Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 695240, 11 pages
http://dx.doi.org/10.1155/2014/695240

Research Article

Intrusion-Tolerant Jini Service Architecture for Integrating Security and Survivability Support in DSN

Sung-Ki Kim,[1] Byung-Gyu Kim,[2] and Byoung-Joon Min[3]

[1] *Division of IT Education, Sun Moon University, 100 Kalsan-ri, Tangjeong-myeon, Asan-si 336-708, Republic of Korea*
[2] *Department of Computer Science and Engineering, Sun Moon University, 100 Kalsan-ri, Tangjeong-myeon, Asan-si 336-708, Republic of Korea*
[3] *Department of Computer Science and Engineering, Incheon National University, 119 Academy-ro, Yeonsu-gu, 406-772 Incheon, Republic of Korea*

Correspondence should be addressed to Byoung-Joon Min; bjmin@incheon.ac.kr

Received 30 August 2013; Accepted 3 December 2013; Published 22 January 2014

Academic Editor: James J. Park

Ubiquitous service environment based on DSN (distributed sensor networks) is poor in reliability of connection and has a high probability that the intrusion and the system failure may occur. In this paper, we propose an intrusion-tolerant Jini service architecture for integrating security and survivability support in order to provide end users with Jini services having a persistent state in ubiquitous environments. The proposed architecture is able to protect Jini service delivery not only from faults such as network partitioning or server crash but also from attacks exploiting flaws. It is designed to provide performance enough to show a low response latency so as to support seamless service usage. Through the experiment on a testbed, we have confirmed that the architecture is able to provide high security and availability at the level that the degradation of services quality is ignorable.

1. Introduction

Jini [1], also called Apache River [2], is a java-based middleware supporting share of resources such as ubiquitous devices and software on networks while it copes with the heterogeneity of the lower levels such as the various types of devices or communication protocols. A service that can cause us to use these pervasive resources is called the ubiquitous service. Jini provides a mechanism that discovers available services through the lookup services and make a connection to the services that clients requested.

It is important to enhance the survivability of the ubiquitous service because we live depending on the ubiquitous service environment providing valuable services through pervasive use of computation in everyday experiences.

However, the networked systems in the environment are apt to be partitioned due to a poor reliability of connection and have a high probability that the intrusion against a system providing services and the failure of the services may happen [3]. Therefore, it is very important to guarantee that the legitimate users make use of trustable services without discontinuance or obstacle of the services they are enjoying.

The standard Jini has two main problems in regard to these requirements. First, the Jini does not support building of fault tolerance services to mask the failure of services leased for resource use [4]. Second, the Jini has an insufficient security mechanism that cannot support dynamic trust establishment within a Jini system [5].

The former problem means that clients cannot keep using their service they are enjoying persistently when a Jini system is partitioned due to both communication link failures and server (service provider) crash failures. To solve this problem, the Jini system should have a mechanism that makes all service components in the system aware of failure events such as both network partition and server crash in a timely manner, and which replicated servers in the Jini system can maintain the user service states on behalf of failed servers persistently. The standard Jini architecture has not a real-time fault detection mechanism and does not support the replication of services for developing fault tolerant service.

The later problem is a security problem caused by the characteristics of Jini technology that sets free developing of a Jini system from the addition, modification of services, and the federation among all existing other services on networks. To respond to the security threats, the Jini system must have a security mechanism that is not only able to establish the trust among entities comprising a Jini system so that malicious services and their proxies are not introduced into the system, but also to respond to the intrusion exploiting flaws in the system.

The Jgroup/ARM framework [6] presented a middleware technology supporting the building of a dependable service in a distributed environment by introducing a concept that is called Java-based object group platform. A set of distributed objects making a group takes the responsibility for a service. It provides a good framework to solve the first problem of the standard Jini system as mentioned above. However, there are some shortcomings to apply this framework in the real environment directly. The first shortcoming is in which the omission of the design that the security provision responding to the intrusion has not been considered in the framework design. The Jgroup/ARM framework focuses on how to reach agreement among the service states of each of the Jini service entities, while the system that is built by this framework tolerates the events such as network partition or server crash, so it does not consider the problems of security. The system that is made through this framework is of greater danger more than the standard Jini system because it is defenseless against intrusion due to no security mechanism and it can be easily compromised caused by group communication protocols performing share of service state data among replicated service objects as well. Once an attacker has succeeded in the intrusion just in a part of the system, the entire Jini distributed systems may be seriously compromised. The second shortcoming is in which the overhead causes both the unnecessary computation and communication costs to delay the reply to the clients. The third shortcoming is in which both the computation and communication costs are needed to merge the user service states among replicas are very high, when the partitioned replicas are merged into one group after a network is recovered, and in the case of having a lot of clients keeping connections to the replicas.

In this paper, we present the solutions to improve the shortcomings that the Jgroup/ARM framework has and then propose a secure Jini service architecture providing ubiquitous services having a persistent state, while it establishes the dynamic trust in a Jini system and tolerates the failures and intrusions. In addition, we discuss the experimental results in terms of the proposed intrusion-tolerant Jini service architecture. Section 2 briefly describes the standard Jini service environment and points out its problems and then discusses the related work. Section 3 introduces the Jgroup/ARM framework and briefly describes what it overcomes when the standard Jini system meets the failure conditions such as network partition or server crash, and then it analyzes their shortcomings. In Section 4, we introduce our secure Jini service architecture. Section 5 discusses the experimental results in terms of our architecture that has been introduced in Section 4. Section 6 concludes the paper.

2. The Problems and Related Work

2.1. The Standard Jini Service Environment. A Jini system consists of the lookup servers, clients, and servers in which service codes are implemented. The servers and clients discover the presence of lookup servers in the ubiquitous network environment by performing the discovery protocol. After discovery of any lookup server anywhere, the servers and clients download a lookup service proxy that the lookup server provides. Afterward, the servers register their own service proxies with the lookup server and the clients discover a set of available service implementations through the lookup service proxy. At this time, if a client selects a specific service of available services, the client gets a service proxy that the corresponding server had registered from the lookup server. Subsequently, the clients call the remote service implementation through the downloaded service proxy. All request/response messages are transferred to be transparent to the lower layer protocols by using Java RMI (remote method invocation) protocols.

A scope that a server or a client can discover a lookup server depends on how far a multicast discovery message can be transferred. In generally, a range of these messages depends on the TTL parameter of multicast discovery packets. However, the lookup servers, servers, and clients comprising a Jini system have to be included in a same multicast group and the routers have to support the delivery of the multicast discovery packets. Thus, the distributed lookup servers on networks can discover the presence of each other and federate the services they introduce through the discovery protocol execution, but even so the scope of the Jini service environment cannot help, but it is limited to the edge of a manageable network.

If the limitations of the lookup server discovery and the service federation as mentioned above are solved through the interdomain cooperation, the scope of a Jini service environment can be more expanded because the range of multicast discovery messages is limited, but the range of RMI messages is not limited.

2.2. The Problems in the Jini Service Environment. Problems in the Jini service environment can be divided into four categories, namely, how to enhance the availability, quality, and security of Jini services and how to ensure the consistency of Jini service states.

The main causes of decreasing the availability of Jini services include the case of which the Jini distributed systems are broken due to both network partition and server crash failures. In general, a solution for enhancing the availability of services by overcoming these problems is to build the system based on the replication of the service [7]. However, the replication of the service carries how to ensure the consistency of replicated service states.

The Jini service environments overcome the heterogeneity of devices by using the JVM (Java Virtual Machine). System development in java in order to overcome the heterogeneity of devices reduces the burden of system development, but it may provide attackers with security holes that will help to exploit flaws in different types of devices comprising

the system. For example, a method to hide attacker's intents in native code in which a Java runtime code calls through JNI (Java Native Interface) is an untouchable security flaw in java-based systems [8]. The reason is that no technologies based on java language can guarantee the security of native code below JVM layer, but they can guarantee the security of code above it. The Jini service environment sets free the discovery, addition, and modification of services on ubiquitous networks on the basis of mobile codes such as service proxies while they bear these risks. Therefore, the Jini systems must have a mechanism not only to provide security services such as enforcement of authentication and authorization among the Jini entities, ensuring of confidentiality and integrity for exchanged messages, but also to respond to intrusion.

The building of a Jini service environment considering the four problems as mentioned above generates the trade-off problems. In other words, applying the replication to the Jini services to enhance the availability makes it more difficult to ensure the consistency of their states and increases the targets that will have to protect from the viewpoint of security. In addition, increasing the degree of replication and enhancing the security incur reducing the quality of services due to considerable delays that is required to reply to the clients.

2.3. Related Work. We have not found works resolving all four problems as mentioned in Section 2.2 yet. Meling et al. [6] proposed Java-based middleware architecture and a programming model, which ensure consistency among the replicated service states in spite of applying replication to service objects in order to enhance their availability. We discuss in detail this work in Section 3.

Kolltveit and Hvasshovd [9] discussed the experimental results on the response time that was measured at the client from transaction initiation to transaction completion on condition that the two different servers can have up to two replicas each and a transaction manager (TM) can have up to four replicas on the middleware architecture that [6] proposed. This work also has investigated whether the response times are allowable latency that can satisfy a seamless service use under failover operations with recovery activities.

Meling and Helvik [10] presented the techniques to maintain consistency between the dynamic server-side group membership and its representations both at the client-side and in the registry as equivalent to the lookup server in the event of server failures and recoveries.

In order to solve the single point of failure problem of servers, Tichy and Giese [11] introduced an approach to which a client has multiple connections to replicated servers as applying the client to smart-proxy providing that function. The ensuring of consistency among the replicated servers relies on the communication protocols that are performed between the servers and the smart-proxy.

Several works [12–14] were carried out on how to enhance the security in the standard Jini system. The security requirements for the Jini system that had been suggested in these works commonly are as follows:

(1) trust establishment among the client, the service proxy and the server;

(2) access control to services;

(3) protecting the client from mobile code;

(4) ensuring the confidentiality and integrity of communication.

The security requirement (1) means that the Jini system needs to have the authentication mechanism that establishes the trust among the Jini entities. The security requirement (2) means that the Jini system needs to have the authorization mechanism forcing authenticated users to make use of services within their own right. The security requirement (3) means that a mechanism is necessary for preventing the client from the execution of download proxy codes that may impair them but are authenticated. The security requirement (4) means that all exchanged messages in Jini system have to be protected from tempering by attacker through cryptography technology.

Hasselmeyer et al. [12] presented a Jini service architecture that meets the security requirement associated with access control and trust establishment among server, service proxy, and lookup server through centralized security server providing authentication and authorization services in the Jini system.

Eronen and Nikander [13] proposed a method that establishes an authentication-authorization chain among users, service proxies, and servers by using the SPKI (Simple Public key infrastructure) certificate in Jini system.

Schoch et al. [14] presented a Jini service architecture similar to that of [12]. However, in this architecture, both the mechanism ensuring the confidentiality and integrity of communication and the client-side proxy supporting considerable security features so as not to break client transparency differ from those of the other works [12, 13].

As regards the safety of download proxy code, [12–14] are based on an assumption that all the authenticated proxy codes are trustable from the view point of safety in the client side. To support the security of communication, the communication implementation in [12] is based on the SSL-based Java RMI implementation using Java-based open source API. In [13], the communication implementation is based on the TLS-based Java RMI implementation using the standard JSSE (Java socket security extension) API. The implementation in [14] uses the DH (Diffie-Hallman)-based algorithm and HMAC-MD5 algorithm to ensure the confidentiality and integrity of all exchanged messages, respectively.

The Jini technology has reinforced the supports to meet four security requirements as mentioned above. Especially, a constraint-based remote invocation mechanism for protecting clients from mobile codes has been added.

There are two works on intrusion tolerance systems based on diverse server replication. One example is HACQIT (hierarchical adaptive control of quality of service for intrusion tolerance) project [15], and another example is SITAR (scalable intrusion tolerance architecture) [16]. The method to commit secure result with replicated servers is found in [17]. A coordination model for improving software

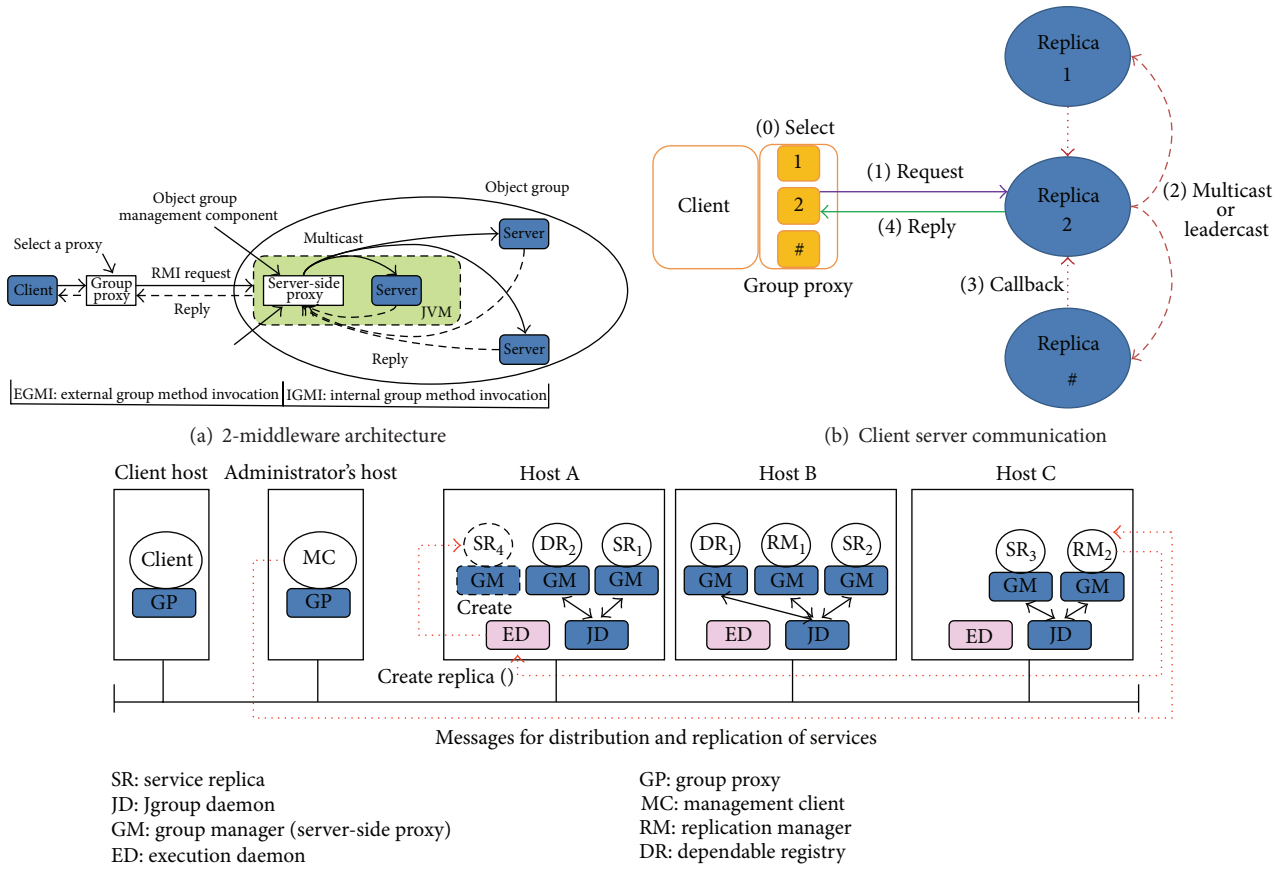

FIGURE 1: The Overview of the Jgroup/ARM system.

system attack-tolerance and survivability is also found in [18]. An analysis of replication enhancement for a cluster-based high availability service is found in [19]. The method for cost-saving key agreement via secret sharing in two-party communication systems is found in [20].

3. The Jgroup/ARM System

3.1. The Jgroup/ARM System Architecture. The architectural features of the Jgroup/ARM system are summarized in Figure 1. As shown in Figure 1(a), the Jgroup/ARM system supports 2 middlewares: one is for client-server communication; the other is for communication among replicated service objects. If a proxy is selected from the group-proxy in the client-side, the selected proxy sends an RMI request corresponding to their service replica. The server-side proxy (GM in Figure 1(a)) in the corresponding replica dispatches the request message to their group members by using multicast or leadercast delivery. The use of multicast delivery means that the replica that is selected from client-side proxy has responsibility of dispatching a request message to other members of replica group in the server-side (SR in Figure 1(c)) and merging the results from them. The use of leadercast delivery means that the replica that is selected from client-side proxy performs their service and then shares the result of their service with other group members. Figures 1(a) and 1(b) illustrate this concept. In order to overcome failure situation that cannot deliver the implemented service, Jgroup/ARM system supports a fault tolerance mechanism such as failure detection, reliable group membership communication, and dynamic resource reallocation. Figure 1(c) illustrates these features.

3.2. The Contribution of Jgroup/ARM System. In this section, we do not describe how Jgroup/ARM system overcomes the failure events such as both network partition and server crash in detail. We just discuss what the Jgroup/ARM framework contributes to the standard Jini system and then point out the shortcoming this framework.

Figures 2(a)–2(d) illustrate service failure situations that may happen in the standard Jini service environment. In Figure 2(a), all 65 users can make use of a Jini service without any problems. However, when a lookup server is crashed as shown in Figure 2(b), all existing users keeping connections to the server can make use of the service within a given lease time. In the case of Figure 2(c), 15 users lose their connection and the opportunities that they can discover other services again. 30 users lose their connection to the server and only can rediscover unreachable service, and the presence of the service disappears from the lookup server. Only 20

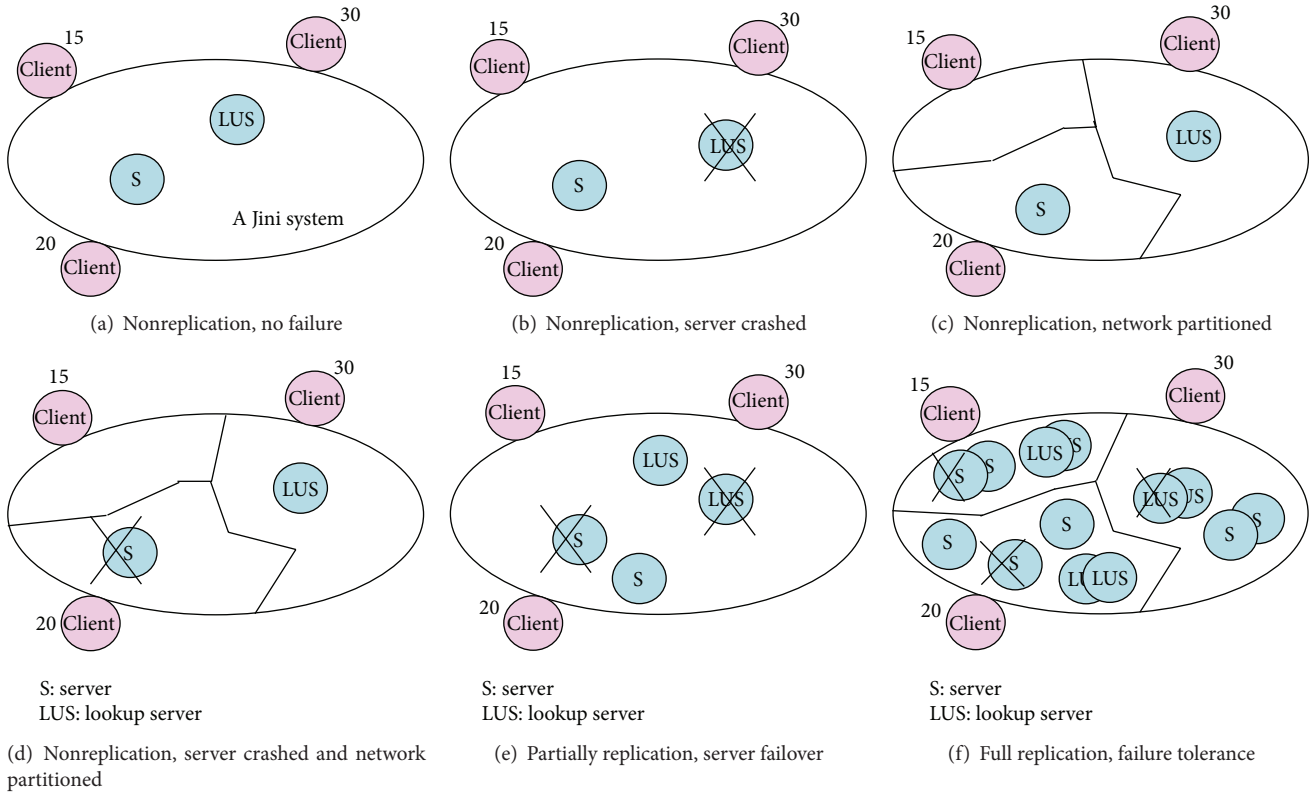

Figure 2: Service failure and tolerance in a Jgroup/ARM Environment.

users can make use of the service within a given lease time. In the case of Figure 2(d), all users lose their connection to the server and only 30 users can rediscover unreachable service. In the case of Figure 2(e), some users are subject to restriction on making use of services according to how a Jini network is partitioned, but the situation that servers are crashed can be overcome. In the case of Figure 2(f), the Jini system overcomes failure events such as a specific server's crash and network partition by replicating the tasks providing the services on distributed computing nodes. The Jgroup/ARM system supports the case of Figure 2(f). In Figure 2(f), though considerable service resources are replicated, the Jgroup/ARM system can support the autonomous management of available resources so as to overcome all failure events through a dynamic service deployment and reconfiguration.

The service replication for supporting the case of Figure 2(f) in the standard Jini service environment without use of the Jgroup/ARM framework is possible. However, this approach generates the following two problems. First, in the standard Jini service environment, the lookup server introduces users to the failed servers repeatedly until the lease time of their proxies given by the lookup server is expired. Second, the users become aware of the failure of services after their proxies disappear in the lookup server because no components comprising the system can detect the failure event such as both network partition and server crash in timely manner. After all, it is hard for users to look forward to the quality of services.

3.3. A Definition and Examples for Discussion

Definition. In this section, we define the notion of a group of the distributed service objects for the further discussions as follows:

$$G_s \longrightarrow \{R_1, R_2, R_3, \ldots, R_n\}^e, \tag{1}$$

where G_s is a server group that consists of a group of n replicated service objects, R_n: the nth replica which is a member of G_s, e is the number of endpoints (i.e., service instances) that is created in each R_n in order to establish connection from clients under the no failure condition that the system is not partitioned.

Example 1. The expression, $G_s \rightarrow \{R_1, R_2, R_3, \ldots, R_7\}^{20}$ means that all of 7 replicas takes the responsibility for a service and each replica has equally 20 connections from clients. This expression describes normal scenarios under no failure condition.

Example 2. The expression, $G_s \rightarrow \{\{R_1, R_2\}_a^3, \{R_3, R_4, R_5\}_b^7, \{R_6, R_7\}_c^{10}\}$, means that one server group is partitioned into 3 partitioned subgroups that take the responsibility for a service. This expression describes failure scenarios that

the system is partitioned due to the intrusions and the system failures. In this scenario, replicas in each partitioned subgroup have endpoints enough to connect to their clients according to how a Jini network is partitioned. In this example, each replica in the subgroups*a*, *b*, and *c* has 3, 7, and 10 endpoints to connect to their client side proxy, respectively.

3.4. The Shortcomings of Jgroup/ARM System

The Computational Overhead for Merging Distributed Service States of End User. When the partitioned subgroups are merged into one group after the network failure is recovered, the Jgroup/ARM system faces a significant problem in bearing a heavy computation overhead for merging distributed service states of end user. For recovery, the Jgroup/ARM system has a 2-phase merging operation as follows:

(i) merging operation between leader replicas (e.g., R_1-R_3-R_6);

(ii) merging operation between a leader and members (e.g., R_1-R_2, R_3-R_4-R_5, and R_6-R_7).

The time-cost for these 2-phase merging operations in the Jgroup/ARM system can be estimated as follows:

$$\text{Time} - \text{Cost}_{(\text{Jgroup System})} = \sum_{i=1}^{p} \left\{ e_i^p * \left(e - e_i^p \right) * (mdt + c) \right\}, \tag{2}$$

where p is the number of partitioned groups, e_i^p is the number of endpoints in each R_n of the ith partitioned subgroups after network is partitioned, and c is a required time to make the service instances in each endpoint, and also *mts* means multicast delay time for message delivery in group communication.

Latency in Callback Handler. In the Jgroup/ARM system, a group proxy [10] for the use of a service is offered, and a selected proxy in the group of proxies sends the requests to server-side replica corresponding to it. The selected replica replies their service result on behalf of all of members after the callback handler in the selected replica receives the service results from all members. This latency in callback handler operation is a cost to pay for fault detection and failover provision.

The Lack of the Security Mechanism. The group communication among replicas relies on the Jgroup Daemon (JD) in every node hosting services [6]. However, the JD supporting group multicast communication does not ensure the confidentiality and integrity of messages. In addition, there is no authentication mechanism providing the trust among replicas. Thus, attackers can easily intrude into the system by tempering all communication messages and configuration files.

4. Our Proposed Architecture

4.1. Architectural Features. The goal of our proposed architecture is to support the integrated services that can provide security and service survivability in order to respond against intrusion and failure scenario. To achieve this goal, we have developed the extension of Jgroup/ARM framework. Figures 3 and 4 show a high-level system overview and architectural extension for security.

As shown in Figure 3, the service replicas can be distributed on necessary computing nodes and the service object repository (SOR) as a service manages the reposition of these distributed service replicas. According to the configuration supporting domain administration policy, these service replicas can be activated or inactivated by control from replication manager (RM). Unlike the Jgroup/ARM architecture, our system supports secure group membership communication in the underlying communication mechanism. Figure 4 illustrates our architectural extension for security and service state management. To support key management for secure group communication, we have adopted the use of the contributory group key agreement protocol [21] in the secure reliable group membership multicast layer.

The service provider provides their service state data to group manager (GM) through setServiceState (SID, KEY, STATE) method. SID is a service ID and KEY is a shared group key used while being on communication. STATE is a reference to a context container that caches service state data including session ID. These service state data is stored in GM's service state repository (SSR) which has interaction with the smart proxy in the client-side. GM gets the data associated with user service state from smart proxy when the absentee data are needed in merging operation.

4.2. An Extension to Session-Based Allocation of Replica Instances. In comparison with the Jgroup/ARM architecture, one of extended features in our proposed architecture is to support the identification and management of endpoints in each replica on the basis of the session while maintaining group membership among replicas regardless of merging subgroups. In our system, a smart group proxy in the client-side manages the user service states whenever it receives reply messages in order to solve the problem above. The smart group proxy is a group of smart proxies. Each smart proxy as a wrapper around a stub performs the predefined functions associated with remote service implementation (i.e., service provider) after the reception of reply message. The predefined function manages a cache for saving the user service states associated with remote service implementation. This cache can be used as a backup to help in the recovery of the user service state when the partitioned subgroups are merged into a single group. That is, just like the use of cookie in the web application, the user service state data cached in each smart proxy can be offered to leadercast contributor which has responsibility of the distribution of the absentee data as shown in Table 1.

Figures 5 and 6 show our architecture and contribution in comparison with Jgroup/ARM system through Example 2 in Section 3.3 and expression (2) in Section 3.4, respectively.

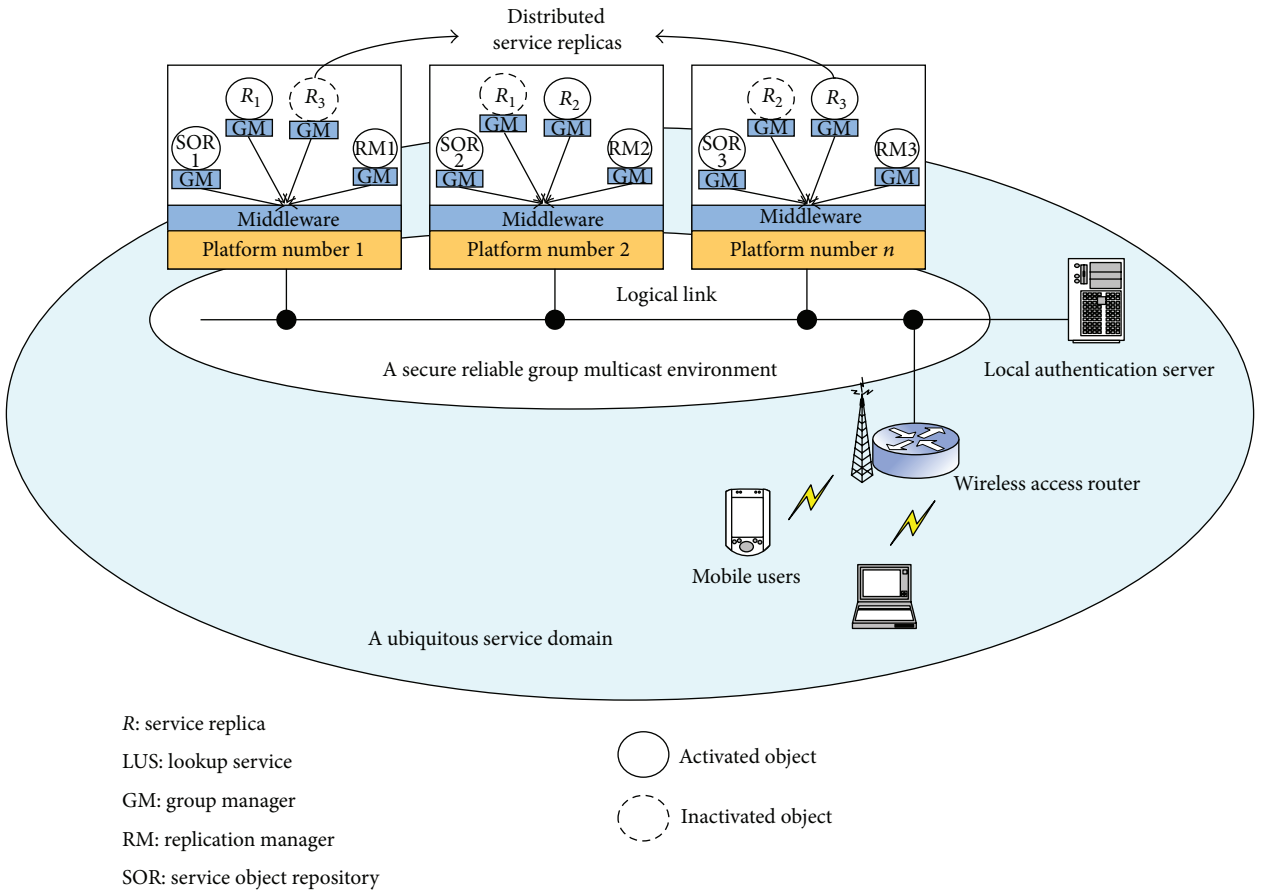

Figure 3: Our system overview.

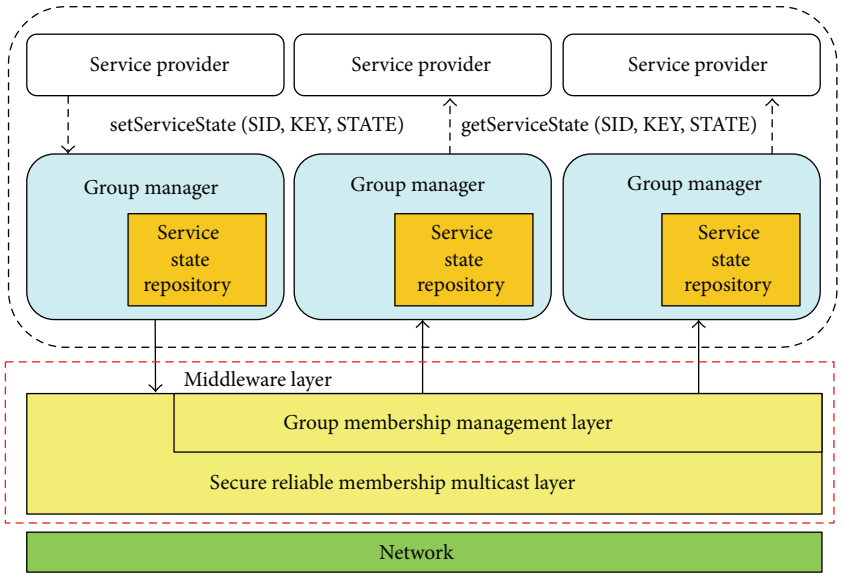

Figure 4: Architectural extension for secure group membership communication.

Table 1: A Simple simulation that is to be illustratedin Figure 6.

Replica	No failure condition	Before recovery	After recovery	Absentee data	Leadercast contributor
R_1	U_1	U_1, U_2	$U_1, U_2, U_5,$	U_3, U_4	R_3, R_6
R_2	U_1	U_1, U_2	$U_1, U_2, U_5,$		
R_3	U_1	U_1, U_3	$U_1, U_3, U_5,$	U_2, U_4	R_1, R_6
R_4	U_1	U_1, U_3	$U_1, U_3, U_5,$		
R_5	U_1	U_1, U_3	$U_1, U_3, U_5,$		
R_6	U_1	U_1, U_4	$U_1, U_4, U_5,$	U_2, U_3	R_1, R_3
R_7	U_1	U_1, U_4	$U_1, U_4, U_5,$		

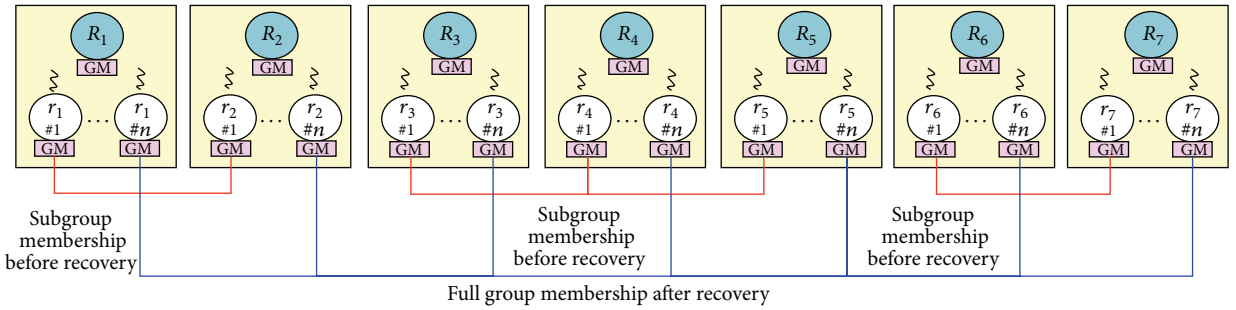

Figure 5: Session-based allocation of replica instances and their membership management.

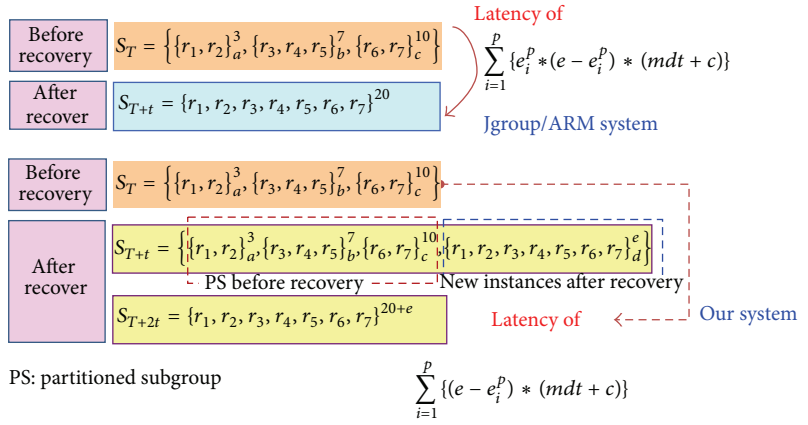

Figure 6: Reduction of time-cost for merging the user service states.

Table 1 illustrates Figure 6 above. In Table 1, U means users who send requests through client for use of a service, and the prefix i indicates a number to identify individuals. The time-cost for merging the user service states in our system can be estimated as follows:

$$\text{Time} - \text{Cost}_{(\text{our system})} = \sum_{i=1}^{p} \left\{ \left(e - e_i^p \right) * (mdt + c) \right\}. \quad (3)$$

4.3. Callback Handler for Intrusion Tolerance. Our proposed architecture applies the byzantine agreement algorithm [17, 22] and design diversity to service implementation in order to mask results introduced from compromised server due to intrusions. To accomplish this goal, we have added an additional function to callback handler in order to act as a voter masking the compromised results. When the number of total replica is N and the number of compromised replica is T of N, if there are replicas of more than 2/3 N to satisfy a condition that $N > 3T$, callback handler replies the service results to client.

4.4. Security Architecture for Secure Communication. Our proposed architecture supports security provisions to satisfy the goals for providing secure Jini services such as authentication, access control and confidentiality as shown in (see Figure 7).

Unlike server-side group communication feature as mentioned in Section 4.1, in our system, the communication between client and server uses a protocol as shown in Figure 7.

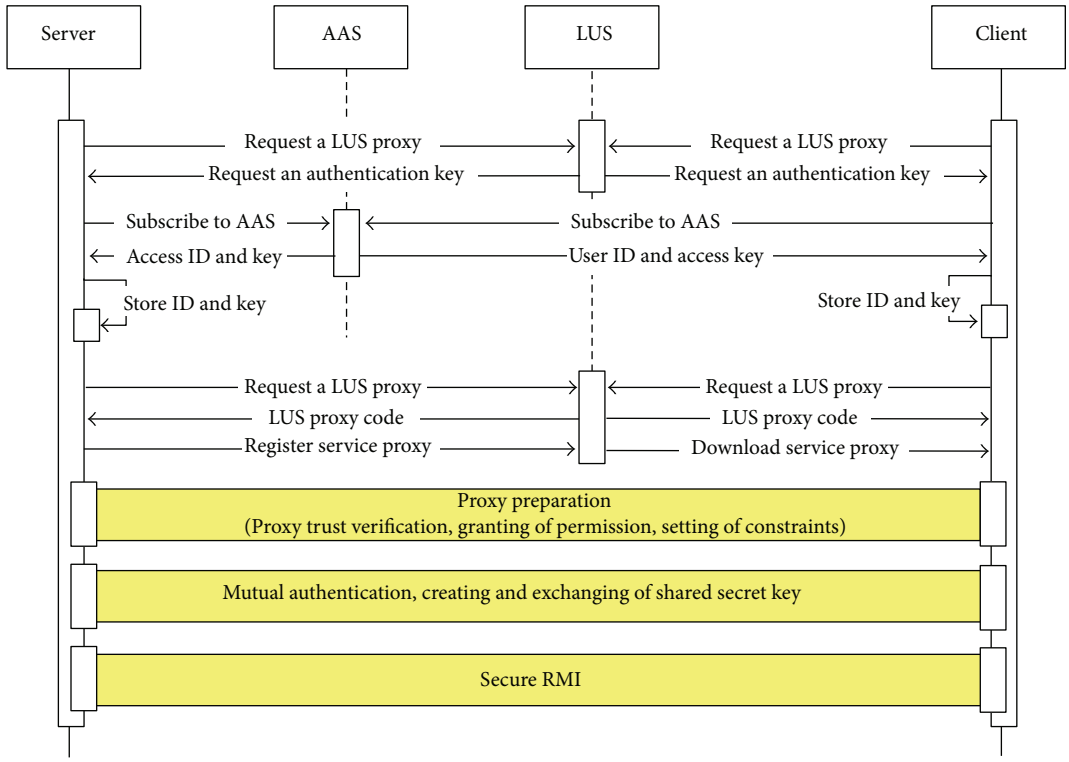

Figure 7: Architecture for Secure Communication between Client and Server.

5. The Discussion on Experimental Results

5.1. Performance Analysis. We have experimented with measures of user state merging times and turnaround times. When a recovery operation to merge the user service states has been performed under system-partitioned condition as shown in Figure 8, it has been shown that the time-cost for recovery operation in our system is 2 times lower than one in Jgroup/ARM system in proportional to e (i.e., the number of users in condition that a user has one connection to the service) of expression (1). In Figure 8, "our systems 2PG, 3P/PG" are meant for the condition that our system has been partitioned into 2 partitioned subgroups and each partitioned subgroup has 3 service replicas.

Figure 9 shows measurement results of turnaround time in concern with leadercast, multicast, and voting application in condition that the security protocol of Figure 7 has not been applied. In Figure 9, if the number of replica increases, the leadercast application shows that the turnaround time is reduced because the resource allocation for serving client's requests is distributed among replicas. In contrast, the multicast application shows that the turnaround time is increased because the client's requests are allocated in all replicas. In particular, the voting application shows that the turnaround time is not impacted by the number of replica because of callback handler for intrusion tolerance as mentioned in Section 4.3.

Table 2: Time cost for seamless service use.

Measurements	Latency
Failover	180 ms
Failure detection	160 ms
Reception of stateful service from other replica	340 ms

5.2. The Implementation Results. Table 2 shows results measured while a file transfer service is keeping up its mission in spite of a server-crashed failure. In this experiment, when a system consists of one client and two replicated services in 100 Mbps Ethernet environment, it has been shown that the failover latency is 340 ms (160 ms for failure detection + 180 ms for receiving stateful service from other replica) for seamless service usage.

6. Conclusion

Ubiquitous service environment based on DSN is poor in reliability of connection and has a high probability that the intrusion and the system failure may occur. In this paper, we propose an intrusion-tolerant Jini service architecture for integrating security and survivability mechanisms in order to provide end users with trustable Jini services having a persistent state in DSN. The proposed architecture is able to protect a Jini system not only from faults such as network partitioning or server crash, but also from attacks exploiting

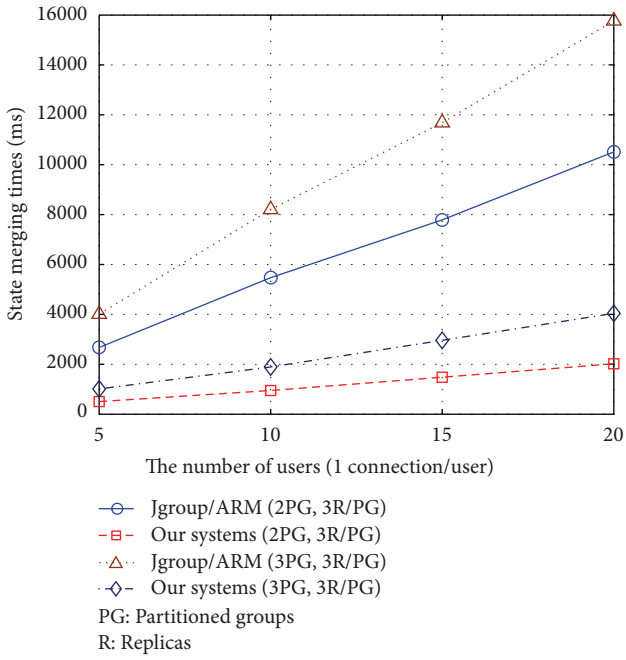

Figure 8: Time cost merging user service states.

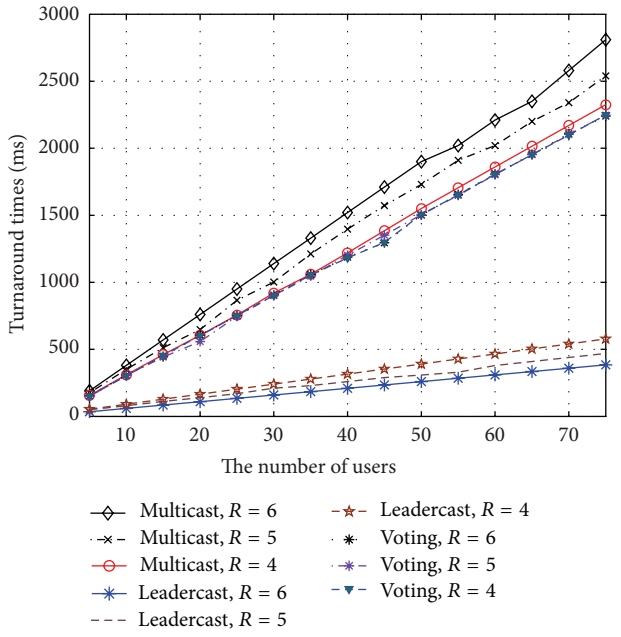

Figure 9: Turnaround time cost.

flaws. It is designed to provide performance enough to show a low response latency so as to support seamless service usage. We believe that our proposed architecture is a good reference model for building a better secure ubiquitous service infrastructure.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was supported by the Incheon National University Research Grant.

References

[1] Apache Software Foundation, "Apache Jini Specifications v2.1.2," http://river.apache.org/doc/spec-index.html.

[2] Apache Software Foundation, "Apache River User Guide," http://river.apache.org/user-guide-basic-river-services.html.

[3] K. Peng, "A secure network for mobile wireless service," *Journal of Information Processing Systems*, vol. 9, no. 2, pp. 247–258, 2013.

[4] D. Szentivanyi and S. Nadjm-Tehrani, "Middleware support for fault tolerance," in *Middleware for Communications*, Q. Mahmoud, Ed., chapter 28, John Wiley & Sons, 2004.

[5] F. Sommers, *Jini Starter Kit 2.0 Tightens Jini's Security Framework*, IEEE Computer Society Press, Los Alamitos, Calif, USA, 2003.

[6] H. Meling, A. Montresor, B. E. Helvik, and O. Babaoglu, "Jgroup/ARM: a distributed object group platform with autonomous replication managements," in *Software Practice and Experience*, John Wiley & Sons, 2007.

[7] J. Osrael, L. Froihofer, G. Stoifl et al., "Using replication to build highly available .net applications," in *Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pp. 385–398, 2006.

[8] M. Schönefeld, "Hunting flaws in JDK," in *Proceedings of the Blackhat Europe*, May 2003.

[9] H. Kolltveit and S.-O. Hvasshovd, "Preventing orphan requests by integrating replication and transactions," in *Advances in Databases and Information Systems*, vol. 4690 of *Lecture Notes in Computer Science*, pp. 41–54, Springer, Berlin, Germany, 2007.

[10] H. Meling and B. E. Helvik, "Performance consequences of inconsistent client-side membership information in the open group model," in *Proceedings of the 23rd IEEE International Performance, Computing, and Communications Conference (IPCCC '04)*, pp. 777–782, April 2004.

[11] M. Tichy and H. Giese, "An architecture for configurable dependability of application services," in *Proceedings of the ICSE Workshop on Software Architectures for Dependable Systems*, pp. 65–70, Portland, Ore, USA, April 2003.

[12] P. Hasselmeyer, R. Kehr, and M. Voß, "Trade-offs in a secure jini service architecture," in *Trends in Distributed Systems: Towards a Universal Service Market*, vol. 1890 of *Lecture Notes in Computer Science*, pp. 190–201, Springer, Berlin, Germany, 2000.

[13] P. Eronen and P. Nikander, "Decentralized Jini security," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '01)*, pp. 161–172, San Diego, Calif, USA, February 2001.

[14] T. Schoch, O. Krone, and H. Federrath, "Making jini secure," in *Proceedings of the 4th International Conference on Electronic Commerce Research*, pp. 276–286, November 2001.

[15] J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich, and K. Levitt, "The design and implementation of an intrusion

tolerant system," in *Proceedings of the International Conference on Dependable Systems and Networks (DNS '02)*, pp. 285–290, June 2002.

[16] F. Wang and R. Upppalli, "SITAR: a scalable intrusion-tolerant architecture for distributed services," in *Proceedings of the DARPA Information Survivability Conference and Exposition*, 2003.

[17] B. J. Min, S. K. Kim, and C. Im, *Committing Secure Results with Replicated Servers*, vol. 3043 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2004.

[18] S. Ren, Y. Yu, K. A. Kwiat, and J. Tsai, "A coordination model for improving software system attack-tolerance and survivability in open hostile environments," *International Journal of Distributed Sensor Networks*, vol. 3, no. 2, pp. 175–199, 2007.

[19] S. Park, I. Y. Jung, H. Eom, and H. Y. Yeom, "An analysis of replication enhancement for a high availability cluster," *Journal of Information Processing Systems*, vol. 9, no. 2, pp. 205–216, 2013.

[20] Y. Amir, Y. Kim, C. Nita-Rotaru, J. L. Schultz, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, pp. 468–480, 2004.

[21] Y. Amir, Y. Kim, C. Nita-Rotaru, J. L. Schultz, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, pp. 468–480, 2004.

[22] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.