Mobility Control Protocol for MOFI

(Release 1.0)

September 2010

Ji In Kim and Seok Joo Koh

Kyungpook National University (KNU), sjkoh@knu.ac.kr

Summary

This document specifies the Mobility Control Protocol (MCP) for the Mobile Oriented Future Internet (MOFI). This study is motivated from the observation that future Internet would be evolved toward 'mobile-oriented' network environment, but current Internet was historically designed for 'static' network environment and thus it is inevitably subject to some architectural limitations in the viewpoint of 'mobile-oriented' future network. The MCP with MOFI is designed with the following principles: Host Identifier (HID) and Network Locator (LOC), ID-based global communication and LOC-based local routing, protocol separation for data delivery in the access and backbone networks, and functional separation for mobility control and data delivery. For data delivery, the Access Delivery Protocol (ADP) is used in the access network, whereas the current IPv4/IPv6 will be used as Backbone Delivery Protocol (BDP) in the backbone network. The MCP provides the mobility control functionality: HID binding, LOC binding and query, and handover control. The HID binding is used for a host to bind its HID with an access router, which is performed between hosts and access routers. The LOC Binding is used for Access Control Agent (ACA) to bind the mapping information between HID and LOC with LOC Management System (LBS). The LOC Query is used for ACA to obtain the LOC information for a HID from LBS. To support the handover control, LOC Update operation is performed to update a new LOC between a local ACA and a remote ACA, when a host moves into a new AR region. Some of implementation issues are discussed.

TABLE OF CONTENTS

1. DESIGN CONSIDERATIONS

1.1 DESIGN PRINCIPLES

The Mobility Control Protocol (MCP) for MOFI is designed with the following principles.

1.1.1 HOST IDENTIFIERS AND NETWORK LOCATORS

Based on the MOFI architecture [1], the MCP is designed with the host identifier (HID) and network locator (LOC).

For delivery of data packets, a host is identified by HID in a static and secure manner. At present, we consider the format of 128-bit HID. This may be similar to the Host Identity Tag (HIT) of Host Identity Protocol (HIP) [2], which generated based on the Overlay Routable Cryptographic Hash Identifiers (ORCHID) [3]. In particular, it is required that a HID should be specified in the hierarchical format for scalable management of HIDs in the world-wide scale, which may include the information of network domain identifier such as Autonomous System (AS) number of a domain. This is still for further study.

Locator (LOC) is used to represent the location of an object in the network. LOC is also used for delivery of data packets between objects in the network. In MOFI, an 'IP address' of the access router (attached to the end host) is used as LOC of the end host. The LOC is used for packet routing only in the backbone network, whereas the HID is used for communication only in the access network. In mobile environments, a host may change its LOCs by movement, but its HID will not change.

1.1.2 ID-BASED COMMUNICATIONS AND LOC-BASED ROUTING

In MOFI, the end-to-end communication between two end hosts will be performed by using HIDs. Each HID shall be globally unique in the network, and it will use an appropriate socket interface that consists of a HID and a port number.

On the other hand, the routing for packet delivery will be done only using the LOCs, i. e., IP address of the access router. The LOCs may be locally used in the backbone network. Furthermore, one or more LOCs and different routing schemes may be used, if the data packets are delivered to the final end hosts through several backbone network domains.

1.1.3 Protocol Separation for Data Delivery in Access and Backbone Networks

In MOFI, the data delivery protocols used in access and backbone networks will be separated from the MCP which is specified in this document. In particular, it is expected that there are many kinds of heterogeneous access networks in the future Internet environment. Each access network may have quite different characteristics such as bandwidth and delay, and hence the protocols used for data delivery may also be different. To support these heterogeneous access networks, we will separate the protocols used for data delivery into Access Delivery Protocol (ADP) and Backbone Delivery Protocol (BDP). Each access network may use its own ADP and routing schemes.

In the mean time, we will use the current IPv4/v6 protocols for data delivery in the backbone network, which is taken as an incremental approach to deployment of future Internet. This is because the backbone network is quite difficult to replace with a completely new protocol at a stretch, compared to the access network. This approach will also be helpful for migration from the current Internet to the clean-slate future Internet.

1.1.4 Functional Separation for Mobility Control and Data Delivery

We note that the control information (e.g., for mobility control) is usually mission-critical and thus requires the fast and reliable delivery in the network. Accordingly, the control plane (or function) needs to be performed differently from the user data plane (function), as done in the current cellular systems.

In particular, MCP will be designed for network-based mobility control, since the network-based mobility scheme is preferred to the host-based mobility scheme in the viewpoint of resource utilization, protocol performance and deployment, as shown in the comparison of MIPv6 [4] and Proxy MIPv6 (PMIPv6) [5]. In addition, the mobility control of MCP will provide the 'intrinsic' route optimization, by which the direct path should be used between two hosts from the beginning of communication with the help of the mobility control.

1.2 FUNCTIONAL ARCHITECTURE

We consider the following functional reference model of MOFI/MCP, as shown in the figure below.

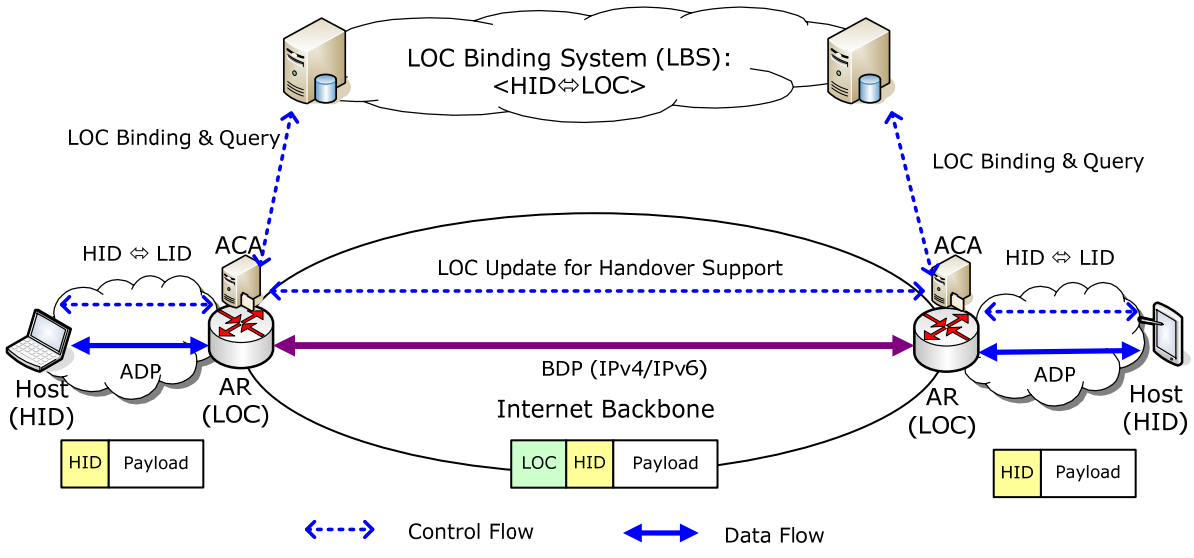

Figure 1 – Functional reference model of MOFI

1.2.1 HOSTS

A host has one or more Link Identifiers (LIDs), depending on whether the host has a single or multiple network interfaces for host multi-homing. We consider Link ID (LID) to identify a link (connection point or interface) of a host to the access network, which is needed for data packet delivery between the access router and a host. Examples of LID include the IEEE 802 MAC address and any other link-layer physical addresses of hosts for wired/wireless network interfaces.

An LID is given for each network interface of a host, and thus a multi-homing host may have two or more LIDs. The specific format of LID depends on the associated link-layer access technologies. At present, we consider the following LIDs:

- IEEE 802 MAC address (for LAN or WLAN);
- GPRS Tunnel ID (for cellular networks);
- IPv6 or IPv4 address (for overlay network).

In a certain case, a LID may consist of a sequence of two or more IDs for an access router to identify the host in the access network (for example, one or more PoAs are located between host and access router).

For communication, a host shall be attached to an AR in the network. When a host is attached to an AR, its HID and LID(s) shall be registered with AR. This operation is called 'HID Binding.'

1.2.2 ACCESS NETWORK AND ACCESS ROUTER (AR)

In future Internet, there may be a variety of wired/wireless access networks between hosts and AR. Depending on the underlying access network, one or more Point of Attachments (PoAs) or routers may be located between host and AR, as shown in the example of wireless ad hoc, sensor, or wireless mesh networks.

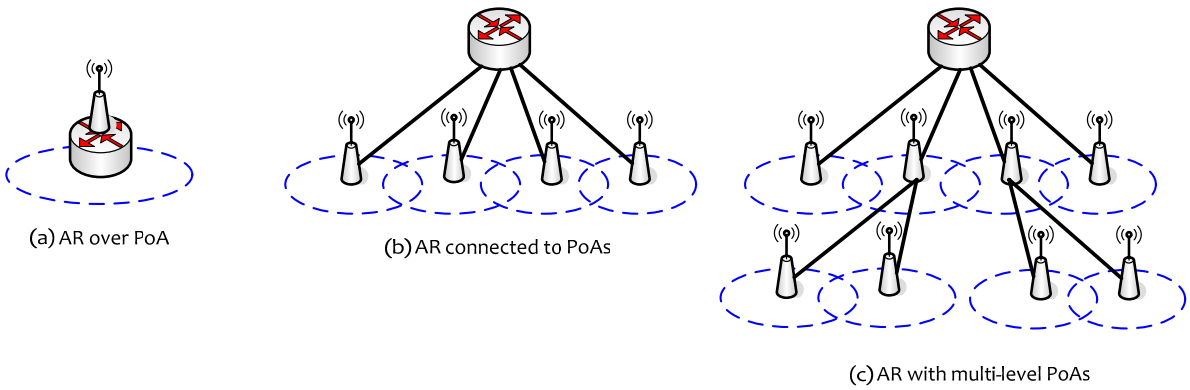

Figure 2 – Configuration of AR and PoA

In the deployment perspective of an access network, as shown in Fig. 2, an AR may be located over the underlying link-layer Point of Attachment (PoA), in which host is directly connected to AR in the link layer (Fig. 2(a)). Alternatively, one or more PoAs may be connected to the AR, as

shown in Fig. 2(b). In this case, a host communicates with AR via one of candidate PoAs, in which the mobility of host across two neighbouring PoAs will be managed by the corresponding link-layer mobility management scheme, which is outside the scope of MCP. In a certain case, a tree hierarchy of PoAs with two or more levels may be configured in the network, as seen in Fig. 2(c) such as the wireless mesh networks of IEEE 802.11s.

With network attachment, a host shall bind its HID and LID to AR via the HID Binding operation. After that, the host sends or receives data packets to or from AR by using the Access Delivery Protocol (ADP). That is, AR receives the data packets from its local host and forwards them to the correspondent host in the network by using the legacy IPv4/IPv6 protocols. When AR receives data packets from the remote AR, it will deliver those packets to the local hosts by using the ADP.

In the perspective of mobility control, AR uses its Access Control Agent (ACA), which will be described later.

1.2.3 BACKBONE NETWORK

In MCP, the backbone network represents the legacy Internet that consists of a lot of site networks, ISPs, and network providers. In the Internet backbone network, the data delivery mechanisms will follow the currently used IPv4/IPv6 protocols.

1.2.4 ACCESS CONTROL AGENT (ACA)

ACA is an agent located with AR, which is in charge of control operations such as LOC binding and query. Each ACA is likely to be implemented with AR. That is, AR and ACA may be just logically (or functionally) separated, but physically co-located over the same equipment, as shown in the example of MSC and VLR in the GSM system.

Each ACA performs LOC binding and query operations with LBS by using the MCP, which is used to get the information of LOC for data delivery. MCP is also used for handover control of mobile hosts, which is performed between ACAs.

1.2.5 LOC BINDING SYSTEM (LBS)

The LBS provides a database for binding between HID and LOC. When LBS receives an LOC binding request message from an ACA, it creates or updates the HID-LOC binding information in the database. In addition, when an ACA asks LBS for the HID-LOC binding information to a specific HID, LBS will reply with the corresponding LOC information to the ACA. LBS may maintain a database that contains information about users' service profile and authentication.

The LBS is a logical overlay network that may consist of a lot of distributed servers in the world-wide scale. For load sharing and/or scalability enhancement of control operations, the LBS may be configured in the hierarchical or distributed manner. In addition, for scalable management of LBS system in the mobile environments, Home LBS and Visited LBS may be used, as seen in the example of Home Location Register (HLR) and Visited Location Register (VLR) in the GSM system.

(Note) For scalable LBS management, a hierarchical format of HID shall be considered.

2. DATA DELIVERY MODEL

The MCP is designed with the HID-based end-to-end data communications, in which the data delivery will be done, based HID between the two end hosts.

2.1 PROTOCOL FOR DATA DELIVERY

The MCP considers a protocol stack for data transport, as shown in the figure below.

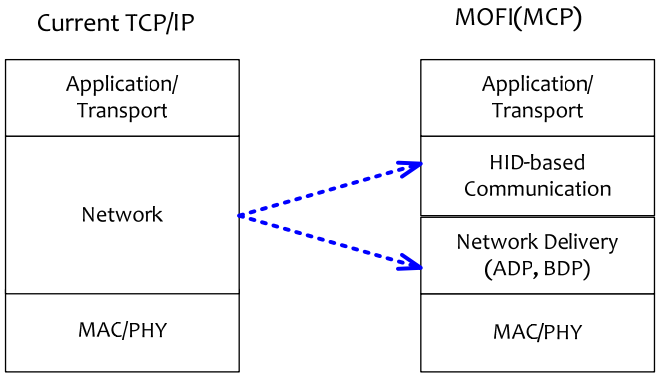

Figure 3 – Comparison of protocol stacks for data delivery: TCP/IP versus MOFI(MCP)

In MCP, the network layer is divided into the two sub-layers: HID-based communication sub-layer and network delivery sub-layer. HID Communication sub-layer is newly defined for end-to-end communication based on HID. This is responsible for end-to-end user data communications between two end hosts, which include interaction with transport layer protocols (TCP, UDP) and the provisioning of socket interface with applications.

The routing protocols for data delivery are divided into ADP and BDP. The current IP is used as BDP for data delivery over Internet, whereas ADP is used for routing in access networks.

Fig. 4 summarizes the protocol stacks associated with data delivery between two end hosts.

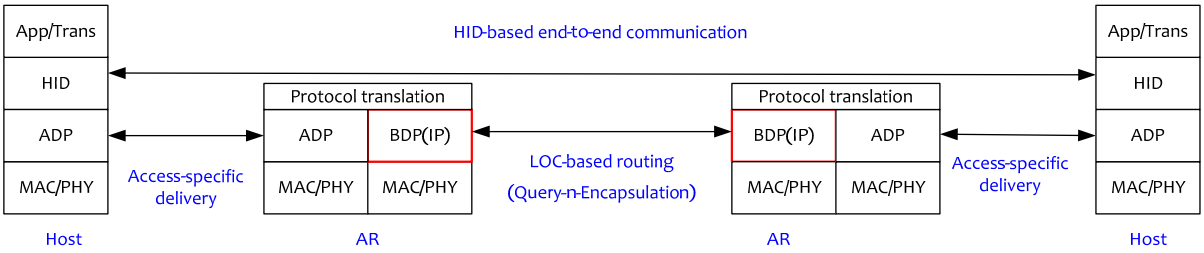

Figure 4 – Data delivery model

HID is used for end-to-end communication between two end hosts. In data delivery, each AR will translate the protocols between ADP and BDP by referring to the HIDs of data packets.

2.1.1 ACCESS DELIVERY PROTOCOL (ADP)

ADP is responsible for data delivery between host and AR. ADP is specific to the underlying access network technology, and the access network may be one-hop or multi-hop networks. In case of multi-hop access networks, the LID will be used as an 'access locator' for data packet delivery between host and AR, possibly via one or more PoAs in the access network.

In a certain access network, ADP may be omitted between host and AR, if a host is directly connected to AR with a single hop (e.g., in the IEEE 802.3 access networks). In this case, the LID shall be used for data delivery between host and AR.

More detailed operations and packet formats of ADP are outside the scope of this document.

2.1.2 BACKBONE DELIVERY PROTOCOL (BDP)

In MOFI, the current IPv4/IPv6 routing/forwarding schemes are used as BDP at present. Thus, the semantics and operations of BDP will follow the current IPv4/IPv6.

2.2 DATA TRANSPORT PROCEDURES

In MOFI, data communications will be essentially performed only with HID, not LOC (IP address). When a host will initiate a communication session with the corresponding host, it shall know the HID of the corresponding host.

For discussion, we consider a simple communication scenario in which a sending host (SH) wants to communicate to a receiving host (RH). If SH knows the HID of RH, the overall data transport operations are performed, as shown in the figure below.

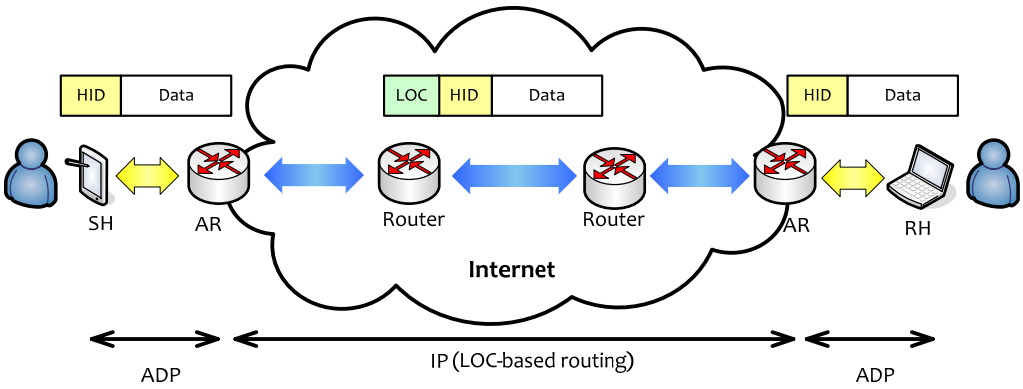

Figure 5 – Data transport procedures in MCP

The data packet transport operations are summarized as follows:

1) Data transmission (SH ⇔ AR)

   SH sends the data packets to AR by using HID of RH. It is noted that SH does not need to know the LOC of RH (that is, the IP address of AR that is attached to RH).

2) Query and encapsulation (AR of SH)

   On reception of data packets from SH, the AR of SH will first identify the LOC of RH by using the MCP protocol, which will be described later. Then, AR of SH will encapsulate the data packets by adding the LOC of RH (IP address of AR of RH) into the outer header of the data packet.

3) Packet delivery in the backbone network

   The encapsulated data packets are delivered from AR of SH to AR of ER by using the current IP routing/forwarding, possibly via one or more routers in the backbone network.

4) Decapsulation (AR of RH)

   On reception of the encapsulated data packets, the AR of RH will extract the original data packets by decapsulation, and then forward them to RH.

5) Data reception (AR ⇔ RH)

   Finally, the RH can receive the original data packets transmitted by SH.

3. PROCEDURES

In MCP, the following mobility control operations are supported:

- HID Binding (HB) between host and AR;
- LOC Binding (LB) and LOC Query (LQ) between ACA and LBS;
- Handover Control (HC) between ACAs.

The HB operation is used to bind the LID and HID of a host with AR, whereas LB and LQ operations are used to bind the LOC information of a mobile host to LBS and for a correspondent host to query the HID of the mobile host, respectively. The HC operation is used to support seamless handover when a mobile host moves on during communications.

3.1 HID BINDING (HB)

3.1.1 OPERATIONS

When a host is attached to the network, it will establish the network connection with the concerned PoA via an appropriate link-layer connection establishment process. In this network attachment process, a certain authentication and/or authorization may be performed between a user and service provider, which is outside the scope of the MCP.

With the network attachment of host, the HID binding operation is required, in which HID and LID of the host will be registered with AR attached to the host. AR will maintain and update its cache table, called "HID Cache" (HC) that contains the information of bindings between HIDs and LIDs for all of the hosts attached to the AR.

In MCP, we assume that the HID binding will be performed with the underlying 'access link layer' protocol, implicitly or explicitly, as shown in the figure below.

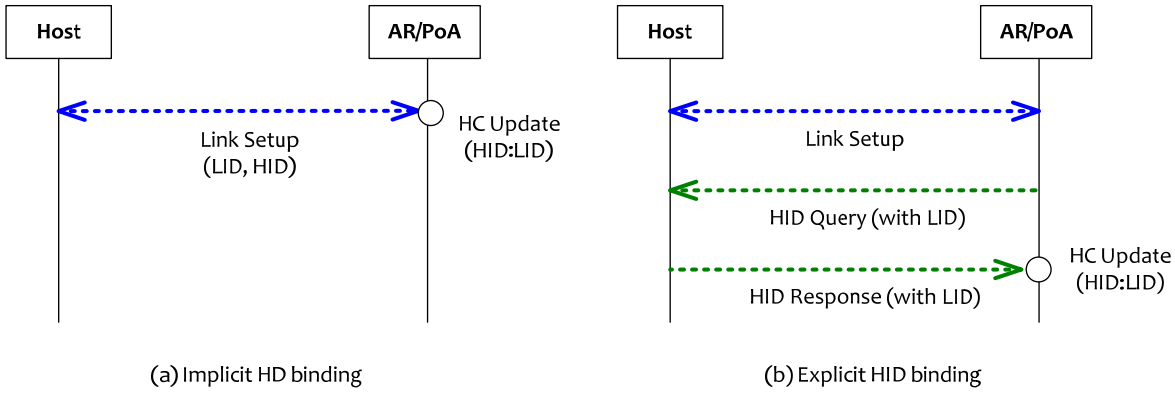

Figure 6 – HID binding between host and AR/PoA

In the implicit HID binding, the binding between HID and LID will be done during the underlying link-layer attachment process. In the explicit HID binding, after the link setup, the AR may request the HID information to the host explicitly by sending the HID query message (in multicast or broadcast), and then the host will respond with its HID to AR (in unicast).

3.1.2 HID CACHE (HC)

In the HID binding operation, AR maintains and updates its own HID cache by recording the HIDs and LIDs associated with the hosts in the local subnet. The abstract format of the HID cache table is shown below.

Table 1 – HID Cache (HC)

No.	HID	LID	Link Information	Status	Type
1	HID1	LID1	…	Idle	Static
2	HID2	LID2	…	Active	Static
3	HID3	LID3	…	Idle	Mobile
4	HID4	LID4	…	Active	Mobile
5	…	…	…	…	…

The cache information will be referred to by AR for delivery of the data packets that are destined to the hosts in the local subnet. In the table, the status field (idle or active) represents whether or not the host is in the active data communication with a certain other corresponding host. That is, 'active' means the HID is bound to the network and also in communication with the other host(s), while 'idle' implies that the host is bound but not in communication. In addition, the HC cache may indicate the type of host (static or mobile), and the information associated with the link may be used, (e.g., information of contact with the PoA of host, etc), which is for further study.

3.2 LOC BINDING (LB) AND LOC QUERY (LQ)

3.2.1 PROTOCOL MODEL

When a host is connected to AR, the corresponding ACA will bind the LOC and HID of host to LBS. For data delivery, each ACA will contact with LBS so as to get the LOC information of the corresponding host. Fig. 7 shows the protocol stack associated with LOC binding and query between ACA and LBS.

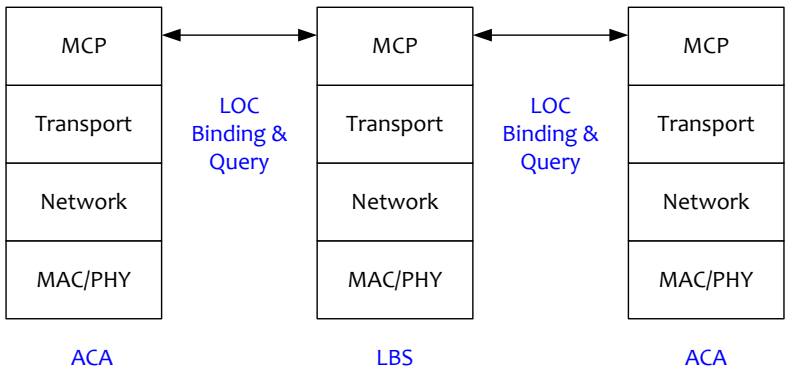

Figure 7 – Protocol model for LB and LQ between ACA and LBS

3.2.2 LB OPERATIONS

When an AR detects a new host in its network region, its associated ACA shall perform the LB operation by sending a LB Request (LBR) message to the LBS. The LBR message shall include the HID and LOC of the host. LBS responds with the corresponding LB ACK (LBA) message to ACA. This LB operation will be performed each time a host moves into a new AR area, as shown in the figure below.

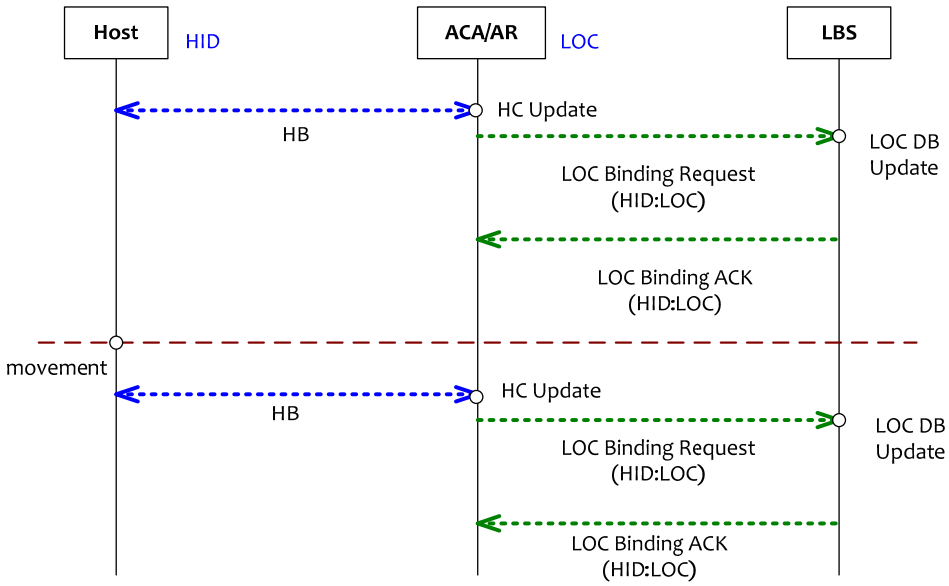

Figure 8 – LOC Binding Operations

3.2.3 LBS WITH LOC DATABASE (DB)

LBS is used to manage the LOC information of all of the users in the network. For this purpose, LBS maintains the location database, which contains information of mapping between HIDs and LOCs for all of the hosts.

When a host moves into a new AR, its ACA will perform the LOC Binding (LB) operation with LBS. From this LB operation, HID and LOC of a host will be registered with LBS. Accordingly, LBS will maintain the following Location DB, as shown in the table below.

Table 2 – LBS LOC Database

No.	HID	LOC	Service Profile
1	HID1	LOC1	Related data
2	HID2	LOC2	Related data
3	…	…	…

3.2.4 ACA WITH LOC CACHE (LC)

For data delivery, each ACA performs the LOC Query (LQ) operations with LBS. When a host wants to communicate with another host, the associated ACA/AR will perform the LQ operation with LBS so as to get the LOC of the corresponding HID. From this LQ operation, each ACA maintains the LOC Cache (LC) for each of the corresponding (remote) HIDs, as shown in the table below.

Table 3 – ACA LOC Cache (LC)

No.	Remote HID	Remote LOC
1	HID1	LOC1
2	HID2	LOC2
3	…	…

3.2.5 LQ OPERATIONS

Now, let us assume that the corresponding host has completed its LB operation. For example, we consider Sending Host (SH) with S-HID that sends data packets to the Receiving Host (RH) with R-HID. In this phase, the AR of SH needs to perform the LQ operation with LBS. An example of data delivery from SH to RH is illustrated in the figure below.

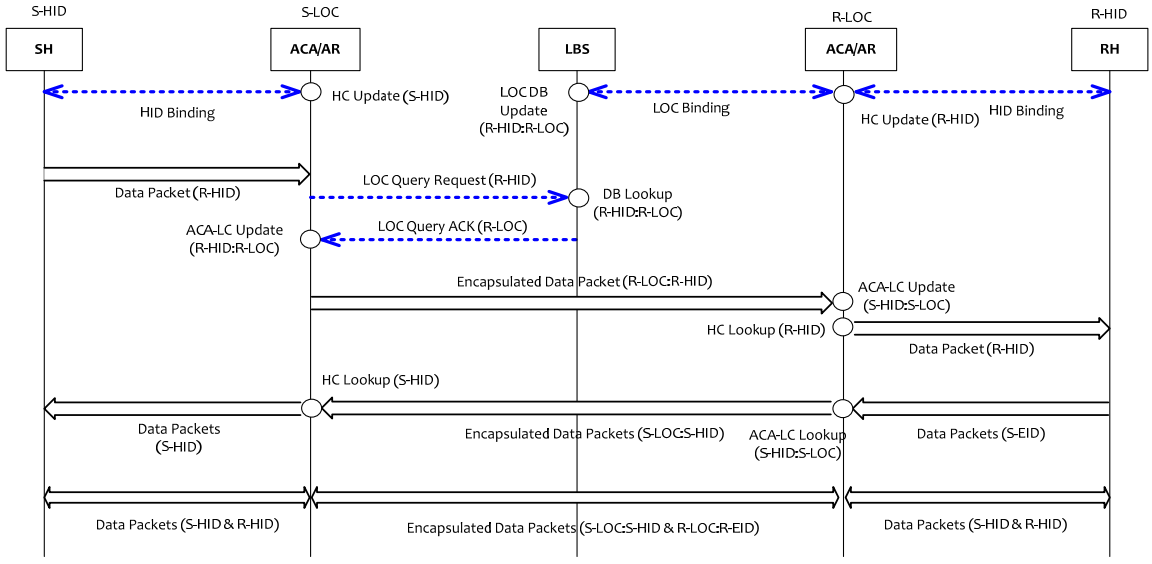

Figure 9 – Location Query for Data Delivery

The data delivery procedures could be summarized as follows:

1) Initially, both SH and RH are bound to S-AR and R-AR by using the respective HBP protocols, in which the associated HBP caches are updated with S-HID:S-LID and R-HID:R-LID. It is also assumed that the LB operations for R-HID and R-LOC have been completed between R-ACA and LBS.
2) SH sends an initial data packet to RH via its attached S-AR;
3) ACA/AR of SH will first look up its LOC Cache (LC) table to find the LOC of R-HID; if yes, S-AR can deliver the data packet to the identified R-AR (R-LOC), which is not shown in the figure.
4) If ACA/AR of SH cannot find the LOC of R-HID in the ACA-LC table, it shall perform the LOC Query operation by sending an LQR message to LBS. In response to the LQR message, the LBS will lookup its Location DB to identify the R-LOC of R-HID, and then it sends the LQA message to the ACA of SH. Based on the received LQA message, ACA of SH will update its LC table by creating the entry for R-HID and R-LOC;
5) AR of SH will send the encapsulated data packets to the AR of RH (R-LOC);
6) On reception of the encapsulated data packets from AR of SH, the AR of RH extracts the original data packets from the encapsulated packets. ACA of RH will update its LC table by creating a new entry for S-HID and S-LOC. This is done for AR of RH to deliver the data packets from RH to SH in the future.
7) Then, AR of RH forwards the data packets to RH. To do this, ACA of RH will lookup its HBP cache to identify the LID of R-HID.
8) Up to now, the ACA-LC of SH and ACA-LC of RH have been constructed. Based on this information, RH can also send data packets to SH. That is, RH (R-HID) and SH (S-HID) can now exchange data packets by referring to the established HBP caches and ACA-LCs.

The following figure shows the abstract information flows of LB and LQ in the previous example.

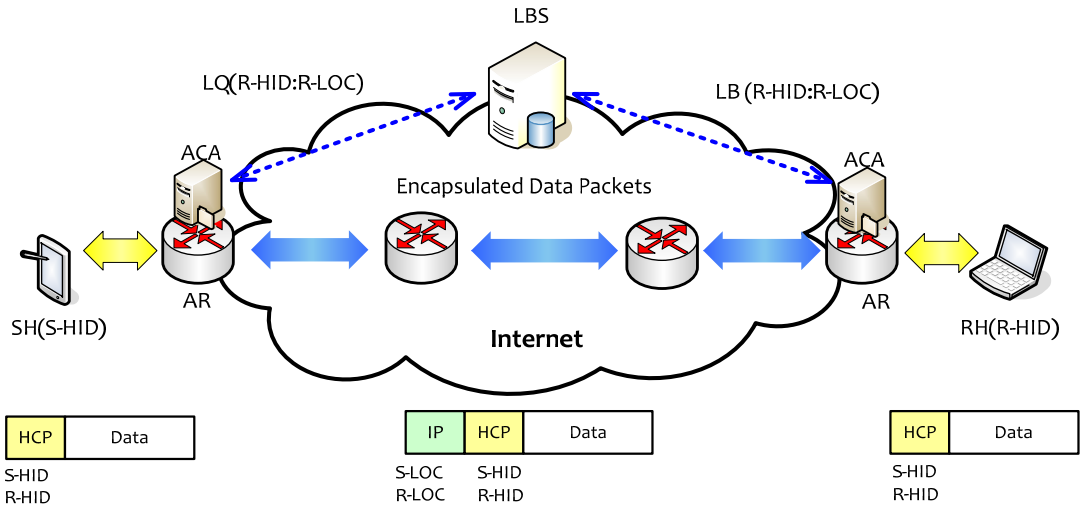

Figure 10 – Location Binding and Query for data delivery

For data transport, the ACA-LC maintains the list of the corresponding (remote) hosts that the local hosts are communicating with. On the other hand, HBP cache is used to forward the data packets to the attached local hosts.

3.3 HANDOVER CONTROL

For handover control, MCP uses the link-layer information such as Link-Up (LU), Link-Down (LD), Link-Going-Down (LGD), and Link Coming-Up (LCU), which are defined in the IEEE 802.21 MIH. At present, we will focus on the LU trigger only. The other triggers can also be examined to provide seamless handover.

3.3.1 PROTOCOL MODEL

MCP is also used to support the fast handover for mobile hosts. Note that the MCP operates based on the IEEE 802.21 Media Independent Handover (MIH) between host and AR, in which appropriate link-layer triggers will be used such as Link-Up, Link-Down, Link-Coming-Up, and Link-Going-Down.

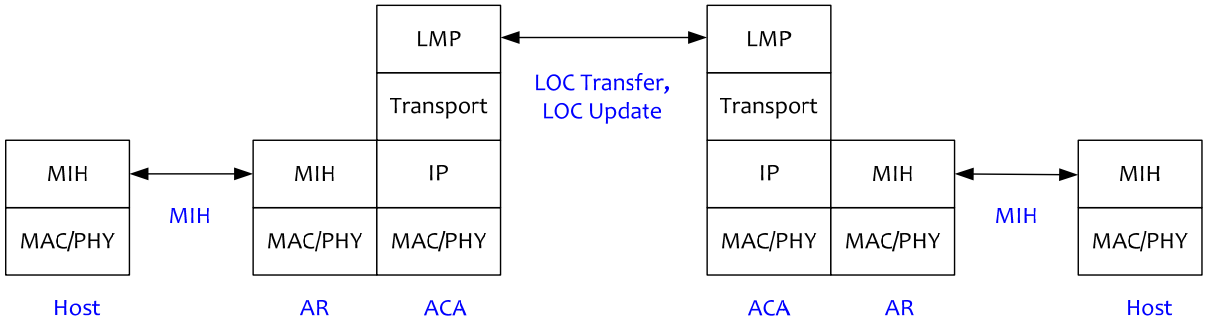

Figure 11 – Handover Control Protocol with MIH

For handover control, the LOC Transfer (LT) operation is performed between the two neighbouring ACAs associated with mobile host, and the LOC Update (LU) operation is used between local ACA and remote ACA to update the LOCs in communication. The details of handover control will be described in the subsequent section.

3.3.2 OPERATIONS

We first consider the following simple handover scenario in which Mobile Host (MH) is communicating with Correspondent Host (CH), and it is moving from AR_old to AR_new.

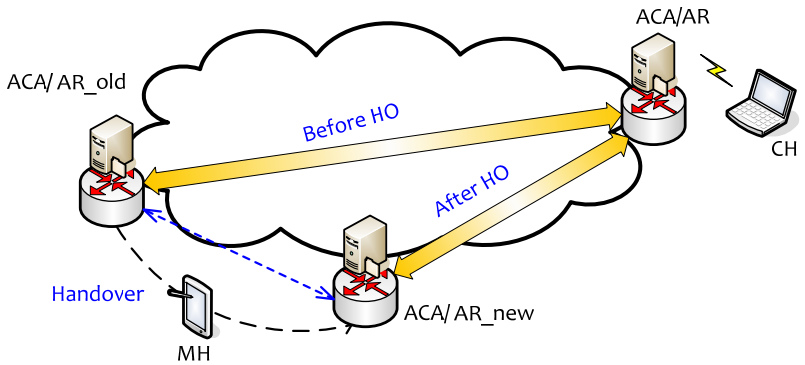

Figure 12 –Handover scenario

Based on the handover event, the handover control operations will be performed as follows.

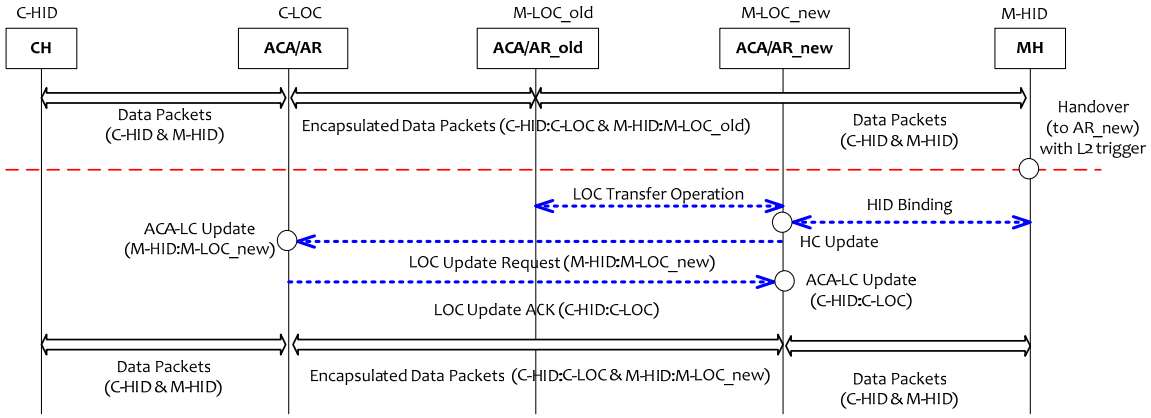

Figure 13 – Handover control by MCP

1) We assume that CH and MH (in AR_old) are communicating before handover. By handover, ACA_old (or ACA_new) of MH will get an LU trigger from the network. After that, ACA_old will exchange LOC Transfer Request (LTR) and LOC Transfer ACK (LTA) messages with ACA_new for handover control, which shall include the information of M-HID, C-HID, C-LOC.
2) AR_new of MH performs HBP with the newly attached MH, and the AR_new will update its HBP cache table for MH.
3) Now, the ACA_new of MH sends the LOC Update Request (LUR) message to the ACA of CH. On reception of the LUR message, ACA of CH will update its LC table with M-HID:M-LOC_old to M-HID:M-LOC_new.
4) In response to the LUR message, the ACA of CH will send the LOC Update ACK (LUA) message to the ACA_new of MH. On reception of the LUA message, the ACA_new updates its LC table by creating the C-HID:C-LOC entry.
5) The data path is now changed to CH ↔ AR of CH ↔ AR_new of MH ↔ MH.

In addition, the ACA will perform the LB operation after handover, which is not shown in the figures. This LB operation is for the newly incoming session to MH, which is performed independently of the handover control.

4. PACKETS

4.1 DATA PACKET

In MOFI/MCP, the HID is used for end-to-end communication between two hosts. This is NOT for data delivery or routing. That is, the HID is not used for routing of data packets in the network. Instead, it will be used for end-to-end communication using the upper layer transport layer protocol (TCP/UDP) and a socket interface with an application program.

4.1.1 HEADER FORMAT

For design of data packet header format, we refer to the current IPv6 header format for backward compatibility. Instead of the conventional IP header, each data packet has the following header.

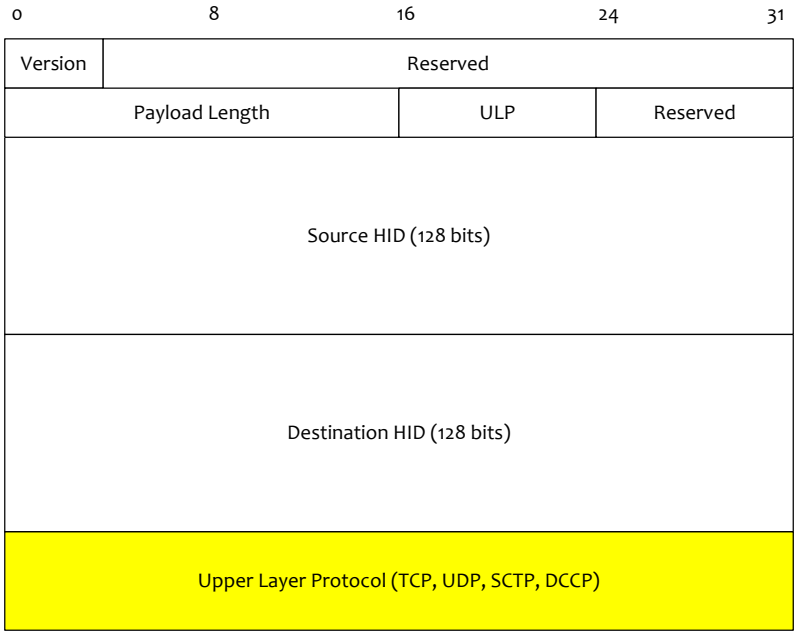

Figure 13 – Header Format of Data Packet

Packet Field Description:

- Version (4 bits): indicates this version of MOFI, which may be used to make it compatible with the current IP version;
- Reserved (12 bits): reserved for future use, and set to 0;
- Payload Length (16 bits): the length of user payload (in byte) following this FIP header;
- Upper Layer Protocol (8 bits): indicates the upper layer protocol such as TCP, UDP, etc.
- Reserved (8 bits): reserved for future use, and set to 0;
- S-HID (128 bits): Source HID
- D-HID (128 bits): Destination HID.

4.1.2 ENCAPSULATION

For data delivery, the data packet will be encapsulated into the ADP in the host side and the BDP in the AR side for data delivery, as shown in the figure below. The ADP header contains the Link IDs of host and AR, whereas IP (BDP) header includes the two IP addresses (LOCs) of ARs, as done in the current IPv4/IPv6.

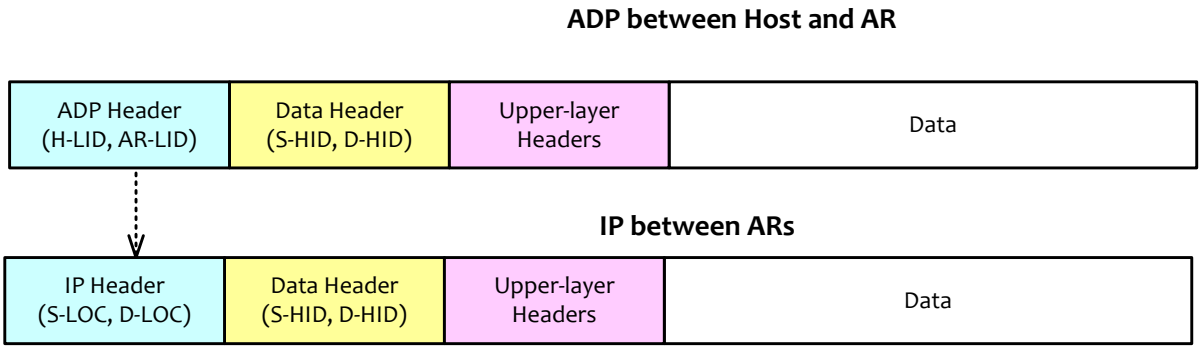

Figure 14 – Structure of data packets

4.2 MCP PACKETS

4.2.1 PACKET FORMAT

The overall packet structure of MCP packet is shown below.

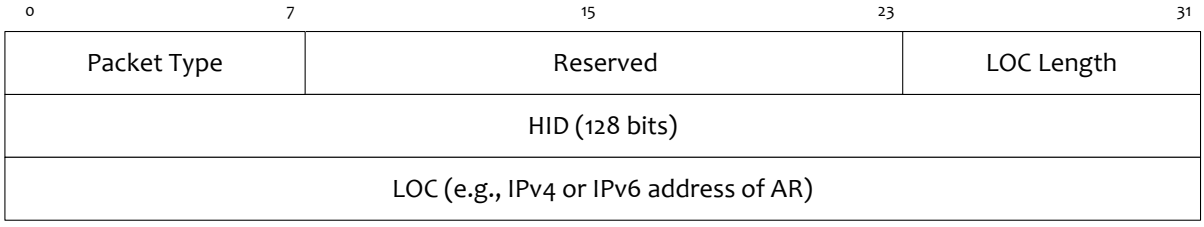

Figure 15 – Abstract Format of MCP Packets

Packet Field Description:

- Packet Type (8 bits): indicates the type of MCP packet, which will be described later;
- Reserved (16 bits): reserved for future use;
- LOC Length (8 bits): the length of LOC in byte (4 for IPv4, 16 for IPv6);
- HID (128 bits): Host ID;
- LOC (4 or 16 bytes): Locator (IPv4 or IPv6 address of AR).

4.2.2 PACKET TYPES

The following table shows the list of the packets used for MCP.

Table 4 – Types of MCP Packets

Packet Type	Full Name	From	To
LBR	LOC Binding Request	ACA	LBS
LBA	LOC Binding ACK	LBS	ACA
LQR	LOC Query Request	ACA	LBS
LQA	LOC Query ACK	LBS	ACA
LTR	LOC Transfer Request	ACA	ACA
LTA	LOC Transfer ACK	ACA	ACA
LUR	LOC Update Request	ACA(local)	ACA(remote)
LUA	LOC Update ACK	ACA(remote)	ACA(local)

5. IMPLEMENTATIONS

5.1 NETFILTER

Netfilter is a framework that provides hook handing for packet mangling, outside the normal Berkeley socket interface. The netfilter is invoked, for example, by the packet reception and send routines from/to network interfaces. As the master netfilter function, which is nf_hook(), is called with a packet, netfilter run through the list of registered hooks and calls the extensions in succession, which then handle packets as they desire. And that consists of four parts.

Firstly, each protocol defines "hooks" which are well-defined points in a packet's traversal of that protocol stack. At each of these points, the protocol will call the netfilter framework with the packet and the hook number.

Secondly, parts of the kernel can register to listen to the different hooks for each protocol. So when a packet is passed to the netfilter framework, it checks to see if anyone has registered for that protocol and hook; if so, they each get a chance to examine and possibly after the packet in order, then discard the packet (NF_DROP), allow it to pass (NF_ACCEPT), tell netfilter to forget about the packet (NF_STOLEN), or ask netfilter to queue the packet for userspace (NF_QUEUE).

The third part is that packets that have been queued are collected by the ip_queue driver for sending to userspace which these packets are handled asynchronously.

The final part consists of cool comments in the code and documentation. This is instrumental for any experimental project. In addition to this raw framework, various modules have been written which provide functionality similar to previous kernels, in particular, an extensible NAT system, and an extensible packet filtering system (iptables).

5.1.1 NETFILTER ARCHITECTURE

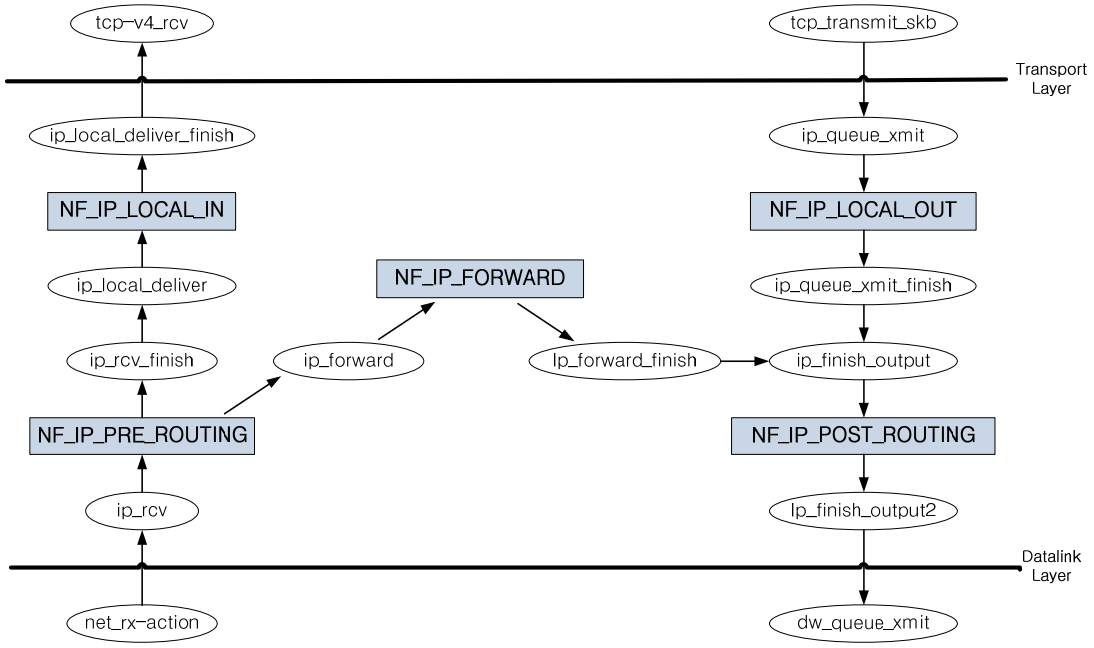

Figure 16– Netfilter Architecture in Linux Kernel

The Netfilter has five hooks in Linux kernel which are NF_IP_PRE_ROUTING, NF_IP_LOCAL_IN, NF_IP_FORWARD, NF_IP_LOCAL_OUT, NF_IP_POST_ROUTING.

Firstly, on the left is where packets come in ip_rcv that have passed the simple sanity checks, for example, not trusted, IP checksum OK, not a promiscuous receive, they are passed to the netfilter framework's NF_IP_PRE_ROUTING hook.

Next, they enter the routing code, which decides whether the packet is destined for another interfaces, or a local process. The routing code may drop packet that are unroutable.

If it is destined for the box itself, the netfilter framework is called again for the NF_IP_LOCAL_IN hook, before being passed to the process.

If it is destined to pass to another interface instead, the netfilter framework is called for the NF_IP_FORWARD hook.

The packet then passes a final netfilter hook, the NF_IP_POST_ROUTING hook, before being put on the wire again.

The NF_IP_LOCAL_OUT hook is called for packets that are created locally. Here you can see that routing occurs after this hook is called, in fact, the routing code is called first to figure out the source IP address and some IP options. If we want to alter routing, we must alter the 'skb->dst' field ourselves, as is done in the NAT code.

Table 5 – Netfilter Hooks

Hook	Description
NF_IP_PRE_ROUTING	Process all packet received network interface
NF_IP_LOCAL_IN	Only process packet to be sent to local machine
NF_IP_FORWARD	Only process forwarding packet
NF_IP_LOCAL_OUT	Only process packet to send another machine
NF_IP_POST_ROUTING	Process all packet to send another machine including forwarding packet

5.1.2 NETFILTER PACKET FILTERING TYPE

Now we have an example of netfilter for IPv4, we can see when each hook is activated. This is the essence of netfilter.

Kernel modules can register to listen at any of these hooks. A module that registers a function must specify the priority of the function within the hook. And then, when the netfilter hook is called from the core networking code, each module registered at that point is called in the order of priorities, and is free to manipulate the packet. The module can then tell netfiter to do one of five things.

1) NF_ACCEPT: Continue traversal as normal.
2) NF_DROP: Drop the packet. Do not continue traversal.
3) NF_STOLEN: Have taken over the packet. Do not continue traversal.
4) NF_QUEUE: Queue the packet (usually for userspace handling).
5) NF_REPEAT: Call this hook again.

5.1.3 HOW TO USE NETFILTER

The netfilter is kernel module. So, we register the netfilter that is needed to use the init_module() function. And we unregister the netfilter that is needed to use the cleanup_module(). There are some examples are shown below.

```
int init_module(void)
{
    int result;

    /* input hook */
    input_filter.list.next = NULL;
    input_filter.list.prev = NULL;
    input_filter.hook = input_handler;
    input_filter.flush = NULL; /* still unused */
    input_filter.pf = PF_INET; /* IPv4 */
    input_filter.hooknum = NF_IP_LOCAL_IN;

    /* output hook */
    output_filter.list.next = NULL;
    output_filter.list.prev = NULL;
    output_filter.hook = output_handler;
    output_filter.flush = NULL; /* still unused */
    output_filter.pf = PF_INET; /* IPv4 */
    output_filter.hooknum = NF_IP_LOCAL_OUT;
```

Figure 17– init_module() function

Figure 17 shows the example of init_module() function. In general, there are two hooks in the init_module() function, because of separating input and output hook. First, we define the hook name is likely to 'input_handler' and 'output_handler' in the figure. And then, we define the protocol family at the pf. Finally, the most important is hook number that determine the hook what we use, for example, NF_IP_LOCAL_IN and NF_IP_LOCAL_OUT in the figure. The hook number is defined in advance at the netfilter.h that exist in '/usr/src/linux-2.6.32.16/include/linux/' directory.

```
void cleanup_module(void)
{
    /* unregister hooks */
    nf_unregister_hook(&input_filter);
    nf_unregister_hook(&output_filter);
```

Figure 18– cleanup_module() function

Figure 18 shows the example of cleanup_module() function. The nf_unregister_hook() function unregister the netfilter hooks that are registered in the init_module() function.

5.2 IMPLEMENTATION ENVIRONMENTS

Table 6 – Implementation Environments

Subject	Version
Operating System	Ubuntu Linux 9.04 desktop Binding Request
Kernel	Linux kernel 2.6.32.16
Compiler	GCC 4.3.3

To implement the MOFI Mobile Control Protocol, we use the operating system which is the Ubuntu linux 9.04 desktop edition i386 version. And we used the linux kernel 2.6.32.16 that was the latest stable kernel when we started this implementation. The compiler is the GNU Complier Collection (GCC) compiler of which version is 4.3.3. And we make the make file to build kernel module because that need to add the kernel library function. That is shown below.

```
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

Figure 19– Makefile to build kernel

And we use the socket buffer functions in the skbuff.h that exists in '/usr/src/linux-2.6.32.16/include/linux' directory. For example, skb_push() function is used. It adds data to the start of a buffer and extends the used data area of the buffer at the buffer start. If this would exceed the total buffer headroom the kernel will panic. A pointer to the first byte of the extra data is returned.

```
extern unsigned char *skb_push(struct sk_buff *skb, unsigned int len);
static inline unsigned char *__skb_push(struct sk_buff *skb, unsigned int len)
{
    skb->data -= len;
    skb->len  += len;
    return skb->data;
}
```

Figure 20– skb_push() function

And skb_pull() is remove the data from the start of a buffer, returning the memory to the headroom. A pointer to the next data in the buffer is returned. Once the data has been pulled, future pushes will overwrite the old data.

```
extern unsigned char *skb_pull(struct sk_buff *skb, unsigned int len);
static inline unsigned char *__skb_pull(struct sk_buff *skb, unsigned int len)
{
    skb->len -= len;
    BUG_ON(skb->len < skb->data_len);
    return skb->data += len;
}
```

Figure 21– skb_pull() function

Another function is skb_copy_expand() which makes a copy of both an socket buffer and its data and while doing so allocate additional space. This is used when the caller wishes to modify the

data and needs a private copy of the data to alter as well as more space for new fields. Returns NULL on failure or the pointer to the buffer on success. The returned buffer has a reference count of 1.

```
extern struct sk_buff *skb_copy_expand(const struct sk_buff *skb,
                       int newheadroom, int newtailroom,
                       gfp_t priority);
```

Figure 22– skb_copy_expand() function

To use netfilter, we must compile the linux kernel. That is out of scope this document. So we do not mention it.

5.3 HOST SIDE

To deliver the data packet between the host and AR, we implement the host protocol stack that is shown below.

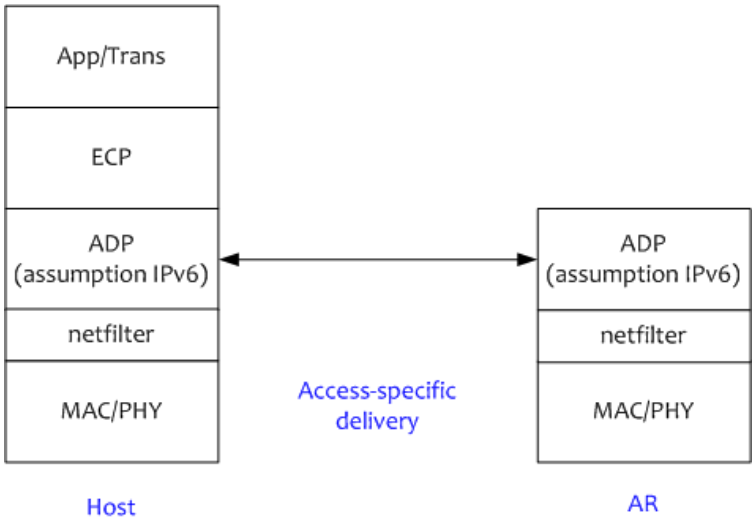

Figure 20 – Data Delivery Protocol Stack in Host

In the host, there are several protocol stack which are MAC/PHY, netfilter, ADP, HID, App/Trans. To modify the protocol stack of host, we use the netfilter and add the two protocol stacks which are ADP layer and HID layer. To achieve this, we modify the network layer in the existed protocol stack. And we have two assumptions easy to implement. First, the HID is globally unique IPv6 address, because we reuse the IPv6 application and consider the backward compatibility. Second, the LID is IPv6 link local address.

5.3.1 DATA HEADER

To implement the MOFI MCP protocol, we define the data (HID) header in the system that is shown below.

```
#include <linux/types.h>
#include <linux/in6.h>
#include <asm/byteorder.h>

struct hcphdr {
    __u8 nexthdr;
    __u8 reserved1;
    __u16 payload_len;
    __u32 reserved2;

    struct in6_addr src_hid;
    struct in6_addr dst_hid;
};
```

Figure 21– Data Header Structure

We refer the IPv6 header file to define the HCP header file, because there are so many similar architecture to reuse of existed Socket API, for example, we suppose the source and destination HID is IPv6 address.

5.3.2 SENDER

The sender uses NF_IP_LOCAL_OUT of netfilter hook to attach the data header between Transport layer header and IP header (ADP header). So, we check whether the Transport layer header and IP header exist or not. At the result, both the IP header and the Transport layer header are existed in the packet from the upper layer that is a transport layer. And then, the packet is separated IP header and Transport layer header by socket buffer functions such as skb_xxx(). We allocate the memory to define the data header. After then, we add the data header between the IP header and Transport layer header.

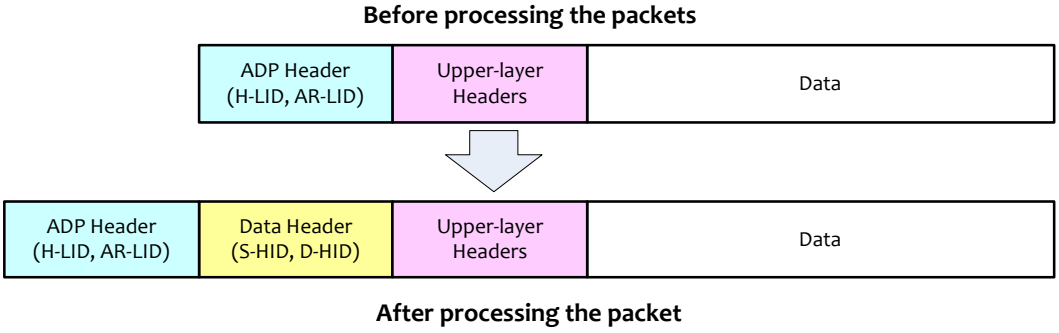

Figure 22 – Process packet in send host

5.3.3 RECEIVER

The receiver uses NF_IP_LOCAL_IN of netfilter hook to remove the HCP header between TCP header and IP header. So, we check whether the Transport layer header and IP header exist or not. At the result, both the IP header and the Transport layer header are existed in the packet from the network layer. And then, the packet is separated IP header and Transport layer header

by socket buffer functions such as skb_xxx(). We remove the data header. After then, we remove the space of the data header as the pointer of IP header moves.

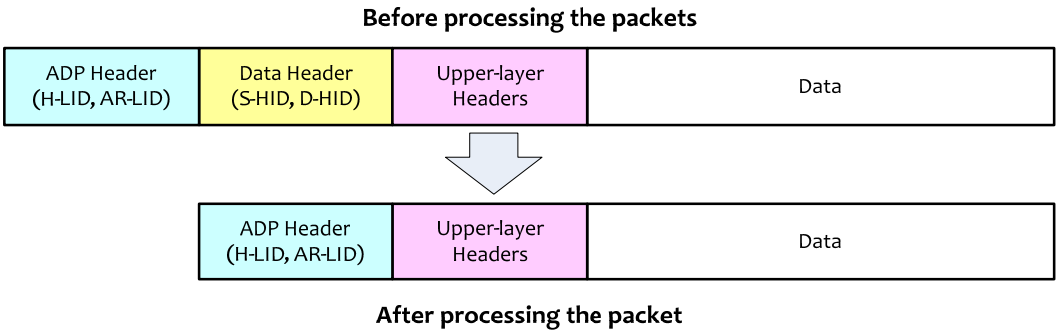

Figure 23 – Process packet in receive host

5.4 Access Router Side

To deliver the data packet between the ARs, we implement the AR protocol stack that is shown below.

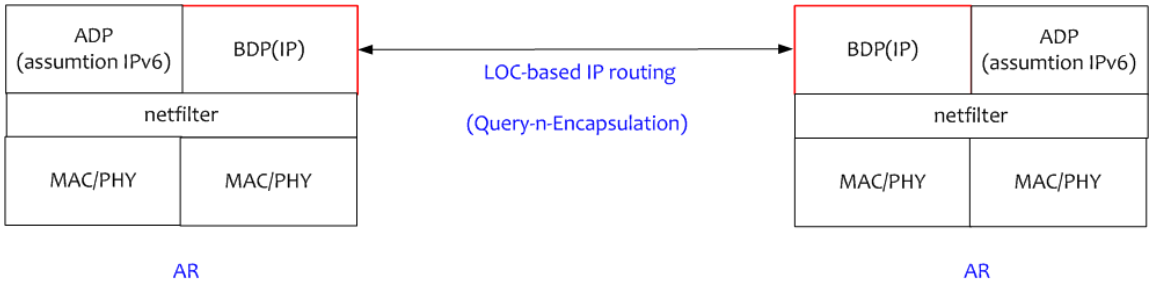

Figure 24 – Data Delivery Protocol Stack in AR

In the figure, the left side is a sender, on the other hand, the right side is a receiver. In the AR, there are several protocol stacks which are MAC/PHY, netfilter, ADP, BDP. To modify the protocol stack of AR, we use the netfilter and add the two protocol stacks which are ADP layer and BDP layer. To achieve this, we modify the network layer in the existed protocol stack such as packet translator which has a dual stack. And we have an assumption that the LOC is used over IPv4 network, because the backbone network is classified access network to use the IPv6 network.

5.4.1 Sender

The sender uses NF_IP_POST_ROUTING of netfilter hook to translate from the IPv6 header (ADP header) to IPv4 header (BDP header). So, we check whether the Transport layer header, HCP header and IP header exist or not. At the result, all the headers are existed in the packet from the host. And then, the packet is translated from ADP header to BDP header. To achieve this, we use the socket buffer functions such as skb_xxx(), And then, we allocate memory to use the BDP header and attach it to the packet and remove the remainder space in the packet.

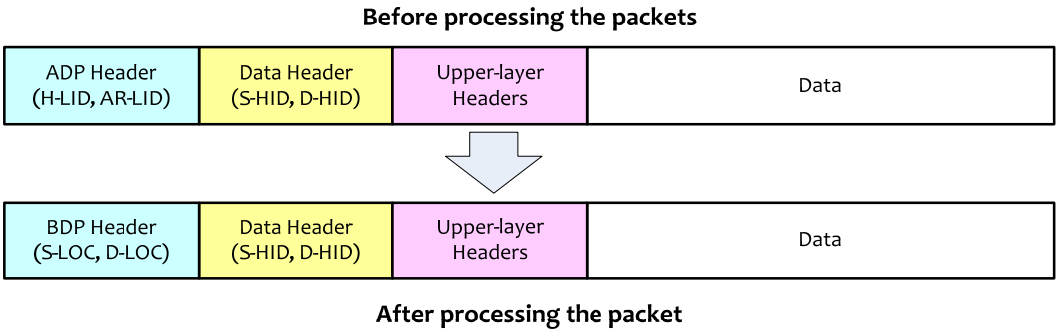

Figure 25 – Process packet in send AR

5.4.2 RECEIVER

The receiver uses NF_IP_PRE_ROUTING of netfilter hook to translate from the IPv4 header (BDP header) to IPv6 header (ADP header). So, we check whether the Transport layer header, HCP header and IP header exist or not. At the result, all the headers are existed in the packet from the host. And then, the packet is translated from BDP header to ADP header. To achieve this, we use the socket buffer functions such as skb_xxx(), And then, we allocate memory to use the ADP header and extend the packet to attach it.

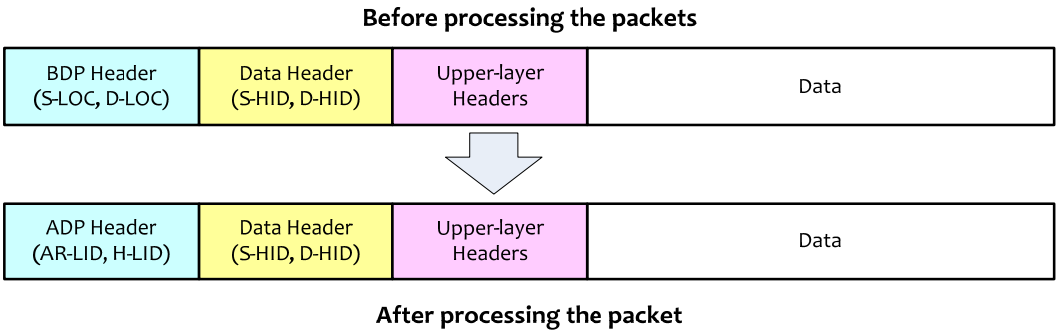

Figure 26 – Process packet in receive AR

REFERENCES

[1] Mobile Oriented Future Internet (MOFI), http://www.mofi.re.kr/
[2] IETF Host Identity Protocol (HIP) WG, http://www.ietf.org/html.charters/hip-charter.html
[3] IETF RFC 4843, An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID), April 2007
[4] IETF RFC 3775, Mobility Support in IPv6, June 2004
[5] IETF RFC 5213, Proxy Mobile IPv6, August 2008
[6] Linux Netfilter Hacking HOWTO, July 2002

ABBREVIATIONS

ACA	Access Control Agent
ADP	Access Delivery Protocol
AR	Access Router
BDP	Backbone Delivery Protocol
DB	Database
FI	Future Internet
HB	HID Binding
HC	HID Cache
HID	Host Identifier
IP	Internet Protocol
LID	Link ID
LB	LOC Binding
LBS	LOC Binding System
LC	LOC Cache
LOC	Locator
LQ	LOC Query
MCP	Mobility Control Protocol
MIP	Mobile IP
MOFI	Mobile-Oriented Future Internet
PoA	Point of Attachment

End of Document