Efficient Communication Protocols for Non DHT-based Pyramid Tree P2P Architecture

[1]INDRANIL ROY, [1]SWATHI KALUVAKURI, [1]KOUSHIK MADDALI, [2]ZIPING LIU, and [1]BIDYUT GUPTA,

[1]School of Computing
Southern Illinois University
Carbondale, IL, USA

[2]Department of Computer Science
Southeast Missouri State University
Cape Girardeau, MO, USA

Abstract : In this paper, we have considered a recently reported 2-layer non-DHT-based structured P2P network. Residue Class based on modular arithmetic has been used to realize the overlay topology. At the heart of the architecture (layer-1), there exists a tree like structure, known as pyramid tree. It is not a conventional tree. A node *i* in this tree represents the cluster-head of a cluster of peers which are interested in a particular resource of type R_i (i.e. peers with a common interest). The cluster-head is the first among these peers to join the system. Root of the tree is assumed to be at level 1. Such a tree is a complete one if at each level j, there are j number of nodes. It is an incomplete one if only at its leaf level, say k, there are less than k number of nodes. Layer 2 consists of the different clusters. The network has some unique structural properties, e.g. each cluster has a diameter of only 1 overlay hop and the diameter of the network is just (2+2d); d being the number of levels of the layer-1 pyramid tree and d depends only on the number of distinct resources. Therefore, the diameter of the network is independent of the number of peers in the whole network. In the present work, we have used some such properties to design low latency intra and inter cluster data lookup protocols. Our choice of considering non-DHT and interest-based overlay networks is justified by the following facts: 1) intra-cluster data lookup protocol has constant complexity and complexity of inter-cluster data lookup is O(d) if tree traversal is used and 2) search latency is independent of the total number of peers present in the overlay network unlike any structured DHT- based network (as a matter fact unlike any existing P2P network, structured or unstructured). Experimental results as well show superiority of the proposed protocols to some noted structured networks from the viewpoints of search latency and complexity involved in it. In addition, we have presented in detail the process of handling churns and proposed a simple yet very effective technique related to cluster partitioning, which, in turn, helps in reducing the number of messages required to be exchanged to handle churns.

Keywords: Structured P2P network, interest-based systems, residue class, pyramid tree, communication protocol, churn

Received: February 15, 2021. Revised: June 29, 2021. Accepted: July 15, 2021. Published: July 23, 2021.

1. Introduction

Peer-to-Peer (P2P) overlay networks are widely used in distributed systems due to their ability to provide computational and data resource sharing capability in a scalable, self-organizing, distributed manner. There are two classes of P2P networks: unstructured and structured ones. In unstructured systems [2] peers are organized into arbitrary topology. It takes help of flooding for data look up. Problem arising due to frequent peer joining and leaving the system, also known as churn, is handled effectively in unstructured systems. However, it compromises with the efficiency of data query and the much-needed flexibility. Besides, in unstructured networks, lookups are not guaranteed. On the other hand, structured overlay networks provide deterministic bounds on data discovery. They provide scalable network overlays based on a distributed data

structure which actually supports the deterministic behavior for data lookup. Recent trend in designing structured overlay architectures is the use of distributed hash tables (DHTs) [3] - [5]. Such overlay architectures can offer efficient, flexible, and robust service [3] - [5], [7], [8]. However, maintaining DHT-based structures in presence of churn is a complex task and needs substantial amount of effort to handle the problem of churn. So, the major challenge facing such architectures is how to reduce this amount of effort while still providing an efficient data query service. In this direction, there exist several important works, which have considered designing DHT-based hybrid systems [1], [6], [9] - [11]; these works attempt to include the advantages of both structured and unstructured architectures. However, these works have their own pros and cons. Another design approach has attracted much attention; it is non-DHT based structured approach [12], [14]-[18], [26]. It offers advantages of DHT-based systems, while it attempts to reduce the complexity involved in churn handling. Authors in [16], [17] have considered one such approach and have used an already existing architecture, known as Pyramid tree architecture, originally applied to the research area of 'VLSI design for testability' [13]. In the present work, we have considered such an architecture as a part of the overlay network to design efficient communication protocols.

1.1 Our contribution

Our main objective is to show the superiority of our non-DHT and interest-based architecture over DHT-based architectures from the viewpoints of search latency and data look up complexity. We have considered a number theoretic approach to build the architecture. Our choice of considering such an architecture is justified by the following facts: 1) intra-cluster data lookup protocol has constant complexity and complexity of inter-cluster data lookup is $O(d)$ if tree traversal is used, d is the number of levels of the tree and number of distinct resource types, $n \approx 2^d$ and 2) search latency is independent of the total number of peers present in the overlay network unlike any structured DHT-based network (as a matter fact unlike any P2P network, structured or unstructured). Experimental results obtained as well show superiority of the proposed protocols to some noted structured networks from the viewpoints of search latency and complexity involved in it. In addition, we have presented in detail the process of handling churns and proposed a simple yet very effective technique related to cluster partitioning, which, in turn, helps in reducing the number of messages required to be exchanged to handle churns. Most of the existing interest-based architectures [19]-[25] are built on an ad-hoc basis without having any solid mathematical foundation. Hence, we have highlighted the main differences of our interest-based architecture with some such existing ones using architectural aspects only and the related pros and cons of those architectures.

The organization of the paper is as follows. In Section 2, we talk about some related preliminaries from our recently reported works. In Section 3, we present the data lookup protocols and a comparison of their performance with some noted DHT-based systems. In Section 4, we have presented our works on churn handling; we have presented an effective way of cluster partitioning in order to reduce the number of messages needed to restructure the network after peers join and leave. In Section 5, to we have compared our proposed work with some existing interest-based works from the viewpoint of the main features of the different architectures considered in these works. Section 6 draws the conclusion.

2. Preliminaries

In this section, we present some relevant results from our recent work on the Pyramid tree based P2P architecture [16], [17] for interest-based peer-to-peer system.

Definition 1. *We define a resource as a tuple $<R_i, V>$, where R_i denotes the type of a resource and V is the value of the resource.*

Note that a resource can have many values. For example, let R_i denote the resource type 'songs' and V' denote a particular singer. Thus $<R_i, V'>$ represents songs (some or all) sung by a particular singer V'.

Definition 2. *Let S be the set of all peers in a peer-to-peer system with n distinct resource types (i.e. n distinct common interests). Then $S = \{C_i\}$, $0 \le i \le n-1$, where C_i denotes the subset consisting of all peers with the same resource type R_i. In this work, we call this subset C_i as cluster i. Also, for each cluster C_i, we assume that C_i^h is the first peer among the peers in C_i to join the system. We call C_i^h as the cluster-head of cluster C_i.*

2.1 Pyramid Tree

The following overlay architecture has been proposed in [16].

1) The tree consists of n nodes (n resource types). The i^{th} node is the i^{th} cluster head C_i^h. The tree forms the layer-1 and the clusters corresponding to the cluster-heads form the

layer-2 of the architecture. Also, $n \approx 2^d$ where d is the number of levels of the tree.

2) Root of the tree is at level 1.
3) Edges of the tree denote the logical link connections among the n cluster-heads. Note that edges are formed according to the pyramid tree structure [13].
4) A cluster-head C_i^h represents the cluster C_i. Each cluster C_i is a completely connected network of peers possessing a common resource type R_i, resulting in the cluster diameter of 1.
5) The tree is a complete one if at each level j, there are j number of nodes (i.e. j number of cluster-heads). It is an incomplete one if only at its leaf level, say k, there are less than k number of nodes.
6) Any communication between a peer $p_i \in C_i$ and a peer $p_j \in C_j$ takes place only via the respective cluster-heads C_i^h and C_j^h and with the help of tree traversal wherever applicable.
7) Joining of a new cluster always takes place at the leaf level.
8) A node that does not reside either on the left branch or on the right branch of the root node is an internal node.
9) Degree of an internal non-leaf node is 4.
10) Degree of an internal leaf node is 2.
11) Diameter of the network is (2+2d); d is the number of levels of the pyramid tree at layer 1.

2.2 Residue Class

Modular arithmetic has been used to define the pyramid tree architecture of the P2P system.
Consider the set S_n of nonnegative integers less than n, given as $S_n = \{0, 1, 2, \ldots (n-1)\}$. This is referred to as the set of residues, or residue classes (mod n). That is, each integer in S_n represents a residue class (RC). These residue classes can be labelled as [0], [1], [2], …, [n – 1], where $[r] = \{a: a \text{ is an integer, } a \equiv r \pmod n\}$.

For example, for n = 3, the classes are:
[0] = {…., −6, −3, 0, 3, 6, …}
[1] = {…., −5, −2, 1, 4, 7, …}
[2] = {…., −4, −1, 2, 5, 8, …}

In the P2P architecture, we use the numbers belonging to different classes as the logical (overlay) addresses of the peers with a common interest (i.e. peers in the same cluster) and the number of residue classes is the number of distinct resource types; for the sake of simplicity we shall use only the positive integer values.

Before we present the mechanism of logical address assignments, we state the following relevant property of residue class.

Lemma 1. Any two numbers of any class r of S_n are mutually congruent.

Proof. Let us consider any two numbers N' and N'' of class r. these numbers can be written as
$N' \equiv r \pmod n$; therefore, $(N' - r) / n$ = an integer, say I' (1) and
$N'' \equiv r \pmod n$; therefore, $(N'' - r) / n$ = an integer, say I'' (2)
Using (1) and (2) we get the following, $(N' - N'') / n = ((N' - r) - (N'' - r)) / n = I' - I''$ = an integer.
Therefore, N' is congruent to N''; that is, $N' \equiv N'' \pmod n$; also, $N'' \equiv N' \pmod n$ because congruence relation (≡) is symmetric. Hence, the proof. □

2.3 Assignments of Overlay (Logical) Addresses

Assume that in an interest-based P2P system there are n distinct resource types. Note that n can be set to an extremely large value a priori to accommodate large number of distinct resource types. Consider the set of all peers in the system given as $S = \{C_i\}$, $0 \le i \le n-1$. Also, as mentioned earlier, for each subset C_i (i.e. cluster C_i) peer C_i^h is the first peer with resource type R_i to join the system and hence, it is the cluster-head of cluster C_i.
The assignment of overlay addresses to the peers in the clusters and the resources happens as follows:
1) The first cluster-head to join the system is assigned with the logical (overlay) address 0 and is denoted as C_0^h. It is also the root of the tree formed by newly arriving cluster-heads (see the example in Fig. 1).
2) The $(i+1)^{th}$ newly arriving cluster-head possessing the resource type R_i is denoted as C_i^h and is assigned with the minimum nonnegative number (*i*) of *residue class i (mod n)* of the residue system S_n as its overlay address.
3) In this architecture, cluster-head C_i^h is assumed to join the system before the cluster-head C_{i+1}^h.
4) All peers having the same resource type R_i (i.e. 'common interest' defined by R_i) will form the cluster C_i. Each new peer joining cluster C_i is given the cluster membership address (i + j.n), for i = 0, 1, 2, …
5) Resource type R_i possessed by peers in C_i is assigned the code *i* which is also the logical address of the cluster-head C_i^h of cluster C_i.

Definition 3. *Two peers of a cluster C_r are logically linked together if their assigned logical addresses are mutually congruent.*

Lemma 2. Each cluster C_r forms a complete graph.

Proof. According to Definition 3, two peers of any cluster C_r are logically linked together if their assigned logical addresses are mutually congruent. Also, from Lemma 1, we note that any two numbers of any class r of S_n are mutually congruent. Therefore, every peer has direct logical connection with every other peer in the same cluster C_r. Hence, the proof. □

Observation 1. *Any intra-cluster data look up communication needs only one overlay hop.*
Observation 2. *Search latency for inter-cluster data lookup algorithm is bounded by the diameter of the tree.*

2.4 Scalability

It may be noted that number of distinct resource types is very small compared to the number of peers in any overlay network [28]. To avoid the possibility of redesigning the architecture as new clusters are formed, a very large value of n can be selected at the design phase to accommodate very large number of possible resource types (if needed in future). It means that if at the beginning number of resource types present is small, only the first few of the residue classes will be used initially for addressing; and as new clusters are formed in future with new resource types in the system, more residue classes in sequence will be available for their addressing. For example, say initially n is set at 1000; so, there are 1000 possible residue classes, starting from [0], [1], [2], [4], [5], …., [999]. If initially there are only three clusters of peers present with three distinct resource types, the residue classes [0], [1], [2] will be used for addressing the peers in the three respective clusters. If later two new clusters are formed with two new resource types, the residue classes [3] and [4] will be used for addressing the pears in the two new clusters in sequence of their joining the system. Moreover, as we see, there is no limit on the size of any cluster because any residue class can be used to address logically up to infinite number of peers with a common interest. Therefore, the proposed architecture does not have any negative issue with scalability.

2.5 Virtual Neighbors [17]

An example of a complete pyramid tree of 5 levels is shown in Fig. 1. It means that it has 15 nodes/clusters (clusters 0 to 14, corresponding to 15 distinct resource types owned by the 15 distinct clusters). It also means that residue class with *mod 15* has been used to build the tree. The nodes' respective logical addresses are from 0 to 14 based on their sequence of joining the P2P system.

Each link that connects directly two nodes on a branch of the tree is termed as a *segment.* In Fig. 1, a bracketed integer on a segment denotes the difference of the logical addresses of the two nodes on the segment. It is termed as *increment* and is denoted as *Inc* This increment can be used to get the logical address of a node from its immediate predecessor node along a branch. For example, let X and Y be two such nodes connected via a segment with increment *Inc*, such that node X is the immediate predecessor of node Y along a branch of a tree which is created using *residue class with mod n.* Then, *logical address of Y = (logical address of X + Inc) mod n*.

Thus, in the example of Fig. 1,
Logical address of the leftmost leaf node = (logical address of its immediate predecessor along the left branch of the root + Inc) mod 15 = (6 + 4) mod 15 = 10.

Also, note that a *left branch* originating at node 2 on the right branch of the root node is *2 → 4 → 7 → 11*. Similarly, we can identify all other left branches originating at the respective nodes on the right branch of the root node. In a similar way, we can identify as well all right branches originating at the respective nodes on the left branch of the root node as well.

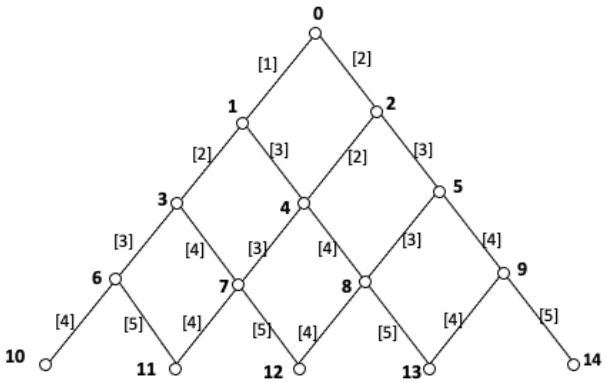

Fig. 1 A complete pyramid tree with root 0

Remark 1. *The sequence of increments on the segments along the left branch of the root, appears to form an AP series with 1st term as 1 and common difference as 1.*
Remark 2. *The sequence of increments on the segments along the right branch of the root, appears to form an AP series with 1st term as 2 and common difference as 1.*
Remark 3. *Along the 1st left branch originating at node 2, the sequence of increments appears to form an AP series with 1st term as 2 and common*

difference as 1. Note that the 1st term is the increment on the segment 0 → 2.

Remark 4. *Along the 2nd left branch originating at node 5, the sequence of increments is an AP series with 1st term as 3 and common difference as 1. Note that the 1st term is the increment on the segment 2 → 5.*

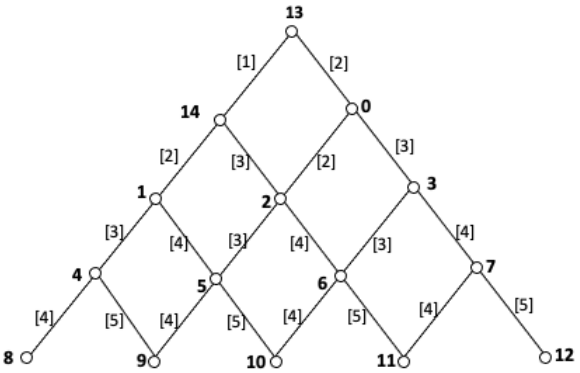

Fig. 2 A complete pyramid tree with root 13

Authors [17] have presented some important structural properties of the pyramid tree P2P system. According to the authors, no existing structured P2P system, either DHT or non-DHT based, possesses these properties. These are stated below.

Let S_Y be the set of logical links that connect a node Y to its neighbors in a complete pyramid tree T_R with root R. Assume that the tree has n nodes (i.e. n cluster heads / n clusters). Let another tree T'_R be created with the same n nodes but with a different root R'. Let S'_Y be the set of logical links connecting Y to its neighbors in the tree T'_R.

Property 1. $S_Y \neq S'_Y$

Property 2. *Diameter of T_R = Diameter of T'_R*

Property 3. *Number of levels of T_R = Number of levels of T'_R*

Property 4. *Complexity of broadcasting in T_R with root R as the source of broadcast is the same for T'_R with root R'*

Property 5. *Both T_R and T'_R are complete pyramid trees.*

An example: Consider the complete pyramid tree of 5 levels as shown in Fig. 2. Note that root of this tree is node 13, whereas root of the tree of Fig. 1 is 0.

It is seen that $S'_4 = \{1,8,9\}$ and $S_4 = \{1,2,7,8\}$. Therefore, Property 1 holds.

Diameters of both trees are same; it is 8 in terms of number of overlay hops. Besides, both trees use the same 15 nodes and have the same total number of levels. Complexity of broadcasting from either root 0 in the tree of Fig. 1 or from root 13 in the tree of Fig. 2 is bounded by the number of levels of each of the trees (here it is 4 in each). Finally, both trees are complete pyramid trees. Thus, all properties as mentioned above hold. These properties have been used to design very high bandwidth efficient inter-cluster broadcast protocols for the overlay network with both complete and incomplete pyramid trees at layer-1 of the architecture [27].

Remark 5. *Set of the neighbors of a given node Z may vary as the root of the tree varies. Hence, it is termed 'virtual'. However, time complexity of broadcasting remains same, i.e. it is O*(d) where *d* denotes the number of levels of the tree [27].

3. Data Look-up

Data lookup can be either intra-cluster or inter-cluster. The former one means that a peer p_i' (ϵ C_i) requests for some resource $<R_i,V''>$ which it does not possess. Note that only some peer(s) p_i'' (ϵ C_i) can possess $<R_i,V''>$ if at all; no other peer in any other cluster C_k can possess it since it is an interest based architecture.

The inter-cluster data lookup is invoked when a peer p_i' (ϵ C_i) requests for resource $<R_j,V^*>$, that can only be possessed, if at all, by some peer p_j' in cluster C_j.

The following data structures will be used for efficient data lookup. Every cluster-head C_i^h will maintain its table of information (*TOI*) in the following way. We assume that when a new cluster-head joins an existing tree, it contacts cluster-head 0, i.e. C_0^h which, in turn, assigns the newly joined one with the next logical address available for assignment (i.e. the next integer in the set $S_n = \{0, 1, 2,\ldots (n-1)\}$, not yet assigned to any cluster-head). Therefore, this logical address becomes the largest address assigned so far in the tree. C_0^h broadcasts this information to all other cluster-heads in the tree [27]. Each receiving node, C_i^h then updates its table of information (*TOI*) that contains a tuple for each node C_k^h (cluster-head h). The tuple for C_k^h appears as

< logical address, IP address, resource code of the resource owned by C_k^h >. Recall that C_0^h is the first one to arrive and it forms the tree with only one node. It is seen that the table remains sorted after addition of any latest entry based on the logical addresses. Also, note that the code of a resource type R_i is the same as the logical address of the corresponding cluster-head C_i^h.

In addition, each member peer in a cluster C_i maintains a list of all its neighbors present in the cluster. Each entry is a tuple consisting of a peer's logical and IP addresses. This list remains sorted w.r.to the logical addresses of the peers joining C_i because any new peer joining this cluster will have the next highest available address from the residue

class [i] used for addressing the peers in the cluster C_i. This information helps to achieve fault tolerance in the event that P_i crashes or leaves (churn handling).

3.1 Intra-Cluster Data Lookup

Without any loss of generality, let us consider a data lookup in cluster C_i by a peer p' possessing $<R_i,V'>$ and requesting for $<R_i,V''>$. The protocol appears below.

Protocol Intra-Data-Lookup

1. *p'* broadcasts its request in C_i for $<R_i,V''>$

/ one hop communication since diameter of C_i is one

2. if ∃ *p''* with $<R_i,V''>$
   p'' unicasts $<R_i,V''>$ to p'
   else
   search for $<R_i,V''>$ fails

/ search latency is minimum, i.e. only two hops

3.2 Inter-Cluster Data Lookup

In our proposed architecture, any communication between a node $p_i \in C_i$ and $p_j \in C_j$ takes place only via the respective cluster-heads C_i^h and C_j^h. Without any loss of generality let a peer p_i' ($\in C_i$) request for $<R_j,V^*>$. Note that Peer p_i' knows that $R_j \notin C_i$.

Protocol Inter-Data-Lookup

1. p_i' sends a data lookup request for $<R_j,V^*>$ to its cluster-head C_i^h
   / one hop communication
2. C_i^h determines the cluster-head C_j^h's IP address from its *TOI* using C_j^h's resource code
   / logical address of C_j^h = resource code of $R_j = j$
   C_i^h unicasts the request to C_j^h
3. if C_j^h possesses $<R_j,V^*>$
   C_j^h unicasts $<R_j,V^*>$ to p_i'
   else
   C_j^h broadcasts the request for $<R_j,V^*>$ in C_j
   / one hop communication
4. if ∃ p_j'' ($\in C_j$) with $<R_j,V^*>$
   p_j'' unicasts $<R_j,V^*>$ to p_i'
   else
   C_j^h unicasts 'search fails' to p_i'

3.3 Data Lookup Complexity

Analytical comparison with some noted DHT-based P2P networks [3]-[5] etc. has appeared in Table 1. Observe that our non DHT-based system offers much better search latency than DHT-based systems. In addition, in the present work, intra-cluster data lookup protocol has constant complexity and complexity of inter-cluster lookup is $O(d)$ if tree traversal is used; otherwise a maximum of only four hops is required. Note that $n \approx 2^d$ and hence d is even much smaller than n, the number of distinct resource types. Anyway, search latency is independent of the number of peers in the whole network unlike the works in [3]-[5]. Thus, we infer that the proposed structured interest-based overlay architecture can be the choice over the non-interest-based structured ones, mainly because of its much smaller search latency.

The point to mention is that use of the same code to denote a resource type R_i and the corresponding cluster-head C_i has made the search process simple and efficient. Note that we do not need to save both logical address as well as the resource code in each entry of *TOI*. One of these two will denote the other one. Therefore, we save only one of the two in addition to the IP address of a cluster-head. It reduces the table size. Besides, only the number of distinct resource types limits the size of the *TOI* table. Fact is, total number of peers N is much larger than the number of distinct resource types n [28] and the size of the *TOI* is independent of the total number of peers present in the P2P system. Besides, *TOI* grows dynamically as new cluster-heads join; therefore, the newly joining one will always have the largest logical address. Hence, *TOI* always remains sorted with respect to the logical addresses of the cluster-heads; thereby any searching in *TOI* becomes very efficient.

Observe that a cluster-head unicasts to another cluster-head in case of inter-cluster data look up. It makes the communication protocol simpler than following the idea of tree traversal to reach a destination cluster. However, even if tree traversal is followed, search latency is independent of the total number of peers present in the overlay system as is proven below in the following theorem.

Theorem 1. *Search latency for inter-cluster data lookup algorithm is bounded by the diameter of the tree* and is independent of the total number of peers present in the system.

Proof. Let us consider a peer p_s in a cluster C_m as the source of unicast communication. Let us consider a destination peer p_d in a cluster C_k. To maximize latency, we assume that both p_s and p_d are not cluster-heads. So, it takes one overlay hop for the packet to arrive at the cluster-head C_m^h of cluster C_m. Thus, in a d level tree, it takes a maximum of 2(d-1) overlay hops to reach the cluster-head C_k^h from

cluster-head C_m^h. Finally, one more overlay hop is needed to reach the destination peer p_d from the cluster-head C_k^h. Therefore, maximum latency is [2(d-1) + 2] = 2d hops. Hence, search latency is not a function of the total number of peers present in the system. It is a function of the number of cluster-heads, i.e. the number of distinct resource types that defines the value of d. □

Table 1 Analytical Comparison of Data Lookup Complexity

	CAN	**Chord**	**Pastry**	**RC-Based Pyramid Tree**
Architecture	DHT-based Structured P2P	DHT-based Structured P2P	DHT – based Structured P2P	Non-DHT - based Structured P2P
Parameters	*N*-number of peers in network *d*-number of dimensions.	*N*-number of peers in network.	N-number of peers in network, *b*-number of bits $(B= 2^b)$ used for the base of the chosen identifier.	*d* - Number of levels of the tree *N* - number of peers in network. n – number of distinct resource types d << N & d< n
Lookup Performance	$O(d\ N^{1/d})$	$O(\log N)$	$O(\log_B N)$	Inter-Cluster: tree traversal $O(d)$ Intra-cluster: $O(1)$

3.4 Experiments

Earlier we have mentioned that the main objective of the present work is to show the superiority of our non-DHT and interest-based architecture over DHT-based architectures from the viewpoints of search latency and data look up complexity. Therefore, in addition to the analytical comparison, we have performed three experiments to compare the data lookup latency in terms of overlay hops of the Pyramid tree p2p architecture with those of the three prominent p2p architectures, viz., CAN [3], Pastry [4] and Chord [5], Results of the experiments with three different numbers of distinct resource types are shown in Figures 3,4, and 5.

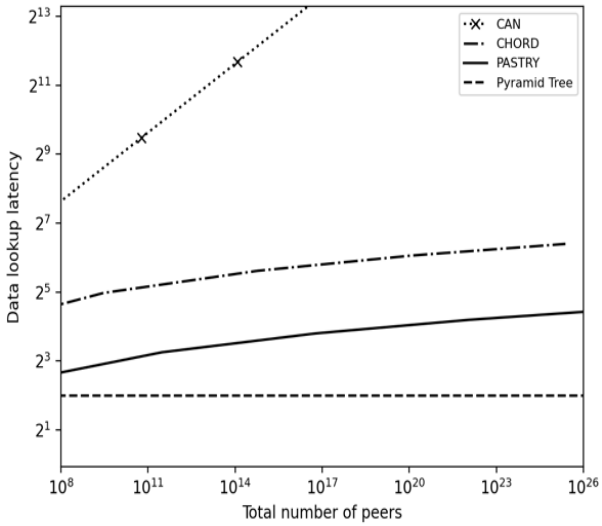

Fig. 3 Lookup latency with 15 resource types

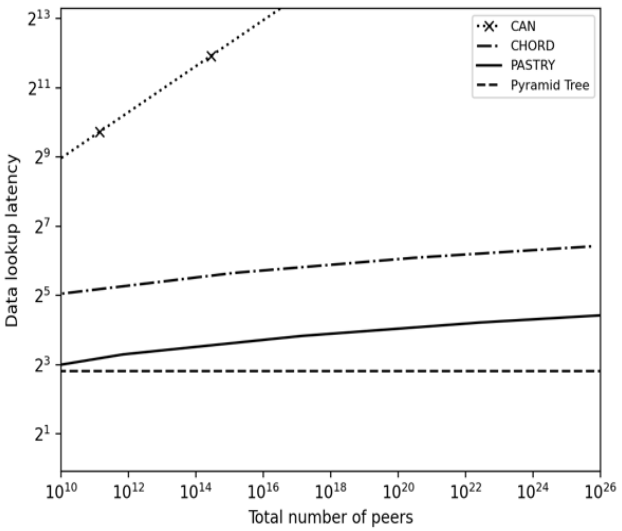

Fig. 4 Lookup latency with 35 resource types

In Fig. 3 we have considered pyramid tree overlay networks with 15 distinct resource types; number of peers in each of the 15 clusters corresponding to the 15 unique resource types has been increased gradually resulting finally in a total of 10^{26} number of peers in the system. Fig. 4 and Fig. 5 correspond to 35 and 65 distinct resource types respectively. Consider N as the total number of peers in the overlay network. Data lookup latency for CAN is known to be $O(dN^{1/d})$; dimension of the co-ordinate space, d is considered as 5 (according to the specification of CAN [3]). For CHORD [5] the data lookup latency is $O(\log_2 N)$, For PASTRY, the data lookup latency is $O(\log_B N)$, where N is the total number of peers in the system and $B = 2^b$. We have selected b = 4 as specified in [4].

It is observed in each figure that with the increase in the number of peers in the system, the data lookup latency each for CAN, CHORD, and PASTRY increases because this latency depends on the total number of peers N in the system. However, in the pyramid tree architecture, the inter-cluster data lookup latency is independent of both individual cluster sizes as well as the total number of peers N in the system. It varies only with the diameter of the tree measured in the number of overlay hops, which depends only on the number of distinct resource types present in the system. Effectively, on an average, the inter-cluster data lookup latency will be half the tree diameter.

Therefore, with the increase in the number of the peers in the system, the pyramid tree architecture will have a constant data lookup latency as is seen in each of the three figures as a straight line with zero gradient. Note that in the figures the respective diameters of the pyramid tree architectures considered are 8, 14, and 20. It may be noted that diameter of each cluster is 1 and so, for an inter cluster communication there will be a maximum of 2 hops more in addition to the 'half of the diameter' (average number of hops travelled from a source cluster-head to a destination cluster-head) provided both sender and receiver are not cluster-heads. Therefore, we have ignored the extra possible 2 hops as it has no impact on our observations of the comparisons of the different latencies.

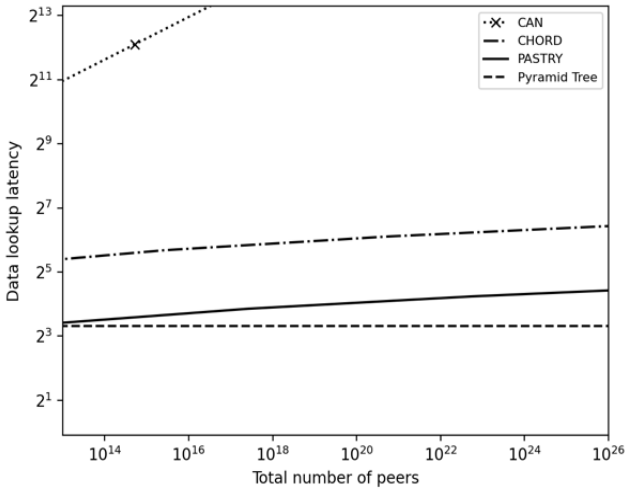

Fig. 5 Lookup latency with 65 resource types

In each of the three figures, we observe significant improvement of inter-cluster data lookup latency which our proposed architecture offers compared to the three other prominent ones especially when the total number of peers N increases beyond 10^8, 10^{10}, and 10^{11} as shown in the respective three figures. In fact, our architecture offers much smaller latency than CAN and Chord even for much lower number of peers than the above-mentioned numbers; differences with Pastry becomes significant when we consider much larger number of peers as mentioned above.

4. Churn Handling

The process of joining and leaving of peers is known as churn. We first consider churn inside a cluster. After that we will consider the pyramid tree.

4.1 Churn inside a cluster

Let us start with the joining of new peers in a cluster, say C_i. Let the new peer be p possessing an instance of the resource type R_i. We have mentioned earlier that each member peer in a cluster maintains locally a list of all its neighbors (i.e. all peers in this cluster) present in the cluster. We denote the list by L_i for the cluster C_i. Each entry in L_i is a tuple consisting of a peer's IP and logical addresses. The

joining peer p is directed by the DNS or the cluster-head C_0^h to contact C_i^h because peer p has the same resource type R_i which is also owned by peers in the cluster C_i. Cluster-head C_i^h broadcasts the IP address and the logical address of peer p to all other peers in the cluster; each receiving peer updates its list. Cluster head C_i^h also updates its list and unicasts the updated list to the joining peer p. Note that cluster C_i is a complete network (a complete graph), therefore, the cluster remains a complete network after the joining of any new peer. Hence, structurally the cluster remains a complete network with overlay diameter 1. Therefore, nothing else need to be done from the viewpoint of maintenance of the structure of the cluster after a join.

Next, we consider that a peer p' leaves cluster C_i. We assume graceful leaving, that is, prior to leave, it broadcasts a leave message in the cluster. After receiving this message each peer in C_i just deletes the address information of peer p' from its list. Note that cluster C_i remains a complete network after the leave. So, nothing else is needed to maintain the structure.

If the cluster-head C_i^h leaves, prior to leaving it takes the following sequence of three actions:

1. It selects the peer with the next higher overlay address as the new cluster head,
2. It assigns the new cluster-head with its own logical address *i*, It is done so in order to keep the cluster-head's overlay address and the resource code of R_i identical,
3. The leaving cluster-head broadcasts a leave message to all peers in the cluster and leaves the network.

The above sequence of actions is followed by the following sequence of three more actions to complete the process of reorganization of the cluster.
4. The new cluster-head broadcasts the tuple < *logical address, IP address of the new cluster-head, i* > to all existing peers in C_i,
5. Each peer in C_i after receiving these two broadcast messages deletes the entry corresponding to the old cluster-head in L_i and updates the information received in the message about the new cluster-head,
6. The new cluster-head unicasts its IP address and logical address to all other cluster-heads. Each receiving cluster-head on the pyramid tree replaces in its *global table, i.e. TOI,* the entry for the old tuple of C_i^h with the new one < *logical address, IP address of the new cluster-head* C_i^h*, resource code i of the resource owned by* C_i^h .

We observe that there are two broadcasts involved (steps 3 and 4) above, one by the leaving cluster-head (step 3), the other one by the new cluster-head (step 4). It can be reduced to only one broadcast in the following way. After the new cluster-head is selected (steps 1 and 2), the old cluster-head leaves (no need for step 3) and the new cluster-head broadcasts a message containing the tuple < *IP address of the new cluster-head, i* > to all existing peers in C_i. Each peer in C_i interprets this message as 'leaving of the old cluster-head' and so it deletes the entry corresponding to the old cluster-head in L_i and updates the information received in the message about the new cluster-head.

From the above discussion we note the following:

1) Let Num_i be the number of peers in cluster C_i prior to the joining of a new peer. Number of messages used for the joining of a peer is ((Num_i -1) + 1), i.e. Num_i and number of overlay hops needed is simply (1+1=2); We use the fact that broadcast in a complete network (as in cluster C_i) is a one-overlay-hop process.
2) Number of messages used to deal with the leaving of a peer is (Num_i – 1) and number of overlay hops needed is just 1.
3) Number of messages used in C_i when cluster-head C_i^h leaves is (Num_i – 2). It is the number of messages broadcasted by the new cluster-head to its peers in cluster C_i (only one broadcast is required). Number of messages to update the rest of the (n-1) cluster-heads is (n-1). So total number of messages used is
[(Num_i – 2) + (n-1)] = [Num_i + n-3]. Number of overlay hops required is (1 + (n-1) = n).

4.2 Churn in the pyramid tree at layer-1

Consider first joining of a new cluster head. We assume that there are already n distinct resource types present; so, the existing cluster-heads are C_0^h, C_1^h, …, C_{n-1}^h. Hence the newly joining cluster-head will be C_n^h. It means that a new peer possessing an instance of a new resource type R_n wants to join. This joining peer contacts first the root cluster head C_0^h of the tree. The root assigns a logical address n to this peer and this peer becomes the new cluster head C_n^h. The root will update its *TOI* with the IP and logical addresses of the new cluster head; it will then unicast this table to all cluster heads including the new one: thereby, requiring a total of n messages.

We now consider the case when an existing cluster head other than the root cluster-head leaves while this cluster does not have any other peer except the leaving cluster head itself. We assume graceful leaving. Therefore, right before leaving the

network, the leaving cluster head will send a 'leave' message to the root C_0^h. The root will update its table (*TOI*) by deleting the entry corresponding to the leaving cluster head. It will assign new (secondary) logical addresses to some existing cluster heads as needed. For example, if the leaving one has logical address (primary) X, the root will assign the existing cluster head with primary address (X+1) an additional logical address X (secondary); similarly, the one with existing primary address (X+2) will have (X+1) as its secondary address, etc. However, addresses of cluster heads with addresses less than X will not have any secondary addresses. Therefore, only those cluster-heads with primary addresses larger than X will have both primary and secondary addresses. Also note that in the updated *TOI* primary address X does not exist. We observe that primary addresses till (X-1) along with secondary addresses starting from X maintain the sequencing of addresses (i.e. (X-1) is followed by X etc.), which is required for the broadcast protocol to work correctly, Effectively, secondary logical addresses as well as those primary addresses not having any corresponding secondary addresses are used by the broadcast protocol for forwarding a broadcast packet. However, only primary addresses will be used to look for resources, since they represent the unique codes for the resources. The root will unicast this updated *TOI* to all other cluster heads, i.e. (n-2) cluster-heads, thereby requiring (n-2) massages. Therefore, a total of ((n-2) + 1), i.e. (n-1) messages are needed to handle the leaving process. As stated above, note that in the updated table primary address X does not exist, i.e. there does not exist any cluster possessing the resource type X; therefore, any lookup request by a peer in a cluster C_i for resource X will eventually fail as the corresponding cluster head C_i^h will not find it appearing as a primary address in its *TOI*. However, note that any look up for a request with resource code (X+1) the cluster-head C_i^h will unicast to the IP address corresponding to the cluster with primary address as (X+1) and secondary address as X.

If the root node leaves, it unicasts a leave message to the node with primary address 1 and the later becomes the new root and its new address (secondary) becomes 0. This new root now deletes the entry for the leaving node from its *TOI*. This new root keeps the existing primary addresses of all other cluster-heads as well as their new secondary logical addresses in its *TOI*; i.e. a cluster-head with existing primary address X will have a secondary logical address as (X-1) as well in the *TOI*. The new root unicasts to the (n-2) cluster-heads the updated *TOI*. It thus requires a total of ((n-2) + 1), i.e. (n-1) messages.

If two or more nodes want to leave simultaneously, they leave in the sequence of arrivals of their 'leave request' messages to the root node. Observe that whatever is required to maintain the structure after any join/leave is done centrally by the current root cluster-head just with the help of few unicasts of the updated *TOI*, approximately equal to the number of distinct resources. It makes the process of churn handling quite simple and efficient.

4.3. Partitioning of a Cluster into Sub-Clusters

Broadcasting by a cluster-head C_i^h is a key factor related to both data lookup algorithms and churn handling in our proposed overlay network. In order to reduce the load (in terms of the number of messages to broadcast in cluster C_i) by the cluster-head C_i^h, we propose the following idea of dividing a cluster, say C_i in several small sub-clusters.

Let us consider that a cluster, say C_i be divided into sub-clusters of identical size each, say Z. Let there be k number of sub-clusters. So, Num_i is equal to (k.Z). Of course, it is most likely that the last sub-cluster will have less than Z number of peers; however, it has no effect on our explanation of the idea used for partitioning. Let the k sub-clusters be denoted as C_{i1}, C_{i2}, …, C_{ik}. We assume that peers in these sub-clusters are in increasing order of their logical addresses. Therefore, the last peer to join will have the largest logical address among all in the parent cluster C_i. It means that any new join always occurs in the last sub-cluster. If the last one C_{ik} already has Z peers, the new peer will form a new sub-cluster $C_{i(k+1)}$ and it becomes its own sub-cluster head $C_{i(k+1)}^h$ as well; otherwise the new one will join as a member of the sub-cluster C_{ik}.. Note that cluster-head C_i^h is also the sub-cluster head C_{i1}^h of the first sub-cluster C_{i1}.

An example: We illustrate the partitioning process using the following example. let us consider a cluster C_i with cluster-head C_i^h. Let C_i consist of 40 peers. Assume that C_i is divided into four sub-clusters C_{i1}, C_{i2}, C_{i3}, and C_{i4} with 10 peers each. It is shown in Fig. 6. Observe that in general the last sub-cluster not necessarily will have the same size as others; it depends on the number of peers initially present in the parent cluster C_i and the number of sub-clusters to be formed; however, as mentioned earlier that it has no effect on our explanation of the idea.

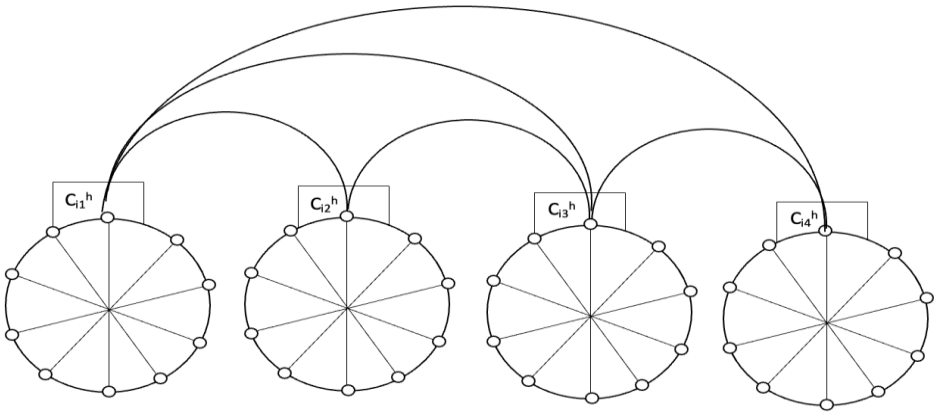

Fig. 6 Partitioning of cluster C_i into four sub-clusters C_{i1}, C_{i2}, C_{i3}, and C_{i4}

Let the logical address of the cluster-head C_i^h be i based on the mod value of n used in designing the RC-based overlay architecture. Then, the other 39 peers in C_i will have the respective overlay addresses as (i+n), (i+2n), …, (i+39n) based on their sequence of joining the cluster. Addresses of the four sub-cluster heads are i, (i+10n), (i+20n), and (i+30n) respectively. Since C_i is completely connected, therefore, any two peers in the cluster have direct logical connections. However, in this partitioning method, we assume the presence of the logical links that connect the neighboring sub-cluster heads as well as the links that connect the first sub-cluster head, C_i^h (i.e. C_{i1}^h) and the other three cluster-heads.

Cluster-head C_i^h assigns the addresses (i+n) to (i+9n) to the first nine peers joining the cluster besides the cluster-head itself. It forms the sub-cluster C_{i1} with itself as the its sub-cluster head. Cluster-head C_i^h assigns the next 10 arriving peers with addresses (i+10n) to (i+19n) and imparts the responsibility of becoming the sub-cluster head of the sub-cluster C_{i2} to the peer with address (i+10n). In this way, the other sub-clusters are also formed. As pointed out earlier, in general the last sub-cluster may have a smaller number of peers than the other clusters at the time of formation. It depends on the present number of peers in a given cluster and the number of sub-clusters. For example, if cluster C_i initially has 39 peers instead of 40, the last sub-cluster C_{i4} will have 9 peers, one less than the number of peers in the other sub-clusters of C_i.

Note that sub-cluster size can be a choice of the designers and it can be dynamically changed based on the total current number of peers present in the cluster C_i.

4.3.1 Reduction of the number of messages broadcasted by C_i^h due to partitioning

Cluster-head C_i^h keeps address information of the peers in its own sub-cluster C_{i1} and the address information of all the other sub-cluster heads. It also knows the largest logical address used so far in this cluster C_i. Any other peer in this sub-cluster (C_{i1}) will keep the address information of all peers in this sub-cluster only. Therefore, when needed, the cluster-head C_i^h will broadcast only to peers in its own sub-cluster and to the other sub-cluster heads (in fact, only to the next sub-cluster head is needed).

A peer in any other sub-cluster C_{ij} will just keep the address information of only the peers in this sub-cluster; the corresponding sub-cluster-head C_{ij}^h will keep the same information; in addition it will keep the address information of the cluster-head C_i^h and those of its two neighboring sub-cluster heads $C_{i(j-1)}^h$ and $C_{i(j+1)}^h$. The sub-cluster head C_{ij}^h will be responsible for broadcasting only inside this sub-cluster. Therefore, whenever needed the load on C_i^h caused by broadcasting can be distributed among the sub-cluster heads. Below we assume that each sub-cluster consists of Z peers.

Lemma 3. Number of hops required to broadcast in cluster C_i is k.

Proof. Broadcasting uses the idea of pipe lining in the following sense.

Broadcasting in cluster C_i takes place as follows. Cluster head C_i^h starts broadcasting in the first sub-cluster C_{i1} followed by unicasting of the message to its neighbor sub-cluster head C_{i2}^h. Broadcasting in C_{i1} takes only 1 hop as C_{i1} is a complete network. Also, the unicast transmission takes only 1 hop.

Therefore, broadcasting is completed in C_{i1} when the message arrives at C_{i2}^h. In this way, at the time the broadcast message arrives at $C_{i(j+1)}^h$ from C_{ij}^h, broadcasting is already completed in the later sub-cluster (hence the pipe lining idea). Thus, after the last sub-cluster head C_{ik}^h receives the message from sub-cluster head $C_{i(k-1)}^h$, broadcasting takes place in the last sub-cluster, which takes 1 hop and there are (k-1) hops needed for the message to travel from the first sub-cluster head to the last one; hence the total number of hops is (1 + (k-1)), i.e. k. □

Observation 3. *Size of the list L_{ij} maintained by each peer in any cluster C_{ij} reduces approximately from (k.Z) to Z.*

Observation 4. *Number of messages to be broadcasted by cluster-head C_i^h of cluster C_i is ((Z-1) +1), i.e. Z; The 1st term is due to broadcast inside the first sub-cluster C_{i1} and the 2nd term for communication from C_i^h to $C_{(i+1)}^h$.*

Observation 5. *Number of messages to be broadcasted by any sub cluster-head C_{ij}^h other than the first and the last ones is ((Z-1) +1), i.e Z; The 1st term is due to broadcast inside the sub-cluster C_{ij} and the 2nd term for communication from C_{ij}^h to $C_{i(j+1)}^h$.*
Observe that for the last sub-cluster head it will be at most (Z-1).

4.3.2 Churn handling in a partitioned cluster

In dealing with churn, we mainly focus on the number of messages required to handle different situations, e.g. peer joining, peer leaving etc. because it is directly related to bandwidth utilization.

Let us consider the k sub-clusters C_{i1}, C_{i2}, …, C_{ik}. Peers in these sub-clusters are in their increasing order of their logical addresses. Therefore, the last peer to join will have the largest logical address among all in the parent cluster C_i. It means that any new join always occurs in the last sub-cluster.

4.3.2.1 Peer joining

The joining peer p is directed by the DNS or the cluster-head C_0^h to contact C_i^h because peer p has the same resource type R_i which is also owned by peers in cluster C_i. Cluster-head C_i^h assigns the largest logical address to the peer and gives the IP address of the current last sub-cluster head C_{ik}^h to the joining peer p and the joining peer p contacts the last sub-cluster head.

If the last one C_{ik} already has Z peers, the new peer will form a new sub-cluster $C_{i(k+1)}$ and it becomes its own sub-cluster head $C_{i(k+1)}^h$; otherwise the new one will join the sub-cluster C_{ik}.. In the second case, sub-cluster head C_{ik}^h broadcasts the IP address and the logical address of peer p to all other peers in its sub-cluster; number of such messages is (Z-1) and the broadcast will take 1 hop. Each receiving peer in sub-cluster C_{ik} updates its list L_{ik}. Cluster head C_{ik}^h also updates its list and unicasts the updated list to the joining peer p. Therefore, total number of messages needed to handle the join process is (Z-1+1), i.e. Z and the required number of hops is only 2. It may be noted that if the joining peer becomes a new sub-cluster head $C_{i(k+1)}^h$, it will unicast this information to the first sub-cluster head C_{i1}^h (i.e. C_i^h) and the last sub-cluster head C_{ik}, requiring only two message transmissions.

Note that initially there are couple other messages used for contacting the last sub-cluster head and it is the same for any join. So, we have ignored it. We basically focus on two parameters, viz., number of partitions and size of a cluster for determining the number of messages because values of these two related parameters can be of interest to the network designers from the viewpoint of reasonably efficient bandwidth utilization.

Observation 6. *Number of messages to handle a joining process is at most Z, where as it is (k.Z) if partitioning is not used.*

4.3.2.2 Peer leaving

When a non-cluster head peer in a cluster C_{ij} leaves, it only broadcasts (Z-1)) leave messages in its sub-cluster. So, the number of such leave messages reduces to (Z-1) from (k.Z).

If sub-cluster head C_{ij}^h leaves, the number of messages broadcasted by the *new* cluster-head to its peers in cluster C_{ij} that has now (Z-1) peers is (Z-2) (only one broadcast is required). Number of messages required to update the two neighboring sub-cluster heads and the first one about this new sub-cluster head is 3. If this sub-cluster is eventually the last one, number of messages is just 2, one to the first sub-cluster head and the other to the previous sub-cluster head. So total number of messages used is at most (Z+1) compared to (K.Z-1) if partitioning is not used and when sub-cluster head is not the cluster head.

If the cluster-head C_i^h leaves, the number of messages broadcasted by the *new* cluster-head to peers in the first sub-cluster C_{i1} is (Z-2) (only one broadcast is required); number of messages required to update the (k-1) other sub-cluster heads in C_i about this new sub-cluster head is (k-1); and number of messages required to update the rest of the (n-1) cluster-heads at level 1 is (n-1). Hence, total number

of messages used is ((Z-2) + (k-1) + (n-1)), i.e. (Z + k + n - 4).

Observation 7. *Without partitioning, when a cluster-head C_i^h leaves, total number of messages required to handle a leaving process is at least k.Z compared to a maximum of (Z + k + n - 4) when partitioning is used.*

To achieve substantial reduction in the number of messages to handle churn caused by the above leaving process, a designer's objective should be to consider those values of Z and k for a given Num_i that will maximize [k.Z – (Z + k + n – 4)]. Since the number of clusters, i.e. the number of distinct resources denoted as n is not a designer's choice, therefore, the revised objective should be to maximize [k.Z – (Z + k)]. Also, care should be taken to select the value of Z because Z is also the number of messages required to handle a joining process.

4.3.3 Churn handling in Pyramid tree at layer-1

There is no effect on churn handling at layer-1 due to partitioning of clusters; it is the same as appeared in Section 4.2.

4.3.4 Churn Handling under Capacity Constrained Situation

Restricted capacity of a peer does not change the number of messages required to handle churn in the network. However, it may take more time (i.e. larger number of hops) to reorganize the network after joining and leaving of peers. For example, even if a cluster is divided logically into sub-clusters of diameters 1 each, the number of hops required to broadcast in a sub-cluster is $\log_c Z$ [18] instead of 1 hop, given that average capacity of a peer is c. It delays reorganization.

Experiments

In the following experiments, we have compared performance of the proposed partitioned architecture with the performance of Chord, arguably the most noted DHT-based structured overlay architecture. Performance has been measured using the number of messages required to maintain the overlay architecture after a peer joins or leaves the system. Results of our observations in the four experiments have appeared in Figs. 7, 8, 9, and 10 with respective cluster sizes as 500, 1000, 2000, 3000 peers. In the experiments cluster sizes have been kept constant; the number of distinct resources n in the system has been varied from 20 to a maximum of 40; it also means that number of clusters of a given size varies from 20 to a maximum of 40.

We have considered partitioning of a cluster C_i into k sub-clusters each with Z number of peers to achieve reduction in the number of messages to handle churn when a cluster-head C_i^h or any other peer in the cluster leaves. As mentioned earlier that this reduction due to a cluster-head leaving can be maximum when for a cluster-size of k.Z, values of Z and k are so selected that maximizes [k.Z – (Z + k + n – 4)] for a given n. For example, in Fig. 7, we found the respective values of Z and k as 25 and 20 for the cluster size of 500. In the other three figures we have mentioned values of Z and k corresponding to the respective cluster sizes used in the three other experiments.

In addition, in Fig. 7, when Num_i is 10^2, the corresponding number of clusters is 20 which is written right below 10^4 in the figure. This number 20 also represents the number of distinct resource types n. Similarly, when Num_i is $2x10^4$, number of corresponding clusters is 40, which is also the number of resource types. It means that for a given cluster size, say 500 peers, we increase the total number of peers in the system by increasing the number resource types n. This is true for the other three experiments as well.

Note that the graphs showing the effects of new cluster-head join / existing cluster-head leaves with no more peer left in the cluster, are overlapped; for example, it is the second one from the bottom in Fig. 7. Since the number of messages required to handle such a join / leave is directly proportional to n, hence the variable used here along the horizontal axis is n which is specified as (n) below each Num_i. Similarly, the variable n is used to draw the graph when a cluster-head leaves and the cluster still exists; it is the second one from the top in Fig. 7.

Besides, the graphs for a peer joining or leaving a sub-cluster are overlapped, for example in Fig. 7 it is the bottom most graph. The reason is that for such a joining the number of messages is at most Z and for leaving it is either at most (Z+1) or (Z-1) depending on if the leaving peer is a sub-cluster-head or not. So, we have considered just the value Z for both joining and leaving.

Each diagram shows reasonable improvement in terms of requiring a smaller number of messages to handle different kinds of joins and leaves of our design with partitioned clusters when compared to Chord.

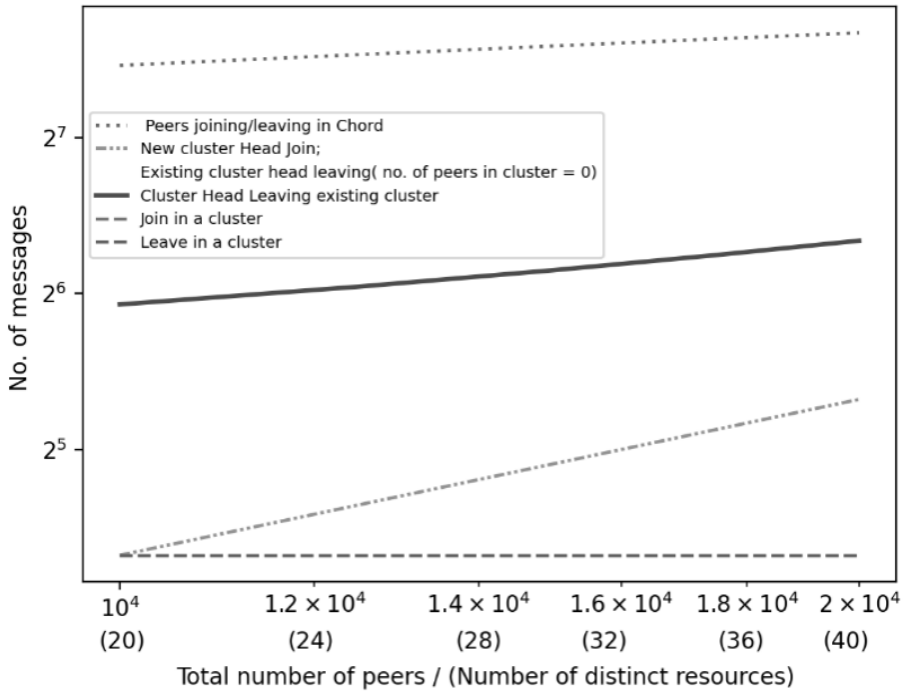

Fig. 7 Cluster size (k.z) = 500, k =25, z = 20

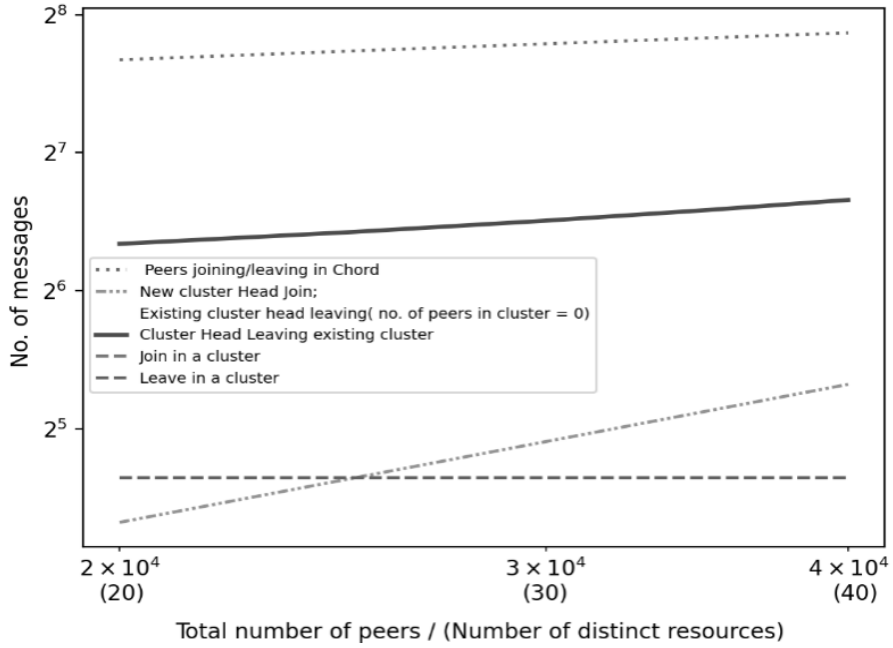

Fig. 8 Cluster size (k.z) = 1000, k = 40, z = 25

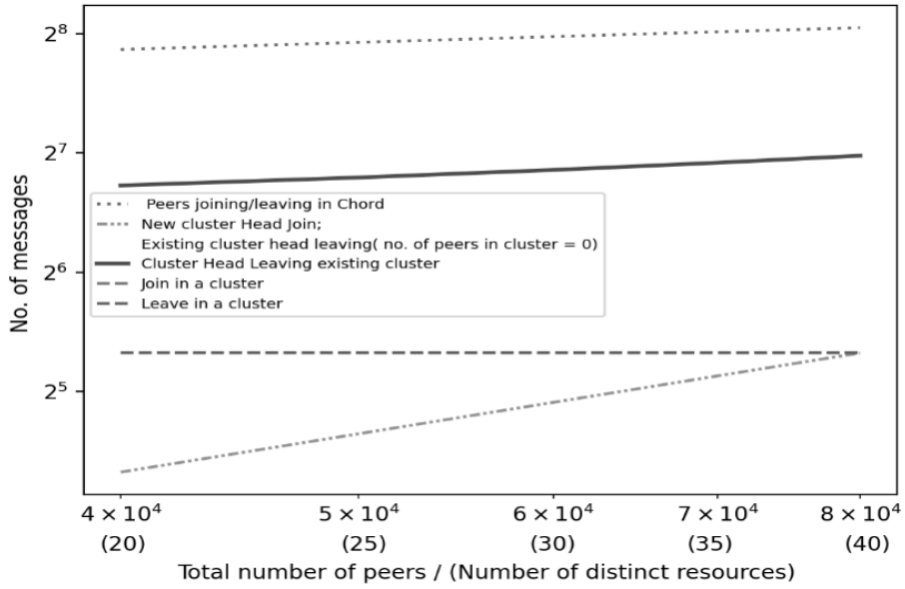

Fig. 9 Cluster size (k.z) = 2000, k = 50, z = 40

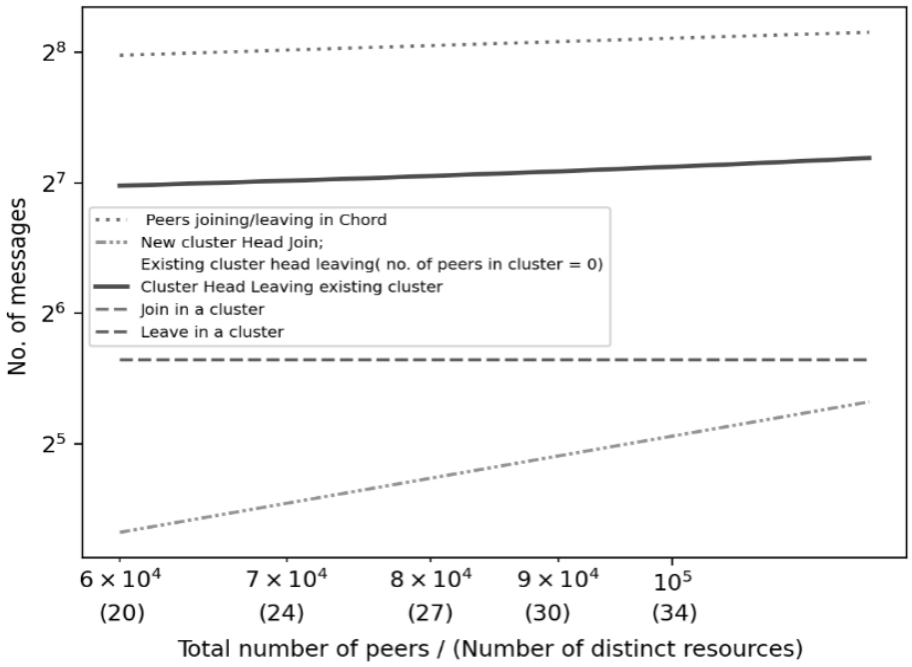

Fig. 10 Cluster size (k.z) = 3000, k =60, z = 50

5. Discussion on some Existing Interest Based P2P Architectures

In this section, we have considered several noteworthy interest-based P2P systems [19]-[25] and briefly state their main features from the viewpoint of their proposed architecture. Then we state the same of our work and justify why our design is superior to these works.

All these works have incorporated the idea of peer heterogeneity in their design. In doing so the work in [19] has used the existing idea of *super peer.* Besides, *gossiping* has been used for cluster formation with peers of common interest.

The work in [20] uses the idea of *popular peer* which is quite similar to the idea of *super peer*. The base architecture is an unstructured network.

In [21], authors have considered *super peer*. It is a hybrid architecture that uses both chord and unstructured network.

In [22], *gossiping* has been used for cluster formation. Besides, at the time of joining a new peer searches from a list of known peers for a particular peer which has most links among all in the list and then gets connected to it.

In [23], authors have considered DHT-based structured P2P system considering both proximity - aware and interest-based cluster formation. Aim is to improve file location efficiency considering physically close peers with common interest. The work starts with the formation of a cluster consisting of physically close peers; it then forms sub-clusters with peers having common interest. Reasonable improvement of file location efficiency has been achieved. However, they have not considered the following highly probable situation: assume that there are q number of such clusters scattered around; each of them has a sub-cluster formed with physically close peers with the same interest, say i. Then what will be the inter-sub-cluster lookup efficiency? This problem has not been addressed in this research. Therefore, the work remains incomplete, even though the basic idea is good.

In [24], a pastry-based P2P e-commerce model based on interest community has been proposed. Users with similar interest form an interest community and the users in such a community are not necessarily physically close. Authors have assumed that all such users are directly connected, i.e. such a cluster has the overlay diameter of one hop. However, there is no mathematical basis for such an assumption.

In [25], Authors have considered the adaptation of P2P architecture to support social network characteristics. This P2P architecture is based on the idea of Chord. Peers with similar interest are linked and these links are created dynamically based on previous communication messages among the peers. These links are called interest links. Authors have proposed an efficient routing algorithm based on such links.

In our proposed work, we do not use *gossiping* for cluster formation; we do not consider either *super peer* or *popular peer* or even a joining peer does not look for the best peer to be connected to as in [22]. We now justify why it is so in our work. First the existing idea of *gossiping* is not at all an efficient way to form clusters of peers of common interests. Second, when new peers join, there is always some probability that the new one will be better than an existing *super peer* or *a popular peer*. So, again a new one may have to be selected. It wastes time, particularly when several peers join at the same time or peers join frequently. The joining of a new peer [22] incurs unnecessary waste of time to search for the best peer to connect to. In addition, some of the above works have considered unstructured network. So, it may involve the typical searching problem inherent to unstructured networks. Work in [23] is quite important because it has considered both physical closeness of peers and peers with common interest. However, inter-cluster communication among clusters of peers with similar interest has not been addressed. Works in [23]-25] are all DHT based approaches. So, they cannot avoid the problems of maintaining the architecture in presence of churn.

We have used modular arithmetic (residue class) to build the clusters with peers of common interests. It is a non-DHT based approach. This mathematical tool helps in assigning overlay addresses to peers in a way that any cluster with peers of common interest becomes a complete network (i.e. from graph theoretic viewpoint, a complete graph). Therefore, the overlay diameter of a cluster is just 1 and hence search latency for intra-cluster communication is $O(1)$. Diameter of the whole architecture is just (2d + 2) where d is the number of levels of the pyramid tree. Therefore, search latency for inter-cluster communication is $O(\mathrm{d})$. Note that nodes on the tree are only the cluster-heads which represent the different distinct resources and value of d only depends on the number of such nodes; therefore, d is negligibly small compared to the total number of peers in the system; note that a cluster may contain any number of peers and we do not put any restriction on the size of a cluster.

Finally, during design phase we have not specifically focused on peer heterogeneity; it has made our design process very simple. So, there is no need to select any super or a popular peer. We do not need to consider any capacity constrained communication between cluster-heads, because on the pyramid tree no cluster-head (a node on the tree) can have more than four overlay links and in fact, we use a maximum of three links of a cluster-head in the proposed broadcast protocol and realistically any node usually has these many number of links or more. We have earlier mentioned that work is under progress for designing capacity constrained communication protocols (broadcast and multicast) inside a cluster; fact is, our objective is to incorporate peer heterogeneity only during broadcast and multicast inside a cluster.

Finally, our process of churn handling is one of the simplest known ones. Since a cluster is a complete network, any new join or a leave does not

change its diameter; thereby restructuring of a cluster is very simple. Besides, location of a new cluster-head with a new resource type is always at the leaf level. Realistically, we believe that our work is a challenge to any existing DHT based architecture from the viewpoints of overlay search latency and churn handling. Churn handling has been discussed in detail in the next section.

6. Conclusion

In this paper, we have considered a 2-layer non-DHT and interest based structured P2P architecture, also known as pyramid tree architecture. Residue Class based on modular arithmetic has been used to realize the overlay topology. Use of such mathematical tool has helped in obtaining some very important structural properties of the network. Our main objective has been to show the superiority of our non-DHT and interest-based architecture over DHT-based architectures from the viewpoints of search latency and data look up complexity. In designing the data lookup protocols, we have used some of the structural properties of our architecture, e.g. each cluster has a diameter of only 1 overlay hop and the diameter of the network is just (2+2d); d is the number of levels of the layer-1 pyramid tree and d depends only on the number of distinct resources. Therefore, the diameter of the network is independent of the number of peers in the whole network. As a result, the proposed intra-cluster data lookup protocol has constant complexity and complexity of inter-cluster data lookup is $O(d)$ if tree traversal is used; note that $n \approx 2^d$ and hence d is even much smaller than n, the number of distinct resource types. The noteworthy point is that search latency is independent of the total number of peers present in the overlay network unlike any structured DHT-based network (as a matter fact unlike any P2P network, structured or unstructured). We have presented both analytical and experimental results comparing the proposed ones with some of the most noted DHT-based structured overlay networks.

We have presented in detail the process of handling churns and proposed a simple yet very effective technique related to cluster partitioning, which, in turn, helps in reducing the number of messages required to be exchanged to handle churns. Most of the existing interest-based architectures are built on an ad-hoc basis without having any solid mathematical foundation. Hence, we have highlighted the main differences of our interest-based architecture with some such existing ones using architectural aspects only and the related pros and cons of these architectures.

The present work is part of an ongoing research project; currently we are working on designing P2P Federation using our model architecture as the basic architectural component of the Federation.

References

[1] P. Ganesan, Q.Sun, and H. Garcia-Molina, "Yappers: A peer-to-peer lookup service over arbitrary topology," Proc. IEEE Infocom 2003, San Francisco, USA, pp. 1250-1260, Vol. 2,March 30 - April 1 2003.

[2] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," Proc. ACM SIGCOMM, Karlsruhe, Germany, August 25-29 2003.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network, CAN," Vol. 31, No. 4, pp. 161-172, ACM, 2001.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large Scale Peer-to-Peer Systems", Proc. FIP/ACM Intl. Conf. Distributed Systems Platforms (Middleware), pp. 329-350, 2001.

[5] I. Stocia, R. Morris, D. Liben-Nowell, D. R. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications", IEEE/ACM Tran. Networking, vol. 11, No. 1, pp. 17-32, Feb. 2003.

[6] M. Yang and Y. Yang, "An Efficient Hybrid Peer-to-Peer System for Distributed Data Sharing", IEEE Trans. Computers, vol. 59, no. 9, pp. 1158-1171, Sep. 2010.

[7] M. Xu, S. Zhou, and J. Guan, "A New and Effective Hierarchical Overlay Structure for Peer-to-Peer Networks", Computer Communications, vol. 34, pp. 862-874, 2011.

[8] D. Korzun and A. Gurtov," Hierarchical Architectures in Structured Peer-to-Peer Overlay Networks", Peer-to-Peer Networking and Applications, Springer, pp. 1-37, March 2013

[9] Z. Peng, Z. Duan, J.Jun Qi, Y. Cao, and E. Lv, "HP2P: a hybrid hierarchical p2p network," Proc. Intl. Conf. Digital Society, pp. 86-90, 2007.

[10] K. Shuang, P Zhang, and S. Su, "Comb:resilient and efficient two-hop lookup service for distributed communication system," Security and Communication Networks, vol. 8(10), pp. 1890 1903, 2015.

[11] M. Kleis, E. K. Lua,, and X. Zhou, "Hierarchical Peer-to-Peer Networks using LightweightSuperPeer Topologies," Proc. IEEE Symp. Computers and Communications, pp. 1-6, 2005.

[12] Bidyut Gupta, Nick Rahimi, Shahram Rahimi, and Ashraf Alyanbaawi, "Efficient Data Lookup in Non-DHT Based Low Diameter Structured P2P Network," Proc. IEEE 15th Int. Conf. Industrial Informatics (IEEE INDIN), pp. 944-950, July 2017, Emden, Germany.

[13] Bidyut Gupta and Mohammad Mohsin, "Fault-Tolerance in Pyramid Tree Network Architecture," J. Computer Systems Science and Engineering, Vol. 10, No. 3, pp. 164-172, July 1995

[14] N. Rahimi, K. Sinha, B. Gupta, and S. Rahimi, LDEPTH: A Low Diameter Hierarchical P2P Network Architecture, Proc. IEEE 14th Int. Conf. on Industrial Informatics (IEEE INDIN), Poitiers, France, pp. 832-837, July 2016.

[15] Indranil Roy, Koushik Maddali, Swathi Kaluvakuri, Banafsheh Rekabdar, Ziping Liu, Bidyut Gupta, Narayan Debnath, Efficient Any Source Overlay Multicast In CRT-Based P2P Networks ─ A Capacity - Constrained Approach, Proc. IEEE 17th Int. Conf. Industrial Informatics (IEEE INDIN), July 2019, Helsinki, 1351-1357, Finland.

[16] Indranil Roy, Bidyut Gupta, Banafsheh Rekabdar, and Henry Hexmoor, A Novel Approach Toward Designing A Non-DHT Based Structured P2P Network Architecture, EPiC Series in Computing, Volume 63, 2019, pages 121-129, (Proceedings of 32nd Int. Conf. Computer Applications in Industry and Engineering).

[17] Indranil Roy, Nick Rahimi, Koushik Maddali, Swathi Kaluvakuri, Bidyut Gupta and Narayan Debnath, Design of Efficient Broadcast Protocol for Pyramid Tree-based P2P Network Architecture, EPiC Series in Computing, Volume 63, 2020, pages 182-188, (Proceedings of 33rd Int. Conf. Computer Applications in Industry and Engineering, San Diego).

[18] Shiping Chen, Baile Shi, Shigang Chen, and Ye Xia, ACOM: Any-Source Capacity-Constrained Overlay Multicast in Non-DHT P2P networks, IEEE Tran. Parallel and Distributed Systems, vol. 18, no. 9, pp. 1188-1201, Sep. 2007.

[19] S. K. A. Khan and L. N. Tokarchuk, "Interest-Based Self Organization in Cluster-Structured P2P Networks," 2009 6th IEEE Consumer Communications and Networking Conference, Las Vegas, NV, 2009, pp. 1-5, doi: 10.1109/CCNC.2009.4784959.

[20] Wen-Tsuen Chen, Chi-Hong Chao and Jeng-Long Chiang, "An Interested-based Architecture for Peer-to-Peer Network Systems," 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06), Vienna, 2006, pp. 707-712, doi: 10.1109/AINA.2006.93.

[21] Z. Tu, W. Jiang and J. Jia, "Hierarchical Hybrid DVE-P2P Networking Based on Interests Clustering," 2017 International Conference on Virtual Reality and Visualization (ICVRV), Zhengzhou, China, 2017, pp. 378-381, doi: 10.1109/ICVRV.2017.00087.

[22] Khambatti, Mujtaba & Ryu, Kyung & Dasgupta, Partha. (2003). Structuring Peer-to-Peer Networks Using Interest-Based Communities. Lecture Notes in Computer Science, 1st International Workshop, DBISP2P 2003, Berlin, September 2003.

[23] H. Shen, G. Liu and L. Ward, "A Proximity-Aware Interest-Clustered P2P File Sharing System," in IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 6, pp. 1509-1523, 1 June 2015, doi: 10.1109/TPDS.2014.2327033.

[24] M. Hai and Y. Tu, "A P2P E-Commerce Model Based on Interest Community," 2010 International Conference on Management of e-Commerce and e-Government, Chengdu, 2010, pp. 362-365, doi: 10.1109/ICMeCG.2010.80.

[25] L. Badis, M. Amad, D. Aîssani, K. Bedjguelal and A. Benkerrou, "ROUTIL: P2P routing protocol based on interest links," 2016 International Conference on Advanced Aspects of Software Engineering (ICAASE), Constantine, 2016, pp. 1-5, doi: 10.1109/ICAASE.2016.7843852.

[26] Swathi Kaluvakuri, Koushik Maddali, Nick Rahimi, Bidyut Gupta, and Narayan Debnath, "Generalization of RC-Based Low Diameter Hierarchical Structured P2P Network Architecture," IJCA, Vol. 27, No. 2, June 2020, pp. 74-83.

[27] Koushik Maddali, Indranil Roy, Swathi Kaluvakuri, Ziping Liu, and Bidyut Gupta, "Design of Novel Low Latency and High Bandwidth-Efficient Broadcast Protocols for Non DHT-based Pyramid Tree P2P Architecture," submitted to International Journal of Computers and Their Applications (IJCA).

[28] Jie Cheng and Ryder Donahue, "The Pirate Bay Torrent Analysis and Visualization," IJCSET, Vol. 3, Issue 2, pp. 38-42, Feb. 2013

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)