Agent-Based Embedded Monitoring System for Ubiquitous Networks Environments*

Hyo-Sung Kang[1], Jong-Mu Choi[1], Jai-Hoon Kim[1], and Young-Bae Ko[2]
hskang@dmc.ajou.ac.kr, {jmc, youngko, jaikim}@ajou.ac.kr
[1]Graduate School of Information and Communication,
[2]School of Information and Computer Engineering
Ajou University, SOUTH KOREA
Tel: +82-31-219-2443, Fax: +82-31-219-1614

Abstract— **Contrary to current wireless network systems, ubiquitous network environment has quite dynamic and diversified devices which aim at best meeting the requirements of specific group of users for a specific purpose. In order to obtain the seamless transparency within mobile devices, accurate and efficient monitoring of dynamically changing environments is one of the most important requirements for ubiquitous network environments. In this paper, we describe the novel agent-based embedded monitoring systems for wireless network devices. By utilizing the light-weight mobile agent technology, our monitoring system can be a useful tool for ubiquitous network environments. This agent-based approach, moreover, provides seamless transparency. Our proposed monitoring system can automate the installation and updating process of monitoring code. We also implement our agent-based monitoring system on embedded Linux-based wireless ad hoc network testbed.**

Keywords— Embedded Systems, Ubiquitous Computing, Mobile Ad Hoc Networks, Resource Monitoring, Networks Managements

I. INTRODUCTION

Ubiquitous network environments are generally composed of mobile, small, embedded, hand-held, and wearable devices, which have limited computational and network resources, even if they are standalone [1]. And most devices are connected to each other through wireless mediums. On this account, constructing a network and utilizing their resources efficiently are much harder than normal wired environment. To exploit these devices with limited resources, it is essential to monitor resources' usage. Resource monitoring enables to facilitate various valuable works. For example, monitored battery information can be used to operate with many energy-aware protocols [2][3] efficiently. Unfortunately, the task of monitoring software for ubiquitous devices has attracted little attention so far.

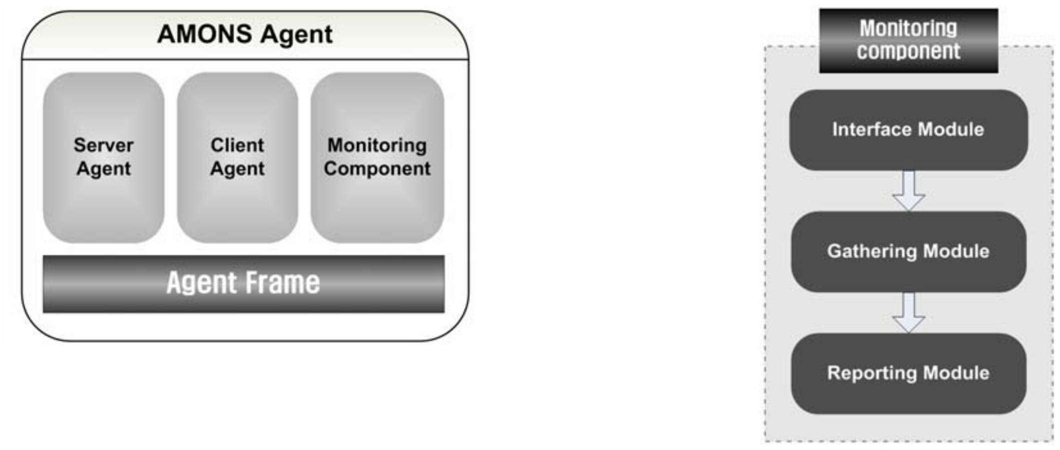
This work is supported by a grant of the 21C Frontier R&D Program, Ministry of Science & Technology in South Korea, ITA Professorship for Visiting Faculty Positions in Korea (International Joint Research Project) by Ministry of Information & Communication in South Korea, and grant No. (R01-2003-000-10794-0) from the Basic Research Program of the Korea Science & Engineering Foundation.

On account of ubiquitous network systems' inborn feature, it consists of many kinds of heterogeneous devices and each device has diverse context information. In this paper, we especially focus on resource usages such as wireless network bandwidth, memory, and CPU usage. There are, moreover, various operating environments in ubiquitous computing environment, such as various kinds of operating system and its different version. Whenever we need a system having different context information, such as CPU load, memory usage, it is very inefficient to design another system. In other words, it is unreasonable that we design one more system per every each required context information.

Previous works on developing wireless network monitoring systems [4][5], however, are not suitable for applying to the ubiquitous network environment. Since they do not follow component-based approach, it is not suitable for coping properly with numerous devices and manifold context information in ubiquitous network environment. It also can not show its monitored results in real time. That is, its monitored results can not be reflected promptly. Because of these problems, we have developed embedded wireless network monitoring system for ubiquitous environments. Our monitoring system has following characteristics.

- **Agent-Based** [6][7]: Mobile agent technology can provide a good framework to develop monitoring systems for ubiquitous network environment, since it can do complicated works on behalf of a node independently and transparently. Agent-based approach, therefore, is quite proper to our proposition. In our infrastructure, the role of agents is maintaining diverse components to send and receive.

- **Component-Based** [8][9]: The component-based systems must be both lightweight and perform reasonably well. That is, component-based software must be small size. By presenting our lightweight component model, small embedded devices with limited storage resource can be applied for ubiquitous network environment. And ubiquitous network environment with numerous nodes becomes more scalable. Thus, we believe that our lightweight



(a) Structure of AMONS Agent (b) Composition of monitoring components

Fig. 1. Overall Structure of AMONS

component-based approach is very appropriate and meaningful in ubiquitous network environment.

- **Real-Time**: It is quite significant that monitored information arrived timely. Because already past information may be useless on the most occasion. Our proposed system, thus, supports sending monitored information immediately and showing them in real time.

To our best knowledge, this paper is the first try to set up an agent-based monitoring framework. Since our monitoring system framework has, moreover, very practical and scalable approaches, we believe that applying our works to real world is easy.

The remainder of this paper is organized as follows. In the next section, we present a structure of our proposed agent-based monitoring system, called AMONS. Section 3 describes a protocol of AMONS to maintain monitoring components in details. Section 4 describes its implementation environment including our testbed, actual implementation of AMONS. Finally, we conclude our works with AMONS' strength points and its limitations in section 5.

II. STRUCTURE OF AMONS

The current implementation of our system is based on mobile agent systems. As shown in Fig. 1(a), our system has the following four components:

- **Agent Frame:** Agent frame is in charge of managing agent components, communication between agents, and downloading required monitoring components. In addition, agent frame with all monitoring components should response with client's request of monitoring components and transfer them to client agent.
- **Server Agent:** Main role of server agent is to request monitoring resources and analyze monitored information to provide helpful information to higher layer's operation. This information can be used for seamless transparency between higher layer and agent frame.
- **Client Agent:** Client agent should response with server agent and determine whether it has corresponded monitoring component or not. Without corresponded monitoring components, it should request to server agent for required components.
- **Monitoring Components:** Monitoring component's main roles can be divided into two parts. One is to monitor and gather raw resource information actually and another is to reporting the monitored results to server agent.

An AMONS monitoring component's internal structure is depicted in Fig. 1(b). It shows that an AMONS monitoring component is composed of three parts.

- **Interface Module:** Interface module plays a role to receive information about what kinds of information should be monitored and monitoring arguments from a client agent, and passes them over to parsing module.
- **Information Gathering Module:** Information gather module can obtain raw data by monitoring, according to specified monitoring type and arguments received from interface module. And then, parses them to semantic monitored results.
- **Reporting Module:** Reporting module transfers results of monitored information from gathering module to the client via agent frame.

Table 1. AMONS monitoring information and their description

Components	Function	Descriptions
Network Resources Collector	Received Packet	The total number of packets of data received by the network device.
	Transmitted Packet	The total number of packets of data transmitted by the network device.
	Error Packet	The total number of transmit or receive errors detected by the device driver.
	Dropped Packet	The total number of packets dropped by the device driver.
	Collision Packet	The number of collisions detected on the interface.
System Resources Collector	CPU Load	The number of jobs in the run queue or waiting for disk I/O averaged over some time.
	CPU Usage	The amount of time spent in idle process out of the uptime of the system
	Total Memory Size	Total memory size installed in the system
	Free Memory	Remained memory size in run time.
	Cached Memory	Cached memory size in run time.
	Remained Energy	Remained energy amount in run time.
System Information Collector	O/S	Installed operating system's official name
	CPU	Installed CPU's official name
	Clock Speed	Installed CPU's clock speed
	Uptime	Elapsed time from bootstrap of the system

A component can be implemented independently in accordance with what information each component requires. These monitoring components are described in Table 1. This information can be a good norm that let us know whether it can provide services or not. Our implemented components can obtain this information easily from in Linux /proc file system [10].

III. AMONS PROTOCOL

The AMONS protocol is a communication protocol between server agent and client agent in order to send and receive required components. All of commands using in AMONS protocol and each command are described in Table 2.

Protocol for AMONS work as follow, first monitoring server sends CONN message to monitored clients. Now, two different cases **CASE 1** and **CASE 2** can be occurred whether the client has requested monitoring component or not.

CASE 1 "PACK" scenario: A client has all components that a server requires. It proceeds, thus, monitoring procedure immediately.

CASE 2 "NACK" scenario: A client does not have all components that a server requires. In advance, a client has to download required components from the server.

Table 2. AMONS protocol's commands and their descriptions

Command	Descriptions
CONN (Connect)	A server intends to monitor a client. This is included in a field denoting what kind of information the server requires.
PACK (Positive Acknowledge)	A client has the component which the server requires. It may proceed to the next procedure.
NACK (Negative Acknowledge)	A client does not have the component which the server requires. In advance, a client should download required components.
MSTA (Monitoring Start)	A client has all components the server requires. A server is ready to receive monitored results.
FFIN (File Transfer Finish)	Component transfer is finished. That is, the client has required components.
RSLT (Result)	Monitored result. This is one-way message from a client to a server.
RFIN (Result Transfer Finish)	Scheduled monitored result transfer is finished. The whole connection may be disconnected.

CASE 1: "PACK" scenario

1. As a monitoring server requires monitoring other client's information, it sends a "CONN" message to the client firstly. A "CONN" message contains "Service Type" field denoting what kinds of monitoring information the server requires, such as network bandwidth.

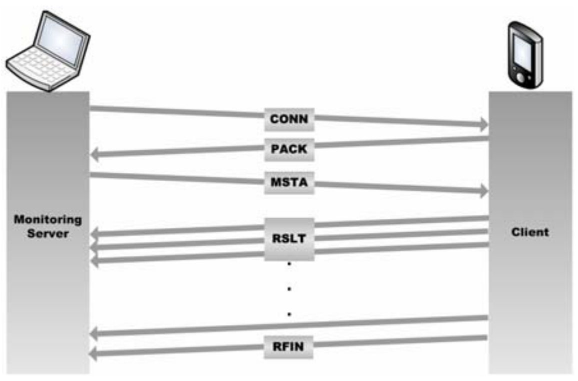


Fig. 2. "PACK" scenario for AMONS

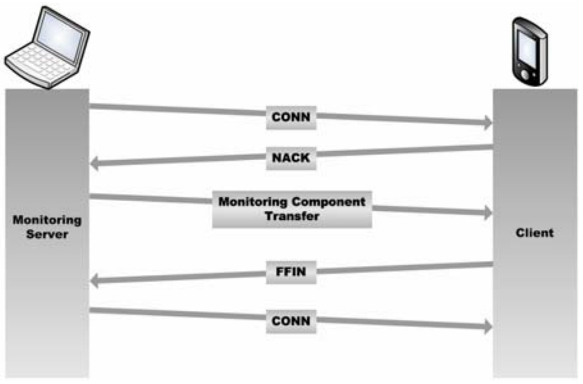


Fig. 3. "NACK" scenario for AMONS

2. The client received this request checks whether it has a corresponding component or not, according to "Service Type" field. With the component, the client sends a "PACK" message to the server.
3. Then, the server sends a "MSTA" message to the client with some arguments, such as whole monitoring time and monitoring period. According to these arguments, the corresponding component sends results of monitoring in multiple "RLST" messages.
4. After finishing sending "RSLT" messages, the client sends a "RFIN" message. As the server receives this "RFIN" message, a whole connection is finished.

CASE 2: "NACK" scenario

1. As the client receives "CONN" message, without required component, the client sends a "NACK" message to the server.
2. As the server received a "NACK" message, it transfers the corresponding monitoring component, according to its "Service Type" field.
3. After finishing the file transfer, the client sends a "FFIN" message.
4. The server received a "FFIN" message sends a "CONN" message again. That is, a "PACK" scenario is performed automatically.

IV. IMPLEMENTATION

Currently, Linux does not provide an ad hoc routing protocol on its original version. Hence, we add an external

AMONS protocols need to follow the special operations listed below to monitor the specified resources. Detailed scenarios for **CASE 1** and **CASE 2** are depicted in Fig. 2 and Fig. 3, respectively.

We modified ad hoc routing implementation to existing Linux kernel. This add-on implementation fully provides AODV (Ad hoc On-Demand Distance Vector Routing) [11] functions stated in AODV draft 13. In this section, we describe our actual implemented results. Firstly, we describe AMONS implemented environment. After that, we address the server agent which shows results by GUI environment.

A. Implementation Testbed

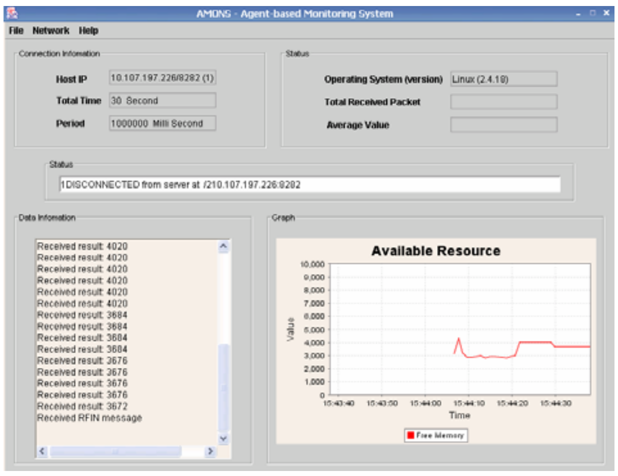
Our embodied testbed is depicted in Fig. 4. Every device in our testbed is equipped with IEEE 802.11b [12] compatible wireless LAN card and set to ad-hoc mode. We adopt AODV-UU [13], AODV routing protocol implementation created at Uppsala University, as ad hoc routing protocol.

Linux /proc file system is memory file system which can provide various kinds of information about the system, such as network bandwidth, memory usage, CPU usage and average system load. Thus, we can easily obtain network resource information as well as system information by simply parsing this `/proc` file system. Our implemented components are operated in the same manner.

Currently, there are only Linux-based components in implemented components. However, it is easily applied to other operating systems (such as Pocket PC [14], VRTX [15], Symbian [16], and TinyOS [17]). They work properly once their agents adopt AMONS protocol and components developed under their development environments. In this manner, component-based framework enables to apply to various environments. And this feature also gives seamless transparency to our system.

B. GUI Environments

As ubiquitous environments have various dynamic situations, one node should perform multiple roles. In other words, a server should be able to function as a client and vice versa. Thus, we implement server agent's GUI environment by using Java AWT [18], in order that a client with JVM (Java Virtual Machine) can function as a server. Nevertheless, only nodes having all components (or components required to monitor) can function as a server.



Host IP: 10.107.197.226/8282 (1)
Total Time: 30 Second
Period: 1000000 Milli Second
Operating System (version): Linux (2.4.18)
Total Received Packet:
Average Value:

Status: 1DISCONNECTED from server at /210.107.197.226:8282

```
Received result 4020
Received result 4020
Received result 4020
Received result 4020
Received result 4020
Received result 4020
Received result 3684
Received result 3684
Received result 3684
Received result 3684
Received result 3676
Received result 3676
Received result 3676
Received result 3676
Received result 3672
Received RFIN message
```

Fig. 4. AMONS monitoring server's GUI environment

We describe about this problem later in Section 5, conclusion and future work.

As a client agent sends monitoring results to a server in real time, a server agent's GUI environment shows its results to users in real time, too. Fig. 4 is an actual example which shows that server agent is receiving a client's physical memory usage. Most of previous work can not show the monitored results in real time. They can show the results only after their connection. Because it collects information over a period of time and then displays monitoring information over that past period of time. Fig. 5 shows that an AMONS monitoring server's GUI shows specific host's physical memory usage in kilobytes in real time.

Before starting a monitoring procedure, monitoring server should set some arguments such as monitoring client's IP address, total monitoring time and monitoring period. Total monitoring time can be set in seconds and monitoring period can be set in milliseconds. Heavy monitoring procedure can make itself burden. Thus, we implement that a server agent can set monitoring period minutely. These arguments are sent to a client agent. And then, a client agent monitors based on these arguments. When exchanging initial messages between a server agent and a client agent, additionally, a server agent obtains client agent's operating system's type and its version and displays them in server agent's GUI environment.

V. Conclusion and Future Works

We have presented an agent-based and embedded monitoring system (AMONS) framework for ubiquitous network environment. An AMONS allows monitoring users to monitor out-of-range nodes though multi-hop ad hoc networks in real time. Besides, our monitoring system facilitates various useful works such as energy-aware computing. Since our proposed monitoring system is fully specialized by using module, its size is very small and compact. It is a very appropriate and viable feature in ubiquitous network devices.

And also, our monitoring prototype can be referenced by developers who intend to manufacture commercial monitoring products as well as other ubiquitous computing researchers. We believe that our works can help set up protocol standard for monitoring system in ubiquitous network environment.

As future works, firstly, an AMONS monitoring server demands for all components of every monitoring server. Without required components, an AMONS can not work on its framework. Our implementation, however, is adopting AODV routing as network layer protocol. It means that a node can obtain required component from neighbor nodes, if they own. Thus, we have a plan to develop a mechanism which enables to obtain required components from not only a server but also other near neighbor nodes, if they own. Secondly, devices in ubiquitous computing environment have no inexhaustible storage resources. Thus, monitoring procedures may be rather burdens than helps to use resource efficiently sometime. That is, clients can not store components without any limitation. In consequence, it is essential to control the number of components which a client has, according to its storage capability. We think, therefore, that a client should maintain the number of components by using LRU scheme. In other words, older components are eliminated earlier.

Acknowledgement

We thank Kwang-Hoon Choi for implementing JAVA AWT based GUI part of AMONS.

Reference

[1] Mark Weiser, "Computer for the Twenty-First Century," *Scientific American*, pp. 94-110, Sep. 1991.

[2] C. K. Toh, "Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks", *IEEE Communication Magazine*, pp. 138-147, Jun. 2001.

[3] R. A. F. Mini, A. A. f. Loureiro, and B. Nath, "Prediction-Based Energy Map for Wireless Sensor Networks," *Proc. of the 8th IFIP PWC 2003*, pp. 12-26, Sep. 2003.

[4] J.-H. Ka and J.-H. Kim, "A Framework for Embedded Systems Management," *Proc. of the Int'l Conf. on Parallel and Distributed Processing Technology and Application*, pp. 510-516. Jun. 2002.

[5] D. Ngo, N. Hussain, M. Hassan and J. Wu, "A Resource Usage Monitoring Tool for Ad Hoc Wireless Networks", *Proc. of the 28th Annual IEEE International Conference on Local Computer Networks (WLN 2003)*, pp. 20-24, Oct. 2003.

[6] K. Takahashi, S. Amamiya, T. Iwao, "An Agent based Framework for Ubiquitous Systems", *Proc. of Challenges in Open Agent Systems '03 Workshop*, Melbourne, pp. 14-18, Jul. 2003.

[7] F. Bagci, J. Petzold, W. Trumler, and T. Ungerer, "Ubiquitous Mobile Agent System in a P2P-Network", *Proc. of System Support for*

Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003), Oct. pp. 12-15, 2003.

[8] D. Garlan, and B. Schmerl, "Component-Based Software Engineering in a Pervasive Computing Environment," *Proc. of the 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction*, May 2001.

[9] Y. Cui, D. Xu, and K. Nahrstedt, "SMART: A Scalable Middleware Solution for Ubiquitous Multimedia Service Delivery", *Proc. of the IEEE International Conference on Multimedia and Expo 2001 (ICME2001)*, Aug. 2001.

[10] /linux/Documentation/filesystems/proc.txt. Documentations/proc.txt,

[11] C. E. Perkins and E. M. Royer, "Ad hoc On-Demand Distance Vector Routing," *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90-100, Feb. 1999.

[12] IEEE Computer Society LAN MAN Standard Committee, "Wireless LAN Medium Access Control and Physical Specification," *IEEE 802.11b Standard*, 1999.

[13] AODV-UU @ Uppsala University, http://www.docs.uu.se/docs/research/projects/scanet/aodv/

[14] Pocket PC - Windows Mobile-based Pocket PC Home Page, http://www.microsoft.com/windowsmobile/products/pocketpc/default.mspx

[15] VRTX Real-Time Operating System, http://www.mentor.com/vrtxos/ , Mentor Graphics.

[16] Symbian OS – the mobile operating system, http://www.symbian.com/

[17] TinyOS – Component-based OS for the networked sensor regime, http://webs.cs.berkeley.edu/tos/

[18] Sun Microsystems, "The AWT in 1.0 and 1.1," http://java.sun.com/products/jdk/awt/