International Journal of Engineering Research and Technology.
ISSN 0974-3154 Volume 11, Number 7 (2018), pp. 1163-1177

http://www.irphouse.com

Mobility Support for MLE Collaborative Design System

Ki Chang Kim

Dept. of Information and Communication Engineering, Inha University, Korea.

Sang Bong Yoo

Dept. of Computer Engineering, Inha University, Korea.

Abstract

MLE (Multiple Level Encryption) collaborative design system builds a design object tree and encrypts each node with different keys. It provides the keys of a set of nodes only to the designer who has the right access permission to these nodes. Therefore with this system the designers can collaborate safely with other designers without exposing his or her design secret. In this paper, we propose to enhance MLE collaborative design system with mobility support. Mobility support means to enhance the MLE system such that the designers now can move his or her location freely while participating in the collaborative design process without being disconnected even if the IP address keeps changing. Providing mobility to internet-connected devices has been researched by numerous researchers and a number of techniques have been proposed: e.g. mobile IP, SIP (Session Initiation Protocol), multi-homed IP, etc. These techniques, however, require extensive construction of middleware to provide smooth handover of IP sessions and are not deployed in general public domain yet. In this paper, we propose a simple but effective technique that can provide mobility to the MLE collaborative design system. Our technique inserts agents in the clients and modifies the server instead of overloading the system with heavy middleware system. These agents monitor the changes in IP addresses and act properly to handle the handoff process without further intervention from the designers.

Keywords: Agent, Collaborative, Design, MLE, Mobility

I. INTRODUCTION

As design complexity increases, it has become commonplace that many designers collaborate together in a single design project. The collaboration of multiple designers is often possible only through the Internet in a distributed fashion when they are

geographically separated[3]. In order to guarantee a successful collaboration in this environment, it is essential that we provide a mechanism through which the designers can cooperate smoothly while being protected on their intellectual rights[1], [5], [17], [20]. MLE (Multiple Level Encryption) file format [8] has been proposed as one of the solutions. Multiple designers can obtain a copy of the same MLE file and work simultaneously.

An MLE file hides the design secrets of the designers by encrypting the design objects with different keys and providing the keys of the design object only to those who has the right access permission. The entire design is represented as a design object tree where a node represents a design object and the link the relationship between the design objects. The participating designers will be registered in this file as regular users and the keys of design objects, stored at the end of the file in encrypted format, will be disclosed only to those who have the proper access rights. For this purpose, the keys are encrypted with the public key of the owner user of the corresponding design objects. The owner of design object A can download an MLE file, extract the encryption key for object A by decrypting it with his or her own private key, and then decrypt the design object. Other portion of the design object tree will be displayed as a black box to protect the intellectual rights of the corresponding owners.

MLE file format allows a safe design collaboration environment. It is safe because designers can cooperate without worrying about the exposure of their design secrets. Individual designers can concentrate only on the relevant design part while other design parts are hidden as black boxes. In this paper we are proposing to add mobility in this MLE file system. Mobility will add great flexibility in collaborated design. Designers can make quick simple fixes while he or she is in transit. A designer may also walk around in the factory while observing and fixing his design comparing it with the real product. The fundamental problem in supporting mobility is to maintain the Internet connection with the server while moving around. The designer needs to keep communicating with the MLE server to download the design file or to upload the modified one.

Mobility support for web application has been studied by numerous researchers[2], [6], [9], [11], [15]. Three solutions are representative of them: Mobile IP, SIP (Session Initiation Protocol), and SCTP (Stream Control Transmission Protocol). However all these three techniques require extensive construction of middleware system in order to handle handoff of IP sessions. The middleware system constantly monitors the changes in IP address and modifies the mapping table (in Mobile IP) or registers the new IP to the proxy server (in SIP) or replaces old IP with new one (in SCTP) when IP address changing is detected. The required middleware system is huge and costly, and is not readily available to general public.

This paper proposes a technique that detects the changes in IP address and handles IP handover smoothly but without the help of middleware system. Our technique gives up the application-level transparency in both server and client side, which was the main reason behind the expensive middleware structure. Instead we insert a simple but effective agents to the clients and modify server application. The agents will monitor the changes in IP address and send a signal to the main code when IP address changing

is detected. The main code contains a function that upon receiving the signal handles the handover process. We explain the algorithm and show a scenario which steps through the handover process. We also discuss the computational overheads with the experiment results.

The rest of the paper is organized as follows. Section 2 surveys related researches, Section 3 briefly introduce the MLE file format and a use case, Section 4 explain the main algorithms of supporting mobility for MLE collaborative design system, Section 5 shows a scenario in which the handover process is explained and discusses the performance overheads for mobility support, and finally Section 6 gives a conclusion.

II. RELATED WORK

Supporting mobility in wireless network has been one of the hot research topics, and various solutions have been proposed. Especially mobility in distributed collaborative teamwork such as design project or video conferencing has been emphasized as one of the key requirements for successful collaboration [3]. Mobile IP [12] supports mobility by providing two separate IPs to each mobile node: Home Address and Care of Address. Home Address is the original IP given to the device, and Care of Address is the current IP allocated to it. Care of Address keeps changing as the mobile node moves around, but Home Address is a fixed one. Mobile IP system provides a mapping service between these two IPs. The server always sends packets to Home Address of the target mobile node. Home Agent, a part of Mobile IP middleware system located at the Home Address, receives them, computes the current Care of Address of the mobile node using the mapping table, and forwards them to Foreign Agent in the target network where the current mobile node is located. Foreign Agent will finally hand the packets over to the target mobile node. With the support of Mobile IP system, the mobile node can freely move around across several LANs, and the IP handover process is application transparent meaning that the application does not have to know about the changes in IP address. However, as mentioned before, the Mobile IP middleware such as Home Agent or Foreign Agent is quite large and complex, and for that reason is not yet readily available to general public.

SIP [4], [20] requires that each user has a SIP-specific URI (Uniform Resource Identifier), and that the user application, called User Agent, knows SIP protocol such as setting up call or registering the current location, etc. Then the user with a SIP phone that implements SIP protocol can communicate with other SIP users wherever the others are located. Thus SIP supports IP mobility, however similarly as in Mobile IP, SIP requires a set of large and complex intermediate servers such as Proxy Servers, Registrar, and Redirect Servers to process SIP commands.

SCTP [20] uses Multihoming technique to support IP mobility. At initialization phase, SCTP-aware clients and servers exchange a set of IP addresses through which they can communicate. For example the client connects to the server using following function:

```
sctp_connect(int sd, struct sockaddr *addrs, int addrcnt, …);
```

In above "addrs" contains the set of IP addresses and "addrcnt" contains the number of available IP addresses. When one of the IP address is disconnected, the server can keep talking with the client through another IP address. This scheme again requires extensive modification in the underlying system and is not generally available yet.

Inouye [7] suggests a scheme which is probably the closest to our technique. The client and the server are both aware and prepared to handle IP mobility. Once IP disconnection is detected the client runs code for reconnection and the server responds appropriately for smooth IP handoff. However, this scheme is originally targeted to multimedia data streaming such as video conferencing, and it needs extensive support from the underlying operating system. The operating system should detect changes in IP address and inform interested processes. It also should support a set of system calls that the client and the server can use to handle IP handoff.

III. MLE COLLABORATIVE DESIGN SYSTEM

III.I MLE File Format

The overall structure of an MLE file is shown in Figure 1 [8]. The figure shows the location of the three major tables: Layer Node Table(LNTAB) at extent 3 to 12, User Node Table(UNTAB) at extent 14 to 23, and Encrypted Session Key Node Table(ESKNTAB) at extent 25 to 34. Layer Node Table stores Layer Nodes (LNODEs) which contain the location of the design objects, called layers, and corresponding session keys that will be used to encrypt the design object data. User Node Table stores User Nodes (UNODEs) which contain the information about the registered users who have access rights for this MLE file. ESKNTAB stores Encrypted Session Key Nodes (ESKNODEs) which contain encrypted session keys and the LNODE indices of the target LNODEs.

For each table, the system needs a bitmap that shows which entry is occupied and which is not. The bitmap for LNTAB, LNBM, is located at extent 2, the bitmap for UNTAB, UNBM, is at extent 13, and the bitmap for ESKNTAB, ESKNBM, is at extent 24. There is another bitmap at extent 1, EBM, which is a bitmap to handle the allocation of extents. Finally extent 0 is the super block that contains the map for the entire MLE file format. When an MLE file is created, the file header is initialized with the above explained meta-data, and two additional extents are allocated for the root layer. In Figure 1, extent 35 is allocated for the root data extent, and extent 36 for the root directory extent. For each layer, the system allocates at least two extents: one for data, the other for directory. Data extent contains the data for that layer. Directory extent contains the name and LNODE number of the sub-layers for the current layer.

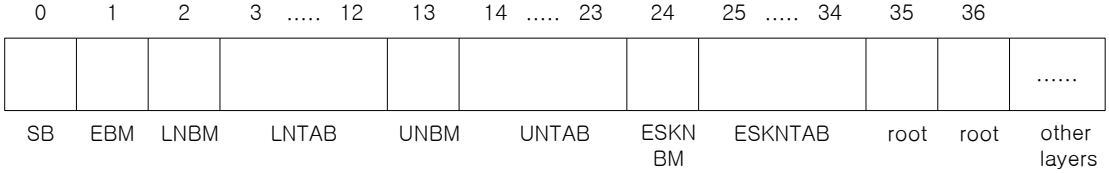
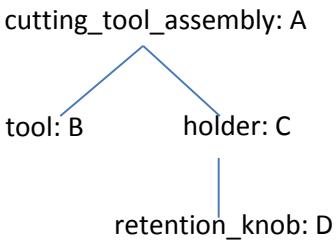
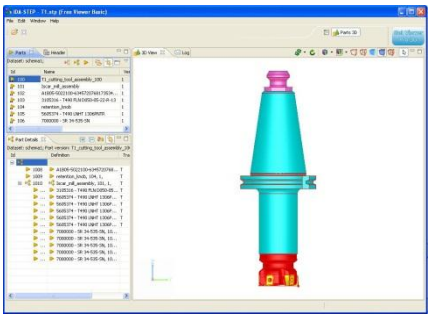


Figure 1. Overall structure of an MLE file.

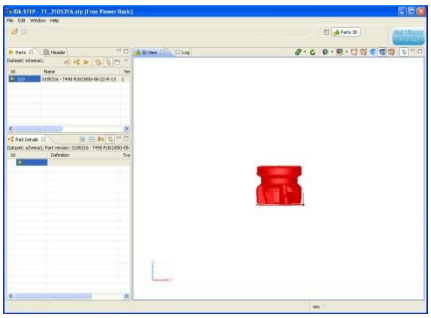
As an example, suppose that a cutting tool assembly is to be designed by company A. As shown in Figure 2, the cutting tool assembly is composed of two parts, i.e., a tool and a holder. The holder has a retention knob as its subpart. We suppose that the tool and the holder are designed by company B and C, respectively. Company C also has company D to design the retention knob. In this situation, the design data of the tool should not be accessed by company C and D. The design data of the holder also protected from company B and D. MLE file enables this kind of hierarchical structure of access rights. Using an MLE file, company A can access all the design data of the cutting tool assemble, company B can access only those of the tool, company C can access those of the holder and the retention knob, and company D can access only those of the retention knob. Without using MLE file, four different files should be created for this collaborative design process, one for each company.



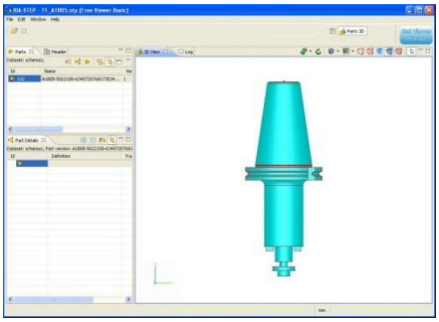
(a) Object Tree



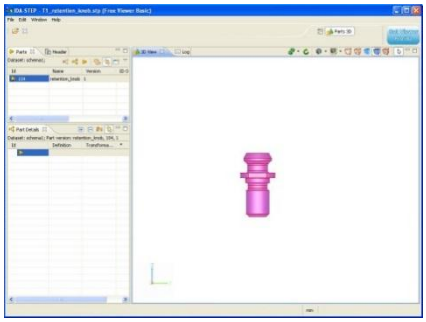
(b) Cutting Tool Assembly



(c) Tool



(d) Holder



(e) Retention Knob

Figure 2. Object tree and component parts of a cutting tool assembly.

III.II MLE Client and MLE Server

MLE collaborative design system consists of one MLE server and multiple MLE clients. MLE server and clients communicate through SSL (Secure Socket Layer) connection for secure communication. MLE server runs the "MLE Server Process" algorithm given below. After opening an SSL port, the server runs infinite loop responding to client packets. Since the server has to take care of multiple clients, it uses "select" mechanism which allows waiting on multiple clients simultaneously. If packets arrive from clients, there are three cases to handle them. One is a connection request packet, the SYN packet from a new client. In this case, the server will simply call "accept" to establish a new connection. Another is SSL clientHello packet. This is the first SSL packet from the client, and the server should start the SSL protocol to establish an SSL connection. The last one is client command packet coming through the SSL channel. This should consist of the majority of packets after the SSL channel is established. In this case, the server should decrypt the command and initiate appropriate action for each command such as opening a design file, read a portion of it, or writing into it, etc. After the command is processed, the server will send the result back to the client.

Algorithm MLE Server Process:

```
open an SSL port;
REPEAT
    wait for client packets on select();
    IF SSL connection request packet
        call accept() to establish a connection;
    ELSE IF client HELLO
        call SSL_accept() to process SSL handshake protocol;
    ELSE IF an MLE client command
        call SSL_read() to receive MLE client command
        process an MLE client command;
        call SSL_write() to send the result
```

MLE client runs "MLE Client Process" algorithm given below, which is much simpler than the server. It opens an SSL port and connects it to the MLE server, and then establish an SSL channel on this port. After that the client runs infinite loop receiving user commands and sending them to the server and displaying the results received from the server.

Algorithm MLE Client Process:

open an SSL port;
connect it to the MLE server;
call SSL_connect() to establish an SSL channel with the server;
REPEAT
read user request;
build MLE command;
call SSL_write() to send the MLE command to the server;
call SSL_read() to receive server response

IV. APPLICATION-LEVEL MOBILITY SUPPORT

IV.I MLE client with mobility support

To support mobility, MLE client will duplicate itself via *fork* system call and generate an agent that will monitor and control the parent code. MLE client generates a child process (the client side agent) which will monitor the parent for IP changing. When the parent acquires a new IP address, it sends a signal to the parent so that the parent can start the prepared signal handler. The parent's signal handler will reconnect to the MLE server with the new IP and replace the socket number with this new connection. After signal handling, the parent will come back to the original code location where the signal has interrupted and resume the task as if nothing had happened. Since the IP handoff is processed by signal handling, the parent code doesn't have to be modified in its core part -- the MLE file processing portion. We just add a signal handler in the parent code. The modified MLE client is as follows.

Algorithm MLE Client Process with Mobile Support:

signal(SIGUSR1, reconn); // reconn is the signal handler
x = fork(); // duplicate itself
IF (x==0) // child process will become the client agent
monitor_IP(); // the client agent keeps monitoring IP change
ELSE // parent runs the core part – MLE file processing
cliSocket=open an SSL port;
connect it to the MLE server;
call SSL_connect() to establish an SSL channel with the server;
call SSL_read() to receive "session_number" for this client
REPEAT
read user request;
build MLE command;
call SSL_write() to send the MLE command to the server;
call SSL_read() to receive server response;

When the *fork()* process successes, the PID of the child process is returned in the parent, and 0 is returned in the child. The newly generated process becomes client agent which *executes monitor_IP()*. This function will keep watching IP changes and send a signal to the parent as follows.

Algorithm Client Agent Monitoring IP:

```
REPEAT
    ip_changed = poll_IP_change();  // monitor IP changes
    if (ip_changed)
        kill(getppid(), SIGUSR1); // send signal SIGUSR1 to the parent
```

Upon receiving this signal, the parent stops whatever it was doing and jumps to *reconn()*, which is the signal handler and its algorithm is as follows.

Algorithm Reconnect New IP:

```
close_current_connection(cliSocket);
cliSocket = reconnect_to_server();
```

cliSocket is the socket variable the parent is using for communicating with the server. When the communication is disconnected due to IP changing, *reconn()* will reconnect to the server with the new IP address and attach this connection to *cliSocket*. This function also sends the original session number of this client with MLEFS_RECONNECT command so that the server properly handles IP handoff. After signal handling, the parent will come back to the original interrupted code (interrupted by a signal). The parent will most likely be interrupted in the infinite loop which reads user requests and sends them to the server. The parent might have read some packets from the server through the old socket before the IP changing and was interrupted before sending a command. After signal handling the parent sends an MLE command via *cliSocket* which now is attached to the new IP address.

IV.II MLE client with mobility support

The MLE server side also needs to be modified properly to support IP mobility. The MLE client will send MELFS_RECONNECT message as the first packet when it reconnects to the server due to IP changing. MLEFS_RECONNECT message should contain the old session number and the server uses it to determine which client needs IP handoff. The MLE server will be modified as follows.

Algorithm MLE Server Process for client packet with Mobile Support:

```
open an SSL port
REPEAT
    wait for client packets on select();
    IF SSL connection request packet
        call accept() to establish a connection;
    ELSE IF client HELLO
        call SSL_accept() to process SSL handshake protocol;
```

assign a unique session number to this client;
call SSL_write() to send the session number to the client
ELSE IF an MLE client command
call SSL_read() to read MLE client command
process an MLE client command;
call SSL_write() to send the result

If the IP address of a client has been changed, the client command should be MLEFS_RECONNECT. The procedure of processing MLEFS_RECONNECT command can be described as the following algorithm.

Algorithm Process MLE client command:
IF MLEFS_RECONNECT
extract old session number of this client from the client packet
find old context information of this client at the old session
transfer old context to new session
remove old session
ELSE
call corresponding MLE API function

In order to process MLEFS_RECONNECT message, the server now has two sockets for this client, one for the old session and the other for the new session. The outgoing packets from the old session should have been failed due to invalid destination IP address. Upon receiving MELFS_RECONNECT message from the client, the server extracts the old session number from this message, finds the context information of the old session and transfers it to the new session. The failed packets then will be resent from this new session.

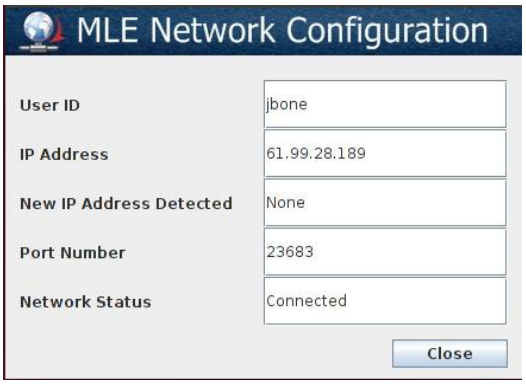
V. WORKING SCENARIO AND PERFORMANCE EVALUATION

As a working example, suppose that a client is running on a notebook and connected to the MLE server from an office where a fixed IP address is provided. Now the notebook is taken out of the office and tethered to a mobile phone in order to be connected to the MLE server. In this case the notebook is allocated a new IP address provide be the tethering service of a mobile phone. Tethering over Wi-Fi is also known as Personal Hotspot, which is available on iOS 4.2.5 (or later), Windows Mobile 6.5 devices, and Android 2.2 (or later) [19].

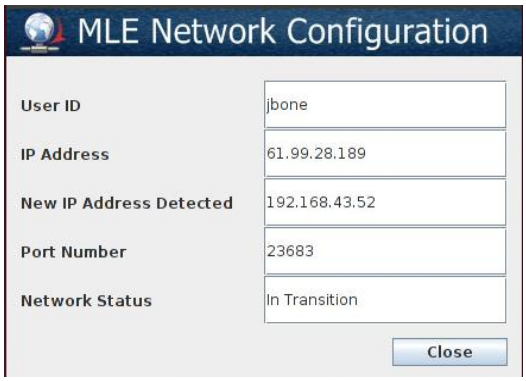
In this example, the client is connected to the Internet via IP address number 61.99.28.189 and port number 23683 as shown in Figure 3(a). While the client

communicate with the MLE server, an agent process is forked and keep watching if the IP address number is changed. As soon as the notebook is taken out of the office and tethered to a mobile phone, the agent process detects the change of IP address and signals SIGUSR1 to its parent process (see Figure 3(b)). Then the client process jumps to reconnecting process which creates a new socket with the new IP number 192.168.43.52 and port number 48259. Now the new IP number and port number is informed to the server.

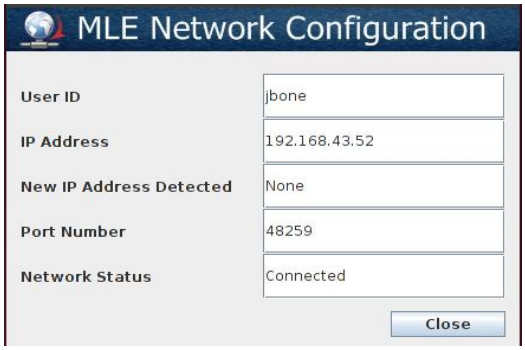
Once the notice of IP address change is received, the server initiate an agent process to handle the change. Firstly, the server side agent creates a new socket with the new IP address 192.168.43.52 and port number 48259. Secondly, it check out any messages to the client is pending in the message queue. If there is any pending message, redirect it to the new socket with the new IP address and port number. Finally the server and client resume normal communication as in Figure 3(c). The sequence of messages of this example is depicted in Figure 4.



(a) Client is connected to MLE server



(b) New IP Address is detected



(c) Client is connected via the new IP Address

Figure 3. Changes of Network Configuration for a MLE Client

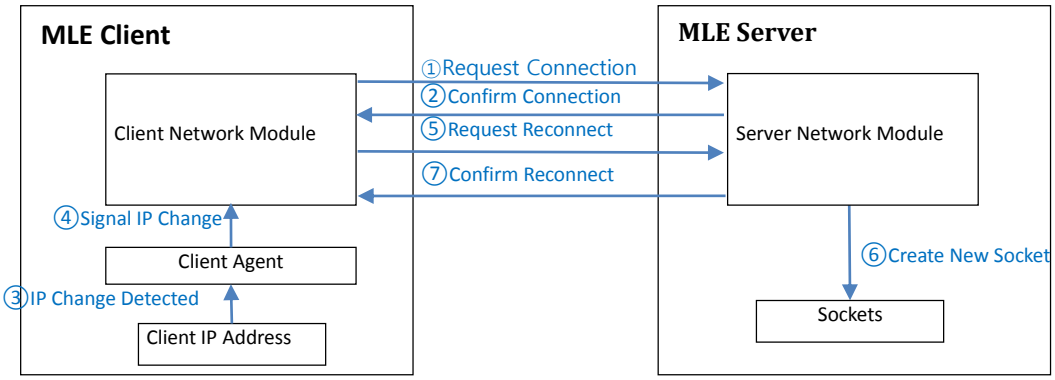


Figure 4. MLE Server and Client with Agents for Mobility Support

In MLE client-server environment, the response time of a client command can be defined as follows.

$$Response\ Time = Network\ Time_{cs} + Network\ Time_{sc} + Computing\ Time_{server}$$

The response time consists of two network times and the computing time of server. *Network Time*$_{cs}$ represents the time to take the command travels from the client to the server and *Network Time*$_{sc}$ represents the time to take the command travels from the server to the client. These network times depend on network configurations and traffic situation on particular time. In this section, we focus on the computing time of MLE server (i.e., *Computing Time*$_{server}$).

Because the computations involved in IP changes are not heavy, the processing of single change of client IPs takes a few nanoseconds and client IPs changes smoothly. In order to validate the mechanism in working environment, we perform experiments with multiple clients. In this experiment a client communicate the server with a series of simple commands. It sends the following *ping* messages to the server.

ping(0), ping(1), ping(2), …, ping(n)

As responses, the server respond the client by the following pong messages.

pong(0), pong(1), pong(2), …, pong(n)

In these experiments, a client send 1000 *ping* messages with increasing identifying arguments for 1000 seconds. The first experiment performed with various number of clients with no IP changes. As shown in the second column of Table 1, the processing times less than 1001 seconds up to 200 clients. It means that responding *pong* messages does not load much overhead on the server. However, the processing times increase when clients change IPs. As we increase the number of IP changes, the processing time also increase as well (see the third and the forth columns of Table 1). From Figure 5, we can characterize the increasing ratios of processing times are linear. The experiments have been run on a server with 4 Intel i5-2320 (3GHz) processors and 8 GB memories.

Table 1. Time to process multiple clients with various IP change ratios

Number of Clients	***Processing Time$_{server}$***		
	No IP changes	Changes IP every 10 seconds	Changes IP every 5 seconds
50	1000.4	1001	1009
60	1000.6	1001	1024
70	1000.4	1006	1035
80	1000.5	1010	1054
90	1000.6	1011	1062
100	1000.6	1012	1073
110	1000.6	1014	1087
120	1000.6	1016	1099
130	1000.6	1018	1108
140	1000.4	1020	1114
150	1000.5	1021	1132
160	1000.5	1025	1151
170	1000.5	1029	1162
180	1000.6	1032	1179
190	1000.6	1034	1183
200	1000.5	1038	1187

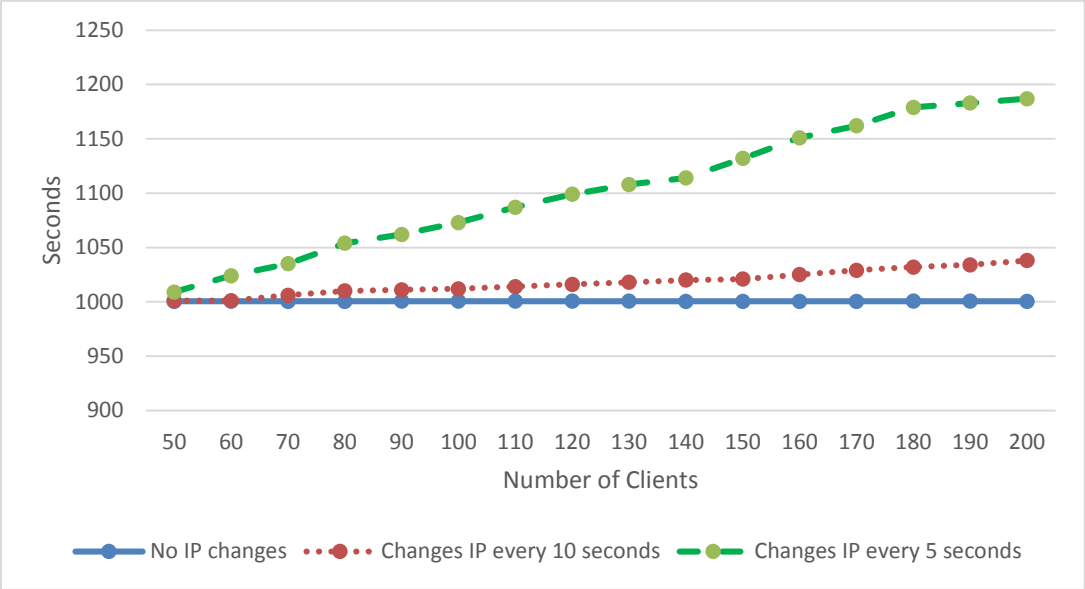


Figure 5. Time to process multiple clients with various IP change ratios

VI. CONCLUSION

One of the recent computing trends is the proliferation of mobile devices such as notebooks, tablets, and smartphones. A number of companies even allow employees to bring their own devices to work, due to perceived productivity gains and cost savings. As a result, collaborative solutions encounter challenges to support mobility as well as data protection. In paper, we extend the MLE (Multiple Level Encryption) collaborative design system so that designers can move his or her location freely without being disconnected even if the IP address keeps changing. Different from middleware approaches, we have insert agents into MLE clients in order to detect IP changes and enable MLE servers to handle the changes efficiently. Experimental studies show that the computational overheads increase linearly with respect to the number of clients and the number of IP changes.

With hierarchical data encryption mechanism and mobility support, MLE collaborative design system is quite suitable for various distributed and cloud computing environments. The proposed technique proposed in this paper can be effectively applied for mobility of other collaborative tools because it works independently from any middleware systems. Recently, the computing environments are rapidly expanded into IOT (Internet of Things) paradigms, where the complexity of collaboration should be increased as we have more participants. Future work of MLE system is to adapt it to IOT framework.

ACKNOWLEDGEMENTS

This work was supported by INHA UNIVERSITY Research Grant.

REFERENCES

[1] R. Ausanka-Cures, Methods for Access Control: Advances and Limitations, http://www.cs.hmc.edu/~mike/public_html/courses/security/s06/projects/ryan.pdf

[2] Christian Bauer, Secure and Efficient IP Mobility Support for Aeronautical Communications, KIT Scientific Publishing, Jan., 2013.

[3] Bellotti V. and Bly S. Walking away from the desktop computer: distributed collaboration and mobility in a product design team, Proceedings of CSCW'96, 209-218, 1996.

[4] Handley M., Schulzrinne H., Schooler E. and Rosenberg J. SIP: Session initiation protocol, IETF RFC 2543, 1999.

[5] S. Hauck, S. Knol, Data security for Web-based CAD, Proceedings of the 35th annual Design Automation Conference, pp. 788-793, 1998.

[6] ICAO Aeronautical Communications Panel, WGI. Analysis of Candidate Mobility Solutions, 13th meeting of the working group N-SWG1, Montrel, Canada, June 2007.

[7] Inouye J., Cen S., Pu C. and Walpole J. System support for mobile multimedia applications, Proceedings of NOSSDAV'97, 143-154, 1997.

[8] Ki Chang Kim and Sang Bong Yoo, Collaborative design by sharing multiple-level encryption files, CONCURRENT ENGINEERING-RESEARCH AND APPLICATIONS Vol. 22, No. 1, March 2014.

[9] Deguang Le ; Xiaoming Fu ; Dieter Hogrefe, A review of mobility support paradigms for the Internet, IEEE Communications Survey & Tutorials, 8(1):38-51, 2006.

[10] LKSoftWare GmbH, www.lksoft.com

[11] Eranga Perera, Vijay Sivaraman, and Aruna Seneviratne, Survey on network mobility support, SIGMDBILE, Mobile Computing and Communicatons Review, 8(2):7-19, 2004.

[12] X.Pérez-Costa and H.Hartenstein. A Simulation Study on the Performance of Mobile IPv6 in a WLAN-Based Cellular Network, Elsevier Computer Networks Journal, special issue on The New Internet Architecture, September 2002.

[13] SCRA, STEP Application Handbook ISO 10303, North Charleston, SC, 30 June 2006, available at http://www.uspro.org/documents/STEP_application_hdbk_63006_BF.pdf

[14] STEP Tools, Inc., http://www.steptools.com/demos/

[15] Sun S., Han L., Han S., Secure IP Mobility Support in Software Defined Networks In: Kim K., Wattanapongsakorn N. (eds) Mobile and Wireless Technology 2015. Lecture Notes in Electrical Engineering, vol 310. Springer, Berlin, Heidelberg, 2015.

[16] Tuexen, Michael; Randall R. Stewart, UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication, IETF. RFC 6951, May 2013.

[17] Shumiao Wang, Siddharth Bhandari, Siva Chaitanya Chaduvula, Mikhail J. Atallah, Jitesh H. Panchal andKarthik Ramani, Secure Collaboration in Engineering Systems Design, J. Comput. Inf. Sci. Eng 17(4), Jun 15, 2017

[18] Wedlund E. and Schulzrinne H. Mobility support using SIP, Proceedings of the second ACM/IEEE International Conference on Wireless and Mobile Multimedia, 1999.

[19] Wikipedia, Tethering, http://en.wikipedia.org/wiki/Tethering

[20] Y. Zeng, L. Wang, X. Deng, X. Cao, N. Khundker, Secure collaboration in global design and supply chain environment: Problem analysis and literature review, Computers in Industry, vo. 63, Issue 6, pp. 545-556, August 2012.