Automatic Formation of Dynamic Decentralized Networks

Pavel Poupyrev, SenYoshida, and Kazuhiro Kuwabara
{pavel, yoshida, kuwabara}@cslab.kecl.ntt.co.jp

NTT Communication Science Laboratories
2-4 Hikaridai, Seika Soraku, Kyoto 619-0237 JAPAN

Abstract: In this paper, we propose a model for the automatic formation of decentralized networks with an ordered systematic structure. We describe an algorithm for adding new nodes to decentralized networks and a strategy for correcting possible conflicts while new nodes are being added. We believe that this model can provide a basis for the further development of efficient network search algorithms and fault-tolerance mechanisms.

1. Introduction

With the rapid growth of the Internet, several architectures have emerged for the network organization of computer systems. Despite the immense variety of these architectures, all of them are based upon two primary architectures. The first is a centralized architecture, distinguished by the presence of a central server that ensures the functionality of the system. The other is a decentralized architecture of network applications that, unlike the centralized one, does not have a central server. The advantages of the decentralized approach are that on some occasions it is faster, more reliable, and requires less system administration and hardware costs. However, the design of the architecture has several drawbacks, one of the most serious involving how to automatically add new participants to decentralized networks.

In this paper, we propose a model for forming decentralized networks. The model has an algorithm for adding new nodes and a solution for correcting conflicts that could conceivably occur while new nodes are being added to such networks. We believe that this model can provide a basis for further development of efficient network search algorithms and fault-tolerance mechanisms.

The rest of this article is organized as follows: in the next section, we give the background and motivation for this work. Then, we describe the proposed model, the algorithm for adding new nodes to decentralized networks, and the message protocol, and consider conflicts that could possibly occur while new nodes are being added. We then continue with experiments on the system and state our conclusions and future work.

2. Background and Motivation

Most networked computer systems use a centralized architecture. For example, multi-agent systems [5], despite their decentralized nature, always require the use of a centralized server. In these systems, the central server has a directory service that contains agent

descriptions, such as their names, properties, network addresses, and so on. When an agent attempts to locate other agents that satisfy given criteria, it has to request the directory service to provide a list of such agents. After obtaining this information, the requester can directly communicate with these agents and use their services, for example, buy a cheap ticket from an airline reservations agent [4].

However, this approach has a number of disadvantages. The first is that it has a central point of failure, i.e., if the central server goes down the whole system fails. Another disadvantage is that because agents (nodes) belong to different organizations, it is not clear which organization should control the central directory service. Therefore, a centralized architecture is intrinsically unsuitable for representing distributed resources.

Decentralization has been proposed to overcome these problems. In this approach, we assume that all nodes are interconnected, and that every node has a list of physical addresses, i.e., an address book, of other participants in the network. The most influential example of such a decentralized architecture is Gnutella [3]. Gnutella is a file sharing system that has no central server; every client acts as both a client and a server. A Gnutella network has a random network topology, and when a new node is added to the network, it builds its address book by sending a “ping” message to one known node of the Gnutella network, which propagates this message to the other nodes from its address book. The message field, TTL (time to live), defines the number of hops the message can be forwarded with. All nodes that receive the message reply with an answer directly to the requester, which collects all replies and generates its address book. Then, the requester queries nodes from its address book for needed resources.

Freenet [1] is another decentralized file sharing system. It also does not have any central server and all data is distributed among nodes. Similar to Gnutella, when a new node shows up in the Freenet network the node sends a message to one known client that takes a random address from its address book and forwards this message. All nodes that receive this message reply directly to the requester. Unlike Gnutella, Freenet limits the number of connections among nodes and the resulting network represents a random graph. The creators of Freenet have attempted to explain the properties of the network using a “small world” model [2, 6].

A decentralized architecture has several problems that need to be considered:

- **Node addition:** How a new node can be added to a decentralized network.
- **Node search:** How an efficient node search can be performed in a decentralized network.
- **Adaptation to failure:** How a decentralized network can adapt if one node fails.

In this work, we attempt to answer the first problem of how new nodes can be added to a decentralized network. Here, not only does a new node have to form its own address book, but also other nodes have to add this newly added node into their address books.

There are two techniques for constructing decentralized networks. The first is a manual technique that involves human intervention. As a rule, this technique is inappropriate in the case of large dynamic networks because it is not feasible for people to track all changes in such networks. The other approach is the automatic addition of nodes into a network [1, 3]. Here, a new node requests a known node of the network to add it to the distributed network. The known node in turn propagates the request to all nodes in its address book, which then reply to the newly added node, thereby forming its address book, and update their own address books by adding this new node. It has been observed, however, that the resulting topology of the network is random [7], since there are no mechanisms to systematically control the addition of nodes based on certain criteria.

We propose a model that allows the automatic formation of a decentralized network with an ordered systematic structure. The distinguishing feature of the proposed model is that the resulting topology of the decentralized network has an ordered systematic structure that allows using effective algorithms for search in the network and adding fault-tolerance mechanisms. The construction of the decentralized network is achieved by assigning a virtual address that is a random value allocated from the interval (0,1), to every node. This is used to define the area of the network where the new node can be added using the algorithm described below.

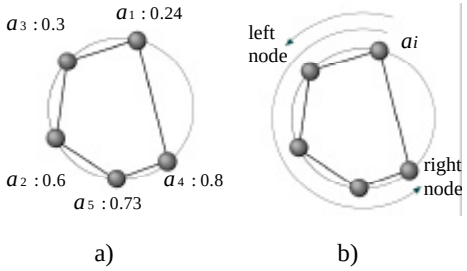

Figure 1. a) Cyclic relationship of nodes, b) left and right nodes relative to a_i that are nodes from the neighborhood of a_i.

3. Architecture

3.1 The model

We consider a set of N nodes $A:\{a_1, a_2,..., a_N\}$. We define the neighborhood of node a_i, $a_i \in A$, as a subset of nodes $A_i:\{a_{i1}, a_{i2},..., a_{ili}\}$ consisting of l_i nodes ($l_i << N$) to whom a_i can send messages.

For each node a_i, we define a virtual address g_i, which is a real value on the interval $R \in (0,1)$. There is a one-to-one correspondence between the set of all nodes and the real numbers from the (0,1) interval, i.e., each node a_i has a unique virtual address g_i. Therefore, each node is represented by its virtual address and neighborhood: $a_i : g_i, \{a_{i1}, a_{i2},..., a_{ili}\}$.

We then define the distance from the given a_i to a_j as follows:

$$d(g_i, g_j) = \begin{cases} g_j - g_i, & if \ \ g_i < g_j \\ 1-(g_i - g_j), & otherwise \end{cases}$$

This function allows defining a continuous cyclic relationship between the virtual addresses of the nodes. For simplicity, we also assume that each node a_i has a neighborhood consisting of only two nodes, where the left node is a node for which the distance function is minimal, i.e., the closest one, and the right one is a node for which the distance function is maximum, i.e., the furthest one on the rim (Figure 1). We also call the left and right nodes the closest nodes of a_i.

3.2 The algorithm for adding new nodes

The algorithm for adding a new node to a decentralized network (Figure 2) consists of the following steps:

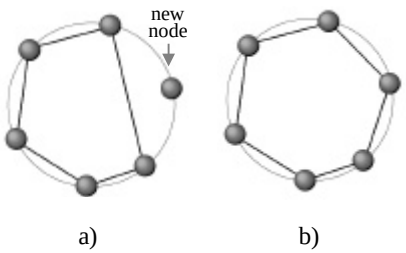

Figure 2. a) Topology before adding a node, and b) topology after a new node is added into a network

1. a_{N+1} allocates $g_{N+1} \in (0,1)$, and selects a_s to connect to,
$(a_i = a_s \quad a_i : g_i, \{a_{i1}, a_{i2}\}, a_i \in A, d(g_i, g_{i1}) < d(g_i, g_{i2}))$

2. a_{N+1} informs the agent a_i with its virtual address g_{N+1}

if $(d(g_i, g_{i1}) < d(g_i, g_{N+1}) \wedge d(g_i, g_{i2}) > d(g_i, g_{N+1}))$

then goto 3

else

2.1 if $(g_{N+1} = g_i) \vee (g_{N+1} = g_{i1}) \vee (g_{N+1} = g_{i2})$ **then**

$g_{N+1} = g'_{N+1} \in (g_{i1}, g_i) \cup (g_i, g_{i2})$

2.2 if $d(g_i, g_{i1}) > d(g_i, g_{N+1})$ **then**

$A_{N+1} = \{a_{i1}, a_i\}, \; A_i = \{a_{N+1}, a_{i2}\}, \; A_{i1} = \{a_{i11}, a_{N+1}\}$

2.3 else if $d(g_i, g_{i2}) < d(g_i, g_{N+1})$ **then**

$A_{N+1} = \{a_i, a_{i2}\}, \; A_i = \{a_{i1}, a_{N+1}\}, \; A_{i2} = \{a_{N+1}, a_{i22}\}$

3. $a_i = \begin{cases} a_{i1}, \; if \; d(g_i, g_{N+1}) - d(g_i, g_{i1}) < d(g_i, g_{i2}) - d(g_i, g_{N+1}) \\ a_{i2}, \; otherwise \end{cases}$

goto 2

Figure 3. An algorithm for adding a new agent into a network

1. Calculate the virtual address of the new node a_{N+1}. The address is calculated as a random number from the interval (0, 1).
2. In this algorithm, we also assume that we know the address of at least one node of the decentralized network. Send a "join" message to the known node a_s of the network. The message contains the virtual address of the new node.
3. The receiving node a_i calculates whether the new node belongs to its neighborhood (for the equation of the condition, which defines if the node belongs to the neighborhood, see the formal algorithm description in Figure 3, step 2).
4. If the new node belongs to the neighborhood of a_i, and the virtual address of node a_{N+1} matches one of any existing addresses, then assign a new virtual address for the new node (see step 2.1). Next, a_i updates its neighborhood to include new node a_{N+1}. Node a_{N+1} forms its neighborhood to include a_i and one of the nodes previously in the neighborhood of a_i (see steps 2.2 and 2.3).
5. If the neighborhood of a_i does not include a_{N+1}, then it selects the next node to forward a "join" message as described in step 3.

Basically, this process involves rearranging connections in a neighborhood of three nodes.

3.3 Message Protocol

The above algorithm defines the topology of the network, i.e., the connections among nodes. Here, we describe the message protocol for building that network topology. Every message has several fields such as sender, receiver, reply-to, and message type. The first three fields carry information about the addresses of the participants in communications. It is important that every participant is represented by two addresses: a physical address and a virtual address. The protocol defines three messages as follows:

- **join** message**:** this message is used to request a node to add a new node (i.e., that pointed to in the reply-to field) into the network. This message is forwarded from one node to another until it reaches a node that initiates the addition.
- **reply** message**:** this message requests a node to add the address of the sender to its address book when a new node is added to the network.

- **announce** message**:** this message is sent by an existing node to ask another node to add the address pointed to in the reply-to field in its address book and to send a reply message to the node pointed to in the reply field.

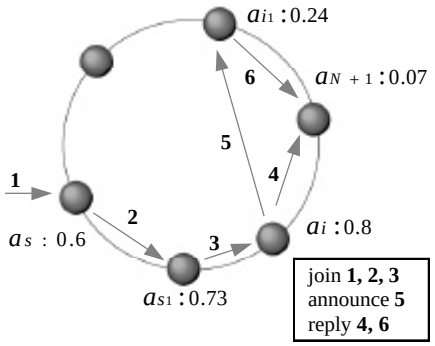

Figure 4. Message passing while a new node is being added into a network

Figure 4 depicts the typical example of message passing while a new node is being added to a network. The new node a_{N+1} first sends a "join" message to a known node a_s in the network. The node a_s forwards the "join" message to one of its neighbors from its address book since the new node a_{N+1} does not belong to the neighborhood of a_s (Figure 3, step 2). The selected neighbor is a node a_{s1} that satisfies the condition in Figure 3, step 3. The "join" message is forwarded until it reaches a_i such that the new node belongs to the neighborhood of a_i. The node a_i initiates the addition of the new node a_{N+1} by sending a "reply" message to it and an "announce" message to node a_{i1}, one of the neighbors of a_i (Figure 3, steps 2.2 and 2.3). On receiving this "announce" message, node a_{i1} sends the "reply" message to the new node.

Therefore, the new node a_{N+1} forms its address book by getting two "reply" messages from its new neighbors on the left and right. These two nodes also change their address books after receiving the "join" and "announce" messages.

3.4 Correction of possible conflicts when nodes are being added

The described algorithm assumes that only one node is added at a time although in reality several nodes can initiate the procedure of being added to a network. With several nodes, however, the described algorithm causes the nodes to form wrong address books. This is shown by the example in Figure 5, where two nodes are trying to connect to a network at the same time, to their closest nodes.

In the figure, the two nodes C and D are trying to simultaneously join a network. They both send a "join" message to nodes B and E, respectively, to be added into the network. Upon receiving the nodes' "join" messages, nodes B and E add new nodes C and D in their address books. Therefore, according to our algorithm, B adds C and E adds D. After nodes B and E send "reply" messages to C and D they both inform each other that each of them has a new neighbor by sending "announce" messages. After receiving the announce messages, B adds D and E adds C in their address books. The resulting contents of the address books for all of the nodes are presented in Table 1 at time t_2. The correct contents of the table, however, are actually different and are presented in the last row of Table 1.

Table 1. The content of the agents' address books, at different times.

	B	**C**	**D**	**E**
Initial	{A, E}	{B}	{E}	{B, F}
t_0	{A, C}	{B}	{E}	{D, F}
t_1	{A, D}	{B}	{E}	{C, F}
t_2	{A, D}	{B, E}	{B, E}	{C, F}
Desired	{A, C}	{B, D}	{C, E}	{D, F}

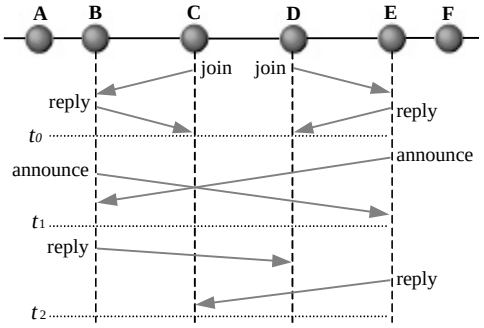

Figure 5. Collisions while two agents simultaneously join the network

We can correct these conflicts by regularly sending "ping" messages to the nodes listed in the address books. Every time a node receives a "ping" message it checks whether its address book has the sender's address. If it does, the node ignores the message. Otherwise, it shows that there are errors in the address book. There are two possible variants: The first is that the sender of a "ping" message is closer than at least one node from the address book. In this case, the receiver should update its address book accordingly. The other possibility is that the sender is not closer than any node from the address book. In this case, the receiver should send an announce message to a closer node from the address book, thus correcting the sender's address book.

4. Experiments

To conduct an initial evaluation of our approach, we built a simulation of the model in Java. It had a number of threads each of which represented a node. Java message objects were used to simulate the network message exchanges. A main thread collected statistics of the sent and received messages.

To evaluate the performance of the model, we consequently added a new node into the network and calculated how many messages were needed to construct a network of N nodes. The new node used a random node of the network to send a "join" message. The results showed that the number of messages that are needed to organize an N-node decentralized network architecture grows approximately as a power function (Figure 6).

However, the addition of other nodes (remote nodes) into address books can significantly decrease the number of messages when constructing the decentralized network of N nodes. To demonstrate this, we evaluated the model with address books consisting of two closest nodes and one remote node, where the closest nodes were nodes obtained as described in the previous sections. The remote node, on the other hand, was the node to which the new node sent the first "join" message when connecting to the network. In order to forward "join" messages more effectively, we modified step 3 in the algorithm, as follows:

$$a_{min} = \begin{cases} a_{i_1}, \ if \ d(g_i, g_{N+1}) - d(g_i, g_{i_1}) < d(g_i, g_{i_2}) - d(g_i, g_{N+1}) \\ a_{i_2}, otherwise \end{cases}$$

$$if \ \left|d(g_i, g_{N+1}) - d(g_i, g_{remote})\right| < \left|d(g_i, g_{N+1}) - d(g_i, g_{min})\right|$$

$$a_{min} = a_{remote}$$

$$a_i = a_{min}$$

goto 2

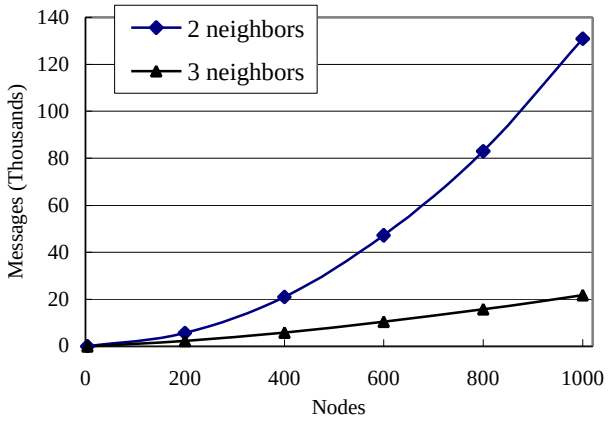

Figure 6. The average number of messages needed to form a network of N nodes.

This resulted in an almost linear performance (Figure 5).

5. Conclusions and Future Work

In this work, we described a model for the automatic formation of a network topology, which has a decentralized architecture, and an algorithm for adding new nodes to the topology. This model is based on assigning virtual addresses from an interval (0,1), which allows a client to form its address book in a systematic ordered fashion. A simulation showed that it is possible to build this topology and exclude possible errors while new nodes are being added. This model demonstrated a simple method of constructing a network topology with an ordered systematic structure.

In our future work, we will continue investigating the proposed model including:

- an investigation on the efficient addition of new nodes into a decentralized network
- the development of a network adaptation mechanism to address node failure
- the implementation of search algorithms
- an evaluation of the scalability considering the transmission speed, possible message delay and the bandwidth
- a performance comparison of our algorithms with other previously reported techniques such as those in the Gnutella network.

References

1. I. Clarke, O. Sandberg, B. Wiley and T.W. Hong, "Freenet: A distributed Anonymous Information Storage and Retrieval System", in *Designing Private Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 46-66, 2001.
2. D. Watts and S. Strogatz, "Collective dynamics of 'Small world' networks", in *letter to nature*, 393, 440-442, 1998.
3. "Gnutella", http://gnutella.wego.com/, 2001.
4. J. Morris and P. Maes, "Sardine: An Agent-facilitated Airline Ticket Bidding System", Software Demos, in *Proceedings of the Fourth International Conference on Autonomous Agents* (Agents 2000), 2000.
5. M. Huhns and M. Stephens, "Multiagent Systems and Societies of Agents", Multi-agent systems, edited by G. Weiss, 1999, Massachusetts Institute of Technology, 79-82.
6. S. Milgram, "The small world problem", *Psychology Today* 1(1), 60-67, 1967.
7. "Gnutella Network Map", http://www.clip2.com/dss_map.html, 2001.