DHT Broadcast Optimisation with ID Assignment Rules

Michael Roth, Julia Schmitt, Florian Kluge, Theo Ungerer
Department of Computer Science, University of Augsburg, Germany
{*roth, schmitt, kluge, ungerer*}@*informatik.uni-augsburg.de*

Abstract

Decision making in a self-optimising distributed Organic Computing system requires information about the system's state. Accurate information enables the overall system to respond better to state changes. Distributed systems can use different network protocols, e.g UPD or TCP/IP, to connect the nodes. There is no guarantee that all of these network protocols are able to send broadcasts. If all used network protocols support broadcast it is still not sure that broadcasts can be sent across different protocol domains, e.g. from UPD to TCP/IP. We use therefore distributed hash tables (DHT) to enable an application layer broadcast for information dissemination, which only sends unicast messages in the network layer to spread node status information in a distributed system. In DHTs the node IDs are used to determine the communication partner. The node IDs are generated randomly in DHTs. In this paper we show how choosing IDs systematically, instead of generating them randomly, influences the network usage by using our DHT broadcast algorithms for information dissemination.

1 Introduction

The Autonomic Computing [1] and Organic Computing [2] initiatives focus on systems which are self-managing, self-organizing, decentralised, and do not possess a single administrative instance. Different methods are used to decide which actions must be taken to hold or guide Organic Computing systems in valid system states: e.g. genetic algorithms [3], automated planners [4], and learning classifier systems [5]. All these approaches require information about the current system state to make accurate decisions. The more accurate the information about the system state, the more fitting is the decision. Each component that influences the system requires such accurate information.

In our research we develop the self-organizing middleware OCμ [6] to manage the growing complexity of computer systems. Like any other middleware, OCμ allows to develop applications for distributed systems and manages their distribution. Additionally, OCμ provides mechanisms for failure recovery and load balancing among different devices, e.g. smartphones, desktop computers, and tablets. These devices are the nodes in our system. They are connected through different networks with varying bandwidths. Different applications, called services, can be deployed on these devices. An Organic Manager [7] on each device is used to provide the self-x properties in a decentralised manner. Each Organic Manager possesses goals which describe a valid system state. Goals can change, be created or be deleted during run-time therefore it is not clear which node possesses which goals. The Organic Manager can start, stop, and relocate services to other devices to achieve self-optimisation, self-configuration and self-healing.

To decide whether the system is in an undesired state the Organic Manager requires accurate information about each node in the network. Since each node runs an Organic Manager each node must send status updates to all other nodes. As the nodes are connected through different network protocols, we cannot guarantee that the networks are capable of sending broadcasts. Therefore we evaluated algorithms for efficient information dissemination for the use in our middleware.

An efficient algorithm for information dissemination must be able to distribute the node status information from each node in the entire network. To avoid flooding the network, we investigate a structured way for decentralised information spreading. We use distributed hash table (DHT) algorithms, a form of peer-to-peer network, to build a structured overlay network without the need of an administrative instance. With this overlay network we are able to spread status information from each node efficiently over the entire network.

In this paper we present an optimisation for the broadcast algorithms for information dissemination published in [8]. In these broadcast algorithms, we use the structure of DHTs for information dissemination in distributed systems. In the used DHT algorithms the node IDs are generated randomly before joining the network. This is the default behaviour described by the used DHT algorithms. By using the DHT algorithms for information dissemination we discovered a correlation between the distance of the sender's and receiver's node IDs. Our research showed that the shorter the distance between the node IDs the bigger the packages sent. In this paper we present an enhanced joining algorithm to assign close node IDs to nodes with good network connection. This optimisation decreases the hop count of large packages.

The rest of this paper is organised as follows. In section 2 we give a short introduction into distributed hash tables and present our previously published broadcast algorithms. Related work is shown in section 3. Section 4 introduces the enhanced joining algorithm for the optimised ID assignment. We describe the evaluation scenario for the presented algorithm and give the evaluation results in section 5. Section 6 concludes the paper.

2 Distributed Hash Tables

In 2001 the first distributed hash table (DHT) algorithms were published. DHT algorithms are decentralised, scalable, fault-tolerant peer-to-peer algorithms which provide a lookup service similar to a hash table. DHTs store key-value-pairs. The key determines on which node the pair is stored. These first algorithms are Chord, Pastry, Tapestry, and CAN.

In Chord [9], the nodes are arranged in a ring. Each node is connected to the nodes with the closest lower and higher node ID. The maximum size of the ring is 2^m, the node IDs range from 0 to $2^m - 1$. Each node stores short cuts to peers with a distance of $2^n, n \in [0, m-1]$ in a local routing table. If a Chord node cannot find the node with the searched ID the node with the next higher ID is used. The maximum number of nodes in the Chord ring must be known by each node to determine the size of the routing table.

Tapestry [10] and Pastry [11] are two similar peer-to-peer networks which both use the Plaxton routing algorithm [12]. In Plaxton node IDs are represented by numbers with radix m and length n. This leads to a maximum of m^n nodes. Each node has a local routing table with $m-1$ rows and n columns. The first column stores node IDs which have no common prefix with the local node ID. The common prefix of the node IDs in each column is one digit longer than the prefix of the previous column. The last column stores node IDs which match the local node ID in all but the last digit. The digit after the matching prefix is defined by the row. If the prefix and the digit after the prefix match the first digits of the local node ID the entry is skipped, therefore only $m-1$ rows are needed. Table 1 shows the routing table for the node 23012 with empty entries for matching prefix IDs. Entries containing an x allow more than one specific ID as long as the prefix matches. The routing table stored in the node only contains non-empty cells. Plaxton uses a proximity metric to decide which node with a matching prefix is selected. If no node ID with a given prefix exists the entry will be left empty. Hop count or ping time can be used as metrics to determine the best node.

0-xxxx	2-0-xxx		230-0-x	2301-0
1-xxxx	2-1-xxx	23-1-xx		2301-1
	2-2-xxx	23-2-xx	230-2-x	
3-xxxx		23-3-xx	230-3-x	2301-3

Table 1: Plaxton Routing Table for the Node 23012

CAN [13] uses an n-dimensional torus with coordinates from 0 to $m-1$ for each dimension. The node IDs represent coordinates on the surface of this torus, which leads to a maximum of m^n nodes. Each node is responsible for an area on the torus containing the node's ID. A node's routing table consists of the node IDs of the direct neighbours in each direction of the node's area. When a new node joins the network, an area is divided. Each node has at least one neighbour in each direction. A message is passed to that neighbour that is closest to the destination.

All three routing algorithms guarantee that each item can be located within a bounded number of hops, using only a small routing table on each host. The hops are the number of nodes a message must pass to reach its destination in the overlay network. No prediction over the hops within the physical network can be given.

2.1 Spreading Algorithms

In the following the term *message* denotes the status information from one node which should be spread. *Packages* are sent from one node to another containing multiple messages. A package is compiled by the sending node and contains all messages for the receiving node. The receiving node opens all received packages and rearranges the messages according to the node ID of the next hop into new packages for sending.

Our goal is to spread status information from each node to all other nodes. In our middleware, all nodes send messages in predefined time intervals. A time interval is larger than the time a message needs to travel to other nodes, therefore each package is delivered within

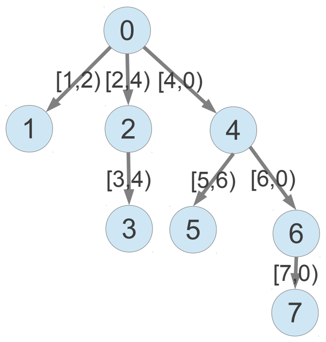


Figure 1: Broadcast Tree of a Chord network with 8 nodes

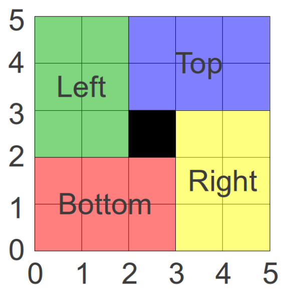


Figure 2: Area division in a 2-dimensional unfolded CAN torus

one interval. Within each time interval all nodes send packages to their neighbours containing all messages for them and additional information on how to forward the messages. We use this time interval to loosely synchronise the nodes. A step is the duration of this time interval. A message travels one hop per step. The information is sent continuously. We combine several messages with identical receivers into one package, to minimize the network overhead. Each message contains the status information from the node and a spreading interval. The spreading interval informs the node to which other node the information must be sent. The interval changes depending on the used spreading algorithm. The receiving node forwards a message to its neighbours within the spreading interval and divides the spreading interval for the new messages. Because of the well defined structure of the DHT routing tables the spreading intervals don't overlap. The full discussion and evaluation of the algorithms can be found in [8].

2.1.1 Chord

To spread information we use trees built from the neighbourhood relationship of the Chord algorithm. Figure 1 shows the trees for all nodes in a Chord network with 8 nodes. The initiating node sends its information and spreading instructions to all its neighbours. In Chord, the spreading instruction is the interval between two neighbour's node IDs. The neighbours then forward the message to their neighbours within the spreading interval and adapt the spreading interval. This Chord based information dissemination algorithm with k nodes can reach every node in $\lceil log_2 k \rceil$ steps.

2.1.2 Plaxton

For Plaxton networks, we use a spreading algorithm similar to the one used in Chord networks. Each node sends its messages to all neighbours and specifies the area to spread the information. Since the IDs in the routing table are organised by the similarity with the local node ID, the messages are forwarded in a different way than with Chord. Instead of a spreading interval with node IDs the spreading algorithm for Plaxton uses the length of the common prefix. The column number of the neighbour's node ID in the node's routing table c_s is identical with the length of the common prefix. Therefore, we use the position in the routing table instead of comparing the node IDs each time. The initiating node sends messages with the routing table column number c_s to all of its neighbours. The receiving node spreads the information to all neighbours which are stored in columns $c > c_s$ and uses the column number c as new spreading instruction. A Plaxton node has more neighbours than a Chord node. It requires $\lceil \log_m k \rceil$ steps, meaning the information dissemination is faster than with Chord. Less steps are required than in CAN based information dissemination but more messages and packages per step are sent.

Plaxton uses a proximity metric to choose neighbours if more node IDs match the required prefix. The chance of finding a node with a good connection increases if more node IDs match the required node ID prefix. In the first column of a Plaxton node routing table all but the first digit can be chosen freely from a large number of matching nodes. Therefore nodes with good connection can be chosen as neighbours. With each column the length of the prefix increases. In the last column of the routing table the node IDs must match the searched node ID completely and the proximity metric has no influence. Therefore the proximity metric has more impact on neighbours with shorter common prefix and thus higher distance to the local node ID.

2.1.3 CAN

In CAN the node IDs represent the position on the surface of an n-dimensional torus. The spreading interval must be a part of this torus. Before sending new information the node divides the surface in $2n$ sections as shown in figure 2 for a 2-dimensional torus. The initiating node sends its messages into all $2n$ directions. A receiving node is always in the corner of the new cover area and a message is only forwarded into n directions. Since we use an n-dimensional torus the maximum distance a message can travel in one direction is $(\frac{k}{2})$ hops in a CAN network with k nodes per edge. This leads to $n(\frac{k}{2})$ hops to reach the farthest node with n dimensions.

2.2 Evaluation Results

Our research in [8] has shown, that we can use these DHT based algorithms for information dissemination to spread the information to all nodes. Plaxton and Chord do not send redundant messages. The CAN based algorithm divides the surface of the torus into spreading intervals. It is possible that a CAN node is responsible for portions of different spreading intervals. Therefore some CAN nodes will receive the same message with different spreading intervals. A Plaxton node has the most neighbours, therefore the node sends the most packages. The Plaxton based algorithm spreads the information fastest. The CAN based algorithm spreads the information slowest, but has the least number of neighbours and number of packages per step. Our evaluations also show that large messages are sent to neighbours with a close node ID in the Chord and Plaxton based algorithm. In the CAN based algorithm the package size is not related to the distance between the node IDs. A CAN node sends a message to all nodes in the spreading interval. The spreading interval does not decrease with the distance of the node IDs like in Chord or Plaxton. As mentioned above, Plaxton's proximity metric works best if the common prefix with the neighbour's node ID is short. Therefore the Plaxton's built-in optimisation has little impact on the network utilization when used for information dissemination.

2.3 Challenges

For information dissemination the node status information must reach the entire network. Therefore we must send messages to nodes with a great distance. To optimise the network usage we want to minimise the package size for distant nodes. Our evaluation showed, that large packages are sent to nodes with a close node ID. Therefore we are looking for ways to assign nearby nodes close node IDs. The used algorithm should not put additional strain on the network. Because of the decentralised nature of our middleware we are looking for an algorithm which works decentralised.

3 Related Work

Proximity aware Peer-to-Peer networks has inspired a lot of research. Ratnasamy et al. presented the topology-aware CAN [14] in 2002. Topology-aware CAN requires landmark servers. Each node calculates the distance to these servers. The nodes are put in bins according to their distance to the different landmark servers. A new node chooses an ID near the nodes in its bin. Topology-aware CAN fits the overlay network to the physical network and thereby optimises the network usage. In [15] Xu et al. optimise CAN and use in the first step landmark servers for rough grouping. In the second step nodes with a close match measure their round-trip time (RTT) and then connect to close nodes. This method performs better than other methods which measure the RTT to all or random nodes. The landmark servers are single points of failure. Organic Computing Systems are failure tolerant and decentralised which contradicts the concept of landmark servers.

Xiong et al. presented Chord6 [16] in 2005. Chord6 is an enhancement for Chord on IPv6 networks. The Chord ID is generated by hashing the IPv6 address. The first half of the Chord ID is the hash of the first digits of the IPv6 address. The last part of the Chord ID is the hash of the remaining IPv6 address. This leads to close Chord IDs if the Chord nodes are in the same subnet. Our middleware can handle a lot of different devices with different connection types (e.g. Bluetooth). Therefore we can not use network addresses for ID generation.

PChord [17] enhances the Chord algorithm to use a proximity list in addition to the routing table. The proximity list contains nodes with low RTT. To determine which node should be placed into the proximity list the node checks the RTT on each received message. The proximity list is used to route messages to nearby nodes instead of the routing table. Using DHT algorithms for information dissemination presents new problems. For information dissemination we must cover the entire network. PChord introduces additional routing information to reach the destination node faster, but destroys the well defined structure of the DHT. Therefore we cannot guarantee that no redundant messages will be sent.

Kyasanur et al. [18] presented an algorithm to spread information through gossiping. Each node forwards a message with a given probability to all nearby neighbours. The messages can be spread in the entire network with a high probability. The authors presented a novel algorithm to reduce the message overhead and send the messages only over short distances in the physical network. This approach still produces a message overhead since a message is sent to all neighbours and a node can receive the same information twice. By using Chord or

Plaxton we can guarantee that the information reaches the entire network without any redundant messages.

Ucan et al. [19] investigated information dissemination in sensor networks. They presented a way to combine messages from multiple senders to minimise the network traffic by building a Minimal Spanning Tree (MST). This MST construction is topology aware and chooses nodes with a good connections. Each node must compute the MST before sending the first broadcast, which is very expensive. By using DHT networks we can use the available routing information to build a broadcast tree for all nodes.

Algorithm 1: Joining Algorithm

```
1 generate node ID;
2 if no node in DHT is known then
3     search nodes;
4 end
5 find nodes to connect with;
6 if ID is already used then
7     go to line 1;
8 end
9 join network;
```

4 ID Assignment Algorithm

To improve the network utilization in information dissemination we looked for a way to assign close node IDs to nearby nodes. According to our analysis of the used spreading algorithms this would shorten the distance the largest packages must travel in the underlying network.

4.1 Original Algorithm

In the used DHT algorithms a node generates a random node ID prior to joining the network. The new node requires at least one node which is already part of the DHT. In most cases such a node is not known a priori, therefore the joining node searches a node in the DHT. Most networks provide a way to discover other devices in the network. The joining node uses these techniques to find other devices in the network. After the devices are discovered the joining node determines if the device is part of the DHT. If a peer is found the node joins the DHT. The used DHT algorithm determines to which nodes the joining node must connect. As soon as one peer in the DHT is known the new node searches for nodes to connect with. The number of possible IDs is by magnitudes larger than the used node IDs. This minimises the possibility of an ID collision. In the unlikely event that a node with the generated ID is already part of the DHT a new ID is generated and the algorithm starts from the beginning. This algorithm is shown in algorithm 1.

4.2 Enhanced Algorithm

We propose a new approach to generate the node ID after the node IDs of nearby nodes are known. Nearby nodes can be found with a proximity metric. Examples for such a metric are bandwidth, hop count or travel time from the joining node. Like the original algorithm our enhanced algorithm discovers some nodes that are already part of the DHT. In difference to the original algorithm the enhanced algorithm waits till a few nodes respond. The joining node extracts the hop count or round-trip time from the response package. With this information the node IDs with the best connections are determined. Because all nearby nodes have close IDs the joining node receives responses from all nearby nodes and can determine unused node IDs. The joining node than chooses an unused node ID which is close to the ID of the nodes with the best connection. The enhanced joining algorithm is shown in algorithm 2.

With our enhanced joining algorithm we can generate a network in which the node IDs of nodes with good connection are numerically closer. Since all nearby nodes have a similar node ID a joining node receives a list of all used node IDs. This minimises the number of node ID conflicts. We cannot eliminate the conflicts entirely since it is possible that e.g. node 71 and 73 have a good connection to the joining node but node 72 is already present in the network with no good connection to the joining node. In this example the joining node must choose a different ID which places it as close as possible to the good connected nodes.

The enhanced joining algorithm can not ensure, that only nodes with good connections have nearby node IDs or that all nearby node IDs have a good connection. This is not necessary since nodes with a greater distance to nodes with close node IDs still have a better performance than nodes in a DHT with the original joining algorithm. Nevertheless the new joining algorithm allows nodes with a good connection to be neighbours in the used DHT network.

The resulting DHT possesses node clusters. In normal DHT this is undesirable and can lead to performance degradation. For information dissemination these clusters do not affect the performance. When single messages are routed in a DHT a node cluster could lead to a longer path length in some cases. For information dissemination a message must reach each node therefore the maximum path is always the longest path in the DHT. This longest path is determined by the used DHT algorithm and the size of the ID space and is not influenced by the node placement.

Algorithm 2: Enhanced Joining Algorithm

```
1  search nodes;
2  while wait for timeout do
3      collect response;
4      determine link quality for responding node;
5  end
6  find nodes with best link quality;
7  find unused node ID close to nearby nodes;
8  assign node ID;
9  if ID is already used then
10     go to line 7;
11 end
12 join network;
```

5 Evaluation

5.1 Analysis

In this section we present an analytical analysis of the original and the enhanced joining algorithm. Both algorithms send messages to find nodes in the DHT. Let a joining node send n_{search} messages to find nodes in the DHT. The original algorithm waits for one answer and joins the DHT. Since the responding nodes do not know if another node has already answered all nodes receiving a search request respond. A node must build its routing table after joining the DHT. The number of messages required for the search varies for each algorithm. Let n_{rt} be the number of messages the joining node requires to build its routing table.

By using the enhanced joining algorithm a node also sends n_{search} messages to find other nodes. The joining node waits for more response messages. In difference to the original algorithm the response of all nodes are used. At this point the number of messages is identical to the original algorithm. The joining node chooses an unused node ID close to the node ID of nearby nodes. This requires no additional messages. After choosing the node ID the joining node builds its routing table. Since the node already knows nodes with close node IDs it can fill some entries in the routing table without sending a message. To find routing table entries the joining node must send messages to find nodes with a greater distance in the ID space. This means that the number of messages required to find neighbours n'_{rt} is smaller than the number of messages required by the original algorithm n_{rt}.

5.2 Evaluation Scenario

We used a network simulator to evaluate the enhanced joining algorithm. In our evaluation scenario the devices are connected over a physical network. The physical network forms a tree with depth 3. We choose the size of the IDs to store 1024 nodes, 1024 being a valid network size for all three algorithms. This leads to a Chord exponent of 10 ($2^{10} = 1024$) and a Plaxton exponent of 5 using radix 4 ($4^5 = 1024$). In CAN we use a 2-dimensional network with a size of 32 in each dimension ($32^2 = 1024$). We build a physical network with 1024 nodes.

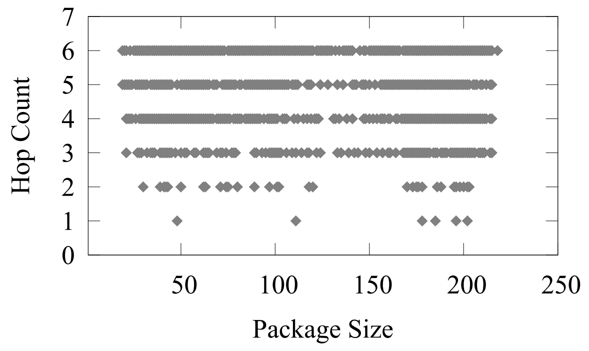
For the simulation we populate the DHT overlay networks with 50 to 1024 nodes in steps of 50. The remaining nodes are not a part of the DHT but were used to calculate the physical network hops. We simulate 10 runs for each number of nodes and each algorithm to reduce the impact of bad ID assignments. In the first simulation we use the original algorithm (algorithm 1). This means that the IDs are chosen randomly before the node joins the network. The second simulation uses the enhanced joining algorithm (algorithm 2) to choose the ID of a joining node in respect of nearby node IDs. In the initial phase of the simulation the nodes were chosen and the DHT network is build. After the DHT network is complete we start the evaluation. Each node sends messages in a predefined time interval to its neighbours. The neighbourhood relation is defined by the used DHT algorithm. Messages are combined into packages according to their receiver. Due to the determinism of the algorithms each message from one node is sent to the same nodes on its way through the network. We stop the evaluation when each node possesses information about each other node. To compare the joining algorithms we measure the packages size by counting the contained messages, the hops for each package and the package count for each physical connection. In this evaluation scenario we are concentrating on the hop count and therefore use the hop count as proximity metric. If the travel time or the bandwidth needs to be minimized these parameters should be used for the proximity metric.

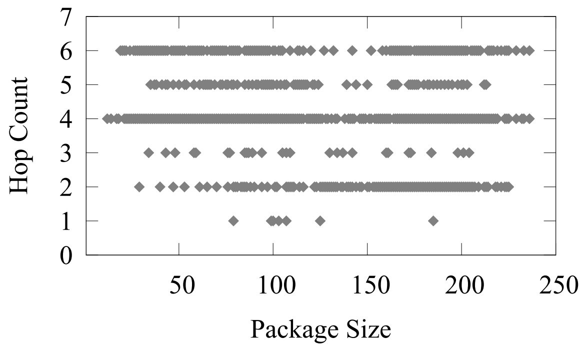
5.3 Simulation Results

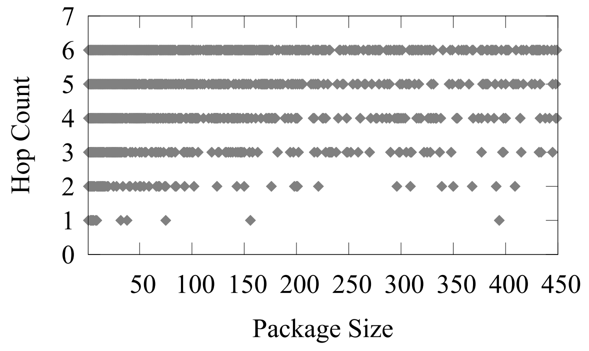
In this section we exemplary discuss results for networks with 450 nodes. Our evaluation shows that there is no significant difference between networks with different populations. Figure 3 shows the distribution of the package size and the hop count of a package for networks with random ID assignment (left) and the enhanced joining algorithm (right).

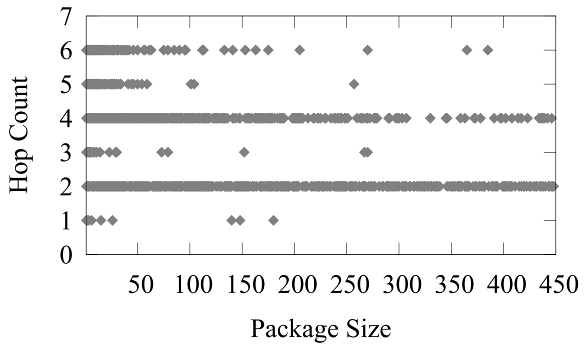
Figures 3a and 3b graphs show the CAN based algorithm. In CAN the relation between the node ID distance and the package size was not high in our previous evaluation. The two graphs confirm our research. The improvement is not significant for the CAN based algorithm.

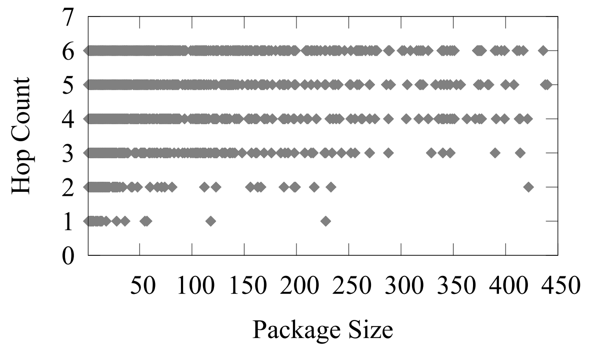
Figure 3c shows the packages size and the hop count for Chord networks with the original joining algorithm. Most large packages require over 4 hops to reach their destination. In figure 3d the distribution for the enhanced


(a) Hop Count and Package Sizes in CAN with Random ID Assignment


(b) Hop Count and Package Sizes in CAN with Enhanced ID Assignment


(c) Hop Count and Package Sizes in CHORD with Random ID Assignment


(d) Hop Count and Package Sizes in CHORD with Enhanced ID Assignment


(e) Hop Count and Package Sizes in Plaxton with Random ID Assignment

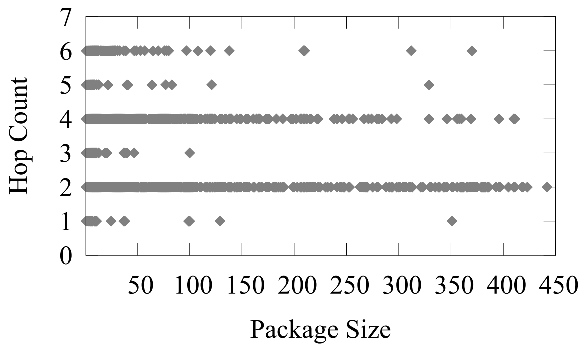

(f) Hop Count and Package Sizes in Plaxton with Enhanced ID Assignment

Figure 3: Hop Count and Package Sizes in different networks with random ID assignment (left) and the enhanced algorithm (right)

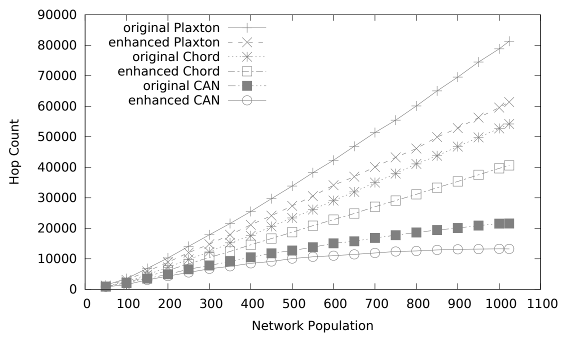


Figure 4: Hops per Network Population

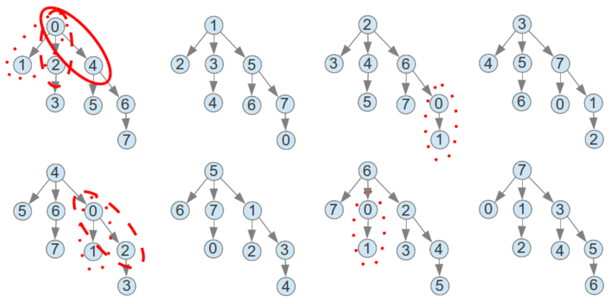


Figure 5: Trees of a Chord network with 8 nodes

joining algorithm are shown. In contrast to the original algorithm the enhanced algorithm reduces the hop count for large packages. Only a few packages with more than 200 messages require more than 4 hops.

Figure 3e shows that the hop count for large messages is high for Plaxton networks with the original algorithm. The enhanced algorithm reduces the hop count for large messages as shown in figure 3f.

Figure 4 shows the sum of required hops over all packages for all three information dissemination algorithm with different network populations. The figure shows that the enhanced joining algorithm significantly reduces the hop count of the packages for all three DHT broadcast algorithms and over all network populations. This behaviour can be explained by looking at the DHT broadcast algorithms.

Figure 5 shows all broadcast trees in the a Chord network with 8 nodes. The nodes sending information are placed at the roots. The originating nodes send the information to all nodes in the local routing table. The receiving nodes forward the message only to other nodes within the spreading interval. In the figure all messages send from node 0 are highlighted. 4 messages are sent to node 1, 2 messages to node 2 and only 1 message is sent to node 4. This means that only half the messages are sent when the distance between the node IDs doubles. All other nodes have the same behaviour.

To spread status information a Plaxton node sends messages to all nodes stored in the routing table. With each hop the number of used columns decreases. This means that the nodes in the first column of the routing table are only used by the initiating node. All nodes that forward a message send the information to the neighbours stored in the last column. The node IDs in the last column are closer to the local node ID because of the common prefix.

In CAN each node sends messages to its neighbours independently of the distance between the node IDs. By placing neighbours in the same network segment we are able to reduce the hops of the packages.

6 Conclusion

We showed that the placement of the node IDs is an important factor for information spreading algorithms. The enhanced joining algorithm for our DHT broadcast algorithm can reduce the hop count of large packages. Since each node must receive status updates it would decrease the speed of the information dissemination if we send messages only to nearby nodes. Therefore the enhanced joining algorithm can not eliminate all packages with high hop count but ensures that only small packages are send over the entire network.

The enhanced joining algorithm can also be used to optimise more than one parameter in DHT broadcast algorithms. By changing the proximity metric to two or more parameters can be considered. A weighted metric can be used to set the importance by more parameters. The influence of such a metric should be further investigated.

References

[1] Kephart, J., Chess, D.: The Vision of Autonomic Computing. IEEE Computer (Jan 2003) 41–50

[2] Müller-Schloer, C.: Organic Computing - On the Feasibility of Controlled Emergence. In: International Conference on Hardware/Software Codesign and System Synthesis, 2004, IEEE (2004) 2–5

[3] Ramirez, A.J., Knoester, D.B., Cheng, B.H., McKinley, P.K.: Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems. In: Proceedings of the 6th international conference on Autonomic computing. ICAC '09, New York, NY, USA, ACM (2009) 97–106

[4] Schmitt, J., Roth, M., Kiefhaber, R., Kluge, F., Ungerer, T.: Realizing self-x properties by an automated planner. In: Proceedings of the 8th ACM

international conference on Autonomic computing, ACM (2011) 185–186

[5] Prothmann, H., Rochner, F., Tomforde, S., Branke, J., Müller-Schloer, C., Schmeck, H.: Organic Control of Traffic Lights. In: Autonomic and Trusted Computing. Springer (2008) 219–233

[6] Roth, M., Schmitt, J., Kiefhaber, R., Kluge, F., Ungerer, T.: Organic Computing Middleware for Ubiquitous Environments. In Müller-Schloer, C., Schmeck, H., Ungerer, T., eds.: Organic Computing A Paradigm Shift for Complex Systems. Autonomic Systems. Springer Basel (2011) 339–351

[7] Schmitt, J., Roth, M., Kiefhaber, R., Kluge, F., Ungerer, T.: Using an Automated Planner to Control an Organic Middleware. In: Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on, IEEE (2011) 71–78

[8] Roth, M., Schmitt, J., Kluge, Florian und Ungerer, T.: Information Dissemination in Distributed Organic Computing Systems with Distributed Hash Tables. In: The 10th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2012, IEEE (2012)

[9] Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM Computer Communication Review **31**(4) (2001) 149–160

[10] Zhao, B.Y., Kubiatowicz, J., Joseph, A.D., Zhao, B.Y., Kubiatowicz, J., Joseph, A.D.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, University of California at Berkeley (2001)

[11] Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-Peer Systems. In: Middleware 2001, Springer (2001) 329–350

[12] Plaxton, C., Rajaraman, R., Richa, A.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. Theory of Computing Systems (1999) 241–280

[13] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable Content-Addressable Network. ACM SIGCOMM Computer Communication Review (2001) 161–172

[14] Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Topologically-aware overlay construction and server selection. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, IEEE (2002) 1190–1199

[15] Xu, Z., Tang, C., Zhang, Z.: Building Topology-Aware Overlays using Global Soft-State. In: Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on, IEEE (2003) 500–508

[16] Xiong, J., Zhang, Y., Hong, P., Li, J.: Chord6: IPv6 based topology-aware Chord. In: Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on, IEEE (2005)

[17] Feng, H., Minglu, L., Minyou, W., Jiadi, Y.: PChord: improvement on Chord to achieve better routing efficiency by exploiting proximity. IEICE transactions on information and systems (2006) 546–554

[18] Kyasanur, P., Choudhury, R., Gupta, I.: Smart Gossip: An Adaptive Gossip-based Broadcasting Service for Sensor Networks. In: IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), IEEE (2006) 91–100

[19] Ucan, E., Thompson, N., Gupta, I.: A Piggybacking Approach to Reduce Overhead in Sensor Network Gossiping. In: Proceedings of the 2nd international workshop on Middleware for sensor networks. MidSens '07, New York, NY, USA, ACM (2007) 19–24