Improved Reinforcement Computing to Implement AntNet-Based Routing using General NPs for Ubiquitous Environments

Hyuntae Park¹, Byung In Moon², and Sungho Kang¹

¹ Department of Electrical and Electronic Engineering, Yonsei University, 134 Shinchon-dong, Sodamun-gu, Seoul, 120-749, Korea
radiob@yonsei.ac.kr, shkang@yonsei.ac.kr
² School of Electrical Engineering & Computer Science, Kyungpook National University, 1370 Sankyuk-dong, Buk-gu, Daegu, 702-701, Korea
bihmoon@knu.ac.kr

Abstract. In the ubiquitous convergence era, the traffic managements and quality of services will be made much of a role. Because traditional routing mechanisms are lacking scalability and adaptability, a kind of adaptive routing algorithm called AntNet has attracted the attention. AntNet is an adaptive agent-based routing algorithm that imitates the activities of the social insect. In AntNet, there are implementation constraints due to complex arithmetic calculations for determining a reinforcement value. Besides, a housekeeping core in network processors will be overwhelmed by increasing routing workload for a processing of agents. In this paper, we propose a new reinforcement computing algorithm to overcome these problems. This can be implemented efficiently on packet forwarding engines of conventional network processors. The simulation results show that the proposed AntNet is more adaptive and effective in the performance of the implementation than the original AntNet.

Keywords: AntNet, Adaptive Routing, Reinforcement Computing, Network Processor, Ubiquitous Environments.

1 Introduction

According to the advent of the ubiquitous convergence era, a unitary autonomous system will have more and more nodes and paths. The controls of an enormous network whether it is wired or wireless in the near future will become more difficult obviously. Besides, in order to support new various ubiquitous applications for a real-time multimedia, the traffic managements and quality of services are made much of a role. Accordingly, new paradigm for routing will be required in next generation networks. A kind of adaptive routing algorithm called AntNet has attracted the attention of researchers, among many different kinds of routing algorithms [1]. It is inspired by the adaptive and distributive behaviors of real ants to locate the shortest
Improved Reinforcement Computing to Implement AntNet-Based Routing using General NPs for Ubiquitous Environments

route from the nest (source node) to the food source (destination node) by depositing a chemical substance called *pheromone* on the trail [1].

In previous works, the efficiency of AntNet has been verified by comparison with state-of-art routing algorithms such as RIP and OSPF in dynamic traffic environments [2]. However, up until now, there have been few attempts to implement it in a real network, and it is not used yet for real network routing systems. If AntNet is implemented by software on a housekeeping core as a traditional approach, an overload of a housekeeping core for processing incoming agents with a line-speed become further burdensome. Therefore, our prospective view is that, in the future, the processing of an agent-based routing protocol is performed on packet forwarding engines instead of the housekeeping core in order to improve the processing of agents.

In other words, we expect that the routing in ubiquitous environments is a part of fast-path processing, not slow-path. A reinforcement computing is the most complex among the tasks of AntNet-based routing because it requires complex calculations. In this paper, the improving reinforcement computing on general packet forwarding engines is proposed to implement AntNet-based routing in real network environments.

Firstly, reinforcement in the original AntNet is briefly introduced in section 2. In the next section, we analyze constraints to implementing the original AntNet. The optimized AntNet algorithm developed to solve these indicated problems is presented in section 4. The performances of each algorithm are compared and evaluated in section 5. Finally, the conclusions of this paper are given in section 6.

2 The Reinforcement in the Original AntNet

AntNet algorithm, an approach to the adaptive learning of routing tables in communication networks, was first introduced in 1998 [1]. The concept of AntNet was derived from the adaptive behaviors of social insects such as ants, which behaviors are called *stigmergy*. Then, the proposed AntNet is introduced in [3]. However, because an improvement of reinforcement computing is insufficient in [3], we refer to the original reinforcement computing method in [1]. And, a detailed description of the operation of AntNet is not given here. According to the original AntNet, the equation of the reinforcement computing is as follows:

\[ r = c_1 \left( \frac{W_{\text{best}}}{T} \right) + c_2 \left( \frac{I_{\text{sup}} - I_{\text{inf}}}{(I_{\text{sup}} - I_{\text{inf}}) + (T - I_{\text{inf}})} \right) \]  \hspace{1cm} (1)

In the above Eq.(1), \( W_{\text{best}} \) is the shortest trip time experience by the ants traveling to the destination node, which is observed within the scope of the window size. This window size reflects the number of considered samples before resetting the \( W_{\text{best}} \) and it is assigned as a base of \( \eta \), which is weighed by the number of samples effectively giving a contribution to the value of \( \mu \) estimate.
\[ \begin{align*}
P_{nf} & \leftarrow P_{nf} + r(1 - P_{nf}) \\
P_{nf'} & \leftarrow P_{nf'} - rP_{nf} \\
, f & \in N, n \neq f
\end{align*} \] (2)

Eq.(1) is used to update a probabilistic entry in the routing table at the node \( k \) coming from the node \( f \). \( P_{nf} \) represents probabilistic value of taking the node \( f \) as the next neighbor node towards the destination; whereas \( r \) represents a positive reinforcement. Eq.(2) is used to update probabilistic entries in the routing table at the node \( k \) for taking a neighbor node other than the node \( f \) as the next node towards the destination. According to Eq.(1) and (2), \( P_{nf} \) is increased but \( P_{nf'} \) of other neighbor nodes is decreased at the point that the sum of \( P_{nf} \) and \( P_{nf'} \)'s is always 1.

3 Constraints of Implementation using NP

We can consider two ways for the implementation of the AntNet-based routing. One is to build AntNet by adding the extra hardware for AntNet routing. The other is to build the AntNet using existing network devices and systems.

In the former case, existing network systems should be changed to become suitable for conventional state-of-the-art routing algorithms. It is a heavy burden with much cost and time because a current communication network is already occupied massively. Moreover, we never know whether it can be used widely and can be available on any network environment. This is a problem of compatibility. According to [1], AntNet is not available on all network environments in spite of the goodness of AntNet in a dynamical network environment. Therefore, we have to implement the AntNet without the extra hardware.

The latter case also has a problem. In order to implement AntNet-based routing with compatibility and efficiency, we premise that AntNet is implemented on network processors. Furthermore, in order to reduce a burden of a housekeeping core in network processors, agent processing for AntNet is performed on packet forwarding engines instead of the housekeeping core.

Reinforcement computing as briefly described in the previous section is not very complex mathematically. However, general packet forwarding engines have a concise arithmetic unit for simple processing called parsing and comparison. Accordingly, if the reinforcement computing based on the original AntNet by itself is performed on packet forwarding engines instead of the housekeeping core, the performance of computing is worse. Nowadays, network devices for routing are mainly made of network processors which are the products of Intel, Agere Systems, Lextra, Sitera, Clearwater Networks and so on. Considering the architecture of each packet forwarding engine in these processors, packet forwarding engines that consist only of ALU and shifter are sufficient to perform packet forwarding functions [4][5][6][7]. Moreover, these only support an integer type processing even though AntNet needs floating point type processing for handling probabilistic values. Therefore, we proposed the improved reinforcement equation to implement AntNet on existing packet forwarding engines as it is.
4 The Proposed Algorithm

The excellent merit of the proposed algorithm is that the reinforcement is computed with simple arithmetic units. The proposed algorithm is induced as follows:

\[ r = \text{normal} \left[ \frac{b\text{Cost}}{\text{curCost}} \right] \]  

(3)

While several factors are induced to get a reinforcement value in the original AntNet, we only use the round trip time of the ants as Eq.(3). In Eq.(3), \(b\text{Cost}\) is the same as \(W_{\text{net}}\) in Eq.(1) and \(\text{curCost}\) is the same as \(T\) in Eq.(1). In Eq.(1), the first and most important term weighs the goodness of the current trip time compared to the best trip time. The second term is used to complement the first term [2]. Even though the second term is less influential than the first term in the performance, it takes more time to calculate than the first term. Therefore, the second term in Eq.(1) is neglected unavoidably. Although it is an approximation of the original AntNet, we can simplify the process to improve the efficiency of the implementation without a serious loss of adaptability.

In Eq.(3), a division mechanism is required for an arbitrary value. According to programming reference manuals of conventional network processors, a multiplication and a division are operated ineffectively by recursive additions or subtractions. Therefore, we propose Eq.(4) and Eq.(5) as an advanced revision of Eq.(3). These formulas are proposed heuristically based on the following rules:

\[ r' = 255 + (b\text{Cost} - \text{curCost}) \]  

(4)

\[ r = \text{normal} \left[ \frac{r' \cdot b\text{Cost}}{C_{b\text{Cost}} \cdot C_{\text{curCost}}} \right] \]  

(5)

* Rule 1: A relative term transforms into an absolute term.

In Eq.(3), the division presents the relation between a denominator and a numerator. Because a denominator is an arbitrary value, the division procedure consumes a lot of time for the conventional network processors which perform the division by recursive subtractions. Thus, we try to transform a relative term as the division into an absolute term as the subtraction. In order to find the effect of \(b\text{Cost}\), if \(\text{curCost}\) value is fixed, \(b\text{Cost}\) is positively in proportion to \(r'\). Therefore, \(b\text{Cost}\) must be the subtracted number in the subtractive term as the second term of Eq.(4) is presented. Because \(\text{curCost}\) is always equal to or larger than \(b\text{Cost}\), it is necessary to add any particular constant so that \(r'\) has a positive value. This particular constant is 255 as determined by a heuristic method. Finally, we can draw Eq.(4).

* Rule 2: Weighed values are required to complement the transformed absolute term, \(r'\)

The transformed absolute term, \(r'\) in Eq.(4) is inferior in adaptability to the original reinforcement \(r\). Considering the mutual relation between \(r'\) and the reinforcement \(r\), if \(b\text{Cost}\) and \(\text{curCost}\) are large values, the reinforcement \(r\) should have a large value.
regardless of \( r' \). On the other hand, if \( bCost \) and \( curCost \) are small values, the reinforcement \( r \) should have a small value. That is, even if \( r' \) is the same as in the above two cases, we should weight \( r' \) according to \( bCost \) and \( curCost \). In order to complement this, we propose that weighted values \( C_{bCost} \) and \( C_{curCost} \), are assigned by \( bCost \) and \( curCost \) for division. In addition, in order to simplify the processing and save the calculation time, we propose that weighted values are multiples of 2 specially. The relation to determine the weighted values are shown in Table 1.

<table>
<thead>
<tr>
<th>( bCost ) &amp; ( curCost )</th>
<th>( C_{bCost} ) &amp; ( C_{curCost} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (-) 7</td>
<td>2</td>
</tr>
<tr>
<td>8 (-) 15</td>
<td>4</td>
</tr>
<tr>
<td>16 (-) 31</td>
<td>6</td>
</tr>
<tr>
<td>32 (-) 63</td>
<td>8</td>
</tr>
<tr>
<td>64 (-) 127</td>
<td>10</td>
</tr>
<tr>
<td>128 (-) 255</td>
<td>12</td>
</tr>
<tr>
<td>256 (-) 511</td>
<td>14</td>
</tr>
<tr>
<td>512 (-) 1024</td>
<td>16</td>
</tr>
</tbody>
</table>

* Rule 3: The reinforcement \( r \) is directly proportional to \( bCost \) sensitively.

In rule 1, we confirm that the mutual relation between \( r' \) and \( bCost \) is proportional, when we assume that \( curCost \) value is fixed. We examine the mutual relation between the reinforcement \( r \) and \( bCost \). Through various experiments, we learned that this relation is directly proportional. In addition, we verify that this relation is very sensitive. That is, the reinforcement \( r \) is nearly directly proportional to \( bCost \). For this reason, \( r \) is multiplied by \( bCost \) itself as the role of the weighted value. There is no need to worry more complex by this. Because the multiplication is simpler than the division about calculation onto network processors, we can expect processing time to increase a little by adding this procedure. Therefore, it is not a great burden.

The proposed procedure for calculating the reinforcement value is presented as the following pseudo code of Fig. 1. Firstly, it stores the weighted values that are the multiples of 2 from 2 to 16. Then, it checks the validity of \( bCost \). Next, if \( curCost \) is less than \( bCost \), \( curCost \) becomes the new \( bCost \). At this time, the reinforcement \( r \) is assigned the defined maximum value. Else, \( C_{bCost} \) and \( C_{curCost} \) are assigned by the relation of Table 1. Finally, the reinforcement \( r \) is calculated by Eq.(4) and Eq.(5). If this calculated reinforcement exceeds the limit of maximum or minimum, it is assigned the available maximum or minimum values.
Improved Reinforcement Computing to Implement AntNet-Based Routing using General NPs for Ubiquitous Environments

reF : Reinforcement value \( r \)
bCost : Best cost value within the available time
curCost : Current cost value reported by backward agent
\( C_{\text{locCost}} \) : Weighted value according to bCost
\( C_{\text{carCost}} \) : Weighted value according to curCost
Comptable : Assigned weighted value by interval

```c
int comptable[] = {2, 4, 6, 8, 10, 12, 14, 16};
void setReinforcement ( int curCost ) {
    if ( ! checking the available time of bCost ) {
        bCost = reset;
    }
    if ( curCost < bCost ) {
        bCost = curCost ;
        reF = max [ r ] ;
    } else {
        // assign by 2n interval
        for ( i=1; i=amount of interval; i++) {
            if ( 2i+2 < bCost < 2i+3 ) \( C_{\text{locCost}} \) = comptable[i];
            if ( 2i+2 < curCost < 2i+3 ) \( C_{\text{carCost}} \) = comptable[i];
        }
        reF = normal{(255 + (bCost-curCost))*bCost / \( C_{\text{locCost}} \) / \( C_{\text{carCost}} \) ;}
    }
    if ( reF > 25 ) reF = 25; // limit max. & min. reF
    else if ( reF < 1 ) reF = 1;
}
```

Fig. 1. Pseudo Code for the Calculation of the Proposed Reinforcement

5 Simulation Results

![Network Topology for the Simulation](image)

Fig. 2. Network Topology for the Simulation
The proposed algorithm is evaluated by the comparison with the original AntNet in the topology presented in Fig. 2. This topology is like NSFnet. The node "0" is the source node and the node "12" is the destination node. The probability for routing of the source node "0" is evaluated and the variation of the best path is observed in the dynamic topology. The update number, which is a value of x axis, equals the frequency of changing probabilities of routing tables on the source node. If a particular path has the most routing probability from the source node to neighborhood nodes "1", "3", and "4", we can confirm that this way is the best path. The routing probability on the source is presented in y axis. We also define the four states of the network traffic patterns as follows:

i) Light: All traffics are equally distributed. It means the routing time between nodes is nearly the same.

ii) Biasing: The routing time of a specific path is shorter than the others. It means the costs of links by a trip time are different from those by hopping.

iii) Heavy: All traffic are uniformly distributed as the light pattern. However, the amount of traffic is 10 times heavier than that of light state.

iv) Dynamic: The network traffic pattern changes periodically at each 500 update. The traffic pattern is rotated sequentially the light, the biasing and the heavy traffic state.

We assume that node "0" is the source node and node "12" is the destination node. There are many paths from node "0" to node "12". The time delay on each link sets as cost according to each of the above traffic pattern from the light, to the biasing, and to the heavy traffic state.

In the light state, the path via 0-4-9-12 has the smallest number of links to reach the destination node. Therefore, since all traffic is equally distributed between nodes, this path is the best path and the probability of choosing the 0-4 link is larger than others. It is shown as Fig. 3.

![Fig. 3. Simulation Results: Light State](image_url)

In the biasing state, we assume that the path via 0-3-5-6-7-9-12 would have the lowest cost even though this has a larger number of links to the destination node. As observed in Fig. 4, the probability of choosing this best path is the highest. This
indicates that the proposed algorithm correctly finds the best path in the same way as the original algorithm.

(a) Original Algorithm  
(b) Proposed Algorithm

Fig. 4. Simulation Results: Biasing State

In the heavy state, because distributed traffic is the same as the light state despite the 10 times larger routing time between nodes needs, we expect the same results as for the light state. We observe this clearly in Fig. 5. Based on this result, we confirm both algorithms can select the best path by relatively distributed costs.

(a) Original Algorithm  
(b) Proposed Algorithm

Fig. 5. Simulation Results: Heavy State

In the dynamic state, when updating the routing probability on the source node become each 500 time, the network traffic state changes to the next state. The order is light, biasing and heavy states. The results are shown in Fig. 6. As the original AntNet are affected by the second term of Eq.(1), it needs to have a lot of time to adapt to new environments. So, the original AntNet delays slightly while adapting to dynamic network topology. However, the proposed algorithm quickly reacts to the traffic variation due to its simpler processing. Therefore, the shapes in Fig. 6(b) are clearer
than those in Fig. 6(a). Although the two graphs differ in detail, both algorithms selected the best path.

![Graphs showing simulation results.](image)

(a) Original Algorithm  
(b) Proposed Algorithm

Fig. 6. Simulation Results: Dynamic State

The simulation results confirm that the performance of the proposed algorithm which is implemented efficiently on existing network devices is similar to that of the original algorithm. Moreover, we can implement AntNet-based routing without the extra hardware or replacement of existing network devices. In addition, to find the best path, the proposed algorithm is clearly more efficient than the original algorithm in a dynamical traffic environment. Selecting the best path of the proposed algorithm is also faster than that of the original algorithm. It is certain that the best path from the source to the destination is found by the proposed algorithm. As it shortens the processing time on hardware for routing, it improves the performance speed as well.

6 Conclusion

If AntNet is implemented on a housekeeping core in network processors around Ubiquitous Environments, a housekeeping core become overwhelmed by increasing routing workload for a processing of agents. There are constraints to implement the original AntNet on conventional packet forwarding engines with simple arithmetic units as it is. In this paper, the reinforcement computing method with simple arithmetic units is proposed. It can be implemented efficiently onto packet forwarding engines of conventional network processors. The results of the simulation show that the proposed AntNet is more adaptive and effective in the performance of the implementation than the original AntNet.

**Acknowledgments.** This research was supported by University IT Research Center Project of Korea.
References

7. Clearwater Networks.: Introducing the CNP810 Family of Network Services Processors. Clearwater Networks(2001)