Volume 6, Issue 5, May 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Survey and Comparison of Structured Peer to Peer Overlay Network Routing Algorithms for Mobile Grid

H. Parveen Begam
Prof. Department of Computer Science,
MAM College of Engineering,
Trichy, Tamil Nadu, India

G. Anusha Bhuvaneswari
M.E. Department of Computer Science,
MAM College of Engineering,
Trichy, Tamil Nadu, India

Abstract— Mobile grid is a class of distributed systems that autonomously engage in sharing of resources and exchange of services to provide higher performance and utilization of the mobile device resources in a seamless, transparent, secure and efficient way. It allows sharing and co-ordinate use of diverse resources in dynamic, heterogeneous and distributed environment. Distributed Hash Table (DHT) plays an important role in distributed systems and applications, especially in large-scale distributed environments like mobile Grid Environment.DHT is a simple and elegant design for distributed systems. It provides the functions like a hash table to deal with the distributed data. DHT does not require a central server and treats all DHT nodes in the distributed system equally. Normally the routing topologies for structured peer-to-peer overly networks are classified based on their routing process and they are torus for CAN, ring topology for Chord, PRR Tree is for Tapestry and Pastry. In this paper we review the DHT under a number of key aspects that concern performance, scalability and self-configurability. Mainly we focused on the different structured P2P Overlay DHT routing algorithms and their various platforms and way it works for Mobile Grid applications.

Keywords— Mobile Grid Computing, P2P Overlay Networks, Distributed Hash Table, Structured DHT, Chord DHT.

I. INTRODUCTION

A. Mobile Grid

Grid computing can be defined as a set of parallel and distributed systems that enable the sharing of geographically distributed, independent and heterogeneous resources like computers, software, etc., for effectively solving computationally intensive problems. Grid computing paradigm has become one of the most important techniques in the area of high performance computing. As the number of prospective users increases rapidly, the available resources can be supplemented with resources available from the mobile devices.

Mobile computing is another computing paradigm of distributed systems, considering mobility, portability and wireless communications. The main motivation of combining the mobile and grid computing is to carry out the user's work while on the move. Due to various constraints with the wireless network, the environment and the user may rapidly change their environment from stationary to mobile and is location dependent.

As the mobile devices in use are huge in number, even a small percentage of their resource utilization in a grid environment shall be highly advantageous and useful. Mobile grid computing broadens the scope of grid computing by including a vast resource pool available in the form of mobile devices. As the number of mobile devices in use is bound to increase in coming years, and with enhanced features, mobile grid computing will offer in the area of high performance computing. Mobile Grid is a branch of Grid where the infrastructure includes mobile devices.

The Mobility of devices is not supported in most of the grids that are in operation today. This problem might seem trivial, but with millions of mobile devices running today and their processing power remaining unutilized to the maximum, it seems a good idea to include these in the existing family of grids. Apart from utilizing resources on the mobile devices, the mobile grid can provide mobile devices with an opportunity to use the resources on the grid, thereby saving their own resources considerably and overcoming the physical shortcoming of the device. Mobile devices having access to the grid as users would thus be able to perform certain tasks on the run which otherwise would have been done only when the user could access a wired device. Certain situations could allow the mobile devices to contribute to the grid resource warehouse.

The main factors that hinder the growth of this paradigm as compared to the grid computing paradigm are the issues related to mobility and the constraints of mobile computing are that mobile devices are poor in available resources as compared to wired systems; mobile devices are more prone to security breaches; mobile connectivity is highly variable in performance and reliability; mobile devices rely on a finite energy source.

The following are the devices predominant in the grid and relative mobility of the service in the grid into four types:

- Fixed wireless grids
- Mobile or dynamic wireless grids
- Sensor Network Grids
- Ad hoc grids

Mobile grid takes on arbitrary and unpredictable topology because of the mobility of nodes, leading to complicated searching and locating of resource; Routing way in mobile environment is different from that of traditional grid.

B. Consistent Hashing

Consistent hashing is an early version of DHT and it was proposed in 1997 for distributed caching systems. In distributed caching systems, to map the content objects to caching servers the regular solutions are typically based on the operation h(object) mod N, where h() is a hash function specifically chosen for individual system, and N is the total number of caching servers. Consistent hashing partitions key-space among nodes. It contact the appropriate node to lookup/store key and the portioning of key-space is done as follows,

- Nodes choose random identifiers:e.g., hash(IP)
- Keys randomly distributed in ID-space: e.g., hash(URL)
- Keys assigned to node "nearest" in ID-space
- Spreads ownership of keys evenly across nodes

Consistent Hashing has three main features

- Balanced
- Smoothness
- Spread and load

Consider network of n nodes, if each node has 1 bucket Owns 1/nth of key space in expectation Says nothing of request load per bucket. If a node fails then its successor takes over bucket. It achieves the smoothness by doing only localized shift, not O(n) and now successor owns 2 buckets and the key space of size 2/n.

C. Distributed Hash Table (DHT)

Distributed Hash Table (DHT) plays an important role in distributed systems and applications, especially in large-scale distributed environments. In normal Client/Server model (C/S model) most of the resources is centralized at server, it becomes the most important part as well as the bottleneck and weak point of the system. On the contrary, distributed model (a typical one is peer-to-peer (P2P) model) distributes the resources on the nodes in the system. This model requires that utilizing all the peers' capability efficiently and providing better robustness. DHT technology meets the requirements so that it promotes the development of P2P greatly. DHT organizes the distributed resources so well that peers only need to know part of the system they can get resources efficiently. DHT only has two basic operations:

- get data from DHT and
- put data into DHT

It provides good robustness and high efficiency, especially in large-scale systems. It is inspired from traditional hash table:

- key = Hash(name)
- put(key, value)
- get(key) -> value

DHT is a simple and elegant design for distributed systems. It provides the functions like a hash table to deal with the distributed data.DHT does not require a central server and treats all DHT nodes in the distributed system equally. Meanwhile, DHT inherits the great properties of hash table (e.g., locate and search an element with high efficiency). DHT provides a global, abstract key space (often referred to as the DHT space), where all resources (e.g., data and DHT nodes) have unique identifiers (IDs).

Properties of DHT

- High efficiency
- Decentralization
- Scalability

Table I Consistent Hashing VS. DHT

	Consistent Hashing	Distributed Hash Tables
Routing table size	O(n)	O(log n)
Lookup / Routing	O(1)	O(log n)
Join/leave: Routing updates	O(n)	O(log n)
Join/leave: Key Movement	O(1)	O(1)

II. PREVIOUS WORK

There are many papers surveyed the DHTs and their topologies. They also focused on their routing algorithms and how they are implemented on different platforms.

Hao Zhang et al (2013) proposed "A Survey on Distributed Hash Table (DHT): Theory, Platforms, and Applications" [1] in which they summarize the development of DHT in both academic and industrial fields. It covers the main theory, platforms and applications of DHT. They explain the principle of several popular DHT structures, many platforms used in both academic and commercial fields, and a wide range of applications DHT can be deployed. It includes five chapters. First the background information about DHT is introduced. Then seven variants of DHT are studied and compared in many aspects. In 3rd chapter they discussed about the two kinds of platforms (academic and open-source platform and commercial platform) containing fifteen different platforms are analyzed, based on which applications can be constructed. In Next one eight kinds of applications are introduced, and the advantages of DHT in these applications are analyzed. Finally they summarize the power of DHT, and analyze the limitation of DHT.

Filipe Araujo et al(2006) proposed "Survey on Distributed Hash Tables"[4] ,In this paper they review distributed hash tables (DHTs) that work as overlay networks on top of the IP network. They compare these DHTs under a number of key aspects .Additionally, since these networks are not suitable for wireless ad hoc networks, we shortly review techniques that can be used to take distributed hash tables to wireless environments. An overlay network is a network operated on top of another underlying network, but organized under an independent logic. For this reason overlay networks are also deemed as "logical" or "virtual" networks. The growing interest of users in peer-to-peer applications, like Napster3 and Gnutella4 ignited the creation of many peer-to-peer overlay networks that implement DHTs. Although different in their details, most of these DHTs share a number of common features. In a DHT, one can often distinguish between the "routing scheme" and the "routing geometry" of the overlay network. Often, the routing scheme compels the nodes to create an underlying graph with a predetermined organization, which is called the "routing geometry".

D. Korzun and A. Gurtov, (2013),"Flat DHT Routing Topology" [7] they surveyed the existing classical DHTs grouped according to their topologies. They described basic architectures for efficient overlay routing and corresponding classes of graphs. They start with introducing Content Addressable Network (CAN) as an example of Torus topology. It is followed by Chord, Kademlia and Accordion as DHTs using the Ring topology. Pastry, Tapestry and Bamboo DHTs are grouped under the PRR tree topology. Trie and balanced trees are represented by P-Grid, skip graphs. Finally, they present several DHTs that are using De Bruijn and Kautz graphs, Butterfly and O(1)-hop topologies. These topologies differ in how they collect local information in a DHT node about the global network.

K. Gummadi et al(2003)," The Impact of DHT Routing Geometry on Resilience and Proximity",[8], The various proposed DHT routing algorithms embody several different underlying routing geometries. These geometries include hypercubes, rings, tree-like structures, and butterfly networks. In this paper they focused on how these basic geometric approaches affect the resilience and proximity properties of DHTs. One factor that distinguishes these geometries is the degree of flexibility they provide in the selection of neighbors and routes. Flexibility is an important factor in achieving good static resilience and effective proximity neighbor and route selection. In this paper they have not introduced any new DHT algorithms, nor has it presented any theorems. However it has provided some pieces of insight that will be useful in future DHT routing designs.

Ville Nuorvala (2005)," Running an i3 overlay network on top of AODV" [18]. In this paper they briefly discussed the P2P and MANET features, and its main contribution of running the Internet Indirection Infrastructure (i3) P2P overlay protocol over an Ad hoc on- Demand Distance Vector (AODV) based MANET. They also listed the key issues and proposed a solution to the identified problems. The apparently large overhead caused by merging overlays may also limit the scalability of this expanded i3 protocol in any current MANET environment. Dynamic address routing (DART) is an attempt to provide a scalable Ad Hoc routing protocol. DART is a proactive hierarchical routing protocol where the participating nodes configure topologically correct addresses based on periodic routing updates from their neighbours.

Du Li-juan et al (2011)," MARM: Mobility-aware Routing Mechanism in Mobile Grid ",IEEE [16]. In this paper they focused on routing mechanism to support mobility on overlay network layer. Mobility-aware routing mechanism (MARM) proposed in this paper combines distributed hash table technology with the idea of mobile IP. Mobile node doesn't participate in routing process directly, but update its location information dynamically to DHT network which is composed of agent nodes. Mobile Grid is a new type of resource-sharing network, which is featured with large-scale, heterogeneous, dynamic and mobile. Layered architecture is proposed for mobile grid based on the concept of overlay network. This layered architecture has two advantages. On the one hand, it is conducive to achieving system scalability; on the other hand, it can shield high dynamic and mobility of general nodes. They also focused on routing mechanism to support mobility under the hierarchical distributed architecture. Traditional peer-to-peer network is flat overlay network; all nodes are fixed nodes and participate in routing process. While in mobile grid, agent nodes with high performance form stable kernel overlay network which carries out routing function. At the same time, kernel overlay network maintains dynamic information and mobility information of general nodes to achieve routing process when general node as destination.

III. PEER TO PEER OVERLAY NETWORKS

A. P2P systems

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or work loads between peers. Peers are equally privileged, equipotent participants in the application. Email, Internet Relay Chat

and Napster are all examples of P2P systems. Routing on these networks is either centralised or statically configured and is therefore unproblematic.

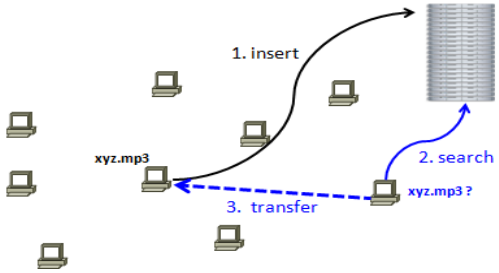

Fig.1. Early P2P 1: Client – Server

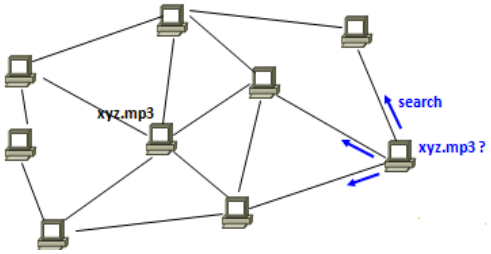

Fig.2. P2P II: Flooding

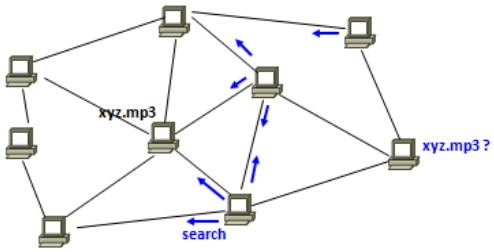

Fig.3. P2P II: Flooding (Search continue towards the peer nodes)

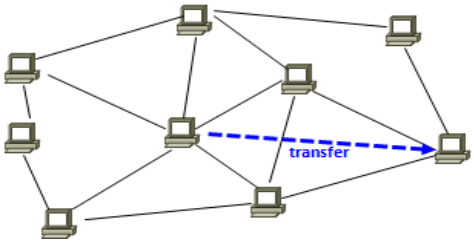

Fig.4. P2P II: Flooding (Transfer the expected file to destination)

B. Overlay Network

Overlay network technology is the rapid development of peer-to-peer network, provides a new idea to distributed management of mobile grid. Overlay networks are routing infrastructures that create communication paths by stitching together more than one end-to-end path on top of the underlying IP network. Overlays have been built to provide

- Multicast
- Reliability
- data storage

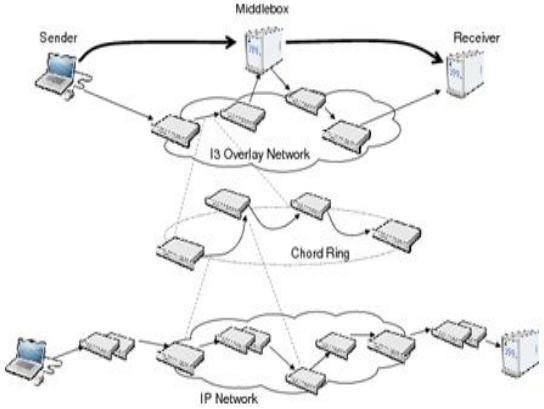

Fig.5.Overlay Network

The term "peer-to-peer" (P2P) is used in many contexts to mean different things. Fig1. Shows that used to refer to the form of cooperation that emerged with the appearance of the music file sharing application Napster. With that application, music files were exchanged between computers (Peers) relying on a central directory for knowing which peer has which file. Fig2 shows that Napster ceased operation due to legal rather than technical reasons and was followed by a number of systems like Gnutella and Freenet, where the central directory was replaced with a flooding process where each computer connects to random members in a peer-to-peer network and queries his neighbours who act similarly until a query is resolved. Thus Flooding is a simple computer network routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on. The random graph of such peers proved to be a feasible example of an overlay network that is an application-level network on top of the Internet transport with its own topology and routing.

C. Classification of P2P Overlay Network

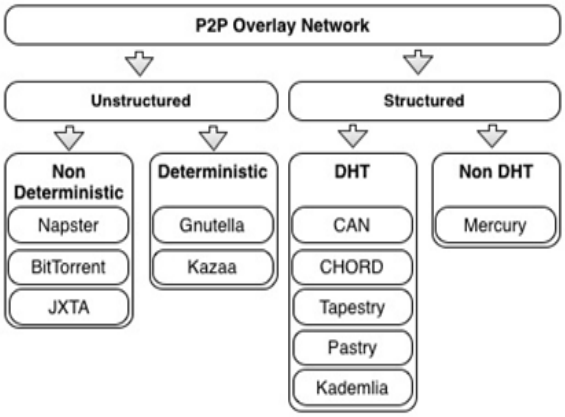

Fig.6. Classification of P2P Overlay Network

The problem of having a scalable P2P overlay network with no central control became a scientifically challenging problem and the efforts to solve it resulted in the emergence of what is known as "structured P2P overlay networks", referred to also by the term Distributed Hash Tables (DHTs).

Table II Comparison to other facilities

Facility	Abstraction	Easy Use/Prg	Scalability	Load-Balance	Fault-Tolerance	Self-Org	Admin
DHT	high	high	high	yes	high	yes	low
Centralized Lookup	medium	medium	low	no	low	no	medium
P2P flooding queries	medium	high	low	no	depends	yes	low
Distributed FS	low	medium	medium	no	medium	no	high

IV. STRUCTURED P2P OVERLAY (DISTRIBUTED HASH TABLE) ROUTING ALGORITHMS

In a DHT, nodes must cooperate to maintain the coherence of the data. To determine the peer, the nodes use a consistent hash function that holds a given value. Usually, nodes have a communication graph which has optimal or near-optimal path length. Nodes also have a small number of neighbours (logarithmic or better) and for this, the DHT needs only a small number of hops (logarithmic or better) to satisfy requests from clients. DHT may resist to single or even multiple node failures, losing only the data that was in the departing nodes. Another advantage of a distributed solution is the reduced congestion. While a centralized server needs to reply to all requests, most DHTs need to reply only to a small fraction of the requests, like O(log n/n), where n is the number of nodes. Furthermore, DHTs scale better than a centralized solution, both in terms of communication and storage requirements. The existing classical DHTs grouped according to their topologies. The basic architectures for efficient overlay routing and corresponding classes of graphs for each types of DHTs are described.Table.2 shows the types of DHTs and their respective topologies.

Table III Types of DHT with their Topology

Types of DHT	Topology
Content Addressable Network (CAN)	Torus
Chord	Ring
Kademlia	Ring
Pastry	PRR Tree Topology
Tapestry	PRR Tree Topology

A. Content Addressable Network

Content Addressable Network (CAN) was one of four DHTs introduced in 2001. In CAN, each node is responsible for an area of key identifier space, called so a zone. CAN maintain the information on nodes which are responsible for adjacent zones. CAN is a distributed, Internet-scale, DHT-based infrastructure that provides hash table-like functionalities. The DHT space in CAN is a d-dimensional Cartesian space. The d-dimensional space is further dynamically partitioned among all nodes and each node only maintains its own individual and distinct zone in the space.

CAN perform the traditional operations such as insert, lookup or delete and they are routed hop-by-hop between zones until the target is reached. In CAN, every node maintains 2d neighbors. The neighbours of two zones overlap along d − 1 dimensions and it have neighbors on one dimension. As a result, CAN does not need to introduce complex structures such as long links (e.g., the finger table in Chord) connecting nodes, which are further away from each other in the d-dimensional space, in order to improve the connectivity and it will helps to reduce the complexity of routing. Note that d is a constant independent from the number of nodes in the system, which means that the number of neighbors each node maintains is a constant, no matter how many nodes the CAN system may have. The main design goals of CAN where

- Scalability
- Fault Tolerance
- P2p Nature.

The CAN is organized as a d-dimensional torus with Cartesian coordinates. This is a purely logical construct where the coordinate space is partitioned between participating nodes.

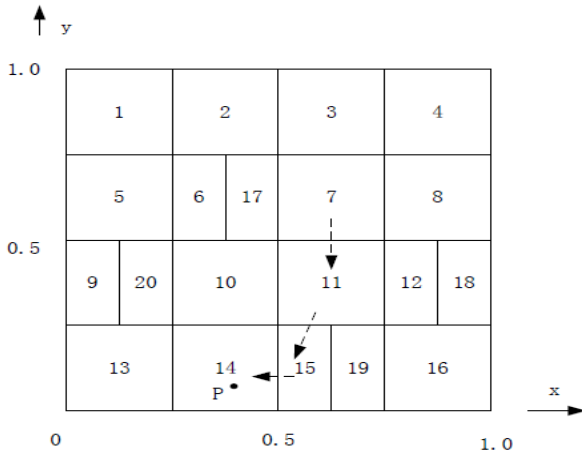

Fig.7. a 2-dimensional CAN with 20 nodes

Fig.7. illustrates a two-dimensional coordinate space which is shared among several nodes. Here the space is drawn as a plane while in fact it wraps up at the edges, forming a torus. Many paths exist between source and destination and greedy routing is used to reach the destination. The routing in CAN works as follows. When receiving a message with a specific destination, a node routes the message towards the destination using a simple greedy algorithm, i.e., the node goes through the list of its neighbours to select the one that is closest to the destination, and then forwards the message to the selected neighbour. This greedy forwarding process continues until the message arrives to the designated destination. The routing path from the node 7 to the point P in the zone maintained by the node 14. The dashed lines illustrate the steps in which nodes greedily forward a message from the source (i.e., node 7) to the destination P.Considering a d-dimensional CAN composed on n nodes, on the average each node would have 2d neighbors and the average path length will be (d/4)(n1/d).

Adding a new node does not affect the node state but increases the average path length as O(n1/d). CAN maintain several independent coordinate spaces called realities which help to reduce the routing delay. Consider the node belongs to r realities, and then it has r independent neighbour sets and r coordinate points in separate zones. Multiple realities improve routing reliability, as a different reality can be used in case routing fails due to node departure within one reality. Other techniques to improve the performance of CAN include using multiple hash functions to map a single key to several points of the coordinate space and hence increase the availability.To improve the robustness and performance of the CAN system, three technologies have been used. They are

- Increasing Dimensions,
- Multiple Realities
- Rtt-Weighted Routing

CAN behave well in large-scale distributed systems and has been applied to many applications in such systems.

B. Chord

Chord, Kademlia DHT's are grouped under the ring topology. The ring is a popular topology for organizing nodes in a DHT. Chord represents a ring DHT which is linked in the clockwise direction. Chord is one of the first and most

influential DHTs. It was developed by Ian Stoica with colleagues at MIT in 2001. In Chord, nodes are organized in a circle of up to 2^m nodes, with each node having an ID from that space. IDs of keys and nodes are of m bit length. Consistent hashing with SHA-1 algorithm are uniformly distributed across the identifier space and they are used to obtain the Ids of keys and nodes. The ID of a node is a hash of its IP address, and the ID of the key is a hash of its attribute, such as a file name. Design of Chord aims at achieving following properties.

1. **Load balancing** assumes even distribution of keys over nodes.
2. **Scalability** requires that the lookup costs grow slower than the number of nodes, thus allowing large system construction.
3. **Decentralization** is the main characteristics of P2P systems in general, and in Chord all nodes are equal without introducing single failure points.
4. **Availability** is guaranteed by automatic maintenance of the node organization in response to node joins and leaves.

In Chord, the DHT space is a circle, which is a one dimensional space referred to as the Chord ring. Nodes organize into a logical ring ordered by increasing order of identifier. To close the ring, the smallest node follows the largest one. To maintain the ring, each node keeps a pointer to the node that follows it. The ring allows defining the notion of successor node of a key. If each node only knows its predecessor and successor in the one-dimensional Chord ring (which is a directed graph), such a DHT system would be inefficient and vulnerable for numerous reasons.

1. The time complexity of DHT lookup is O(n), where n is the number of peers in such a system
2. Each node can only send messages to its successor on the ring
3. Since each node has only one choice for routing, if one node fails, the connectivity of the graph will be destroyed.

To improve routing performance, Chord nodes use "finger tables" with m entries, where 2^m is the number of possible identifiers.

Finger Table

Let m be the number of bits in the key/node identifiers. Each node, n, maintains a routing table with at most m entries called the finger table. The ith entry in the table at node n contains the identity of the first node, s, that succeeds n by at least 2i-1.

s = successor(n+2i-1)

s is called the ith finger of node n

The ring of nodes is linked in clockwise direction. The predecessor is the first node in counter-clockwise direction and the successor is a next node in clockwise direction. A key k is assigned to successor of node with ID equal or greater than k.In naive lookup routing, the ring can be traversed sequentially until a node responsible for a key is located. This is quite slow as about half of the ring would need to be traversed on the average. To accelerate the lookup progress, each node keeps a table of fingers, each pointing to a successor node of $(n+2^{i-1}) \bmod 2^m$, where m is the number of fingers per node, n is the node ID, and i is the finger table index. Using fingers, the lookup time can be significantly reduced, as only O(logN) nodes need to be contacted to locate a key in a ring of N nodes. Using each finger halves the remaining distance to the destination node. Each entry in the finger table includes the node ID, its IP address and port number.

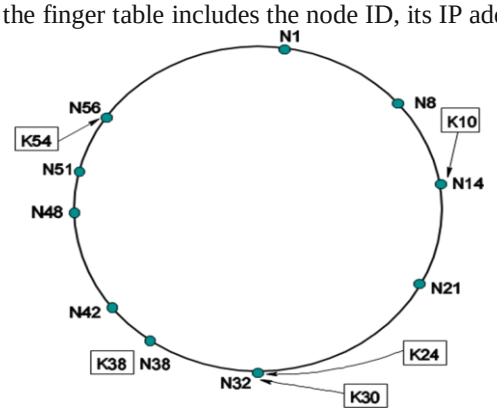

Fig.8. Structure of Chord Ring

Chord supports the following primitive operations such as node join/stabilize, search and node leave/failure. Nodes can join (and leave) at any time. Preserving the ability to locate every key in the network is the biggest challenge. For lookups to be fast, it is desirable for the finger tables to be correct. Each node in Chord maintains a predecessor pointer. This consists of the Chord ID and IP address of the immediate predecessor of that node. It can be used to walk counterclockwise around the identifier circle. The new node to be added learns the identify of an existing Chord node by some external mechanism

- Assume n is the node to join.
- Find any existing node, n'.
- Find successor of n from n'. Label this successor(n).
- Ask successor(n) for its predecessor.

This is labelled as predecessor(successor(n)).

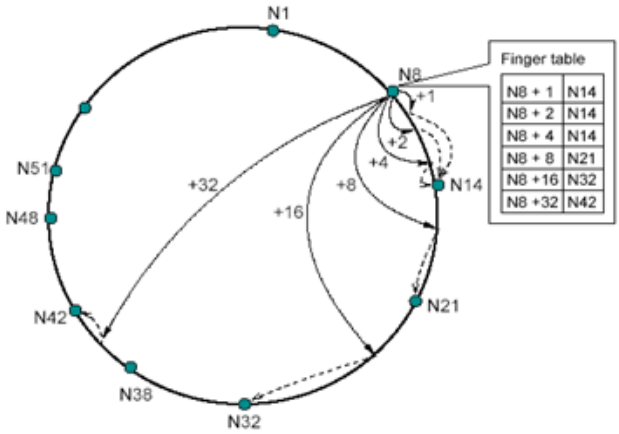

Fig.9. Example of Finger Table

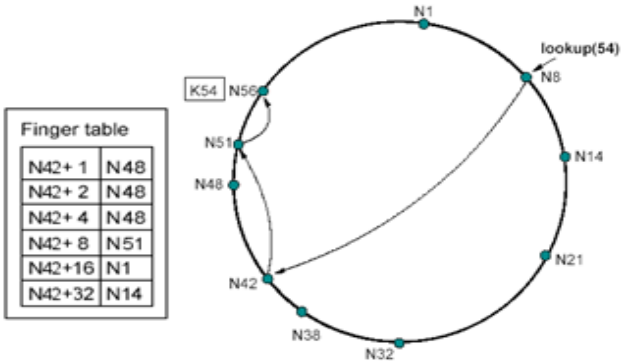

Fig10. Scalable node localization

Join/Stabilize Example

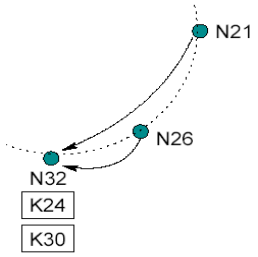

Fig.11. N26 joins the chord ring

N26 joins the system and then N26 acquires N32 as its successor then N26 notifies N32; N32 acquires N26 as its predecessor

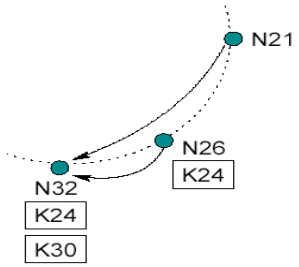

Fig.12. N26 copies keys

N26 Copies the keys and N21 runs stabilize() and asks its successor N32 for its predecessor which is N26

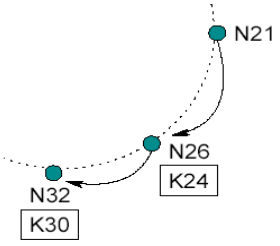

Fig.13. N21 aquires N26 as its successor

Node Failure Example

The solution for node failure is to use successor lists and each node knows r immediate successors. After failure, will know first live successor. Stabilize messages correct finger tables. Replicas of the data associated with a key at the r successor nodes might be used.

Node Search steps

Assume node n is searching for key k. For this node n does the following:

- Find ith table entry of node n such that k∈[finger[i].start, finger[i+1].start]
- If no such entry exists then return the node in the last entry of the finger table

The above two steps are repeated until the condition in the first step is satisfied.

Properties of Chord

In a system with N nodes and K keys, with high probability…

- Each node receives at most K/N keys
- Each node maintains info. about O(log N) other nodes
- lookups resolved with O(log N) hops
- Insertions O(log2N)
- No consistency among replicas
- Hops have poor network locality

Main Considerations in Chord DHT

The main purpose of chord DHT is to locate a target node and for that it should try to get closer to locating target node in each step. There are 3 main aspects have to consider for effective and secure routing,

- Performance
- Scalability
- **Security – 1. Incorrect Lookup**
  2. Inconsistent Behaviour

For security related issues we have a solution and for in correct lookup, the finger table should maintain with correct interval of nodes. Then for Inconsistent Behavior, there are two solutions and they are using either public key or benzantine protocol. Researchers have studied how to improve Chord extensively and there has been a few of such studies, Flocchini et al. proposed a method that combines multiple Chord rings to achieve data redundancy and reduce the average routing path length; Joung et al. proposed a two-layer structure called Chord2, where super peers are introduced to construct a conduct ring which could reduce the maintenance cost; Kaashoek et al. introduced a Chord-like structure Krrode where the bruijn graphs substitute the finger table; Cordasco et al. proposed a family of Chord-based P2P schemes, F-Chord(α), using the Fibonacci numbers to improve the degree, diameter and average path length of Chord. H-F-Chord(α) using the NoN (Neighbors of Neighbors) technique is more efficient in terms of its average path length O(log n/ log log n); Ganesan et al.optimizes Chord routing algorithms by exploiting the bidirectional edges, both clockwise and counterclockwise, which reduces the average routing path length.

C. Kademlia

Kademlia is a DHT introduced in 2002. Although Kademlia is using a tree for routing, with binary identifiers its XOR metric is similar to the prefix metric and the ring topology. Therefore, Kademlia represents a bi-directional version of Chord. It reduces the number of administrative messages needed to maintain routing tables, as nodes learn about the peers during lookup operations. The lookups are made in parallel and non-blocking way to minimize the delays. Its performance characteristics are formally proven. Kademlia is widely used for example by BitTorrent clients for distributed tracking of torrents. Kademlia uses 160-bit IDs for nodes and keys. The keys/value pairs are stored on nodes with sufficiently close ID. XOR metric is used to determine a distance between nodes.

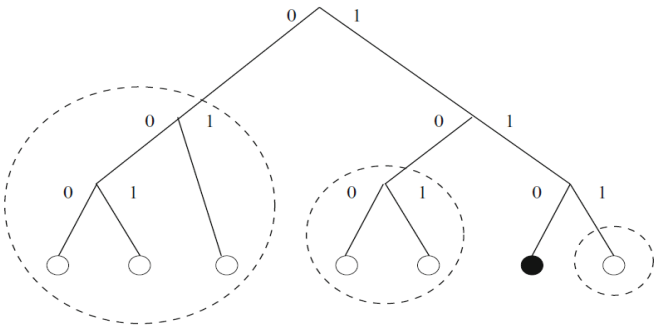

Fig.14. The Kademlia tree for node 110

Kademlia does not separate the routing process into phases but uses the same mechanism until the target node is reached. On each step, the XOR distance to target node is roughly halved. XOR metric is unidirectional that ensures convergence of requests towards the target. Nodes in Kademlia store information about other nodes in a form of (Node

ID, IP address, UDP port) of a distance in the range of 2i and 2i+1, where i is 0. . . 160. These lists form k-buckets sorted by the time of latest communication. Buckets for smaller i tend to be empty, while for larger i will be of size k (e.g., 20) which is a system parameter. The buckets are filled in whether a new message arrives from another node. Live nodes are never removed from the buckets; only nodes that fail to respond to pings are replaced by new nodes. That ensures that oldest live nodes remain as long as possible in the buckets, as those nodes are least probable to fail also in future. Unlike in Chord and CAN, a node updates its neighbors dynamically upon receiving any messages from them. More specifically, when a node i receives a message from another node j, which is located in the m-th k-bucket, this k-bucket of node i will be updated in the following way. If j already exists in the k-bucket, i moves j to the tail of the list, as node j is the node that is the most recently seen. If j is not in the k-bucket and the bucket has fewer than k nodes, node i just inserts j at the tail of the list. If the bucket is full, i pings the node at the head of this k-bucket. If this head node responds, node i moves it to the tail and ignores node j. Otherwise, i removes the head node and inserts j at the tail. Kademlia has four RPC-like primitives, i.e., PING, STORE, FIND_NODE and FIND_VALUE.

1. The PING primitive probes a node to check whether it is online or not.
2. The STORE primitive is used to store a key-value pair.
3. The FIND NODE primitive finds a set of nodes that are closest to a given node; in other words, it returns k nodes from one or multiple k-buckets, whose IDs are closest to the given node's 160-bit ID.
4. The FIND VALUE primitive behaves like FIND NODE, except that it returns the stored value.

When a node i is joining the network, it is assumed that it knows a node j which is active and already in Kademlia. The joining process consists of multiple steps. First, node i inserts j into its k-buckets. Second, node i starts a node lookup procedure for its own ID, from which i learns some of the new nodes. Finally, node i updates the k-buckets. During this process, node i strengthens its k-buckets and inserts itself into other nodes' k-buckets. When a node fails or leaves, it does not notify any other node. There is no need for a special procedure to cope with node departures, as the mechanism of k-buckets ensures that the leaving nodes will be removed from the k-buckets.The simple distance metric and the k-bucket mechanism, Kademlia becomes the most widely used DHT system and it has been adopted by many popular P2P applications such as Overnet , eDonkey/eMule and BitTorrent.

D. Pastry

Pastry, Tapestry DHTs is grouped under the PRR tree topology. Plaxton, Rajaraman, and Richa proposed a PRR tree in 1997. PRR trees allow routing in a distributed publication system. They considered static network topology and did not provide a real-world implementation of the system. Later on, PRR trees were adopted by Tapestry and Pastry DHTs that also include management of dynamic membership of nodes in a tree.

Fig.13, shows shows neighbors of a node in PRR tree. Each node has O(b * logb(N)) neighbors in its routing table on several levels. A neighbor on ith level shares i digits with node ID. Hence, L0 node shares no common digits and L3 node shares three prefix digits. A node with ID closest to the key is the owner of that key.

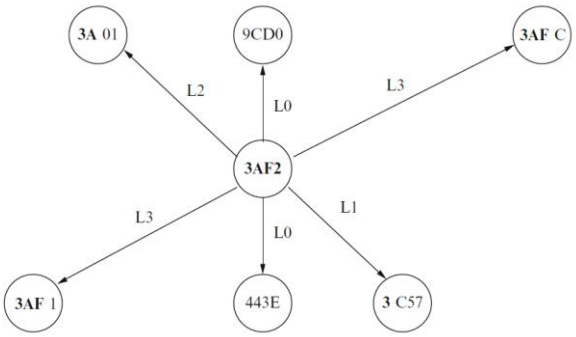

Fig.15. Routing Table of a PRR Tree

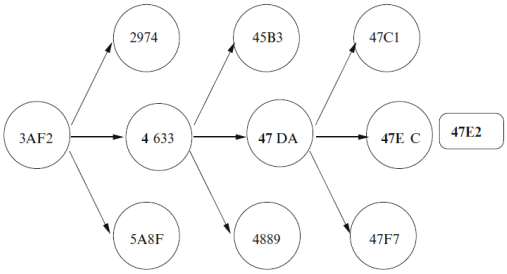

Fig.16. Routing Process in a PRR Tree

Fig.16.shows routing process initiated from node with ID 3*AF*2 to a key with ID 47*E*2. The routing goal is to find a node with the longest matching prefix of digits. The query proceeds to node 4633 first since it shares the first digit with destination. The second hop is node with ID 47*DA* with two digits and finally node 47*EC* which is numerically closest to the key ID and hence responsible for it.

Pastry is a self-organizing overlay network with a targeted basic capability of routing messages efficiently. Every node in Pastry has a 128-bit ID. A node ID is divided into multiple levels, each of which represents a domain. Each domain is represented by b. It is an integer by which 128 is divisible which is contiguous bits in the node ID, *i.e.*, the domain at level *l* is specified by the bits at positions b × *l to* b × (*l* + 1) − 1. Each level contains 2^b domains numbered from 0 to 2^{b-1}. Each node has a routing table, which has a neighbourhood set and a namespace set. The routing table of a node contains 2b − 1 nodes for each level l; these nodes have the same prefix up to level l − 1 as the local node. Hence, the routing table contains L × 2b − 1 nodes, where L is the number of the levels. The neighbourhood set contains M nodes, which are closest to the local node. The namespace set contains L nodes which are closest to and centered on the local node. The namespace set is used during the message routing and object insertion. When a node routes an incoming message, the node first checks if the destination's ID falls in its namespace set. If so, the message will be sent directly to the destination node. Otherwise, the node uses the routing table to choose the domain at a level l, where the nodes at this level l share the longest prefix with the destination node's ID. Then the node selects a node in this domain as the next hop. The selected node has to be alive and be closer to the destination than other nodes in the same domain. Every data object v in Pastry has an object ID k that is at least 128 bits long, which can be generated by a hash function.

When storing an object v into the Pastry network, Pastry routes a message containing the data object v to the node whose ID is numerically closest to k (*i.e.*, the ID of v). In order to improve the data availability, each data object is not only stored on one node but also on a set of extra nodes whose IDs are numerically close to the object ID. In Pastry for Node addition, it is similar to Tapestry,i.e. node routes a message to itself resulting in message being routed to current surrogate (Z)

- Contacts a nearby node A, asking it to route a message to itself
- Obtains i-th row of its routing map from i-th node encountered from A to Z
- Obtains neighbourhood set from A
- Obtains leaf set from Z

For Node deletion in Pastry,

- Nodes exchange keep-alive messages with nodes in their leaf set, if failure detected in leaf set, obtain leaf sets from a live node in the leaf set with largest index (on the side of the failed node); select a node from this leaf set
- To repair failed routing entries, get corresponding entry from another node in the same row of routing table; if that fails, try another node in the next row.

E. Tapestry

Tapestry is one of first DHTs introduced in 2001. It represents a structured peer-to-peer overlay network that guarantees key location in the system when node does not fail badly. Like Pastry, it is based on PRR trees. Tapestry has an alternative term to DHT, Decentralized Object Location and Routing (DOLR). A special feature of Tapestry is ability to exploit locality in accessing object replicas. Each application-specific endpoints (e.g., objects) is assigned with a 160-bit globally unique identifier (GUID), both of which can be generated by using a hash function such as SHA-1. The "distance" between two nodes is assessed digit by digit; for example, a node whose ID is "2341" which is closer to the node "2347" than the node "2338".

Surrogate Routing

An object's root or surrogate node is the node which matches object id in the greatest number of trailing bits. Plaxton's algorithm relies on global knowledge (total ordering) to route queries when it encounters an empty neighbor entry. Tapestry uses a deterministic algorithm to incrementally compute a unique root node. It may involve additional hops in comparison to Plaxton's approach. Expected number of additional hops is 2.

Fault-tolerant Routing

It detects failures via soft-state probe packets and route around problematic hop via backup pointers. It has 3 forward pointers per outgoing route (2 backups).

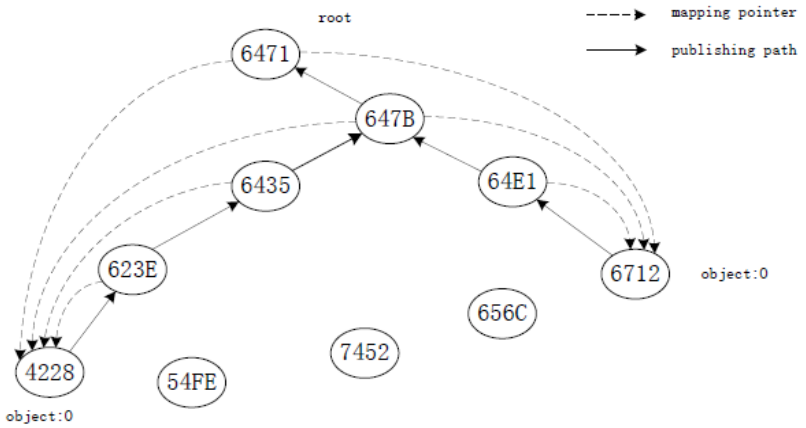

Fig.17. An example of storing (publishing) a data object in Tapestry

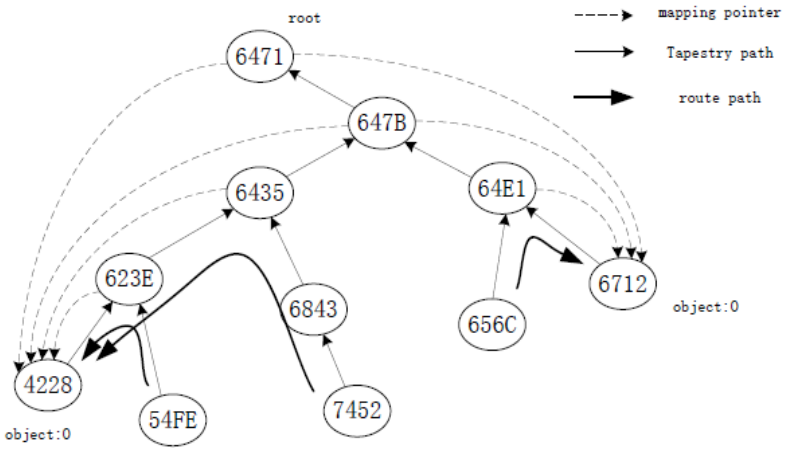

Fig.18. An example of retrieving a data object in Tapestry

Tapestry solves the problem in two cases: voluntary node deletion and non-voluntary node deletion. In the first cast, the leaving node i notify all nodes that are related to i and moves the objects it maintains to the new root. In the second case, node i leaves or fails without any notification. In this case, nodes rely on periodical messages to detect whether the outgoing link and node fails. Furthermore, Tapestry builds redundant pointers in routing tables to improve robustness to node departures. A combination of these techniques retains nearly 100% success rate for routing messages.

F. Global Information Sharing Protocol (GISP)

GISP is a protocol for fully decentralized P2P networks. GISP does not make any assumption on the network structure. It is applicable to both structured and unstructured P2P networks. To design GISP there are a set of key principles.

1. Each node should maintain as much peer information as possible, so the network has great connectivity.
2. Each node should discard information of unreachable peers and keep more information about nodes that are numerically closer.
3. Each node may possess different levels of capability; however, the more capability one node possesses, the greater responsibility the node should take.

GISP introduces the notion of "peer strength" to quantify the capability of a node. The powerful nodes should keep more data and have more connectivity than those with less peer strength. The routing model in GISP works as follows. GISP leverages hash functions such as MD5 and SHA-1 to map any binary data including a keyword into a number with fixed bit length. The hash space for GISP is similar to Chord and CAN and each node has a unique ID. GISP defines the distance between two nodes by distance (i, j)/(2si−12sj−1), where i and j are the IDs of two nodes, si and sj are the values of the two nodes' "peer strength". GISP adopts a greedy routing strategy,i.e., it finds the shortest distance to reach the destination. The data storage and retrieval are similar to Chord. GISP selects the node that has the shortest distance to the hash value of the data. For example, suppose that a GISP system has four nodes, whose ID are 100, 110, 115, 122 respectively. If a piece of data with key 107 is pushed in the system, then the node with ID is 110 would be selected to maintain the data. Data retrieval is similar to the data storage process. More specifically, given a key k, the node that is responsible for maintaining the data is located and queried, and then the data will be routed back to the requesting node. Data storage and retrieval are conceptually straightforward; however, it is difficult to completely delete any stored data.

In GISP, when a node stores a piece of data, it also set up a timer for the data; each node periodically check the data it maintains and delete the expired data. Since node failures in P2P networks are not uncommon, in GISP the data is duplicated and the replicas are distributed to multiple nodes whose IDs are closer to the hash value of the corresponding data. The number of data replicas is determined either statically or dynamically. The larger the number is, the more robust the system is to multiple node failures, and the more storage capacity is required. As a result, routing messages are not sent to only one node but to a group of peers i.e., as per the distance from the nodes to the hash value of the data. GISP can still work well even when too many peers fail at the same time and as a result too much data stored in the network is lost. GISP introduces a latency-based mechanism to relate the overlay network topology with the real underlying physical network topology. Such a mechanism can reduce the cost of network routing in GISP. By doing so, nodes that are closer in the underlay network topology are likely to form clusters (i.e., the distances/latencies between nodes in such clusters are lower) in the overlay network topology.

G. Comparison of DHT Routing Algorithms

To compare the DHTs, we evaluate their most significant features:

- performance and scalability
- self-configurability
  - Path Latency
  - Fault Tolerance
  - Resistance to Churn

Performance and scalability - as the network size increases up to thousands or millions of nodes, scalability depends heavily on the behaviour of the following factors: ability of nodes to share the load, growth of path lengths versus degree of nodes and congestion at the nodes. While most DHTs assume that load sharing is ensured by hashing (and possibly randomization), figures for path lengths and node congestion may vary.

Self-configurability - To evaluate the self-configurability of the DHTs, we focus on the ability of the network to choose better paths, on the tolerance to faults and on churn. Resistance to churn can be seen as a particular case of tolerance to faults.

Table IV Comparison of Structured P2p Overlay (Dht) Routing Algorithms

Algorithm	**Overlay Network Topology**	**Distance from i to j**	**Routing path length**	**Routing path selection**	**# of maintained peers**	**Node's joining**	**Node's leaving**
Chord	one-dimensional ring	(j - i) mod 2m	O(logN)	Greedy Algorithm	O(log2 N)	find successor; construct finger table	run stabilization protocol periodically
CAN	d-dimensional cube	Euclidean distance	$\frac{?}{4} N^{\frac{1}{?}}$	Greedy Algorithm	2d	generate neighbor list	update neighbor lists
GISP	structureless	objects: the difference of the two IDs; nodes: (i, j)/(2si- 12sj-1); si, sj are "peer strength"	uncertain	Greedy Algorithm	as many as possi-ble	generate routing list	delete failing nodes from routing list
Kademlia	XOR-tree	i + j	O(log2 b N)	Greedy Algorithm	at most 160 × k(O(log2 N))	construct k-buckets	detect the target node before routing
Pastry	tree+ring	assessed digit by digit	O(log2 b N)	Greedy Algorithm	O(log2b N)	generate routing table, neighborhood set and a namespace set	detect the target node before routing
Tapestry	tree	same as Pastry	O(log2 b N)	Greedy Algorithm	O(log2b N)	construct routing table	heartbreak message

H. DHT Platforms

Based on the theory of DHT, many researchers develop platforms that implement different kinds of DHT and provide interfaces and services to applications. There are many issues and requirements will be exposed, such as load balance, multiple replicas, consistency, and latency. Some platforms satisfy only the basic functions including implementing specific DHT and providing interfaces to the upper applications, such as Open Chord, Chimera, Free Pastry and etc. Some other DHTs also supply specific services, such as CoDeeN and CoralCDN for caching, hazelcast for data distribution. Some supply additional guarantee, such as GNUnet for privacy and security. Some focus on providing a platform for connecting all kinds of devices, such as JXTA. We surveyed the several DHT implementations in both academic/open source platforms and commercial platforms respectively and they are as follows, Bam-boo, CoDeeN, CoralCDN, OpenDHT, JXTA, GNUnet, Open Chord, hazelcast, i3, Overlay Weaver, Cassandra. These all are open-source platform that are free and allow other people to contribute to the system, which facilitates the platform to grow and improve, but on the other hand, limits the maintenance and stability of the system. Finally the Commercial DHT platforms, including WebSphere eXtreme Scale,Dynamo, SandStone, and Service Routing Layer. Thus DHT s are implemented in several different platforms and they are commonly grouped as Open Source and Commercial DHT Platforms.

V. CONCLUSIONS

Mobile grid uses overlay network to organize nodes and its routing mechanism uses structured peer-to-peer routing protocol. Traditional peer-to-peer network is a flat overlay network. Thus all stable nodes and nodes in routing process are fixed. While in mobile grid, agent nodes with high performance form stableP2P overlay network which carries out routing function. At the same time, P2P overlay network maintains dynamic information and mobility information of general nodes to achieve routing process in a efficient manner. In this paper we surveyed some of the most well-known DHTs along with their corresponding Topologies. The classic DHTs of CAN, Chord, Pastry and Tapestry are based on d-dimensional torus, a ring, and PRR tree topologies.These DHTs are implemented as peer-to-peer overlay networks working on top of IP. We also make a detailed comparison in the following aspects: performance, scalability and self-configurability. Thus we focused and surveyed on the different structured DHT routing algorithms and way it works for Mobile Grid applications.

REFERENCES

[1] Hao Zhang, Yonggang Wen, Haiyong Xie, and Nenghai Yu (2013), "*A Survey on Distributed Hash Table (DHT): Theory, Platforms and Applications*".

[2] Distributed hash table, http://en.wikipedia.org/wiki/Distributed_hash_table.

[3] Gnunet: Gnu' framework for secure peer-to-peer networking. https://gnunet.org/.

[4] Filipe Araujo and Luis Rodrigues. Survey on distributed hash tables, 2006. https://estudogeral.sib.uc.pt/handle/10316/13603.

[5] Siamak Sarmady. A survey on peer-to-peer and dht, 2010. http://www.sarmady.com/siamak/papers/p2p-290808-2.pdf.

[6] Sameh El-Ansary and Seif Haridi (2004), "*An Overview of Structured P2P Overlay Networks* ".

[7] D. Korzun and A. Gurtov, (2013),"*Flat DHT Routing Topology",Structured Peer-to-Peer Systems: Fundamentals of Hierarchical Organization, Routing, Scaling, and Security*, DOI 10.1007/978-1-4614-5483-02,© Springer 2013

[8] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy,S. Shenker, I. Stoica_,"*The Impact of DHT Routing Geometry on Resilience and Proximity"*,(2003), SIGCOMM'03, August 25–29, 2003, Karlsruhe, Germany.

[9] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan (2001), "*Chord: A Scalable Peertopeer Lookup Service for Internet Applications*", Research was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Space and Naval Warfare Systems.

[10] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. (2003)"*Graph-Theoretic Analysis of Structured Peer-to-Peer Systems; Routing Distances and Fault Resilience*. In Proceedings of the ACM SIGCOMM

[11] Fonseca, P., Rodrigues, R., Gupta, A., Liskov, B.: Full-information lookups for peer-to-peer overlays. IEEE Trans. Parallel Distrib. Syst. 20(9), 1339–1351 (2009)

[12] Manku, G.S.: Routing networks for distributed hash tables. In: PODC '03: Proceedings of 22nd Annual Symposium on Principles of Distributed Computing, pp. 133–142. ACM (2003). doi: http://doi.acm.org/10.1145/872035.872054

[13] Gurmeet Singh Manku, Stanford University, manku@cs.stanford.edu*," Routing Networks for Distributed Hash Tables*",(2003), PODC 2003, July 1316, Boston, MA.

[14] J. Aspnes, Z. Diamadi, and G. Shah. "*Fault-tolerant routing in peer-to-peer systems*." In Proc. 21st ACM Symp. on Principles of Distributed Computing (PODC 2002), pages 223–232, Jul 2002.

[15] Nico Saputro a, Kemal kkaya , Suleyman Uludag (2012), "*A survey of routing protocols for smart grid communications*"- Survey Paper published in Elsevier.

[16] Du Li-juan, Cao Zhi-wen (2011)," *MARM: Mobility-aware Routing Mechanism in Mobile Grid* ",IEEE.

[17] ALI GHODSI(2006)," *Distributed k-ary System: Algorithms for Distributed Hash Tables* ",TRITA-ICT/ECS AVH 06:09ISSN 1653-6363 ISRN KTH/ICT/ECS AVH-06/09.SE and SICS Dissertation Series 45ISSN 1101-1335

[18] Ville Nuorvala (2005)," *Running an i3 overlay network on top of AODV"*, HUT T-110.551 Seminar on Internetworking 2005-04-26/27

[19] Eric Rescorla, *Introduction to Distributed Hash Tables*, Eric Rescorla IAB Plenary, IETF 65.

[20] Min Cai, Kai Hwang, Yukwong Kwok, Shanshan Song, And Yu Chen University of Southern California,(2005),IEEE.

[21] Shahbaz Akhtar Abid And Mazliza Othman, Nadir Shah, (2014), "*A Survey on DHT-Based Routing for Large-Scale Mobile Ad Hoc Networks*", ACM Computing Surveys, Vol. 47, No. 2, Article 20, Publication date: August 2014.

[22] X. Shen, H. Yu, J. Buford, and M. Akon. 2010. *Handbook of Peer-to-Peer Networking*. Springer, Berlin,Heidelberg.

[23] V. V. Thanh, H. N. Chan, B. P. Viet, and T. N. Huu. 2009. *A survey of routing using DHTs over wireless sensor networks*. In Proceedings of the 6th International Conference on Information Technology and Applications (ICITA'09).

[24] Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: *SkipNet: a scalable overlay network with practical locality properties*. In: USITS'03: Proceedings of 4th USENIX Symposium on Internet Technologies and Systems. USENIX Association (2003)

[25] Sushant Jain, Ratul Mahajan, and David Wetherall. "*A Study of Performance Potential of DHT-based Overlays"*. In Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems (USITS), Seattle, WA, USA, March 2003.

[26] CHEN Yan(2014*)," Distributed hash table based routing algorithm for wireless sensor networks*", 2014 Fifth International Conference on Intelligent Systems Design and Engineering Applications,IEEE

[27] Eng Keong Lua, Jon Crowcroft, And Marcelo Pias, "*A Survey And Comparison Of Peer-to-peer Overlay Network Schemes*",(2005),IEEE Communications

[28] Paul Stacey1, Damon Berry1 and Eugene Coyle1,(2004) "*Using Structured P2P Overlay Networks to Build Content Sensitive Communities "Parallel and Distributed Systems*, ICPADTenth International Conference on IEEE Communications

[29] Angela Sara Cacciapuoti, Marcello Caleffi, and Luigi Paura , chapter 6 in "*Mobile P2P: Peer-to-Peer Systems over Delay Tolerant Networks"*

[30] Isaac Woungang, Fan-Hsun Tseng, Yi-Hsuan Lin, Li-Der Chou,(2013)," *MR-Chord: Improved Chord Lookup Performance in Structured Mobile P2P Networks*",IEEE Systems Journal

