

Mobile-C Based Agent System for Detecting Improper Computer Usage at Computer Laboratories

Zhixin Tie
School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, P. R. China
Email: tiezx@zstu.edu.cn

***Abstract*—Mobile agent based computing is one of the powerful technologies for the development of distributed complex systems. There is little research regarding the effectiveness of mobile agent based detecting of improper computer usage at computer laboratories. This paper presents a Mobile-C Based Agent System (MCBAS) for Detecting Improper Computer Usage at Computer Laboratories. Based on the Mobile-C library, the MABAS supports the dynamic sending and executing of control command, dynamic data exchange, and dynamic deployment of mobile code in C/C++, and thus can detect improper computer usage conveniently and efficiently. The experiment was conducted at number of computer laboratories in a university computer center to detect improper usage of the computer workstations, such as playing computer games. The experiment shows that the mobile agent based monitoring system is an effective method for detecting and interacting with students playing computer games at public computer laboratories.**

***Index Terms*—mobile-C library, mobile agents, computer laboratories, improper computer usage detecting, key and mouse event sequence**

I. Introduction

University computer laboratories usually have hundreds of computers with many different hardware configurations. They are typically used by all university students for campus-wide courses, tests, web surfing, and other education related tasks. Because computers in these laboratories are widely used, there are a lot of different kinds of software installed on these computers and students can also install their favorite software on them. The issue, how to detect if someone is using the computer improperly or for non-educational purposes, such as playing computer games, is one of the most outstanding one that contribute to the highest amount of workload for the computer center staff.

The common way to find improper computer users is by means of manual inspections. Manual inspection of these public computer laboratories is time consuming. New technology is needed to solve this problem. The best way to solve this problem is by running some monitoring programs on the computers. Such monitoring programs should have the following characteristics:

(1) Can be deployed easily, quickly, and dynamically.
(2) Have a small footprint.
(3) Can regularly send result to the server or send result when asked for.

The most common way to solve this problem is to develop a system with Client/Server architecture [1] [1]. In this method, the server polls the client to gather the monitoring data. This kind of monitoring system has several disadvantages. For instance, polling clients will result in high server load, clients reporting the monitoring data will use more network bandwidth, and the monitoring programs running on the clients will need to be upgraded on a regular basis. There are several ways to avoid these disadvantages, such as using freeware tools from Sysinternals[1][2], which can remotely execute a program on the client node. However, it requires heightened access privileges, which tend to cause safety problems.

Another possible way to detect a computer game player is to monitor and analyze the network traffic on a gateway. The major advantage of this method is that the detecting program only needs to run on the gateway. However, it has some disadvantages. For instance, it cannot catch non-network computer games, and it must analyze many network packets in order to be effective. This will cause a high load on the gateway. It must also have the right to access to the gateway, and in some network topologies, it will be hard to implement to sniff the network traffic.

Based on the above conditions, mobile agent technology is ideal for the task. Mobile agent technology has shown great potential for solving problems in large scale distributed systems. It has been successfully used in a variety of distributed applications, such as manufacturing[3][4], computer vision [5], power electronic systems [6][7], data mining [8][9][10], e-commerce [11][12][13], network management [14][15], Intrusion Detection System[16][17][18], transportation systems [19][20][21], distributed sensor networks [22][23][24], information management [25][26], supply chain management[27][28], and structural health monitoring [29][30]. It can significantly enhance the design and analysis of systems whose problem domain is geographically distributed, and whose subsystems exist in

Manuscript received January 18, 2013; revised March 13, 2013; accepted April 6, 2013.

doi:10.4304/jsw.8.9.2262-2271

a dynamic environment and need to interact with each other more flexibly [31]. Its benefits are known, as Lange listed seven good reasons in [32].

To the best of our knowledge, there have been no mobile agent-based works on monitoring the computing usage of computers. In this paper, we present a Mobile-C Based Agent System (MCBAS), which is based on the Mobile-C library to detect improper computer usage at computer Laboratories. In MCBAS, an agency starts on each of computers during boot time. Monitoring agents are dynamically sent to a group of or all of computers from a monitoring server when monitoring tasks are needed. Monitoring results are taken from computers to monitoring server by the agent itself.

The rest of the paper is organized as follows. Section II introduces the monitoring environment. Section III presents a Mobile-C Based Agent System (MCBAS) based on the Mobile-C agent. Section IV demonstrates an application of the system. Finally, conclusions are drawn in Section V.

II. MOBILE-C BASED AGENT SYSTEM (MCBAS)

A. Mobile C library

The Mobile-C [33][34][35] was originally developed as a standalone, FIPA compliant mobile agent system. It uses an embeddable C/C++ interpreter, Ch [36][37][38], as the Agent Execution Engine (AEE) to support the interpretive execution of C/C++ mobile agent codes.

The major components of the Mobile-C library include Agency, Agent Management System (AMS), Agent Communication Channel (ACC), Agent Security Manager (ASM), Directory Facilitator (DF), the Agent Execution Engine (AEE), Agent, Synchronization, and Miscellaneous APIs. The Mobile-C library has extended most of the API functions from the host program space to the mobile agent space. The mobile agent space APIs allow a mobile agent to interact with an agency, different modules of an agency, and other agents. The right part of Fig. 1 shows how mobile agent code interfaces with the Mobile-C library. When the function mc_Function() is called in mobile agent code, Ch searches the corresponding interface function MC_Function_chdl() in the Mobile-C library, and passes arguments to it by calling the function. Subsequently, the interface function MC_Function_chdl() invokes the target function MC_Function(), and passes the return value back to the mobile agent space.

Since the AEE of the Mobile-C is based on Ch and Ch is capable of calling functions in binary static and dynamic libraries without recompilation [39][40][41], all existing binary static and dynamic C libraries and modules can be used as the Mobile-C agent code. For example, functions in OpenGL and XML libraries can be called from Ch directly [42][43], Ch communicates with the functions in binary libraries using a dynamically loaded library, as shown in the left part of Fig. 1. The example shows how mobile agent code invokes the functions in the C dynamic library called HookKeyMouse.dll, which is developed to hook key and mouse events by using Microsoft Visual Studio.net.

B. Architecture of MCBAS

The architecture of the MCBAS is shown in Fig. 2. Each computer in the computer center, known as a client node, runs a monitoring program that encompasses a Mobile-C agency. Multiple mobile agents can run in the agency at the same time. When the agency receives a mobile agent that is sent from a monitoring server, the agent is executed immediately. If the monitoring server needs some results, the result will be sent to the monitoring server automatically after the agent is executed. If the agent has a migration task, it will migrate to its next destination automatically after its task on the current client node is completed. When the agent is running on the client node, it can access all the client node resources via the method mentioned in Part A of this section.

There is a monitoring server program running on the monitoring server. The monitoring server program includes the monitoring knowledge management module (MKM), mobile agent module (MAM), and monitoring result management module (MRM).

The functions of MKM are to add, delete, modify, and retrieve the monitoring knowledge. For example, if a sequence of key and mouse events is the shortcut for a computer game "GA", we can add it to the monitoring knowledge database via this module. Then, this sequence can be sent to client nodes by a mobile agent to find if there is anyone who is playing the game "GA".

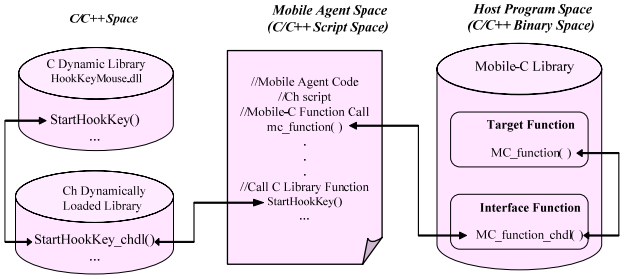

Figure 1. Interface of mobile agent code with the Mobile-C library and common C dynamic library

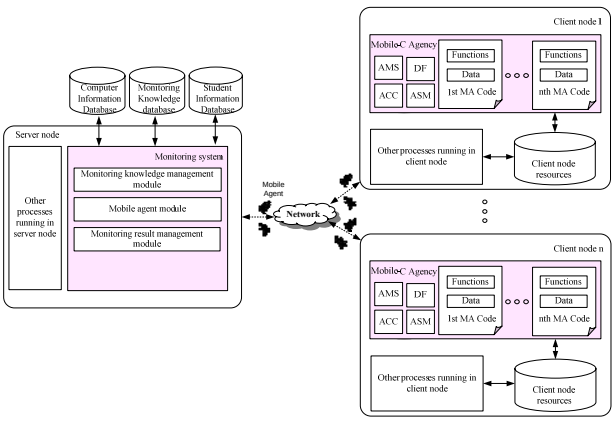

Figure 2. Mobile agent monitoring system architecture

The functions of MAM are to manage mobile agent code, send mobile agent to client nodes, and receive the monitoring result data by inspecting returning mobile agents and store data in the monitoring knowledge database. For example, if a new computer improper usage detecting algorithm has been developed, we can use this module to store the new algorithm code in the monitoring knowledge database. We can send it to client nodes by mobile agents for testing. Then, we can get the testing results from the return data of the mobile agent.

The MRM is used to analyze the monitoring data and generate reports. By accessing the computer information database, student information database, and monitoring knowledge database, the report of who played games on which computer at what time will be generated. More information regarding the game player may be obtained in real time. Using the same system, a report about the computing resource usage of the server will be automatically generated and sent to the designated recipients daily.

III. Application: Detecting Improper Usage of Computers at Computer Laboratories

Computer laboratories are accessible to all university students. Students may use computers to do anything. However, some activities are forbidden in computer laboratories during certain hours, like playing computer games. Thus, a monitoring program is desirable. It can monitor computers to prevent improper usage of computers. Our general strategy is as follows.

(1) Record the entire key and mouse event to the client node's main memory after the user logs on to that client node.

(2) Detect the game player. Scanning the record of key and mouse events of a client node, if shortcuts for games occurred many times in a given time period T, we can determine if the computer user is playing a game.

(3) Send a warning message to the game player to warn him not to play the game any more.

(4) For users that continue to play games, disrupt the game play by intercepting signals to the game application process.

A. Monitoring Environment Introduction

The network topology of the computer laboratories that we use for experiments is shown in Fig. 3. It has one server machine room and nine computer laboratories which are numbered from 1 to 8. For computer laboratories 1, 2, 3, and 7, each contains 100 computers. For computer laboratories 4, 5, 6, and 8, each has 80 computers. The computer center network is connected to the campus network through a firewall. Computers are connected to switches of computer laboratories through 100M super category 5 twisted pair line. Switches of computer laboratories are connected to the main switch through 1000M fiber cable. Computers in the same computer laboratory are in the same Virtual Local Area Network (VLAN), and computers in the different computer laboratories are in the different VLANs.

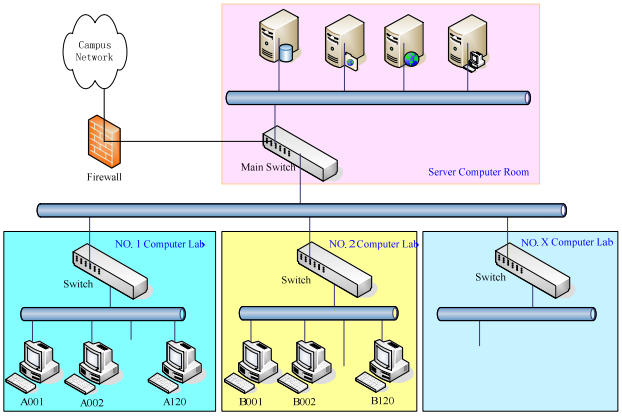

Figure 3. Network topology of the computer center.

All computer laboratories are available from 8:00am to 9:00pm, 7 days a week. There are laboratory hours for some lessons from 8:00am to 4:00pm on weekdays. During these laboratory hours, the computers should be used for teaching and learning. All non-academic related activities are prohibited in computer laboratories.

B. System Hook Implementation

A hook is a mechanism by which an application can intercept events, such as messages, mouse actions, and keystrokes. A function that intercepts a particular type of event is known as a hook procedure. A hook procedure can act on each event it receives, and then modify or discard the event [44]. Among 13 types of hooks that Microsoft Windows provides, only two of them, WH_MOUSE and WH_KEYBOARD, are chosen to implement in the hook dynamic library HookKeyMouse.dll. When one key or mouse event takes place, it will use the Windows API function PostMessage() to post the message to the monitoring program MAMonitor. MAMonitor is launched at system startup and placed into the system tray in the client node. The monitoring program records the received event using format as defined in Fig. 4 to the client node main memory. Key events are recorded using their virtual key codes [45]. Mouse events' virtual key codes are shown in TABLE I. For the event combinations, such as [Ctrl] + [C], [Shift] + [LBUTTONDOWN], which include modifiers, the modifiers are logically OR'd with the modifier's logical mask, as described in TABLE II. [46].

There are eight functions exported by the hook dynamic library HookKeyMouse.dll, as described in TABLE III. Mobile agents can call them to control the start or stop of event hooking and to configure what key and/or mouse event(s) to hook.

```
typedef struct {
    DWORD dwTicks;  //CPU TICKS
    HWND hwnd; //windows ID that the key or
            // the mouse event will be sending to
    WORD uCode; // Virtual key code of the key
            // or the mouse event
    WORD x; //Coordinate X, if it is a mouse event
    WORD y; // Coordinate Y, if it is a mouse event
} LOG_RECORD;
```

Figure 4. Key and mouse event record structure

C. Mobile Agents Design

The Mobile-C mobile agent is composed of mobile agent code in Ch, encapsulated in XML format. For detailed format, please refer to [28][29]. Each Mobile-C agent has several attributes, including the name, owner, home address, and tasks of the agent. Each agent task has attributes such as the ordinal number of the task, return variable, completeness, persistence, execution host of the task, and agent code of the task. The persistence attribute can be enabled to create an agent that will not be removed from the agency after the agent code is executed. This way, the variables and functions in the agent code can still be accessed later on. The Mobile-C agents can have sophisticated dynamically generated task lists and can move around a network autonomously.

In this application, multiple mobile agents are designed to fulfill the monitoring task. They are described as follows.

TABLE I.
VIRTUAL-KEY CODES OF MOUSE EVENTS

Mouse events	Virtual-Key Codes
WM_MOUSEMOVE	0x88
WM_LBUTTONDOWN	0x89
WM_LBUTTONUP	0x8a
WM_LBUTTONDBLCLK	0x8b
WM_RBUTTONDOWN	0x8c
WM_RBUTTONUP	0x8d
WM_RBUTTONDBLCLK	0x8e
WM_MBUTTONDOWN	0x8f
WM_MBUTTONUP	0x97
WM_MBUTTONDBLCLK	0x98
WM_MOUSEWHEEL	0x99
WM_XBUTTONDOWN	0x9a
WM_XBUTTONUP	0x9b
WM_XBUTTONDBLCLK	0x9c

TABLE II.
KEYBOARD MODIFIER MASKS

Modifier	Logical Mask
Ctrl	0x100
Alt	0x200
Shift	0x400

TABLE III.
HOOK DYNAMIC LIBRARY EXPORTED FUNCTIONS

Function name	Function's function
StartHookKey	Start key event hook
StopHookKey	Stop key event hook
AddKeyHookEntry	Add a key hook entry
DeleteKeyHookEntry	Delete a key hook entry
StartHookMouse	Start mouse event hook
StopHookMouse	Stop mouse event hook
AddMouseHookEntry	Add a mouse hook entry
DeleteMouseHookEntry	Delete a mouse hook entry

- *Monitoring Start Agent (MSA)*

The task of the monitoring start agent is to start a hook procedure on a client node. When the agency, which is embedded in the client node monitoring program MAMonitor, receives the monitoring start agent, the function StartHookKey() and StartHookMouse() in the hook dynamic library HookKeyMouse.dll are called to start key and mouse event hooking. Thus, a sequence of key and mouse events, denoted by S , will be stored into the client node's main memory.

- *Mining Sequential Patterns Agent (MSPA)*

The mining sequential patterns agent is designed to mine frequent sequential patterns in key and mouse event sequence S . Mining frequent sequential patterns in a large database has been studied by several researchers [47][48][49]. Many algorithms were proposed, such as AprioriAll, AprioriSome, and Dynamicsome in [47], and GSP in [48], and WINEPI and MINEPI in [49]. All these algorithms can be implemented to find the user frequently inputted key and mouse event sequence that is named interesting patterns (IPs). They can be sent to monitored clients by means of mobile agents. Using the mobile agent technology, these algorithms can be switched dynamically, and there is no need to reinstall the monitored clients' program.

We are only interested in IPs which are associated with computer gameplay. We define an interesting pattern used for playing games as a game shortcut sequence (GSS). Appropriate GSSes should not be long IPs so that we do not need to find long sequence patterns in key and mouse event sequence S . In this paper, the GSSes that only have no more than three key or mouse events are considered. Therefore, it appears that the algorithms that we mentioned above are a little complex. A simple algorithm Mining Sequential Patterns (MSP), as shown in Fig. 5, is presented to find the IPs in key and mouse event sequence S . In algorithm MSP, we use $S = \{S_1, S_2, \ldots, S_L\}$ denoting key and mouse event sequence that the monitoring program MAMonitor record. Each event $S_i (i = 1,2, \ldots, L)$ has the structure that is shown in Fig. 4. We call the number of events in an event sequence its size, and call an event sequence of size k a k-event sequence. Thus S is a L-event sequence. We use $I_k (k = 1,2,3)$ to denote the set of the IPs, each element of $I_k (k = 1,2,3)$ is a k-event sequence. We call the occurring times of a k-event sequence in S its support count. The notation I is used to denote the set of the IPs that are found in a client node. The notation $C_k (k = 1,2,3)$ is used to demote the candidate of the set of the IP. When the support count of an element of $C_k (k = 1,2,3)$ is no less than $MinSup$, which will be selected by the user for the different cases, then it is an element of $I_k (k = 1,2,3)$, otherwise, it does not.

```
for i=0 to L-1 do begin
      if (x=S(i).uCode ∈ I1) then
            x.sup ++;
      else
            x.sup =1; Add x to C1;
      endif
endfor
for any x∈ C1 do begin
      if (x.sup <MinSup) then
            Add x to I1;
      end if
endfor
for i=0 to L-1 do begin
       for any p,q,r∈ I1 do begin
            CountSup(S,i,p,q,r);
            CountSup(S,i,q,r,p);
            CountSup(S,i,r,p,q);
      Endfor
Endfor
for any x∈ C2 do begin
      if (x.sup >MinSup) then
            Add x to I2;
      end if
endfor
for any x∈ C3 do begin
      if (x.sup >=MinSup) then
            Add x to I3;
      end if
endfor
output ,I1, I2, I3;

Procedure CountSup(S,i,p,q,r)
     if (S(i).uCode <>p) then exit;
     if (i+1>=L) then exit;
     if (S(i+1).uCode==q) then
          if (x =(p,q) ∈ C2) then
            x.sup ++;
          Else
               x.sup =1; Add x to C2;
          Endif
          if (i+2>=L) then exit
          if (S(i+2).uCode==r) then
               if (x =(p,q,r) ∈ C3) then
                    x.sup ++;
               Else
                    x.sup =1; Add x to C3;
               Endif
          Endif
     elseif (S(i+1).uCode ==r) then
          if (x =(p,r) ∈ C2) then
               x.sup ++;
          Else
               x.sup =1; Add x to C2;
          Endif
          if (i+2>=L) then exit
          if (S(i+2).uCode ==q) then
               if (x =(p,q,r) ∈ C3) then
                  x.sup ++;
               Else
                  x.sup =1; Add x to C3;
               Endif
           Endif
     Endif
```

Figure 5. The algorithm MSP used in the MSPA to find the IPs.

The main idea of the algorithm MSP is to scan S, which is recorded in a client node, twice to find the IPs. It consists three steps. First, it scans S to determine I_1. Second, it scans S for the second time, when at the event $S_i(i = 1,2,\ldots,L)$, for any three element of I_1, say p, q, r, the algorithm MSP check all possible combination of any two of p, q, r and all possible combination of p, q, r to get $C_k(k = 2,3)$. Third, put any element $x \in C_k(k = 2,3)$, which support count is greater than or equal than $MinSup$ to $I_k(k = 2,3)$.

In the monitoring server system, we manually determine which IP is the GSS and save it in the monitoring knowledge database. Algorithms that can automatically find the GSSes from the IPs need to be considered in the future.

- *Game Player Finding Agent (GPFA)*

The task of the game player finding agent is to send the GSSes to client nodes to monitor whether someone is playing games or not. The Game Player Find (GPF) algorithm, which is used to find game player on client nodes, is shown in Fig. 6.

In algorithm GPF, the sequence $S = \{S_1, S_2, \ldots, S_L\}$ is the key and mouse event sequence that the monitoring program MAMonitor record which is also used in the algorithm MSP shown in Fig. 5. We use $G = \{G_1, G_2, \ldots, G_M\}$ to denote the set of the GSSes. Each of the M elements of G demotes a GSS, its structure is shown in Fig. 7. We call the times that a GSS occurs in S the support count.

The main idea of the algorithm GPF is to scan S, which is recorded in a client node, to find the support count of each GSS. When the support count of a GSS is greater than the given frequency threshold $mconf$, and the support count of this GSS in S in last T_S minutes greater than one, then the one who are using this client node is suspected to play games.

There may be many programs running in a client node at the same time. Each program is analyzed separately using the Algorithm shown in Fig. 5. When the game player is detected, the program's windows ID will be recorded for later use. In this application, a ring memory buffer, whose length is L that can be defined by the system administrator, is employed to store the key and mouse event sequence S. The recorded data in memory over the length of L will be replaced by the new data. Furthermore, upon a successful detection of a game player, a screenshot of the client node will be taken and saved in a temporary file. Finally, the GPFA will report its findings back to its originating server.

- *Collecting Information Agent (CIA)*

The collecting information agent is designed to perform following tasks:

(1) To find client nodes that did not enter the monitoring state

(2) To collect the frequent key and mouse event sequences that the MSPA found in client nodes.

- *Get Screenshot Agent (GSA)*

The get screenshot agent will be sent to the client node when the return value of the GPFA for that node is true.

The GSA will get back the temporary file that the GPFA saved on that node to the monitoring sever. Thus, as an option, the administrator can use this screenshot to verify whether the user on that node is playing games or not.

- *Warning Agent (WA)*

The warning agent is used to send a warning message to the one who is detected playing computer games. The warning message box will be the topmost window on the client node so that the game player can see the warning message.

- *Game Play Disruption Agent (GPDA)*

The game play disruption agent is used if a user ignores the warning messages generated by the warning agent. At this time, a GPDA is send to the client node to inform the monitoring program to drop some key or mouse event for the window on which the IP's were detected. For example, if we find someone playing computer games on a client node with a GSS of "ABC", we use the hook dynamic library HookKeyMouse.dll to drop the keystroke "C" on the computer game window. However, other programs will not be affected because the key or mouse event is only dropped in the game program window that is recorded by the GPFA.

```
Let F = false;
for all GSS g in G do begin
    g.sup = g.sup1=0;
end for
Let T= Current CPU ticks;
for i=L-1 to 0 do begin
    for all GSS g in G do begin
        for j=0 to g.nLen - 1
            if (i+j>=L) then exit for
            if (g.nCode[j] != S(i+j)) then exit for
            if (j == g.nLen-1) then begin
                g.sup ++;
                if (T - S(i).dwTicks <= T_S) then
                    g.sup1 ++;
                endif
            endif
            if (g.sup>=mconf && g.sup1>=1) then begin
                F = true;
                exit for;
            endif
        end for
        if (F == true) then exit for;
    end for
    if (F == true) then exit for;
end for
```

Figure 6. The algorithm GPF used in the GPFA to detect game players.

```
typedef struct {
    WORD nCode[3]; // Virtual key code of the key
                   // or the mouse event
    WORD nLen;   // The number of key or mouse event
                 // in the GSS
    WORD nSup;     //the GSS's support count in the whole
      //key and mouse event sequence S
    WORD nSup1; // the GSS's support count in S
                // in last T_S minutes
} GSS_t;
```

Figure 7. The GSS structure.

- *Monitoring End Agent (MEA)*

The function of the monitoring end agent is stopping a hook procedure on a client node.

D. Mobile agents' migration process

Among these eight kinds of agent, the MSA and MSPA are persistent, and others are non-persistent. Thus the MSA and MSPA do not migrate while the others migrate from one node to the next. Fig. 8 shows mobile agents migration process from the monitoring server to a node client.

(1) The monitoring server sends the MSA to the client node. The key and mouse hooking procedure will be started.

(2) In order to get the client node monitored status, the monitoring server sends the CIA to the client node to get monitored status.

(3) If the client node is in monitored status, then go on to the next step. Otherwise, the monitoring server may resend the MSA.

(4) The monitoring server sends the MSPA to the client node. The MSPA uses the WINEPI algorithm to find IPs.

(5) The monitoring server sends the CIA to get IPs that the client node found.

(6) The monitoring server sends the GPFA to find that there is someone playing games on the node or not. If a game player was found, the GPFA takes a screenshot of that node and saves it to a temporary file. The agent saves the result to its persistent status and migrates to the next target node, or home if there are no remaining target nodes.

(7) If the return value of the GPFA indicates a user may be playing a game, the monitoring server sends the GSA to retrieve the screenshot file. The administrator can use this file to determine whether the user of that client node is playing games or not.

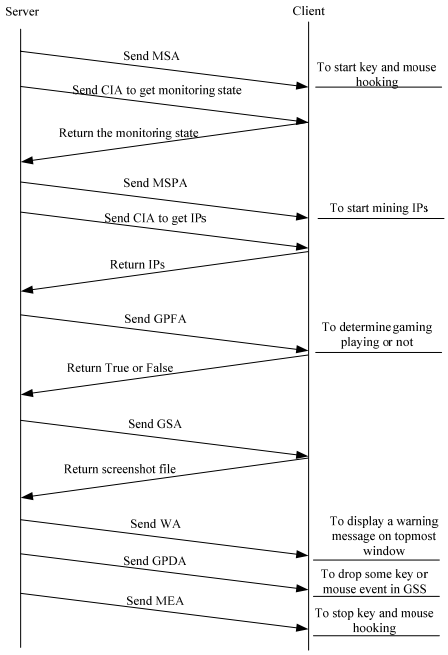

Figure 8. Agents migration process

(8) If the user of the client node is determined to be playing a game, the WA will be sent to the node. If the user does not stop playing games, the administrator can directly ask him/her to stop playing it or enter next step automatically.

(9) The monitoring server sends the GPDA to the client node to try to stop the user playing games.

(10) If monitoring is no longer needed, the monitoring server may send the MEA to the client node to stop the monitor.

Described above is a general process. In some special cases, this may not be the appropriate process. The GPFA can be sent several times for different GSS's. Each new GPFA agent will replace the old one automatically. The CIA can be sent to client nodes anytime that we need to get the information. Client nodes may not need to be monitored in some times, for example, during weekends. Thus, the MEA can be sent to client nodes to stop monitoring.

Although a sequence of key and mouse event S is stored in main memory of the client node, the application preserves students' privacy as well. In this system, we consider the student's passwords to be private data. Therefore, in order to preserve a student's privacy, it must be considered impossible for even a system administrator to retrieve a student's plaintext passwords.

In this system, the system only scans S to find interesting patterns (IPs), and transfers the IPs to the monitoring server. No other key patterns are recorded or transferred. During processing, all the data is stored in S, and the IPs is stored in binary format and never appears as plaintext on the screen. No one can access this data using normal methods. Low level attacks such as buffer overflow attacks are beyond the scope of this paper and the operating system is assumed to have secure virtual memory. Because the IPs are calculated from the user's frequently inputted key and mouse events, the private data is not normally contained in the IPs.

Also because a fixed length buffer in the monitored clients' main memory is used to store the key and mouse events, the oldest event records will be overwritten by the new ones. Furthermore, the buffer will disappear when the user logs off or the computer is restarted or powered off.

F. Experiment on Two Computers

Before testing the MCBAS in the computer laboratories, a small experiment has been carried out on two computers. One computer serves as the monitoring server, the other serves as the monitored client node. The experiment follows the steps described in part E of this section. The tester used the monitored client node to play the game "3D Pinball". Only a few keys, including keys 'Z', '/', 'X', '.', are used in this game. When the CIA is sent to get the IPs after five minutes, part of the IPs that were obtained from the monitored client node are shown in TABLE IV. We defined the third IP as the GSS and send the GPFA to the client node. The GPFA returns true, which means we found a potential game player. Then, the GSA is sent to the client node to retrieve the screenshot file which includes the interface of the game "3D Pinball". Next, the WA is sent to the client node. The tester using the client node continued playing the game "3D Pinball" until the GPDA arrived. Afterwards, the tester found that the keys 'Z' and '/' fail to respond within the "3D Pinball" game. Finally, to test the effectiveness of the MEA, the MEA is sent to the client node, the monitor is stopped. The tester was able to smoothly play the game "3D Pinball" again.

G. Experiment in the Computer Center

A large scale real environment experiment was carried out in a university computer center part A of this section to validate the performance of the MCBAS. The monitoring server program is installed and runs on one of PC server machines. Its interface is shown in Fig. 9. The client node program MAMonitor is installed on each computer in computer laboratories.

In the experiment, there were computer programming classes in computer laboratories 1 to 8, more than ninety percent of computers in these computer laboratories were used by students. Twenty testing volunteers were assigned to computer laboratories 1 to 4, five volunteers for each computer laboratory. The game Warcraft III, which is popular among students, was selected to perform the experiment. Each volunteer played the game Warcraft III on assigned computer.

The experiment steps are described in part E of this section. It was divided into two phases. Each phase lasted for one hour. The first five steps were performed in phase I in order to collect the IPs. In phase II, all steps were performed.

In the experiment, the $MinSup$ is selected to 30. Some of the IPs that were detected in phase I were shown in TABLE V. Ten of them, which is shown in TABLE V with value "Yes" in the third column, are to defined as the GSSes to find the game Warcraft III player in phase II.

In the phase II, the parameter in algorithm GPF, which is shown in Fig. 6, is selected as follows:

TABLE IV.
THE IPS FOUND IN TEST

NO	The IP	Corresponding key	GSS or not
1	0x5a	'Z'	No
2	0xbf	'/'	No
3	0x5a, 0xbf	'Z', '/'	Yes
4	0xbf ,0x5a	'/', 'Z'	No

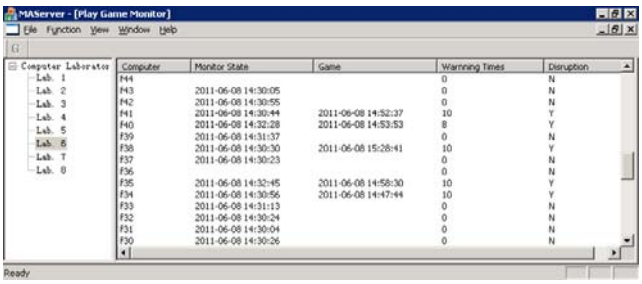

Figure 9. The interface of the monitoring server program.

$$mconf = 10$$
$$L = 3600$$
$$T = 2Min$$

The MCBAS successfully detected all twenty volunteers playing the game, as well as five additional students who were not among the volunteers. Because we do not know when the students started to play the game and how often did the students use the GSSes that we defined in TABLE V, the playing game detected time of each student is recorded. The playing game detected time of a game player is defined as the time elapses between the beginning of the Phase II and the time when he is detected. The playing game detected time of these 25 students is shown in Fig. 10. The minimum detected time is 8 minutes 22 seconds and the maximum detected time is 58 minutes 11 seconds.

In this experiment, all used computers can start or stop the hook procedure when they receive the MSA or MEA. The agent MSPA and CIA worked very well, thus the IPs were got, as shown in TABLE V. When the GSSes are determined, the agent GPFA is sent to all used computers with these GSSes. Then 25 game players were detected at different time, as shown in Fig. 10. After game players were detected in a client node, the system automatically sent the agent WA to that client node every one minute. All 25 detected game playing client nodes received the agent WA several times, and each WA popped up a warning message window, which is shown in Fig. 11, to try to persuade the user not to play game any more. All 20 testing volunteers were instructed to continue playing the game. The other five students stopped playing the game on their own. After the second warning message was displayed on their windows, all 20 testing volunteers noticed that the mouse that they were using for the game window does not respond suddenly. This is because the system sent the agent GPDA, which drops the last key or mouse event when the user input key or mouse sequence matching one of the GSSes, to the game playing client node.

The client node program MAMonitor has a small footprint. Because except the 25 game playing students, no one found that a test was carrying out on his computer, and computer programming lab classes went normally and smoothly.

TABLE V.
THE IPS FOUND IN THE EXPERIMENT.

NO.	The IPs	GSS or not
1	F1	No
2	F2	No
3	F3	No
4	F6	Yes
5	#	No
6	F1, A	No
7	F2, A	No
8	F3, A	No
9	F1, C	No
10	#, A	No
11	F1, RBUTTONDOWN	Yes
12	F2, RBUTTONDOWN	Yes
13	F3, RBUTTONDOWN	Yes
14	#, RBUTTONDOWN	Yes
15	F1, A, LBUTTONDOWN	Yes
16	F2, A, LBUTTONDOWN	Yes
17	F2, A, LBUTTONDOWN	Yes
18	F1, C, LBUTTONDOWN	Yes
19	#, A, LBUTTONDOWN	Yes
20	LBUTTONDOWN	No
21	RBUTTONDOWN	No
22	LBUTTONDOWN, RBUTTONDOWN	No
23	RBUTTONDOWN, LBUTTONDOWN	No
24	LBUTTONDOWN, RBUTTONDOWN, LBUTTONDOWN	No
25	RBUTTONDOWN, LBUTTONDOWN, RBUTTONDOWN	No

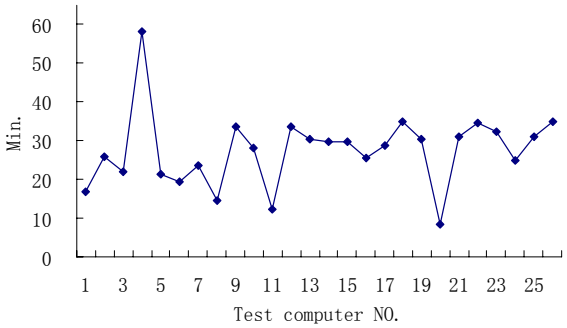

Figure 10. The game player detected time.

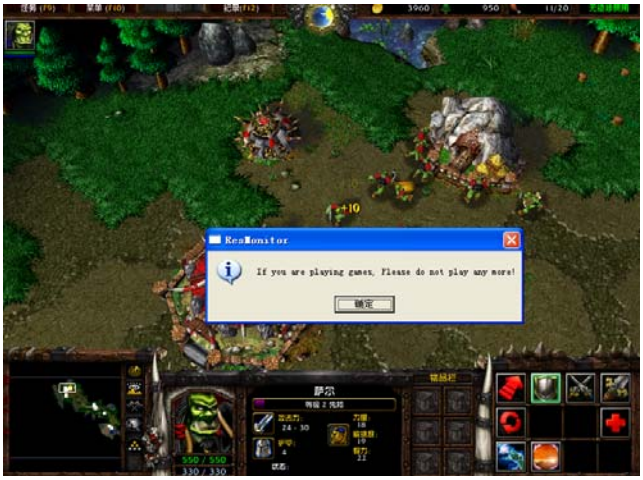

Figure 11. The warning message box.

V. CONCLUSIONS

A mobile agent-based system called the Mobile-C Based Agent System (MCBAS) for monitoring of computer usage at public computer laboratories was

presented. An IEEE FIPA compliant mobile agent system called the Mobile-C is used as the base for the MCBAS. The monitoring server can send control command, exchange data, and deploy data mining algorithms to any group of or all of monitored client nodes easily, quickly, dynamically, and silently. An experiment in a university computer center with hundreds of computer workstations has been conducted to validate this system. The experimental results show that it is an effective way for dynamic software component deployment in public computer laboratories. It is also an effective way for detecting and stopping improper usage of computers in computer laboratory classes'. Future works include designing more improper usage detection algorithms and applying this system to detect computer failure.

ACKNOWLEDGMENTS

This work is supported by the Qianjiang Talent Project of Zhejiang Province under Grant No. 2012R10056 and the Zhejiang Provincial Natural Science Foundation of China.

REFERENCES

[1] Patrício Domingues, Luís Silva and João Gabriel Silva. "DRMonitor – A Distributed Resource Monitoring System", *Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)*, Genova, Italy, Feb 2003, pp. 127-133.

[2] Patrício Domingues, Paulo Marques and Luís Silva. "Distributed Data Collection through Remote Probing in Windows Environments", *Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)*, Lugano, Switzerland, Feb 2005, pp. 59-65.

[3] Mark Russinovich and Bryce Cogswell "Windows Sysinternals". http://technet.microsoft.com/en-us/sysinternals.

[4] Antonella Di Stefano and Corrado Santoro, "A3M: an agent architecture for automated manufacturing", *Softw. Pract. Exper*, Vol. 39, No. 2, Feb. 2009, pp. 137–162.

[5] Stephen S. Nestinger, Bo Chen, Harry H. Cheng, "A Mobile Agent-Based Framework for Flexible Automation Systems", *IEEE/ASME Transactions on Mechatronics*, Vol. 15, No. 6, Dec 2010, pp 942-951.

[6] Stephen S. Nestinger and Harry H. Cheng, "Flexibile Vision: Mobile Agent Approach to Distributed Vision Sensor Fusion", *IEEE Robotics and Automation Magazine*, Vol. 17, No. 3, Sept. 2010, pp. 66-77.

[7] S. D. J. McArthur, M. Davidson, V. M. Catterson, A. L. Dimeas, N. D. Hatziargyriou, F. Ponci, and T. Funabashi, "Multi-agent systems for power engineering applications—Part I: Concepts, approaches, and technical challenges", *IEEE Trans. Power Syst.*, vol. 22, no. 4, Nov. 2007, pp. 1743–1752.

[8] Ferdinanda Ponci and Aalhad A. Deshmukh "A Mobile Agent for Measurements in Distributed Power Electronic Systems", *IEEE Transactions on instrumentation and measurement*, Vol. 58, No. 5, May 2009, pp. 1657-1669.

[9] Matthias Klusch, Stefano Lodi, and Gianluca Moro, "Agent-Based Distributed Data Mining-The KDEC Scheme", *Lecture Notes in Computer Science*, Vol. 2586/2003, 2003, pp. 104-122.

[10] S. Krishnaswamy, A. Zaslavsky1, S.W. Loke, "An Architecture to Support Distributed Data Mining Services in E-Commerce Environments", *In Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems*, Milpitas, California, USA, June 08 - 09, pp. 239–246.

[11] Khalil Amiri, David Petrou, Gregory R. Ganger, Garth A. Gibson, "Dynamic function placement for data-intensive cluster computing", *Proceeding ATEC '00 Proceedings of the annual conference on USENIX Annual Technical Conference.* USENIX Association Berkeley, CA, USA, June 2000, pp. 25-25.

[12] P. Maes, R. H. Guttman, and A. G. Moukas. "Agents that buy and sell", *Communications of the ACM*, Vol. 42, No. 3, Mar. 1999, pp. 81–91.

[13] M. Yokoo, S. Fujita, "Trends of internet auctions and agent-mediated web commerce", *New Generation Computing*, Vol. 19, No. 4, 2001, pp. 369–388.

[14] T. Sandholm, "eMediator: a next generation electronic commerce server", *Computational Intelligence*, Vol. 18, No. 4, 2002, pp. 656–676.

[15] Bieszczad, A., Pagurek, B. and White, T., "Mobile agents for network management", *IEEE Communications Surveys*, Vol. 1 No. 1, 1998, pp. 1-9.

[16] Jinho Ahn, "Fault-tolerant Mobile Agent-based Monitoring Mechanism for Highly Dynamic Distributed Networks", *International Journal of Computer Science Issues*, Vol. 7, No 3, May 2010, pp. 1-7.

[17] O Kachirski, R Guha, "Intrusion detection using mobile agents in wireless ad hoc networks", *Proceedings. IEEE Workshop on Knowledge Media Networking*, 2002, Kyoto, Japan, July 2002, pp. 127-133.

[18] Akyazi, Ugur, Uyar, A. Sima, "Distributed detection of ddos attacks during the intermediate phase through mobile agents", *Computing and Informatics*, Vol. 31, No. 4, 2012, pp. 759-778.

[19] Wenjuan Wang, Weihui Dai, Weidong Zhao, et.al, "Research on Mobile Agent System for Agile Supply Chain Management", *Journal of Computers*, Vol. 6, No. 8, 2011, pp. 1498-1505.

[20] J.L. Adler, V.J. Blue, "A cooperative multi-agent transportation management and route guidance system", *Transportation Research Part C: Emerging Technologies*, Vol. 10, No. 5–6, Oct. – Dec. 2002 pp. 433–454.

[21] Bo Chen, Harry H, Cheng, Joe Palen, "Integrating Mobile Agent Technology with Multi-Agent Systems for Distributed Traffic Detection and Management Systems", *Transportation Research Part C: Emerging Technologies*, Feb. 2009, Vol. 17, No. 1, pp. 1-10.

[22] Bo Chen and Harry H. Cheng, "A Review of the Applications of Agent Technology in Traffic and Transportation Systems", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 11, No. 2, June 2010, pp. 485-497.

[23] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu, "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications", *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, Columbus, Ohio, USA, June 2005, pp. 6-10.

[24] Qishi Wu, Nageswara S.V. Rao, Jacob Barhen, et al, On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks, *IEEE Transactions on Knowledge and Data Engineering*, VOL. 16, NO. 6, JUNE 2004.

[25] Nan Zhang, Jianhua Zhang, "A Self-adapted Anycast Routing Algorithm Based on Mobile Agent in Wireless

Sensor Network", *Journal of Networks*, Vol. 6, No. 2, 2011, pp. 206-213.
[26] K. Stathis, O. DeBruijn, S. Macedo, "Living memory: agent-based information management for connected local communities", *Interacting with Computers*, Vol. 14, No. 6, Dec. 2002, pp. 663–688.
[27] H. Tu, J. Hsiung, "An architecture and category knowledge for intelligent information retrieval agents", *Decision Support Systems*, Vol. 28, No. 3, May 2000, pp. 255–268.
[28] Zhisong Hou, Zhou Yu, Wei Zheng, and Xiangang Zuo, "Research on Distributed Intrusion Detection System Based on Mobile Agent", Journal of Computers, Vol. 7, No. 8, 2012, pp. 1919-1926.
[29] Weidong Zhao, Haifeng Wu, Weihui Dai, et.al, "Multi-agent Middleware for the Integration of Mobile Supply Chain", *Journal of Computers*, Vol. 6, No. 7, 2011, pp. 1469-1476.
[30] Bo Chen,Wenjia Liu, "Mobile Agent Computing Paradigm for Building a Flexible Structural Health Monitoring Sensor Network", *Computer-Aided Civil and Infrastructure Engineering*, Vol. 25, No. 7, Oct. 2010, pp. 504–516.
[31] Stuart G Taylor, Kevin M Farinholt, Eric B Flynn et al, "A mobile-agent-based wireless sensing network for structural monitoring applications", *Measurement Science and Technology*, Vol. 20, No. 4, Apr. 2009, pp. 1-14.
[32] N. R. Jennings, "An agent-based approach for building complex software systems - why agent-oriented approaches are well suited for developing complex, distributed systems", *Communications of the ACM*, Vol. 44, No. 4, April 2001, pp. 35-41.
[33] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Commun. ACM*, Vol. 42, No. 3, Mar. 1999, pp. 88–89.
[34] Mobile-C: A Multi-Agent Platform for Mobile C/C++ Code. (2005). [Online]. Available: http://www.mobilec.org.
[35] Bo Chen, Harry H. Cheng, Joe Palen, "Mobile-C: A Mobile Agent Platform for Mobile C/C++ Code", *Software Practice & Experience*, Vol. 36, No. 15, 2006, pp. 1711-1733.
[36] Yu-Cheng Chou, David Ko, Harry H. Cheng, "An embeddable mobile agent platform supporting runtime code mobility, interaction and coordination of mobile agents and host systems", *Information and Software Technology*, Vol. 52, No. 2, Feb. 2010, pp. 185–196.
[37] H.H. Cheng, "Scientific computing in the Ch programming language", *Scientific Programming*, Vol. 2, No. 3, 1993, pp. 49–75.
[38] H.H. Cheng, "Ch: A C/C++ interpreter for script computing", *C/C++ User's Journal*, Vol. 24, No. 1, 2006, pp. 6–12.
[39] H.H. Cheng, "Ch — an Embeddable C/C++ Interpreter", [Online]. Available: http://www.softintegration.com.
[40] Softintegration Inc, Ch – an embeddable C/C++ interpreter, Available: htth://www.softintergration.com/.
[41] Softintegration, "The Ch Language Environment – SDK User's Guide", Softintegration, Inc, Available: http//www.softintegration.com.
[42] Embedded Ch User's Guide, Softintegration, Inc. [Online]. Available: http://www.softintegration.com/products/sdk/embedded ch/
[43] Bo Chen, Harry H. Cheng, "Interpretive OpenGL for Computer Graphics," *Computers & Graphics,* Vol. 29, No. 3, June 2005, pp. 331-339.
[44] Z. Wang, H.H Cheng, "Portable C/C++ Code for Portable XML Data", *IEEE Software*, Vol. 23, No. 1, 2006, pp. 76 – 81.
[45] Microsoft Corporation, Hooks Overview, Available: http://msdn.microsoft.com/en-us/library/ms644959(v=VS.85).aspx.
[46] Microsoft Corporation, Virtual-Key Codes, Available: http://msdn.microsoft.com/en-us/library/dd375731(v=vs.85).aspx.
[47] Jason Alexander, Andy Cockburn, and Richard Lobb, "AppMonitor: A tool for recording user actions in unmodified Windows applications", *Behavior Research Methods*, Vol. 40, No. 2, 2008, pp. 413-421.
[48] R. Agrawal and R. Srikant. "Mining Sequential Patterns", *Proc. of the 11th Int'l Conference on Data Engineering*, Taipei, Taiwan, March 1995, pp. 3-14.
[49] R. Srikant and R. Agrawal. "Mining sequential patterns: generalizations and performance improvements", *Lecture Notes in Computer Science*, Vol. 1057/1996, 1996, pp. 1-17.
[50] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences", *Data Mining and Knowledge Discovery*, Vol. 1, No. 3, 1997, pp. 259-289.

Zhixin Tie received his Ph.D. degree in computer science from Zhejiang University in 2000. He is an Associate Professor in the School of Information Science and Technology at Zhejiang Sci-Tech University (ZSTU). His research interests include mobile agent systems, embedded systems, data mining, and power automation systems.