On the Implementation of End-to-End Mobility Management Framework (EMF)

1E. Elahi  2M. Yousaf  3A. Sheikh  4M. Maaz Rehan  5M. Omer Chughtai  6A. Qayyum  
Center of Research in Networks and Telecom (CoReNeT)  
M. A. Jinnah University, Islamabad, Pakistan  
{1ehsan, 2myousaf, 3anbreen, 4maaz, 5omer}@corenet.org.pk, 6aqayyum@ieee.org

Abstract—Traditionally the mobility problem has been solved mostly at network layer. However the End-to-End Mobility Management Framework (EMF) [1] provides the solution to this problem above the transport layer. EMF overcomes some limitations of current mobility management solutions by effectively providing mobility services such as soft handover, willful handover, location updates etc. EMF neither requires any support of additional entities in the network nor requires the changes in the current implementation of TCP. This paper describes a portable implementation design of the EMF framework along with the tradeoffs involved in implementing the framework. The results of some experiments to measure the performance of EMF are also presented that quantify the protocol and computational overheads.

Keywords-component;Mobility Management, Willful Handover, Network Protocol Implementation, Code Portability, Application Transparency

I. INTRODUCTION

The proliferation of handheld devices with the support of multiple interfaces makes it a common requirement of users to have access to a number of wired or wireless networks simultaneously. The users would like to have the liberty to select among several concurrently available access networks the one which best supports the user’s current communication needs. Users also wish to remain connected while on the move. This requires mobility management solutions to support both soft and willful handovers for both mobile and stationary users. When users move away from the coverage area of one access network to another with different network technologies e.g. from WLAN to WIMAX, they might wish to migrate their connections in this new access network. This migration is termed as vertical handover [5]. In order to support vertical handovers for applications running over TCP, various solutions have been proposed in the literature. Most of these solutions either require additional entities in the network such as home agent, foreign agent, proxy etc or they require changes in the current TCP implementation [1].

Mobility management solutions, proposed so far, can be classified according to the layers of the TCP/IP protocol stack on which these solutions operate [2]. The solutions provided at network layer e.g. Mobile-IP [3] do not provide willful handovers. Moreover such solutions are not able to exploit the available multiple interfaces for bandwidth aggregation using data stripping [4]. The transport layer solutions to handle end-to-end mobility management such as TCP-MIGRATE [6], MSOCK [7], pTCP [8] etc require changes in existing TCP implementation. The detailed analysis of such solutions can be found in [9].

End-to-End Mobility Management Framework (EMF) [1] has been designed while keeping in view the limitations of other mobility management solutions. As will be briefly described in section 2, EMF is designed in conformance with Internet’s legacy philosophy of “smart-edges simple-network model” [10]. It does not require any extra entity in the network to provide mobility services. It also does not require changes in the current TCP implementation. EMF framework supports both forced handover as well as willful handover. EMF framework provides both horizontal and vertical handover as well as utilizes the availability of multiple network interfaces to support data stripping for bandwidth aggregation [11]. This paper describes an implementation of End-to-End Mobility Management Framework (EMF) [1]. The implementation is designed to support various scenarios of willful and forced handovers as well as bandwidth aggregation.

The rest of the paper is organized as follows: In section II, an overview of EMF framework including the protocol mechanism is presented briefly. The detailed protocol mechanism and EMF architecture can be seen in [1] and [11]. Section III describes the current implementation in detail, including the relevant primitives, architectures, data structures and user interfaces and all related modules. This section also includes the details about the extended socket API. This extended socket API is designed to help developers in order to develop new EMF aware applications. The framework is implemented in such a way that it can be ported on other platforms easily. Section IV describes that how the framework is made portable. It is a big challenge to provide EMF support to the legacy applications which are built using standard socket API. Section V describes how the mobility services have been provided to such applications transparently. Section VI describes some limitations of EMF framework and its implementation. Section VII describes the performance of the EMF protocol and its implementation. Finally, some concluding remarks are made in section VIII with future directions.

II. EMF OVERVIEW

At the transport layer, a TCP connection is distinguished by the combination of source IP, source port, destination IP and destination port. When one end moves from one network to another, its IP address, which is the unique identifier of its physical attachment to the Internet, gets changed. This causes the TCP connection loss. Consequently the socket descriptors

---

This R&D project was sponsored by ICT R&D Fund, Ministry of IT, Govt. of Pakistan

---

Accepted for publication in “6th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2010)” being held on 11-13 October 2010 at Niagara Falls, Canada
open at both ends become invalid. As a result the client applications need to re-establish their TCP connections with the application servers in order to communicate with each other. The EMF framework is based upon the principal of connection diversity [1]. It means that if the TCP connection becomes lost and if there is another network available, a new TCP connection can be established for service continuation and this must be done transparent to both application end points. All this is done by introducing a shim layer between application layer and transport layer in TCP/IP protocol stack.

A. EMF Objectives

The basic objective of EMF is to develop an end-to-end framework that would be capable of handling mobility management across heterogeneous networks without requiring additional entities in the network. It would also do not require any changes in the current implementation of TCP/IP protocol stack. EMF framework would also support bandwidth aggregation by simultaneously utilizing multiple network interfaces, which resulted in better throughput and effective resource utilization. EMF is designed to operate above the transport layer and attempts to overcome some of the limitations of current mobility management solutions.

The goals of EMF framework include the provision of seamless and broad range of pervasive end-to-end services across heterogeneous networks regardless of the wireless or wired technology. Any user equipped with a multi-home terminal should be able to exploit multiple available links at the same time and move transparently from one to the other.

B. Protocol Mechanisms

As mentioned earlier that detailed protocol design specifications and EMF architecture can be seen in [1] and [11], therefore in this text, EMF architecture is discussed very briefly. The block diagram of EMF architecture as proposed in [1] is shown in Figure 1. The architecture comprises of four main modules:

1. EMF Data Handler (DH)
2. EMF Control Module (CM)
3. EMF Host Agent (HA)
4. EMF User Agent (UA)

EMF Data Handler (DH) is responsible for the data scheduling and in-order reassembly. EMF control module is responsible for the session creation, maintenance and graceful termination. EMF control module also receives the handover and bandwidth aggregation commands from EMF Host Agent (HA) and takes appropriate actions. EMF User Agent (UA) is the graphical user interface (GUI) module through that user can give its own preferences regarding the use of multiple network interfaces. In UA, user can choose the traffic type that should be given the EMF services. Through this interface, users can also initiate the willful handovers and bandwidth aggregation of multiple available links through this interface.

IEEE 802.21 (Media Independent Handover) standard is in the process of implementation and testing therefore an alternate module called Common Layer 2 Interface (CL2I) [11] has been designed so that it can inform HA about the network changes occurred at link layer. CL2I sends triggers to HA if a link down and/or link up events occur. The HA than takes appropriate decisions on the basis of user preferences and then sends commands to EMF control module for handover, bandwidth aggregation etc.

When the client application needs to connect with some remote server, the EMF Data Handler (DH) sends request to HA to check for the local compliance of such traffic. The HA checks, based upon the user preferences given in UA, whether the requested traffic type should be given EMF services or not. If such traffic type found to be not EMF compliant then upon the reply of HA, the DH will not initiate the association establishment process and consequently such traffic type will not be provided EMF services. If HA finds that this application is EMF compliant than it replies the DH with most preferred network interface’s IP address so that DH could bind through that interface for this association. Now the Association Handler, which is the sub-module of EMF control module, sends an Association Establishment Request to the EMF control module of the remote end. The remote end’s Association Handler replies with Association Establishment Response message. These association request/response messages contain the public keys of each end for the generation of session key. This session key will be used to encrypt/decrypt the 32 bit Association Identifier (AID) in the control messages. This mechanism will be discussed in detail in section III-D.1. This completes the association establishment process. Now when one end wants to send data to its remote peer, the EMF DH module takes the data from the application, append EMF header [11] containing AID, EMF_SEQ_NO etc and send it to the DH of the remote end. The receiving DH receives the data and if it finds that data is received in-sequence then it immediately delivers that data to the receiving application otherwise buffer that data until the arrival of the remaining bytes. Although at single link, TCP would make the bytes in-order but data may become out of order if sent over multiple links for bandwidth aggregation. In this case DH makes the data in-order.

![Figure 1: EMF architecture as proposed in [1]](image-url)
In case of link down, the CL2I sends a message to the HA about the event. HA then sends handover message to the CM containing the IP address of the alternate network interface. The control module then creates a new TCP connection with the remote end and keep it under the same association i.e. value of AID remains the same. In this way, both application level end points are unaware of the change occurred underneath and data continues on sending and receiving through the new TCP connection. The same procedure is followed if the user initiates a willful handover through the UA. In that case the UA sends a message to the HA to initiate handover.

In order to aggregate the bandwidth of multiple access networks available through multiple network interfaces attached to the device for a particular traffic, the user select the bandwidth aggregation option from the UA. Then HA will send the aggregation command to the CM containing the IP addresses of the locally attached interfaces to be added for bandwidth aggregation. The CM then makes new TCP connections using each interface with the remote end. The AID will remain same for each new TCP connection. The DH of sending end then start the process of data stripping and sending data on each connection depending upon the observed throughput for each individual link. It means that data size being sent on each link is directly proportional to the average throughput observed on that link. The DH of the receiving side will receive data from each individual link, reassemble it according to the sequence number found in the EMF header and pass on to the application.

When one of the application end-points closes the connection, the DH of that end schedule all the pending data from its send buffer and then CM will send an Association Termination Request [11] to the other end on any one of the available TCP connections and wait for the response. The other end, on reception of the termination request, will send all the data from its send buffer and then reply with the Association Termination Response [11] and closes all the connections opened under that association. Upon reception of termination response, this end will also close its opened connections under that association. This completes the graceful association termination procedure.

III. IMPLEMENTATION DESIGN

EMF framework comprises of four main modules. These modules are EMF Data Handler (DH), EMF Host Agent (HA), EMF User Agent (UA) and EMF Common Layer 2 Interface (CL2I). Each module runs as independent process. The block diagram of the implementation design is shown in Figure 2. Unlike the proposed design in [1], we used TCP for exchange of control messages. Because we argue that to start an association or to initiate handover we have to establish a new TCP connection in any case that could also be used for control messages. Moreover for terminating an association or remove from BA, we already have a TCP connection that could also be used for control messages. The EMF Data Handler and EMF Control Modules are combined in a single process named as EMF Control and Data Handler (CDH). As shown in the Figure 2, the client application is placed above the CDH while server application is kept independent to the EMF. This is because that the server end must have support for both EMF enabled connections as well as legacy TCP connections. The client application will use EMF_Sock_Lib API instead of socket API in order to avail the mobility services. When the framework started, initially HA process opens a socket and starts listening for connection requests coming from localhost (127.0.0.1) only. After that CDH, CL2I and UA will start and make connections with HA. In the following text, the functionality of each module and its sub-module is discussed in detail:

A. User Agent

It provides the interface where users can choose their preferences. It is a multi-tab GUI program developed in C++ language with QT 3.0 and QT 4.0 as GUI design tools. When UA starts, it creates a connection with the HA. After that it loads the previous user preferences saved in a file and sends it to HA. If HA finds some dissimilarity in the attached interface information, it sends a message to UA to update its information about the attached interfaces. HA always have the updated information about the attached interfaces because it is directly connected with CL2I which is continuously sensing link changes underneath.

In UA, there are three major tabs, general, advanced and help. In general tab, the user can enable/disable EMF startup, handover and bandwidth aggregation services. In advanced tab, there are six more sub-tabs. The detailed description of the complete user interface can be found in [11]; here the purpose of each tab is discussed very briefly.

- **The interfaces tab**: Displays the list of all those interfaces for which layer 3 connectivity is available. The user can select the interfaces to be placed under the EMF.
- **Authentication**: User can select the authentication mechanism to be used for key exchange and encryption or decryption of Association_ID in control messages. Details are given in section III-D.1 of this paper.
- **Protocol Preferences**: EMF facilitates the users to select the traffic type for which EMF services are required. In this tab, user can enter the application layer protocol name and the port number on which the server would be listening for
connections e.g. for HTTP, user will enter 80 in port number field. This is because when user application calls `connect()` and pass the destination address structure, the EMF_Sock_Lib grab the destination IP and port number from the structure in order to check for the user preferences.

- **Handover:** In this tab the user can give the handover preferences for attached network interfaces. It means that if one link becomes down then HA can choose the next preferred link so that all the associations be shifted on.

- **Bandwidth Aggregation (BA):** If the user wants to aggregate bandwidth for two or more available networks then he/she can select the application layer protocol for which BA is required and then select the interfaces from the list to be used for BA.

- **Monitoring:** This tab shows the network statistics of various parameters e.g. for WLAN, user can view line chart of the received signal strength. Based upon those graphs, user can opt for willful handovers and add/delete network interfaces in bandwidth aggregation manually for a selected application protocol type.

- **Manual Commands:** As mentioned earlier that one of the goals of EMF is the willful handovers. In order to provide the interface for willfull handover, this new tab is developed. Here user can initiate manual handover of all the traffic of a selected link to the other link or user can shift some selected type of traffic from one link to another available link. Moreover the user can manually add/remove a link to BA for a specific type of traffic e.g FTP.

The preferences given by the user are stored in the form of a structure. When the UA makes any changes in values of members of this structure, it stores the updated structure in a file shared between UA and HA and then sends a `User_Preferences_Changed_Indication` message to HA. The HA then reads that file and updates its own structure of preferences. The purpose of using such shared file for preferences storing is that the contents of this structure can be changed both by HA and UA. HA will change this structure when it receives any trigger from CL2I about the changes in the attached interfaces. Moreover when system restarts, the preferences stored previously are required to be loaded so that user need not to enter his/her preferences every time the system restarts. When link up or down events occurred, the UA pop-ups a message in the system tray of the desktop of the user so that the user can initiate willful handover or bandwidth aggregation if needed.

### B. Common Layer 2 Interface (CL2I)

It is envisioned in [1] that IEEE 802.21 is being developed to standardize an abstraction layer in order to provide Media Independent Handover (MIH) to enable handover and interoperability between heterogeneous networks. IEEE 802.21 has been now standardized [13] but still its implementation is underway that could be used for event services such as “link up”, “link down” etc. Therefore it was decided to design a new abstraction layer that can provide only those MIH like services which are required by EMF HA for decision making related to handover and bandwidth aggregation [11]. This layer is named as Common Layer 2 Interface (CL2I) [11].

IEEE Media Independent Handover (MIH) provides two kinds of services i.e. local services and network services [13]. The CL2I design provides many local services including event management of periodic and threshold events, Link Information Capture and Process (ICP) and Candidate Link ON/OFF Policy (CLOP) but in this paper a simplified implementation of CL2I is presented. In this implementation, there are no subscription or un-subscription services by the user application. Just the “link up” and “link down” services have been implemented. When the CL2I starts, it immediately connect with the HA server locally. After that, it opens a socket of type SOCK_DGRAM and goes into an infinite loop where it starts getting interface information and their IP addresses. The CL2I maintains two main structures, `Trigger_Info` and `Interface_Info`. `Trigger_Info` structure is used to send any type of link up or down trigger to the Host Agent while `Interface_Info` is a linked list used to store configuration information of all the configured interfaces. Figure 3 describes the main functionality of CL2I in the form of pseudo code:

```plaintext
Figure 3: Pseudo code for CL2I implementation
```

### C. Host Agent (HA)

Host Agent is a cross layer module whose main purpose is to get the information about the available links from the CL2I, collect the user level preferences from the UA, takes decisions about handovers and bandwidth aggregation based upon the collected information and then sends commands to the CDH for the implementation of those decisions.

EMF HA is the entity that starts as a server process and all the other local modules connect with it through stream pipes. HA reads the shared file of user preferences, created by UA, as discussed in section III-A. HA maintains a linked list that is used to store information about the current associations running under EMF. The attributes of this list are a 32 bit AID, traffic type of the association, number of interfaces being used under this association and connection_id array that contains socket descriptors of the connections running under this association. Each connection_id is present at the index where there is a valid IP address available at the same index in the `char *IP[INTERFACES]` array maintained by HA. The pseudo code for HA implementation design is shown in Figure 4.
1) Location management through DNS dynamic updates

Location management is not a big issue for the mobile nodes acting only as a client but when a mobile node acts as a server and changes its location, its IP address also changes. DNS dynamic updates provide the facility to dynamically update the name-address mapping of the hosts [14]. Sending DNS dynamic update message doesn’t affect the ongoing connection(s) because the problem for maintaining ongoing connection(s) is resolved with EMF handover mechanism. Location update mechanism is helpful for new connection requests only. In order to accomplish this task, two functions add_DNS_Record(..) and delete_DNS_Record(..) have been developed. Whenever add or delete a record entry is required, the HA creates a new thread to do that because this process may take some time that may cause delays in the normal functionality of the HA. In add or delete DNS record functions, nsupdate command is called along with requisite parameters. The essential parameters include DNS specific IP address of the node, DNS record type, DNS IP and node’s IP.

D. EMF Control Module and EMF Data Handler (CDH)

These two modules are implemented in such a way that they are combined in a single process. As discussed in section II-B that EMF Control Module is responsible for creation and termination of associations and management of handovers and bandwidth aggregation. EMF Data Handler is responsible for sending and receiving EMF data packets to and from remote applications. Along with several other structures, the CDH maintains three main data structures as shown in Figure 5.

EMF proposes three fixed ports for its inter-module communication. Therefore in this implementation, we selected 3634 as listening port for HA, 3635 as listening port in CDH for connection requests from EMF_Sock_Lib and port number 3636 is used by CDH as listening port to entertain connection or association requests from remote CDH. The structure EMF_Send_Buffer is the main buffer that is used to store data sent by local application. Its size is equal to the sum of all TCP send buffers sizes. For example if there are three TCP connections under the same association and each TCP send buffer size is equal to 16KB then the EMF_Send_Buffer size will be 48KB. When one EMF data packet is sent to one of the TCP connections, it has to be buffered until it is received at CDH of remote end. This is done through adding that packet as a new node in the connection_send_list. As shown in Figure 4. Each member of the linked list represents one EMF data packet. Unlike TCP, there is no acknowledgement mechanism in EMF. We have to rely on the return value of send() call. When a packet is sent and if the return value is equal to the size of that packet, it means that there was enough room in TCP send buffer to accommodate this packet, so the value of Performance_Measure(PM) is set to 1 for that connection (Figure 5). Equation 1 shows how the PM is calculated.

\[
\text{Performance Measure}(PM) = \frac{\text{Return value of send}}{\text{Total size of packet}}
\]

If the total size of the data packets present in the connection_send_list becomes equal to or greater than the size of TCP send buffer and this time the value of PM becomes equal to 1, it is assumed that the oldest packet must have been received at the remote end therefore the oldest node in the list is deleted and new node be added in order to accommodate the newly sent packet.

If at any time the value of PM is less than one, after that, in the next iteration, CDH do not create new data packet rather it just try to send remaining bytes of the previous packet. Because PM value less than 1 means that TCP send buffer does not have enough space to accommodate whole packet. Here each send call is made on non-blocking sockets. One node in the Assoc_List structure represents a running association as shown in Figure 5. There is a 32 bit integer, assoc_id, which is a unique identifier of a single association. The process of creating this identifier is described in [11]. The pseudo code for CDH module is given in Figure 6 that describes the complete flow.

It is a single process that sequentially handles all the associations. There is a design choice to create independent threads for each association but in that case HA have to maintain connections with each individual thread in order to send commands which may be a costly procedure with respect to HA as well as CDH.

1) Secure Association Handler

There are several control messages which are to be exchanged between both ends during lifetime of association.
These messages are related to handover, bandwidth aggregation and association termination. An attacker can hijack the ongoing session by sending spoofed handover message to indicate the corresponding node that the mobile node has changed its location and subsequent packets would be delivered to the new location (attacker’s location). In order to avoid such hijacking attacks, proper authentication mechanism is essential. EMF uses the Association Identifier in order to communicate control messages. This identifier is unique within the communicating host systems for one association. This identifier is exchanged with cryptographically secured mechanism. When CDH starts, it creates a public private key pair. The public key is shared to the other peer at start of the association. After that both ends will generate a unique session key using their own private key and other end’s public key. All this have been done using Elliptic Curve Diffie-Hellman (ECDH) mechanism for key exchange [15][16]. When one end needs to send a control message, it will encrypt the 32 bit Assoc_ID using AES and append it with the control message and send. The only drawback of this mechanism is that the encryption of 32 bit ID using AES will result in 128 bit encrypted block. But this overhead can provide protection against session hijacking.

**E. EMF_Sock_Lib**

In order to help application developers to develop such applications that require mobility services, a library is developed that contains some wrapper functions for socket APIs. These wrapper functions start with ‘EMF’ prefix e.g. wrapper function for `send()` call is `EMF_send()`. Each wrapper function takes same number and type of arguments as of original function. The pseudo code for major wrapper functions is given in Figure 7.

```
start()
create two Listening sockets, one for remote connections and other for EMF_Sock_lib connect with Host Agent

do{
    select(..., timeout value=300s);
    schedule_data(); /*go and check if there is any data to send under any association*/
    if(PD_Isset(listening_socket_for_sock_lib))
        handle_connection_request_from_application();
    else if(PD_Isset(listening_socket_for_remote_connection))
        handle_new_connection_or_association_request_from_remote_end();
    else if(PD_Isset(Socket_for_host_agent))
        handle_MA_commands();/*Commands may be harden, add or remove BA etc*/
    end_if;
for each association node from head to tail{
    if(PD_Isset(assoc_node->lib_sock_fd))
        recv as much data from application as the available space in EMF_Send_Buffer
    end_for
for each connection i under this association{
    recv WF header and parse the packet type;
    if type is some control message e.g. association terminate request then act upon it as per protocol specifications;
    if type is a data packet then recv and reassemble data.
    if i==end
        send the remaining data from the Connection_Send_List for connection i
    end_if
    end_for
}end_for_loop
}
end

handle_MA_commands()
receive the command and check CMD_TYPE and TRAFFIC_TYPE
if CMD_TYPE = hangover
for all associations with matching TRAFFIC_TYPE
    create new stream socket(...); bind it to IP address given by MA
    call connect and send HANGOVER command to remote end; close old TCP connection
end_for_loop
else if CMD_TYPE = BA
for all associations with matching TRAFFIC_TYPE
    for all IP addresses given by MA
        create new stream socket(...); bind it to IP address given by MA
        create new TCP connections and send BA command to remote end
        increment connection counter under this association
end_for_loop
}
end_for_loop
}
end
```

Each application has to include `EMF_Sock_Lib.h` file instead of `socket.h` in order to avail EMF services. Section V describes how legacy applications can use EMF services if they do not include `EMF_Sock_Lib.h` file.

### IV. FRAMEWORK PORTABILITY

Initially the framework was implemented on Fedora core 9 distribution for Linux with kernel version 2.6.25 and 4.4BSD release 2004. This implementation worked fine on other Linux distribution e.g. Red Hat, Ubuntu etc. It was part of the goals that this framework be portable to Microsoft Windows also. In Windows the socket API implementation is available in the form of Winsock API. There are many calls and routines that are not available or available with different syntax in Winsock API which is a socket API for Windows. In order to make the framework completely portable for both Linux and Windows, a glue layer has been developed for the framework.
On Fedora core 9, this task has been achieved by overriding the socket calls using $LD_PRELOAD$ environment variable. This environment variable is used to point to our own shared library i.e. $EMF_Sock_Lib$, which contains a function with the same name as the one which is going to be overloaded. In order to make $EMF_Sock_Lib$ as shared library, following compilation procedures are used:

```bash
# gcc -fPIC -rdynamic -g -c -Wall EMF_Sock_Lib.c
# gcc -fPIC -static -shared -o EMF_Sock_Lib.so EMF_Sock_Lib.c -lc -ldl
```

After that the .so file is placed in the same directory where the legacy application’s binaries exist. Then set the path as follows:

```bash
# export LD_PRELOAD=./EMF_Sock_Lib.so
```

This completes the transparency support for legacy applications. The API redirection/overloading is limited to a single directory level because when it is implemented system wide, system instability was observed. This is due to the fact that Linux uses socket API calls very often, therefore every time call interception and redirection to EMF CDH is a very costly procedure that’s why system shows unexpected behavior. Due to this fact, it is decided to overload the API calls of only that application whose directory path is specified by the user.

### VI. Design Limitations

For an end-to-end reliable transmission of data, only single TCP connection is used between client and server applications. In order to provide end-to-end mobility services through EMF framework, there are three TCP connections through which data will be relayed. First connection is a local stream pipe between client application and EMF CDH at client side. Second is a TCP connection between two ends of EMF CDH and the third TCP connection is also a local connection between server application and EMF CDH at server side. In this way the server application will never know the IP address and port number of actual connected client because EMF CDH at server side always portray as a client. In the future an interface could be provided for the server applications to grab the IP address of actual connected end point.

In order to provide the reliable service, TCP keeps data in its send buffer until acknowledged from the other end. If on one end point, an abrupt link down occurred, the EMF CDH has to resend the data that is still unacknowledged on the old link. Being in user space, there does not exist any mechanism through that the unacknowledged data could be extracted from the TCP send buffer therefore CDH has to buffer at least as much data as in the TCP send buffer. This causes redundancy. This issue can be resolved if the EMF will be implemented in kernel space and have the access over TCP buffers.

### VII. Performance

In order to analyze the performance of this implementation with respect to protocol and computation overheads, a test bed on a WLAN based internetwork is established as shown in the Figure 9. The test bed includes two IEEE802.11b access points (APs), one multi-homed mobile device having connectivity with both APs. One stationary node that is connected with a router and that router is connected with both...
APs. The experiments involved FTP based file transfer client and server applications. Three scenarios for the file transfer were selected; file transfer over direct TCP connection without EMF support, file transfer with EMF support, file transfer with EMF support with handovers occurred two times and file transfer over two links with bandwidth aggregation enabled. For all experiments, the file size and network conditions remain the same and the total time consumed to transfer the file from one end to the other was studied.

Through the multiple experiments, it is shown that on the average the file transfer took 0.6% more time to complete when it is sent through the EMF as compared to the direct TCP connection. EMF induced overheads that could affect the performance of TCP connection are negligible. EMF data packets are of variable sized depending upon the size of data in the EMF_Send_Buffer. But maximum size of a data packet could be 32KB. If the average packet size is 15KB and with each packet a 12 bytes EMF header is attached which means that the protocol overhead involved is 0.08% only, which is negligible. Consequently, the computational overhead is 0.52% (aprox). Moreover when handovers occurred two times out of which one is willful and the other is due to link down, the file transfer took 7% on the average more time as compared to the direct TCP connection. This is due to the exchange of in-band control messages for handover and the slow start phases of new TCP connections. As the last experiment when the file is transferred end to end using two disjoint links with same bandwidth with bandwidth aggregation feature enabled, it took 40% lesser time to transfer the same file.

Table 1: Experiments and results (Average of 5 runs for each experiment)

<table>
<thead>
<tr>
<th>Experiment Description</th>
<th>File Size</th>
<th>Time Taken</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>File transfer without EMF support over single link</td>
<td>75MB</td>
<td>71.8 sec</td>
<td></td>
</tr>
<tr>
<td>File transfer with EMF support over single link</td>
<td>75MB</td>
<td>72.432 sec</td>
<td>0.6% increase</td>
</tr>
<tr>
<td>File transfer without EMF support over single link with two times handover</td>
<td>75MB</td>
<td>77.38 sec</td>
<td>7% increase</td>
</tr>
<tr>
<td>File transfer with EMF support over two links (BA)</td>
<td>75MB</td>
<td>43.2 sec</td>
<td>40% decrease</td>
</tr>
</tbody>
</table>

A user space implementation of End-to-End Mobility Management Framework is presented in this paper. The implementation was tested to be working well and portable to other platforms. Each module of EMF is discussed in detail along with the data structures and pseudo code. While implementing the framework, the portability issues were also resolved by introducing a glue layer. The experiments for performance analysis of the framework shows very encouraging results as there are negligible overheads being faced during the file transfer.

As a future work, it is planned to extend the implementation to port it on Windows Mobile and Android platforms in order to provide EMF support for handheld multi-homed mobile devices. Moreover, in order to make EMF interoperable with IEEE 802.21 (MIH) [13] standard, CL21 module will be replaced with MIH and Host Agent will be integrated with IEEE 802.21 (MIH) implementation.

REFERENCES