International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:4, 2010

Resource Constraint Mobile Agent Framework For Ambient Intelligence

Yung-Chuan Lee, Shahram Rahimi, and Bidyut Gupta

Abstract—In this paper, we introduce an mobile agent framework with proactive load balancing for ambient intelligence (AmI) environments. One of the main obstacles of AmI is the scalability in which the openness of AmI environment introduces dynamic resource requirements on agencies. To mediate this scalability problem, our framework proposes a load balancing module to proactively analyze the resource consumption of network bandwidth and preferred agencies to suggest the optimal communication method to its user. The framework generally formulates an AmI environment that consists of three main components: (1) mobile devices, (2) hosts or agencies, and (3) directory service center (DSC). A preliminary implementation was conducted with NetLogo and the experimental results show that the proposed approach provides enhanced system performance by minimizing the network utilization to provide users with responsive services.

Keywords—Ambient intelligence, load balancing, multiagent systems, ubiquitous computing.

I. Introduction

THE concept of ambient intelligence (AmI) is a vision of a world in which humans are surrounded by computing and networking technology unobtrusively embedded in their surroundings [8], [9]. Here, people live easily in digital environments in which the electronics are sensitive to people's needs, personalized to their requirements, anticipatory of their behavior and responsive to their presence. With the recent advancement of embedded device technology, this vision is merely a fiction but can be realized.

AmI is the convergence of three major key technologies: ubiquitous computing, ubiquitous communication, and interfaces adapting to the user [8], [9]. First of all, ubiquitous computing represents new types of computing devices invisibly embedded into our everyday environment. Rather than explicitly being the "physical user" of a computer, a human will implicitly profit from services running between computers without taking notice of them. Secondly, ubiquitous communication denotes that information-processing communication systems should be human-centric in the ubiquitous informational society. This would allow anyone to access any desired information, anytime, anywhere, easily and immediately. Finally, interface indicates that unlike today's interfaces which often allow only very limited forms of interaction with computers and require extensive specialist knowledge to operate, tomorrow's interfaces should be much more intuitive and responsive to prompts that include speech, vision and touch.

The proposed AmI framework [11] allows users to obtain their personal information such as preferences, profiles, likings and habits, while having minimum interactions with the surrounding environment. The information of the user will be available to him/her at the new location effortlessly. The system is formed from multiple geographically distributed environments. Each environment provides different services to users such as shopping, banking, etc. There are three major components in the environment: mobile devices, hosts or agencies, and the Directory Service Center (DSC). Note that an environment is a cluster that contains multiple mobile devices, multiple hosts and one or more DSC(s).

Due to limited capabilities of the mobile devices such as limited battery capacity, insufficient computing speed, small amount of memory, and a lack of input or output facilities, mobile agent technology is employed in AmI environments. Contrary to traditional client-server approach, agents can migrate into different hosts to perform computation, communicate with other agents, and bring the results back to its user [3]. Thus, users dispatch agents from portable devices and the majority of computations and power consumptions of agents are done on the hosts. Comparing to portable devices, hosts have ample computing capability and extensive power resource.

This paper presents a framework that utilizes mobile agents for ambient intelligence in a distributed ubiquitous environment. It provides users with personalized knowledge and intelligent interactions as well as sustains expeditious performance under dynamic resource demands. In this paper, we describe each component of the framework and formulate strategies to balance network and host demands. We then implement the framework and evaluate the merit of the proposed load balancing method.

II. Related Works

There are several studies that utilize mobile agents in Ambient Intelligent (AmI) environments. For example, Satoh employed RFID and software agents, creating location-aware service to provide personalized information to users [16]. Every user is assumed to carry a unique RFID tag which will be identified by the RFID sensors in each equipped location server. When the user enters into the location server's coverage area, the location server identifies the host with the needed

Yung-Chuan Lee is a PhD candidate and Computer Information Specialist with the Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901 USA (phone: 618-453-6051; fax: 618-453-6044; email: ylee@cs.siu.edu).

Shahram Rahimi is a Associate Professor and Director of Undergraduate Program with the Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901 USA (email: rahimi@cs.siu.edu).

Bidyut Gupta is a IEEE Senior Member and Professor with the Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901 USA (email: bidyut@cs.siu.edu).

capabilities for the user according to his preferences or profile. Then, a mobile agent will be assigned to each user and the agent moves from one host to another to assist the user.

A similar approach was developed by the AMILAB, in which they implemented the AMICO system to create an integrated AmI environment in manufacturing. There were three main functions provided by AMICO. First of all, AMICO provides useful contextual information to users based on three criteria: the user profile, the most suitable device and the current user location. Secondly, in the AMICO environment, users are allowed to access the machine functionalities according to the user context such as professions or skills. Finally, AMICO learns from the interactions with users and adapts them to the user's preference.

Furthermore, Costantini and others proposed the DALICA project, which utilizes a multi-agent system to enhance user interactions with culture assets and administering these assets [4]. Agents observe user's locations from a satellite signal and proactively assist users and propose culture assets according to their preferences. Users' preferences are then refined based on what culture asserts they visited and interacted with to improve future visits. Another main feature of DALICA project is to securely and actively monitor transportations of culture assets from one location to another through agents.

On the other hand, Zhang and colleagues proposed a context-aware AmI-Space based on a multi-agent architecture [19]. In their approach, users use portable devices to obtain services provided by the system such as controls of household appliances or multimedia service. The system can also automatically measure the room temperature and humidity, the user's psychological or physiological status, and provide personalized initiative services. Similar to AmI-Space, Hagras and colleagues proposed an ambient-intelligence approach called iDorm where it utilities embedded sensors, actuators, and software agents to construct an context-aware environment [7]. Users can interact with agents in the embedded controller, robots or mobile devices to control the environment. The system evolves from those interactions to provide a more precise and user-friendly living environment.

Several initiatives have attempted to build AmI environments. The Aware Home project constructs a home to produce an environment that is capable of collecting information about itself, and the whereabouts and activities of its inhabitants [10]. The Philips' home lab is a test bed for ambient intelligence that is more like a real home than a laboratory [14]. MIT's Intelligent Room and Oxygen project [12], Georgia Tech's Aware Home and eClass projects [5], [6], and Stanford's Interactive Workspace [2] are all attempts to study the impact of ubiquitous computing on education. They have built prototype classroom environments and the necessary software infrastructure to seamlessly capture much of the rich interaction that occurs in a typical university lecture, reduce the need for mundane note-taking and allow students to engage in better classroom discussion. Finally, the PERSONA and SOPRANO projects attempt to develop open standards and improvements to the AmI environments in order to create a seamless Ambient Assisted Living (AAL) environment for the elderly [13], [17].

None of the previously mentioned approaches have addressed the performance issue of the system or load balancing among agencies in an AmI environment. Because of the autonomy of the agents and the dynamicity of the environment, any approach should take the performance criteria into consideration to provide users with responsive services. Hence, based on our previous work [11], we propose a resource

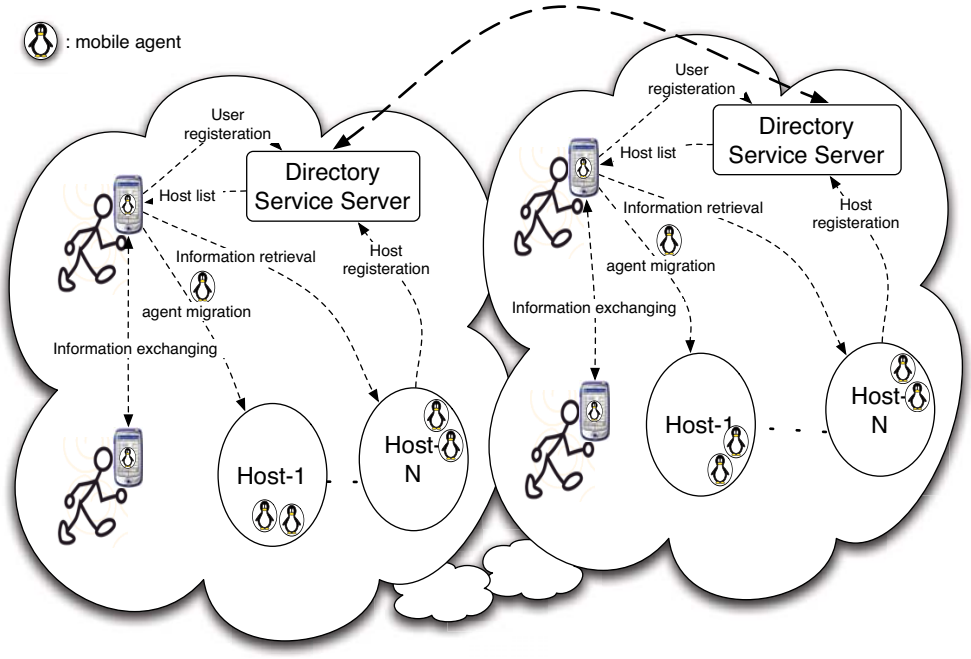

Fig. 1. Overview of proposed framework [11]

constraint agent-based AmI system with a focus on optimizing communication costs and load balancing among agents and agencies.

III. Proposed Methodology

Derived from our previous study on AmI environments [11], this work focuses on formulating the load balancing technique to evaluate its performance. In this section, we will begin by briefly describing the main concept and components of our framework to provide a foundation for the current study. We will then formulate algorithms in an attempt to minimize network bandwidth usage and maximize host utilization.

A. Overview of Framework

Figure 1 illustrates the overview of the framework. The system is formed from multiple geographically distributed environments. Each environment may provide different services to users such as banking, shopping, etc. Each environment consists of three main components: (1) mobile devices, (2) hosts or agencies, and (3) a directory service center (DSC). Different environments are connected through their directory service centers. When a mobile device or a host joins an environment, it registers itself to the DSC. Others can later inquire the DSC to obtain a list of preferred mobile devices and hosts to communicate with. In the following sections, we briefly describe each of the components.

1) Mobile Device: A mobile device is a pocket-sized computing device, typically with a display screen with touch input or a miniature keyboard. Smartphone is an example of such a device. In an AmI environment, we assume that each user carries at least one mobile device that holds his or her personal information. Four modules have been proposed in a user device: *Registration Module*, *Admin Module*, *Load Balancing Module* and *Learning Module*. The user profile as well as the hosts and agents' information are stored in the *Master Agent Database* of the mobile device. A master agent resides in the mobile device to communicate with the hosts and DSCs, handles user interactions, spawns and dispatches agents to agencies when needed, and administers dispatched agents. According to the user's goal, these dispatched agents will migrate to hosts to perform computations and services for its owner.

2) Host or Agency: Hosts or agencies provide facilities and services for mobile agents to migrate onto and perform various tasks for their owners. Two modules are defined in this component: *Registration Module* and *Information Module*. In addition, Java Application Development Framework (JADE) [1] is utilized to provide an agent runtime environment. Interactions between the host and user's agents are stored in a *User History Database* as references for the host to improve its services to the same user in the future.

3) Directory Service Center (DSC): The Directory Service Center (DSC) is responsible for managing mobile devices, hosts/agencies, and their intercommunication in its environment, as well as communicates with the DSCs in other environments. Two modules are defined in this component: *Registration Module* and *Communication Module*. Registrations of mobile devices and hosts are stored in a *Registration Database* to provide directory service to others.

B. Load Balancing Strategy

The *Load Balancing Module* in the mobile device is responsible for spreading the communication and computation loads among the hosts in the environment to minimize host service delay. Whenever the *Admin Module* in the mobile device creates a mobile agent, it forwards a list of candidate hosts for the mobile agent to invoke the *Load Balancing Module* to compute the optimal communication method and identify which host to communicate to. The list is constructed from its *Master Agent Database* and by querying the DSC in the network.

The *Load Balancing Module* then checks the current network usage from the DSC to determine whether migrating an agent to a host or traditional client-server approach is more beneficial. In addition, the module sends a message to each host in the list to obtain their resource utilization to determine their response time. It then estimates the cost of migrating the agent to each of these hosts and provides the most underutilized one to the *Admin Module*. Algorithm 1 provides a broad representation of the operations performed by the *Load Balancing Module*.

Because the load balancing task is performed by the individual mobile device, we expect that our proposed AmI framework will provide high scalability and reliability regardless of the ad-hoc dynamicity of AmI environments. In addition, based on user preferences, the *Load Balancing Module* obtains a snapshot of the system load and response time of interested hosts and then chooses an optimal host from this snapshot. However, this soft real-time operation does not prevent the possibility that agents of all users with the same requests migrate to the same host, especially when there are just few hosts in the network.

IV. Framework Implementation

The simulation of our approach was developed by using NetLogo software. NetLogo was developed by the Center for Connected Learning and Computer-based Modeling at Northwestern University [18]. It is written in Java and provides both modeling and simulation tools for agent systems. NetLogo utilizes a simple and efficient scripting language that allows entities to be controlled and their interactions with the environment to be described.

On the other hand, Netlogo does not provide a full-scale network emulator nor an agent communication language (*ACL*) for agents to communicate in its contained environment. To allow us to measure the network performance and study the proposed framework, we defined and constructed a simple network emulator to simulate sending and receiving packets. There is no communication protocol except for *ACK* and *NACK* packets for acknowledgement purposes, and several control packets used in the *Admin Module* to administer agents.

Algorithm 1 Load-Balancing(Hosts H)

```
Let nu be the current network usage
Let t be the threshold of Network-Usage
Let m be the communication method
Let h_best be current optimal host
if client-server mode is OK and nu is less than t then
    m = client-server mode
    h_best = NULL
else
    m = agent migration mode
    Let sl_best be the current lowest system load
    Let rt_best be the current shortest response time
    for every h_i in H do
        Let sl_i be the current system load of host h_i
        Let rt_i be the current response time of host h_i
        if Host priority is system load then
            if sl_i is less than sl_best then
                sl_best = sl_i
                rt_best = rt_i
                h_best = h_i
            end if
        else
            if rt_i is less than rt_best then
                rt_best = rt_i
                sl_best = sl_i
                h_best = h_i
            end if
        end if
    end for
end if
return (m, h_best)
```

A. Networking Simulation

NetLogo does not provide communication interface for agents to send and receive packets. Therefore, we defined a special agent, *Network Agent*, to simulate network facility of an environment and handle all communications among mobile devices, hosts and DSCs within that environment. Figure 2 depicts the flow of a regular communication between a sender and a receiver through the network agent. In our implementation, the *Network Agent* employs a FIFO queue to store and route packets. To simplify the implementation complexity of the *Network Agent*, we loosely defined a packet structure in which the payload size is flexible and can be divided into any size to completely utilize the network bandwidth if possible.

A packet is constructed with 6 fields: operation, sender ID, receiver ID, message ID, message content and message size. The operation field indicates the type of the packet, and can be either *forward_msg* or *send_msg*. The *forward_msg* packets are sent from senders to the *Network Agent* to be "forwarded" to the receivers. On the other hand, any packet from the *Network Agent* is always the *send_msg* type since it is "sent" directly to receivers. The message ID is used for simple referencing and identification. The message content field is the actual payload of the packet and the message size field indicates the size of this packet.

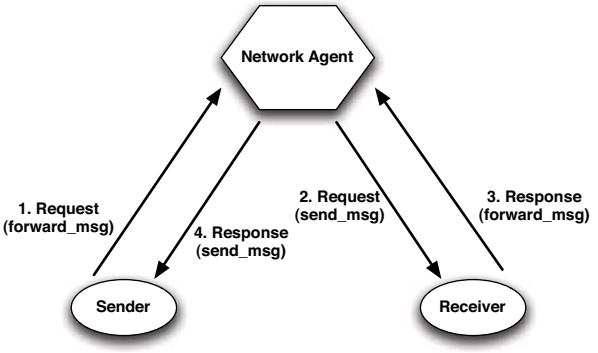

Fig. 2. Flow of a communication routed through a Network Agent

Two special types of packet, the *ACK* and *NACK*, act as *Yes* and *No*, respectively, to provide interactions between the users and hosts (i.e., to inquire the progress of migrated agents). More details are given later in the *Admin Module* subsection. Since both *ACK* and *NACK* packets are control packets, their size are assumed to be 1 unit.

Because of this simulated network facility, the network performance is now two-fold. On the one hand, agents, hosts and DCS communicate with each other using *forward_msg* function. On the other, the *Network Agent* delivers the packets using the *send_msg* method. Hence, the *Network Utilization (NU)* can be calculated as follows:

$$NU_i = \frac{\sum_{j=1}^{i}(S^j_{forward\_msg} + S^j_{send\_msg})}{T_i * NB}$$

where S^j is the size of the packets sent via *forward_msg* and *send_msg* methods at time j, T_i is the total time elapsed at time i, and *NB* is the network bandwidth of which is one of the input parameters.

B. Migration Module

To simulate agent migration and execution on hosts, we introduced a system load variable on each host. When an agent "migrates" to a host, the host actually receives a packet in which the message content is the initial execution time of the migrated agent and the message size is the size of the agent. After the host receives an agent, it puts the agent in the *RemainingJobList* queue and "executes" all agents in the queue at each iteration. Once an agent finishes its execution, if the agent does not request more execution time, the host puts the agent into the *FinishedJobList* queue. The agent execution scheduling uses the *Round Robin method* and each time slice (*TS*) of host *j* at iteration *i* can be computed as follows:

$$TS^j_i = \frac{h^j_{CPU}}{size(h^j_{RemainingJobList})}$$

where h^i_{CPU} is the number of instructions that host *i* can process per iteration and $size(h^i_{RemainingJobList})$ is the size of the *RemainingJobList* queue of the host *i*.

The system load (*SL*) of host *j* at iteration *i* can then be updated as follows:

$$SL_i^j = \frac{\sum_{t=1}^{i} \sum_{m=1}^{size_t(h^j_{RemainingJobList})} time(a_m^j)}{T_i * h^j_{CPU}},$$

$$time(a_m^j) = \begin{cases} execute\_Time, & if\ execute\_Time \leq TS_i^j \\ TS_i^j, & otherwise \end{cases}$$

where *execute_Time* is the remaining execution time of agent a_m on host j.

C. Admin Module

Agent inquiry and retrieval functions are implemented in the *Admin Module*. Whenever an agent migrates to a host, the master agent keeps the agent ID, host ID and agent size in the *ObjectInfoList* table inside *Master Agent Database*. The owner of an agent can inquire the status of the mobile agent object by sending a request containing the "agent ID" to the migrated host. After the host receives the inquiry request, it checks the ID of the mobile agent objects in its *FinishedJobList*. If the searched object exists in the *FinishedJobList*, it will reply with the *ACK* status. If it does not find the object in the *FinishedJobList*, it will send a message with the *NACK* status.

A user might wish to retrieve his mobile agent object from the host to access the data held by the agent. In this case, the user sends a retrieval request with the "agent ID" to the host to retrieve the object. After the host receives the request, it tries to locate the object by first searching the mobile agent object in the *FinishedJobList*. If the agent exists in the *FinishedJobList*, it replies with the *ACK* status along with the requested agent. On the other hand, if the object exists in the RemainingJobList, the host sends a reply message with the requested agent and the *NACK* status.

D. Registration Module

The concept behind the mobile agent registration is that whenever a mobile agent moves from one cluster to another, the mobile agent submits its registration details (unique id, specifications or requirements) to the DSC of the new cluster. The DSC stores the registration details of the mobile agent in its Registration database. The whole registration communication is processed through the *Network Agent*.

On the other hand, the hosts are stationary and assumed to be inside the cluster. When each host initializes, it registers itself to the DSC with its host ID and service profile (capabilities or specialties). Host registration is also routed through the *Network Agent*. Table I illustrates the communication details of this registration process.

Through this registration process, the DSC then provides a directory facility to allow all users to inquire host services as well as other users. In general, user to user communication does not impose high volumes of network traffic. Thus, in this implementation, we ignore user to user communications and concentrate on the performance comparison with different ways to access host services.

TABLE I
AGENT/HOST REGISTRATION

From Agent/Host to Network Agent

```
ask user/host [
   ask Network [
      (forward_msg sender DSC sender-msg msg-id
        msg-size)
   ]
]
```

From Network Agent to DSC

```
ask Network [
   ask DSC [
      (send_msg sender DSC sender-msg msg-id
        msg-size)
   ]
]
```

From DSC to Network Agent

```
ask DSC [
   ask Network[
      (forward_msg DSC receiver reply-msg msg-id
        msg-size)
   ]
]
```

From Network to Agent/Host

```
ask Network [
   ask receiver [
      (send_msg DSC receiver reply-msg msg-id
        msg-size)
   ]
]
```

V. FRAMEWORK SIMULATION

To evaluate the performance of the proposed framework, we conducted experiments on three different communication methods: a traditional client-server approach, a pure agent migration and our proposed load balancing method. The client-server approach requires a continuous connection among the users and hosts throughout the entire requested session. On the other hand, the agent migration technique requires only a split second of network connection to migrate an agent to a host, and the agent with the computed results can be retrieved later.

To the best of our knowledge, none of the current agent frameworks for AmI environments consider a host's resource utilization before migrating an agent. As a result, a host providing similar services in an AmI environment might be full of agents while others only have a few. Hence, with negligible computation costs to each mobile device to perform load balancing, we expect our method to produce a better network and host utilization in an AmI network.

A. Experiment Setup

For simulation purposes, we simplify the AmI environment to only contain one DSC with multiple hosts and users. By eliminating the directory synchronization among the DSCs and

the inter-environment agent migration problems, the experimental results obtained contain less noise and can be served as a baseline. More sophisticated AmI environments can be derived from this baseline.

To create reasonable network usage for the client-server approach, we assume that users would continuously interact with the hosts until they fulfill their request. The sum of the network usage per request is then defined as one experimental packet size. Furthermore, the implementation of the simulating agent size, agent execution time, and data size computed by each agent is randomly generated based on the experimental packet size to simulate a realistic AmI environment.

There are 500 users and 25 hosts in our simulated AmI environment. Experimental results are obtained in regard to 4 different experimental packet sizes, 100, 200, 500 and 1000. Other simulation parameters are shown in Table II.

Figure 3 illustrates the simulation model of the framework in NetLogo. There are input parameters on the left and realtime network bandwidth usage in the right. The realtime network utilization is plotted on the lower left and the realtime overall system load is plotted on the lower right. The realtime system map is in the center window in which the human icons represents users, house icons denotes hosts and smiley-face icon in the center of the white area symbolizes the DSC. In each iteration, each user randomly walks and turns, and a house icon is covered by a human icon when a user "walks into" it. When a user is inside an AmI environment, it randomly communicates with the DSC and requests services from the hosts. Instead of assuming that the whole system map is an AmI environment, the white area indicates the coverage of the simulated AmI environment to simulate the registration

TABLE II
EXPERIMENT INPUT PARAMETERS

Number of Users: 500
Number of Hosts: 25
Number of DSCs: 1
Number of iterations to run: 10000 units
Network Capacity: 1000 units
ACK/NACK package size: 1 unit
Experimental packet size: 100, 200, 500 and 1000 units

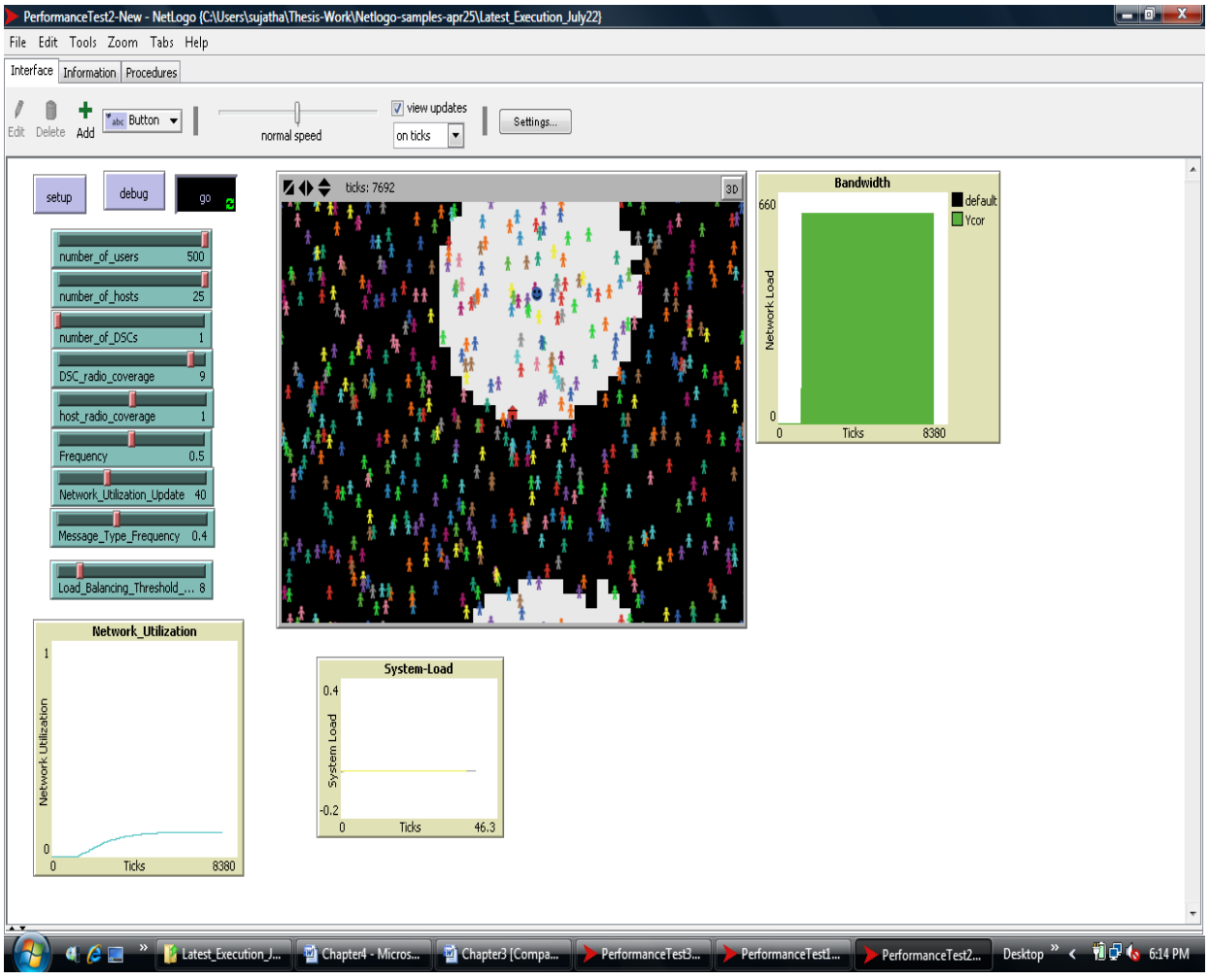

Fig. 3. Simulation Screenshot

process mentioned earlier. The radius of this white area can be changed using the *DSC_rado_coverage* input parameter on the left.

B. Experiment results

Figure 4 shows the performance comparison among the three approaches with 4 different experimental packet sizes. Compared to the other two approaches, our load balancing method results a better performance in each packet size. The less the network utilization, the more efficient the approach is. When the network activity is scattered (i.e., experimental packet size is 100), the performance is only slightly enhanced; however, there are extra load-balancing computation costs on each user's mobile device. On the other hand, when the network resource demand is high (i.e., experimental packet size is 1000), the performance is significantly better. Even through our load balancing method does not prevent exponential network demand in which the network is flooded with packets, it provides better stability and scalability than the other two approaches.

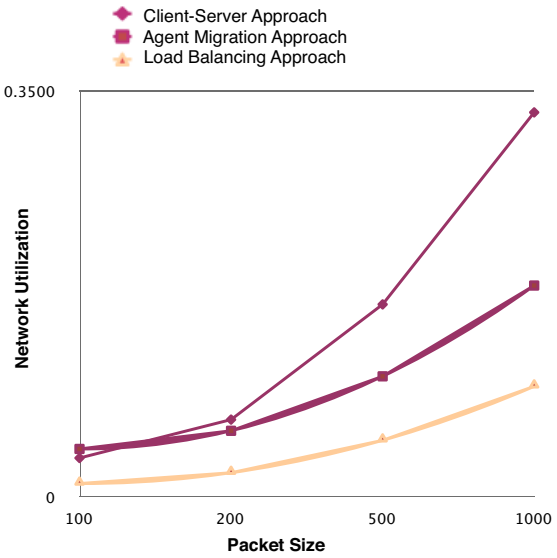

Fig. 4. Packet Size vs. Network Utilization

Every host in the simulated AmI environment maintains its own realtime system load information. Since we do not use mobile agents in the client-server approach, Figure 5 only shows the comparison between the agent migration approach and the load balancing approach. Since we did not model the system usage for client-server communication, some of the hosts in the load balancing approach will result in a near 0 system load when the load balancing module uses client-server communication rather than agent migration. The hosts in the simulated AmI environment are created with overlapping services. Thus, we expect to see similar system loads among all hosts.

From Figure 5, we see that there are several hosts with higher loads in the agent migration approach. This occurs when the majority of agents migrates to the same host. This outburst situation is managed as expected in our load balancing approach. However, as mentioned previously, the load balancing module obtains a snapshot of the current network utilization and the system loads of each interested host, and determines which method to utilize in a split second. The result of our proposed approach may be the same as that of the traditional agent migration approach under extreme and unrealistic situations (i.e., immensely high user-host ratio with constant agent migration).

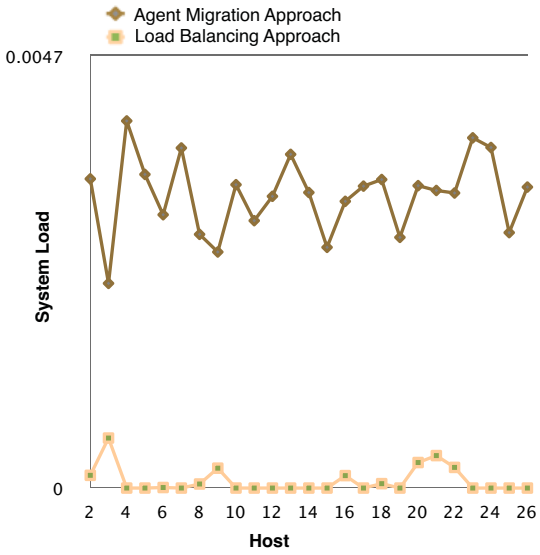

Fig. 5. System Load Comparison

VI. Conclusion

The purpose of this study is to implement and evaluate the general purpose of the mobile agent framework for ambient intelligence (AmI) environments [11]. From our simulated results, we confirm that it is beneficial to implement the load balancing method in an AmI system. The load balancing method can provide better network stability and extends the AmI system scalability by maintaining network utilization. Furthermore, the extra costs introduced by this method are very minimal (i.e., inquiring current network usage from the DSC and current system load from interested hosts to determine which communication method to use).

In addition, our approach also ensures that hosts with the same services in an AmI system are utilized evenly. Current research neglects this topic and mainly focuses on the interactions between users and hosts in an AmI environment. Although the host utilization may not be an immediate issue, it massively impacts service availability of the AmI systems when a full-scale AmI implementation is deployed into our daily life.

In this preliminary implementation, we assumed a simplified AmI system by eliminating directory synchronization among DSCs and agent migration from one AmI system to another. In the future, we need to create multiple AmI environments in our simulation model to study the effects of these two features.

The system resource usage of client-server communication of a host should also be modeled to provide a more comprehensive and accurate system load. Inter-environment agent migration needs to be investigated to validate our proposed method under multiple AmI environments. Ultimately, a realistic AmI environment will be implemented and deployed to conduct an empirical study to verify the merit of our framework.

REFERENCES

[1] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, "Jade Programmer's Guide," http://sharon.cselt.it/projects/jade/, 2002.
[2] J. Borchers, M. Ringel, J. Tyler, and A. Fox, "Stanford Interactive Workspaces: A Framework for Physical and Graphical User Interface Prototyping," http://hci.stanford.edu/research/istuff/iRoom_SmartHomes.pdf.
[3] P. Braun and W. R. Rossak, *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. San Fransisco: Morgan Kaufmann, 2004.
[4] S. Costantini, L. Mostarda, A. Tocchio, and P. Tsintza, "DALICA: Agent- based ambient intelligence for cultural-heritage scenarios," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 3441, Mar. 2008.
[5] Georgia Tech., "Aware Home Research Initiative Residential Laboratory at Georgia Institute Of Technology," http://awarehome.imtc.gatech.edu/research.
[6] Georgia Tech., "eClass," http://www.cc.gatech.edu/fce/eclass/index.html.
[7] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an Ambient-Intelligence Environment Using Embedded Agents," *IEEE Intelligent Systems*, Vol. 19, pp. 12-20, Nov. 2004.
[8] ISTAG,"Scenarios for Ambient Intelligence in 2010," http://www.cordis.lu/istag.htm.
[9] ISTAG, "Ambient Intelligence: from vision to reality," http://www.cordis.lu/istag.htm.
[10] C. Kidd, G. Abowd, C. Atkeson, I. Essa, B. MacIntyre, E. Mynatt, and T. Starner, "The Aware Home: A Living Laboratory for Ubiquitous Computing Research," in the *Proceedings of the Second International Workshop on Cooperative Buildings*, 1999, pp. 191-198.
[11] Y. Lee, E. S.Khorasani, S. Rahimi, B. Gupta, "A Generic Mobile Agent Framework for Ambient Intelligence," *Proceedings of the 2008 ACM Symposium on Applied computing*, 2008, pp. 1866-1871.
[12] MIT, "MIT Project Oxygen Pervasive Human-Centered Computing," http://www.oxygen.lcs.mit.edu/Overview.html.
[13] PERSONA, http://www.aal-persona.org/.
[14] Philips Research, "Philips Research Technologies," http://www.research.philips.com/technologies/misc/homelab.
[15] M. A. Perez, L. Susperregi, I. Maurtua, A. Ibarguren, F. Tekniker, and B. Sierra, "Software Agents for Ambient Intelligence based Manufacturing," *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Application*, 2006.
[16] I. Satoh, "Software Agents for Ambient Intelligence," *IEEE International Conference on Systems man and Cybernetics*, 2004.
[17] SOPRANO, http://www.soprano-ip.org/.
[18] U. Wilensky, "Netlogo". *Center for Connected Learning and Computer-based Modelling*, Northwestern University, http://ccl.northwestern.edu/netlogo.
[19] Y. Zhang, Y. Hou, Z. Huang, H. Li, and R. Chen, "A context-aware AmI system based on MAS model," *IEEE Proceedings of the 2006 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2006, pp. 703-706.

Yung-Chuan Lee received the B.S. and M.S. degree in computer science from the Southern Illinois University, Carbondale, in 2002 and 2005, respectively.

He is currently a PhD candidate and Computer Information Specialist in computer science at the Southern Illinois University, Carbondale. His research interests include distributed computing, bio-inspired intelligence, ubiquitous computing, and trustworthy computing.

Shahram Rahimi received two M.S. degrees in Computer Science and Engineering and the Ph.D. degree in computational sciences from Stennis Space Center/University of Southern Mississippi, in 2002.

He is the Undergraduate Program Director and an Associate Professor of computer science at Southern Illinois University Carbondale. He is the editor in chief for International Journal of Computational Intelligence theory and Practice and the associate editor for Informatica. He has over 130 peer reviewed publications in computational intelligence and distributed computing areas.