HUP: An Unstructured Hierarchical Peer-to-Peer Protocol

Guruprasad Khataniar and Diganta Goswami [*]

Abstract— **This paper presents an organized network architecture for unstructured peer-to-peer systems where nodes are added to the network in a systematic way to efficiently utilize the node resources. This network architecture is characterized by $O(\log_m n)$ network diameter and $O(\log_m n)$ messages for node joining and node failure, where n is the number of nodes in the network and m is the number of children of a node. Purely decentralized systems like Gnutella route the query in an environment where the node capabilities are not identified. Whereas the proposed Hierarchical Unstructured p2p (HUP) routes the query towards the *high capable nodes*. This organization of nodes improves the probability of query success rate than that of purely unstructured systems.**

Keywords: P2P network, Token, Overlay structure, Node capability, Time-to-Live (TTL)

1 Introduction

In Unstructured systems, key is randomly assigned to the nodes which saves network maintenance cost. Peers are also not organized in the overlay network. For processing any query in the system the source node simply floods the query through all nodes. Since there is no particular rule for assigning data to nodes, these systems do not ensure 100% success rate in resolving the query. But these systems address security and anonymity issues efficiently in comparison to structured peer-to-peer systems. This research is focussed on the design of an efficient unstructured system which approaches the overlay routing performance as that of structured systems without sacrificing the functionality of the system.

This work is motivated from free riding on Gnutella [1] , weakness of DHT-based system to support range query, fuzzy queries and problem of transient nodes population and heterogeneity among them which have recently motivated considerable research in P2P network. By sampling messages on the Gnutella [9] network over a 24-hour period , it has been found that 70% of the Gnutella users share no files, 90% of users do not response to the query and nearly 50% of all responses are returned by the top 1% of sharing hosts. Another Gnutella analysis shows that 7% peers share more files than all other peers and 47% queries are responded by the top 1% peers and failure of Gnutella is due to the non identification of heterogeneity of nodes and lack of co-operation. An observation of Gnutella [2] reveals that out of 100 random searches, 95 searches failed to provide results.

In our approach the query is directed towards the nodes that have high capabilities and thus ensures higher chances of query resolving. Rest of this paper is organized as follows: section 2 presents related work, section 3 describes the overlay network, section 4 describes the algorithms to handle node joining and failure and query processing, section 5 describes HUP system performance, section 6 concludes the paper.

2 Related Work

B. Hudzia et al. [4] describe a tree based peer-to peer network that constructs the tree based on leader election algorithm where propose a hierarchical P2P network architecture based on a dynamic partitioning of a 1-D space.

A multi-tier capacity aware topology is presented in [5] to balance the load across the nodes so that low capable nodes do not downgrade the performance of the system.

S. Min et al. [6] describe a super peer based framework in which the peers are organized based on their responsibilities. Normal peers always send the query to the super peers. The main contribution is reduction of the bandwidth cost by selection of best SP. Also the workload of SP can be improved by dynamic prediction based CPU load.

Terpstra et al. [8] propose a simple probabilistic search system, BubbleStorm, built on random multigraphs. their primary contribution is a flexible and reliable strategy for performing exhaustive search.

In our approach of hierarchical unstructured peer-to-peer, a hierarchical overlay network structure is designed based on the nodes sharing to the network and its stability. A node gets access to the resources based on its contribution level. This approach of routing the queries regulates free riders, which consume the system resources.

[*]Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati-781039, INDIA Tel/Fax:91-9864055805 Email: {gpkh, dgoswami}@iitg.ernet.in, Manuscript submission: December'2009

3 System Model

Here we introduce a 2-tier Hierarchical Unstructured peer-to-peer system which can be a answer to the many problems that we have explained in the previous sections.

3.1 Terminology

i. ***Complete Node***: A node in the network is said to be complete, if it has m child nodes, where m is the maximum number of children a node can have in the network.

ii. T_{stab}: This is the time for which a new node is delayed before updating the routing table to deal with the transient node population.

iii. ***High capability node***: A node P is said to be a high capability in comparison to node Q, if it provides more sharing to the network than that of Q and is more stable than Q. Stability of a node is defined by the amount of time for which it is in the P2P network. More time a node is in the network, more is the stability.

iv. ***Low capability node***: A node P is said to be a low capability node in comparison another node Q if it provides less sharing to the network than that of Q and is less stable than Q.

v. ***Token***: This is a control frame that moves through out the cluster. Only the token holder is allowed to add nodes to the cluster. Token flow in the network controls the network growth and ensures that nodes form a hierarchy in the system.

3.2 Proposed Overlay Structure

Nodes in a network differ in terms of their computing power, communication capacity, stability, available memory and sharing size. It has been observed that unidentified node capabilities and unorganized network are the reasons for the failure of Gnutella. In our approach of hierarchical unstructured p2p, a nodes position in the network is determined by its capabilities (sharing size). In this model, we categorize nodes into two types: *Super Node* and *Normal Node*. Super Nodes are placed in the first level of hierarchy. These are assumed to be in the network for most of the time and are connected by mesh topology. Every Super node has a *Token* to build the hierarchy of nodes in its cluster. All the Super node *ids* are maintained in Super-Node table. Table 1 shows the Super node table for Super node S_0 of Figure 1. Figure 1 shows the hierarchical structure for $m = 2$, where the network is divided into four clusters. Number of nodes in a cluster may vary from one cluster to another. For instance, some clusters may have thousands of node while others may have only few.

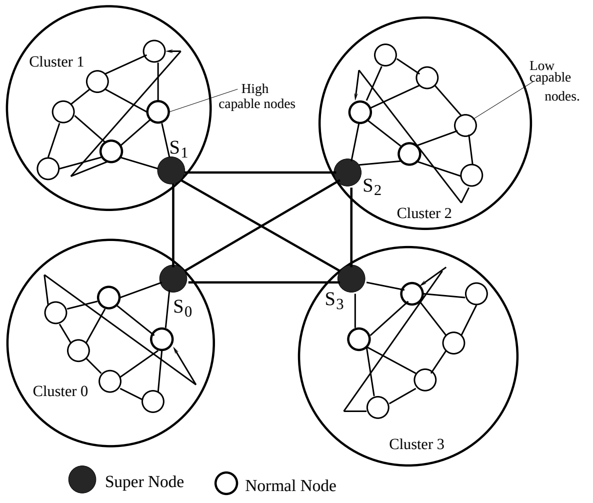

Figure 1: 2-tier Unstructured Hierarchical system

Nodes (Normal Nodes) in a cluster are differentiated by the amount of sharing, bandwidth and processing power etc. Nodes with relatively high sharing, which are called as High capability nodes are placed in the upper levels of hierarchy and Low capability nodes are placed in the lower levels of hierarchy.

Table 1: Super Node table

Super Node ID	Time
–	–
–	–
–	–
–	–
–	–

Peers in a network differ in terms of their computing power, communication capacity, available memory, stability and sharing size. In purely unstructured networks nodes render to scoped flooding because of the unorganized network and un-identified node capabilities.

4 Algorithms

This section provides the basic algorithms used in this overlay system.

a. **Handle token**: This algorithm describes how a node handles the token. Token moves from one node to other in the hierarchy to ensure that network grows in an ordered fashion. It works as follows -

- Whenever a node receives the token from its sibling node and if it is not a complete node it will try to add node to itself to become a complete node. So it keeps the token.

- Otherwise if it is already a complete node then it pass the token to the its right sibling so that it may add nodes and become a complete node.
- If there is no right sibling then give the token to parent node. Continue this till the parent is null.
- Then give the token to the first left child. Continue till the left most node of the tree is found.

Whenever a token is lost in the system due to high churn rate or any other reason then the nodes will not be able to get the token and therefore will not be able to add nodes. In such case the Supernode generates a new token and place the token with Algorithm [*a*].

b. **Find Token**: This is run by a node whenever a new node contacts it for joining the network or when some node sends “Find-Token” message. Let a new node P has contacted the node Q for joining the network.Then

- If Q has the token, P is directly added to the network using algorithm [*c*] (explained after this).
- If Q does not have the token, it extracts the *id* of the node to which it has passed the token from the log and passes the request to it with its *id* as the source *id*. Source *id* here means the *id* of the node which is initiating the request, whereas log of a node contains the *id* of the nodes to which it has passed the token most recently.
- If it does not have the token repeat step 2 with that node till the token is found and report that node *id* to the source node.
- If there are no entries in the log, node forwards the request to a parent node with its *id* as the source node. Then that parent node will repeat step 1,2,3 to find the token holder and report the source node.

c. **Add-Node**: This algorithm is run by a node when it wants to add a new node to the group. Here new node means the node to be added to the system, old node means the node in which we are going to add the new node and forwarded node means those nodes which can not handle the new node and passes the request to the upper level.
Sharing size of new node, $P = P.S$
Sharing size of old node, $O = O.S$

- If $O.S > P.S$ and O is not complete then simply add P as a child node of O.
- Increment the child count of node O.
- Handle token (node O)
- If $O.S < P.S$, add the node to O and call Balance network(O, P)
- Update active and passive routing tables of the nodes.

  Algorithm [*c*] adds a new node to the network. With respect to the sharing of the new node, it is placed in the appropriate level in hierarchy. Algorithm [*d*] describes the way the hierarchy is balanced if a new node P replaces a node Q already in the network.

d. **Balance-Network**(O, P): Whenever a node that has higher sharing size than the node to which it comes to add as a child we have to balance the cluster. Because in our approach we keep the node in hierarchy according to their sharing size, so higher sharing nodes should go to higher level.

- If $O.S < P.S$, replace node O with node P
- If node O holds the token pass it to node P
- Increment the child count of node P.
- If $P.S < Parent(P).S$ then stop, otherwise Balance Network$(Parent(P), P)$, where $Parent(P) =$ parent of node P
- Update routing tables for the nodes.

e. **Remove node**: This algorithm is run whenever a node leaves the network and there is a requirement of balancing the whole network.

- If the leaving node is a leaf node decrement the child count of its parent node.
- If it holds the token pass it to the right sibling.
- If it is the rightmost leafnode then pass it to the parent untill the parent is null and then pass it to the leftmost child of the network.
- If the leaving node is not leaf node then lift its leftmost child as the parent node and call Balance network(P, C), where P is the parent of the leaving node and C is the leftmost child of the leaving node.
- Update the routing tables of the nodes.

4.1 Node Joining

Based upon the above algorithms we can explain the node joining process in this system.

i. When a node wants to join the network, it sends k ping messages to already connected nodes in the network.

ii. Based on the observed Round Trip Time (RTT), new-node sends the “Join” request to the node which is within its proximity.

iii. The receiving node monitors the new node for T_{stab} time units and acts as a proxy for the new node till that time. Then it uses Algorithm [*b*] to find the *id* of the current token holder and gives it to the new node.

iv. The new node sends the message to the token holder.

v. Token holder runs the Algorithm [*c*] to add the node to the network

4.1.1 Analysis

Major steps involved in node joining are :

i. Find-token

ii. Add-node.

As explained in Algorithm [*a*], the token frame of the network moves from one level to other in an ordered fashion. Token movement ensures that the peer-to-peer network grows level by level.

In hierarchical unstructured architecture, number of children of a node is fixed to m. With n nodes in the cluster and m being the number of children of each node, the height of the network is $O(\log_m n)$. In worst case, the token holder is at the left most bottom of the hierarchy and a new node contacts the node at right most bottom of the hierarchy. In this case "Find-Token" message will be transferred from each node to its parent node and then from the topmost node it will again be transferred to its leftmost child node. Since the height of the network is $O(\log_m n)$ it will completely traverse twice the height of the network. Thus the total number of messages transferred is $2\log_m n$ in the worst case. Algorithm [*c*] adds the node to the corresponding hierarchy based on its capabilities. This in turn uses Algorithm [*d*] which balances the network along the path of insertion. When a node is added to the network, at most $4\log_m n$ messages are passed in the overlay network. Concluding the procedure, addition of a node takes $O(\log_m n)$ messages.

4.2 Query Processing

Initially a node forwards the query towards the upper levels of the network. If the node can not find the data with maximum allowable TTL, it forwards the query towards the lower levels of the network. For the first alternative, Source node forwards the query to parent, left and right nodes. For the second alternative, Source node forwards the query to left child, left and right nodes. Then each node forwards the query to it's left child. Algorithm [*f*] and [*g*] describes the way a query is processed in the further steps.

Algorithm *f*: Process-Query

```
Process-Query(Query $Q$, Source Node $S$, Forward Node $F$)
// Node $P$ is running the procedure.
  i. Send the response if the Query $Q$ matches the local index of the node.
  ii. Decrement $TTL$ of the Query $Q$.
  If ($TTL$ of Query $Q$ is greater than zero)
   begin-if
     If ($P$ is a Super node)
       Forward the Query $Q$ to left child of $S$ and broadcast the Query $Q$ to Super nodes of other clusters.
       Executes Algorithm [$g$].
     Else If ($F$ is child node of node $P$)
       Forward the Query $Q$ to left node, right node and parent node.
     Else
       Forward the query $Q$ to other node at same level.
   End-if
End Process-Query
```

Algorithm *g*: Query-Down

```
Query-Down(Query $Q$, Source Node $S$, Forward Node $F$)
// Node $P$ is running the procedure.
  i. Send the response if the Query $Q$ matches the local index of the node.
  ii. Decrement $TTL$ of the Query $Q$.
  If ($TTL$ of Query $Q$ is greater than zero)
   begin-if
     If ($P$ is a Super node)
       Forward the Query $Q$ to left-child.
     Else If ($F$ is parent node of node $P$)
       Forward the Query $Q$ to left node, right node and left child node.
     Else
       Forward the query $Q$ to other node at same level.
   End-if
End Query-Down
```

The peer-to-peer network is organized in two levels with Super Nodes and Normal Nodes at the first and second levels of hierarchy respectively. Super Nodes are connected by mesh network and each Super Node has a cluster of nodes organized in a hierarchy according to their sharing. In order to decrease the number of unnecessary messages and increase the probability of finding the desired data soon, the query is forwarded to the upper levels initially. If a Super node gets the query, it forwards the query to all the Super nodes which in-turn apply the Algorithm g for routing the query. But in the neighboring clusters the query moves down the hierarchy, giving priority to the High capable nodes than the Low capable nodes. If the querying node does not get the desired response then the query is forwarded to the lower levels of the network. Algorithm g describes the way the query is forwarded in the network.

Modeling the Gnutella as a random graph, assuming the maximum degree for each node, Figure 3 shows the way the query is forwarded in purely unstructured system. In HUP, as shown in Figure 2, query is initially forwarded towards more capable nodes, whereas in Gnutella the query is forwarded to unidentified nodes.

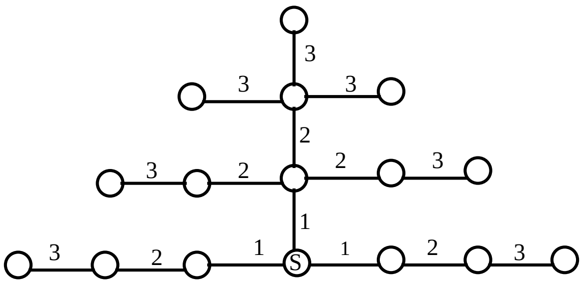

Figure 2: Query routing in HUP, label indicates TTL value

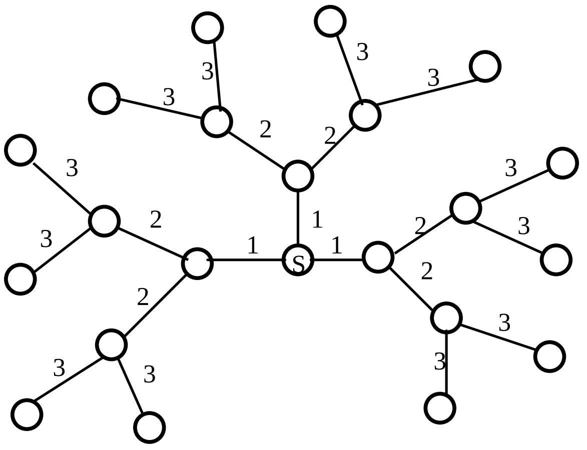

Figure 3: Routing in Gnutella for degree 3, label indicates TTL value.

5 Performance Evaluation

We have evaluated the performance of the proposed system against Gnutella in Java. An analysis [2] of Gnutella shows the top 20 file types and queries in the Gnutella network. We have mapped each of the file types on to the number space. Each file type corresponds to a part of the number space. Data that is available to the nodes in the simulation environment is according to the query rates.

5.1 Node Joining

Figure 4 shows the number of messages vs sample nodes added to the network with different cluster sizes. Nodes which take equal number of messages are grouped together and denoted as *samples*. As described in the section 4.1 number of messages for node joining are $O(\log_m n)$. In the simulation we fixed the number of children of each node is 2 i.e. $m = 2$. Figure 4 is in correspondence to the analysis in section 4.1. Maximum number of messages consumed are multiple of $O(\log_2 n)$.

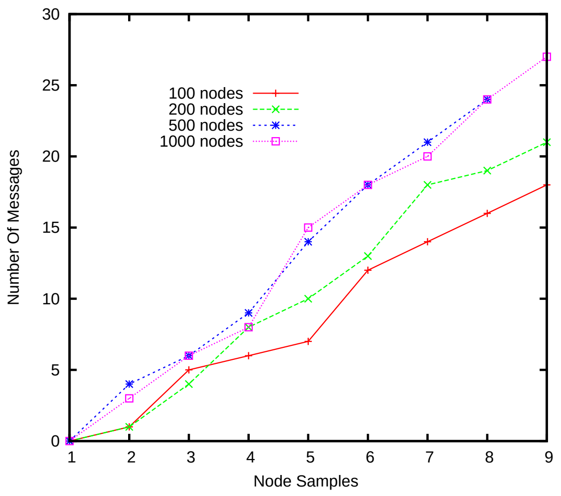

Figure 4: Node Addition

5.2 Query Processing

Table 2 shows the top 20 file types requested on Gnutella network with mapping on to the number space for simulation.

Figures 5 shows the query success rate vs TTL of the query. As the TTL of the query increases the success rate of queries increases. For smaller networks, the success rate for small TTL is high. As the network size increases, for small TTL it is less. Figures 6 show the success rate vs network size. Observations reveal that with TTL=$log_M N$, success rate is at least 50%.

6 Conclusion

We have presented a hierarchical overlay network in which node joining and failure are handled in an organized fashion. Because of the way the nodes are organized improves the connectivity, reduces the number of unnecessary messages and increases the success rate. With n nodes in the system and $\log_2 n$ TTL, query success rate in HUP is atleast 50%. The kind of policy adapted for routing the queries regulates the new nodes which are greedy to consume the resources.

Table 2: Top 20 Filetypes Requested on Gnutella in Queries with mapping of file types on to number space.

#	Filetype	%	Mapped Number space
1	avi	18.72	1 – 1000
2	mp3	17.84	1001 – 2000
3	mpg	13.10	2001 – 3000
4	ra	8.5	3001 – 4000
5	rm	2.79	4001 – 5000
6	zip	2.64	5001 – 6000
7	mpeg	2.63	6001 – 7000
8	jpg	1.9	7001 – 8000
9	asf	1.11	8001 – 9000
10	ps	0.97	9001 – 10000
11	mov	0.95	10001 – 11000
12	pdf	0.51	11001 – 12000
13	rar	0.44	12001 – 13000
14	exe	0.4	13001 – 14000
15	wav	0.25	14001 – 15000
16	doc	0.21	15001 – 16000
17	txt	0.07	16001 – 17000
18	gz	0.07	17001 – 18000
19	html	0.02	18001 – 19000
20	jpeg	0.02	19001 – 20000

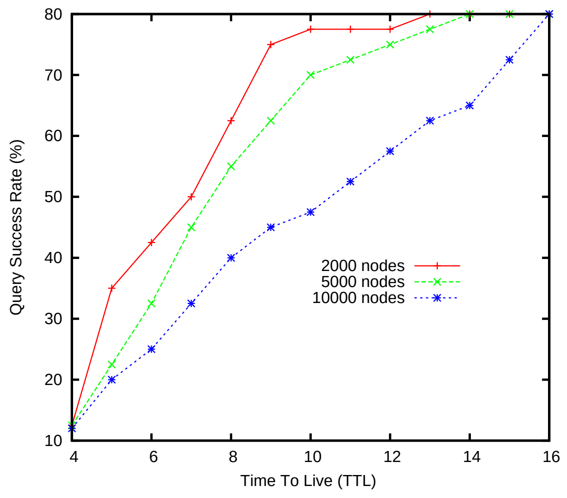

Figure 5: Query Success Rate vs TTL

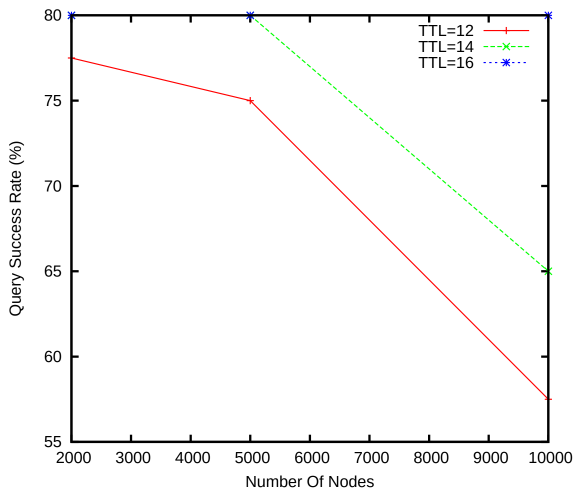

Figure 6: Query Success Rate vs Network Size

References

[1] Eytan Adar and Bernardo A. Huberman, "Free Riding on Gnutella," *Technical report, Xerox PARC*, August 10, 2000.

[2] Demetris Zeinalipour-Yazti, Theodoros Folias. "A Quantitative Analysis of the Gnutella Network Traffic," *Course project for "Advanced Topics in Networks"*, Department of Computer Science, University of California, April-2002.

[3] A. Dhamdhere, "Determining Characteristics of the Gnutella Network," *Techincal report, College of Computing, Georgia Tech*, September 24-2002.

[4] B. Hudzia, M-T. Kechadi, A. Ottewill, "Treep: A Tree Based P2P Network Architecture," *International Workshop on Algorithms, Models and tools for parallel computing on heterogeneous networks*, Boston, Massachusetts, USA, September 27-30, 2005.

[5] M. Srivatsa, B. Gedik and L. Liu, "Scaling Unstructured Peer-to-Peer Networks With Multi-Tier Capacity-Aware Overlay Topologies," *In proceedings of the Tenth International Conference on Parallel and Distributed Systems*, Washington, DC, USA, 2004.

[6] S.Min, Dongsub Cho, "Super Peer Selection Baased Framework Using Dynamic Capacity and Similarity," *ISCIS 2006*, LNCS 4263, pp. 803 – 812, 2006.

[7] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *in Proceedings of Multimedia Computing and Networking*, 2002.

[8] W.W. Terpstra, J. Kangasharju, C. Leng, and A.P. Buchmann, "Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search," *SIGCOMM Comput. Commun. Rev.*, 2007.

[9] Gnutella Network Snapshot Homepage: *http://www.Gnutellameter.com*