sensitivity analysis of peer bandwidth. Those parameters were chosen according to the amount of contributed bandwidth.

Table 4

Modified transfer slots and piece size

Experiment	Piece Size	Transfer Slots
1	30 KB	2
2	30 KB (default)	2 (default)
3	50 KB	4
4	150 KB	4

Following figure 13 shows the propagation time during the consumption of contributed bandwidth of peers. The graph shows that our proposed protocol perform less propagation delay than it master-slave counterpart.

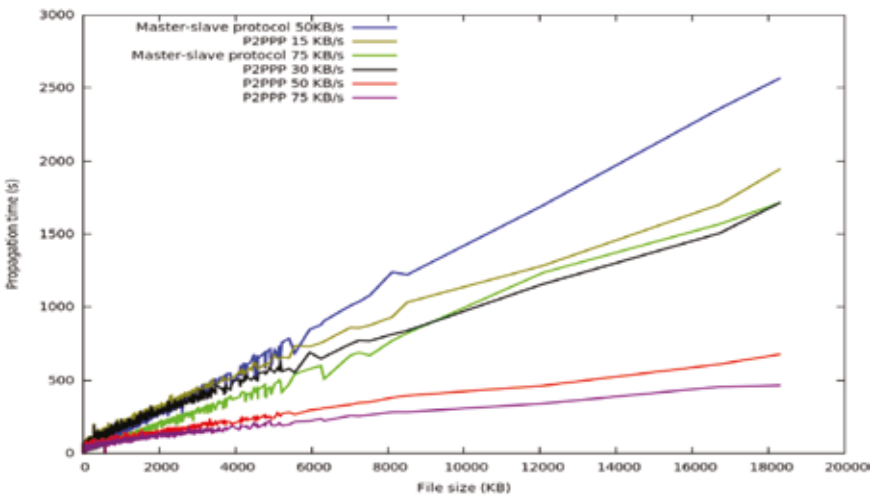

Figure 13

Propagation speed changes with peer bandwidth contribution

Figure 14 depicts the consumption of overhead due to the propagation of push messages during sensitivity analysis. It can be seen from the graph that there is a reciprocal relationship between overhead consumption and utilization of bandwidth. For instance, the consumption of overhead is highest when there is a lower utilization of peer bandwidth.

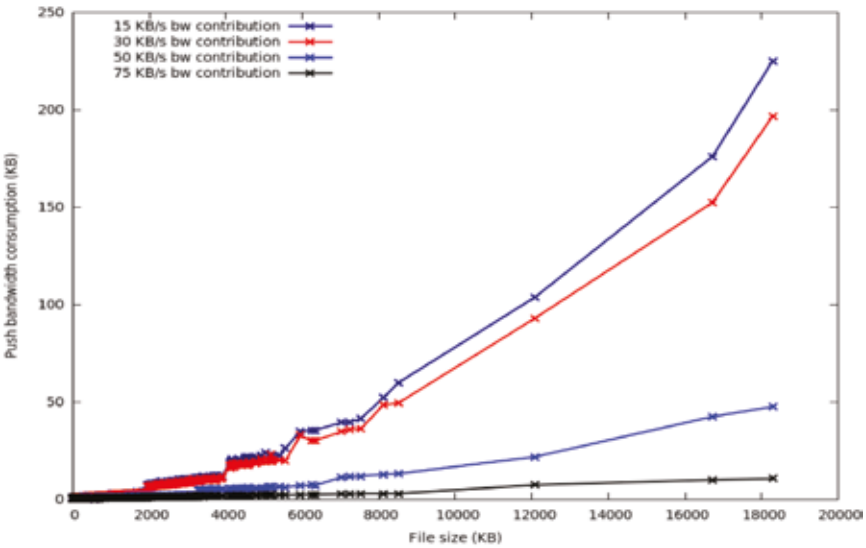

Figure 14
Effect of bandwidth contribution on overhead.

7. Conclusion

This section summarizes our study and discusses propagation time and overhead management challenges.

7.1 Propagation time

During the evaluation of our dataset, specific instances of elevated propagation time were identified. It is possible to employ various strategies to mitigate these spikes in propagation delay. One approach is mapping different items to different peers, thereby reducing the likelihood of propagation delays. Another potential solution is to modify the piece-pulling policy of peers, favoring faster peers for piece retrieval during propagation. Additionally, an alternative to the existing DHT lookup-based replication policy could involve implementing a more structured, non-random placement of replicas. Furthermore, considerations may include optimizing transfer slots and ensuring an appropriate piece size to minimize propagation delay. However, these optimizations must be made about peer bandwidth, aligning transfer slots and piece size with the available download bandwidth.

7.2 Overhead

Our evaluation has revealed that both bitmap size and propagation time are responsible for overhead consumption. Specifically, bandwidth utilization,

piece sizes, and push frequency are key factors contributing to overhead. For instance, it is unwise to fragment a 2 GB file into 5 KB pieces due to the extensive mapping required for file representation. Reducing the size of push messages is pivotal in limiting overhead consumption; hence, the number of maps within a push message should be restrained. Moreover, optimizing push frequency may be necessary to minimize overhead.

7.3 Future directions

The proposed protocol exhibits significant potential for efficiently propagating multimedia applications. In future research, improvements in piece and specific file selection policies could be explored to enhance the seamless delivery of user applications. For instance, a linear piece policy could be implemented for streaming services, ensuring file pieces' sequential ordering or retrieval. Furthermore, researchers in this field can utilize this article as a valuable reference for further exploration and study.

References

Franchi, E., & Poggi, A. (2019). Blogracy: A peer-to-peer social network. In *Censorship, surveillance, and privacy: concepts, methodologies, tools, and applications* (pp. 675-696). IGI global. doi: 10.4018/978-1-5225-7113-1.ch063.

Galuba, W., & Girdzijauskas, S. (2009). Peer-to-Peer System. In: LIU, L., ÖZSU, M.T. (eds) *Encyclopaedia of Database Systems*. Boston, MA; Springer. https://doi.org/10.1007/978-0-387-39940-9_1230.

Gupta, R. K., Hada, R., & Sudhir, S. (2017). *2-tiered cloud-based content delivery network architecture: An efficient load balancing approach for video streaming*. Paper presented at the 2017 International Conference on Signal Processing and Communication (ICSPC), (pp. 431-435). doi: 10.1109/CSPC.2017.8305885.

Hassanzadeh-Nazarabadi, Y., Kupcu, A., & Ozkasap, O. (2020). Interlaced: Fully decentralized churn stabilization for skip graph-based DHTs. *Journal of Parallel and Distributed Computing, 149,* 13–28. https://doi.org/10.1016/j.jpdc.2020.10.008.

Hassanzadeh-Nazarabadi, Y., Kupcu, A., & Ozkasap, O. (2021). Lightchain: Scalable DHT-based blockchain. *IEEE Transactions on Parallel and Distributed Systems, 32*(10), 2582-2593. doi: 10.1109/TPDS.2021.3071176.

Hentschel, A., Hassanzadeh-Nazarabadi, Y., Seraj, R., Shirley, D., & Lafrance, L. (2020). Flow: Separating consensus and compute–block formation and execution. arXiv preprint arXiv:2002.07403, 1 -41, https://doi.org/10.48550/arXiv.2002.07403.

Kaur, R., Gabrijelcic, D., & Klobucar, T. (2022). Churn handling strategies to support dependable and survivable structured overlay networks. *IETE Technical Review, 39*(1), 179-195, doi: 10.1080/02564602.2020.1830001.

Kwon, G., & Ryu, K. D. (2004). BYPASS: *Topology-aware lookup overlay for DHT-based P2P file locating services.* Proceedings of the Tenth International Conference on Parallel and Distributed Systems, ICPADS 2004, (pp. 297-304). doi: 10.1109/ICPADS.2004.1316108.

Lamport, L. (2019). Time, clocks, and the ordering of events in a distributed system. In *Concurrency: The Works of Leslie Lamport* (pp. 179-196). doi:10.1145/3335772.3335934.

Maymounkov, P., & Mazières, D. (2002). Kademlia: A Peer-to-Peer information system based on the XOR metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds) *Peer-to-Peer Systems. IPTPS 2002. Lecture Notes in Computer Science* (pp. 2429). Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-45748-8_5.

Nakayama, T., & Asaka, T. (2017). *Peer-to-peer bidirectional streaming using mobile edge computing.* Paper presented at the 2017 Fifth International Symposium on Computing and Networking (pp. 263-266), IEEE. doi: 10.1109/CANDAR.2017.38.

Nasir, M., Muhammad, K., Bellavista, P., Lee, M. Y., & Sajjad, M. (2020). Prioritization and alert fusion in distributed IoT sensors using Kademlia based distributed hash tables. *IEEE Access, 8,* 175194-175204, doi: 10.1109/ACCESS.2020.3017009.

Nguyen, B. M., Hoang, H. N. Q., Hluchy, L., Vu, T. T., & Le, H. (2017). Multiple peer chord rings approach for device discovery in IoT environment. *Procedia Computer Science, 110,* 125-134. https://doi.org/10.1016/j.procs.2017.06.133.

PeerSim P2P Simulator. (2009). Retrieved from http://peersim.sourceforge.net/

Raj, S., & Rajesh, R. (2016). *Descriptive analysis of hash table-based intrusion detection systems.* Paper presented at the 2016 International Conference on Data Mining and Advanced Computing (pp. 233-240), IEEE. doi: 10.1109/SAPIENCE.2016.7684112.

Rodrigues, R., & Druschel, P. (2010). Peer-to-Peer Systems. *Communications of the ACM, 53*(10), 72-82. doi:10.1145/1831407.1831427.

Sonbol, K., Ozkasap, O., Al-Oqily, I., & Aloqaily, M. (2020). EdgeKV: Decentralized, scalable, and consistent storage for the edge. *Journal of Parallel and Distributed Computing, 144,* 28 – 40, https://doi.org/10.1016/j.jpdc.2020.05.009.

Stein, C. A., Tucker, M. J., & Seltzer, M. I. (2002). *Building a reliable mutable file system on peer-to-peer storage.* Paper presented at the 21st IEEE Symposium on Reliable Distributed Systems (pp. 324-329). doi: 10.1109/RELDIS.2002.1180204.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., & Balakrishnan, H. (2003). Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking, 11*(1), 17-32. https://doi.org/10.1109/TNET.2002.808407.

Taheri-Boshrooyeh, S., Hassanzadeh-Nazarabadi, Y., & Ozkasap, O. (2020). *A proof-of-concept implementation of guard ¨ secure routing protocol.* Paper presented at the 2020 International Symposium on Reliable Distributed Systems (SRDS) (pp. 332-334). doi: 10.1109/SRDS51746.2020.00041.

Tahir, A., Abid, S. A., & Shah, N. (2017). Logical clusters in a DHT paradigm for scalable routing in MANETs. *Computer Networks, 128,* 142-153. https://doi.org/10.1016/j.comnet.2017.05.033.

Tran, M. H., Nguyen, V.S., & Ha, S.V.U. (2016). Decentralized online social network using peer-to-peer technology. *REV Journal on Electronics and Communications, 5*(1-2). doi: http://dx.doi.org/10.21553/rev-jec.95.

Ucar, S., Higuchi, T., & Altintas, O. (2019). *Collaborative data storage by a vehicular micro cloud.* Paper presented at the 2019 IEEE Vehicular Networking Conference (VNC) (pp. 1-2). doi: 10.1109/VNC48660.2019.9062818.

Urdaneta, G. A. (2011). Collaborative wikipedia hosting. Published PhD thesis: Vrije Universiteit, Amsterdam. doi: https://research.vu.nl/en/publications/collaborative-wikipedia-hostin.

Xie, X.-L., Wang, Q., & Wang, P. (2017). *Design of smart container cloud based on DHT*. Paper presented at the 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNCFSKD) (pp. 2971-2975). doi: 10.1109/FSKD.2017.8393255.

Zahid, S., Abid, S. A., Shah, N., Naqvi, S. H. A., & Mehmood, W. (2018). Distributed partition detection with dynamic replication management in a DHT-based MANET. *IEEE Access, 6,* 18731-18746. doi: 10.1109/ACCESS.2018.2814017.

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., & Kubiatowicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications, 22*(1), 41-53. doi: 10.1109/JSAC.2003.818784.

Zhou, M., Dai, Y., & Li, X. (2006). *A measurement study of the structured overlay network in P2P file-sharing applications.* Paper presented at the Eighth IEEE International Symposium on Multimedia (ISM'06) (pp. 621-628). doi: 10.1109/ISM.2006.5.

Corresponding author
MD Jiabul Hoque can be contacted at: jiabul.hoque@iiuc.ac.bd