MobileSocket: Enhanced Socket Library for Application Layer Continuous Operations

Tadashi OKOSHI[1] Yoshito TOBE[2] Hideyuki TOKUDA[1,3]
[1]Graduate School of Media and Governance, Keio University
[2]Keio Research Institute at SFC, Keio University
[3]Faculty of Environmental Information, Keio University
5322 Endo, Fujisawa, Kanagawa 252, Japan
slash@ht.sfc.keio.ac.jp, tobe@mkg.sfc.keio.ac.jp, hxt@ht.sfc.keio.ac.jp

We propose "MobileSocket", a new simple user-level enhanced socket library written in pure-Java. MobileSocket provides Java applications with application layer **mobility** and **connection continuity** in order to support their **continuous operation**. Existing Java applications can attain socket-level mobility and connection continuity without any modification to their source code merely by replacing Java standard socket class with our MobileSocket. Two mechanisms inside the library, Dynamic Socket Switching (DSS) and Application Layer Window (ALW) provide application layer mobility and connection continuity without any kernel-level modifications or additional proxy software. In this paper, we describe design and implementation of MobileSocket and show evaluated results on our systems with FreeBSD.

1 Introduction

In mobile computing environment, several kinds of computing entities, such as users, hosts, applications, or even users' desktops can "rove" around. By carrying their own computer, users rove around between their home, office, and wherever they want to use it. In such a situation, provision of the continuous working environment for the users is required, although many of the surrounding environment elements of the host, such as network address or power consumption level, can change dynamically.

To provide users with "work continuity" achieved by the continuous operation framework is the key, since main purpose of mobile computing is "computing at anywhere".

In this paper, we present the "MobileSocket" library, which is the user-level, pure Java[1]-based, enhanced Socket interface library. In order to realize the "continuous operation" for the users, the MobileSocket library provides both "mobility" and socket-level "connection continuity" for any Java applications which use `java.net.Socket` class as a means of their Inter Process Communication (IPC).

Internally switching the actual socket connection used for data transmission, the MobileSocket library realizes the socket layer mobility, even when a mobile host is once disconnected from a network and moves to another network obtaining a different IP address. Also, MobileSocket provides its byte stream consistency essentially provided TCP protocol semantics by exploiting the application layer sliding window inside its library which retain the byte data consistency after the mobile host's relocation.

MobileSocket realizes both mobility and connection continuity for Java applications with only user-level implementation, while other approaches such as Mobile-IP[2], MSOCKS[3], Mobile TCP Socket[4] and TCP-R[5] require either kernel modifications or additional agents. In spite of simplicity, MobileSocket completely realizes the application layer mobility and connection continuity.

In the remainder of this paper, we address our definitions of "mobility", "connection continuity", and "continuous operation" in Section 2. Section 3 describes MobileSocket design and Section 4 describes MobileSocket mechanism. Then Section 5 presents the performance evaluation of our implementation, and Section 6 discusses some related works and functional comparisons. Finally we mention our future work and conclude this paper in Section 7.

2 Continuous Operation

In this section, we define the notions of "mobility", "connection continuity", and "continuous operation", followed by descriptions of our two redirection schemes, "Explicit Redirection" and "Implicit Redirection".

2.1 Mobility

With "mobility", a mobile host can maintain the transparent host identifier in network protcol architecture, even after the host is disconnected from a network and reconnected to a different network.

With any framework which supports mobility, the mobile host can be identified transparently from other hosts in wide area network at a certain layer of the network structure.

2.2 Connection Continuity

With "continuity", applications in the mobile host can retain their activities and can offer their own services to users, even after the host has moved to a different network (or even after the applications has moved to another host in a different network, in the future).

Specifically with respect to communications among applications, "connection continuity" means that the information for applications' connections is definitely retained in spite of moving of the host or the applications themselves.

For instance, TCP-R provides the TCP layer connection continuity in addition to mobility. With TCP-R, TCP connections between applications can be maintained and they are able to communicate continuously even if the mobile host has relocated.

2.3 Continuous Operation

We define "continuous operation" as a service that application can offer to users when it obtains both "mobility" and "connection continuity". With either "mobility" or "connection continuity" applications in the mobile and the correspondent host can both identify each other and maintain their connections.

Provision of "mobility" is enough for "continuous operation" in the cases of using connection-less and non-reliable communication. Mobile-IP provides not only "mobility" but "continuity" for applications like video conferencing, which usually uses UDP/IP in their communication. But in contrast, "mobility" does not always imply "continuity" for applications with reliable circuit communication, such TCP/IP. For such communications, "continuity" together with "mobility" is required for "continuous operation."

There are two issues: connection timeout and byte stream consistency. MSOCK provides transport layer "mobility", but it does not provide any interface for setting timeout variable or ones for explicit connection suspending and resuming. It is because MSOCKS mainly focuses on the "temporary disconnection" from the network, such as during the roaming in wireless LAN. It cannot adapt to long period disconnection, such as ones longer than TCP retransmission timeout. The other issue, byte stream consistency, is critical for reliable protocol connection. Protocol characteristic of stream consistency must be maintained even after the mobile host is relocated.

The provision of "mobility" and "continuity" are obviously necessary for the accomplishment of "continuous operation".

2.4 Implicit and Explicit Redirections

Mainly to realize the connection continuity, there are two kinds of schemes to achieve the connection redirection. The one is "Implicit Redirection" and the other is "Explicit Redirection".

Having the mobility and connection continuity support with Implicit Redirection, additional lines of source code for declaring the connection redirection, such as "suspend" or "resume" are not necessary in applications. With TCP-R, for example, all of ordinally TCP applications can acquire the connection continuity without any modification in their source code. This scheme seems very effective and useful particularly at the wireless network environment where temporal disconnections and relocations happen very often. But also very often, adjustment of redirection timeout like the retransmit timer in TCP protocol is the problem. According to the timeout value, it can be unuseful, and also causes waste of resources.

On the other hand, exploiting the "Explicit Redirection" scheme, applications at the both ends of connection can make sure whether the connection is suspended or it encounters temporal network congestion.

3 MobileSocket Design

In this section, we present the design of MobileSocket in order to accomplish our goal, "Application Layer Continuous Operation". To achieve this, we adopt the following four characteristics to MobileSocket.

3.1 Mobility and connection continuity support in application layer

MobileSocket achieves the socket-level mobility and connection continuity with an enhanced socket interface which is independent for the actual socket used for data communication. Dynamic Socket Switching (DSS) mechanism and Application Layer Window (ALW) described in next section support MobileSocket's mobility and connection continuity.

3.2 Simple user-level library implementation

MobileSocket is implemented as a user-level library. In some other related works for mobility and connection continuity, they need modification in kernel of the mobile host or the correspondent host, Home Agent and Foreign Agent, or the proxy server. Our application layer mobility approach needs only user-level library and does not need any modification in kernel or any additional software. Using Java language, this approach for mobility and connection continuity can be proved on many Java-compatible platforms.

3.3 Support of both implicit and explicit redirection operations

MobileSocket offers both implicit and explicit operations of connection redirection to applications. None of existing application needs to be modified and be added optional explicit operation APIs to their source code, in order to adopt MobileSocket mobility and connection continuity. On the other hand, applications can also use explicit redirect operation APIs which are effective for longer period disconnection of mobile hosts or cases that application tends to suspend its connection explicitly.

3.4 Adaptation in Applications

MobileSocket has the Java-event-based adaptation interface for the applications. When once the mobile host disconnect from network, how can the correspondent host be noticed it? It is obviously necessary to provide applications with certain interface and up-call mechanism for mobility and connection continuity.

4 MobileSocket Mechanism

In this section, we present the mechanism of MobileSocket. We describe MobileSocket state diagram and Dynamic Socket Switching (DSS) time sequence, containing the detail of each DSS phase during the disconnection and reconnection.

4.1 Dynamic Socket Switching (DSS)

Dynamic Socket Switching (DSS) mechanism inside the MobileSocket library enables the application layer mobility and connection continuity. Figure 1 shows the concept of DSS.

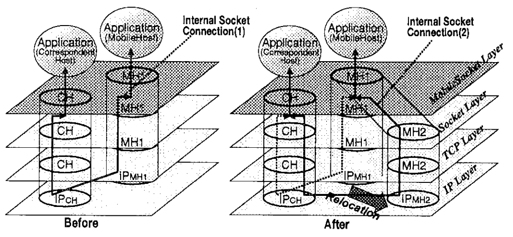

Figure 1: Concept of DSS

Once a MobileSocket connection is established between the mobile host and the correspondent host, applications in both side of the socket can read and write byte stream each other with one lasting socket object, even after the mobile host's relocation. In contrast, inside the MobileSocket library, the new socket connections as the actual connection between both applications is made every after the mobile host's relocation, and "switch"ed dynamically to provide connection continuity for the applications.

4.2 Application Layer Window (ALW)

Application Layer Window (ALW) is a user-level sliding window implemented in the MobileSocket library and maintains the stream consistency of the MobileSocket connection. Figure 2 shows the overview of ALW. After the mobile host's reconnection with the implicit redirection operation, the user data already written by application can remain in the lost socket connection between two MobileSocket libraries, in the buffers of the local protocol stacks, in the network, and in the buffers of protocol stacks in the remote host. This causes byte stream inconsistency problem for the MobileSocket connection. ALW guarantees the byte stream consistency of the MobileSocket by resending the lost data after the reconnection.

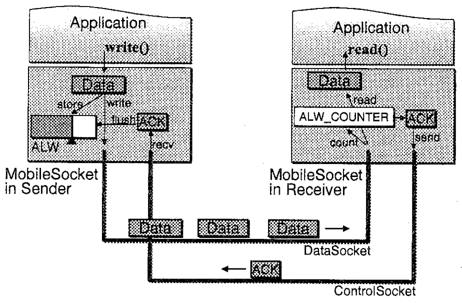

Figure 2: Application Layer Window

4.3 MobileSocket State Transition

Figure 3 shows the state transition of MobileSocket. There are mainly four states in MobileSocket, "Closed ", "Established", "ImplicitlySuspended", and "ExplicitlySuspended".

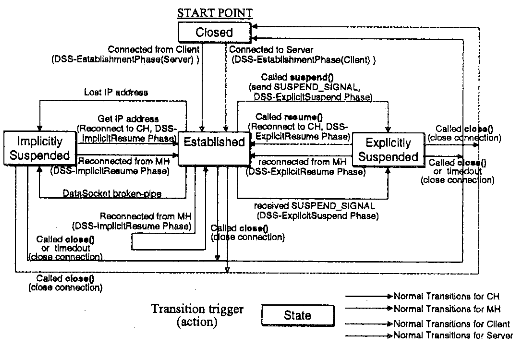
START POINT
Closed
Connected from Client (DSS-EstablishmentPhase(Server))
Connected to Server (DSS-EstablishmentPhase(Client))
Called suspend() (send SUSPEND_SIGNAL, DSS-ExplicitSuspend Phase)
Lost IP address
Get IP address (Reconnect to CH, DSS-ImplicitResume Phase)
Called resume() (Reconnect to CH, DSS-ExplicitResume Phase)
Implicitly Suspended
Established
Explicitly Suspended
Called close() (close connection)
Reconnected from MH (DSS-ImplicitResume Phase)
reconnected from MH (DSS-ExplicitResume Phase)
Called close() or timedout (close connection)
DataSocket broken-pipe
received SUSPEND_SIGNAL (DSS-ExplicitSuspend Phase)
Reconnected from MH (DSS-ImplicitResume Phase)
Called close() (close connection)
Called close() or timedout (close connection)
Called close() (close connection)
Transition trigger (action)
State
Normal Transitions for CH
Normal Transitions for MH
Normal Transitions for Client
Normal Transitions for Server

Figure 3: MobileSocket State Transition Diagram

At "Closed" state, the MobileSocket connection is not connected to the remote host. At "Established" state, the connection between two MobileSocket libraries is established and applications at the both ends can communicate with each other through the MobileSocket. At "ExplicitlySuspended" state, the connection between the libraries is disconnected after the explicit **suspend** API is called by the application. The applications cannot communicate with each other unless they call `resume` API of MobileSocket. At "ImplicitlySuspended" state, the MobileSocket connection is disconnected implicitly by the libraries itself without any explicit API called from the applications.

In the state transition of MobileSocket, Closed state transits to Established state by connecting the initial socket connection. State transitions between Established and Explicitly Suspended are triggered by calling `suspend()` and `resume()` interfaces at the mobile host. Transitions between Established and Implicitly Suspended are triggered by the mobile host's sensing of the IP address reconfiguration.

4.4 DSS Time Sequence

In Dynamic Socket Switching (DSS), there are four particular phases, "DSS-EstablishmentPhase", "DSS-ExplicitSuspendPhase", "DSS-ExplicitResumePhase ", and "DSS-ImplicitResumePhase". Figure 4 shows the overview of DSS time sequence at the connection establishment, suspending, and resuming.

DSS-EstablishmentPhase

DSS-EstablishmentPhase is performed whenever the MobileSocket connection is being established. Figure 5 shows DSS-EstablishmentPhase.

DSS-EstablishmentPhase is described as follows.

(1) The client connects a DataSocket connection to the server.
(2) The server starts ControlSocket, a server socket, after the DataSocket acceptance, and send its port number and a seed for authentication to the client.
(3) The client makes a ControlSocket connection to the server with the port number and seed client just received.

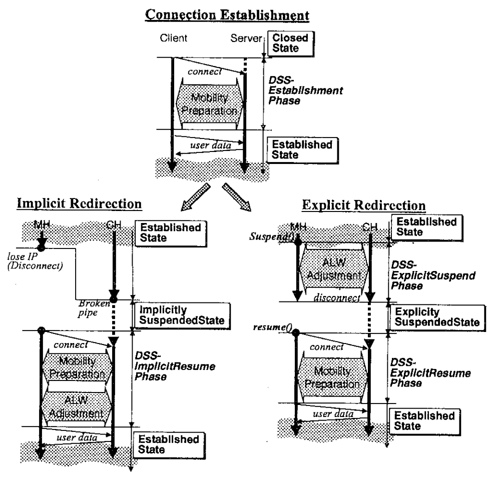

Figure 4: DSS Time Sequence

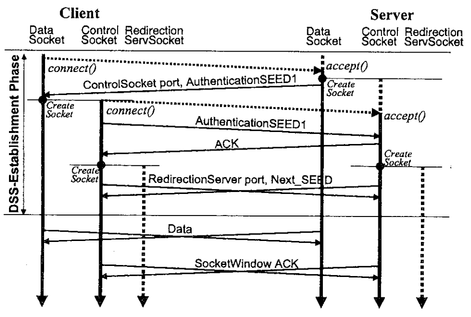

Figure 5: DSS-EstablishmentPhase

(4) After the authentication succeeded, both side makes RedirectionServerSocket, which is a server socket for next connection after the mobile host relocation.
(5) The client and the server server exchange the port numbers and the authentication seeds of RedirectionServerSockets.
(6) Actual byte stream communication between applications starts.

Relation between the client and the server does not depend on which side will be the Mobile Host (MH) that suspends and resumes connection, and which side will be the Correspondent Host (CH) that is suspended and resumed connection by the MH. Therefore the libraries at both sides make RedirectionServerSocket for the mobility.

DSS Explicit Redirection

DSS-ExplicitSuspendPhase is triggered by **suspend()** API (Java method) called from the application at the MH. In this phase MobileSocket locks writing and reading to and from the socket, confirms that all of byte stream data was read by remote host, and closes connection. Figure 6 shows the time sequence of DSS-ExplicitSuspendPhase.

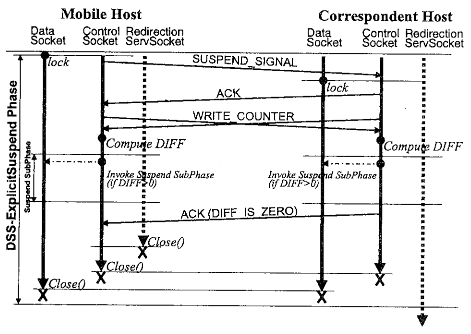

Figure 6: DSS-ExplicitSuspendPhase

DSS-ExplicitSuspendPhase is described as follows.
(1) As **suspend()** API is called by the application on the MH, the MH informs the CH about the explicit suspend phase by sending SUSPEND_SIGNAL through the ControlSocket.
(2) After both sides of connection locked the stream, they exchange WRITE_COUNTER which indicates the number of bytes the host wrote to the socket.
(3) Each side calculates the difference between its own READ_COUNTER and the WRITE_COUNTER from the remote.
(4) The library unlocked reading from the socket once if there is any difference, because it means that the host should read this "difference" of bytes more. Confirming that the application have read appropriate bytes of data, the library locks reading again.
(5) After the MH makes sure that both the MH and the CH have locked the stream finally, it close both DataSocket and ControlSocket connection.

When DSS-ExplicitResumePhase is triggered by **resume()** API called from the application at the MH during "ExplicitlySuspended" state, using Authentication seed received from the CH, MH tries to reconnect to the CH same way as the establishment phase.

DSS Implicit Redirection

When MobileSocket senses that the host has lost its IP address, the library transits into "ImplicitlySuspended" state. And the DSS-ImplicitResumePhase is triggered by sensing the host's reconnection to the network. In DSS-ImplicitResumePhase, after the MH obtains a new IP address, the MH connects to the RedirectionServerSocket of CH and reconstructs the MobileSocket connection, supported by ALW retransmission. Figure 7 shows the time sequence of the implicit suspending and DSS-ImplicitResumePhase.
(1) After MobileSocket in the MH senses obtaining a new IP address, the MH establishes new DataSocket

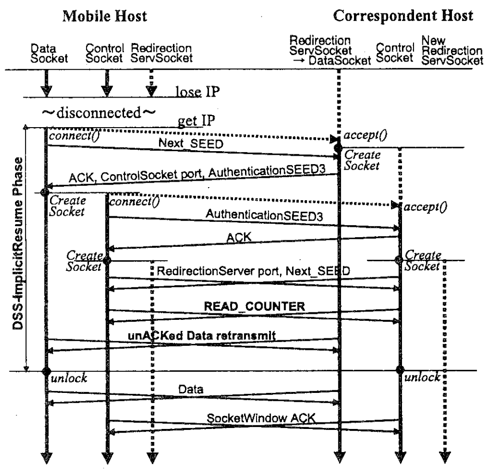

Figure 7: Implicit Suspending and DSS-ImplicitResumePhase

connection to the CH's RedirectionServerSocket.

(2) As the CH accepts this connection, the CH switches the socket and treats this socket as a new DataSocket.
(3) After the authentication checking, the CH sends the port number of ControlSocket and the next seed back to the MH as well as starts ControlSocket server.
(4) After the authentication checking of ControlSocket, both sides exchange READ_COUNTERs, which indicate the number of bytes each host already read from last internal socket connection.
(5) Both of the MH and the CH calculate the difference between their own WRITE_COUNTER and the READ_COUNTER from remote individually and retransmit the "difference" bytes of data to the remote from their own ALW.
(6) Both libraries unlock the DataSockets and applications restart to communicate with the new socket.

5 Performance Measurement

In this section, we present the performance evaluation of the connection redirection in the MobileSocket library. We can observe some overheads which can be reduced more by source code optimization, while the performance of internal socket depends on the Java environment.

5.1 Evaluation Environment

MobileSocket is implemented in Java. We use Java Development Kit (JDK) 1.1.6 on FreeBSD 2.2.1R. The TCP MobileSocket implementation consists of about 1,800 lines of Java source code.

Table 1 shows the platform we evaluated MobileSocket. The mobile host and the correspondent host are connected through closed 10Mbps Ethernet. In both of these hosts, we use FreeBSD 2.2.1R version with PAO-970616[6], PC Card support package, and Java Development Kit (JDK) 1.1.6. The following results are the mean values of 100 times measurements.

Table 1: Hosts Spec for Performance Evaluation

Host	Mobile Host	Correspondent Host
PC	Dynabook SS-R590 (TOSHIBA)	VAIO PCG-737 (SONY)
CPU	Pentium 90MHz	Pentium MMX 233MHz
Memory	40MB	40MB
OS	FreeBSD 2.2.1R & PAO-970616	
JavaVM	JDK 1.1.6.V98-9-23 for FreeBSD	

5.2 Explicit Suspending and Resuming

We measured the time consumed in `MobileSocket.suspend()` method, the explicit API to suspend MobileSocket connection, and `MobileSocket.resume()` method, the explicit connection resuming API. After the two Java application establish a MobileSocket connection, we measured the time with the `suspend()` and `resume()` method at the mobile host.

Result

Table 2 shows the detail times which are consumed in each process of DSS-ExplicitSuspend Phase, and table 3 shows those of DSS-ExplicitResume Phase. `suspend()` takes 46.67 milli-seconds, and `resume()` takes 270.28 milli-seconds.

Table 2: Detail of DSS-ExplicitSuspend Phase

Steps	Time(ms)	%
manage phase transition	1.76	3.77
lock Socket	7.40	15.86
kill sub-thread	8.12	17.40
send SUSPEND_SIGNAL	1.17	2.50
send WRITE_COUNTER	5.35	11.46
receive ACK from CH (wait for process in CH)	11.01	23.59
receive port number	1.11	2.38
receive auth seed	1.85	3.96
close Socket	3.28	7.03
prepare Info. of NextSocket	1.02	2.19
miscellaneous	4.60	9.86
Total	46.67	100.00

Table 3: Detail of DSS-ExplicitResume Phase

Steps	Time(ms)	%
make new DataSocket	80.75	29.88
switch Socket in stream	0.36	0.13
auth check for DataSocket	2.95	1.09
receive port of ControlSocket	1.11	0.41
receive authentication seed	1.89	0.70
make new ControlSocket	80.80	29.90
auth check for ControlSocket	3.30	1.22
make new NextServerSocket	60.44	22.36
exchange next port and seed	6.62	2.45
restart sub thread	26.56	9.83
manage phase transition	0.90	0.33
miscellaneous	4.60	1.70
Total	270.28	100.00

In the DSS-ExplicitSuspend Phase, except the waiting for ACK from the correspondent host, locking of Socket and termination of sub thread takes relatively higher ratio of whole operation. The mutual exclusion class, used in the locking part, is made for the serializable class, in order to make MobileSocket class serializable, and it causes overhead. Thread termination in Java depends on the implementation of Java Virtual Machine. Concerning about the waiting for the acknowledgment from the correspondent

Table 4: Functional Comparison

Name	Mobility of Layer	Connection Continuity		Implementation					
			Redirection	MH		CH		Others	Existing Apps
				K	U	K	U		
Mobile-IP	IP	NO		x				HA, FA	
PMI(+M-IP)	IP	NO		x	x			HA, FA	
MSOCKS	TCP	LIMITED	I		x			Proxy	x
TCP-R	TCP	YES	I	x		x			
VNC	Screen	YES	I		x		x		
MobileSocket	Socket	YES	I & E		x		x		

"I"...Implicit Operation, "E"...Explicit Operation, "K"...Kernel-Level, "U"...User-Level, "x"... necessary

host, two MobileSocket library in both ends of connection need to confirm that all bytes data each of them wrote to the socket has already read by the remote library. Therefore, it takes time both libraries to be synchronized.

In the DSS-ExplicitSuspend Phase, establishments of three internal sockets are big overhead and take 82.14% of whole operation, although these sockets are necessary for our approach. In contrast we can optimize the rest about 20% of operation by polishing our implementation, while the socket performance depends on Java compiler and the Java Virtual Machine (VM).

6 Discussion

In this section, we compare several related works[2, 3, 4, 5, 9]' functionalities with MobileSocket and discuss the efficiency of MobileSocket.

Table 4 shows the result of functional comparison between the related works and MobileSocket. We compared these works in the points of view of "mobility", "connection continuity", and "implementation".

We have adopted the application layer, socket level approach to mobility and connection continuity support. The main argument why we have this approach is that currently the most popular network programming interface for the applications is the socket. Under the situation that any protocol stacks are hidden from applications by socket interface, to simplify the structure and the implementation, we see that realizing the mobility and continuity in the socket layer is the most appropriate way.

On "Connection Continuity", MobileSocket provides connection continuity with both implicit and explicit redirection mechanism.

One of the design goal of MobileSocket is the user-level implementation. MobileSocket does not require the implementation any other than in the user-level of the MH and the CH. It means that the advantages of MobileSocket can be distributed to users by the application including MobileSocket.

7 Conclusion and Future Work

In this paper, we have presented the application layer solution for mobility and connection continuity, the MobileSocket library.

The MobileSocket library provides mobility and connection continuity for existing Java applications without any modification to their source code. The user-level library in the mobile host and the correspondent host simplifies and minimizes the implementation. The combination of Dynamic Socket Switching and Application Layer Window achieves byte stream consistency for the TCP Socket connection.

According to our functional comparison between some related works and MobileSocket, we see that MobileSocket provides both mobility and connection continuity in spite of its implementation simpleness, comparing other works. MobileSocket library is the competitive solution of mobility and connection continuity support for applications.

We have two major future work left. The one is the issue of the implementation optimization, especially in performance of socket creation in Java which causes current serious overhead. The other is the application of the user level approach for mobility and connection continuity to other resources, such as the file descriptor or the host devices. This will be effective for mobile applications and agents, which migrate to another hosts during their activities.

References

[1] Gosling, J., Joy, B. and Steele, G.: *The Java Language Specification*, Addison Wesley, Reading, Massachusetts (1996).

[2] Perkins, C.: IP mobility support (1996). RFC 2002, Internet Request For Comments.

[3] Maltz, D. and Bhagwat, P.: MSOCKS: An Architecture for Transport Layer Mobility, *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 1037–1045 (1998).

[4] Qu, X., Yu, J. and Brent, R.: A mobile TCP Socket, *Technical Report, TR-CS-97-08, Department of Computer Science, Australian National University* (1997).

[5] Funato, D., Yasuda, K. and Tokuda, H.: TCP-R: TCP Mobility Support for Continuous Operation, *Proceedings of IEEE International Conference on Network Protocols 97*, pp. 229–236 (1997).

[6] Hosokawa, T.: PAO: FreeBSD Mobile Computing Package. http://www.jp.freebsd.org/PAO/.

[7] Inouye, J., Cen, S., Pu, C. and Walpole, J.: System Support for Mobile Multimedia Applications, *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 143–154 (1997).

[8] Inouye, J., Binkley, J. and Walpole, J.: Dynamic Network Reconfiguration Support for Mobile Computers, *Proceedings of The Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97)*, pp. 13–22 (1997).

[9] Richardson, T., Stafford-Fraser, Q., Wood, K. and Hopper, A.: Virtual Network Computing, *IEEE Internet Computing*, Vol. 2, No. 1, pp. 33–38 (1998).