Efficient Behavior Selection for Immunologically-inspired Agents in the NetSphere Architecture

Chonho Lee and Junichi Suzuki
{chonho, jxs} @ cs.umb.edu
Department of Computer Science
University of Massachusetts, Boston
Boston, MA 02125-3393, USA

ABSTRACT

This paper describes and evaluates a biologically-inspired adaptation mechanism that allows network application components (agents) to autonomously adapt to dynamic environment changes in the network (e.g. changes in network traffic and resource availability). Based on the observation that the immune system elegantly achieves autonomous adaptation, the proposed mechanism, called the iNet artificial immune system, is designed after a scheme in which the immune system produces specific antibodies against an antigen invasion. iNet models an agent behavior (e.g. migration and replication) as an antibody, and an environment condition (e.g. network traffic and resource availability) as an antigen. iNet allows each agent to (1) autonomously monitor its surrounding environment conditions (i.e. antigens) to evaluate whether it adapts to the environment conditions, and if it does not, (2) adaptively perform a behavior (i.e. antibody) suitable for the environment conditions. This paper focuses on the first process, the environment evaluation process, and describes its design and implementation. The experimental results show that the environment evaluation process works efficiently with a high degree of accuracy.

Keywords: artificial immune system, biologically-inspired adaptation

1. INTRODUCTION

As computing devices and networks are becoming more powerful and ubiquitous, the networking landscape is evolving into new paradigms, such as autonomic networks [1], pervasive networks [2], grid networks [3] and space networks [4]. In these networking paradigms, future network applications will be orders of magnitude more complex and larger than current ones, and they are expected to be more autonomous, scalable and adaptive to dynamic network environments [5]. As inspiration for a new network application design paradigm, the authors of the paper observe that various biological systems have already developed the mechanisms necessary to achieve the key requirements of future network applications such as autonomy, scalability and adaptability. For example, a bee colony scales to support a huge number of bees, and autonomously adapts to a wide variety of weather and food conditions. The authors of the paper believe if network applications are modeled after certain biological concepts and mechanisms, they may be able to meet these requirements of future network applications.

The NetSphere architecture applies key concepts and mechanisms in biological systems to design network applications[1]. One of the key concepts in biological systems is emergence. In biological systems, beneficial system properties (e.g. adaptability) often emerge through the simple and autonomous interactions among diverse biological entities. The NetSphere architecture applies the concept of emergence by implementing network applications as a group of distributed, autonomous and diverse agents. This is analogous to a bee colony (a network application) consisting of multiple bees (agents). Each agent implements a functional service related to the application and follows simple behaviors similar to biological entities, such as reproduction, death, migration and environment sensing.

Similar to entities in the biological world, agents in the NetSphere architecture are designed to provide a sufficient degree of diversity. Different agents may implement different services. For instance, an agent may implement an airline reservation service, while another agent may implement a hotel reservation service. An agent may implement a web service and contain web pages. Different agents may implement different behavior policies. For instance, an agent may have a migration policy of moving towards a user, while another agent may have a migration policy of moving towards a network node where resource availability is higher.

Similar to an entity in the biological world, each agent in the NetSphere architecture may store and expend *energy* for living. Each agent may gain energy in exchange for performing a service, and they may pay energy to receive a service from other agents and to use network and computing resources. The abundance or scarcity of stored energy may affect various behaviors of an agent. For example, an abundance of stored energy is an indication of higher demand for the agent; thus the agent may be designed to favor reproduction in response to higher levels of stored energy. A scarcity of stored energy (an indication of lack of demand or ineffective behaviors) may eventually cause the agent's death.

This paper addresses and evaluates autonomous adaptability of network applications (i.e. agents) in the NetSphere architecture. Autonomous adaptability is an ability of network applications to intelligently adapt to dynamic changes in the network without any administrative interventions from and to other entities (e.g. other agents and users). Modeled after the mechanism behind how the immune system produces specific antibodies against an antigen invasion, the proposed adaptation mechanism, called iNet, allows agents to autonomously monitor their surrounding environment conditions (e.g. traffic volume and resource availability) and adaptively perform their biological behaviors (e.g. migration and replication) suitable for the current environmental conditions. For instance, agents may invoke migration behavior for moving towards network nodes that accept a large number of user requests for their services.

[1] The NetSphere architecture is an extension to the Bio-Networking Architecture [6, 7].

This results in the adaptation of agent locations, and agents concentrate around the users who request their services.

The structure of this paper is organized as follows. Section 2 overviews key design principles of agents in the NetSphere architecture. Based on the design principles, Section 3 describes the design of agents in the NetSphere architecture. Section 4 presents the design of the iNet artificial immune system and how it can contribute to autonomous adaptability of agents. Section 5 shows several experimental results to evaluate efficiency and accuracy of the proposed environment evaluation facility. Sections 6 and 7 conclude with comparison with existing related work and some discussion on future work.

2. DESIGN PRINCIPLES IN THE NETSPHERE ARCHITECTURE

In the NetSphere architecture, agents are designed based on the three principles described below in order to interact and collectively provide network applications that are autonomous, adaptive, scalable and simple [6, 7].

(1) Decentralization: Agents in the NetSphere architecture are decentralized. There are no central entities to control and coordinate agents (i.e. no directory servers and no resource managers). Decentralization allows network applications to be scalable and simple by avoiding a single point of performance bottleneck and failure and by avoiding any central coordination in developing and deploying agents.

(2) Autonomy: Agents in the NetSphere architecture are autonomous. Agents monitor their local network environments, and based on the monitored environmental conditions, they autonomously interact without any intervention from other entities (e.g. other agents and human users).

(3) Adaptability: Agents in the NetSphere architecture are adaptive to dynamically changing environmental conditions (e.g. user demands, user locations, and resource availability). This is achieved through designing agent behavior policies to consider local environmental conditions [8]. For example, an agent may invoke replication and death behaviors when its energy level becomes over and below thresholds. This results in the adaptation of agent availability. The population of agents can be adaptable against changes in workload or the number of users.

3. AGENTS IN THE NETSPHERE ARCHITECTURE

Each agent in the NetSphere architecture is implemented as a Java object and runs on a NetSphere platform. The NetSphere platform is implemented in Java[2], and each platform runs on a Java virtual machine. The platform provides reusable software components for developing, deploying and executing agents. The components abstract low-level operating and networking details (e.g. network I/O and concurrency control for executing agents), and implement high-level runtime services that agents use to perform their services and behaviors [6, 7]. The iNet artificial immune system is implemented as a component in the NetSphere platform [8].

Each agent consists of three parts: *attributes*, *body* and *behaviors* (Figure 1). *Attributes* carry descriptive information regarding the agent (e.g. agent ID and description of a service it provides). The *body* implements a service that the agent provides and contains materials relevant to the service (e.g. application data and user profiles). For instance, an agent may implement control software for a device in its body, while another agent may implement a hotel reservation service in its body. An agent that implements a web service may contain web pages in its body. *Behaviors* implement non-service related actions that are inherent to all agents.

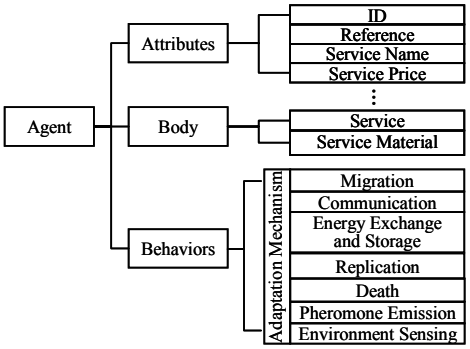

Figure 1. Design of an Agent in the NetSphere Architecture

Some example behaviors are explained below.

- ***Migration****:* Agents may migrate from one platform to another platform.
- *Communication*. Agents may communicate with other agents for the purposes of, for instance, requesting a service or exchanging energy.
- ***Energy exchange and storage***: Agents may receive and store energy in exchange for providing services to other agents. Agents also expend energy. For instance, agents may pay energy units for services that they receive from other agents. In addition, when an agent uses resources on a platform (e.g. CPU and memory), it may pay energy units to the platform.
- ***Lifecycle regulation****:* Agents may regulate their lifecycles. Agents may make a copy of themselves (replication), possibly with mutation of the replica's behavioral policy. Agents also may die (death) as a result of lack of energy. If energy expenditure of an agent is not balanced with the energy units it receives from providing services to other agents, it will not be able to pay for the resources it needs, i.e., it dies from lack of energy. Agents with wasteful behavioral policies (e.g. replicating or migrating too often) will have a higher chance of dying from lack of energy.
- ***Pheromone emission****:* Agents may emit and leave a pheromone (or a trace) behind on a platform when they migrate to another platform. This is to indicate their presence to other agents. A pheromone contains the emitter's ID and a reference to the platform that the emitter migrated to. Pheromones are emitted with certain strength and may decay over time. Pheromones may have a variety of uses, including improving the performance of discovery.
- ***Environment sensing***. Agents may sense their local environment. For instance, an agent may sense the local environment to learn which agents are in the environment and what services they provide. An agent may also sense pheromones (e.g. which agents left pheromones on local and neighboring platforms) and resources (e.g. CPU processing power and memory space available on local and neighboring platforms).

[2] The current code base of the NetSphere platform contains approximately 37,300 lines of Java code [7].

4. INET ARTIFICIAL IMMUNE SYSTEM

Each agent has its own artificial adaptation mechanism for behavior selection (Figure 1). This adaptation mechanism allows an agent to monitor its surrounding environmental condition (e.g. network traffic, resource availability), identify behaviors suitable for the monitored environmental condition, and decide which behavior to perform so that the agent adapts to the current environmental condition. The design of the adaptation mechanism is based on how the natural immune system produces specific antibodies against an antigen invasion.

4.1. Natural Immune System

The natural immune system is known as a distributed adaptive system with defense mechanism to maintain the system against dynamically changing environment (i.e. antigens' invasion). It consists of a tow-line defense, innate immune system and adaptive immune system. Both systems depend upon the activity of white blood cells, the leukocytes, where the innate immunity is mediated mainly by macrophages, and the adaptive immunity is mediated by lymphocytes. The immune response involves a number of components such as macrophages, lymphocytes and antibodies. Macrophages circulate throughout the body ingesting and digesting antigens and fragment them into small peptides, called self-MHC (Major Histocompatibility Complex.) They have the ability to present the self-MHC to lymphocytes. Lymphocytes are one of many types of white blood cells, and two major populations of lymphocytes are T lymphocytes (T-cells) and B lymphocytes (B-cells).

T-cells have receptors (TCR) on their surface, which can recognize antigens by binding to self-MHC presented by macrophage. T-cells are produced in thymus through two main processes, positive selection and negative selection. With these processes, the immune system obtains the ability to accomplish the self/non-self discrimination, which is one of the remarkable features in the immune system. It discriminates foreign molecules (i.e. Non-self) from the body's own cells and proteins (i.e. Self).

Immature (Naïve) T-cells, which are stem cells, are originated in a bone marrow and migrate to thymus (Figure 2-A) where some of them are allowed to mature (i.e. become mature T-cells) into immunocompetent cells capable of recognizing "Self" (positive selection), and others are removed from repertory due to a strong reaction of "Self" (negative selection). These processes guarantees that T-cells leave the thymus and recognize "Self", e.g. self-MHC, at the same time that it is ready to be stimulated by macrophage with non-self antigens (Figure 2-B). Once "Non-self" is recognized, that is, T-cells receive signals from macrophage; T-cells secrete chemical signals to the participation of a variety of cells and molecules such as B-cells to mount an appropriate response in order to eliminate them (Figure 2-C). B-cells have receptors on their surface, which can recognize chemical signals, and antibodies which can attack the recognized antigens invading a human body, e.g. viruses. Once the B-cells are activated by the chemical signals secreted by the activated T-cells, they proliferate with their own copies. This is called a clonal selection. Then, proliferated B-cells start to produce a bunch of the same type of antibodies specific to the recognized antigen (Figure 2-D).

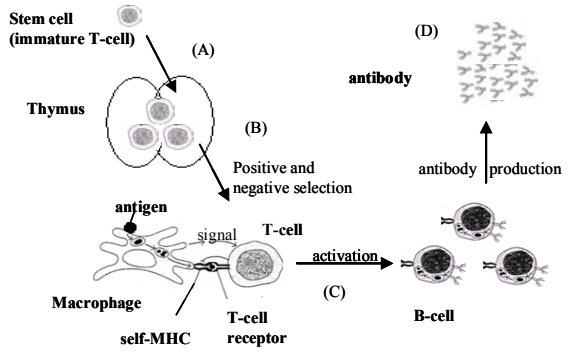

Figure 2. Natural Immune System

4.2. Artificial Immune System

An agent in iNet is designed based on the concept of the natural immune system described in previous subsection. It consists of two architectural components: environment evaluation facility and behavior selection facility corresponding to the innate immune response and adaptive immune response. Figure 3 illustrates the overview of the adaptation mechanism. How the concepts of the iNet artificial immune system map to the natural immune system is shown in Table I.

In the previous version of iNet (which has only behavior selection facility), each agent periodically obtains its surrounding environmental condition and always invokes a behavior. This means that each agent periodically behaves even when it adapts to the current environment condition well (i.e. even when it does not have to behave). As a result, it might perform unnecessary, unsuitable behaviors. However, the environment evaluation facility, which this paper focuses on, allows each agent to evaluate the current degree of adaptation (based on the

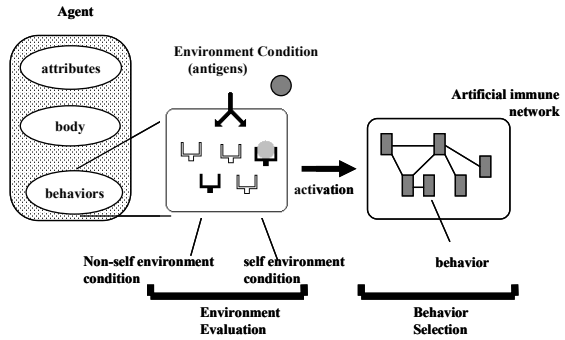

Figure 3: iNet artificial immune system (adaptation mechanism)

Concepts in Natural Immune System		Concepts in iNet	Implementations in iNet
immune response	innate	environment evaluation facility	Initialization and classification of feature vectors
	adaptive	behavior selection facility	behavior selection in the artificial immune network
antigen	self antigen	self environment condition (higher degree of adaptation)	feature vector classified with class value of "self" (0)
	non-self antigen	non-self environment condition (lower degree of adaptation)	feature vector classified with class value of "non-self" (1)
antibody		agent's behavior	agent's behavior
macrophage circulation		environment sensing	collection of the current environment condition through using Environment Sensing Service
stem cell		randomly generated environment condition	feature vector with no class value (unclassified feature vector)
self-MHC		user-defined self environment condition	feature vector classified with class value of "self" (0)
T-cell		detectors for non-self environment condition	feature vectors in a feature table
T-cell maturation	positive selection	N/A	N/A
	negative selection	initialization	feature vector matching
self/non-self discrimination		self/non-self classification of environment conditions	classification of feature vectors

Table 1. Mapping between iNet artificial immune system and natural immune system

monitored current environment condition) and determine whether to need invoking a certain behavior. Thus, each agent avoids invoking unnecessary behaviors and utilizes computing and networking resources efficiently.

In iNet, an antigen is implemented as a feature vector representing a current environment condition supplied by environment sensing behavior. The feature vector might consist of several particular features indicating a degree of adaptation for networks, and one class value (e.g. X = (F, C) where feature F = (a1, …, an) and class C = {0,1}). For example, the number of agents on the same node, the number of messages transferred per sec, energy level, and memory utilization might represent the current system condition (e.g. X_{cur} = (10, 500, low, 5, 1)). A "Non-self" class value, 1, implies a lower degree of adaptation for the network; on the other hand, "Self" class value, 0, tells us a higher degree of adaptation.

The environment evaluation facility has two basic steps, initialization and self/non-self classification of environment condition, which are based on the concept of T-cell maturation and self/non-self discrimination. In the initialization step, environment detectors to determine whether to invoke a certain behavior against a current environment condition are created by a generator (i.e. feature vector matching). The environment detectors are designed as T-cells, a certain type of immune cells that detect antigen invasions. The generator is designed as T-cell maturation process to produce T-cells that only detect non-self molecules and do not react to self molecules. In the classification step, monitored antigens are classified into self antigens and non-self antigens, through the mechanism of immunological self/non-self discrimination. Non-self antigens represent environment conditions that an agent needs to increase the degree of adaptation (i.e. it needs to invoke behaviors). Self antigens represent environment conditions that an agent adapts well to (i.e. it does not need to invoke behaviors to).

(1) Initialization Step

The initialization step creates an initial dataset (i.e data samples for classification algorithm), which is detectors for non-self environment condition. They are a collection of temporarily classified feature vectors, distributed by the feature vector matching of randomly generated feature vectors(X) and user-defined self feature vectors (S) given by users (Figure 4).

This procedure follows the process of T-cell maturation described in Section 4.1. Randomly generated feature vectors and user-defined self feature vectors are compared by a measure of vector distance. There are many ways to calculate the vector distance. One of them is the similarity of two vectors using simple Euclidean distance method. We assume that vector distance represents TCR affinity. Closer vectors (i.e. distance is less than threshold) represent higher affinity between cells, and further vectors (i.e. greater than threshold) represent low affinity. As shown in Figure 4, randomly generated feature vectors, X, distributed by the feature vector matching are temporarily classified as candidates of self and non-self environment condition (Cs and Cn). They are stored into an initial dataset[3], called a feature table (described in the next

[3] In the natural immune system, T-cells strongly responding to self-MHC will die in thymus (i.e. self candidates Cs should be removed due to self tolerance). However, for classification accuracy (described in Section 5), we obtain self candidates in a feature table.

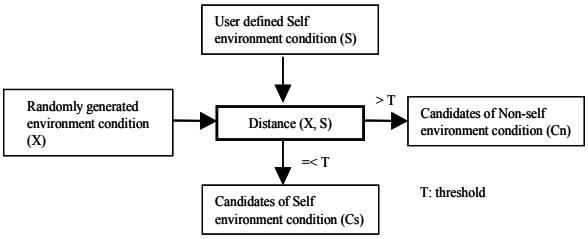

Figure 4: Feature Vector Matching

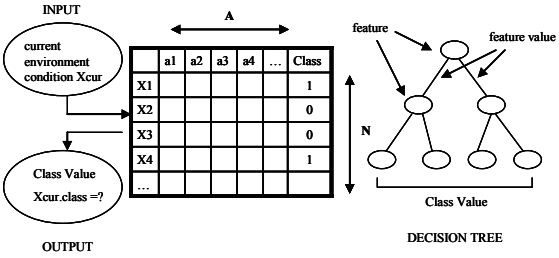

Figure 5: Input and Output of self/non-self classification

subsection), and they are used for training of classification algorithm

(2) Self/Non-self Classification Step

In this step, a classification algorithm classifies a current environment condition (X_{cur}), Input, into "Self" and "Non-self", which is the classification result, Output. This paper introduces the decision tree algorithm (DT) as its classification algorithm. The complexity of query time in DT is very fast, and the classification process is interpretable and often easy.

In Figure 5, there is a table called a feature table of size *N*, containing feature vectors (i.e. detectors, temporarily classified self and non-self feature vectors), each of them has *A* distinct features. These feature vectors in a feature table work as data samples and form a tree used for their classification. (it will be retrained to improve the quality of the algorithm). In DT, each internal node represents a feature of environment conditions, each branch indicates a value or a range of the feature, and each leaf assigns the class value. That is, the role of DT is to label an environment condition with the class value of leaf (i.e. "Self" or "Non-self") reached by searching in the tree.

Building DT, it is useful to calculate "information gain", describing how much clearly classes are distributed. "Information" is defined by Entropy (ranges from 0 to 1) representing a degree of randomness or pureness between two classes, "Self" and "Non-self". Higher Entropy implies the state that classes are equally randomly occupied, and lower Entropy

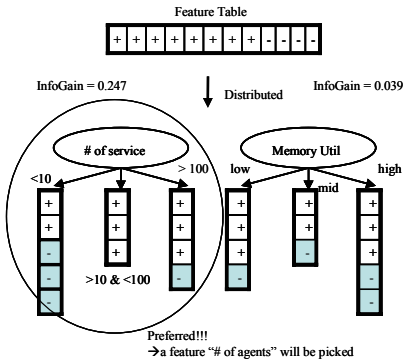

Figure 6: How to select a feature as a branch of tree

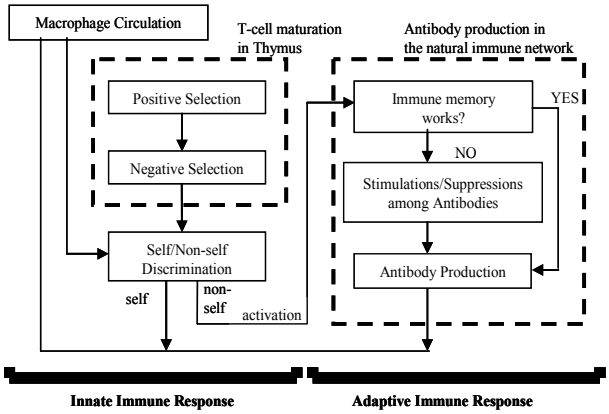

Figure 7: Key Steps in Natural Immune Response

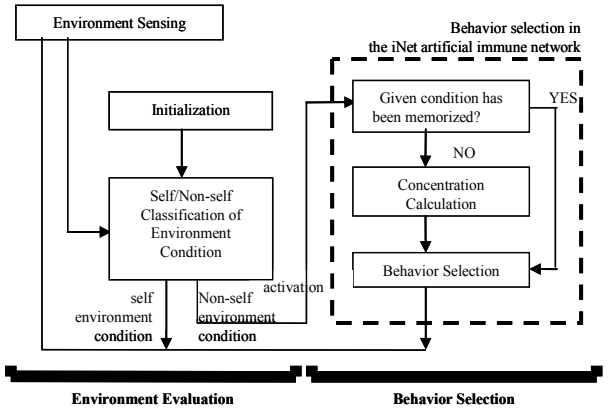

Figure 8: Key Steps in iNet Artificial Immune Response

does the state is biased by a particular class. DT prefers the state having lower Entropy because the goal is to determine the class value of an environment condition (Figure 6). How to build the decision tree is as follows. First, it selects a feature, whose information gain is the highest (i.e. prefers lower Entropy state from high Entropy state), as a splitting feature at the root of the tree. Then, it continues to select remaining features recursively until the data samples can't be split any further.

The environment evaluation facility reports the classification result (X_{cur}.class), which is Output (Figure 5), to the behavior selection facility only when the classified environment condition is "Non-self". Otherwise the immune system does nothing and waits for next environment condition. It is beyond the scope of this paper to describe the details of the behavior selection facility. For more details, see the following references [4]. Finally, the key steps of both the natural immune response and the iNet artificial immune response are summarized in Figure 7 and Figure 8.

5. EXPERIMENTAL RESULTS

This section evaluates the efficiency and accuracy of iNet through several experimental results.

5.1 Configurations for Experiment

An experiment test bed is built to deploy and run iNet. Both iNet and agents are implemented in Java. All the measurements are performed on a Java 2 standard edition virtual machine (version 1.3.1 from Sun Microsystems) on a PC with a 2.5GHz Celeron CPU and 1GB memory.

5.2 Efficiency of Self/Non-self Classification

This experiment measures the overhead of the proposed classification algorithm (i.e. decision tree algorithm) in order to understand its efficiency. The overhead of the classification algorithm includes initialization time, T_{init}, and classification time, $T_{classify}$. T_{init} consists of t1, the time to create a feature table, and t2, the time to build its decision tree. $T_{classify}$ is the time to classify a given feature vector. So, the overhead of the environment evaluation facility is considered as $T_{EE} = T_{init} + T_{classify}$. This value is affected by three parameters: N, the size of a feature table, A, the number of features in a feature vector (i.e. the number of environment conditions), and S, the number of user-defined self conditions provided in initialization step. Figure 9 shows the time t1 and t2 for initializing the classification algorithm having N samples (i.e. # of feature vectors) for each parameter A and S. The experimental results show that T_{init} gradually increases in proportion to the parameters. However, in this experiment, we assume that the proposed classification algorithm will not be trained during runtime execution. So the environment evaluation facility consumes only an amount of time $T_{classify}$ shown in Table II because T_{init}(=t1+t2) is required only once in the initialization step but during runtime execution.

The overhead of behavior selection facility, T_{BS}, just takes the time for selecting a behavior suitable for the current environment condition [8]. In the behavior selection facility, among several antibodies which respond to antigens, one antibody is selected as an agent's behavior. The number of antibody, AB, is determined by the number of features A, the number of distinct values of each feature, V, and the number of behaviors that each agent supports, B, as $AB = (A*V)*B$. Figure 10 shows that how much time the behavior selection facility spends to select a behavior.

Based on assumption that V=3 and B=3, only the parameter A affects the amount of T_{BS}. Compared with the value of T_{BS}, $T_{classify}$ is quite fast. For example, when A=5 (i.e. AB=75 with V=3 and B=5), T_{BS}=135.41msec will be skipped if the classification algorithm spends $T_{classify}$=3msec (Table II) and says a current environment condition is "Self". In fact, the decision tree, constructed from N feature vectors each of which has A features, might have the height of A at most. Since this

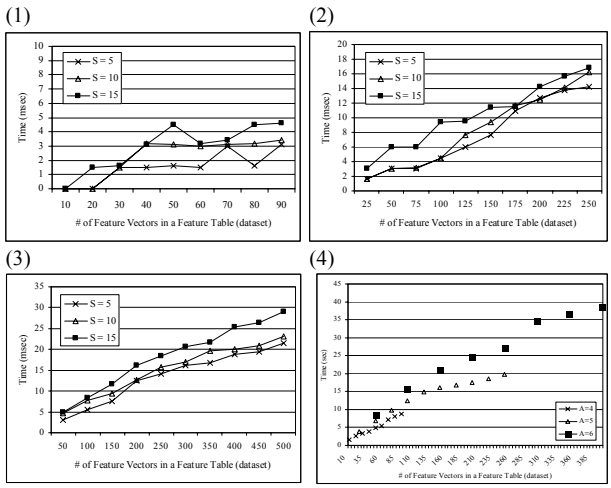

Figure 9: T_{init}: overhead of initializing a classification algorithm. (1)-(3) shows t1: time to create a feature table of size N when A=4, 5 ,and 6, and (4) shows t2: time to build its decision tree of size N.

# of Features (A)	3	4	5	6
$T_{classify}$ (msec)	1.5	3.0	3.0	3.0

Table II: $T_{classify}$: overhead of classifying a feature vector when *A*=3, 4, 5, and 6

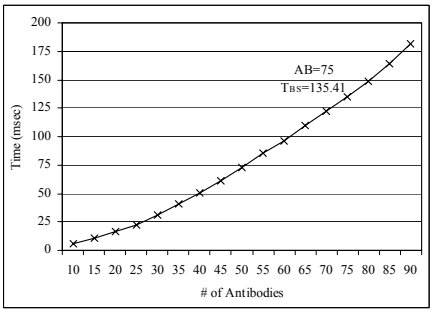

Figure 10: T_{BS}: overhead of behavior selection facility, i.e. time for selecting a behavior

experiment assumes that *A* is 4, 5 or 6 (e.g. Section 4.2), the classification needs only 4, 5 or 6 comparisons to search the leaf indicating "Self" or "Non-self". Therefore, $T_{classify}$ is dramatically fast as compared with T_{init} and also T_{BS}.

5.3 Accuracy of Self/Non-self Classification

The accuracy of the self/non-self classification is measured by comparing the classes of input feature vectors with those of output feature vectors. The accuracy is also affected by two parameters: *N* and *A*. In general, a classification algorithm classifies a testing data (e.g. a current environment condition) more accurately on more samples and features (i.e. the larger *N* and *A*). How these parameters affect the accuracy of classification is shown in Figure 11.

The proposed classification algorithm will not be retrained during runtime execution although its accuracy might be improved by retraining the decision tree. So it is important to decide an appropriate size of the parameter *N* (i.e. the size of a feature table) for both reducing overhead and improving classification accuracy. There is a trade-off between the efficiency and accuracy of the classification algorithm; application developers need to determine the value of *N*, based on the experimental results shown in Table III, depending on the requirements of their application.

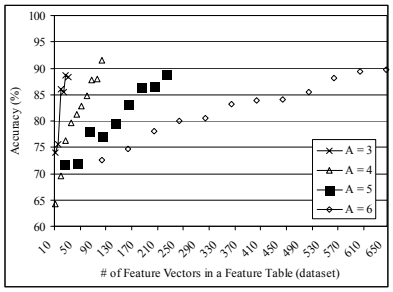

Figure 11: accuracy of classification on 100 testing data when *A*=3, 4, 5, and 6

Accuracy	# of Features (A)		3	4	5	6
	# of Possible combination of features		27	81	243	729
	# of user-defined self		10	10	10	10
80%	Tinit (sec)	t1	0.0015	0.0016	0.0034	0.0127
		t2	2.210	3.913	9.874	24.762
		total	2.212	3.915	9.877	24.775
	N: size of feature table		**20**	**40**	**75**	**200**
85%	Tinit (sec)	t1	0.0015	0.0031	0.0094	0.0209
		t2	2.767	7.121	16.101	41.060
		total	2.769	7.124	16.111	41.081
	N: size of feature table		**25**	**70**	**150**	**450**
90%	Tinit (sec)	t1	-	0.0034	0.0141	0.0281
		t2	-	8.748	18.528	52.392
		total	-	8.751	18.542	52.420
	N: size of feature table		-	**90**	**225**	**650**

Table III. Recommended size of feature table (underlined) according to the overhead for initialization T_{init} and accuracy of classification algorithm.

5.4 Performance Impact of the Environment Evaluation Facility on iNet

In order to understand how the environment evaluation facility effectively contributes to iNet in terms of efficiency, the overhead of behavior selection facility (T_{BS}) is compared with the overhead of executing both environment evaluation and behavior selection facility (T_{EE}+T_{BS}) on the following three different scenarios.

- ***Self environment conditions only*:** The environment evaluation facility faces only self environment conditions. This simulates a static network environment where environment condition does not dynamically change and an agent always adapts to the environment well (i.e. the degree of adaptation is always high).
- ***Non-self environment conditions only*:** The environment evaluation facility faces only non-self environment conditions. This simulates a dynamic network environment where environment conditions dynamically change and a situation where an agent tries to adapt to environmental changes by performing their behaviors, but changed too often, so the degree of the agent adaptation is always low.
- ***Random environment conditions*:** The environment evaluation facility randomly faces self and non-self environment conditions. This simulates a dynamic network environment where environment conditions dynamically change and an agent does not always adapt to the environment well.

Figure 12 shows the computation overhead of executing iNet (i.e. both the environment evaluation and behavior selection facility) to given environment conditions (i.e. antigens) according to each scenario described above. The number of given conditions (x-axis), each of which is provided at particular time intervals, implies how long iNet is run.

If iNet does not have the environment evaluation facility, then it always executes the behavior selection facility even for the case the degree of adaptation is high (i.e. environment condition does not dynamically change and an agent always adapts to the

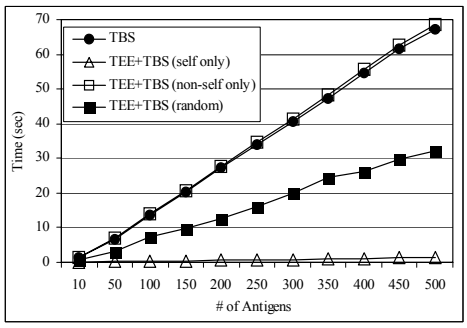

Figure 12: Computation overhead of executing iNet according to three different scenarios when *A*=5 (T_{EE}: overhead of environment evaluation facility, T_{BS}: overhead of behavior selection facility)

environment well). In other words, iNet can avoids unnecessary processes and does not have to execute the behavior selection facility whenever the environment evaluation facility evaluates a current environment condition as "Self".

6. RELATED WORK

Artificial immune systems have been proposed and used in various application domains such as intrusion detection [9], pattern recognition [10], data mining [11] and fault detection [12]. This paper proposes an artificial immune system for designing autonomous and adaptive network applications. To the best of our knowledge, this work is the first attempt to apply an artificial immune system to achieve autonomous adaptability of network applications.

This work extends our previous work, which only focused on the behavior selection facility in iNet [8]. This work proposes the environment evaluation facility in iNet, which introduces new immunological mechanism (T-cell maturation and self/non-self discrimination), and combines the new facility with the behavior selection facility (Figures 3 and 8). Experimental results show that the environment evaluation facility improves the efficiency of the immunological process in iNet (Section 5).

Existing mobile agent platforms allow agents to perform behaviors such as replication and migration [13, 14, 15] as the NetSphere platform does. However, they do not provide a general-purpose adaptation engine for agents to autonomously adapt to dynamic changes in the network. Therefore, human developers need to implement and configure behavior policies for each agent from scratch. It tends to be complicated, time consuming and error-prone. On the contrary, iNet is a general-purpose adaptation engine for agents to perform environment sensing, environment evaluation and behavior selection in a consistent manner.

7. CONCLUDING REMARKS

This paper investigates an artificial immune system, called iNet, to achieve adaptive behavioral selection for agents (network applications) in the NetSphere architecture. iNet allows agents to monitor their surrounding environment conditions and perform their behaviors adaptively to the monitored conditions. This paper describes and evaluates the environment evaluation facility in iNet, which allows agents to determine when to, and when not to, perform their behaviors. Experimental results that the environment evaluation facility works efficiently with a high degree of accuracy.

Extended experiments are planned to investigate more performance implications of iNet, such as resource utilization to execute the iNet immunological process. In addition to the current experimental environment, iNet and agents will be deployed on larger-scale experimental environments (e.g. PlanetLab [4]). It will provide more realistic performance implications on the autonomous adaptability with iNet. For the self/non-self classification process in iNet, other classification algorithms will be investigated, such as statistical and clustering algorithms.

[4] http://www.planet-lab.org

REFERENCES

[1] A. G. Ganek and T. A. Corbi, "The Drawing of the Autonomic Computing Era," In *IBM Systems Journal*, vol. 42, no. 1, 2003.

[2] M. Weiser, "The Computer for the 21st Century," *Scientific American*, September, 1991.

[3] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, 15(3), 2001.

[4] E. Criscuolo, K. Hogie and R. Parise, "Transport Protocols and Applications for Internet Use in Space," In *Proc. of the IEEE Aerospace Conference*, 2001.

[5] Large Scale Networking Coordinating Group of the Interagency Working Group for Information Technology Research and Development (IWG/IT R&D), *Report of Workshop on New Visions for Large-scale Networks: Research and Applications*, March 2001.

[6] T. Suda, T. Itao and M. Matsuo, "The Bio-Networking Architecture: The Biologically Inspired Approach to the Design of Scalable, Adaptive, and Survivable/Available Network Applications," In *K. Park (ed.) The Internet as a Large-Scale Complex System*, Princeton University Press, April 2005. to appear.

[7] J. Suzuki and T. Suda, "A Middleware Platform for a Biologically-inspired Network Architecture Supporting Autonomous and Adaptive Applications," In *IEEE Journal on Selected Areas in Communications (JSAC)*, vol 23, no.2. February 2005.

[8] J. Suzuki, "Biologically-inspired Adaptation of Autonomic Network Applications," In *International Journal of Parallel, Emerging and Distributed Computing*, 2005, to appear.

[9] D. Dasgupta and S.Forrest, "An Anomaly Detection Algorithm Inspired by the Immune System." In *Artificial Immune Systems and Their Applications*, Springer, Nov, 1998.

[10] Z. Tang, K. Tashima, Q.P. Cao, "Pattern Recognition System Using a Clonal Selection-based Immune Network," In *System and Computers in Japan*, vol 34, no 12, November 2003 p56-63

[11] T. Knight, J. Timmis: "AINE: An Immunological Approach to Data Mining," In *Proc. of ICDM* 2001.

[12] D.W. Bradley and A.M. Tyrrell., "Immunotronics: Hardware Fault Tolerance Inspired by the Immune System.", In *Proc. of the 3rd International Conference on Evolvable System,* Springer lncs, 2000.

[13] D. Lange and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets," Addison Wesley, 1998.

[14] N. J. E. Wijngaards, B. J. Overeinder, M. van Steen and F. M. T. Brazier, "Supporting Internet-Scale Multi-Agent Systems," In *Data and Knowledge Engineering*, (41)2-3, 2002.

[15] A. Fuggetta, G. P. Picco and G. Vigna, "Understanding Code Mobility," In *IEEE Trans. on Software Engineering*, 24(5), 1998.