2009 International Conference on Computational Science and Engineering

HyperCircle: An Efficient Broadcast Protocol for Super-Peer P2P Networks

Feiyu Lin, Christopher Henricsson, Syed Muhammad Abbas,Kurt Sandkuhl
Jönköping University
Gjuterigatan 5,
551 11 Jönköping, Sweden
{feiyu.lin, df06hech, df06AbSM, kurt.sandkuhl}@jth.hj.se

Abstract

The challenge of peer-to-peer (P2P) applications is how to efficiently broadcast in large-scale P2P networks. This paper presents the 8-point HyperCircle protocol for efficient broadcasting and routing of messages in P2P networks. The HyperCircle topology consists of n-dimensional 8-point circles where each point in each dimension can consist of an 8-point circle. A HyperCircle node requires $2*\log_8 8^k$ steps to spread the message to all nodes. The construction and maintenance of the topology including the broadcast algorithm with HyperCircle are introduced. A simulation framework with the HyperCircle protocol is implemented based on OverSim. Initial experimental results comparing protocols like Chord, Kademlia and HyperCircle are reported. HyperCircle provides the most efficient broadcasting messages.

1. Introduction

Peer-to-peer (P2P) networks have been well developed and widely implemented in distributed applications on the Internet. For example, well known file sharing P2P networks, like Napster [9], or P2P for the semantic web, like Edutella [2], confirm the suitability of P2P for retrieval of information and discovery of web services.

A major challenge of P2P networks is how to achieve scalability of large P2P networks with millions of nodes. Pure P2P networks, which broadcast information through the whole network, have efficiency problems for large numbers of nodes. Structure P2P networks based on a distributed hash table (DHT) or based on super peers were proposed for solving the scalability problem with large numbers of nodes. For example, Chord uses a variant of consistent hashing to allocate keys to nodes and lookup the node by key [18]. Super peer networks introduce super peers as a middle level into the network. Super peers provide and maintain an index about content and services offered by sub-ordinate peers.

However, super peer networks put additional tasks on the super peer nodes and have to be carefully constructed to work as desired. The construction and maintenance of the topology of the network is of specific interest, i.e. the policy for communication among the super peers and with the sub-ordinate nodes and how new nodes are added to the topology or nodes dropping out of the network are removed from the topology.

This paper presents the 8-point HyperCircle protocol for efficient broadcasting and routing the messages in P2P networks. Inspired by work of Schlosser et. al on HyperCubes, 8-point HyperCircle protocol was already described in our previous work [5]. The HyperCircle topology consists of n-dimensional 8-point circles where each point in each dimension can consist of an 8-point circle. A HyperCircle node requires $2 * k$ steps to spread the message to all nodes (8^k). Due to the equal distance of all nodes to the center of the circle, every node on the circle can potentially be the initiator of a broadcast, i.e. they have identical power. This structure allows for advantages as compared to HyperCube with respect to the required number of hops for broadcasting messages to all nodes in the topology.

The rest of this paper is structured as follows. The construction and maintenance of the topology including the broadcast algorithm with HyperCircle are introduced in section 2. Section 3 presents our implementation for the HyperCircle protocol based on OverSim (the Overlay Simulation Framework) [1] simulation framework. Initial experimental results comparing protocols like Chord, Kademlia[6] and HyperCircle are reported in section 4. Section 5 summarizes the work and gives an outlook on future work.

2. The Hypercircle Topology

The HyperCircle topology aims to be symmetric. Each node in the network should have the same power and task. But the topology should be redundant, the failure or leaving node will not hamper the broadcast. For broadcasting mes-

978-0-7695-3823-5/09 $26.00 © 2009 IEEE
DOI 10.1109/CSE.2009.247

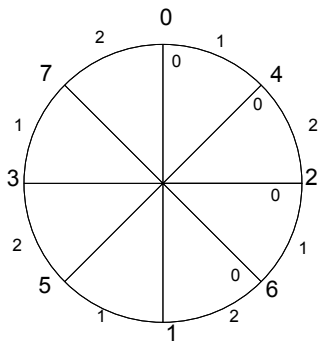

Figure 1. 8-point Circle

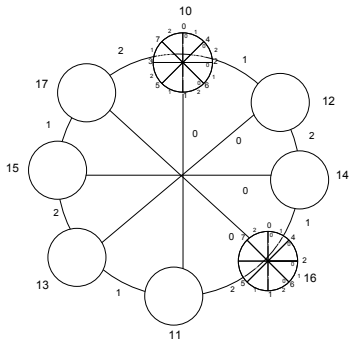

Figure 2. 64-point Circle

sages, each node can be the originator of a broadcast and nodes get a message only once.

This section describes how new nodes join the system, how to manage leaving nodes, and how to broadcast messages to all the nodes in the topology. These aspects partly were already described in our previous work [5], but are required to understand section 3.

2.1 HyperCircle Topology Construction

The HyperCircle topology consists of n-dimensional 8-point circles where each point in each dimension can consist of an 8-point circle. For example, Figure 1 shows the basic element 8-point circle of HyperCircle. Figure 2 shows a two dimension 64-point circle which consists of 8 8-point circle. The construction of the HyperCircle topology is based on the following rules:

1. Every circle has at most eight nodes.
2. Each node has at most three relationships (denoted neighbor-0, neighbor-1 and neighbor-2) with the other nodes in the same circle.
3. Each node has a neighbor-0. The neighbor-0 is the 180 degree neighbor. This relationship will not change unless neighbor-0 or the node leave the topology.

We will illustrate how to construct a two dimension 8-point circle (64-point circle) by example as follows:

1. Peer 0 is active and opens a new circle.
2. Peer 1 connects to peer 0 to join. Peer 1 is assigned as peer 0's neighbor-0 (see Figure 3a).
3. Peer 2 joins by contacting either peer 0 or 1. A virtual neighbor-0 called 2' is created since both peer 0 and peer 1 already have a neighbor-0 relationship. If peer 0 is contacted, peer 0 becomes peer 2 neighbor-1 and peer 2' neighbor-2. Peer 1 is peer 2' neighbor-1. Peer 1 neighbor-2 is peer 2. Now every node has the link set 0, 1, 2 with the other nodes (see Figure 3b). Peer 2 and peer 1 are notified that 2' is the vacant point. If peer 1 is contacted, peer 1 becomes the neighbor-1 of peer 2.
4. If peer 3 wants to join, peer 0, 1 or 2 can be contacted. It will replace the simulative peer 2' and inherit peer 2' link set (see Figure 3c).
5. If peer 4 wants to join and for example contacts peer 0, a simulative peer 4' is added to be neighbor-0. The balance of the circle is destroyed. Peer 0 arranges its neighbor-1 to 4, peer 2 is peer 4 neighbor-2. Peer 4' neighbor-1 is 3 and neighbor-2 is 1 (see Figure 3d). Every node knows that there is a vacant point.
6. Peer 6 and 7 can contact one node in the circle to join in the same way. Finally, 8-point circle can be constructed as Figure 1.
7. If more peers want to join, e.g. peer 8 contacts peer 7 to join, peer 7 will open a new circle 11 and mark itself as circle 10. All peers in circle 10 are notified that a new circle 11 was created being the neighbor-0 in that dimension and that peer 8 is representing this new circle. Circle 11 is circle 10 neighbor-0. If circle 11 is full, the circle 12 will be opened. Finally, the 64-point circle will be constructed as shown in Figure 2.

2.2 Topology Maintenance

For (8^k) nodes HyperCircle topology, each node need store $3 * \log_8 8^k$ neighbor nodes information. If one peer leaves the topology, it's neighbors will take over its position. For example, it's neighbor-0 peer will occupy it's position first. But if it's neighbor-0 leaves also, then neighbor-1 will occupy its position. If one dimension disappears, all the peers in the circles are notified that the dimension does not exist any more.

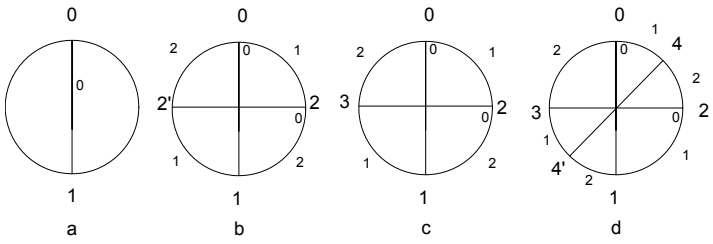

Figure 3. Topology Construction

2.3 Broadcast Algorithm

The broadcast algorithm works as follows :

1. The initiating node will send the message in the same circle and - if existing - the next dimension neighbor nodes marked 0, 1 and 2.

2. The receiving nodes will forward to their own circle and the next dimension (if they have) two neighbor nodes marked 0, 2 (in case they received the message from neighbor-1) or 0, 1 (in case they received it from neighbor-2). The receiving nodes will stop forwarding the message if:

   - They received the message from their neighbor-0.
   - The layer circle has 1 or 2 or 3 or 4 nodes, the second step will stop forward. If the circle has 5 or 6 or 7 or 8 nodes, the third step will stop forward.

3. To remove redundancy in the second step, the receiving nodes will just forward to neighbor-0 if the same layer circle has 5 or 6 nodes.

Figure 4 shows an example where peer 0 (see Figure 1) initiates the broadcast according the above broadcast algorithm. Figure 5 shows for peer 2 in circle 10 (see Figure 2) how to broadcast a message to the peers in circle 16. We can see that if one dimension is added two steps more are needed. HyperCircle needs $2 * \log_8 8^k$ steps to spread the message to all nodes (8^k). Since the circle has the characteristic that all points have the same distance to the center of the circle, every node on the circle can potentially be the initiator of a broadcast.

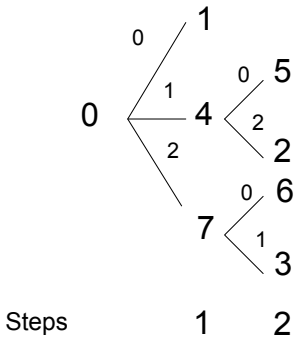

Figure 4. 8-point Circle Broadcast

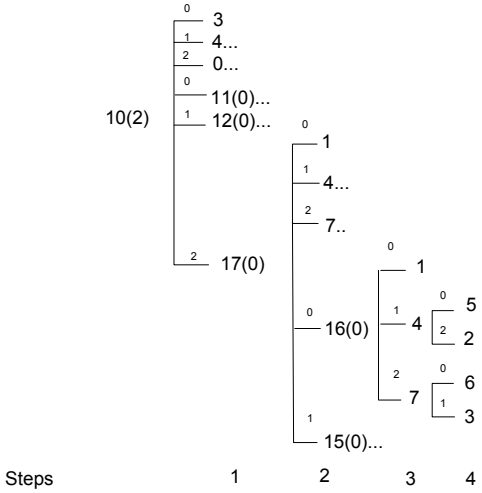

Figure 5. 64-point Circle Broadcast

3. Implementation Simulation Framework

This section discusses the OverSim architecture and our simulation framework architecture based on the simulator OverSim.

3.1 OverSim Simulator

A comprehensive survey of P2P network simulators can be found in [7][8]. The most frequently used simulators, like P2PSim[14], PeerSim[15], Query-Cycle Simulator[17], Narses[10], Neurogrid[11], GPS[3], Overlay Weaver[13], PlanetSim[16], etc. are compared according the criteria: simulator architecture, usability, scalability, statistics, underlying network simulation. Limitations of the simulators, such as poor scalability, little or no documentation, are discussed in [7][8]. Based on the above comparison and criteria like scalability and documentation, OverSim was selected as framework simulator for our experiments.

OverSim is an open-source overlay network simulation framework for the OMNeT++ [12] simulation environment and available for Linux/Unix. OverSim consists of the following modules [1]:

- Highly modular OMNeT++. Discrete event simulation (DES) is used to simulate exchange and processing of network messages.
- Underlying network model. Three network models are implemented: Simple, SingleHost and INET. There is an UDP/IP interface between the underlying network model and overlay protocols.
- Overlay protocols. Several structured P2P protocols like Chord, Kademlia , etc., and unstructured P2P protocols like GIA are implemented. There is a key-based routing (KBR) interface between overlay protocols and application.
- Application. Key-based routing test is implemented. Based on the parameter values, messages are sent to random overlay keys or nodeIDs during the testing time. Statistical data such as delivery ratio, packet hop count and delay time are collected.

3.2 Implementation

The basic intentions with our simulation framework are to evaluate the performance of the HyperCircle protocol to explore possible directions for the future development of the approach. Figure 6 shows the architecture of HyperCircle framework based on OverSim as follows [4]:

- OverSim Underlay. Three network model are implemented like Simple, SingleHost and INET in OverSim. These are all transparent to the overlay using a consistent UDP interface.
- OverSim BaseOverlay. OverSim provides basic common functions for the overlay protocols, such as bootstrapping support and message handling. A few methods in the layer are overriding OverSim's methods because of the generic lookup function is not suitable for our broadcast algorithm. The OverSim API is used to connect the overlay protocols.
- HyperCircle Overlay Protocol. This layer consists of topology and messaging implementation. The topology implementation handles the arrivals and leaving nodes based on HyperCircle topology (see Section 2 and 2.2). For the messaging implementation, a few methods are overriding OverSim's methods to be able to send messages to more than one node at same time based on HyperCircle broadcasting algorithm (see Section 2.3).
- Application. Using the KBR interface, key-based routing test implemented in OverSim's application layer can be used to test HyperCircle overlay protocols.

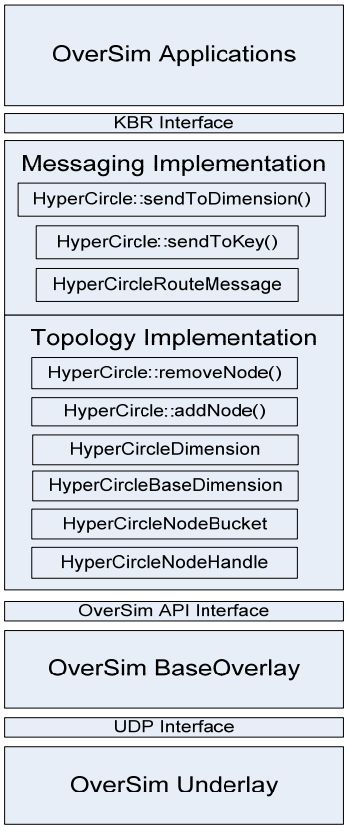

Figure 6. Architecture of HyperCircle Framework Based on OverSim (source: [4])

The most important HyperCircle protocol topology is the three neighbor relationships. HyperCircleBucket stores the first-level nodes. HyperCircleBaseDimension stores

the second-level nodes and is a vector containing HyperCircleNodeBuckets. HyperCircleDimension stores the second-level and upwards nodes and is a vector containing HyperCircleBaseDimensions or HyperCircleDimensions. Pseudo-code of construction the first-level circle (see Figure 3 b, c and d) is showed as follows.

```
oldNeighbor = contactNode->neighbor0;

contactNode->neighbor1 =newNode;
newNode->neighbor1 = contactNode;
newNode->neighbor2 = oldNeighbor;

virt->circle = bucketNo;
vacantPoint.push_back(virt);

newNode->neighbor0 =virt;
virt->neighbor0 =newNode;
if([bucketno]->size() == 2) {
   contactNode->neighbor0->neighbor1
    =virt;
   contactNode->neighbor0->neighbor2
    = newNode;
   n->neighbor2 =contactNode->neighbor0;
   virt->neighbor1
    = contactNode->neighbor0;
   virt->neighbor2 = contactNode;
   contactNode->neighbor2 = virt;
   [bucketno]->push_back(virt); }
else if ([bucketno]->size() == 4) {
    virt->neighbor1
     =contactNode->neighbor2;
    virt->neighbor2
     = contactNode->neighbor0;
    contactNode->neighbor2->neighbor1
     = virt;
    contactNode->neighbor2->neighbor0
    ->neighbor2 = newNode;
    contactNode->neighbor2->neighbor0
    ->neighbor1=contactNode->neighbor0;
    virt->neighbor2->neighbor1
     = newNode->neighbor2;
    virt->neighbor2->neighbor2 = virt;
    [bucketno]->push_back(virt); }
else if ([bucketno]->size() == 6) {
    virt->neighbor1
     = contactNode->neighbor0;
    ...    }
```

4. Evaluation

In this section, we evaluate the 8-point HyperCircle protocol by simulation. We would like to compare 8-point HyperCircle protocol to the other implemented structure P2P protocols in OverSim like Chord, Kademlia. Chord protocol uses consistent hashing to allocate keys to nodes. Message is spread by lookups key/node pairs. Kademlia protocol uses key/node pairs lookup system, but it uses novel XOR metric for distance between nodes in the key spaces.

We report on some initial experimental results comparing protocols like Chord, Kademlia and HyperCircle. The collected statistical data such as delivery ratio, packet hop count and delay time are presented and compared.

4.1 Evaluation Setting Up

We have run our test cases on a PC with 1.5Gb memory and two Intel(R) Pentium(R) 4 CPU 2.80 GHz processors. During the simulation, the nodes were joining and leaving the network randomly. The HyperCircle topology is constructed as described in section 2. The test messages are sent from random nodes and received by the destination nodes following the HyperCircle broadcasting algorithm (see section 2.3).

In the file *omnetpp.ini*, a set of configuration options can be set. For example, the *network* parameter specifies the underlay network model, where in our case SimpleNetwork is chosen. The *overlayType* parameter specifies the overlay protocol, e.g., HyperCircle protocol, or one of the stock protocols, like Chord , Kademlia, etc. The *tier1Type* specifies the application modules, where in our case *KBRTestAppModules* is used. The *targetOverlayTerminalNum* is the number of nodes that should be added to the network at the start of the simulation and 256 nodes are chosen. The *creationProbability* is the probability that a new node will be created during the simulation and 0.5 is set. The *removalProbability* is the probability that a node will be removed during the simulation and 0.8 is set. The *gracefulLeaveProbability* is the probability that a node will perform a graceful leave on exit. Our implementation does not distinguish between these possibilities. These three protocols (Chord [18], Kademlia [6] and HyperCircle) simulations have the same network type, network size, probability in the file *omnetpp.ini* as follows:

```
[Run 34] description = "HyperCircle"
network = SimpleNetwork
**.overlayType = "HyperCircleModules"
**.tier1Type = "KBRTestAppModules"
**.overlay.iterativeLookup=false
**.useCommonAPIforward = false
**.targetOverlayTerminalNum=256
**.creationProbability=0.5
**.migrationProbability=0.0
**.removalProbability=0.8
**.gracefulLeaveProbability=0.3

[Run 36] description = "Kademlia"
```

```
network = SimpleNetwork
**.overlayType = "KademliaModules"
**.tier1Type = "KBRTestAppModules"
**.overlay.iterativeLookup=true
**.targetOverlayTerminalNum=256
**.creationProbability=0.5
**.migrationProbability=0.0
**.removalProbability=0.8
**.gracefulLeaveProbability=0.3
**.tier*.kbrTestApp.lookupNodeIds=true
**.overlay.lookupRedundantNodes = 8
**.overlay.lookupMerge = true

[Run 37] description = "Chord"
network = SimpleNetwork
**.overlayType = "ChordModules"
**.tier1Type = "KBRTestAppModules"
**.targetOverlayTerminalNum=256
**.creationProbability=0.5
**.migrationProbability=0.0
**.removalProbability=0.8
**.gracefulLeaveProbability=0.3
```

Every simulation evaluation is running for around 15-minutes and 16-seconds simulation time.

4.2 Results

During the simulation, the statistical data such as delivery ratio (the ratio of sent messages successfully delivered to their destination), hop count (the average number of nodes that the messages traversed between the sending and receiving nodes) and delay time (the time between message sent and message received at the end-point) are collected. All the diagrams are plotted by the Linux tool *plove* from the vector file which was output from OverSim.

Figure 7, 8 and 9 show the delivery ratio of HyperCircle, Chord and Kademlia separately. For well-designed and well-implemented protocols, the delivery ratio will be dependent on simulation parameters relating to network instability when some nodes are leaving the network and can not forward the message. For 8^k nodes HyperCircle topology, the worst case of one leaving node (for example, node 4 or 7 in Figure 4, node 12 or 17 in Figure 5) will effect $3 * 8^{k-1}$ neighbor nodes for receiving messages. The HyperCircle's delivery ratio is effected when a node is leaving since our current implementation does not resend messages in these cases. The topology maintenance which take care of handing over the leaving nodes have not been implemented yet. The Chord implementation seems to be sensitive to this as well.

Figures 10, 11 and 12 show the hop count of HyperCircle, Chord and Kademlia separately. HyperCircle hop count is $2 * k$ to all nodes (8^k). Chord protocol needs

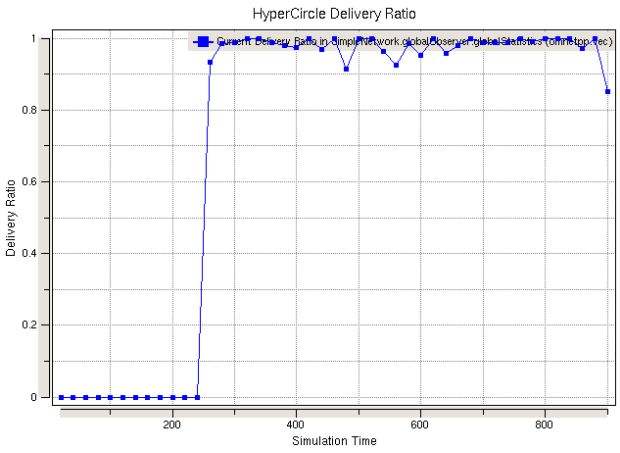

Figure 7. HyperCircle Delivery Ratio

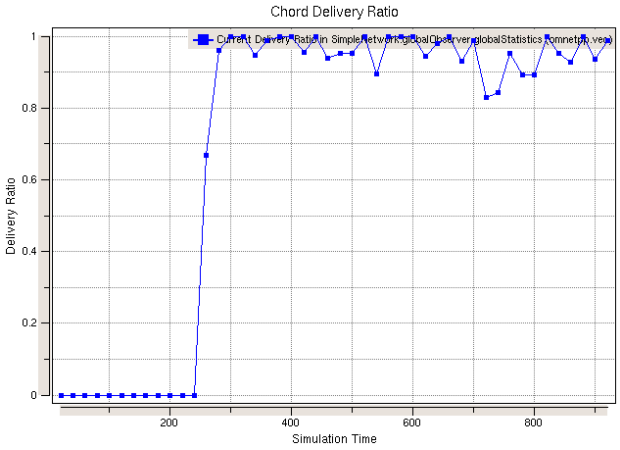

Figure 8. Chord Delivery Ratio

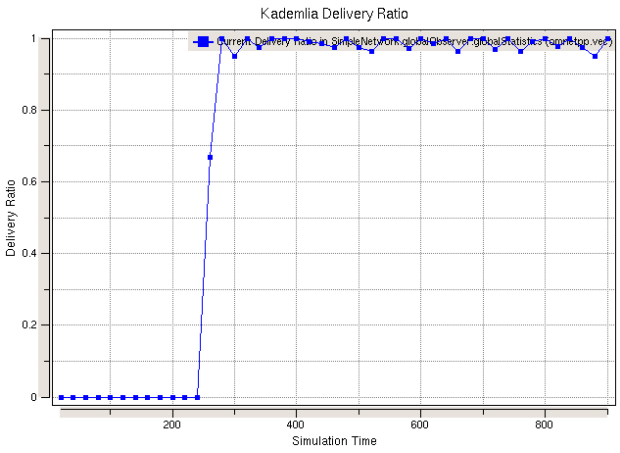

Figure 9. Kademlia Delivery Ratio

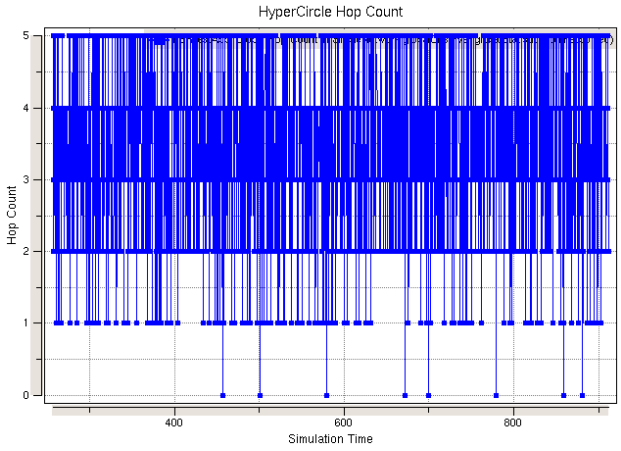

Figure 10. HyperCircle Hop Count

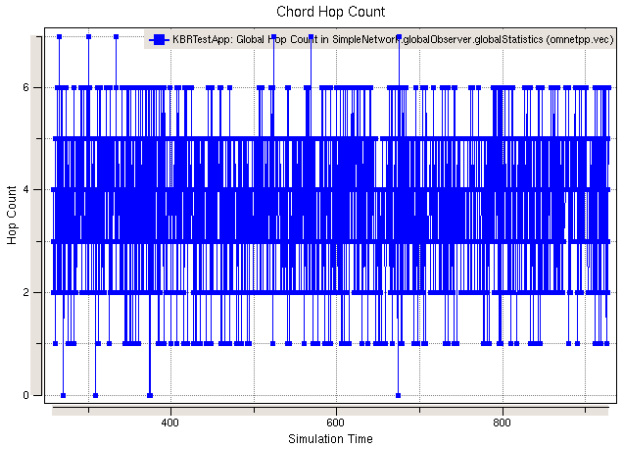

Figure 11. Chord Hop Count

$O(\log N)$ nodes for lookuping messages. Since Kademlia uses a lookup function optimized for finding the shortest path to a node, it has the smallest hop count $h - \log k$ (h is the depth, k is a system-wide replication parameter).

Figures 13, 14 and 15 show the delay time of HyperCircle, Chord and Kademlia separately. HyperCircle has the lowest average delay time, despite having a higher hop count than Kademlia.

From the above figures, compared to Chord and Kademlia, HyperCircle offers the most efficient broadcasting of messages. The HyperCircle delivery ratio could be improved by resending forwarded messages from leaving nodes. Kademlia has the lowest hop count and the best delivery ratio. Chord has both the highes hop count and delay time.

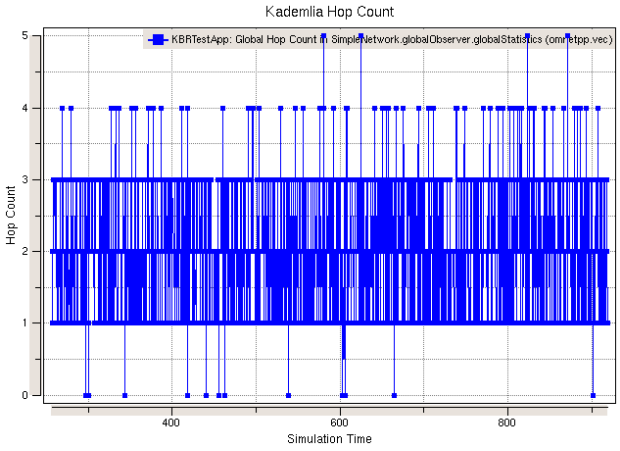

Figure 12. Kademlia Hop Count

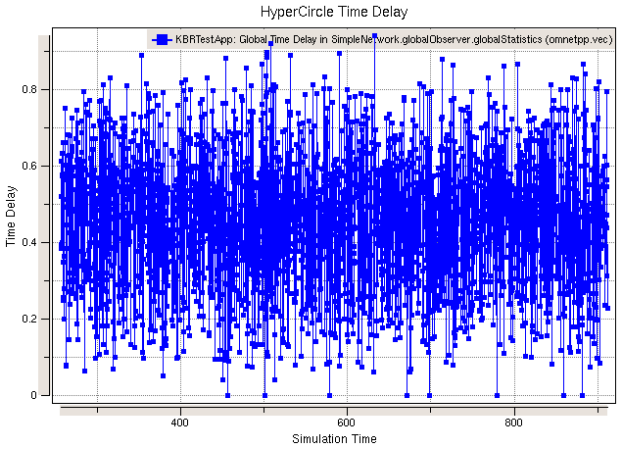

Figure 13. HyperCircle Time Delay

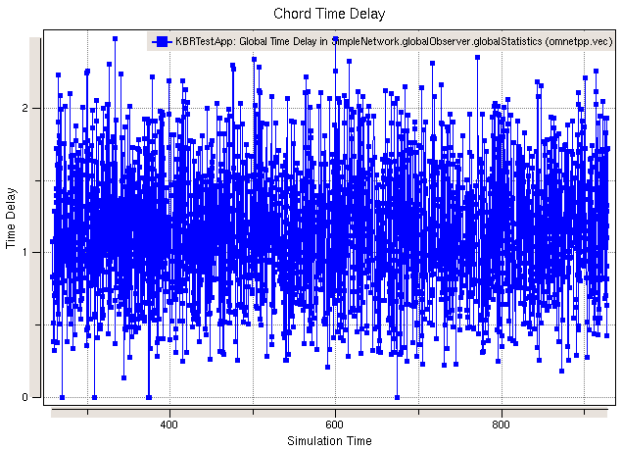

Figure 14. Chord Time Delay

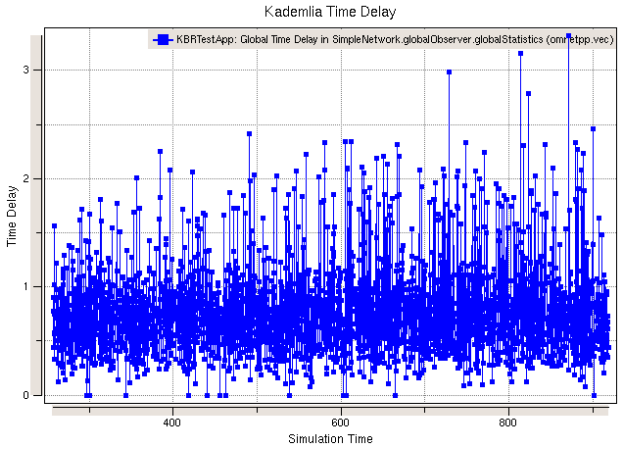

Figure 15. Kademlia Time Delay

5. Conclusion and Future Work

Uncontrolled large scale P2P networks face scalability problems, like unacceptably high broadcast time. The HyperCircle protocol tries to solve this challenge. Using an 8-point circle as main element, a graph topology is proposed for efficient broadcasting and routing of the messages in the network. The construction and maintenance of the topology including the broadcast algorithm were introduced in the paper.

The initial simulation evaluation results comparing to Chord and Kademlia show that the HyperCircle offers the most efficient broadcasting of messages. Based on this promising result, we believe that HyperCircle will be a useful component for large scale P2P applications.

Since our current work mostly focuses on the construction of super peers in P2P networks, the communication among the super peers and with the sub-ordinate nodes will be developed in the future. Furthermore, P2P applications using HyperCircle will have to be implemented.

Acknowledgements

This work is partly funded by the project DEON (Development and Evolution of Ontologies in Networked Organizations) based on a grant from STINT (The Swedish Foundation for International Cooperation in Research and Higher Education); grant IG 2008-2011.

References

[1] I. Baumgart, B. Heep, and S. Krause. Oversim: A flexible overlay network simulation framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, pages 79–84, May 2007.

[2] Edutella. http://www.edutella.org/edutella/edutella.shtml.

[3] GPS. http://www.cs.binghamton.edu/wyang/gps/.

[4] C. Henricsson and S. M. Abbas. A simulation framework for efficient search in p2p networks with 8-point hypercircles. Master's thesis, Jönköping University, 2008.

[5] F. Lin and K. Sandkuhl. Towards efficient search in p2p networks with 8-point hypercircles. In *Proceedings of IADIS International Conference WWW/Internet 2006*, pages 545–551, Murcia, Spain, October 2006.

[6] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[7] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A survey of peer-to-peer network simulators. *Proceedings of the 7th Annual Postgraduate Symposium (PGNet '06)*, 2006.

[8] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37:95–98, 2007.

[9] Napster. http://www.napster.com.

[10] Narses. http://sourceforge.net/projects/narses.

[11] Neurogrid. http://www.neurogrid.net/.

[12] OMNeT++. http://www.omnetpp.org/.

[13] OverlayWeaver. http://overlayweaver.sourceforge.net/.

[14] P2PSim. http://pdos.csail.mit.edu/p2psim/.

[15] PeerSim. http://peersim.sourceforge.net/.

[16] PlanetSim. http://planet.urv.es/planetsim/.

[17] QueryCycleSimulator. http://p2p.stanford.edu/.

[18] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, Feb 2003.