International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012

DOI: 10.7321/jscse.v2.n7.5

CLTChord: Improving lookup at the Chord protocol using cache location table

[1]Jaber Karimpour, [2]Majid Moghaddam, [3]Ali A. Noroozi
[*1,]*Department of Computer Science, University of Tabriz, Tabriz, Iran*
[2,] *Department of Computer Engineering, Zanjan Branch, Islamic Azad University, Zanjan, Iran*
[3,] *Department of Computer Science, University of Tabriz, Tabriz, Iran*
E-mail: [1]*karimpour@tabrizu.ac.ir*, [2]majid6565@yahoo.com,[3] *aliasghar.noroozi@gmail.com*

Abstract. **Peer-to-peer networks have emerged as a common method for sharing large amounts of data. The main challenge in these networks is efficiently locating information distributed across the hosts/peers of the network by decentralized approach. In this paper, first we refer to the Chord protocol that is a distributed lookup service, and then propose a new method, called CLTChord, to optimize this protocol. In this new method, in addition to using the local finger tables for routing requests, the cache location tables are used, in which each node stores nodes which are at its close geographical range. When a node receives a new request, it first checks its cache location table and if a desired response is not received, the algorithm continues to work like the basic Chord protocol. Our simulation shows that this optimization improves the parameters of the hop count, the lookup latencies and number of the sent packets; In this paper, hop count is the distance between the source node which initiates the lookup and the target node which has the desired value; and latency is the duration of time needed for resolving file lookups from the time when it was initiated until it was responded to (measured in milliseconds).**

Keywords: *Communication networks, Peer-to-peer, Chord, Finger table*

**Corresponding address:*
Jaber Karimpour,
Department of Computer Science, University of Tabriz,
Tabriz, Iran.
E-mail: karimpour@tabrizu.ac.ir

1. Introduction

In recent years, P2P systems have emerged as a common method for sharing large amounts of data. P2P Overlay networks are distributed systems that have no central organization or centralized control. In these networks, nodes have the same roles and capabilities, and Information and services are directly exchanged with each other [1].

The key to the usability of the P2P system and the main challenge to data-sharing is providing techniques for efficient lookup and retrieval of data. The best techniques for a given system depend on the application requirements [2].

Generally, P2P networks are categorized into two categories: structured and unstructured networks. In Structured networks, such as CAN [3], Tapestry [4], Pastry [5] and Chord [6], each item of information is stored on a specific node. Therefore, content is found after scanning of several nodes, if it exists. In these networks, data placement and topology is controlled within the network. However, in unstructured networks, such as Gnutella [7], Freenet [8], KaZaA [9] and BitTorrent [10] items are

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

distributed randomly and unknown between nodes. The main advantage of structured networks over unstructured networks is high scalability of these networks.

To implement a structured P2P network, two main mechanisms are needed: 1) a mechanism to allocate each item to one node in the network, and 2) a mechanism to lookup and retrieve desired item from the node containing the item based on its key.

For implementing these two mechanisms in structured networks based on a distributed hash table (DHT), first, name of each node and each key is hashed separately to a unique identifier by the distributed hash table using a consistent hash algorithm like SHA-1 [11], and then this identifier is considered as a special ID for the node and the key. With respect to the used hash space, each node is considered responsible for storing all the contents that their ID is close to the node ID. To find the key content, each node sends key content request message to the next appropriate node using a small hash table, which keeps limited information compared to the total number of nodes in the network. This operation scans a limited number of nodes, until the request message reaches to the node responsible for the key content and the requested content is retrieved from the node. Structured networks, as introduced above, are based on a distributed hash table [12, 13].

1.1. The problem

Chord is one of the most popular P2P systems based on distributed hash table. This protocol is published in 2001 by Stoica et al, and is one of the first protocols that use a distributed hash table.

The main features of this protocol could be scalability, load balancing, good performance, simplicity, etc. Despite these favorable characteristics, the Chord protocol has some disadvantages; one of them is that nodes in the network do not consider physical location of nodes that are in their geographic range; i.e., each node directs its lookup requests based on routing table (or finger table) rather than first searching for the desired data in the nodes that are in its geographic range. This increases the lookup time and network bandwidth is wasted. To fix these drawbacks, we add a new functionality to the Chord protocol that considers physical location of nodes present in the network. This new functionality doesn't cause any disruption to the search procedure.

1.2. Paper outline

The rest of this paper is organized as follow. In Section 2, we briefly describe the Chord protocol and the procedure of searching and finding the desired data. In Section 3, some work on the Chord protocol presented that is related to considering the physical location of nodes. Then, in Section 4, the proposed method to improve this protocol is presented; in this method, we add the new functionality to the protocol that uses the cache location table. In Section 5, we illustrate simulation results by using of OMNeT++ simulator. Finally, in Section 6, the conclusions of this paper are described.

2. Chord protocol

As previously noted, a major problem associated to P2P applications is efficiently locating nodes that have stored particular data item. This paper discusses the Chord, one of the most popular P2P systems, that is a distributed/decentralized protocol/service. The protocol stores pairs of (key, value) in network nodes and manages the problem of locating node; the key is data key and value is information contents corresponding to the key, which is stored in one of peers of the Chord network. The key is assigned through the consistent hashing to one of network peers. The peer/node identifier could be generated by hashing its IP address and the key identifier could be generated by hashing the key. This identifier is unique [6].

The Chord system is a distributed lookup service that is based on Chord protocol. This system supports five types of operations: node join and leave the system, insert and update and lookup of (key,

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

value) pairs. All the main operations are provided by the Chord protocol. In Table 1, these operations are explained.

The Chord protocol supports only one operation: A given key map to a node. As we mentioned, Chord uses a type of consistent hashing to assign keys to Chord nodes. Consistent hashing tends to load balancing, because each node receive roughly the same number of keys and results in moving roughly a few number of keys, when nodes join and leave the system. This protocol gets an m bits identifier as input and returns the node, which have stored corresponding key value.

Locating the data can easily be done; assigning single key to each data item and storing (key, value) pair at a node on which the key is mapped. This protocol, when nodes join or leave the system, it efficiency organizes itself, even if the system is constantly changing, it can respond to requests.

Previous works have done related to the consistent hashing assume the nodes know other nodes of the system, which prevents scalability of consistent hashing to large numbers of nodes.

In contrast, each Chord node requires routing information a few other nodes. Because the routing table is distributed, a node resolves the hash function by communicating with a few other nodes.

In a steady state, of a system with N nodes, each node in the network keep information of the O(logN) other nodes and performs all its lookup through O(logN) messages to other nodes. The Chord maintains its routing information when nodes join and leave the system; it can be done with up to O(log2N) messages to other nodes.

Table 1. Operations of the Chord network [6]

Operation	Comment
join(n)	The node adds itself to the Chord network that node n is part of this network. This operation returns success or failure.
Leave ()	A node leaves the Chord network. No return value.
lookup (key)	Gets and returns the value corresponding with the key.
Update (key, new_value)	Inserts the key and itself the new value at z nodes. In stable conditions, exactly z nodes contain the (key, new_value).
Insert (key, value)	Inserts the key and itself the new value at z nodes. In stable conditions, exactly z nodes contain the (key, value).

Chord protocol has a simple workflow; it routes Keys through a sequence of O(logN) other nodes toward the destination. Chord node for efficient routing, needs the O(logN) information of other nodes, but when information isn't updated, routing efficiency reduce. Only part of the information of each node is required which be correct; then, Chord can properly route requests to the destination. Hence, the routing is done slowly [6].

2.1. Consistent hashing

The consistent hash function by using a hash function (e.g., SHA-1) assigns each node and key an m bits identifier. The node ID is chosen from hashing of the IP address of the node, but the ID key is generated from hashing of the key itself. The ID Length (i.e., m) must be large enough that the probability hashing of two nodes or two keys to the same ID be venial. The consistent hash assigns the keys to the nodes so that the identifiers are sorted in an identifier circle modulo 2m. Key k is assigned the first node, which the node identifier is equal to or greater than the key ID. This node is called the successor node of key k and is shown with a symbol of successor(k). If the identifiers are displayed as a loop from 0 to 2m-1, then successor (k) is the first node in the clockwise direction from k.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

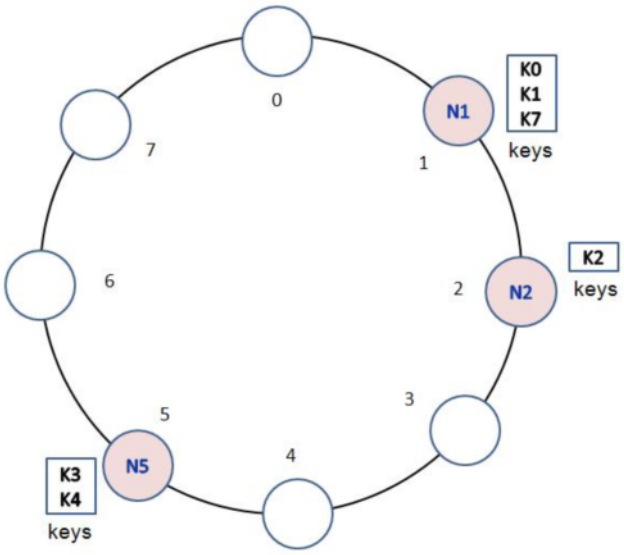

Figure 1. An identifier circle consisting of three nodes storing six keys [6]

To understand this better, see Figure 1. The figure shows one ring ID with m = 3. The ring ID has three nodes (N1, N2, N5) and stores six keys (k0, k1, k2, k3, k4, k7). The successor of IDs 0, 1 and 7, in clockwise direction, is the node 1, so the key k0, k1, and k7 are stored in the node N1. Similarly, key k2 in node N2 and keys k3, k4 are stored in the node N5.

```
// ask node n to find the successor of keyID
n.find_successor(keyID)
if (keyID ∈ (n, n.successor))
    return n.successor;
else
    n_nextHop = closest_preceding_node(keyID);
    return n_nextHop.find_successor(keyID);
// search the local table for the highest predecessor of keyID
n.closest_preceding_node(keyID)
  for i = logN downto 1
     if (finger[i] ∈ (n, keyID))
        return finger[i];
  return n;
```

Figure 2. Pseudocode for key lookup using the finger table [6]

The consistent hashing is designed in a way that causes nodes join and leave the network without interruption. To maintain the consistent hashing and mapping when a node (n) joins the network, certain keys that were previously assigned to n's successor now become assigned to node n. In the above example, if the node with ID 7 joins the network, this node will capture the key with ID 7 (i.e., k7) of the node with ID 1. When node n leaves the network, all keys assigned to it are reassigned the successor n.

2.2. Key lookup method

As also mentioned above, we consider m as the number of bits of the node ID / key and consider N the total number of network nodes. Each node n has the routing table with up to logN entries; the table

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

is called finger table. The ith table entry for node n, contains the identity of the first node (s) that is after an identifier interval of at least 2i-1 to n on the identifier circle, means, s = successor(n +2i-1) ($1 \leq i \leq m$) and all calculations is in modulo 2m. We call node s the ith finger of node n and show with n.finger[i]. Chord's finger table entry includes the Chord identifier and the IP address (and port number) of the corresponding node. Note that the first finger of node n is immediate successor of node n in the ring; the first finger is called the successor; the node, which is before the identifier n in the ring, is called predecessor.

Figure 2 shows the pseudo code of find_successor(keyID) procedure. If the keyID falls between node n and node n's successor, return n.successor command line is executed and node n returns its successor; Otherwise, node n lookups its finger table for the node nnextHop with executing nnextHop=closest_preceding_node (keyID) command line. Note that nnextHop has the largest ID that precedes keyID. Then, node n calls find_successor(keyID) in the node nnextHop with executing return nnextHop.find_successor(keyID) command line. The reason of choosing nnextHop is that its closer to the keyID. In other words, the nnextHop will know more about the identifier ring around of keyID area [6].

3. Related work

As stated in the previous section, the basic operation in Chord protocol is mapping given key to one of the nodes in the network. While doing this, the Chord protocol ignores the physical network topology. In other words, each node first sends its requests to geographically distant nodes, rather than sending to geographically close nodes. This disadvantage increases lookup time, and network bandwidth is wasted. In recent years, there has been a large amount of work done to fix this problem, which we give a brief introduction of them.

PChord [14] protocol considers the physical proximity of nodes in the network by using the proximity list. Each node has a proximity list, which contains a list of neighboring nodes that are in its close geographical range. To route the lookup requests, each node uses both the proximity list and the finger table. In this protocol, some finger table entries get replaced with appropriate entries of proximity list; therefore, routing is performed among the nodes physically closer. The major problem with this approach is its inefficiency in the high rate of entry and exit of the nodes, which increases replacement overhead and we may not achieve our desired result.

BChord [15] protocol exploits two-sided lookups to reach the target node. In this protocol, each node, in addition to maintaining the finger table, which contains the successors of the node, stores the predecessor table. Generally, kind of lookup locality for routing the requests is used. In addition, BChord lookups benefit from two greedy strategies: obtaining physically shorter path and getting overlay path. Based on the achieved paths, routing requests is done. Disadvantage of this protocol is its incompatibility with these two strategies.

In Chord6 [16], hierarchical structure of IPv6 addresses is used. The hash function generates identifiers of the nodes in a way that all nodes in the same domain should be having identifiers close to one another. Thus, physical network topology is applied in a P2P network. The problem is that due to increasing number of Internet domains and low-average P2P network nodes that are physically in the same range, the efficiency of this protocol isn't optimal. Currently the Internet is based on IPv4. As a result, this protocol isn't usable.

4. The proposed approach

As stated in sections 1 and 2, the Chord protocol doesn't consider the geographical location of nodes. In this section, we intend to add new functionality to the Chord protocol in order to solve this problem as much as possible. This functionality is called cache location table.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

In the Chord protocol, each node while joining the system, communicates with a server node. The server keeps current status of the network including number of nodes in the network, port and IP address of all peers, and order of the nodes in the ring. If a node wants to join the network, it sends a join request to the server. Regarding its information of the network and the identifier of the new node, the server sends a message to the new node. This message contains the IP address of the node that the new node must precede in the Chord ring.

Our proposed solution is to modify the server to store physical location of the nodes in the network, in addition to preserving previously stated information. It must be done while joining the node to the network. For this purpose, each new node must attach its own geographical range to the join request and send to the server. As soon as receiving this request message, the server must send a response message to the new node. The message contains the IP address of its successor node and one of the nodes in the same geographical range. The new node, after full joining to the network, communicates with the node that is in its geographical range to fill the cache location table. The table is in the cache memory. We call the proposed protocol CLTChord, because we added the new functionality of cache location table to the basic Chord protocol.

Because network is continuously changing, the nodes in each geographical range must communicate with one another in specified intervals, or in other words, the nodes ping each other. Some nodes join the network and some others leave it. Therefore, the cache location tables should be updated to route the requests properly. The nodes in the cache location table must have a certain order. They should be arranged based on goodness, so that node begins lookup requests from the first better node and if it doesn't receive its desired response, send the request to the second better node and so on. Thus, some criteria should be defined for goodness, like lifetime of a node, desired number of responses sent to other nodes, and time delay between the nodes.

Figure 3, shows the pseudo code of how to do a lookup on the CLTChord protocol.

```
// search the cache location table for the keyID
n. Search_CLTable(keyID)
    for i = 1 to p
            if (keyID exsits at CLTfinger(i).node){
                result= CLTfinger(i).node;
                return result; }
    return n;
     if (Search_CLTable(keyID) = fail)
        n.find_successor(keyID);

// ask node n to find the successor of keyID
n.find_successor(keyID)
if (keyID ∈ (n, n.successor])
    return n.successor;
else
    n_nextHop = closest_preceding_node(keyID);
    return n_nextHop.find_successor(keyID);

// search the local table for the highest predecessor of keyID
n.closest_preceding_node(keyID)
  for i = logN downto 1
     if (finger[i] ∈ (n, keyID))
        return finger[i];
  return n;
```

Figure 3. Pseudocode of key lookup using the cache location table

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

5. Simulation results

In this section, we discuss the simulation of the proposed protocol and compare the results with the Chord protocol. In the simulation process, we examined these parameters: lookup hop count, look up latency and number of the sent packets. As already mentioned, the simulation was performed by the OMNeT++ simulator. Simulation started with a network consisting of 400 nodes and repeated until the number of the nodes reached 2000. The number of the nodes increased by 400 in each repeatition. Each repetition took 3 hours.

The values required for the simulation is presented in table 2. Values obtained from each of the studied parameters are shown in tables 3 and 4.

The zero values in tables 3 and 4 is caused by the satisfaction of lookups in its node and it hasn't been sent to other nodes.

In table 5, we have shown the improvement percentage of the CLTChord protocol on the original Chord protocol.

As you can see in Figure 4, the average hop count of lookups in the original Chord protocol oscillates between 0 and 11 and after 2120 seconds the oscillation remains almost constant; also, the average hop count of lookups varies in the CLTChord protocol from 0 to 10.

As you can see in Figure 5, the average latency of lookups in the original Chord protocol oscillates between 0 and 6.951 milliseconds and after 2110 seconds the oscillation remains almost constant. Less oscillation of lookup latencies in the CLTChord protocol is because queries are sent to the nodes in close neighborhood (in terms of physical distance) and network instability has little effect on lookups latencies.

In Figure 6, the average number of sent packets varies in the original Chord protocol from 1665.03 to 3187.84 and the average number of sent packets varies in the CLTChord protocol from 1191.29 to 2158.86.

Table 2. Main simulation parameters

The successor list size	8
The average life time of each peer (sec)	10000s
The maximum hop count number	50
The retries number of each peer for joining to the network	2
The latency between the retries of each peer for joining to the network (sec)	10s
The duration of stabilizing procedure (sec)	120s

Table 3. The obtained values from simulation of the original Chord protocol

Chord	Max	Min	Mean
hop count	11	0	6.202
messages' latency (ms)	6.951	0	0.973
number of sent packets	3187.84	1665.03	2445.61

Table 4. The obtained values from simulation of the CLTChord protocol

CLTChord	Max	Min	Mean
hop count	10	0	5.200
messages' latency (ms)	1.956	0	0.434
number of sent packets	2158.86	1191.29	1681.19

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

Table 5. The improvement in the proposed CLTChord protocol

The studied parameters	Mean (CLTChord)	Mean (Chord)	The percentage of improvement
hop count	5.200	6.202	16.16
messages' latency (ms)	0.434	0.973	55.39
number of sent packets	1681.19	2445.61	68.74

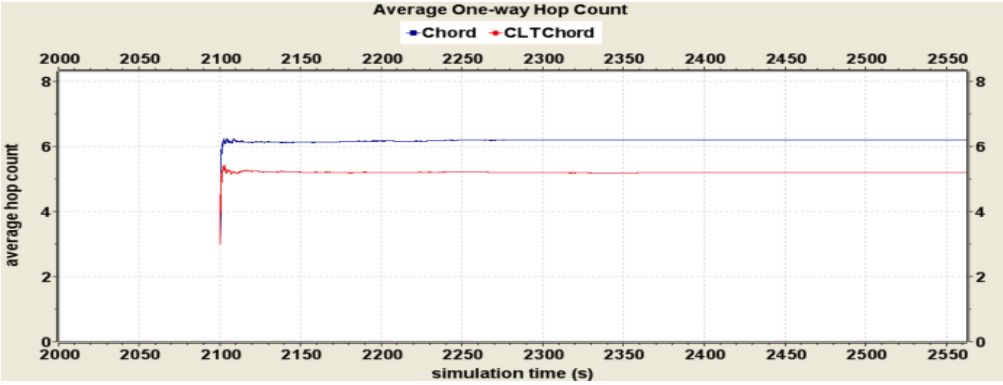

Figure 4. a plot of changes of lookup hop count parameter at reaching to the node target

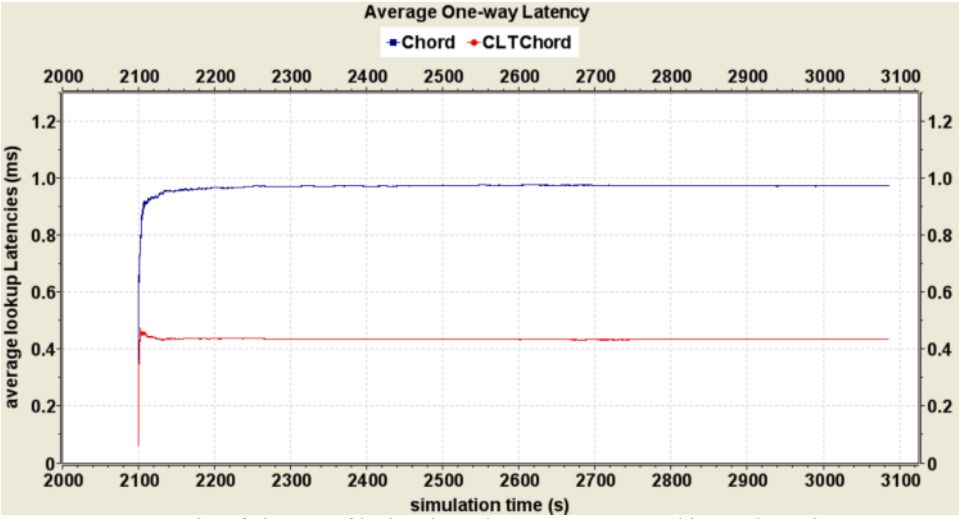

Figure 5. a plot of changes of lookup latencies parameter at reaching to the node target

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

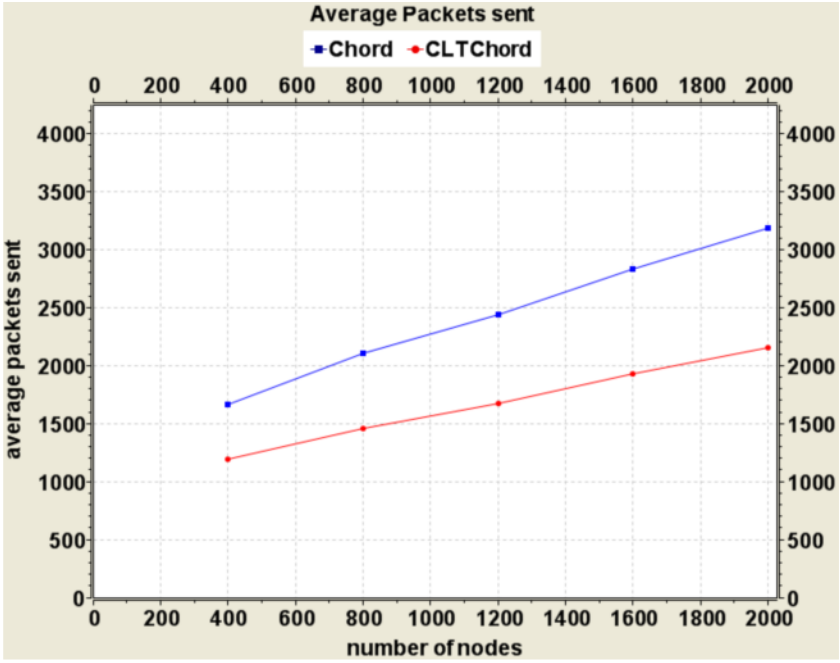

Figure 6. a plot of the number of the sent packets

6. Conclusion

With the rapid development and wide application of computer and telecommunications technology, many requirements have emerged in the field of communications that show disadvantages of the traditional client/server communication model. In this model, a single point of failure will lead to total system disability. Fortunately, the P2P model appeared. The P2P is a type of distributed system in which all the nodes are in a similar situation. The Nodes in the system share their resources and manage infrastructure together. The P2P systems prevent the bottleneck problem and have the benefits of scalability and load balancing while the traditional network model don't support any of them. Currently, P2P is a hot issue in network research.

The main aim of P2P networks is finding a node that has stored a particular data item by itself; given a key, the node which contains the key value will be found. The Chord protocol does this in an efficient way: In the steady state and the network of N nodes, each node holds routing information of O(logN) other nodes, and all queries are handled with O(logN) messages to other nodes. When nodes join and leave the network, updates the routing information of other nodes with O(log2N) messages.

In this paper, we illustrated the Chord protocol and mentioned its main disadvantage. We proposed the cache location table functionality to consider the physical network topology and simulated the proposed functionality in OMNeT++ simulator. Results of the simulation showed that the proposed procedure achieves a higher performance than the basic Chord protocol, in terms of hop count, latency, and number of sent packets.

References

[1] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE, 2002.

International Journal of
Soft Computing And Software Engineering (JSCSE)
e-ISSN: 2251-7545
Vol.2,No.7, 2012
Published online: July 25, 2012
DOI: 10.7321/jscse.v2.n7.5

[2] B. Yang, H. Garcia-Molina, “Improving search in peer-to-peer networks”, In Proc. Of the 22nd IEEE International Conference on Distributed Computing, pp.5-14, 2002.
[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network”, In Proc. ACM SIGCOMM, August 2001.
[4] B. Y. Zhao, et al., “Tapestry: A resilient global-scale overlay for service deployment”, IEEE JSAC, Jan. 2004.
[5] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems”, In Proc. of the 18th IFIP/ACMInternational Conference on Distributed Systems Platforms (Middleware 2001), November 2001.
[6] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup protocol for internet applications”, IEEE/ACM Trans. Netw. 11, pp. 17-32, Feb. 2003.
[7] M. Ripeanu, “Peer-to-peer architecture case study: Gnutella network”, In International Conference on Peer-to-Peer Computing, 2001.
[8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system”, in Proc. ICSI Workshop Design Issues in Anonymity and Unobservability, Berkeley, CA, June 2000.
[9] N. S. Good and A. Krekelberg, “Usability and privacy: a study of Kazaa P2P file-sharing”, In Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '03), ACM, New York, NY, USA, pp. 137-144, 2003.
[10] J. Pouwelse, P. Garbacki, D. Epema, H. Sips, M. Castro, R. van Renesse, "The Bittorrent P2P File-Sharing System: Measurements and Analysis", Peer-to-Peer Systems IV, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005.
[11] D. R. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web,” in Proc. 29th Annu. ACM Symp. Theory of Computing, El Paso, TX, pp. 654–663, May 1997.
[12] J. Kim and R. Srikant, "Real-Time Peer-to-Peer Streaming Over Multiple Random Hamilton Cycles" in Proc. of the Information Theory and Applications Workshop, San Diego, CA, Feb. 2012.
[13] J. Kim and R. Srikant, "Achieving the Maximum P2P Streaming Rate Using a Small Number of Trees" IEEE ICCCN 2011.
[14] F. Hong, M. Li, J. Yu, and Y. Wang, “Pchord: Improvement on chord to achieve better routing efficiency by exploiting proximity”, in Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW’05), June 2005.
[15] Jie Wang, Zhijun Yu, “A New Variation of Chord with Novel Improvement on Lookup Locality”, In Proceedings of GCA, pp.18-24, 2006.
[16] J. Xiong, Y. Zhang, P. Hong, and J. Li, “Chord6: IPv6 based topology-aware Chord”, In Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005), Aug 2005.