社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

アトラクター選択による自己適応型DHT経路表設定

林　　銳[†]　ライプニッツ　賢治[†]　村田　正幸[†]

[†] 大阪大学大学院情報科学研究科　先進ネットワークアーキテクチャ講座
〒 565-0871 大阪府吹田市山田丘 1-5
E-mail: †{r-lin,leibnitz,murata}@ist.osaka-u.ac.jp

あらまし　本稿では、**分散型ハッシュテーブル** (DHT) における経路表サイズの自己適応的な制御について議論する。多くの DHT 機構では静的に経路表の設定を行うため、環境変化によってパラメータ設定が適切でなくなった場合は、通信オーバーヘッドが大きくなり、ネットワーク全体の性能が劣化する。そこで、本稿では、制御不可能かつ予測困難なアンダーレイネットワークの変動によって生じるトラヒック変動に対する適応性を実現するために、生物の未知の変化に対する適応性をモデル化した**アトラクター選択**に基づく経路表サイズの自己適応的制御手法を提案する。アンダーレイネットワークの状態が変化したときの通信オーバーヘッドの最小化を目的とし、Pastry の経路表の制御にアトラクター選択を適用することで適応的な経路表制御を実現する。

キーワード　DHT, Pastry, アトラクター選択, 自己組織, 経路表

Self-Adaptation of DHT Routing Table Size with Attractor Selection

Rui LIN[†], Kenji LEIBNITZ[†], and Masayuki MURATA[†]

[†] Osaka University, Graduate School of Information Science and Technology
Advanced Network Architecture Laboratory
1-5 Yamadaoka, Suita, Osaka, 565-0871 Japan
E-mail: †{r-lin,leibnitz,murata}@ist.osaka-u.ac.jp

Abstract　In this paper we discuss the self-adaptation of the routing table size of a distributed hash table (DHT) by applying the biologically inspired attractor selection method to adapt to variations in the traffic caused by uncontrollable and unpredictable fluctuations in the underlay network. Since the common DHT mechanisms provide only static settings of the routing table, our goal is to perform an adaptive control, as unsuitable parameter settings would cause too much overhead traffic and deteriorate the overall network performance. We adopt attractor selection as adaptation scheme, since it provides better self-adaptablity and robustness features than other methods and we demonstrate its applicability to control the DHT node state for the case of the routing table size of Pastry, a well-known DHT algorithm. Our goal is to minimize the overhead traffic when the the conditions in the underlay network change.

Key words　DHT, Pastry, attractor selection, self-organization, routing table

1 Introduction

In structured *peer-to-peer* (P2P) overlay networks, each node (or peer) and file key is assigned a unique ID, based on a consistent hash function. The file keys are mapped to nodes according to their IDs and a *distributed hash table* (DHT) definition. The DHT maintains topological relationships between the nodes and supports a routing protocol to locate a node responsible for a required key. Representatives of DHT systems include variable-degree DHTs, such as CAN [11], Chord [16], Pastry [13], Tapestry [17], Kademlia [10], or constant-degree DHTs like Cycloid [15], Koorde [3] and Vicery [9]. Most of the variable-degree DHTs require $O(\log n)$ hops per query request with $O(\log n)$ neighbors per node, where n is the network size. In contrast, constant-degree DHTs achieve a query path length of $O(d)$ with $O(1)$ neighbors, where d is the network dimension and $n = d\,2^d$.

Most of the existing DHT algorithms are defined with static parameter settings, but the underlay network and user behaviors are uncontrollable and unpredictable and greatly influence the performance of the DHT network. Suboptimal conditions cause deterioration in the performance of the system, such as heavy-tailed query distributions (*skewness*), high rates of node joining and leaving (*churn*), and wide variations in network and storage capabilities and capacities (*heterogeneity*) [5]. So in order to cope with these aforementioned issues, adaptation of the nodes' state is a necessary and important condition in designing robust DHT networks.

Attractor selection is a biological dynamical adaptation model introduced by Kashiwagi et al. [4]. It is a stochastic approach that determines the system state according to the suitability of the current system state. In this paper we propose a variation of Pastry that can achieve traffic-based adaptation by adopting attractor selection. Each node has an internal state, e.g., the routing table size. This state controls the behavior of forwarding the query requests and affects the total query traffic of the DHT network. After receiving the query reply, the *querist* (i.e. source of the query) keeps track of how many hops it is away from the *destination* (i.e. the node which holds the queried key). This *hop count* contributes to an overhead in the network *query traffic* as occasionally non-optimal routes are chosen, and its measurements reflect the network state. Beside the query traffic, the *maintenance traffic* also needs to be considered. A node must periodically contact its neighbors that are stored as entries in the routing table and leafset, to assure that they are still alive. Node or connection failures will result in query failures and additional traffic and latency. The maintenance traffic is proportional to the routing table size and the update interval. After a fixed interval, if the querist accumulates sufficient data to obtain an updated view of the network, it will change its state by using attractor selection to achieve better performance causing less overhead traffic.

The rest of this paper is structured as follows. Section 2 briefly discusses related work on current adaptation schemes in DHT. Section 3 gives the details of attractor selection and in Sect. 4 we demonstrate how this concept can be applied to the self-adaptation of the Pastry routing table size. In Sect. 5 we evaluate the performance of the proposed adaptation scheme through numerical simulations. Section 6 summarizes our approach and we give some remarks on possible future extensions.

2 Related Work

There exists some similar work to ours on adapting the DHT routing table size. Shen et al. [14] proposed an elastic routing table to deal with the query load balancing and avoid congestion. This mechanism allows each node to have a routing table of variable size corresponding to node capacities. The in-degree and out-degree of the routing table are also adjusted dynamically in response to the change of file popularity and network churn. Li et al. [8] presented a performance versus cost framework (PVC) that allows designers to compare the effects of different protocol features and parameter values. PVC analysis shows that the key to efficiently using additional bandwidth is for a protocol to adjust its routing table size. However, these above-mentioned proposals focus only on specific aspects of DHT and cannot deal with more complex network conditions, which is the goal of our research.

3 Attractor Selection Model

In this section we will give an outline of the principle of attractor selection which is the key component in our method. The original model for adaptive response by attractor selection is given by Kashiwagi et al. in [4] and is based on the dynamics of biological experiments on mutually inhibitory operons in the gene expression of E. Coli cells.

Basically, we can outline the attractor selection method as follows. Using a set of differential equations, we describe the dynamics of an M-dimensional system, see Eqn. (1).

$$\frac{dm_i}{dt} = f(m_1, \ldots, m_M)\, g(\alpha) + \eta_i \qquad i = 1, \ldots, M \tag{1}$$

The state of the system is given by the vector over all m_i values and its dynamic behavior is influenced by the two functions f and g. When the system evolves over time, it converges to certain attracting equilibrium points that are defined by the product of functions f and g. Each differential equation is furthermore under the stochastic influence of a noise term η_i that corresponds to an inherent Gaussian noise found in the original gene expression model. This random noise term causes the system to be constantly in motion. However, once the system has converged to an attractor, it remains there as long as the attractor is stable.

Additionally, we introduce an activity term α, which changes the sensitivity of the system to the influences from the noise terms. For example, if $g(\alpha) \to 1$, the system behaves rather deterministic and converges to attractor states predefined by the structure of the differential equations. However, for $g(\alpha) \to 0$ the noise term will become the dominating factor in the behavior of the system and essentially a random walk is performed. When the input values require the system to react to modified environmental conditions, activity α changes accordingly, causing the system to search for a more suitable state. In the course of this random search, the activity value increases again as soon as a better solution is approached and the influence of the random term is reduced.

In our approach, we control the selection of the appropriate attractor by the activity term α, which indicates how well the current system state corresponds to the influencing factors from the environment. The activity directly influences the differential equation system by causing attractors

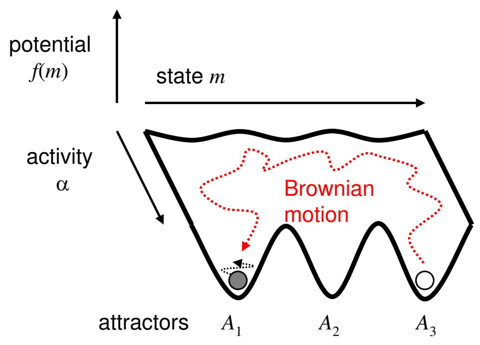


Fig. 1 Schematic figure of attractor selection principle

to become unstable if the current system state is not suitable for the environmental conditions.

The principle of attractor selection is sketched in Fig. 1. The x-axis represents the system internal state m, which is for simplicity only shown in the one-dimensional case. Let us assume that three attractors A_1, A_2, and A_3 exist and the system is currently attracted to A_3. If now an influencing environmental factor makes this solution no longer suitable, the function α will cause a decrease in the growth function $g(\alpha)$, leading to a "flatter" potential landscape defined by f. Since the noise term always exists, the system will be driven away from attractor A_3 and perform a Brownian motion in the phase space. Once m approaches a more suitable attractor (in this case attractor A_1, the system will become more deterministic keeping the system state at this attractor in spite of the still existing noise terms.

4 Proposed Extension of Pastry

In the following we will consider an extension to Pastry, which is a popular DHT algorithm that specifies how keys are distributed among the nodes and how the node responsible for holding a key can be found. Its variation Bamboo [12] is considered the most promising candidate for IETF P2PSIP.

4.1 Routing Table Handling in Pastry DHT

Each node in the Pastry network has a unique identifier (*nodeId*), maintains a *routing table*, a *neighborhood set*, and a *leafset*. When presented with a message and a key, a Pastry node efficiently routes the message to the node with a *nodeId* that is numerically closest among all currently live Pastry nodes to the key. The expected number of routing steps is $O(\log 2^N)$, where 2^N is the current number of Pastry nodes in the network. Each Pastry node keeps track of its immediate neighbors in the *nodeId* space and notifies the users of the routing algorithm (for example, the querist and *forwarders*) of new node arrivals, node failures, and failure recoveries.

4.1.1 Node Identifier and Neighborhood Set

Each node in the Pastry overlay network is assigned a 128-bit *nodeId*, which is used to indicate a node's position in a circular *nodeId* space ranging from 0 to $2^{128}-1$. The *nodeId* is assigned randomly using a hash function such as SHA-1 when a node joins the system, such that the resulting set of *nodeIds* is uniformly generated. Assuming a network consisting of N nodes, Pastry can route to the numerically closest node for a given key in less than

$$\left\lceil \log_{2^b} 2^N \right\rceil = \frac{N}{b}$$

steps under normal operation (b is a configuration parameter with typical value 4). For the purpose of routing, *nodeIds* and keys are thought of as a sequence of digits with base 2^b. Pastry routes messages to the node whose *nodeId* is numerically closest to the given key. This is accomplished as follows. In each routing step, a node tries to forward the message to a node whose *nodeId* shares a key with a prefix that is at least one digit (or b bits) longer than that of the present node. If no such node is known, the message is forwarded to the node whose *nodeId* shares a same length prefix with the key as the current node, but which is numerically closer to the key.

4.1.2 Routing Table

A node's routing table is organized into N/b rows with 2^b entries each. The 2^b-1 entries at row n of the routing table each refer to a node who shares the first n digits in the *nodeId*, but whose $(n+1)$-th digit has one of the 2^b-1 possible different values than that in the present node's id. The entries are divided into 3 categories: *valid*, *empty*, and *self*. Each row of the routing table includes one *self* entry corresponding to the own node's id. The other entries in this row are *valid* or *empty*, meaning that they do or do not contain other nodes' information. Thus, the appropriate choice of b requires a trade-off between the size of occupancy of the routing table and the maximum number of hops required between any pair of nodes.

4.1.3 Leafset

The leafset $\mathcal{L}$ is the set of nodes with the numerically closest $|\mathcal{L}|/2$ larger *nodeIds* and the $|\mathcal{L}|/2$ nodes with smaller *nodeIds* relative to the node's own id. Typical values for $|\mathcal{L}|$ are 2^b or 2^{b+1}. The leafset allows forwarding even in the case that the routing table is incomplete. Moreover, the leafset greatly contributes to the static resilience of the geometry [2].

4.2 Definition of the Parameters

In the following sections we describe the parameters used in our proposal.

4.2.1 Id Space and Network Size Exponents

The parameter N is the *id space size exponent*. Typical values are 128, 160, or 256 and the actual value depends on the hash algorithm. In the simulations in this paper, we consider a smaller network with $N = 16$. The *network size exponent* is expressed by n. This value represents that the number of the active nodes of the DHT network is between

2^{n-1} and 2^n. The possible range of n lies between 1 to N.

4. 2. 2 Routing Table Parameters

We define b as the *Pastry parameter* with possible values from 1 to N and R is the *routing table size* of a Pastry node with

$$R = \left\lceil \frac{N}{b} \right\rceil 2^b. \tag{2}$$

Another parameter is expressed by R_1, which is the *number of valid entries* in the routing table. Thus, R_1/R gives us the *routing table utilization.* The appropriate choice of R_1 depends on b and n, since the first n/b rows should be kept full, which means that this node gets enough routing information from the network.

$$R_1 \approx \frac{n}{b} \left(2^b - 1\right) \tag{3}$$

Figure 2 shows the relationship between b, R, and R_1. In this example, $N = 8$ and $nodeId = 01101100$. If $b = 2$ and $R = 4 \times 4 = 16$, *nodeId* is treated as a sequence of 2-bit digits 01-10-11-00. If $b = 3$, $R = 3 \times 8 = 24$, and *nodeId* is represented as 011-011-00. In each row, there is an entry (denoted as *self id*) with the same number as the corresponding digit of this *nodeId* and this entry is not used for routing purposes. When $n < N$, which is a common case with large N, only the first several rows are filled with neighbor information. R_1 depends on the network size and is an estimation of the overall network condition from the viewpoint of this node.

4. 2. 3 Timer Values

We define t_s as the *traffic sample interval.* Every t_s time interval, a node calculates its traffic and performs a selection of a new routing table. Furthermore, t_u denotes the *routing table and leafset update interval.* Every t_u time interval, a node validates the entries of the routing table and leafset through ping messages. If there is no response from the queried node, that entry is removed. Reasons for such a case may be a node's silent departure, node failure, or congestion. If the routing table or the leafset of a node are not full, update requests are sent to its neighbors. The neighbors' routing table or leafset are included in the update responses and merged by the node. Finally, t_q is defined as the *query interval.*

4. 2. 4 Traffic Metrics

Our traffic metrics are based on the *number of hop counts*, expressed by value h_i for query request i. Many factors affect the hop count, such as the network condition (size, churn, connection failures), the node state (routing table size, etc.), and the routing mode (iterative or recursive). We denote T_q as the *query traffic*, which is the sum of all h_i for all i in t_s, i.e., $T_q = \sum_{t_s} h_i$, and T_m is the *maintenance traffic.* For instance, if $n < N$, the routing table is not fully occupied. At every t_u interval, a node pings the existing entries in the table and receives acknowledgment messages, then sends update requests and receives their responses. If R_1 is kept constant during t_s (with fixed n and the leafset is full), the maintenance traffic T_m will be

$$T_m = 4\ (R_1 + |\mathcal{L}|)\ \frac{t_s}{t_u}.$$

The *total traffic* is then defined as the weighted sum of T_m and T_q and we use β to balance their influence. For a busy node, T_q may be much larger than T_m, whereas for an idle node T_q may be 0. If not stated otherwise, we consider in this paper $\beta = 0.5$, i.e., both traffic values contribute in equal proportion to the total traffic T.

$$T = \beta\, T_m + (1 - \beta)\ T_q$$

4. 3 Adaptation with Attractor Selection

In our model, the Pastry parameter b represents the node state. Each node uses this parameter to control the routing table size R and adapts to the changes in the network conditions. Some possible values of b may be pre-configured by an operator as especially desirable configurations, e.g., from $b = 1, \ldots, N/2$, which we define as set of attractors $\mathcal{A}$.

If the node chooses a small b, the values of R and R_1 become smaller, leading to a decrease of the rate of maintenance traffic over time. However, the query request through this node will now require more hops, because the node doesn't know as many neighbors as before and can't choose the next hop which is closer to the destination.

On the other hand, if the node uses a large b, it has larger R and R_1, leading to more maintenance traffic, but less query traffic. Our goal is to minimize the total traffic of the entire DHT network. So the activity function $g(\alpha)$ in Eqn. (1) must be defined inversely proportional to the total traffic T.

Let us now formulate an algorithmic description of the attractor selection problem. Note that this type of problem we are facing requires an entirely different way of formulation from previous applications of attractor selection (e.g. [6], [7], [18]), since only discrete values of a state value $attractor(t)$ can be used. Let $T(t)$ be the total traffic at time t and $T_{min}(t)$ be the minimum value of T over a sliding window, which stores several previous values of $T(t)$. If $T(t)$ exceeds the threshold of $(1 + \gamma)\ T_{min}(t)$, the activity is $\alpha = 0$ and the next attractor is selected uniformly randomly among all possible attractors. Otherwise, if the current selection is suitable, it is deterministically maintained and, thus, this formulation resembles the basic dynamic behavior of attractor selection.

$$attractor(t + t_s) = \begin{cases} attractor(t) & T(t) < (1 + \gamma)\ T_{min}(t) \\ \text{rand}\,(1, |\mathcal{A}|) & T(t) \geqq (1 + \gamma)\ T_{min}(t) \end{cases} \tag{4}$$

00xxxxxx	self id	10xxxxxx	11xxxxxx	$n = 2, R_1 = 3$
0100xxxx	0101xxxx	self id	0111xxxx	$n = 4, R_1 = 6$
011000xx	011001xx	011010xx	self id	
self id	01101101	01101110	01101111	

(a) $b = 2,\ R = 4 \times 4 = 16,\ \mathit{nodeId} = $ 01-10-11-00

000xxxxx	001xxxxx	010xxxxx	self id	100xxxxx	101xxxxx	110xxxxx	111xxxxx
011000xx	011001xx	011010xx	self id	011100xx	011101xx	011110xx	011111xx
self id	01101101	01101110	01101111				

(b) $b = 3,\ R = 8 \times 3 = 24,\ \mathit{nodeId} = $ 011-011-00

Fig. 2 Illustration of Pastry routing table with parameter b, N and n

5 Simulation Results

In order to show the efficiency of our proposal we conduct simulation experiments and compare the results with the standard Pastry implementation. In this section, we first describe the simulation environment and then present our comparative simulation evaluation.

5.1 Description of the Simulation Environment

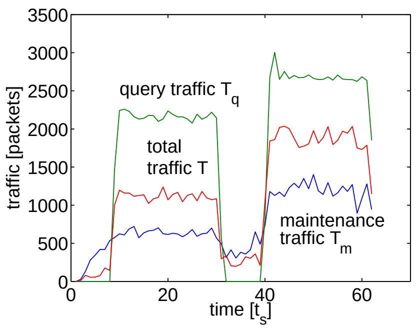
As simulator for DHT we use Overlay Weaver, which is an overlay construction toolkit written in Java. It enables overlay designers to implement a structured overlay algorithm with only a few hundreds of lines of code and to improve it rapidly by iterative testing on a single computer. The structure in this toolkit is composed of three parts: *routing driver*, *routing algorithm*, and *messaging service*. This decomposition enables an implementation of a number of well-known overlay algorithms with only minor modifications of the code. The available toolkit currently contains implementations of Chord, Kademlia, Koorde, Pastry, and Tapestry. Additionally, this decomposition allows multiple implementations of the routing driver and the toolkit provides iterative and recursive routing drivers.

We used the key-based routing layer by Dabek et al. [1] and built an enhanced scenario generator. In this generator, each node's online time t_{on} and offline time t_{off} follow exponential distributions and these two parameters control the network size with a mean of $2^n = 2^N\, t_{\text{on}}/\,(t_{\text{on}} + t_{\text{off}})$ nodes and t_{on}/t_s is the node's life time in the unit of the sample interval.

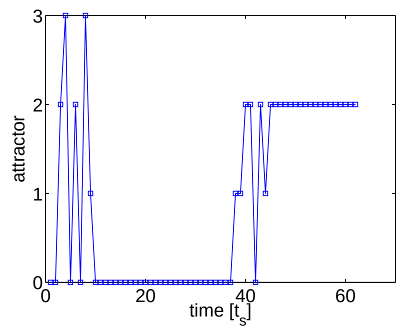
5.2 Discussion of the Simulation Results

The following figures are simulation results of a single scenario, where Node 0 is the bootstrap node that introduces other nodes to the DHT network. All the *nodeId* are uniformly distributed and Node 0 uses our modified Pastry routing algorithm with attractor selection based on (4).

We define 4 attractors (0 to 3) that are represented by the value of $b = 1, \ldots, 4$. All other nodes in the DHT operate with the standard Pastry with the fixed parameter $b = 2$, i.e., they stay at attractor 1 without any adaptation. Furthermore, we use the parameters $\beta = 0.5$, $\gamma = 0.2$, $t_s = 10$s,



(a) Traffic of Node 0



(b) Attractor of Node 0

Fig. 3 Simulation results of our prosal

$t_u = 1$s, $t_q = 10$ms. So in each sample interval, Node 0 processes 10 routing table updates and 1000 queries. The unit of traffic is given in packets.

Initially, 63 nodes were introduced by Node 0 in the time interval $[0, 10]$. These nodes form a static network without any joining or leaving nodes. Then, Node 0 queries the keys with uniform distribution and accumulates the hop counts. From time 30 to 40, further 192 nodes join the network. In this 256-node network, nodes know the existence of more neighbors (with increasing R_1) and feel more crowded (with increasing h_i). After the network is stable again, Node 0 continues performing queries until time 60.

Figure 3(a) shows the maintenance traffic, the query traffic, and the total traffic of Node 0. The unit on the x-axis is given in t_s. The traffic of Node 0 in the larger network (between time $[40, 60]$) is higher than that in the small network

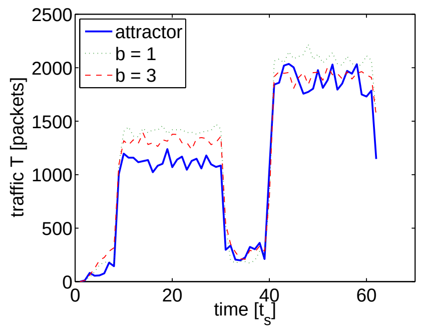


Fig. 4 Comparison of adaptive and fixed parameter settings

(between time [10, 30]). More query traffic T_q means that Node 0 should use a larger routing table and more maintenance traffic T_m means that Node 0 is currently using a larger routing table. After some fluctuations (time intervals from [0, 10] and [30, 40]), the traffic becomes stable in a static network.

In Fig. 3(b), we see how Node 0 chooses the attractors. In time intervals [0, 10] and [35, 45], it performs a random selection due to the fluctuations of the traffic. The selection of the new attractor is performed at time 35, following at a slight delay after the node joinings (time 30). In the other time, Node 0 remains at either attractor 0 or 2, depending on which attractor results in the least traffic.

Figure 4 is the comparison of the proposed modification to Pastry with attractor selection with standard Pastry. The curves show the traffic of Node 0 in the same scenario, but compares it with different parameter settings (constant $b = 1$ and $b = 3$). In most of the time, the traffic of our adaptive proposal is below that of Pastry with constant routing table size. It should be noted that the traffic values are rather close, as we selected nearly optimal settings for constant b values to which our proposal automatically adapts.

6 Conclusion and Outlook

In this paper, we proposed the application of attractor selection for self-adaptively configuring the routing table size in a DHT network. Based on Pastry we proposed a mechanism that minimizes the traffic of the whole network. With our approach, a node self-adaptively changes its routing table size based on the query hop counts and maintenance traffic it obtains only by local communication with its neighboring nodes. Simulation results confirmed that our method is capable of adaptively reducing the traffic of the whole network.

As part of our future work, we plan to examine the state consistency, load balance, and convergence time in more detail, as well as consider alternate forms for the definition of the attractors. Furthermore, more detailed performance studies are required to highlight the benefits of our proposal over conventional DHT strategies in the presence of churn.

Acknowledgments

This research was supported in part by “Global COE (Centers of Excellence) Program ” of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, February 2003.

[2] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM*, August 2003.

[3] M. F. Kaashoek and R. Karger. Koorde: A simple degreeoptimal distributed hash table. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[4] A. Kashiwagi, I. Urabe, K. Kaneko, and T. Yomo. Adaptive response of a gene network to environmental changes by fitness-induced attractor selection. *PLoS ONE*, 1(1), 2006.

[5] J. Ledlie and M. Seltzer. Distributed, secure load balancing with skew, heterogeneity, and churn. In *Proc. IEEE INFOCOM*, March 2005.

[6] K. Leibnitz, M. Murata, and A. Nakao. Biologically-inspired path selection scheme for multipath overlay networks. In *Proc. ISABEL*, Aalborg, Denmark, October 2008.

[7] K. Leibnitz, N. Wakamiya, and M. Murata. Biologically inspired self-adaptive multi-path routing in overlay networks. *Commun. ACM*, 49(3):62–67, 2006.

[8] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proc. IEEE INFOCOM*, March 2005.

[9] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[10] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. IPTPS*, 2002.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, August 2001.

[12] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proc. USENIX Technical Conference (USENIX '04)*, Boston, MA, June 2004.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[14] H. Shen and C.-Z. Xu. Elastic routing table with provable performance for congestion control in dht networks. In *Proc. ICDCS*, 2006.

[15] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constantdegree p2p overlay network. *Performance Evaluation*, 63(3):195–216, March 2006.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, August 2001.

[17] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE JSAC*, 22(1), 2004.

[18] 若宮直紀, ライプニッツ賢治, 村田正幸. 生物の適応性・頑健性に学ぶ：自己組織型ネットワーク設計手法・制御技術. 電子情報通信学会誌, 91(10):870–874, 2008.