Procedia Information
Technology & Computer Science

Vol 03 (2013) 000-000

4th World Conference on Information Technology 2013

Robust and lightweight routing with attractor selection

Naotaka Onzuka*, Naoki Wakamiya, and Masayuki Murata, Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan

Suggested Citation:
N. Onzuka, N. Wakamiya, and M. Murata, "Robust and lightweight routing with attractor selection", *AWERProcedia Information Technology & Computer Science*. [Online]. 2013, 00, pp 000-000. Available from: www.awer-center.org/pitcs

Received date; revised date; accepted date. [19 January, 2012]
Selection and peer review under responsibility of Prof. Dr. Hafize Keser.

Abstract

The rapid growth of information networks in size, complexity, and dynamics makes traditional and conventional mechanisms unsuccessful and unfeasible. As an example, OSPF requires complete and consistent information about the whole topology. In addition, its scalability is limited for the high computational complexity. Because of them, it is not possible to perform routing adaptive to network conditions by using dynamic link metrics, e.g. delay and utility. In our former research, we adopted a biologically-inspired nonlinear model, called attractor selection, to realize a highly scalable, robust, and adaptive routing mechanism. It imitates adaptive gene expression of bacteria cells, which autonomously and adaptively synthesize nutrients in accordance with the environmental conditions. The proposal, designed for reactive routing in mobile ad-hoc networks, required the computational cost in the order of N and exhibited more robust and adaptive behavior than traditional mechanisms. Therefore, in this paper, we take the same approach to realize robust and adaptive routing in a large-scale wired network. However, routing in a wired network must be proactive rather than reactive. As such, each node has to establish and maintain paths to all the other nodes, which results in the considerable amount of control overhead. Furthermore, in facing to a failure, it takes time to recover. To tackle the problems, we propose methods of overhead reduction and fast recovery. We compare our proposal with OSPF from viewpoints of computational complexity and show that our proposal has lower computational complexity and communication overhead than OSPF while achieving the robustness as high as OSPF.

Keywords: Routing, biological model, control overhead, computational complexity, robustness;

* ADDRESS FOR CORRESPONDENCE: Naotaka, Onzuka, Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
E-mail address: n-onduka@ist.osaka-u.ac.jp / Tel.: +81-6-6879-4542

1. Introduction

As size and complexity of information networks increase, traditional network technologies would become unsuccessful and unfeasible in the near future [1]. For example, Open Shortest Path First (OSPF) , which is a standard Inter Gateway Protocol (IGP), suffers from high computational complexity ranging from $\mathrm{O}(E + N\log N)$ to $\mathrm{O}(N^2)$, where N and E are the number of nodes and links, respectively [2]. Furthermore, it introduces high control overhead to maintain up-to-date information about the whole network topology at all nodes. Because of them, routing adaptive to dynamically changing conditions in a large scale network is not possible with OSPF.

In our former research such as [4, 5], we proposed a highly scalable, robust, and adaptive routing mechanism, called MARAS, for MANETs (Mobile Ad-hoc Networks) by adopting a biologically-inspired nonlinear model, called an attractor selection model [3]. The model imitates adaptive gene expression of bacteria cells. A cell seems to autonomously and adaptively select nutrients to synthesize to compensate nutrients missing in the environment and improve its growth rate. In MARAS, we regard the selection of nutrients to synthesize as selection of a next-hop node in hop-by-hop packet forwarding. In addition, MARAS uses dynamic measures to evaluate the quality of path, e.g. path length, delay, and utility, to accomplish adaptive routing. In addition to its higher robustness and adaptability than conventional routing mechanisms, the computational complexity of MARAS is only $\mathrm{O}(N)$ for a node to update routing information for all other $N-1$ nodes. Therefore, we can expect highly robust, adaptive, scalable, and computationally inexpensive routing in a large-scale and complex wired network by using the attractor selection model.

In this paper, we propose a novel routing mechanism for wired networks based on MARAS. In application to routing in wired networks, we need to overcome two impediments, they are, high control overhead and long recovery time. First, on the contrary to reactive routing in MANETs where a path is established between a pair of nodes on demand, paths to all other nodes must be proactively established and maintained independently of existence of communication needs in routing in a wired network. Although an attractor selection-based routing mechanism does not require complete information about the whole network, establishment and maintenance of $N \times (N-1)$ paths incur non-negligible control overhead. Therefore, we consider a method to reduce the overhead by allowing nodes to reuse control messages exchanged between other nodes.

Second, our proposal is robust and adaptive to changes as bacteria are, but the attractor selection model needs long time to reach a stable state. It results in long-lasting deterioration of performance, i.e. delivery ratio, after a failure of node or link. To tackle the problem, we introduce a method to shorten the recovery time by resolving a loop which is occasionally formed during path exploration and disturbs fast recovery after a node failure.

The rest of paper is organized as follows. First, in section 2, we briefly describe the attractor selection model. Next, in section 3, we explain our attractor selection-based proactive routing scheme for wired networks. Then, in section 4, we evaluate performance of our proposal by the simulation. Finally, in section 5, we conclude the paper and remark future work.

2. Attractor selection model

The attractor selection model is in the form of nonlinear stochastic differential equations derived from autonomous and adaptive behavior of *E. coli* cells in the dynamically changing environmental nutrient condition [3]. The authors genetically modified an *E. coli* cell to have a metabolic network consisting of two mutually inhibitory sequences of chemical reactions corresponding to synthesis of two different nutrients. In their experiments, a cell seems to appropriately select which nutrient to synthesize in accordance with the amount of nutrients in the environment. When one of the nutrients becomes scarce in the environment, the protein concentration activating a sequence for the missing nutrient eventually increases. Then, a cell stably maintains the state, i.e. gene expression, because it can compensate the missing nutrient and grow well.

In [3], the model describing this behavior of protein concentrations m_1 and m_2 is formulated as ,

$$\frac{dm_1}{dt} = \frac{syn(\alpha)}{1+m_2^2} - ded(\alpha)m_1 + \eta_1\,, \qquad (1)$$

$$\frac{dm_2}{dt} = \frac{syn(\alpha)}{1+m_1^2} - deg(\alpha)m_2 + \eta_2\,, \qquad (2)$$

where $syn(\alpha)$ and $deg(\alpha)$ are functions of the rate coefficients of protein synthesis and degradation, respectively. Both of them depend on α which is called an *activity* representing the cell activity or cell volume growth. Terms η_1 and η_2 are independent white noise in gene expression.

The dynamics of a nonlinear system following the above equations has two stable states, i.e. attractors, where $m_1 > m_2$ or $m_1 < m_2$. When the activity is high, the dynamics is dominated by the first two terms of the right-hand side of equations, which describe deterministic entrainment to push a system toward one of attractors and let it stably stay there. On the contrary, when the activity is low, a system takes stochastic behavior by being driven by the noise terms. When it occasionally approaches an attractor suitable for the current condition, the activity slightly increases. The increase strengthens the influence of deterministic dynamics, which results in further increase in the activity as the driving force toward the attractor. Therefore, the attractor selection model enables selection of a suitable attractor among possible stable conditions by switching stochastic behavior and deterministic behavior with mediation of activity, i.e. feedback.

3. Attractor selection-based proactive routing scheme for wired networks

3.1. Basic mechanisms

To accomplish robust and lightweight routing in wired networks, we extend our previous proposal called MARAS [5]. When a node newly communicates with another node in a MANET, MARAS first establishes an initial path between them using a mechanism similar to AODV, i.e. flooding [6]. Then, data packets are forwarded over the established path. Because of instability of radio signals and node movement, a path is not static or deterministic. Routing tables are dynamically updated based on the attractor selection model during communication. The activity of a path, representing the goodness of the current path, is derived from the transmission rate [4] or path length [5] using feedback packets sent back from a destination node.

On the contrary, in the case of proactive routing, all nodes need to establish and maintain paths to all the other nodes in a wired network. For this purpose, each node periodically sends a route control message to each of the other nodes in our proposal. We hereafter call an origin of a route control message a source node and its destination a destination node. Once a route control message reaches a destination node, a feedback message is sent back to the source node along the reverse path. In the following we collectively call a route control message and a feedback message control messages. In this paper, we consider one-way delay as a measure of the goodness of a path. By exchanging control messages between a source node and a destination node, the source node and intermediate nodes on the path calculate the one-way delay to the destination node. Then, an activity is derived and the corresponding routing table is updated using the attractor selection model at each node.

3.1.1. Routing table

A node maintains a routing table for each of the other nodes and a neighbor table. A neighbor table is updated by exchanging a Hello message at regular intervals of $T_{hello} (> 0)$ [s]. A routing table for destination node d consists of routing vector $\overrightarrow{m_d}$ and activity α_d. A routing vector $\overrightarrow{m_d}$ is an M-dimensional vector as $\overrightarrow{m_d} = (m_{d,1}, \dots, m_{d,j}, \dots, m_{d,M})$, where M is the number of neighbors. $m_{d,j}$ is a non-negative scalar value called a *state value* which represents the goodness of neighbor j as a next-hop node for destination node d. State values are updated on reception of a feedback message. If a destination node is one of neighbors, its state value is fixed to 1 and the others are 0. A node forwards data packets to its neighbor which has the maximum state value in a routing vector. Activity α_d is a scalar value which represents the goodness of the current path. It is derived by comparing the latest one-way delay to the minimum of the last W measurements, called a feedback window of size W.

When a node does not receive a Hello message from neighbor j for $4 \times T_{hello}$ for some reasons, e.g. failures, the corresponding state values $m_{*,j}$ are removed from a routing table. If node j is a next-hop node to a certain destination d, activity α_d is initialized to 0. Then, the node updates routing vector $\overrightarrow{m_d}$ to appoint one of remaining neighbors as a new next-hop node. Furthermore, the corresponding feedback window is cleared to forget invalid historical information.

3.1.2. Path establishment and maintenance

At the very beginning, an initial path is established for each of the other nodes by using a mechanism similar to AODV. On an initial path, a node sets a state value of the next-hop neighbor as β and state values of the others as 0. An activity value is set at 1, because an initial path is the shortest.

Then, paths are maintained by exchanging control messages. On reception of a feedback message at a source node, a one-way delay is first obtained, next an activity is derived, and then a routing vector is updated. To measure a one-way delay to a destination node, a source node generates a *route control message* at regular intervals of T [s] using a timer for each of destination nodes. When a route control messages is emitted, a source node remembers time of emission, e.g. t_1, and sets the corresponding timer to T [s]. When the timer expires, it sends a next route control message. A route control message contains a source address, a destination address, and an address list of intermediate nodes which is empty at the beginning. It is sent to a destination node along the established path.

When an intermediate node receives a route control message, it first checks whether it has already received the message or not based on an address list contained in the receiving message. If it has, it silently discards the message. Otherwise, it adds its own address to an address list and forwards the message to a neighbor based on a routing table. It also remembers sent time of message, e.g. t_2. On reception of a route control message, a destination node sends a feedback message. A feedback message has a destination address, a source address, receipt time, e.g. t_3, and the address list of intermediate nodes of the route control message. A feedback message reaches a source node traversing a reverse path based on an address list. When a source node or an intermediate node receives a feedback message, it calculates a one-way delay to the destination node by subtracting t_1 or t_2 from t_3. They keep the past W one-way delays to calculate the activity.

3.1.3. Routing table update

When a node receives the h-th feedback message from destination node d, it first calculates instantaneous activity $\alpha'_d(h)$ $(0 \le \alpha'_d(h) \le 1)$ by using one-way delay $w_d(h)$ derived from the h-th feedback message as follows.

$$\alpha'_d(h) = \frac{\min_{0 \le k \le W-1} w_d(h-k)}{w(h)} \qquad (3)$$

Furthermore, to mitigate path flapping caused by instantaneous fluctuations in measured delays, we derive activity $\alpha_d(h)$ by using a moving average with coefficient c $(0 \le c \le 1)$ as follows.

$$\alpha_d(h) = \begin{cases} \alpha'_d(h), & \text{if } \alpha_d(h-1) \le \alpha'_d(h) \\ c\alpha'_d(h) + (1-c)\alpha_d(h-1), & \text{otherwise} \end{cases} \qquad (4)$$

Using $\alpha_d(h)$ and $\overrightarrow{m_d}(h-1)$, routing vector $\overrightarrow{m_d}(h)$ is updated as follow.

$$m_{d,j}(h) = m_{d,j}(h-1) + \frac{syn(\alpha_d(h))}{1+\max_{1\le j\le M}(m_{d,j}(h-1)^2) - m_{d,j}(h-1)^2} - deg(\alpha_d(h))m_{d,j}(h-1) + \eta \quad (5)$$

Here $syn(\alpha) = \alpha(\beta\alpha^{\gamma} + \phi^*)$ and $deg(\alpha) = \alpha$. η is the Gaussian noise with average 0 and variance 1.

When a source node does not receive a feedback message from a destination node until a next route control message is sent, it would indicate a failure or serious congestion. Therefore, the corresponding activity decays by multiplying parameter D (≥ 0). Then, routing vector $\overrightarrow{m_i}$ is updated.

3.2. Overhead reduction method

In the above-mentioned basic mechanisms, control messages are periodically exchanged between all pairs of nodes. This apparently incurs non-negligible control overhead. It spoils the scalability of our proposal, whereas ours benefits from the low computational complexity. Therefore, we introduce a method to reduce the control overhead which allows reuse of a pair of a route control message and a feedback message by nodes on a path to update multiple routing tables.

3.2.1. Extension of control messages

We extend a route control message to have two more fields: receipt time at intermediate nodes and sent time at a source node and intermediate nodes. Furthermore, a feedback message is also extended to have two more fields: sent time of the corresponding route control message at a source node and intermediate nodes and its receipt time at intermediate nodes. Since in the remainder of this section we only consider extended control messages, we omit "extended" in referring to them.

3.2.2. Behavior of nodes on receipt of extended messages

By receiving a feedback message, a source node knows time when the route control message was received at a destination node and intermediate nodes. Therefore, it can obtain one-way delays to all of a destination node and intermediate nodes by subtracting the emission time. Then, routing tables for those nodes can be updated. Consequently, the source node defers emission of a next route control message for those intermediate nodes by setting their timers to $T' = T + \max(T/100, J)$. J is the jitter of measured one-way delays defined as $\max_{0\le j\le W-1} d(h-j) - \min_{0\le j\le W-1} d(h-j)$.

When a destination node receives a route control message, it knows time when the route control message was sent from the source and intermediate nodes and it can derive one-way delays from them. Assuming symmetric delays, those one-way delays can be regarded as delays from the destination node to them. Consequently, the destination node can update routing tables for the source and intermediate nodes. It also defers emission of route control messages for them by T' [s].

When an intermediate node receives a route control message, it can derive a one-way delay from the source node and intermediate nodes on a path from the source node to itself, i.e. upstream nodes. Assuming symmetric delays again, routing tables for the source and upstream nodes can be updated at this time. In addition, on reception of a feedback message, an intermediate node can obtain one-way delays to the destination and downstream nodes. Using these one-way delays, it updates routing tables for them. In both cases, an intermediate node sets the corresponding timers to T' [s].

3.3. Fast recovery method

Our proposal is adaptive to changes and robust to failures. However, it sometimes needs long time to find a new valid path. Specifically, our proposal suffers from looping. As stated in 3.1.2, each node forwards a packet to a neighbor with the largest state value. When a failed node is a next-hop node for a certain destination node at a neighbor, the neighbor sends a route control message to another neighbor to find an alternative path. If the selected neighbor occasionally has a path that did not contain the failed node, it becomes a new path. Otherwise, the neighbor forwards the route control message to another neighbor. Unless the message encounters a node which has a valid path, it wanders about in search of the destination node. During the random walk, a route control message would arrive at a node which it has visited. A loop is formed at this time.

Looping prevents a route control message from reaching a destination node and a node from receiving a feedback message. If a state value of another neighbor becomes the maximum after a control interval, a next-hop node changes and there arises the possibility of finding a non-looping path. However, such stochastic trials prolong recovery time. Therefore, we propose a fast recovery method consisting of two phases: loop detection and loop resolution. We should note that during these phases, a source node keeps periodically sending route control messages based on a routing table.

3.3.1. Loop prevention message and extended routing table

For purpose of illustration, we hereafter assume that node t tries to recover a path to node d. Now node t detects a failure of a neighbor which was a next-hop node for node d. It first removes the failed node from routing tables and a neighbor table and next evaluates the attractor selection model. Then, node t checks whether a path to node d via a new next-hop node, say, node u, forms a loop or not by sending a new kind of message, called a loop prevention message.

A loop prevention message has sender address t, destination address d, a flag to distinguish between detection mode and avoidance mode, which correspond to the loop detection phase and the loop resolution phase respectively. A loop prevention message used to detect a loop in a path is in the detection mode. In addition to the above-mentioned fields, it maintains an address list of visited nodes. It is called an *f-list* and updated when being forwarded. In the avoidance mode for loop resolution, a loop prevention message uses an *l-list* instead of an f-list. An l-list is an address list of nodes that does not have a neighbor with a valid path to a destination node. A loop prevention message in the avoidance mode additionally has an address list of neighbors of node t, called an *n-list*. A loop prevention message in the avoidance mode is used to ask a neighbor to find a valid path. With an l-list and an n-list, an asked node can select an untested node to find a path.

Furthermore, we extend a routing table to have an additional flag field, called a loop flag, for each of destination nodes. A loop flag consist of M bits, where M is the number of remaining neighbors. Each bit indicates whether selection of the corresponding node results in a loop or not. When node t detects a loop on selecting neighbor u, a bit for node u is set and the corresponding state value is fixed to 0 not to select node u as a next-hop node.

3.3.2. Loop detection phase

On detection of a failure of a next-hop node or reception of a loop prevention message in the avoidance mode from a neighbor, the loop detection phase is initiated at node t. Assume that node t newly selects neighbor u as a next-hop node. Node t first sends it a loop prevention message in the detection mode. On reception, node u adds its address to an f-list and forwards the message to a next-hop node. A node receiving a loop prevention message first checks whether its address is in an f-list or not. If not, it adds its address to the f-list and further forwards the message to a next-hop node. Otherwise, it silently discards the message. If destination node d successfully receives a loop prevention message, it silently discards the message.

When node t does not receive a loop prevention message it sent for a certain amount of time, e.g. four times as long as the maximum of the previous one-way delays in this paper, it considers that there is a valid path to destination node d. Then, node t resumes normal operation. Otherwise node t detects a loop by receiving a loop prevention message it sent. Node t first sets a bit for node u in a loop flag for node d. Next, routing vector $\overrightarrow{m_d}$ is updated. Then, node t sends a loop prevention message, which is in detection mode, to a new next-hop node. The process is repeated until node t

finds a valid path or all bits of a loop flag field are set. In the latter case, node t unsets all bits of the loop flag field and moves to the loop resolution phase.

3.3.3. Loop resolution phase

In the loop resolution phase, node t generates a loop prevention message, which is in avoidance mode. If node t detects a failure by itself, an l-list has only its own address. In case it was asked to find a path by receiving a loop prevention message in the avoidance mode from a neighbor, an l-list in the received message is succeeded by a new loop prevention message and an address of node t is added. Node t sends a loop prevention message to randomly selected neighbor u, which is not in an l-list, to ask it to find a valid path. Then, node t resumes normal operation.

On receiving a loop prevention message, node u first sets bits of a loop flag for node d. Specifically, bits for a neighbor from which it received the message, a neighbor which is currently selected as next hop, and nodes in an n-list, because they do not have a valid path. Next, a feedback window is cleared. Then, activity α_d is set to 0 and routing vector $\overrightarrow{m_d}$ is updated. Finally node u moves to the loop detection phase and performs the process in 3.3.3 to find a valid path.

4. Evaluation

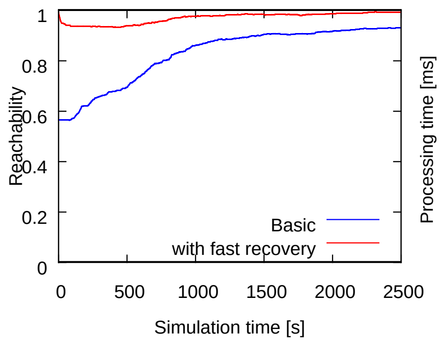
4.1. Simulation environment

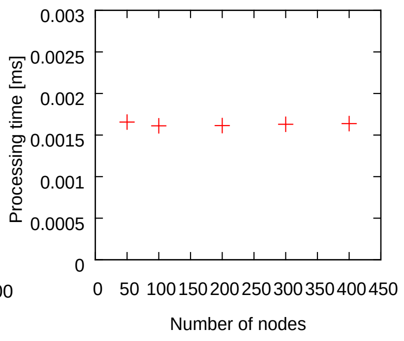
We use OMNet++ [7] version 4.1 in our evaluation. We generate network topology by Waxman model [6]. The propagation delay of each link is set at 1 [ms]. In evaluation of the overhead reduction method and the fast recovery method, the numbers of nodes and links are 100 and 200 respectively. In comparative evaluation, we change the number of nodes from 50 to 400. As for OSFP, we use a code implemented by INET framework with reference to RFC 2328. We extend the code by adding standardized timers *SPF Delay* and *SPF Hold Time*, which are used to reduce the number of calculations on topology changes. Parameters are set as $\beta = 10$, $\gamma = 3$, $\phi^* = 1/\sqrt{2}$, $c = 0.1$, $W = 20$, $D = 0.1$, $T_{hello} = 10$ [s], and $T = 100$ [s] for our proposal. We use default values for OSPF as Hello Interval of 10 [s], LSA lifetime of 30 [min], SPF delay of 5 [s], and SPF Hold Time of 10 [s].

In our proposal, a route control message and a feedback message have common header fields constituting of a packet type field of 1 [byte], a packet length field of 2 [byte], and a reserved field of 1 [byte]. In addition to them, both of a timestamp filed and an address field has a length of 4 [byte]. As we assume a processing delay in forwarding a control message is negligible, the sent time and the receipt time at intermediate nodes can be represented by one timestamp. Therefore, the total size of a route control message without extension is $4(H + 3)$ [byte] on arriving at a destination node where H is the number of intermediate nodes. With extension, the size increases to $4(2H + 4)$ [byte]. Regarding a feedback message, sizes are $4(2H + 5)$ and $4(H + 4)$ [byte] with and without extension, respectively. The average of size of a route control message and a feedback message are 42.3 [byte] and 45.9 [byte] respectively in the simulation with 100 nodes and 200 links.

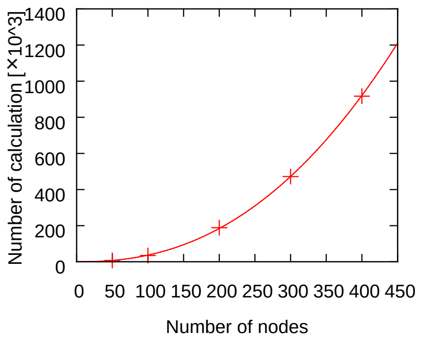
4.2. Evaluation of overhead reduction method

In this section, we verify the overhead reduction method. The control overhead is defined as the total size of control messages emitted by all nodes. Hello messages are inherent and they are not taken into account. The control overheads in a period of T are 236 [Kbyte] and 107 [Kbyte] for cases of without and with the method. Therefore, the control overhead is reduced by about 55%. Regarding the number of control messages, the percentage of reduction is about 68%. A reason of the difference is the increased size for extended fields. Reduction in the size and the number are helpful in saving bandwidth of a link and processing power of a node, respectively.





(a) Processing time



(b) The number of calculation

Figure 3. Reachability of our proposal

Figure 4. Processing time

4.3. Evaluation of fast recovery method

In In this section, we verify the fast recovery method. After routing converges, we removed a randomly selected intermediate node on a path between a certain pair of a source node and a destination node. We use randomly generated 100 scenarios with different topology, a source and destination pair, and a failed node. In all of scenarios, there is at least one valid path after node removal. For each of scenarios, we conduct 10 simulation runs by chancing a random seed.

We consider the reachability $P_R\ (t)$ which is defined as follows.

$$P_R\ (t) = \sum_{k=1}^{S_n} \left(P_k(t)\right) / S_n\ , \qquad (6)$$

$$P_i(t) = \begin{cases} 1, & \text{if source has valid path to destination at } t \text{ in the } i\text{th simulation} \\ 0, & \text{otherwise} \end{cases}, \qquad (7)$$

where S_n is the number of simulations, that is, 1000 in this evaluation.

In Fig. 3, we show the reachability of our proposal with and without the fast recovery method. In the figure, simulation time 0 is when a neighbor of a removed node detects the failure. It is apparent that our proposal with the fast recovery method recovers a failure faster and achieves the higher reachability than the basic proposal. A reason that the reachability with the method decreases in the first few seconds is the stochastic behavior of the attractor selection model. Right after a failure of a next-hop node, state values of remaining neighbors are low and similar to each other. Therefore, even if a newly selected neighbor has a valid path, it would be occasionally overtaken by another neighbor, which does have a valid path, due to a noise term. The lowest reachability is about 0.931 at around 750 [s]. Although the stochastic behavior has such an adverse effect, it acts as a driving force to find a better path. As shown, the reachability eventually increases by finding a reachable and shorter path.

4.4. Comparative evaluation

4.4.1. Computational overhead

First, we evaluate the computational overhead of our proposal and OSPF. As the computational overhead, we consider the processing time required for a node to update routing information by calculating the attractor selection model in our proposal and the Dijkstra algorithm in OSPF. We use a computer equipped with Intel Core i7-2600 3.4 [GHz] and the memory of 16 [Gbyte].

In Fig. 4(a), we show the computational overhead per path in our proposal. It is shown that the processing time is not affected by the number of nodes. It is because a node evaluates one equation to update an activity and M equations to update state values per destination node. However, the processing time shown in the Fig. 4(a) is for a single destination whereas the Dijkstra algorithm gives the shortest paths to all other nodes. In Fig. 4(b), we show the number of calculations in a period of T in the whole network in our proposal. A solid line is a fitting line by least squares fitting and it is in the order of $O(N^{2.3})$. Therefore, the number of calculations per node is in the order of $O(N^{1.3})$.

In Fig. 5, we show the total processing time of our proposal and OSPF in T [s]. For our proposal, we multiply the processing time in Fig. 4(a) and the number of calculations per node derived from Fig. 4(b). Regarding OSPF, we consider a node calculates the Dijkstra algorithm once per T [s]. In Fig. 5, solid lines are fitting lines. It is shown that our proposal suppresses the increase in processing time due to network growth from $O(N^2)$ to $O(N^{1.3})$.

4.4.2. Control overhead

In this section, we compare our proposal and OSPF from a viewpoint of the control overhead in a single control interval. In OSPF, the control overhead takes into account DD, LSR, LSU, and LSAck messages used in LSA exchanging. Figure 6 shows that the control overhead is in the order of $O(N^2)$ in both of our proposal and OSPF, but our proposal can suppress the overhead to one sixth of that of OSPF. When we consider a large-scale network consisting of, for example, 1000 nodes [9], the control overhead of OSPF grows to 5.37 [Mbyte] while that of our proposal is 865 [Kbyte] in the same interval. Furthermore, in a network of 10000 nodes, they are 537.8 [Mbyte] and 86.5 [Mbyte].

4.4.3. Robustness to node failure

In this section, we compare the robustness to a single node failure as in 4.3. Figure 7 shows that the reachability of OSPF drops to zero right after a failure. It is because OSPF waits for a period of SPF Delay of 5 [s] to evaluate the Dijkstra algorithm after detection of a change in network topology. After that, the reachability recovers to one at around 6 [s]. On the contrary, the reachability of our proposal is nearly one right after a failure. It slightly decreases once for the same reason explained in 4.3 and then gradually increases as shown in Fig. 3. Even though the reachability does not reach one in the figure, there is no pair of nodes that never has a valid path throughout a simulation run. We can improve the reachability by suppressing the effect of a noise term, but it sacrifices the ability of finding a better path. Investigation of the trade-off remains one of future work.

Furthermore, in Fig. 8, we compare the average length of paths established after a failure. The figure shows how much a path established by stochastic routing of our proposal is longer than the shortest path of OSPF. At the beginning, the difference is as high as 0.82. However, the difference eventually decreases to about 0.56, because our proposal finds a shorter path by stochastic searches.

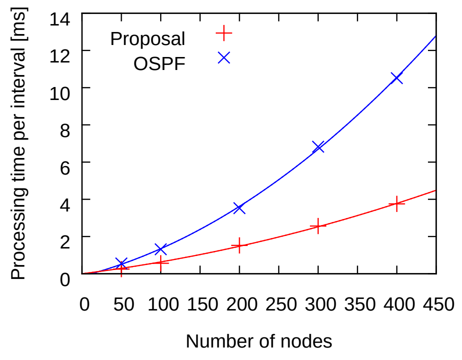


Figure 5. Processing time per interval

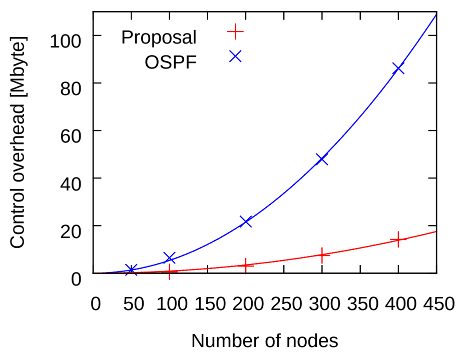


Figure 6. Control overhead

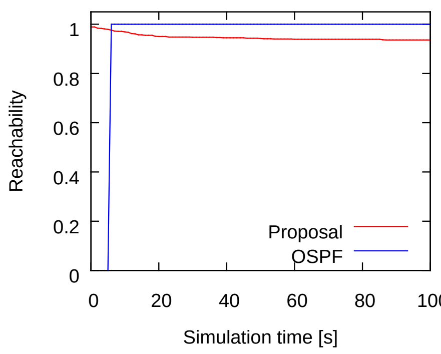


Figure 7. Reachability of our proposal and OSPF

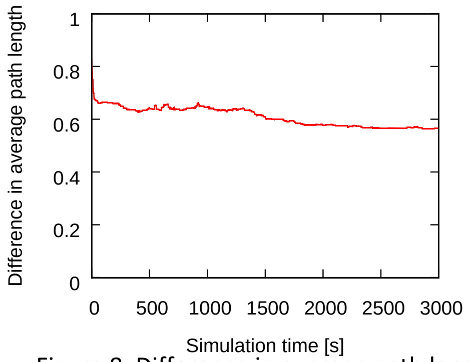


Figure 8. Difference in average path length

5. Conclusion

In this paper, we propose a proactive routing mechanism for wired networks based on the attractor selection model. With the overhead reduction method and the fast recovery method, our proposal can accomplish scalable, lightweight, and robust routing. Although our proposal is shown to be superior to OSPF, it suffers from slightly low reachability during failure recovery. The main reason is in a stochastic hop-by-hop routing mechanism, which originally comes from our previous research on MANET routing. As future work, we plan to consider multi-path routing. Specifically, each node adaptively selects one of alternative paths to a destination node in accordance with dynamically changing conditions by using the attractor selection model.

Acknowledgements

This research was partially supported by "R&D for fundamental technology for energy-saving network control compatible to changing communication status" in FY2013 of the Ministry of Internal Affairs and Communications, Japan.

References

[1] S. Paul, J. Pan, and R. Jain, “Architectures for the Future Networks and the Next Generation Internet: A Survey,” Computer Communications, vol. 34, pp. 2-42, Jan. 2011.
[2] M. Sniedovich, “Dikstra's algorithm revisited : the dynamic programming connexion,” Control and cybernetics, vol. 35, pp. 599-620, 2006
[3] 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, vol. 1, p. e49, Dec. 2006.
[4] K. Leibnitz, N. Wakamiya, and M. Murata, “A bio-inspired robust routing protocol for mobile ad hoc networks,” Computer Communications and Networks, vol. 16, pp. 321-326, Aug. 2007.
[5] N. Asvarujanon, K. Leibnitz, N. Wakamiya, and M. Murata, “Robust and adaptive mobile ad hoc routing with attractor selection,” in Proceedings of Adaptive and DependAble Mobile Ubiquitous Systems, July 2010.
[6] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing,” RFC 3561, pp. 1-37, July 2003.
[7] “OMNeT++ Network Simulation Framework”, http://www.omnetpp.org/
[8] B. M.Waxman, “Routing of multipoint connections,” IEEE Journal of Selected Areas in Communications, vol. 6, pp. 1617-1622, Dec. 1988.
[9] H. Tangmunarunkit, J. Doyle, R. Govindan, W. Willinger, S. Jamin, and S. Shenker, “Does AS size determine degree in as topology?”, SIGNOMM Computer Communication Review, vol. 31, pp. 7-8, Oct. 2001.