A Group Communication Approach for Mobile Computing MobileChannel: an ISIS Tool for Mobile Services*

Kenjiro Cho
Kenneth P. Birman

TR 94-1424
May 1994

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* This work was performed at Cornell University, and was supported under ARPA/ONR grant N00014-92-J-1866, and by a grant from Canon Inc., currently visiting the Dept. of Computer Science at Cornell University. Ken Birman is a Professor in the Dept. of Computer Science at Cornell University, and Chief Scientist of Isis Distributed Systems, a division of Stratus Computer, Inc. Author's email addresses are {kjc, ken}@cs.cornell.edu.

A Group Communication Approach for Mobile Computing

MobileChannel: an ISIS tool for mobile services

Kenjiro Cho Kenneth P. Birman

Department of Computer Science, Cornell University
Ithaca, New York 14853-7501

May 17, 1994

Abstract

This paper examines group communication as an infrastructure to support mobility of users, and presents a simple scheme to support user mobility by means of switching a control point between replicated servers. We describe the design and implementation of a set of tools, called MobileChannel, for use with the ISIS system. MobileChannel is based on a combination of the two replication schemes: the primary-backup approach and the state machine approach. MobileChannel implements a reliable one-to-many FIFO channel, in which a mobile client sees a single reliable server; servers, acting as a state machine, see multicast messages from clients. Migrations of mobile clients are handled as an intentional primary switch, and hand-offs or server failures are completely masked to mobile clients. To achieve high performance, servers are replicated at a sliding-window level. Our scheme provides a simple abstraction of migration, eliminates complicated hand-off protocols, provides fault-tolerance and is implemented within the existing group communication mechanism.

1 Introduction

Emerging handheld computers with communication capabilities will have a significant impact on distributed computing. Whereas the traditional distributed systems are designed assuming a static network connectivity, mobility of machines in the system poses numerous problems that need to be solved. As mobile users travel around in a network, the network topology must be dynamically reconfigured. Especially, in a cellular wireless network, users can travel between wireless cells while maintaining connectivity. This process, switching a route in the middle of communication, is called a *hand-off*. Hand-off requires the system to maintain the end-to-end connectivity in the presence of a dynamically changing network topology.

When designing a complex system, layers of simpler abstractions should be employed not only for ease of design but also for performance. Layers need not be costly: a well-defined and well-engineered infrastructure often performs better than a complex flat design. Current mobile computing technologies lack such layers of abstractions. This paper examines *group communication* as an infrastructure to support mobility of users.

Group communication systems offer primitives in support of *distributed groups of cooperating processes*. Their technologies are based on multicasting and membership service [Birman93][Amir91] [Cheriton85][Kaashoek91]. Though group communication has been studied mainly for fault-tolerance,

*This work was performed at Cornell University, and was supported under ARPA/ONR grant N00014-92-J-1866, and by a grant from Canon Inc. Kenjiro Cho is an employee of Canon, Inc., currently visiting the Dept. of Computer Science at Cornell University. Ken Birman is a Professor in the Dept. of Computer Science at Cornell University, and Chief Scientist of Isis Distributed Systems, a division of Stratus Computer, Inc. Authors' e-mail addresses are {kjc, ken}@cs.cornell.edu.

experience has demonstrated the approach can considerably simplify any distributed program which needs coordination among multiple processes.

Group communication is also intrinsically appealing for handling mobility. In group communication, multicast messages are sent to abstract groups and senders do not need to know the members of the group. Dynamic group membership management allows one to dynamically join or leave groups. Hence, one can leave a group, then move to another place, re-join the same group and continue working with the other members. In this sense, group communication already provides location independence and is able to adapt to a dynamically reconfiguring network topology. Reliable ordered multicasting guarantees that all group members will see the same set of messages in a guaranteed order, so that synchronizing multiple processes or coordinating cooperative actions is easy to achieve. Moreover, *fault-tolerance* (a system can continue to work in the presence of failures) is essential part of group communication design.

Prior work related to host mobility has focused on schemes to implement transparent mobile support in the network layer [Ioannidis91][Teraoka91]. Some of them address multicasting as an effective way to improve performance and reduce the cost [Ioannidis91][Keeton93][Acharya93]. Hand-off schemes and protocols from a higher level layer are presented in [Acharya94] by means of multicasting to mobile hosts and multicast-groups of mobile hosts. However, these approaches require complicated protocols for handshaking and buffering, and fault-tolerance is not often addressed.

On the other hand, the real power of group communication resides in its ability to support *consistent replicas*, and a replication mechanism can be used for handling mobility of users. This paper presents a simple scheme to support user mobility, which is based on an abstraction of *replicated servers* provided by group communication. In our scheme, hand-off is realized by switching a control point between replicated servers. Our scheme provides a simple abstraction of migration, eliminates complicated hand-off protocols and hand-off time, provides fault-tolerance and is implemented within the existing group communication mechanism.

To demonstrate our approach, we have developed a system called MobileChannel. Our goal is to identify and implement a suitable tool to support building mobile services and to integrate those services into the existing distributed environments based on group communication.

MobileChannel employs server replication to support migration of mobile clients. A migration is realized by switching a control point from one replicated server to another replicated server. A one-to-many channel is implemented between a client and servers. A client sees a single reliable server on the other end of the channel; hand-offs or server failures are completely masked to clients. On the other hand, servers see multicast messages from clients and hold consistent server replicas acting as a state machine. All servers observe a same set of incoming messages from clients but the only primary server actually sends messages to the client. A hand-off or a primary failure invokes a primary switch in which one of the other replicated servers takes over the channel on the fly. A hand-off is triggered by a single multicast message to the server group; no hand-off protocol is necessary between a client and servers. To achieve high performance, servers are replicated at a sliding-window level; the sliding-window itself is designed as a state machine. On top of the channel mechanism, two types of communication, a RPC style and a "diffusion" style, are supported. These styles are designed to tolerate clients to be unreachable for a short period, which often occurs in a wireless network. The approach scales well, provided that the underlying group communication infrastructure scales up.

MobileChannel is implemented as libraries. The server library runs on top of ISIS and the client library does not require that ISIS itself operates on the mobile computer. The client library is light-weight; the current implementation consumes only 130K bytes on Microsoft Windows[1]. MobileChannel also can be used for immobile and / or wired small computers as a light-weight fault-tolerant tool.

[1] All trademarks appearing in this paper are recognized registered trademarks of their respective companies.

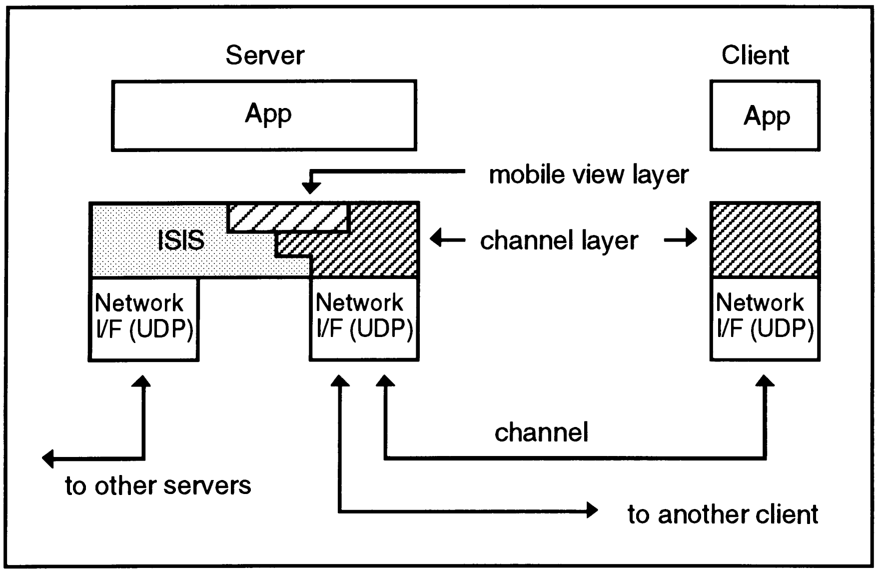


Figure 1.1 MobileChannel Programming Model

MobileChannel consists of two layers: the mobile view layer and the channel layer. Figure 1.1 shows the programming model of MobileChannel. The mobile view layer implements mobile client management on the server side. The channel layer implements a communication mechanism between a server and a client. Other than these layers, utilities are implemented to support programming. Figure 1.2 presents the run-time structure of replicated servers.

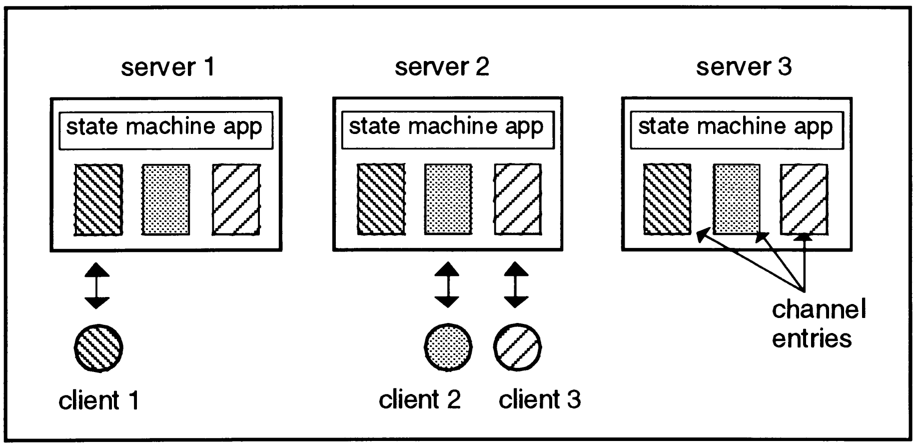


Figure 1.2 replicated server run-time structure

2 Design of the MobileChannel tool

In this section, we presents the design of MobileChannel. The MobileChannel tool is designed to provide continuous services to mobile clients and tolerate failures of servers. Services are available as long as one of the servers survives. For geographically-defined wireless cells, physical layout should provide fault-tolerance by means of overlapping cells or backup stations so that a client can reach the services as long as one of the reachable servers survives. MobileChannel provides a reliable but cheap communication means for unreliable mobile clients. A communication channel looks like an one-to-many channel carrying ISIS style messages, and tolerates transitory unreachability. On top of this channel, MobileChannel supports various utilities for communication. MobileChannel manages mobile clients and provides a paradigm in which programmers do not need to know the status of clients or server failures.

2.1 System Model

The system consists of two distinct sets of computers: static servers and mobile clients (Figure 2.1). A group of servers, running ISIS on a static local area network, provides services to mobile clients. Servers act as a proxy for a mobile client. Mobile clients talk to these servers through a wireless channel. The wireless network is organized by geographically-defined cells. Mobile clients can cross the cell boundaries maintaining communication connectivity. The static network and wireless network are connected by gateway servers, called Mobile Support Stations (MSS), that are the control points of mobile clients. Scalability of the system is discussed in section 2.7.

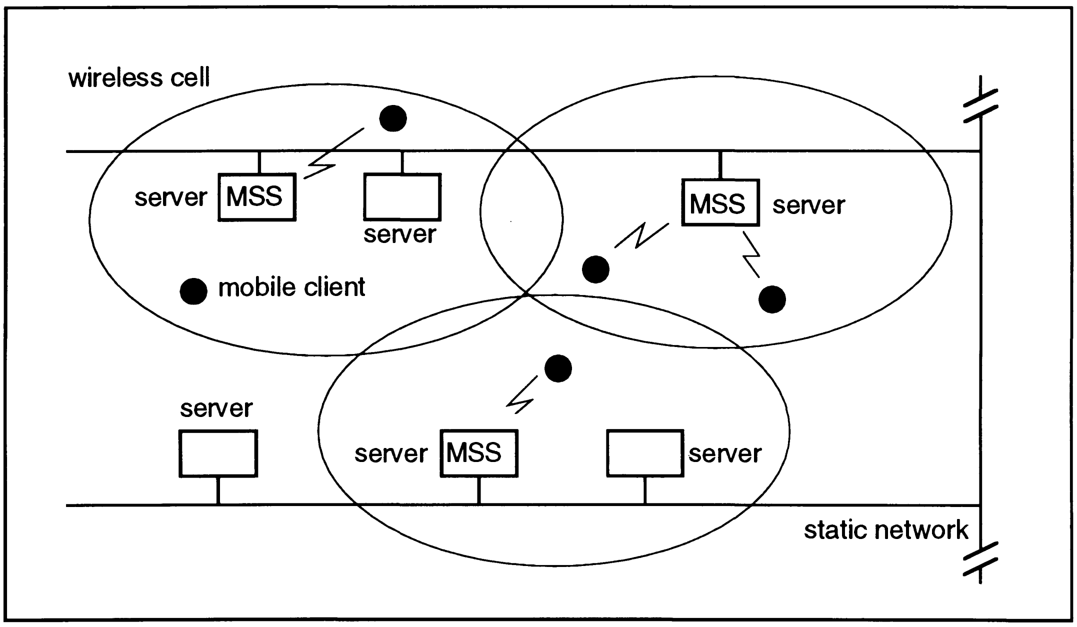


Figure 2.1 System Model

2.2 Requirements of Mobile Computing

Our group communication platform is the ISIS system. ISIS is a toolkit for building applications consisting of *cooperating processes* in a distributed system [Birman93]. Group management and group communication are two basic building blocks provided by ISIS. The most important communication primitive in ISIS is CBCAST. CBCAST implements reliable causal ordering delivery based on the potential causality relationship between events in the system. Another important communication primitive is ABCAST. ABCAST implements total ordering (every member observes events in the same order) in addition to the properties of CBCAST [Birman91].

ISIS has been widely used for applications which need to access consistent data scattered over a heterogeneous network. Many of those applications are also suitable for mobile terminals such as factory automation systems, stock trading systems, ticket reservation systems and store inventory systems.

Although ISIS supports a wide range of group communication mechanisms, ISIS does not normally support mobility of users. Several difficulties arise when running ISIS on mobile computers.

One practical impediment for running ISIS on a small computer is that ISIS is a fairly large system. ISIS provides a variety of mechanisms to support various types of distributed applications, but such broad functionality is too large and too heavy for a simple client program on a small computer.

A related problem is that ISIS runs on UNIX or more advanced operating systems but such operating systems are not designed for small mobile computers. Those operating systems have an implicit assumption in the design that the workstation will remain at a particular location, and hooked up to power and Ethernet. Design modifications to such operating systems for supporting small mobile computers need elaborate engineering efforts [Bender93]. Among these, power-saving is the most serious issue for battery-based computers. People use battery-based computers because they cannot plug into the power source while using computers, and thus, battery life is critical to their jobs. Since battery life will not improve much in the near future, power management is essential to increasing the working time with limited battery life. Less power consuming schemes should be employed when possible, and components or the entire machine should go into a sleep mode when it is idle. However, operating system design has been evolving in favor of high-performance, and as a result, modern operating system components tend to consume more power and have difficulty implementing power management because of elaborate designs, such as data caching, lazy execution, back ground processes, etc. Ironically, a less sophisticated operating system is easier to modify for power management and, in the current market, has an advantage with regard to battery life.

Another problem is that the frequently unreachable nature of mobile devices creates a communication model that contradicts some basic ISIS assumptions. ISIS processes should be responsive to messages because of the design of multicast ordering and failure detection. However, a mobile machine needs to go into a sleep mode to save power consumption and will not receive messages in the sleep mode. A mobile machine that must be continuously responsive can never go into the sleep mode. Moreover, a wireless channel tends to be temporarily unreachable because of obstacles or barriers as the user moves.

ISIS will consider these transitory communication gaps as a failure, then throw the unreachable client out of the group. When the client restores communication, it needs to re-join the group. These membership changes are costly in ISIS, since the design of ISIS assumes that a membership change is infrequent and it is acceptable that membership management is expensive.

In light of these considerations, to apply ISIS to mobile computing in practical settings, the system needs to provide mechanisms to satisfy a series of requirements: A mobile client needs a very light-weight tool. In order to take advantage of a power-saving mechanism, the tool should run on small operating systems and it is desirable that a client can start or stop communication at any time without any negotiation in order to go into the sleep mode. Mobile clients become frequently unreachable so that the system needs a cheap way to manage temporarily unreachable clients. The system has to provide some kind of reliable transport mechanism for unreliable clients and cannot expect any coordination among unreliable clients (as oppose to the "peer" design of ISIS).

2.3 Communication Model

The communication structure assumed by MobileChannel is that a mobile client talks to a server group on a static network. ISIS supports various types of group structures [Birman93] but MobileChannel focuses on two of them that are suitable for mobile applications. One is RPC communication; the other is "diffusion" communication. In the RPC case, a group of processes act as servers on behalf of a large set of clients. Clients interact with the servers in a request / reply style. The diffusion style is a type of client-server group in which the servers multicast messages to the full set of clients. Clients are passive and simply receive messages. A diffusion style arises in any application that broadcast information to a large number of sites. These two types of communication styles are common for many applications. For example, a trading system will use the diffusion communication to disseminate stock quotes and the RPC communication to make stock transactions.

2.4 State Machine Approach and Primary-Backup Approach

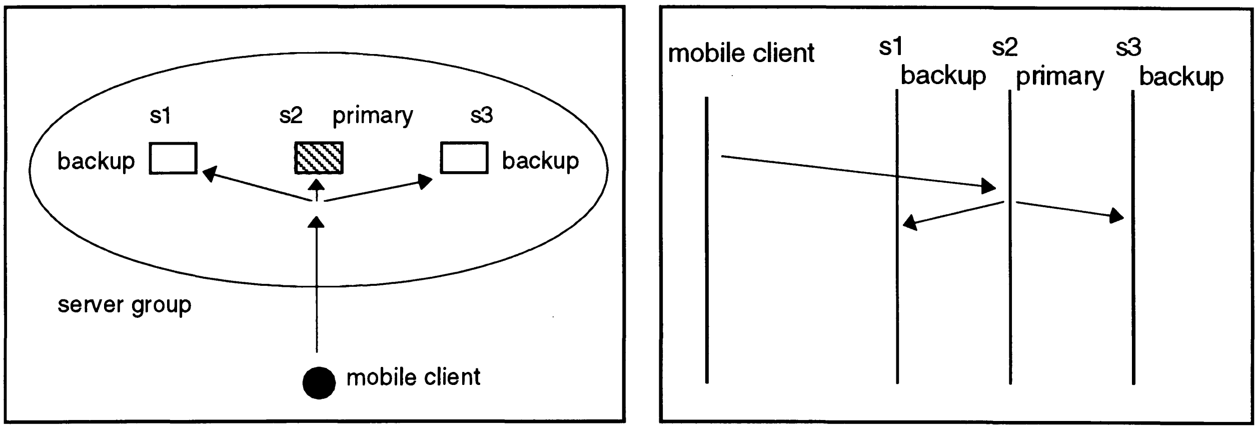
To implement a highly available service, the service should be replicated and distributed among multiple servers. Updates should be coordinated so that even when a server fails, the service remains available. Data replication has the additional benefit of performance, by placing a replica at sites where the service is needed. In general, replication is implemented in one of two ways. One is the *state machine* approach, which has no centralized control [Shneider85][Shneider93][Birman85]; and the other is the *primary-backup* approach, which has a centralized control [Budhiraja93].

In the state machine approach, a client makes a request by multicasting to all servers. All the servers change their states identically in lock step. A server failure is invisible to clients and does not introduce any response delay. In the primary-backup approach, a client makes a request by sending a message only to the primary server. If the primary server fails, then a *failover* occurs and one of the backup servers takes over. Requests can be lost at a primary failure and clients need to be informed about the failure to re-try the lost requests, which introduces a failover time. In general, the primary-backup approach is simpler and less costly, especially on the client side, than the state machine approach that requires reliable ordered multicasting.

For mobile clients, the cheaper mechanism of the primary-backup approach is more attractive. Clients do not need to implement multicasting that may be considerably more complicated than a point-to-point communication. In addition, mobile clients in a cellular wireless network can reach a limited number of servers within the communication range so that multicasting to all servers may not be possible.

On the other hand, ISIS supports the state machine approach and provides a sophisticated programming paradigm in which programs take actions according to events. The state machine approach is more flexible for several reasons: the mechanism is completely decentralized, a program can take actions only from its local state, clients can expect real-time responses even in the presence of failures, the system can tolerate arbitrary failures.

MobileChannel employs a combination of the two schemes: the primary-backup approach between a client and servers and the state machine approach among servers. Figure 2.2(a) shows the channel structure of MobileChannel. A client talks only to a primary but the primary forwards incoming messages to backups in order to provide an illusion that the client has a multicasting capability (Figure 2.2(b)). This allows programmers to utilize the state machine approach on the server side. From programmers' point of view, clients see a reliable channel and primary failures are unnoticeable. Server side programming is based on the state machine approach similar to the ISIS model and the primary-backup mechanism is hidden in MobileChannel.

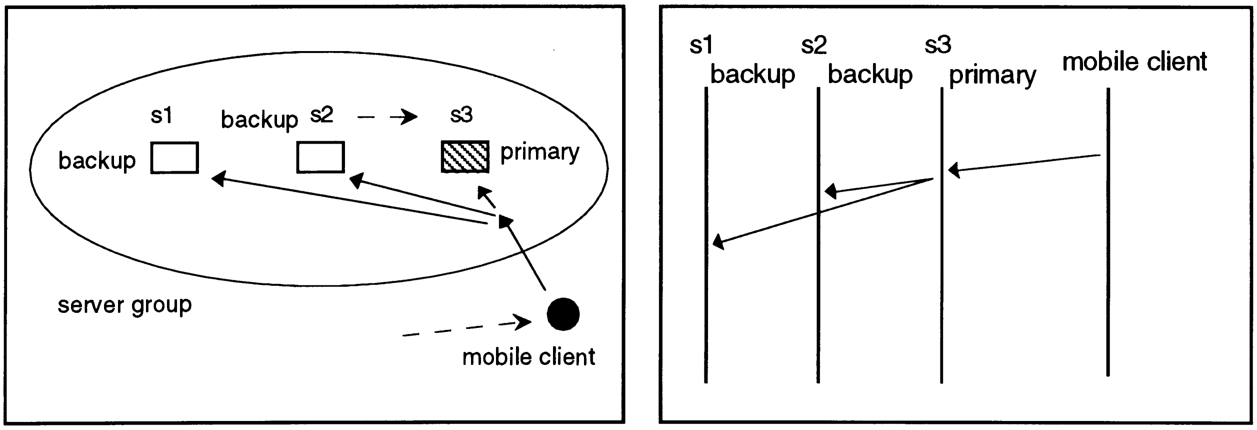


(a) client talks to primary (b) primary forwards msg to backups

Figure 2.2 primary-backup

2.5 Migration as an Intentional Primary Switch

The most unique feature of MobileChannel is that migrations of mobile clients are handled as an intentional primary switch. In the primary-backup approach, a failover usually occurs only at a primary failure. However, this same mechanism can be used for a migration control in order to move the primary to a nearby backup. Especially in a cellular wireless network, this mechanism allows the system to place the primary within a desired communication range. In short, the primary status can be handed over from one server to another to geographically follow the user. Figure 2.3 shows a primary switch to s3 from s2 that was primary in Figure 2.2. In the primary-backup approach, backups are designed to take over a primary at any time in case of a primary failure so that there is no difficulty to invoke a primary switch at any time. The only difference is that the primary switch is invoked not by a primary failure but by a migration of a mobile client.



(a) (b)

Figure 2.3 migration as an intentional primary switch

2.6 Sliding-Window Replication

Synchronization mechanism is necessary for any communication. When multiple processes need to synchronize, a handshake of two processes introduces a handshake time, and thus, buffering is necessary for messages from the other processes during this period. Handshaking and buffering are sources of the difficulties of hand-off protocols [Acharya94][Keeton93]. Such hand-off procedures require handshaking among three parties: a new server, an old server and a mobile client. However, our scheme does not need explicit handshaking at all but handshaking is automatically achieved by the underlying structures.

Handshake between servers can be acceded in the ISIS *virtual synchrony* model [Birman93], for the following reasons. Virtual synchrony is an illusion that every event happens synchronously in the system. Events such as receiving messages are synchronous in terms of logical time [Lamport78], though each process receives a same message at different physical time. Virtual synchrony can be used to synchronize multiple processes and release programmers from problems of handshaking and buffering.

MobileChannel converts messages from mobile clients, generated outside ISIS, to virtually synchronous ISIS events by forwarding messages from mobile clients to all servers. Thus, all servers observe the same set of incoming messages in the same order, which allows servers to take the state machine approach. When a hand-off of a mobile client is requested by multicasting to a server group, all servers can take appropriate actions without any handshaking. The need for a handshake between two servers can thus be eliminated by exploiting virtual synchrony.

Handshake between a mobile client and servers are confined to a low-level transport mechanism. Prior hand-off protocols assume a FIFO channel between a server and a client [Acharya94][Keeton93]. Although a FIFO channel is a useful abstraction for a static network, it does not work well for a hand-off procedure.

The problem of a FIFO channel is that a hand-off procedure needs to terminate the connection to the old server and then establish a new connection to a new server. These two procedures require handshaking between both ends and introduce delays and buffering. Furthermore, a channel takeover of a FIFO channel is not fault-tolerant. When a server fails, another server should take over the channel for fault-tolerance. Since there is no way in a FIFO channel to know the state of the other party , e.g., the last received message id, additional protocols between the client and the new server are necessary to restore the states. A server failure in the middle of a hand-off will need further complicated protocols.

However, if a lower level of transfer abstraction is used, such as packet level, the communication state can be replicated on a per packet basis. If a new server knows the last packet transferred or acknowledged, the new server can take over and continue communication without any handshaking. Moreover, protocols of this level already assume that packets may get lost or duplicated, so that the protocol is inherently robust to packet loss, duplicates or timing delays, which might be introduced by a takeover. Thus, no special protocol is necessary for a hand-off, and explicit handshaking between a mobile client and servers can be eliminated.

MobileChannel implements a sliding-window protocol for communication between a mobile client and servers, and replicates the sliding-window state on the server side. A sliding-window protocol is used to implement a FIFO channel and to control message flow in a communication channel. The protocol can make full use of network bandwidth if carefully tuned, which is important especially for wireless communication media because of their relatively low bandwidth. Replicating sliding-window states allows MobileChannel to achieve high performance as well as keeping the state machine model.

2.7 Scalability

Scalability is essential to a mobile computing infrastructure. The design of a mobile support system should have reasonable scalability.

A hierarchical group structure will be used to scale up MobileChannel. When a service needs to be available to hundreds of mobile users at tens of sites, it is desirable to replicate the service at each site but replicate the state of a client only at a small subset of sites. Such a group will be structured in the way that the group of the service, which consists of the whole sites, has subgroups; each subgroup corresponding to a mobile user. Each user subgroup consists of neighbor sites of the current location of the user. Figure 2.4(a) shows an example of site layout; each box represents a site and the whole sites constitute a service group. The shadowed sites form a subgroup around the black primary site where a user currently resides.

Dynamic group membership can support user mobility. The idea is that a subgroup moves along a user as a flock. As a user travels, new sites, located in the direction the user heads for, join the subgroup. Then member sites located behind the user leave the group in order to keep the group size. In Figure 3.4(b), the subgroup moves from 3.4(a) to follow the user. When a disconnected user establishes a new connection at a remote site from the previous location, a new subgroup sprouts at the new location, and then, the old subgroup vanishes.

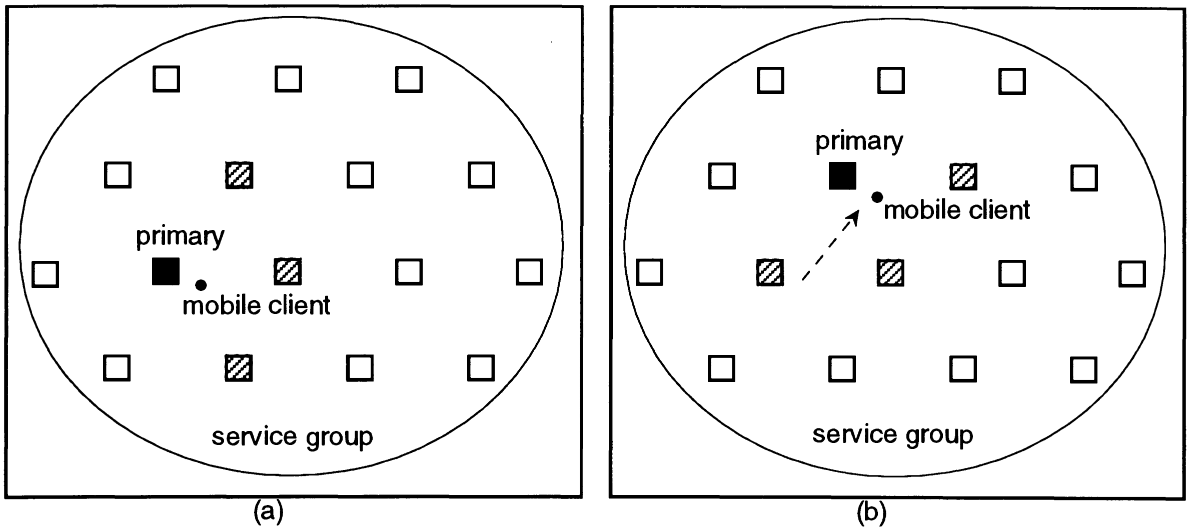


Figure 2.4 subgroup follows a user as a flock

Moreover, sites which do not belong to any subgroup can leave the service group to reduce the service group size. Those sites will re-join the service group when a user come close to them. By holding only member sites of subgroups, the entire service group changes the shape to adapt to movements of users.

To scale the system up to thousands or millions of sites, further hierarchy would be necessary. Though the current group communication infrastructures do not scale up to this level, research effort into this question is underway [Glade92].

3 Implementation

Programming model of MobileChannel is derived from ISIS and UNIX. The callback style of MobileChannel is similar to ISIS but does not require a thread system. The ISIS message library [Birman87] is used for marshalling / unmarshalling messages. The channel connection management uses a semantics similar to the UNIX socket / TCP model [Leffler89].

3.1 Mobile View Layer

The mobile view layer manages mobile clients. Two types of migration are defined: one is reconnection and the other is hand-off. A reconnection is a disconnected migration and occurs when a user establishes a new connection. A hand-off is an on-line migration and usually occurs when a user crosses the cell boundary. A hand-off requires a dynamic channel switch.

A primary failure is handled as a migration. A primary failure can be a reconnection or a hand-off depending on the channel state at a primary failure. If the channel state is not active, the new primary takes over the primary status but does nothing. When the client reconnects to one of the servers later, an ordinary reconnection occurs. On the other hand, if the channel is active at the primary failure, a hand-off takes place and the new primary takes over the channel.

Migration can be triggered by requesting a migration to a current primary; the easiest way is multicasting a request to the server group, another way is to request from a mobile client. Upon receiving a migration request, the primary multicasts a primary switch command to the group. The primary switch command uses ABCAST to guarantee that all servers observe the command in the same order with regard to other messages and a mobile client has a single primary at one (virtual synchrony) time. Each server keeps a consistent (identical) state list of mobile clients called a mobile view. The mobile client state includes the current primary information and the channel status. When a client establishes a connection for the first time, a new entry for the client is created and the channel is attached to this entry. When a client disconnects from the system, the channel is closed and detached from the entry but the entry remains in the list even after the client's disconnection to keep track of the client's activity. When a reconnection or a hand-off occurs, the corresponding entry is updated.

It is important that a migration is informed by a single ABCAST message to all servers. This means that a migration looks like an atomic (indivisible) action to the observers. This atomicity makes it easier to write a fault-tolerant program since servers do not need to keep intermediate states and restore them for a failure recovery.

The migration decision mechanism, which decides which server should take over which client, is outside the system. A decision is made by location information of a mobile client, usually by detecting the signal level. Decision mechanism varies from system to system. For example, decisions can be made by servers, by mobile clients, or by negotiation between a server and a client. When a backup server is instructed to be a new primary, it invokes a channel switch to take over the channel. The mechanism of channel switch is discussed in the next subsection.

3.2 Channel Layer

The channel layer provides an abstraction of a communication channel which acts like a reliable one-to-many FIFO channel. The channel layer assumes some kind of datagram service underneath and currently uses UDP.

The channel mechanism is based on a point-to-point sliding-window protocol. One-to-many connection is realized by replicating the sliding-window state of the primary to the backups. When the primary receive a message from a mobile client, the primary forwards this incoming message to the backups by ABCAST or CBCAST. Thus, all servers see the same set of incoming messages in the same order and the input window state of each server is kept identical. According to an incoming message, all servers take identical actions as a state machine. Hence, the output window state of each server is also kept identical. When servers send messages to a mobile client, only the primary actually sends out messages. Each channel on the servers has a flag which shows if the server is primary or not. The output routine checks this flag and if it is not set, the output routine does nothing. The mobile view layer is responsible to maintain at most one primary per client at one time. Outgoing messages in the output buffer of each server are discarded when the corresponding acknowledgment is forwarded from the primary. Those delayed acknowledgments are piggybacked in incoming messages.

To keep the output windows consistent, it is essential that all servers take identical actions. For a same set of incoming messages, a deterministic action suffices. Two other event sources should be mentioned which may affect deterministic actions. One is a timer. A timer produces events local to the machine and the sliding-window requires timeouts for delayed acknowledgments or retransmission. However, sending acknowledgments or retransmission does not change the window state. Hence, timer events of the sliding-window mechanism do not affect the determinism. The other is scheduling. Even when a server's action blocks, ordering of actions should not be changed at all servers. The ISIS scheduler employs a strict FIFO order scheduling, and unblocking is triggered only by ordered events. As a result, blocked actions resume in the same order at all servers and the scheduling does not affect the determinism either.

Client programs can specify either ABCAST or CBCAST to forward a message from the primary to the backups. When a message changes some state shared with other mobile clients, ABCAST should be used in order to guarantee the total delivery order.

When a mobile client becomes unreachable, the channel tries retransmission with exponential back-off then informs the upper layer of the channel state change, but never drops the connection unless the client establishes a new connection. Send operations will block when the output buffer is full. Servers have a fairly large output buffer to tolerate transitory unreachability for a diffusion style communication. The details of the mechanism is discussed in the next subsection.

The connection management is based on the TCP model that is defined as a finite state machine [Postel81]. The connection establishment and termination mechanisms are the same as TCP. To support a channel switch between servers, a special flag "SWITCH" is defined in a packet header. When a primary receive a migration request, it clears the primary flag and multicasts a primary switch command specifying a new primary. Upon receiving this command, the new primary sets the primary flag in the channel entry and sends a SWITCH segment to the client specifying the port number of the new primary. Upon receiving this SWITCH segment, the client updates the address and port number of the peer. The acknowledgment of the SWITCH segment is processed in the same manner as ordinary packets. Figure 3.1 presents the primary switch procedure.

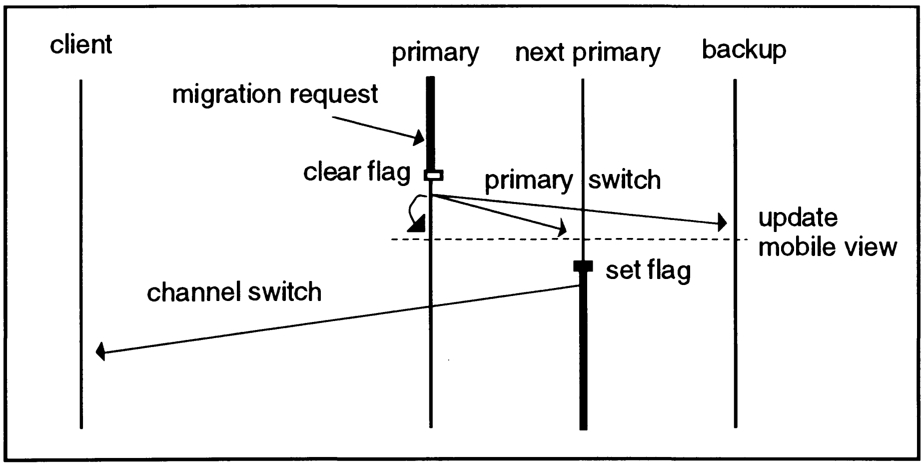


Figure 3.1 primary switch

In a channel switch procedure, the sliding-window states, including sequence numbers and buffered messages, of both ends are not initialized and remained intact. Thus, both ends can continue communication from the same state just before the channel switch. The only difference is the primary flag on the previous and current primary servers, and the peer address and port number on the client.

There is a small timing gap between clearing the primary flag by the current primary and setting the flag by the new primary, where no server has a primary flag set. Packet loss or duplicates may occur during this period, but the quick switching mechanism narrows the possibility and those errors are suppressed by the sliding-window mechanism. Note that sending a primary switch command from a current primary to a new primary creates a causal chain; message forwarding by the primary is thus FIFO ordered by CBCAST even in the presence of primary switch.

One might concern about the increase of message traffic caused by the sliding-window replication but the increase of message traffic is surprisingly small. The replication scheme forwards only incoming messages. Most of incoming messages will be for data updates, since most read operations will be a one-way diffusion style from a server to a client. Even without the sliding-window replication, it is necessary to multicast updates. Thus, increase of message traffic comes basically from ack packets. The mechanism of delaying and piggybacking acknowledgments in the sliding-window protocol works well under heavy traffic. Ack packets are much fewer than data packets because of the delayed ack mechanism. In addition, ack packets can be forwarded by cheaper CBCAST because the window state is local to the client. As a whole, the increase of message traffic is negligible.

Another concern would be state transfer at joining a group. When a new process joins a group, the entire sliding-window state including buffered messages should be transferred by the ISIS state transfer mechanism. Nevertheless, the sliding-window normally holds zero or less than a few messages, and ISIS tries to pack multiple message into one for efficiency.

One restriction with regard to the state transfer of the sliding-window is that no thread should be waiting to write to the channel at a state transfer. The state transfer sends out the sliding-window state but it cannot send the states of blocked threads. To avoid this situation, MobileChannel inhibits joining a group while a write is blocked.

For a security reason, a 32-bit connection key is created at a connection establishment. To take over the channel, this key must be replicated to backup servers and a backup needs to send this connection key in the SWITCH segment.

3.3 Utilities

The following utilities are implemented to support programming.

3.3.1 RPC style and Diffusion style Communication

Although the channel layer provides a reliable channel, MobileChannel also provides a simplified interface. MobileChannel directly supports the RPC style and the diffusion style of communication. In the RPC style, a client blocks until the corresponding reply comes back. The server callback provides a way to send a reply to the requester. In the diffusion style communication, a single call on the server side sends a message to all clients. Most applications will use a mixture of the above two styles.

The diffusion style needs a special data flow control. Since messages are usually generated unrelated to the states of clients, data flow does not stop even when a client is unreachable so that a point-to-point flow control scheme does not work well.

The two styles behave differently when the output buffer is full. In the MobileChannel sliding-window mechanism, when the output buffer is full, send blocks until a slot becomes available. However, in a diffusion communication, it is possible that blocked send operations pile up during the client is temporarily unreachable. Though applications can specify the output buffer size according to their demand, dynamic message traffic and unreachable duration are unpredictable. In addition, a blocked send prevents a new server from joining the group as discussed in Section 3.2. To avoid this situation, the diffusion mechanism implements the following scheme: when the output buffer is full, a diffusion send just discards the message and marks the message overflow, and when a slot becomes available, the system notifies the application about the overflow. It is applications' responsibility to take an appropriate action. For most applications, reinitializing the client's state by sending the current state saved at the server will be enough. Even when diffusion messages are being discarded, however, a RPC reply is never lost. An alternative scheme is to implement an infinite backlog as ISIS News does [Birman87]. This scheme, however, may cause an unfavorable flood of messages when the communication is restored.

The design of MobileChannel assumes applications for operational clients. MobileChannel does not directly support the delivery guarantee: everyone receives a message even when it is disconnected for a long time. If an application requires such a guarantee, the best way is to use ISIS News within servers. ISIS News implements publish / subscribe schemes by means of virtually infinite backlogs.

3.3.2 Primary Actions

In MobileChannel, it is common that all servers take the same action corresponding to an event. However, it is sometimes necessary that only a primary takes an action, especially for hierarchical groups. To make such an action fault-tolerant is not as easy as it seems at first. When the action has some side effect, such as sending a message, the action should be executed exactly-once. If the primary fails in the middle of the action, the backups could not tell if the primary fails before or after sending the message. The Primary Action mechanism supports these programming models by using the ISIS coordinator-cohort mechanism. The coordinator-cohort scheme supports a computation in which an elected coordinator conducts the computation in a fault-tolerant manner.

3.3.3 Mobile Monitor

The Mobile Monitor mechanism provides a way to monitor state changes of servers and clients. A client can be notified when a server joins or leaves the group, or other clients change their states. Three client states are defined: connected, disconnected and temporarily unreachable.

4 Current Status

4.1 System Environment

The current system environment is as follows: The server machines are SUN SparcStations. The client machines are subnote PCs (40MHz 486DX2 and 8M RAM) running Microsoft Windows3.1, a desktop PC running WindowsNT3.1 and SparcStaions running SunOS4.1.1. The subnote PCs are equipped with NCR WaveLAN PCMCIA wireless Ethernet cards (2Mbps) and can talk to the servers through a bridge. The other client machines and servers are hooked up to a 10Mbps Ethernet. We do not have a device to locate mobile clients nor a wireless device capable of cell hand-off so that migration is triggered by an emulator and the wireless clients are physically in a single cell.

The MobileChannel server library runs on top of ISIS. ISIS runs on most UNIX based platforms and WindowsNT. The client library runs on the X Window System / UNIX or Microsoft Windows. The interface of Microsoft Windows uses the WIN32s API and Winsock v1.1 so that it runs on both WindowsNT3.1 and Windows3.1. The MobileChannel client library consumes 70K bytes on Microsoft Windows and the ISIS message library consumes another 60K bytes.

We have built demo applications with a graphical user-interface. The "reserve" application implements a train ticket reservation system and uses a diffusion style communication to monitor the reservation status and a RPC style communication to make reservations. The "grid" application shows the throughput of MobileChannel.

4.2 Performance

The following throughputs of the prototype were measured with Sun SparcStation1 workstations on an ordinarily loaded Ethernet. The data in parentheses were measured with the subnote PC clients via the wireless network and about twice slower than the Sparcstations due to the lower bandwidth, the presence of the bridge, the difference of byte order and the slower CPU. Table 1 presents performance of the diffusion style in which one-way messages are continuously sent from a server to a client. Table 2 presents performance of the RPC style in which a request message from a client is forwarded to replicated servers by ABCAST and then the client waits for a null reply from the primary. The cost of the RPC style to replicated servers is governed by the cost of ISIS multicast that grows roughly linearly with the size of a group [Birman91]. The throughputs include creation / deletion of ISIS style messages that support marshalling / unmarshalling complex data structures.

A hand-off time is typically less than 20 ms in MobileChannel. A hand-off time can be defined as a time between receiving a migration request by a primary and switching the channel to a new primary by the client. One ISIS ABCAST message for a primary switch command and one channel message for a channel switch segment are required. ABCAST costs less than 10 ms for four servers and a channel switch segment costs less than 5 ms, provided that no message loss occurs.

user data size (bytes)	4	64	1024
throughput (messages / sec)	429 (220)	408 (204)	300 (127)

Table 1 Diffusion Throughput

user data size (bytes)	4	64	1024
1 server (rpcs / sec)	123 (56)	122 (58)	105 (40)
2 replicated servers (rpcs / sec)	85 (54)	95 (53)	77 (37)
4 replicated servers (rpcs / sec)	75 (45)	73 (41)	60 (31)

Table 2 RPC Throughput

5 Conclusion

We presented a mobile support facility, MobileChannel, for the ISIS system. MobileChannel employs a unique scheme of a combination of state replication and intentional primary switch to support mobility of users. This scheme practically eliminates hand-off protocols and hand-off time yet provides fault-tolerance. The scheme fits well into the current static network environment and programming model, and is easily built on the current technologies.

As computer and network technologies are rapidly evolving, expanding and diversifying, the idea of cooperating process groups is increasing its importance. MobileChannel is an example how group communication can be used in support of small hand-held devices. Replication is employed to provide a simple abstraction of migration as switching a control point between consistent replicas. Simple abstractions, provided by a well-engineered infrastructure, will be essential to attack challenging new fields. Our experience suggests that server replication will be a simple but powerful abstraction in mobile computing for development of robust and sophisticated applications.

Acknowledgments

The authors gratefully thank the members of the ISIS group at Cornell University for their helpful comments and suggestions.

References

[Acharya93] Arup Acharya and B. R. Badrinath. Delivering Multicast Messages in Networks with Mobile Hosts. *Proceedings of 13th International Conference on Distributed Computing Systems.* IEEE, May 1993, 292-299.

[Acharya94] Arup Acharya and B. R. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. *Technical Report*, Department of Computer Science, Rutgers University, 1994.

[Amir91] Yair Amir, Danny Dolev, Shlomo Kramer and Dalia Malki. Transis: A Communication Sub-System for High Availability. *Technical Report* CS91-13, The Hebrew University of Jerusalem, Israel, November, 1991.

[Bender93] Michael Bender, Alexander Davidson, Clark Dong, Steven Drach, Anthony Glenning, Karl Jacob, Jack Jia, James Kempf, Nachiappan Periakaruppan, Gale Snow and Becky Wong. UNIX for Nomads: Making UNIX support mobile computing. *Proceedings of USENIX Symposium on Mobile & Location-Independent Computing.* USENIX, August, 1993, 53-68.

[Birman85] Kenneth P. Birman. Replication and fault tolerance in the ISIS system. *Proceedings of the Tenth Symposium on Operating Systems Principles,* Orcas Island, WA, 1985, 79-86.

[Birman87] Kenneth P. Birman, Thomas Joseph and Frank Schmuck. ISIS - A Distributed Programming Environment, Version 2.1 - User's Guide and Reference Manual. July, 1987.

[Birman91] Kenneth P. Birman, Andre Schiper and Pat Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems,* Vol. 9, No 3, August, 1991, 272-314.

[Birman93] Kenneth P. Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM,* December 1993.

[Budhiraja93] Navin Budhiraja, Keith Marzullo, Fred B. Schneider and Sam Toueg. The Primary-Backup Approach. Distributed Systems second edition. Addison-Wesley, 1993, 199-216.

[Cheriton85] David R. Cheriton and Willy Zwaenepoel. Distributed Process Groups in the V Kernel. *ACM Transactions on Computer Systems,* Vol. 3, No. 2, ACM, May 1985, 77-107.

[Glade92] Bradford B. Glade, Kenneth P. Birman, Robert C. B. Cooper and Robbert van Renesse. Light-Weight Process Groups. *Proceedings of the OpenForum '92 Technical Conference,* November 1992, 323-336.

[Ioannidis91] John Ioannidis, Dan Duchamp and Gerheld Q. Marguire Jr. IP-based Protocols for Mobile Internetworking. *Proceedings of SIGCOMM'91.* ACM, September, 1991, 235-245.

[Kaashoek91] M. Frans Kaashoek and Andrew S. Tanenbaum. Group Communication in the Amoeba Distributed Operating System. *Proceedings of the IEEE International Conference on Distributed Computing.* IEEE, May 1991, 222-230.

[Keeton93] Kimberly Keeton, Bruce A. Mah, Srinivasan Seshan, Randy H. Katz and Domenico Ferrari. Providing Connection-Oriented Network Services to Mobile Hosts. *Proceedings of USENIX Symposium on Mobile & Location-Independent Computing.* USENIX, August, 1993, 83-102.

[Lamport78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM,* 21(7):558-565, July 1978.

[Leffler89] Samuel J. Leffler, Marshall K. McKusick, Michael J. Karels and John S. Quarterman. The Design and Implementation of the 4.3BSD UNIX Operating Systems. Addison-Wesley, 1989.

[Postel81] J. B. Postel. Transmission Control Protocol. RFC793. SRI Network Information Center, Menlo Park, CA, September, 1981.

[Shneider85] Fred B. Schneider. Paradigms for distributed programs, in Distributed Systems - Methods and Tools for Specification, Lecture Notes in Computer Science, Vol 190, Springer-Verlag, New York, NY, 1985, 343-430.

[Schneider93] Fred B. Schneider. Replication Management using the State-Machine Approach. Distributed Systems second edition. Addison-Wesley, 1993, 169-197.

[Teraoka91] F. Teraoka, Y. Yokote and M. Tokoro. A Network Architecture Providing Host Migration Transparency. *Proceedings of SIGCOMM'91.* ACM, September, 1991, 209-220.

Appendix: Programming Interface

The following sample code shows a MobileChannel database service for maintaining a mapping from names (ascii string) to salaries. The code is derived from the sample in [Birman93].

```
/*
 * sample client of the simple database service
 */
#include "MobileChannel.h"
#define UPDATE    1    /* entry number for update */
#define QUERY     2    /* entry number for query */
#define SIMPLE_DB 1    /* application type */


void main(int argc, char **argv)
{
    McStruct *mcp;    /* pointer to a mobile channel structure */

    /* first, create a channel, then connect to a server */
    mcp = McClntCreate();
    McClntConnect(mcp, my_name, SIMPLE_DB, server_name, server_port);

    ...       /* do some input process to call update or query */

    /* disconnect from the server, then close the channel */
    McClntDisconnect(mcp);
}

/* send update using abcast */
void update(McStruct *mcp, char *name, int salary)
{
    McClntAbcast(mcp, UPDATE, "%s, %d", name, salary);
}

/* ask for a salary value using rpc */
void query(McStruct *mcp, char *name)
{
    int salary;

    McClntRPC(mcp, QUERY, "%s", name, "%d", &salary);
    return (salary);
}
```

```
/*
 * sample server of the simple database service
 */
#include "MobileChannel.h"
#include "mss.h"            /* for mobile-support-station */
#define UPDATE    1    /* entry number for update */
#define QUERY     2    /* entry number for query */

void main(int argc, char **argv)
{
    isis_remote_init(NULL, 0, 0, 0);    /* initialize isis */

    /* set callbacks for channel entries */
    McAddCallback(UPDATE, update);
    McAddCallback(QUERY, query);

   /* set callbacks for state transfer */
    mss_add_callback(MSSEC_SEND_XFER, send_state);
    mss_add_callback(MSSEC_RECV_XFER, recv_state);

    /* mss_maintask is a built-in mobile-support-station task that
       joins a mss_group and manages mobile clients and channels */
    isis_main_loop(mss_maintask, NULL);
    /* never returns */
}

/* callback for entry UPDATE */
void update(McStruct *mcp, message *mp)
{
    char name[32];
    int salary;

    msg_get(mp, "%s,%d", name, &salary);
    set_salary(name, salary);              /* update the database locally */
}

/* callback for entry QUERY */
void query(McStruct *mcp, message *mp)
{
    char name[32];
    int salary;

    msg_get(mp, "%s", name);
    salary = get_salary(name);             /* get salary value by name */
    McSvrRPCReply(mcp, "%d", salary);     /* reply to the client */
}


/* send out database state */
void send_state(int locator)
{
    struct sdb_entry *sp;
    for (sp = head(sdb); sp != tail(sdb); sp = sp->s_next)
        xfer_out("%s, %d", sp->s_name, sp->s_salary);
}

/* receive state at pg_join */
void recv_state(int locator, message *mp)
{
    update(NULL, mp);
}
```