MCE: An Integrated Mobile Computing Environment and Simulation Testbed*

Ramki Rajagopalan†
Bell Northern Research
2201 Lakeside Blvd
Richardson, TX 75082-4399
ramki@bnr.ca

Sridhar Alagar and S. Venkatesan
Computer Science Program, EC 31
The University of Texas at Dallas
Richardson, TX 75083-0688
{sridhar,venky}@utdallas.edu

Abstract

Integrated Mobile Computing Environment and simulation testbed (MCE) is a software platform to design, develop, and test algorithms and applications for a network of mobile and static computers. MCE is designed with the idea of providing users with a transparent system so that an application may be tested on the simulation testbed and then used directly on the network without any source code changes. MCE provides default functionalities like routing, location tracking, and handoff along with service access points to add or change services resulting in a unique integrated environment for development and testing. MCE may also be used as a platform to evaluate and compare various mobile network algorithms.

1 Introduction

MCE is a software platform for a network of mobile and static computers on which various protocols, algorithms and applications can be built, tested, evaluated and supported. When mobile systems have a large number of users, it is not practical to test applications with real mobile systems. Conventional simulators like event driven simulators suffer from the problems that they are not close to reality and, when the number of mobile users is large, the time taken to test an application is enormous. Hence, we also provide facilities within MCE so that it can be used as a simulation testbed to test protocols developed for a mobile network.

MCE simulates a mobile host by a process, and mobility is accomplished by checkpointing and process migration. In this aspect, MCE differs from other platforms such as the one described in [5]. This approach also provides for a transparent platform where the application developer need not make any changes when moving from the simulation testbed to the actual mobile environment.

MCE provides features such as routing, location tracking and multicasting as default features. We have taken a modular design approach so that any communication protocol [3, 4, 6, 7] may be integrated into MCE and evaluated. Currently, we have our initial version of MCE simulation testbed running in our lab. The design and implementation details are described in Section 7. The current status report is given in Section 8. We are planning on setting up an *ftp* site for access to MCE information, and these details are given in Section 9.

2 Mobile Computing Systems

A mobile computing system consists of a network of *static hosts* and *mobile hosts*. A *static host* has a fixed geographical location in the network and a fixed network address. A *mobile host* can move and has no fixed location. However, every *mobile host* has a unique network address. All the *static hosts* are part of a wired network that forms the backbone of the mobile computing network. Some of the *static hosts* may be designated as *Mobile Support Stations* (MSS). An MSS has the necessary infrastructures to support *mobile hosts* and related functionalities. Each MSS supports *mobile hosts* within a certain geographical area called the *cell*. A *mobile host* may belong to at most one cell at any time.

Communication among MSSs is supported by using the backbone network. We assume that a *logical* communication channel exists between every pair of MSSs. Communications between an MSS and *mo-*

*This research was supported in part by NSF under Grant No. CCR-9110177 and by the Texas Advanced Technology Program under Grant No. 9741-036.

†This work is being done as part of author's graduate work at The University of Texas at Dallas

bile hosts in its *cell* are over a wireless medium. All communications among the *mobile hosts* are through their MSSs. Each *mobile host* has an MSS designated as the *home MSS* where all relevant data for the *mobile host*, such as the current location, subscribed services etc., are maintained.

A *mobile host* may migrate from one cell to another at any time. In order to keep track of the number and identity of *mobile hosts* in its *cell*, an MSS periodically broadcasts an MSS specific *beacon* [6]. When a *mobile host* h receives the *beacon* from MSS s_{new}, h verifies the MSS address; if the MSS has changed from what h remembers, say s_{prev}, then h sends a response to s_{new} that includes the address of s_{prev}. When s_{new} receives this message, a *handoff* procedure is executed between s_{new} and s_{prev}.

3 MCE Architecture

MCE consists of several networked UNIX machines. We consider every *static* machine as a mobile support station and each MSS will have a server process, MSS Daemon (**mssd**), running in the background on the MSS. These daemon processes will control all communications that occur within the mobile network. We use *sockets* for establishing and controlling various communication interfaces between *static* hosts. In our integrated platform, we provide software to support real mobile hosts along with features to support mobile host simulation.

In case of simulation, we can develop and test applications for a mobile environment without the presence of actual mobile hosts and wireless interfaces. Each mobile host h in the cell of an MSS s_1 is simulated by a process p_h running on the MSS s_1. Migration of MH h from an MSS s_1 to another MSS s_2 is simulated by checkpointing and process migration of p_h from s_1 to s_2 followed by a handoff. The wireless link between an MSS and an MH is simulated using sockets. The messages sent along the sockets are delayed to match the bandwidth capabilities of wireless links. The mobility of MHs will be controlled by a mobility simulator (**msim**) process running on MSS machines. The correspondence between some of the entities in a mobile network and in the simulation is shown in Table 1. The **msim**s communicate among themselves using sockets. The **msim**s communicate with the MH processes (to checkpoint and migrate) using signals. A pictorial representation of the architecture is shown in Figure 1. Although, MCE is defined on the basis of networked UNIX machines, we have taken sufficient care to ensure that MCE could be ported easily to different hardware plaforms and operating systems. This has been done to make it easy to support MCE applications on different PDAs, notebooks etc., that become available in the market.

Mobile Network	Simulation
Mobile Hosts	Processes
Wireless Links	Sockets
Mobility	Process Migration

Table 1: Correspondence between Mobile Network and Simulation in MCE

In order to support actual mobile network application development, MCE provides two libraries with identical Application Programming Interfaces (**API**), namely, **MCE Simulation Library** and **MCE Mobile Environment Library**. Application developers may use the following steps to develop, test and deploy their mobile applications within MCE.

- Develop the application using the APIs provided by MCE.
- Link the application with **MCE Simulation Library** and get an executable for the simulation testbed.
- Test the application on the simulation testbed.
- Once the application has been debugged, create a new executable by linking the application to the **MCE Mobile Environment Library**.
- The application is now ready for use on a mobile host.

MCE application developers need not be aware of any OS/hardware dependencies within MCE as the MCE API resides on top of the operating system. Thus, MCE application developers need not be concerned with any machine and/or operating system dependencies within MCE. MCE does not require any modifications to the operating system. Moreover, MCE-specific processes such as **mssd** may be run as user level processes. This makes it easy to install MCE and develop and test MCE-based applications. This view of MCE is illustrated in Figure 2.

In the initial version of MCE, we have provided only the simulation and testbed features and hence **MCE Simulation Library** is the only library available at this time. We provide details about the simulation testbed environment in the following sections.

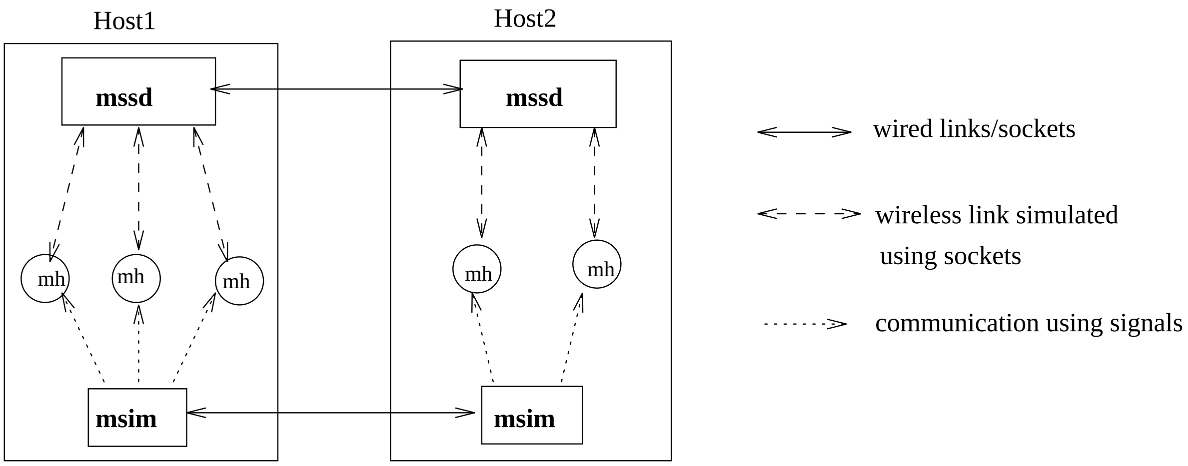


Figure 1: The Simulation Testbed

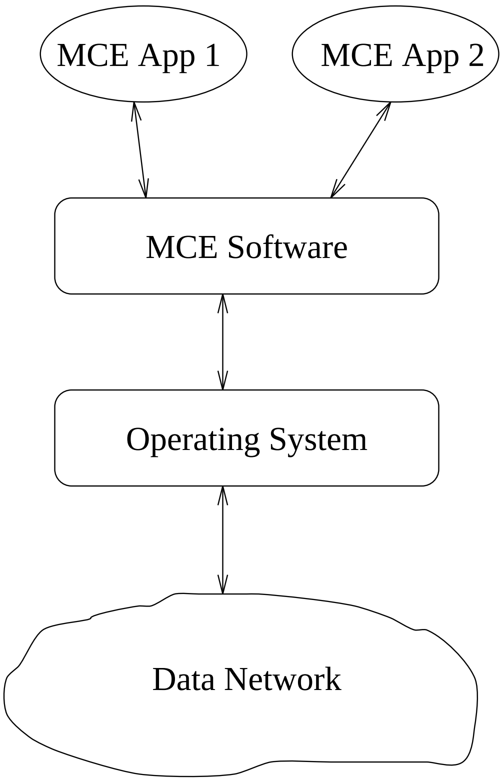


Figure 2: MCE Software on a System

4 Mobility Management

We use checkpointing to simulate mobile host migration from one MSS to another. Complexities introduced by checkpointing and process migration can be avoided by letting a single host simulate multiple MSSs. We chose not to take that approach for several reasons, some of which are:

- One of our design goals is to minimize development that is not common to both the simulation testbed and real mobile network within MCE.
- If mobile host processes were the only elements that were different from actual mobile computing network, then the performance of the mobile applications and algorithms can be measured effectively. The only difference we need to account for would be the mobile host processing and the wireless interface.
- By using the backbone network as part of the simulation, the static network part of MCE gets tested completely. Moreover, this lets us use an MSS as a simulation node as well as a real MSS supporting mobile hosts. This facility may help in testing and evaluating performance of either end involved in a protocol independently.

In the simulation testbed, when a mobile host enters an MSS, the corresponding **msim** process will decide when the mobile host will leave that MSS.

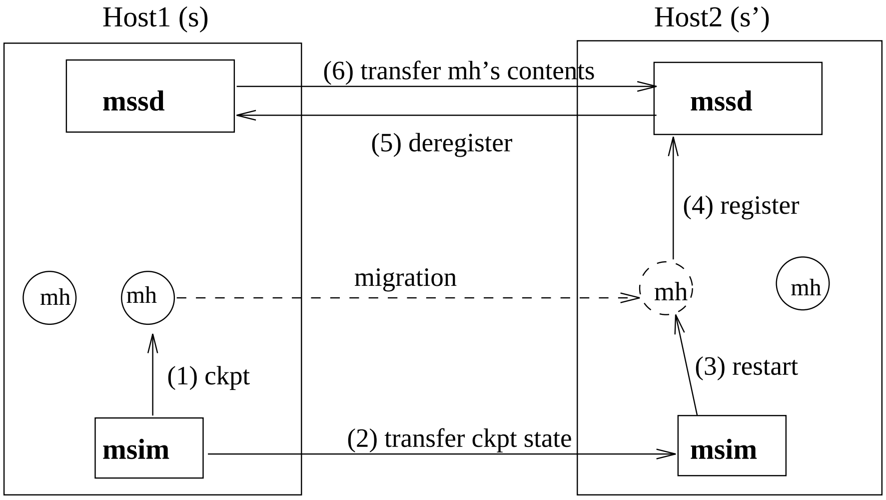


Figure 3: Mobility and Handoff

We use a simple scheme that assumes the time of stay in an MSS to be random (uniform distribution) and choose a destination. More sophisticated schemes may be designed based on the velocity of a mobile host and direction of its movement. For such schemes to work, **msim** should be aware of the topology and dimensions of the cells. We also provide an option to let users specify mobility patterns.

Let MH h move from the cell of MSS s to the cell of MSS s'. Let m and m', respectively, be the **msim** processes in the machines s and s'. The steps involved in the migration of MH h from s to s' are as follows.

- **msim** m sends a signal to mobile host h to checkpoint h's state ((1) in Figure 3).
- **msim** m then transfers the checkpointed state of mobile host h to **msim** m' in the MSS s' ((2) in Figure 3).
- **msim** m' then restarts the MH h from its checkpointed state ((3) in Figure 3). As part of restart, socket connections are released with the **mssd** of s and are established with the **mssd** of s'.

4.1 Checkpointing

For checkpointing, we are using an implementation available in the public domain. To checkpoint the state of an MH, **msim** sends a signal to MH. The MH executes a signal handler to checkpoint its state. First, the data region of the process is saved. Next a *setjmp()* call is made to save the register contents into a file. The **msim** process at the destination MSS uses the checkpointed data to restart the process at the destination. Currently, we do not support file operations.

The files associated with the MH are not moved along with the MH. At present, we assume that the files in a remote machine are accessible using NFS automount (which is the case in our testbed environment). This simple option is chosen to reduce the time taken to simulate mobility.

In order to support file operations, we are planning to implement a scheme similar to that of condor [8]. The file position of all open files will be saved, and the buffers will be flushed. Since we checkpoint the state of an MH (and flush the buffers) only when the MH has to migrate, we can support combinations of read and write operations. (In condor, only idempotent file operations are supported.)

5 Handoff

After MH h is restarted by **msim** m', the following steps take place as part of a handoff procedure executed between the MSS s and the new MSS s' within the MCE. The handoff procedure described below is the same for both the actual mobile environment and the simulation testbed. In case of actual mobile environment, handoff is triggered by reception of *beacon* by MH h from MSS s'. In case of simulation, we simulate reception of beacon from MSS s' during process restart. This triggers the handoff between s and s'.

- MH h registers with MSS s' by sending a message *register*(h, s) ((4) in Figure 3). MSS s' then sends a message to the home MSS of MH h to request verification and to update the current location of h to be MSS s'.
- MSS s' sends a *deregister*(h) message to s ((5) in Figure 3).
- MSS s, on receiving deregister message, removes MH h from its list of local MHs and sets the forwarding pointer for MH h to MSS s'.
- MSS s then transfers the state of MH h stored in s (if any) to MSS s' ((6) in Figure 3). MSS s also transfers any undelivered messages buffered for MH h to MSS s'. If s receives any messages for h, they will be forwarded to MSS s'.
- MSS s' then sends a registration accepted message to h.

6 Features

The simulation testbed provides a set of features, such as *search*(h) and *deliver*(m, h), that could be used while developing, evaluating or testing new applications. These features are outlined in this section. MCE provides these as default features and the implementation details are provided in Section 8.

search(h) returns the current location of the mobile host h. The current location of h is the MSS s' in whose cell h currently resides in. Let *search*(h) be invoked at MSS s. MSS s then sends a message to the home MSS of h enquiring the current location of MH h. The home MSS, on receiving the enquiry, checks to see if the MH has been registered with any MSS by checking its database. If h is registered with MSS s_{curr}, then the location enquiry message is forwarded to s_{curr}. When s_{curr} receives the message, it verifies that mobile host h is present in its cell and sends a reply to the originator of the *search*(h) operation. If h is not present in s_{curr} and if s_{curr} has a forwarding address, say s_{next} for h, then the location enquiry message is forwarded to s_{next}. In case there is no forwarding address for h, a "not found" message is sent as a reply.

deliver(m, h) delivers message m to MH h using a routing strategy similar to the scheme designed by IETF [1]. Every mobile host has a home MSS. Whenever a mobile host registers with an MSS, it will inform the home machine of its current location. $deliver(m, h)$ will send message m to h's home MSS, which will then forward it to the current MSS of h.

7 MCE Platform Design and Implementation Details

Our MCE platform is currently implemented on Pentium PCs running the Linux operating system (version 1.0.8) and on Sun workstations running SunOS 4.1.3. In our initial setup, we do not have true mobile hosts. As stated earlier, mobile hosts are simulated by processes running on the MSS itself. This will be useful to test various protocols and algorithms for the MCE platform. In our initial version, we restrict migration of an MH from MSS s to MSS s' only if s and s' have the same hardware and OS. This is due to the fact that we do process migration by checkpointing and transferring the process to the new host. Our scheme for checkpointing (as explained earlier in section 4.1) is hardware/OS specific which imposes this restriction. This is a limitation only on the simulation testbed part of MCE, since in a real mobile network, mobile hosts migrate on their own to a new MSS and MCE does not have to simulate the mobility aspect.

The MSS server process (**mssd**), which supports the MCE platform, runs in the background as a daemon process. The **mssd** server will be a hybrid server with certain services like *date and time* provided in an iterative fashion and others like *file transfer* provided in a concurrent fashion. The nature of service will dictate whether the server will do it iteratively or concurrently.

The **mssd** process handles all the MCE-related messages going through the MSS. When an MSS boots, the **mssd** process is initiated. The process then sets up the required communication interfaces using sockets to handle message traffic within the MCE. We have three distinct interfaces to handle:

- MSS to MSS static network-based communications between their respective **mssd** daemons.
- messages between an MH and the **mssd** process of its current MSS.
- messages between processes (other than **mssd**) in an MSS and its **mssd** process (intra-MSS MCE communications). This includes supporting **msim** communications.

Once the sockets are created and bound to addresses, we set up the **mssd** process to handle the asynchronous messages arriving on any of the above mentioned interfaces. Another step in the initialization phase is to setup tables needed to verify/authenticate mobile host data. In our initial version, we have two tables as follows:

- **HMSS** database table to keep track of mobile hosts whose home MSS is the local host.
- **VMSS** table to keep track of all mobile hosts currently getting serviced by the MSS. This includes all the mobile hosts that have been registered with the local MSS regardless of their home MSSs.

In order to support the development of mobile host processes, we have developed a *MCE Application Programming Interface* (**MCE-API**). The purpose of this interface is to provide interface routines that will make the underlying issues of mobility etc., transparent to the application developer so that the developer does not require an in-depth understanding of 'how' the MCE works. The knowledge of 'what' the MCE provides is sufficient. The MCE-API also provides a way to ensure that the applications need not be rewritten or modified irrespective of whether the process is being simulated on the MSS itself or is actually run on a mobile host with a wireless communication interface. The MCE-API provides interfaces to handle response to beacon messages for registration, to setup connections, to send and receive messages and to terminate connections. The software architecture and message flow paths are shown in Figure 4.

8 Current Status Report

At present, we have MCE simulation testbed working in our lab with a default routing scheme, default search facility, default registration and default deregistration features. We describe these features in detail and state the current development status along with immediate and future plans. We are planning to add support to let users add similar services without having to modify MCE source code. We view this as a subscription based service and the motivation behind this idea is to be able to use MCE as an evaluation and/or comparison platform to evaluate different algorithms. For example, if there are two routing algorithms to be compared, one can run test cases on both algorithms using the same benchmark applications and compare the results. MCE keeps environment used on both algorithms the same which helps the user get more dependable results.

The following is a list of terms used in the rest of this section.

- h - A mobile host.
- s_{new} - The MSS of the cell into which h has just moved into during cell to cell migration.
- s_{prev} - The MSS of the cell from where h migrated.
- s_{curr} - The MSS of the cell where h currently resides.
- s_{home} - The home MSS of h.

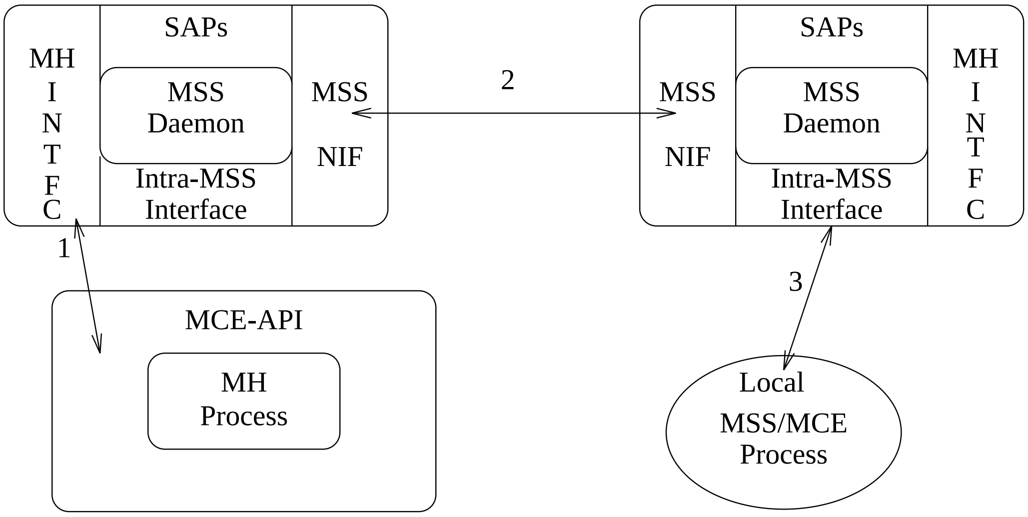
8.1 Simulation Testbed Setup

Figure 5 shows the backbone local area network on which MCE is being developed and tested. Files in any of the machines may be accessed from any other machine using *automount* facility. Each machine is represented by a node in the diagram and the node contains the name[1] and the operating system of the machine. As can be seen, the MCE simulation platform is running on different flavors of the UNIX operating system. The checkpointing scheme that we use is also OS/hardware specific. But the evolution of MCE is being planned to be independent of any operating system and/or hardware specific idiosyncracies as much as possible. The PDAs, notebooks etc. are mostly available on the DOS/Windows operating environments and we are planning to tune MCE-API to be usable on these platforms as well.

8.2 Limitations

We simulate a mobile host by a single process at this time. This poses a limitation on the use of MCE as this restricts the ability to test multiple concurrent applications on a single mobile host. However, the current version of MCE may be effectively used to test applications that are independent of other processes on the mobile host. This implies that we simulate these independent mobile host processes as if they were running on different mobile hosts. Note that this still does not impose or introduce application code that is specific to the simulation environment. But this does not aid in the development of inter-dependent, concurrent mobile host applications. We are working on extending the MCE

[1]The internet address of the machines may be derived by just appending "*.utdallas.edu*" to the names shown



1. Mobile Host <-> MSS Daemon Interface (MH INTFC)
2. MSS Daemon <-> MSS Daemon Network Interface (MSS NIF)
3. MSS Local Process <-> MSS Daemon Interface (Intra-MSS Interface)

MCE-API - MCE Application Programming Interface
SAPs - Service Access Points

Figure 4: MCE Software Architecture

simulation facility to support development of interdependent concurrent processes.

The checkpointing method used is another limitation within MCE simulation testbed. We are exploring methods to do checkpointing and recovery in a machine independent way to overcome the limitation imposed by our current method.

8.3 Utilities

Each mobile host must have subscribed for service with its home MSS. MCE provides a **subscribe** utility that takes subscription data from a user and adds the data to the HMSS database. The HMSS database stores the mobile host's unique network identifier, subscribed services, privileges and last known location for the mobile host. In the current implementation, we use a 2-tuple, [home MSS network address, mobile host identifier] to form the unique network-wide mobile host identifier. Whenever a mobile host registers for service with any MSS, the HMSS database is updated to reflect the current location of the mobile host. There are no service subscription data stored in HMSS database now but as MCE evolves to support various subscribable services like *ftp, email* etc., the subscription tool will evolve to support them. MCE does not have a browser tool to browse the HMSS database and we are working on adding such a tool in the near future. Each MSS has an in-memory database called **VMSS**, to keep track of the mobile hosts that are getting serviced by the MSS at any given time.

8.4 Registration

MCE requires a mobile host to register with the local MSS prior to requesting or receiving any service. MCE provides a default registration service for mobile hosts to register with their local MSS. Mobile hosts may request registration services with the local MSS under (at least) two different circumstances and both of these are explained below.

- MH h is powered up and hence does not have an MSS that is currently providing service. In this scenario, when h receives a *beacon* from the MSS of the cell it currently resides in, say s_{new}, h sends a registration request to s_{new}. s_{new} forwards the message to the home MSS of h,

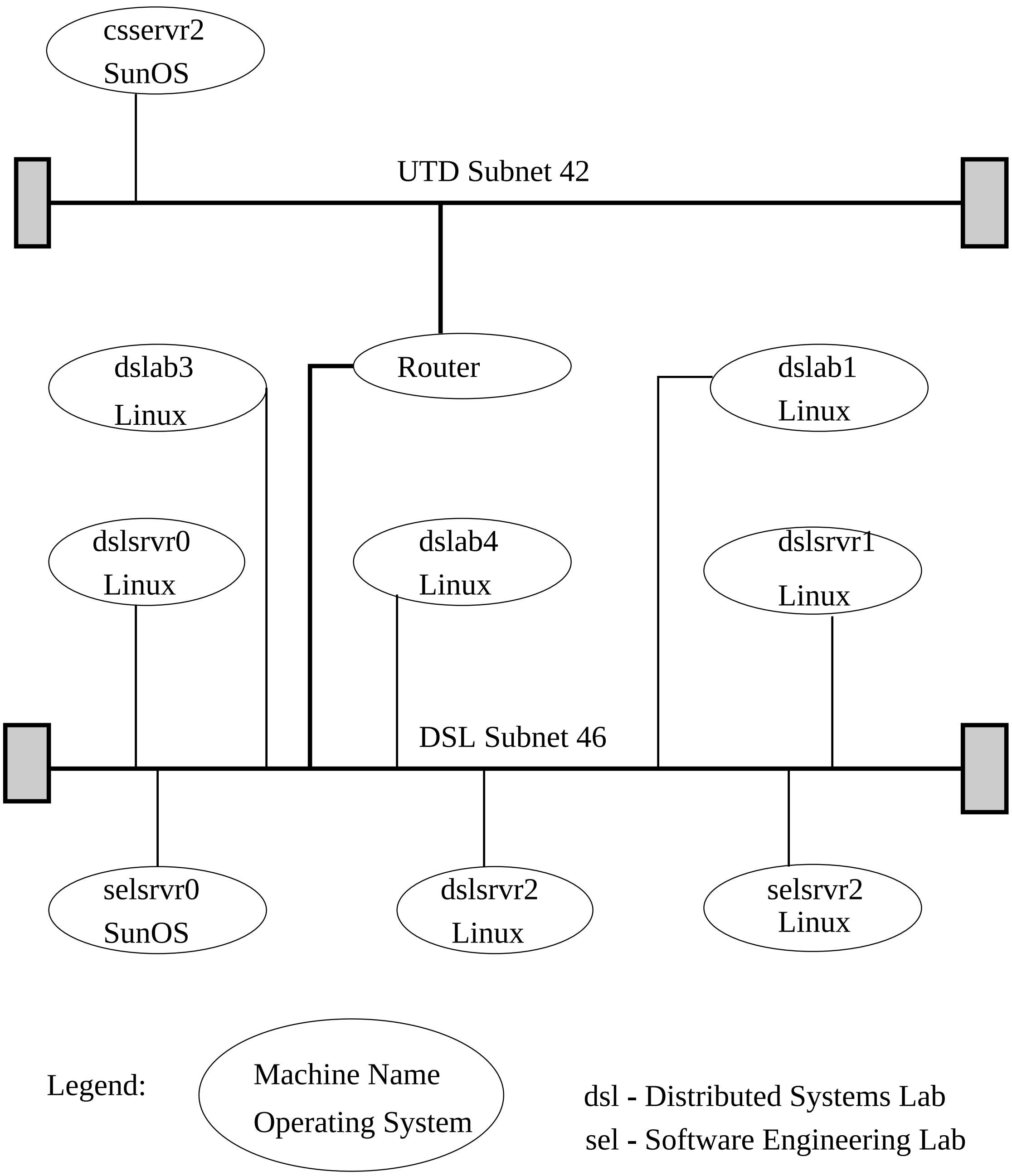


Figure 5: MCE Simulation Testbed Setup

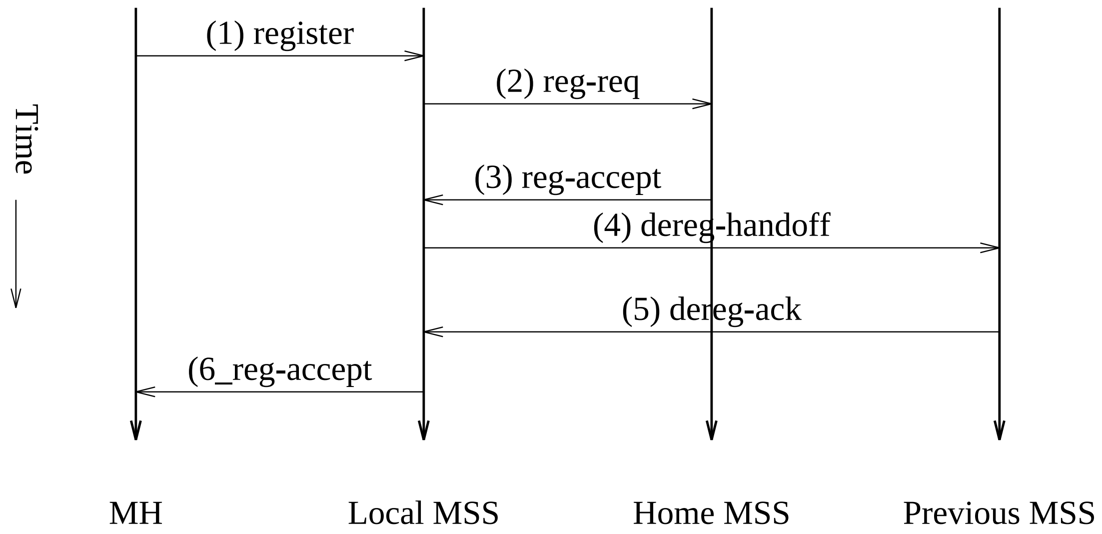


Figure 6: Mobile Host Registration Message Ladder

say s_{home}. When s_{home} receives the registration request, it validates the information and sends an *accept* or *deny* message. If s_{home} accepts the registration request, the HMSS database is updated with the location information about h. s_{new} receives this message and forwards it to h. If the registration was accepted, s_{new} also modifies the VMSS tables with data about h. Once this registration process is completed, MH h may receive messages from anywhere in the network at the new location via s_{new}.

- MH h has just moved into the cell of s_{new} and receives a *beacon* from s_{new}. The MSS address is different from what h remembers and so h sends a registration request to the new MSS s_{new}. The sequence is the same as the previous case till s_{new} receives a reply from s_{home}. If the registration succeeds, s_{new} sends a deregistration due to handoff request to the previous MSS as remembered by h, say s_{prev}. When s_{prev} receives the deregistration due to handoff message from s_{new}, all buffered messages for h are forwarded to s_{new} and then the VMSS table entry for h in s_{prev} is deleted. s_{prev} then sends a deregistration acknowledgement to s_{new}. When s_{new} receives this acknowledgement, it updates its VMSS table for h and then forwards an accept message. If registration was denied, no deregistration request is sent by s_{new}.

Message ladder depicting the second registration scenario is shown in Figure 6. The default registration algorithm used by MCE is presented in Appendix A.1.

8.5 Deregistration

MCE uses a deregistration message to keep the location tracking of mobile hosts up-to-date in VMSS and HMSS databases. In case of deregistration also, there are two scenarios that are supported by MCE.

- Mobile host h is going to be shut down. This may result in the mobile host sending a deregistration with disconnect as the reason. When s_{curr}, the current MSS of h, receives it, s_{curr} will forward the request to s_{home}. Then s_{curr} will clear the data about h from its VMSS table. When s_{home} receives the deregistration *disconnect* message, it updates its HMSS database to reflect h's disconnected state from the network.

- The disconnect message is sent as part of the handoff sequence as described in registration earlier. The disconnect message sent by s_{new} will indicate handoff as the reason. When s_{prev} receives this message, it forwards all the buffered messages for h to s_{new}, then sends a deregistration acknowledgement and clears VMSS table of h's data.

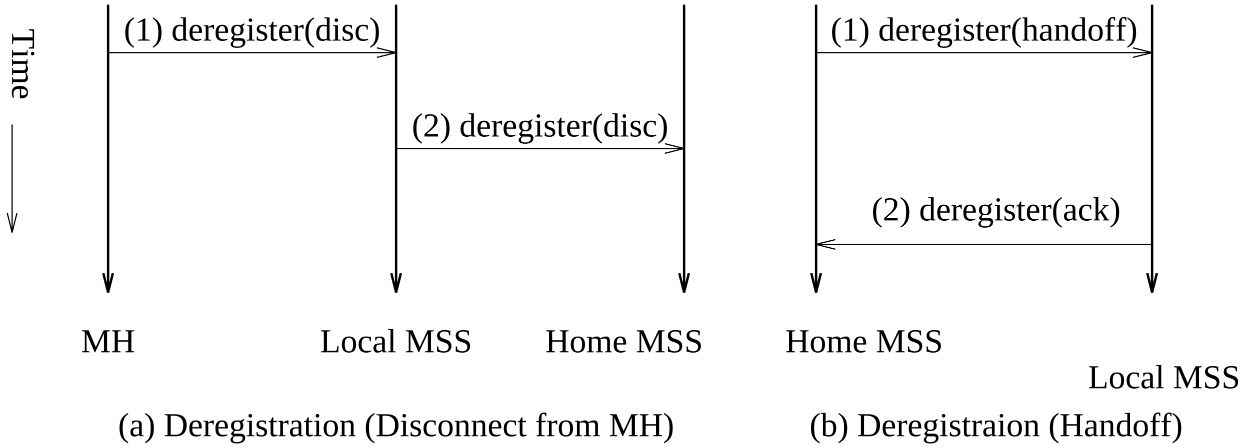


Figure 7: Mobile Host Deregistration Message Ladders

Message ladders for these deregistration scenarios are presented in Figure 7. The default deregistration algorithm used by MCE is presented in Appendix A.2.

8.6 Routing Scheme

MCE provides a message routing mechanism, which is similar to IETF [1]. MCE assumes that message delivery between mobile hosts is done using their respective mobile support stations. This assumption not only helps in simulation but also in providing other services like tolerating MSS failures [2]. The default scheme provided for routing within MCE is built using datagram sockets on UDP/IP protocols above the transport layer. This may be modified to be part of IP layer in the future. MCE's default routing scheme is illustrated in Figure 8. When a mobile host h_{src} sends a message m_1 to another mobile host h_{dest}, the message m_1 is first received by the current MSS of h_{src}. The message is then forwarded to the home MSS of h_{dest}. Home MSS of h_{dest} looks up h_{dest}'s location in its HMSS database and forwards m_1 to the current MSS of h_{dest}. When the current MSS of h_{dest} receives m_1, it looks up its VMSS database for information on h_{dest} and delivers the message to h_{dest}. The routing scheme is not reliable and assumes that applications have mechanisms to ensure reliable message delivery. We are planning to add reliable message delivery options in the near future. The routing algorithm executed at every MSS is described in Appendix A.3.

8.7 Mobile Host Search Facility

The *search*(h) feature as described earlier in Section 6 is provided by the current version of MCE. The implementation uses the default routing scheme. The location enquiry message is forwarded to the MSS where h resides. Instead of delivering the message to h, the local MSS returns a message giving the current location information to the originator of the *search*(h) request.

9 Future Work

We plan to enhance MCE so as to make it a complete simulation testbed as well as a platform to support a mobile network environment. Currently we are planning to add support to provide subscribable services like *ftp, email, talk* etc. We are working on schemes to let the user subscribe for a service either on an iterative or on a concurrent basis. We plan to add fault tolerance [2] and study the performance. In addition, we are planning to provide support to do performance comparison and analysis of routing schemes. Support for concurrent mobile host processes is also being planned for the near future.

An *ftp site* for accessing MCE software and documents will be setup in the very near future. Once

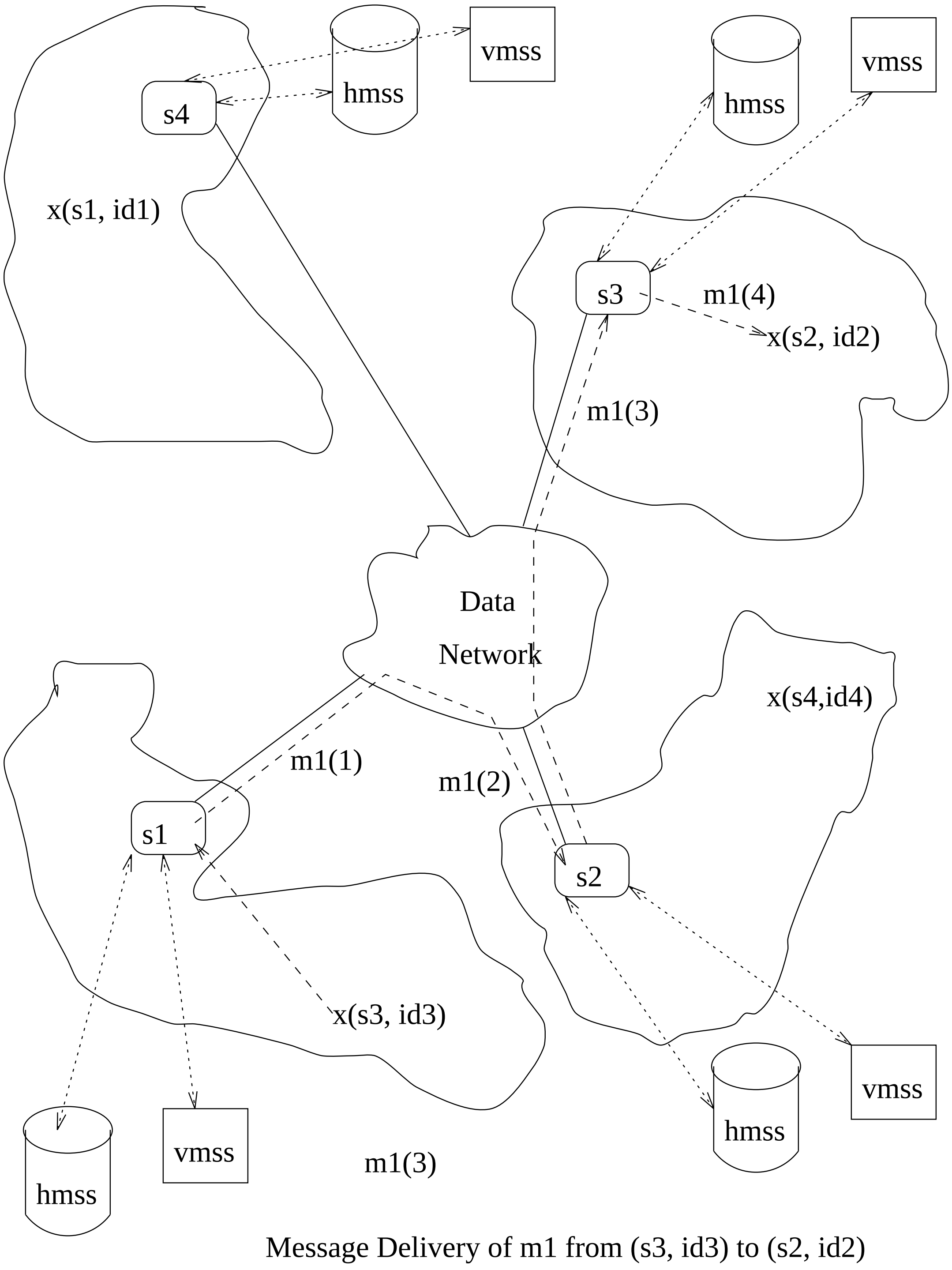


Figure 8: MCE Default Message Routing Scheme

setup, the access will be provided through *anonymous ftp* to *ftp.utdallas.edu*. The proposed directory is *pub/cs/venky/MCE* and a README file in that directory will explain the setup procedures for MCE.

Acknowledgements: We would like to thank the anonymous referees for their insightful comments and suggestions.

References

[1] draft-ietf-mobileip-protocol-07.txt. In *INTERNET DRAFT*, C. Perkins, Ed. Internet Engineering Task Force, 1994.

[2] Alagar, S., Rajagopalan, R., and Venkatesan, S. Tolerating mobile support station failures. Computer science technical report, The University of Texas at Dallas, November 1993.

[3] Alagar, S., and Venkatesan, S. Causal ordering in distributed mobile systems. In *Workshop on Mobile Computing Systems and Applications* (1994).

[4] Badrinath, B., and Acharya, A. Delivering multicast messages in networks with mobile hosts. In *Proceedings of the 13th International Conference on Distributed Computing Systems* (1993), IEEE, pp. 292–299.

[5] Diehl, N., and Held, A. A system platform for mobile computing applications. In *MobiData Workshop* (1994), Rutgers University.

[6] Ioannidis, J., Duchamp, D., and Maguire, G. IP-based protocols for mobile internetworking. In *Proceedings of ACM SIGCOMM Symposium on Communication Architecture and Protocols* (1991), pp. 235–245.

[7] Johnson, D. Scalable and robust internetwork routing for mobile hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems* (1994), IEEE.

[8] Litzkow, M., and Solomon, M. Supporting checkpointing and process migration outside the unix kernel. In *Proceedings of Usenix Winter Conference* (1992).

A MCE Default Algorithms

A.1 MCE Registration Algorithm

---

if the message is from a local mobile host
    if home MSS of the mobile host == local MSS
        update current location information on HMSS databse;
        update the VMSS table for mobile host interface data;
        if the registration is part of handoff
            send deregister message to previous MSS;
        else
            send regsiter accept message to mobile host;
    else if home MSS is not the local MSS
        forward the register message to the hmss MSS of the mobile host;
else if the message was received on the MSS network interface
    if the message is a registration request message
        if local MSS is not the home MSS of the mobile host
            ignore the message and return;
        retrieve the profile of the mobile host from HMSS database;
        verify the mobile host request and validate;
        if registration is valid request
            update current location information on HMSS database;
            send registration accepted message to the new MSS of the mobile host;
        else
            send a registration denied message to the new MSS of the mobile host;
    else if the message is a registration accepted message
        if registration is due to handoff
            send a deregister message to the previous MSS of the mobile host;
        else
            send registration accepted message to the mobile host;
    else if the message is a registration denied message
        send registration denied message to the mobile host;
        delete any current information on the mobile host;

A.2 MCE Deregistration Algorithm

---

if the deregister message is from a local mobile host
    update the VMSS table for mobile host interface data;
    if the home MSS of the mobile host is local MSS
        update current location information on HMSS databse;
    else if home MSS is not the local MSS
        forward the deregister message to the home MSS of the mobile host;
else if the message was received on the MSS network interface
    if the message is handoff deregistration message
        if local MSS is not the home MSS of the mobile host
            ignore the message and return;
        forward all the buffered data for the mobile host to the source MSS of the message;
        cleanout mobile host data in VMSS table;
        send deregistration accepted message to the source MSS of the message;
    else if the message is a disconnect deregistration message
        cleanout mobile host data in VMSS table;
    if this MSS is the home MSS of the mobile host that is getting deregistered

```
        update the HMSS database to reflect the disconnected state of the mobile host;
    else if the message is a deregistration acknowledgement message
        /* this is the reply to deregistration message */
        /* to the previous MSS during handoff */
        update VMSS table with the mobile host data;
```

A.3 MCE Routing Algorithm

```
if message was received on the mobile host (wireless) interface:
    if home MSS of destination is the local MSS
        find location information of mobile host from HMSS database;
        if mobile host is in cell of home MSS
            send message to destination mobile host;
        else
            forward the message to the MSS of current location of the destination mobile host;
    else if the home MSS of destination is different
        forward the message to the home MSS of the destination;
else if message was received on the MSS network interface:
    if home MSS of destination is the local MSS
        find location information of mobile host from HMSS database;
        if mobile host is in cell of home MSS
            send message to destination mobile host;
        else
            forward the message to the MSS of current location of the destination mobile host;
    else /* not home MSS of destination mobile host */
        if destination mobile host is registered with local MSS
            send message to destination mobile host;
        else if destination host has buffering setup
            buffer the message to be forwarded later;
        else
            message gets dropped;
```