

AUTONOMIC ADAPTATION OF NETWORK APPLICATIONS WITH THE INET ARTIFICIAL IMMUNE SYSTEM

Chonho Lee and Junichi Suzuki
Department of Compute Science
University of Massachusetts Boston
100 Morrissey Blvd. Boston, MA 02125
{chonho and jxs} @ cs.umb.edu

ABSTRACT

This paper describes and empirically evaluates a biologically-inspired adaptation mechanism that allows network applications to autonomously adapt to dynamic environment changes in the network. Based on the observation that the immune system has elegantly achieved autonomous adaptation, the proposed mechanism, called the iNet artificial immune system, follows the mechanisms behind how the immune system detects antigens (e.g. viruses) and specifically reacts to them. iNet models a behavior of network applications (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 network application to autonomously sense its local environment conditions (i.e. antigens) to evaluate whether it adapts well to the sensed conditions, and if it does not, adaptively perform a behavior (i.e. antibody) suitable for the conditions. Measurement results show iNet works efficiently and makes network applications adaptive.

KEY WORDS

Autonomous adaptive networks, artificial immune system

1. Introduction

Future network applications are expected to be more autonomous and adaptive to dynamic changes in the network (e.g. changes in network traffic and resource availability) in order to improve user experience, expand application's operational longevity and reduce maintenance cost. As inspiration for a new paradigm for network application design [1, 2, 3], the authors of the paper observe that various biological systems have already overcome the requirements (i.e. autonomy and adaptability).

The NetSphere architecture¹ applies key biological concepts and mechanisms to design network applications. It implements each network application as a group of distributed and autonomous software agents. This is analogous to a bee colony consisting of multiple bees (agents). Each agent implements a functional service and follows biological behaviors such as migration, communication, energy exchange, replication and death.

¹ The NetSphere Architecture is an extension to the Bio-Networking Architecture [6, 7, 8].

This paper addresses autonomous adaptability of network applications. The proposed adaptation mechanism, iNet, is designed after the mechanisms behind how the immune system detects antigens and produces specific antibodies to kill them. iNet models an environment condition (e.g. network traffic and resource availability) as an antigen and a behavior of agents as an antibody. iNet allows each agent to autonomously sense its local environment conditions (i.e. antigens) to evaluate whether it adapts well to the sensed conditions, and if it does not, adaptively perform a behavior (i.e. antibody) suitable for the conditions (i.e. antigens). For example, iNet may suggest an agent to invoke migration behavior for moving towards network nodes that accept a large number of user requests for their services. This leads to the adaptation of agent locations, and agents can reduce their response time for users.

This paper is organized as follows. Section 2 overviews the design of agents in the NetSphere architecture. Section 3 describes the design of the iNet artificial immune system. Section 4 shows several empirical measurements to evaluate iNet. Sections 5 and 6 conclude with comparison with existing related work.

2. NetSphere Architecture

In the NetSphere architecture, agents are designed based on the three principles described below [6, 7].

- **Decentralization:** Agents 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 performance bottleneck and any central coordination in deploying them [4, 5].
- **Autonomy:** Agents are autonomous. Agents monitor their local network environments, and they autonomously behave and interact without any intervention from/to other agents, platforms and human users.
- **Adaptability:** Agents are adaptive to changing environment conditions (e.g. user demands, user locations and resource availability). Each agent contains iNet, which allows the agent to adaptively behave against the current local environment conditions.

Each agent is implemented as a Java object and runs on a NetSphere platform [7]. The platform is also implemented

in Java and runs atop a Java VM on a network host. Each agent consists of three parts: *attributes*, *body* and *behaviors*. *Attributes* carry descriptive information regarding the agent (e.g. agent ID). *Body* implements a service the agent provides. For instance, an agent may implement a genetic algorithm for an optimization problem, while another agent may implement a physical model for scientific simulations. *Behaviors* implement actions inherent to all agents. Although NetSphere defines nine standard agent behaviors [7], this paper focuses on five of them.

- **Migration:** Agents may move between platforms.
- **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 may also expend energy for using resources available on a platform (e.g. memory space). The abundance and scarcity of stored energy affect various behaviors of an agent. For example, an abundance of stored energy indicates 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.
- **Replication:** Agents may make a copy of themselves in response to higher energy level.
- **Death:** Agents also may die as a result of lack of energy.

3. The iNet Artificial Immune System

This section overviews how the immune system works (Section 3.1), and describes how iNet is designed after the immune system (Section 3.2).

3.1 Immune System

The immune system (Figure 1) is an adaptive defense mechanism to regulate the body against dynamic environment changes (e.g. antigen invasions). Through a number of interactions among various white blood cells (e.g. macrophages and lymphocytes) and molecules (e.g. antibodies), the immune system evokes two immune responses: innate and adaptive immune response.

In the innate immune response, the immune system performs self/non-self discrimination to detect foreign molecules (e.g. viruses). This response is initiated by macrophages and T-cells, a type of lymphocytes. Macrophages move around the body to ingest antigens and present them to T-cells so that T-cells can detect the antigens. T-cells are produced in thymus and trained through the negative selection process. In this process, thymus removes T-cells that react with the body's own cells (self cells). The remaining T-cells are used as detectors for non-self cells. When T-cells detect non-self cells, they secrete chemical signals to activate the second immune response, the adaptive immune response.

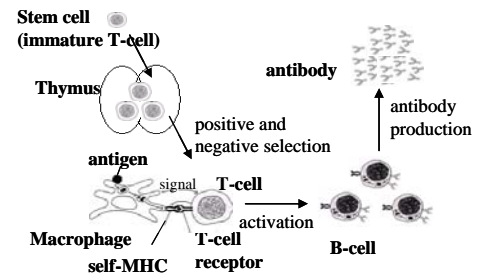


Figure1. Immune System

In the adaptive immune response, the immune system produces antibodies that specifically react and eliminate an antigen identified by T-cells. Figure 2 shows how antibodies recognize antigens and how antibodies interact with each other. A key portion of antigen recognized by antibodies is called epitope, which is the antigen determinant. Paratope is a portion of antibody that corresponds to a specific type of antigens. Once an antibody combines an antigen via their epitope and paratope, the antibody starts eliminating the antigen. Each type of antibody has its own antigenic determinant, called idiotope. This means an antibody is recognized as an antigen by other antibodies. Antibodies interact with each other through stimulation and suppression relationships. Thus, the adaptive immune response eliminating foreign antigens is offered by multiple antibodies in a collective manner, although the dominant role may be played by a single antibody whose paratope fits best with the epitope of an invading antigen.

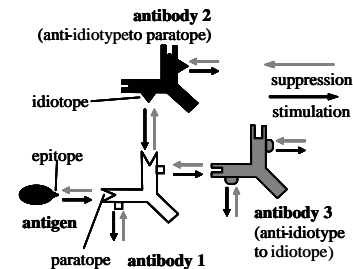


Figure2. Interactions among antigens and antibodies

3.2. Design and Implementation of iNet

The iNet artificial immune system consists of two facilities: the environment evaluation (EE) and behavior selection facility (BS) (Figure 3) corresponding to the innate and adaptive immune response, respectively. In EE, Each agent continuously senses a current environment condition as an antigen and examines it if it is self or non-self. A self antigen indicates the agent adapts to the sensed environment conditions, and a non-self antigen indicates it

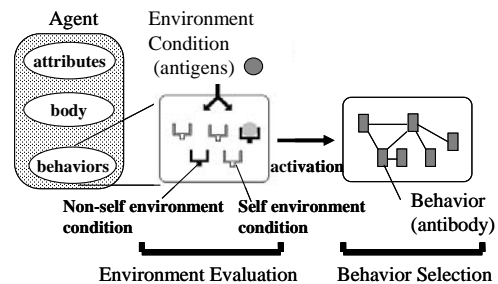


Figure3. iNet (adaptation mechanism)

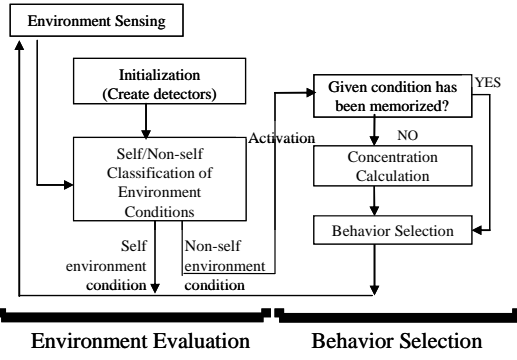


Figure4. Flow of the iNet adaptation mechanism

does not. When a non-self antigen is detected, BS is activated. Then it allows each agent to choose a behavior as an antibody that specifically matches the detected non-self antigen (i.e. non-self environment conditions).

3.2.1 Environment Evaluation Facility (EE)

The EE facility performs two steps (Figure 4): initialization and self/non-self classification. The initialization step randomly generates feature vectors first, each of which consists of features that represent environment conditions. In iNet, a set of environment conditions (i.e. an antigen) is implemented as a feature vector (X) consisting of features (F) and class value (C). F contains a series of the environment conditions such as service request rate, energy level, memory utilization, and the number of agents running on a local platform (e.g. $X_{current} = ((30, 500, low, 10), C)$). C indicates whether a given environment is self (0) or non-self (1). If it is 0, that means an agent adapts to the current environment. If it is 1, the agent does not. Like the negative selection in the immune system, the initialization step removes the feature vectors that closely match with the self environment conditions, where agents do not have to behave for adaptation. This is performed via vector matching between randomly generated feature vectors and self feature vectors that human users supply. Similar to T-cells, the remaining feature vectors are used to detect non-self environment conditions (i.e. antigens), where agents need to behave for adaptation.

The second step implements self/non-self discrimination in the immune system. It uses the feature vectors provided from the initialization system (i.e. artificial T-cells) to classify the current environment condition into self or non-self. This classification step is performed with a decision tree built from the feature vectors (Figure 5). The reasons for the choice of decision tree classifier are ease of implementation and algorithmic efficiency. Since this classifier

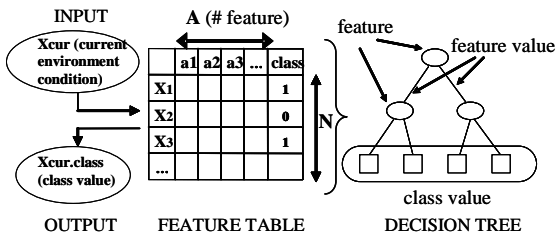


Figure5. Input and Output of self/non-self classification

is easy to understand and implement, iNet can maintain a lower barrier for developers to design adaptive network applications. Also, this classifier executes classifications much faster than other techniques using clustering, naïve bayes and Markov model algorithms [18, 19, 20]. For example, a time complexity for the query (i.e. classification) of decision tree is $O(\log(N))$ while that of clustering or naïve bayes algorithm is at least $O(N)$ (although it is comparable in some applications). The efficiency of classification is one of the most important requirements in iNet because each agent periodically senses and examines its surrounding environment.

3.2.2. Behavior Selection Facility (BS)

Once the current environment condition is classified as non-self, EE facility activates BS facility (Figure 4); then BS facility selects an antibody (i.e. agent's behavior) suitable for the detected non-self antigen (i.e. environment conditions). Each agent contains a network of antibodies, which are linked with each other using stimulation and suppression relationships. Each antibody (Figure 6) has its own concentration value corresponding to the number of the antibody. The value is used as priority in behavior selection. When an environment changes (i.e. when a non-self antigen is reported by EE), BS engine identifies candidate behaviors suitable for the current environment, prioritizes them based on their concentrations, and then selects the most suited one from the candidates. When prioritizing behaviors, stimulation relationship between behaviors contributes to increase the concentration value, and suppression relationship contributes to decrease it. Each relationship has its own strength (affinity), which indicates the degree of stimulation or suppression.

Figure 7 shows an interactions among antibodies in iNet. The antibody i stimulates M antibodies and suppresses N antibodies. m_{ji} and m_{ik} denote affinity values between antibody j and i , and between antibody i and k . m_i is an affinity value between an antigen and antibody i . The concentration of antibody i , denoted by a_i , is calculated with the following equations.

$$\frac{dA_i(t)}{dt} = \left(\frac{1}{N} \sum_{j=1}^N m_{ji} \cdot a_j(t) - \frac{1}{M} \sum_{k=1}^M m_{ik} \cdot a_k(t) + m_i - k \right) a_i(t) \dots (1)$$

$$a_i(t) = \frac{1}{1 + \exp(0.5 - A_i(t))} \dots (2)$$

In the equation (1), the first and second terms in a big bracket denote the stimulation and suppression from other antibodies. The affinity values between antibodies (i.e. m_{ji} and m_{ik}) are positive between 0 and 1. m_i is 1 when antibody i is stimulated directly by an antigen, otherwise 0. d denotes the dissipation factor representing the death of an antibody. The equation (2) is a sigmoid function used to squash the $A_i(t)$ value between 0 and 1. Every antibody's concentration is repeatedly calculated the particular number of times. If no antibody exceeds a threshold during the calculation, the antibody whose concentration value is the

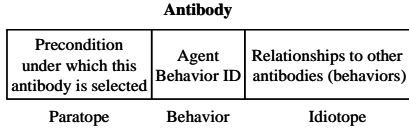


Figure 6. Antibody structure

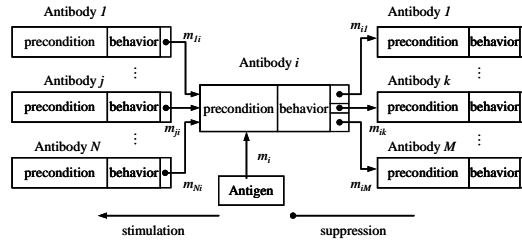


Figure 7. Interactions among antibodies

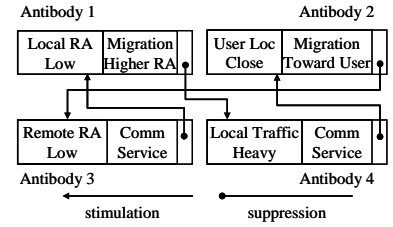


Figure 8. Example of antibodies in BS

highest is selected (i.e. winner-takes-all selection). If one or more antibodies' concentration values exceed the threshold, an antibody is selected based on the probability proportional to the current concentrations (i.e. roulette-wheel selection).

Figure 8 shows an example of antibodies in BS facility. It contains four antibodies representing behaviors: communication and migration with two different policies (Table I). Antibody1 represents that migration behavior is stimulated when resource availability is low on the local platform. Antibody1 suppresses Antibody3 when it is stimulated (i.e. when resource availability is low on the local platform). Now, suppose that (1) local resource availability is low on the local platform, (2) network traffic is low on the local platform, and (3) user location is close. In this case, three antigens stimulate Antibodies 1, 2 and 4 simultaneously. The populations of these antibodies increase, and it is likely that Antibody2's concentration value becomes highest because Antibody2 suppresses Antibody4, which suppresses Antibody1. As a result, Antibody2 (i.e. migration behavior) would be selected.

4. Preliminary Measurement Results

This section shows several preliminary results of empirical measurement to evaluate the performance of the iNet, both the EE and BS facility, and the adaptability of an application developed with iNet.

4.1 Measurement Configuration

The first set of measurements is to evaluate the performance of the iNet (Sections 4.2, 4.3 and 4.4). They were performed on a Java 2 standard edition VM (version 1.5 from Sun Microsystems) on a Windows XP PC with a 2.5GHz Celeron CPU and 1GB memory.

Table I. Paratopes and Agent Behaviors Supported in Antibodies

Paratope		Agent Behavior	
Major ID	Minor ID	Major ID	Minor ID
LOCAL RESOURCE AVAILABILITY (RA)	HIGH, LOW	MIGRATION	TOWARDS_USER
REMOTE RESOURCE AVAILABILITY (RA)	HIGH, LOW		HIGHER_RA
LOCAL TRAFFIC	HEAVY, LIGHT		HIGHER_TRAFFIC
REMOTE TRAFFIC	HEAVY, LIGHT	COMM-SERV	N/A
NUM LOCAL AGENTS	LARGE, SMALL	REPLICATION	N/A
ENERGY LEVEL	HIGH, LOW	REPRODUCTION	RANDOM
USER LOCATION	FAR, CLOSE		HIGHER_ENERGY

The second part of measurements (Section 4.5) is to evaluate the adaptability of an application developed with iNet. It was conducted assuming varying numbers of agents (from 1 through approx 100 agents) and NetSphere platforms (from 1 through 16 platforms). A maximum of 8 Windows XP PCs are used, each running a Java2 standard edition JVM (version 1.4.2_04 from Sun Microsystems). These 8 PCs were divided into 4 groups of 2 PCs, depending on their CPU speed and memory size (Table II). They were connected through 100Mbps Ethernet.

Table II. Configurations of PCs used in empirical evaluation

Group	CPU	Memory
A	Intel Celeron 2.0 GHz	512 MB
B	Intel Celeron 2.4 GHz	640 MB
C	Intel Pentium4 2.8 GHz	1000 MB
D	Intel Pentium4 3.0 GHz	1000 MB

4.2. Performance of Environment Evaluation Facility

This subsection shows the overhead of the EE facility containing Decision Tree classifier in order to verify its efficiency. The overhead includes initialization time T_{init} and classification time $T_{classify}$. T_{init} consists of $t1$, the time to create a feature table (i.e. to generate environment detectors), and $t2$, the time to build its decision tree. $T_{classify}$ is the time to classify a current environment condition. So, the total overhead of the EE 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 (i.e. environment conditions) in a feature vector, and S , the number of user-defined self environment conditions used in initialization step. Figure 9 shows T_{init} ($t1$ (1)-(3) and $t2$ (4)) to create a feature table having N feature vectors as parameter A and S vary; and Table III shows the classifi-

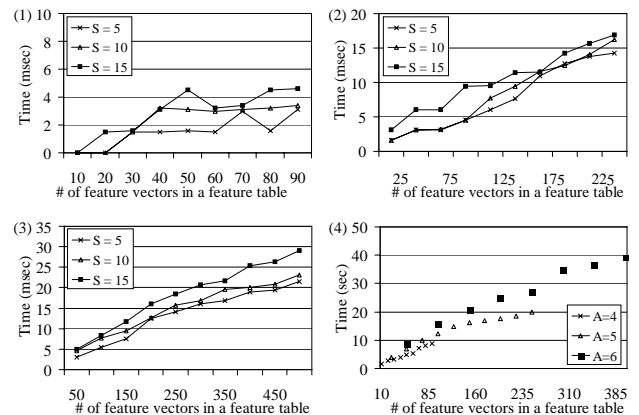


Figure 9. T_{init} : Initialization overhead of the EE facility.

(1)-(3) shows $t1$: time to create a feature table of size N when $A=4, 5, \text{ and } 6$, and (4) shows $t2$: time to build its decision tree of size N .

Table III. T_{classify} : Overhead of classification

# of features (A)	3	4	5	6	...	10
# of nodes in fully expanded DT	27	81	243	729		59049
T_{classify} (msec)	1.5	3.0	3.0	3.0		4.5

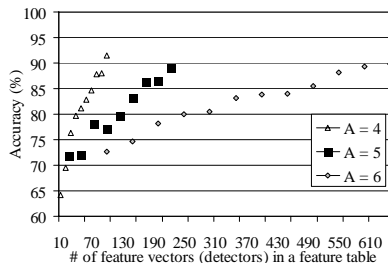
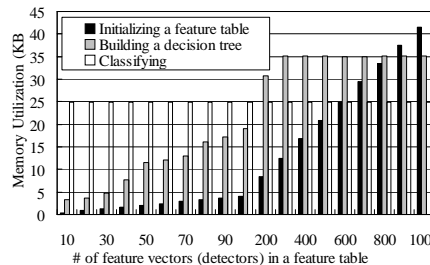
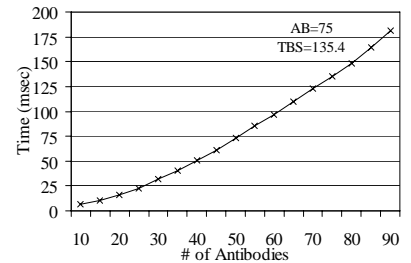
cation time, T_{classify} . In the results, T_{init} (both t1 and t2) gradually increases in proportion to the parameters. However, at runtime the EE facility only consumes an amount of time T_{classify} because we assume that the classification algorithm will not rebuild the decision tree; i.e. T_{init} is considered only once in the initialization step. Once the decision tree is given, iNet always uses the same decision tree classifier during runtime execution.

The accuracy of classification algorithm is measured by comparing the actual class of the current environment condition (i.e. the class that it should be in) with the classified result. The accuracy is also affected by two parameters: N and A , and how these parameters vary the accuracy is shown in Figure10. In general, a classification algorithm achieves high classification accuracy with more sample data and features (i.e. larger N and A). As explained above, the decision tree will not be rebuilt at runtime (although its accuracy might be improved by rebuilding). So, in this experiment, it is important to decide an appropriate size of the parameter N for both reducing overhead and improving classification accuracy. There is a trade-off between the efficiency and accuracy; so application developers need to determine the value of N , based on the experimental results shown in Table IV, depending on the requirements of their application.

Figure11 shows memory footprints which the EE facility consumes for (1) initializing a feature table, (2) building a decision tree and (3) classifying a feature vector. They are measured when N is 150 and A is 5 (i.e. guaranteed 85% classification accuracy from Table IV). The result indicates that the EE facility requires more memory space for a feature table as N increases (see black bar). Also it requires more space to build a decision tree until the tree is fully expanded (see gray bar; e.g. # of tree nodes is 364 when $A=5$ and $V=3$; V is # of distinct values of each feature, i.e. # of branches); however, the EE facility consumes a constant memory space (around 25KB, see white bar) for classification regardless of parameter values.

4.3 Performance of Behavior Selection Facility

The BS facility considers the time to initialize antibodies and the time to select a behavior (Section 3.2.2). However,

**Figure10. Accuracy of classification on 100 testing data when $A=3, 4, 5,$ and 6** **Figure11. Memory utilization that EE facility consumes****Figure12. T_{BS} , Overhead of BS facility, i.e. time for selecting a behavior****Table IV. Recommended size of feature table (underlined) according to the initialization overhead T_{init} and classification accuracy. (with $A=3$, DT couldn't achieve 90%)**

		# of features (A)	3	4	5
		# of possible combination of feature vectors	27	81	243
80%	T_{init} (sec)	t1	0.0015	0.0016	0.0034
		t2	2.21	3.913	9.874
		Total	2.212	3.915	9.877
		Size of feature table (N)	<u>20</u>	<u>40</u>	<u>75</u>
85%	T_{init} (sec)	t1	0.0015	0.0031	0.0094
		t2	2.767	7.121	16.101
		Total	2.769	7.124	16.111
		Size of feature table (N)	<u>25</u>	<u>70</u>	<u>150</u>
90%	T_{init} (sec)	t1	-	0.0034	0.0141
		t2	-	8.748	18.528
		Total	-	8.751	18.542
		Size of feature table (N)	-	<u>90</u>	<u>225</u>

the BS initialization does not affect runtime execution (like T_{init} in EE). So, the overhead of BS facility, T_{BS} , just takes the behavior selection time.

The number of antibodies, 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 $AB = (A*V)*B$. Figure12 shows how much time the BS facility spends to select a behavior. It is shown that the selection time exponentially increases as AB increases (i.e. with larger A , larger V , larger B). Figure13 shows memory footprints that the BS facility consumes for (1) initializing the antibodies and (2) selecting a behavior. The BS facility requires more memory space in proportion to the number of antibodies.

4.4 Performance Impact of the EE Facility on iNet

As described in the previous subsections, T_{classify} is dramatically fast as compared with T_{init} and also T_{BS} . For example, based on assumption that $V=3$ and $B=5$, only a parameter A affects the amount of T_{BS} . When $A=5$ (i.e. $AB=75$), $T_{\text{BS}}=135.41\text{msec}$ (Figure12) will be skipped if the EE facility spends $T_{\text{classify}}=3\text{msec}$ (Table III) and says that 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. For example, when A is 4, 5 or 6 in a feature vector, the classification needs only 4, 5 or 6 comparisons to search a leaf node (i.e. class value).

In order to verify how the EE facility effectively contrib-

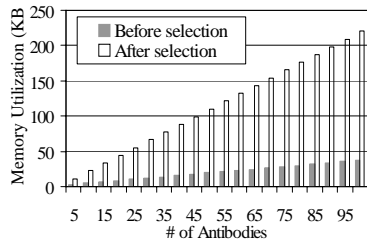


Figure13. Memory utilization that BS facility consumes

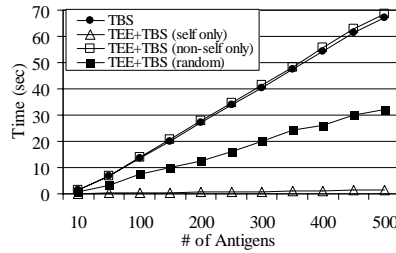


Figure14. Execution overhead of iNet according to three different scenarios ($A=5$)

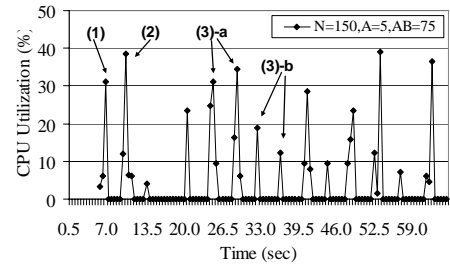


Figure15: CPU utilization that iNet consumes

utes to iNet in terms of efficiency, the overhead of the BS facility (T_{BS}) is compared with the overhead of executing both EE and BS ($T_{EE}+T_{BS}$) under the following three scenarios.

- **Self environment conditions only:** EE facility monitors only self environment conditions. This simulates a static network environment where the environment 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:** EE facility monitors 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:** EE facility randomly monitors 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.

Figure14 shows the execution overhead of iNet (i.e. both EE and BS) according to each scenario described above. X-axis, the number of antigens (i.e. environment conditions) each of which is monitored at particular time intervals, implies how long iNet is running. Y-axis indicates the accumulated time overhead of iNet. Figure15 describes CPU utilization that iNet consumes in the case of the third scenario (random antigens). The peaks indicates CPU usage for (1) initializing a feature table, (2) building a decision tree, (3-a) both classification and behavior selection (EE+BS), and (3-b) only classification (EE). It is shown that the execution of BS facility is skipped at some point (e.g. 3-b) where EE facility classifies a current environment condition as “Self”. These results (Figure14 and 15) exactly show that EE facility effectively works. If iNet does not have EE facility, then it always executes BS facility regardless of whether if an agent adapts to the environment well or not (i.e. the degree of adaptation). In other words, iNet will avoid unnecessary execution and computations thus save the selection time which might exponentially increase (Figure12), and resource consumptions such memory and CPU.

4.5 Autonomous Adaptability of Agents

This measurement evaluates autonomous adaptability of agents. In this measurement, 16 NetSphere platforms are

deployed on 8 PCs (i.e. 2 platforms per PC), and they are connected with each other based on a grid topology (Figure16-(1)). At the beginning of a measurement, a single web service agent is randomly deployed on a platform. Each web service agent contains a behavior selection engine that is configured with 5 behavior policies (i.e. antibodies) and 7 environment conditions (i.e. paratopes) shown in Table I. A workload generator generates HTTP request messages and randomly sends them to web service agents. It keeps track of their locations. When a web service agent migrates, the agent notifies its new location to the workload generator. The workload generator pays energy units to a web service agent when it receives a HTTP response message from the agents.

In Figure16, (2) shows the workload for web service agents, i.e. how many HTTP request messages are generated and sent to web service agents. The workload gradually grows in this measurement. (3) shows how the number of web service agents changes against the workload change. As web service agents replicate in response to enough energy gain from the workload generator, they autonomously change their population, up to 84 agents, so that they can process more HTTP request messages. (4) shows how many HTTP response messages web service agents send back to the workload generator. Since these agents migrate to platforms where resource availability is higher, they change their locations so that they can process HTTP request messages more efficiently.

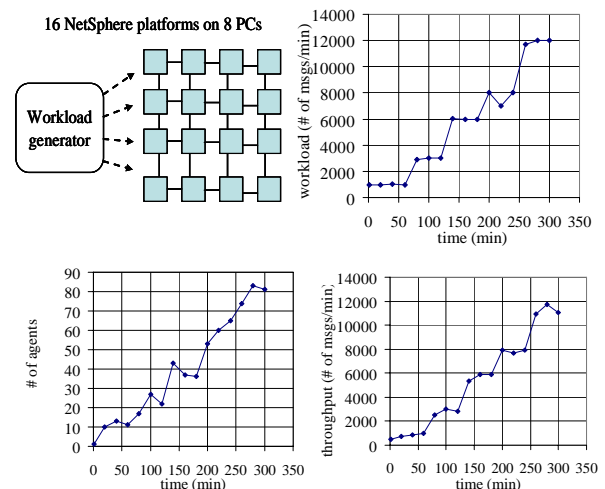


Figure16. (1) Testbed Architecture, (2) generated workload, (3) the number of agents, (4) throughput of agents.

5. Related Work

Artificial immune systems have been proposed and used in various application domains such as anomaly detection [10] and pattern recognition [11]. [10] mainly focuses on the generation of detectors for self/non-self classification and improves the negative selection process of the artificial immune system. [11] focuses on the accuracy for the matchmaking of an antigen and antibody. On the other hand, this paper proposes an artificial immune system to improve autonomous adaptability in network applications. To the best of our knowledge, this work is the first attempt to apply an artificial immune system to this automatic applications domain.

This work is an extension to the Bio-Networking project [6, 7, 8]. In the Bio-Networking architecture, each agent does not have the EE facility. It periodically performs one of its behaviors regardless of whether the agent adapts well to the current environment conditions. This results in wasting resources caused by unnecessary behavior selection. In the NetSphere architecture, each agent has the EE facility, which examines whether it adapts well to the current environment conditions. It activates the BS facility only when the agent does not adapt to the current environment. This way, agents can save resources and reduce execution overhead in their adaptation activities.

Similar work has been proposed in Organic Grid [12] project. [12] attempts to the decentralized task scheduling for large-scale computation on grid environment over the Internet. Similar to iNet, mobile agents autonomously executes their services (e.g. computing subtasks) on the platform embedded in each host and perform their replication behavior to achieve their objectives (e.g. compute as fast as possible). Yet, iNet project focuses on the adaptation of network applications. Not only a replication behavior but agents consider more behaviors for the adaptation. Because of those various adaptation decisions, the high service adaptability of network applications (i.e. agents) is achieved.

SORA [13] attempts to achieve efficient resource allocation in sensor networks. In SORA, an agent runs on each sensor node, based on economic principles, it selects sensor behaviors (e.g. sending/receiving messages and idling) adaptively to the current network environment. In NetSphere architecture, each agent adaptively selects its biological behaviors (e.g. migration, replication and death), based on immunological mechanisms. Also, SORA does not provide a mechanism like the EE facility. Each agent periodically selects one of its behaviors without examining whether it needs to adapt to the network environment by invoking the behaviors.

There are several research efforts that allow network systems to adapt to application and end-user requirements with a technique of runtime component replacement. For example, [14, 15] can dynamically replace running components (e.g. byte code) with others. Since they can up-

date any components, it is possible to change the functional aspect of components. iNet does not modify the body (functional part) of agents at runtime; it focuses on adapting the behavior (non-functional part) of agents in order to minimize runtime adaptation overhead generated by iNet. The overhead should be cheap because iNet assumes that each agent often perform its behavior to keep adapting to dynamically changing environments. It should be considerably expensive to perform even a single runtime component replacement. [14, 15] assumes a centralized network architecture where a centralized server collects environment conditions from each component and make component replacement decisions. In contrast, iNet assumes a decentralized network architecture where each agent monitors its surrounding environment conditions and makes adaptation decisions.

Similar to [16, 17], iNet contributes to develop an adaptive monitoring system for Grid computing. One main difference is that [16, 17] have a centralized environment monitoring system. Probes and gauges [16] or Sensor-hosts [17] can be distributed to collect environment conditions, but the environment conditions will eventually go to a centralized entity (e.g. Gauge Consumer). In our architecture, the environment sensing service on each platform (i.e. each host) collects and stores environment conditions, and makes them available to agents running on the same host. There are no centralized entities to store global environment conditions.

In addition, adaptation strategies and tactics tend to be more fine-grained when making systems more adaptable, resulting in the greater number of strategies and tactics. The greater the number of them, the more complicated and difficult it is to maintain and coordinate them. Fine-grained strategies and tactics are often not orthogonal with each other, but have complicated constraints with each other. [17] does not address the process to inspect the dependencies between strategies/tactics and coordinate/prioritize them. iNet provides a generic and consistent mechanism for each agent to identify its behaviors suitable for given environment conditions, prioritize them based on the relationships between them, and choose the most suitable behavior.

6. Concluding Remarks

This paper describes and empirically evaluates an immunologically-inspired mechanism that allows network applications to autonomously adapt to dynamic changes in the network. The proposed mechanism, called iNet artificial immune system, allows each application component to autonomously sense its local environment conditions to evaluate whether it adapts well to the conditions, and if it does not, adaptively perform a behavior suitable for the conditions. Measurement results show that iNet works efficiently and makes network applications adaptive.

Extended simulations are planned to investigate more performance implications of iNet, such as resource utilization

to execute the iNet immunological process. In addition, iNet will be evaluated on larger-scale experimental environments (e.g. PlanetLab²). It will provide more realistic results of the performance and adaptability of applications developed with iNet.

References

- [1] P. Dini, W. Gentsch, M. Potts, A. Clemm, M. Yousif and A. Polze, "Internet, Grid, Self-adaptability and Beyond: Are We Ready?," In *Proc. of IEEE Int'l Workshop on Self-Adaptable and Autonomic Computing Systems*, Aug. 2004.
- [2] 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*, Mar 2001.
- [3] R. Sterritt and D. Bustard, "Towards an Autonomic Computing Environment," In *Proc. of 14th IEEE Int'l Workshop on Database and Expert Systems Applications*, Sep 2003.
- [4] N. Minar, K. H. Kramer and P. Maes, "Cooperating Mobile Agents for Dynamic Network Routing," In A. Hayzelden and J. Bigham (eds.) *Software Agents for Future Communications Systems*, Springer, 1999
- [5] G. Cabri, L. Leonardi and F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications," In *IEEE Computer*, Feb 2000.
- [6] T. Suda, T. Ito 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 and W. Willinger (eds.) *The Internet as a Large-Scale Complex System*, Princeton University Press, June 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*, vol 23, 2005.
- [8] T. Nakano and T. Suda, "Adaptive and Evolvable Network Services," In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Incs. vol. 3102, Springer, pp. 151-162.
- [9] N. K. Jerne, "Idiotypic Networks and Other Preconceived Ideas," In *Immunological Review*, vol. 79, 1984.
- [10] F. A. González, D. Dasgupta, "Anomaly Detection Using Real-Valued Negative Selection," *Genetic Programming and Evolvable Machines*, 4(4): 383-403 (2003).
- [11] L. N. de Castro, J. I. Timmis, "Artificial Immune Systems: A Novel Paradigm to Pattern Recognition," In *Artificial Neural Networks in Pattern Recognition*, J. M. Corchado, L. Alonso, and C. Fyfe (eds.), SOCO-2002, University of Paisley, UK, pp. 67-84.
- [12] A.J. Chakravarti, G. Baumgartner, M. Lauria, "The Organic Grid: Self-organizing Computational Biology on Desktop Grid," In A. Zomaya (ed.), *Parallel Computing for Bioinformatics*, John Wiley & Sons, 2005.
- [13] Geoff Mainland, David C. Parkes, and Matt Welsh. "Decentralized Adaptive Resource Allocation for Sensor Networks (SORA)," In *Proc. of the 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005)*, May 2005
- [14] J. Appavoo, K. Hui, C. A. N. Soules, R. W. Wisniewski, D. M. Da Silva, O. Krieger, M. A. Auslander, D. J. Edelsohn, B. Gamsa, G. R. Ganger, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, and J. Xenidis, "Enabling Autonomic Behavior in Systems Software with Hot Swapping," In *IBM Systems Journal* 42, 2003.
- [15] C. Poellabauer, K. Schwan, S. Agarwala, A. Gavrilovska, G. Eisenhauer, S. Pande, C. Pu, and M. Wolf, "Service Morphing: Integrated System- and Application-Level Service Adaptation in Autonomic Systems," In *Proc. of the 5th Annual International Workshop on Active Middleware Services*, June 2003.
- [16] S. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, and N. Hu, "Software Architecture-based Adaptation for Grid Computing," In *the 11th IEEE Conference on High Performance Distributed Computing*, July 2002.
- [17] K. Shirose, S. Matsuoka, H. Nakada, and H. Ogawa, "Autonomous Configuration of Grid Monitoring Systems," In *the 2004 Symposium on Application and the Internet (SAINT2004)*, Japan, January 2004.
- [18] S. T. Brugger, "Data Mining Methods for Network Intrusion Detection," University of California, Davis.
- [19] P. Berkhin, "Survey of Clustering Data Mining Techniques," Accrue Software, Inc., San Jose, CA, USA, 2002.
- [20] T. Mitchell, "Decision Tree Learning," In *T. Mitchell, Machine Learning*, The McGraw-Hill Companies, Inc., 1997, pp. 52-78

² <http://www.planet-lab.org>