

# Treck Inc.

## Treck Mobile IPv6



**Complete Internet Solutions**

**Technical Support:**

5041 Lamart Drive #240 Riverside, California 92507  
Phone: (909) 787-7056 Fax: (909) 787-8803

**Corporate Headquarters:**

Treck, Inc. 431 Ohio Pike Suite 210 North Cincinnati, Ohio 45255  
Phone: (513) 528-5732 Fax: (513) 528-5740



# Contents

<b>Treck Mobile IPv6 .....</b>	<b>1</b>
<b>1 Introduction .....</b>	<b>5</b>
<b>2 Description .....</b>	<b>7</b>
2.1 Modes of Operation .....	7
2.1.1 CN without Route Optimization .....	7
2.1.2 CN with Route Optimization .....	7
2.1.3 MN without Route Optimization .....	7
2.1.4 MN with Route optimization .....	7
2.1.5 Combined MN/CN with Route Optimization .....	7
2.2 Mobile IPv6 MN .....	8
2.2.1 Move Detection and Handovers (Handoffs) .....	10
2.2.1.1 <i>Mobile-initiated Handover</i> .....	10
2.2.1.2 <i>Network-initiated Handover</i> .....	10
2.2.1.3 <i>Undirected Handover</i> .....	10
2.2.2 L2 Triggers .....	11
2.2.3 MN Use Cases .....	12
2.2.3.1 <i>Access Internal System behind Corporate Firewall</i> .....	12
2.2.3.2 <i>Secure Access to Mail Server over MN-to-HA Tunnel</i> .....	12
2.2.3.3 <i>MN Performs L2 Handover w/out Change of CoA</i> .....	12
2.2.3.4 <i>MN Performs L3 Handover with Change of CoA</i> .....	12
2.2.3.5 <i>MN Moves from Home Network</i> .....	13
2.2.3.6 <i>MN Returns Home</i> .....	13
2.2.3.7 <i>MN Initializes (i.e. Cold Boot) on Home Network</i> .....	13
2.2.3.8 <i>MN Initializes (i.e. Cold Boot) on Foreign Network</i> .....	13
2.3 Mobile IPv6 CN .....	15
2.4 Integration with IPsec .....	15
2.5 PPPv6 and Mobile IPv6 Interaction .....	16
2.6 New Public APIs .....	17
2.7 Code Example .....	18
<b>3 Limitations .....</b>	<b>34</b>
<b>4 Function Reference .....</b>	<b>35</b>
4.1 Changes to Existing Public APIs .....	35
4.1.1 <i>tfInitTreckOptions</i> .....	35
4.1.2 <i>tfSetTreckOptions</i> .....	35
4.1.3 <i>getsockopt</i> .....	36
4.1.4 <i>setsockopt</i> .....	36
4.2 New Public APIs .....	37
4.2.1 <i>tf6MnGetHomeAgentAddress</i> .....	37
4.2.2 <i>tf6MnGetStatus</i> .....	39
4.2.3 <i>tf6MnMoveNotify</i> .....	40
4.2.4 <i>tf6MnRegisterBinding</i> .....	41
4.2.5 <i>tf6MnStartEnhancedIke</i> .....	44
4.2.6 <i>tf6MnStartMobileIp</i> .....	45
4.2.7 <i>tf6MnStopMobileIp</i> .....	49
<b>5 Macros .....</b>	<b>50</b>
<b>6 Acronyms .....</b>	<b>54</b>



# 1 Introduction

Mobile IPv6 allows a *Mobile Node* (MN) to send and receive packets with its global IPv6 home address, regardless of the IP address of its current point of attachment to the Internet. This unchanging global IPv6 address becomes a global network ID for that user that enables “always on” Internet connections as well as push technologies where the network initiates the connection to the user.

For example, Voice over IP (VoIP) – the global IPv6 home address could be used as a phone number, in which case Mobile IPv6 implements the Internet equivalent of *Local Number Portability* (LNP), allowing someone to use the same Internet “phone number” regardless of where they travel with their Mobile IPv6-enabled phone. The capability of making an IPv6 address “virtual” by providing an additional level of indirection via the Care-of Address (CoA) may have other useful applications as well.

---

**NOTE: Mobile IPv6 MN functionality requires IPsec to secure signaling between the MN and HA. IPsec adds approximately 100 Kbytes of ROM usage and 15 Kbytes of RAM usage.**

---

This section gives a brief overview of the features Treck, Inc. has implemented for phase 2 IPv6, which includes:

## 1. Mobile IPv6 Mobile Node:

Mobile IPv6 MN functionality. The MN sends *Binding Update* messages to both the *Home Agent* (HA) and the *Correspondent Node* (CN). IPsec is used to protect the Binding Update packets (BUs) that the MN sends to the HA.

The *Return Routability Procedure* is used to protect the BUs that the MN sends to CNs. The MN must detect when it has moved to a new foreign link (which mainly consists of detecting that its default router has become unreachable and discovering a new default router), and when it has returned to its home link.

When the MN moves to a new foreign link, it forms a new *Care-of Address* (CoA) for use on that foreign link. It then sends a BU to the HA and to each CN that it is communicating with to notify them of the binding of the home address to the new CoA. The MN must support subnet renumbering of its home addresses even when it is no longer attached to its home link. This is accomplished using home prefix discovery. When operating on a foreign link, the MN attempts to use route optimization to send packets directly to a CN by using its CoA as the source address of the packet and specifying its home address in a new Home Address destination option extension header. If the CN does not support route optimization, then all communication between the MN and the CN is tunneled through the HA.

## 2. Mobile IPv6 Correspondent Node:

Mobile IPv6 Correspondent Node (CN) functionality. The CN creates a Binding Cache Entry in response to a received Binding Update message from a MN, which maps the MN's home address to its CoA, and then uses the *Binding Cache Entry* when sending packets to the MN to perform *Route Optimization*, which consists of sending packets directly to the MN's CoA using a Type 2 Routing header containing the MN's home address. The CN uses the *Return Routability* procedure to authenticate the BUs that it receives from the MN.

Mobile IPv6 enables a MN to keep the same global IPv6 home address for use at the application layer even when it changes the CoA IPv6 address that it uses at the network layer. A MN must change its CoA IPv6 address at the network layer whenever it moves to a different link (possible exceptions to this rule are non-on-link IPv6 addresses or movement between different wired and wireless links which share the same subnet).

IPv6 supports dynamic subnet renumbering that enables administrators to maintain the hierarchical IPv6 address space by changing the global scope IPv6 address prefixes assigned to specific network links. This is accomplished through configuring routers to send *Router Advertisements* with the new prefix information, which causes nodes attached to the link to auto-configure new addresses having the new prefix and also to auto-unconfigure old addresses having the deprecated/invalidated prefix. This process of IPv6 address auto-configuration/unconfiguration based on prefix information contained in Router Advertisements is referred to as *Stateless Address Auto-Configuration*. This same mechanism is used by MNs to obtain a Care-of-Address (CoA) on a foreign subnetwork.

When a MN is not directly attached to its home network it won't receive the *Router Advertisements* that would normally be invalidating or refreshing the prefixes it had previously auto-configured on its home network. In order for subnet renumbering (i.e. prefix deprecation, prefix invalidation, and auto-configuration of a new home address based on a new home prefix) when the MN is attached to a foreign network, there must be some way for the HA to send prefix information to the MN to use for *Stateless Address Auto-Configuration* purposes. In addition, the MN may not remember its home address after it is powered off. When it initializes on a foreign network, it must auto-configure a global IPv6 address on its home network (which it could easily do if it received prefix information from the HA). *Home Prefix Discovery* is provided for these purposes.

Security of BUs is a very important area of Mobile IPv6 research, so we will track the draft RFCs in this area very closely.

We will **not** use IPsec to provide security for Mobile IPv6 BUs exchanged between the MN and CN (i.e. route optimization), instead, we use the *Return Routability* procedure, which is the “infrastructureless” method.

*“It is expected that Mobile IPv6 route optimization will be used on a global basis between nodes belonging to different administrative domains. It would be a very demanding task to build an authentication infrastructure on this scale. Furthermore, a traditional authentication infrastructure cannot be easily used to authenticate IP addresses, because these change often. It is not sufficient to just authenticate the MNs. Authorization to claim the right to use an address is needed as well. Thus, an “infrastructureless” approach is necessary.”*

## 2 Description

Mobile IPv6 consists of Mobile Node (MN), Correspondent Node (CN) and Home Agent (HA) functionality. Mobile IPv6 does not use a Foreign Agent (FA).

### 2.1 Modes of Operation

The Treck Mobile IPv6 product has 4 main modes of operation, selectable via feature macros:

#### 2.1.1 CN without Route Optimization

This is the simplest subset of Mobile IPv6 functionality. This is a host which supports indirect communication with a MN via the HA. No tunneling support is required on the CN. This is the same as the current functionality of our IPv6 stack. Because we do not support the Mobile Header protocol (IPPROTO\_MH) in this mode, we will send an ICMPv6 Parameter Problem: Unrecognized Next Header (Code 1) error message in response to a Return Routability procedure or in response to receiving a BU.

#### 2.1.2 CN with Route Optimization

```
#define TM_6_USE_MIP_CN
```

```
#define TM_6_USE_MIP_RO
```

To support Mobile IPv6 route optimization, the CN must also support the return routability procedure for authenticating BUs sent by the MN.

In this mode, the CN can directly send packets to the MN when the MN is operating on a foreign network by including a Type 2 Routing Header in the packet, with the destination address of the packet being the MN's care-of address and the first hop in the routing header being the MN's home address. However, the MN must have first created a mobility binding in the CN associating the home address with the care-of address.

#### 2.1.3 MN without Route Optimization

```
#define TM_6_USE_MIP_MN
```

In this mode, when communicating with a CN while on a foreign network, the MN must tunnel/detunnel everything through the HA, i.e. it cannot send packets directly to the CN and the CN cannot send packets directly to it.

#### 2.1.4 MN with Route optimization

```
#define TM_6_USE_MIP_MN
```

```
#define TM_6_USE_MIP_RO
```

To support Mobile IPv6 route optimization, the MN must also support the return routability procedure. When communicating with a CN using route optimization while on a foreign network, the MN sends packets directly to the CN, and these packets have the source IPv6 address set to the MN's CoA and include the Home Address destination option set to the MN's home address. The Home Address destination option is needed to pass the MN's home address to ULPs on the CN, so that Mobile IPv6 can be transparent to the ULPs.

Route optimization is generally considered more optimal than tunneling traffic through the HA. However, in some cases it may be preferable to tunnel specific types of traffic through the HA. Therefore, we provide the user with a new socket option `TM_6_IPO_MN_USE_HA_TUNNEL` (*protocolLevel* IPPROTO\_IPV6) that they can set to disable the use of route optimization on specific sockets and instead tunnel all packets originating on those sockets through the HA. By default, all socket sends attempt to use RO when the MN is not at home, if the source address of the packet is a home address configured on the virtual home interface and the destination address is global scope.

#### 2.1.5 Combined MN/CN with Route Optimization

```
#define TM_6_USE_MIP_MN
```

```
#define TM_6_USE_MIP_CN
```

```
#define TM_6_USE_MIP_RO
```

MN functionality is separate from CN functionality. Normally, a MN is expected to also implement CN functionality, however that is entirely up to the user, since they may want to save codespace by not including the CN code (i.e. not `#define`'ing `TM_6_USE_MIP_CN`).

## 2.2 Mobile IPv6 MN

There are certain aspects of Mobile IPv6 MN (Mobile Node) functionality that are our responsibility, and the remainder we expect the user application/device driver to perform. Specifically, the user application/device driver is responsible for the following:

### 1. Specify HA address

The user application must tell us the IPv6 address of the HA (Home Agent) when they call **tf6MnStartMobileIp** to start Mobile IPv6 MN (Mobile Node) functionality. This HA address must be site-local scope or global scope, preferably global scope.

### 2. Specify MN home addresses, select primary care-of address, create mobility bindings

When the MN is operating on a foreign network, the user application picks a new care-of address on the foreign subnet (usually, this is done when they are notified of a `TM_6_DEV_ADDR_CONFIG_COMPLETE` address configuration event on the currently active mobile interface, refer to the `dev6AddrNotifyFuncPtr` parameter of **tfNgOpenInterface/tfNgConfigInterface**), and then calls **tf6MnRegisterBinding** for each MN home address to associate it with this new care-of address. The mobile node will automatically attempt to deregister active mobility bindings when it returns home. However, the user can also deregister active mobility bindings by calling **tf6MnRegisterBinding** specifying a binding lifetime of 0, or by calling **tf6MnStopMobileIp** to disable Mobile IPv6 mobile node functionality.

### 3. Decide HA is permanently unreachable, select new HA

The user application is responsible for deciding that the HA is permanently unreachable (i.e. due to subnet renumbering of the home network, or due to that specific HA being shutdown), and when this is the case, the user application is responsible for recovery. Recovery consists of shutting down the MN by calling **tf6MnStopMobileIp**, discovering a new HA address (we provide the API **tf6MnGetHomeAgentAddress** that they can call to perform DHAAD), and then restarting the MN by calling **tf6MnStartMobileIp** with the new HA address. This handles the case where the home network has been renumbered, because **tf6MnStopMobileIp** completely resets the state of all mobility bindings (they will all be deregistered) and of all addresses previously configured on the virtual home interface, i.e. it calls **tfDeviceClose** on the virtual home interface.

### 4. Recover from DAD failure on foreign link

If the mobile node fails to auto-configure a link-local scope address on a foreign link, it won't be able to auto-configure a new primary CoA to register with the HA. The mobile node should set a random interface ID on the mobile interface when this occurs, to recover from this DAD failure. We provide the IPv6 device flag `TM_6_USE_AUTO_IID` for this purpose.

If the user has `#define'd` `TM_6_USE_MIP_MN` in their `trsystem.h`, then we can start functioning as a Mobile IPv6 MN as soon as they call **tf6MnStartMobileIp**. After **tf6MnStartMobileIp** is called, a MN maintains two interfaces:

1. a *virtual home interface* returned by **tf6MnStartMobileIp**. When Mobile IPv6 is active (i.e. after **tf6MnStartMobileIp** is called but before **tf6MnStopMobileIp** is called) and the mobile node is operating away from home, we maintain all of the home address information on the virtual home interface. The user can manually configure home addresses on this virtual home interface when the mobile node is operating away from home. Also, if `TM_6_USE_PREFIX_DISCOVERY` is `#define'd` and an interface ID is set by the user on the virtual home interface, Mobile IPv6 Home Prefix Discovery will auto-configure home addresses on this virtual home interface when the mobile node is operating away from home. The virtual home interface cannot be opened or closed by the user, and is automatically opened by the stack when the mobile node detects that it has moved from home, then later automatically closed by the stack when the mobile node detects that it has returned to home or when the user calls **tf6MnStopMobileIp** to disable Mobile IPv6 mobile node functionality. When `TM_6_MN_DISABLE_HOME_DETECT` is `#define'd`, the mobile node is assumed to never be operating at home, therefore the virtual home interface is opened immediately by **tf6MnStartMobileIp** in that case. When the virtual home interface is closed and there are active mobility bindings, these bindings will be automatically deregistered.
2. the *mobile interface* associated with the real physical device driver, which the user added by calling **tfAddInterface** and which is physically attached to the home or foreign network. Care-of addresses will be auto-configured on this interface when the mobile node is visiting a foreign network. Note that there can be multiple mobile interfaces (for example, a Bluetooth interface, and a 802.11b interface), but we only track the one that the user most recently notified us of via the APIs **tf6MnStartMobileIp** and **tf6MnMoveNotify**. When Mobile IPv6 is active, we use the mobile interface to filter Router Discovery, and then indirectly also to filter Prefix Discovery and Parameter Discovery (since those are filtered based on the current IPv6 default router, which is discovered on the currently active mobile interface).

The user can be notified of address configuration events on both the virtual home interface as well as on the mobile interface. We recommend that they use this notification on the mobile interface to detect when a new global scope care-of address is available to use to create mobility bindings. The user calls **tf6MnRegisterBinding** to create a new mobility binding associating a home address with a new care-of address. At any given time, there will be only one care-of address associated with a given home address per the user-specified mobility binding, and this is referred to as the *primary care-of address*.

When the user calls **tf6MnStartMobileIp**, we may be operating on the home network, or we may be operating on a foreign network (i.e. away from home). When they call **tf6MnStartMobileIp**, they must provide the stack with the address of a Mobile IPv6 HA (Home Agent) on their home network, as well as a home prefix that we can use to determine if we are operating at home. This home prefix could be the same as the mobile node's global scope home address (or possibly even a global scope home agent address), however we leave this up to the user to specify, because it is very important that it is correct since it is the only means we have for accurately determining that are operating on the home network.

After **tf6MnStartMobileIp** has been called, when we first detect that the IPv6 default router on our home network has become unreachable (or when the mobile interface that that default router was discovered on is closed), and we subsequently discover a different default router on a foreign network, we then invalidate all of the home address information (and associated prefix entries in the routing tree and non-on-link prefix list) that was discovered using the old default router. We also purge all old entries from the default router list, and purge any Neighbor Cache and other host routing entries associated with the home network. At this point, if `TM_6_USE_PREFIX_DISCOVERY` is `#define'd` and the user set an interface ID on the virtual home interface, we start doing Mobile IPv6 Home Prefix Discovery to auto-configure home addresses on the virtual home interface.

When operating on a foreign link, the MN normally attempts to use Route Optimization to send packets directly to a CN by using its CoA as the source address of the packet and specifying its home address in a new Home Address destination option extension header. This Route Optimization can only be performed if a mobility binding is first created in the HA by the user calling **tf6MnRegisterBinding**. If the CN or the MN does not support Route Optimization, then instead all communication between the MN and the CN is tunneled through the HA. Route Optimization can be disabled a few different ways: the user can disable it at compile-time by commenting out the macro `TM_6_USE_MIP_RO`, or they can disable it at run-time by setting the Treck option `TM_6_OPTION_MIP_RO_ENABLE` to 0, or the MN can disable it per application socket by setting the socket option `TM_6_IPO_MN_USE_HA_TUNNEL` to 1.

When operating on a foreign link, any packets that the MN sends to nodes on its home network are tunneled through the HA, so we need to redirect those packets out the virtual home interface so that this tunneling can be accomplished. This redirection is done by a modification to **tf6IpDestToPacket** to have it check the source address of the packet (if one is specified) to determine if it is configured on the virtual home interface, in which case when we are not at home the packet is then redirected out the virtual home interface. The link-layer send function for the virtual home interface will then check the destination address to see if it is site-local scope, in which case this packet must be tunneled through the HA. If the destination address is global scope unicast, the BUL is consulted to see if we have an active mobility binding for the destination address. If we find a matching BUL entry, if it is a home registration (i.e. destination is the HA) then we must use the HA tunnel to send this packet, otherwise we normally use RO to send it directly to a CN (which results in us swapping the home address with the care-of address as the source address of the packet in the IPv6 packet header, so that the next time we call **tf6IpDestToPacket** we will instead lookup the outgoing interface using the destination address).

The MN must detect when it has moved to a new foreign link (which mainly consists of detecting that its IPv6 default router has become unreachable, discovering a new IPv6 default router, and when it has returned to its home link. We provide L2 triggers via **tf6MnMoveNotify** that enable the user to do this move detection for us. Otherwise, the default Mobile IPv6 move detection can be very slow, since it is based on using Neighbor Unreachability Detection to detect when the current IPv6 default router has become unreachable. Additionally, we support Eager Cell Switching which is one of the fastest methods of movement detection (refer to Treck option `TM_6_OPTION_MN_EAGER_CELL_SWITCH`, disabled by default), however this is only appropriate for some Mobile IPv6 deployment scenarios, specifically when there is only a single default router per link. We also support the user disabling NUD, implementing their own move detection and notifying the stack of subnet movement by calling **tf6MnMoveNotify** specifying the move event `TM_6_MNME_L3_NEW_SUBNET`.

When a mobility binding is created by the user, the MN sends Binding Update messages to the Home Agent and to all CNs that have an entry in the MN's Binding Update List matching the home address of the mobility binding. IPsec can (the RFC says MUST) be used to protect the BUs that the MN sends to the HA, subject to the security policies that the user has created in the IPsec Security Policy Database. The Return Routability procedure is used to protect the BUs that the MN sends to CNs.

The MN must support subnet renumbering of its home addresses, even when it is no longer attached to its home link. This is accomplished using Mobile IPv6 Home Prefix Discovery.

## 2.2.1 Move Detection and Handovers (Handoffs)

A MN always needs to know when it is operating on its home network versus when it is operating on a foreign network. To be able to detect when we are at home, we first need to know an on-link prefix on the home network (preferably global scope, but site-local scope might work). We require the user to provide us with this on-link home prefix when they call `tf6MnStartMobileIp`. The recommended method of determining if we are home is by soliciting all of the routers on the attached link (i.e. multicast a RS), and then if our home prefix is one of the on-link prefixes advertised in the solicited RAs, we know that we are home. In some network implementations, the MN will never be physically attached to its home network – for that case, we provide the feature macro `TM_6_MN_DISABLE_HOME_DETECT` which can be `#define'd` to reduce Mobile IPv6 ROM code size by disabling home detection and return to home functionality.

Handover is the process of a MN moving from an old link to a new link. In some wireless network implementations, a change of physical link may not always require a change of CoA (i.e. both the old link and the new link may be on the same subnet). This change of physical link is referred to as a L2 handover. If the L2 handover results in a change of subnet and therefore a change of CoA, then it is referred to as a L3 handover. The primary goal of move detection is to detect L3 handovers.

Once the MN has detected that it has moved to a new link, it first needs to determine if there has been a change of subnet (i.e. L3 handover), in which case it must form a new primary CoA to use when operating on that new link. L2 may have information that could be used to make that determination. If there is no L2 trigger to notify the MN that there has been a change in subnet, then the MN uses other means such as checking the AR on the old link to see if it is still reachable. In general, if the old AR is not reachable, that does not necessarily mean that the MN must form a new primary CoA, because the same subnet could be advertised by multiple ARs.

When the MN determines that it must form a new primary CoA, it multicasts a Router Solicitation to discover the prefixes that are used for Stateless Address Auto-Configuration on the new link. After forming a new primary CoA, the MN must notify its HA as soon as possible of the new CoA so that the HA can continue to route/tunnel packets to it at the new CoA. It can also notify CNs that it is communicating with using route optimization of the change in its CoA. These notifications are accomplished by the MN sending BUs to the HA and to the CNs.

### 2.2.1.1 Mobile-initiated Handover

In some wireless network implementations, the MN can discover a list of RF channels that are in the same or adjacent cells, and can monitor these periodically to see if they offer a higher RSSI (i.e. stronger signal) than the link to which it is currently attached. It can then use this information to move to a new link when it determines that signal quality on the current link has degraded past some threshold value (NOTE: some degree of hysteresis is needed when making this signal quality determination, to avoid moving too frequently). L2 can use a combination of RSSI, transmit BER and receive BER to determine signal quality, however this is implementation-dependent and is not specified.

### 2.2.1.2 Network-initiated Handover

In some wireless network implementations, the network (i.e. wireless AR or AP) can serve as the master in directing the MN to perform a handover to a new link. This is implementation-dependent and is not specified.

We support this type of handover via a movement detection algorithm referred to as Eager Cell Switching, however this algorithm is disabled by default because it does not work well in specific deployment scenarios where the MN can receive Router Advertisement messages from multiple routers without experiencing link movement (i.e. multiple routers on a single link, or overlapping coverage areas). Refer to the Treck option `TM_6_OPTION_MN_EAGER_CELL_SWITCH`.

### 2.2.1.3 Undirected Handover

In this case, L2 (either the MN or the network) has not made any decision in advance to move to a new foreign link, however the link that the MN was attached to is no longer usable so the MN must change its point of attachment. This is typically caused by a RF null. The MN may or may not implement L2 triggers to notify MIPv6 of this situation.

It is possible for the MN to detect a change of link at L3 by monitoring received Router Advertisement messages. When the Router Advertisement contains the Advertisement Interval option, the MN assumes that a L3 handover has occurred when it fails to receive a specified number of Router Advertisement beacons in sequence from the current default router (refer to the Treck option `TM_6_OPTION_MN_BEACON_THRESHOLD`). The MN detects that a Router Advertisement was sent by the current default router by comparing the source IPv6 address of the advertisement against the address of the current default router, however this is a link-local scope IPv6 address, so it is not guaranteed to be unique across links. If the Router Advertisement contains a Router Address (R) bit Prefix Information Option, when the feature macro `TM_6_USE_MIP_RA_RTR_ADDR` is enabled then we use this option to more reliably determine if the Router Advertisement is from the current default router.

## 2.2.2 L2 Triggers

Generic Mobile IPv6 move detection consists of using Neighbor Unreachability Detection to detect when the current default router becomes unreachable. This method is slow and inefficient, and is prone to error since subnet movement may not have occurred; for example, the MN may be experiencing temporary RF interference or the router may be temporarily down. L2 may have better information about link and subnet movement. For example, CDPD uses the area color code at the MAC layer to detect subnet movement. Per [MIPV6\_18++].R11.5.1:10, the MN should use this L2 information whenever it is available to optimize move detection and/or minimize handover latency. This L2 information is communicated to the MN via L2 triggers.

From [L2\_TRIG\_REQ]:

*“Handover occurs when a Mobile Node moves from one radio Access Point to another. If the new radio Access Point is associated with a new subnet, a change in routing reachability may occur and require L3 protocol action on the part of the Mobile Node or Access Routers. If no change in subnet occurs, the Access Point may still need to take some action to inform the Access Router about a change in on-link reachability. In either case, prompt and timely information from L2 to the parties involved about the sequencing of handover can help optimize handover at the IP level.”*

L2 triggers can provide the MN with some of the L2 information it needs to optimize move detection. As described by [L2\_TRIG\_REQ], notification of “link up”, “link down” events and “setup for handover” events (i.e. preparing the MN for a mobile-initiated or network-initiated handover) can enable the MN to optimize move detection and/or minimize handover latency. However, L2 triggers can also complicate MN design, and can increase MN power consumption due to additional protocol overhead if they aren't designed appropriately to fit MN move detection requirements. For example, the “link up” and “link down” events in themselves do not indicate any actual subnet movement (in fact, they may not even indicate movement to a new link); they only tell the MN that there was a period of lost link connectivity, which may or may not be due to a link quality issue such as RF interference.

Advanced Mobile IPv6 move detection algorithms such as [FASTHO\_04] use the “setup for handover” events to control/direct the process of handover from the old AR to the new AR. These events provide better indications of subnet movement. Since there is still a lot of work going on in the area of fast handovers, we do not currently plan to implement any of this advanced functionality.

In general, we assume that L2 (which includes the user's device driver, and the functionality below that) has all of the necessary information to directly control IPv6 mobility, and we provide the low-level API **tf6MnMoveNotify** that the user application and/or device driver can call to exercise this direct control. For example, **tf6MnMoveNotify** can be called to notify the MN of L2 and L3 handovers. Along similar lines, with regards to move detection, since the generic Mobile IPv6 move detection algorithm is sub-optimal, we will generally minimize the high-level decision making that the MN does and instead make that the responsibility of the user's application. For example, the stack is not responsible for picking a new CoA for the MN, instead the user's application calls **tf6MnRegisterBinding** to notify the MN of a change of CoA. To simplify this task, we already provide a means via the parameter *dev6AddrNotifyFuncPtr* of the API **tfNgConfigInterface** for the application to be notified of when an IPv6 address has completed configuration on the interface – all the user needs to do is register their address configuration callback function on the mobile interface when they open/configure the interface, and then in their callback function, when they are notified that a new global scope IPv6 address has completed configuration on the mobile interface they pass that address as the new CoA to **tf6MnRegisterBinding**.

When we get a movement notification (either L2 or L3 handoff), we will call TCP using a new internal API **tfTcpBackOnLink** so that TCP can attempt a fast recovery from a temporary loss of network connectivity.

## 2.2.3 MN Use Cases

### 2.2.3.1 Access Internal System behind Corporate Firewall

In some cases, the user wants to communicate with an internal system behind the corporate firewall. The internal system in this case has a site-local scope address. The user has configured (either auto- or manually) a site-local scope address with a matching prefix on their virtual home interface. When they send to the site-local address of the internal system, we will find a prefix match on the virtual home interface, and because the address is site-local scope we will always use the MN-to-HA tunnel to send to that destination. Alternatively, if we don't find a prefix match but if they specify a source address which is configured on the virtual home interface, that will cause us to select the virtual home interface as the outgoing interface, and then in the link-layer send function on that interface we will see that the destination address is site-local scope and so we must send it using the HA tunnel in this case.

It is much less likely that the Mobile IPv6 infrastructure will support site-local scope home addresses. Using VPN with IPv6 is an alternate method.

### 2.2.3.2 Secure Access to Mail Server over MN-to-HA Tunnel

Because of various requirements that mandate the use of IPsec between the MN and the HA, the MN should be able to establish a security association with the HA to protect its application data traffic using ESP. Typically, in a corporate environment, the user's mail server will be located in the corporate network, and may or may not be directly accessible through the corporate firewall (this scenario assumes it is directly accessible through the firewall). If the mail server is a CN, by default the MN would use RO to communicate with it. In this case, the user does not want the MN to use RO, because they want to take advantage of a security policy that they've created to encrypt all application data traffic that is tunneled through the HA. However, the user doesn't want to globally disable RO, only just when accessing the mail server. We provide the `TM_6_IPO_MN_USE_HA_TUNNEL` socket option for this purpose.

### 2.2.3.3 MN Performs L2 Handover w/out Change of CoA

For some time after the MN has moved to a new link (even when there was no subnet move), the network may not be aware that it has moved, and in that case the network continues forwarding packets sent to the MN to the old link and these packets never reach the MN on the new link to be received. Per [L2\_TRIG\_REQ]:

*“... current trends in wireless networking suggest that future wireless networks will consist of a variety of heterogeneous wireless APs bridged into the wired network, potentially on the same subnet. A change in wireless AP, either between an AP supporting one wireless link technology and an AP supporting another, or between two APs supporting the same wireless technology, necessarily results in a change in the on-subnet reachability. Packet delivery within the subnet can be optimized if this information can be propagated to the AR, so it can update its on-subnet L2 address to IP address mapping.”*

For this scenario when we are notified of a L2 handover, we could send an unsolicited NA for our primary CoA to notify the AR that we have moved to a new physical link. Otherwise, if we don't do this, the AR may not route packets to the MN on the correct physical link.

### 2.2.3.4 MN Performs L3 Handover with Change of CoA

The MN should still continue to accept packets sent to its old CoA. This is to better support overlapping coverage areas, where the MN may be moving back and forth between 2 different wireless links, and may even not finish forming and registering the new CoA before it moves back to the old link. We will support this by not immediately unconfiguring the old primary CoA after we've moved to a new link, but instead deprecating the associated prefix and setting its valid lifetime to a small value so that it is invalidated soon after the move completes.

Other address prefixes discovered on the old link will be immediately invalidated if they aren't advertised in the solicited Router Advertisement that we receive on the new link. This is done prior to performing Prefix Discovery on the solicited Router Advertisement, because we want to make sure that there are slots available on the mobile interface to auto-configure new care-of addresses.

The user is responsible for selecting a new care-of address. Normally, they will obtain this via a `TM_6_DEV_ADDR_CONFIG_COMPLETE` address configuration event occurring on the currently active mobile interface (i.e. they register a callback function for this event when they open the mobile interface using the `dev6AddrNotifyFuncPtr` parameter of `tfNgOpenInterface/tfNgConfigInterface`).

After the user selects a new care-of address on the new link, they notify the stack of that address by calling `tf6MnRegisterBinding` for each MN home address to associate it with this new care-of address.

When we detect that we have moved to a new link, we need to re-join the non-link-local scope multicast groups that were previously joined by the application.

### 2.2.3.5 MN Moves from Home Network

When the MN moves from its home network to a foreign network, it may have previously established sessions with nodes on its home network using their link-local scope addresses. These sessions cannot be preserved across the move, because the HA does not tunnel any link-local scope traffic to the MN. However, site-local scope and global scope sessions that the MN had before the move can be preserved, provided that the MN registers the home addresses associated with those sessions with the HA to associate those home addresses with the primary CoA that the MN is using on the foreign network.

### 2.2.3.6 MN Returns Home

If the user has not `#define'd TM_6_MN_DISABLE_HOME_DETECT`, then we determine that the MN has returned home when we receive prefix information on the mobile interface for global scope on-link prefixes that are configured on the virtual home interface – it is the responsibility of the user to provide us (via the `homePrefixPtr/homePrefixLen` parameters to **tf6MnStartMobileIp**) with a specific on-link home address prefix that we can use for home detection. When the MN returns to its home network, it must first deregister any active mobility bindings that it has in the HA before it can configure its home addresses. If the MN instead configures its IPv6 home addresses before deregistering them, the HA in this case would defend those addresses and DAD will fail. To recover, the MN must deregister any active mobility bindings that it has in the HA, but this is complicated by the fact that the MN cannot use its home addresses until they are first deregistered and the source address in the deregistration BU must be the home address that is being deregistered. Therefore, when the MN returns home it must do the following:

1. The MN resolves the HA's link-layer address by receiving a Source Link-Layer Address option in a Router Advertisement from the HA, which causes the MN to create a Neighbor Cache entry for the HA in the REACHABLE state. This is done so that the MN can deregister with the HA. If the MN cannot resolve the HA's link-layer address via a Router Advertisement, then it just does regular Neighbor Solicitation to resolve the HA's link-layer address, since we never register a mobility binding specifying L=1 therefore the MN's auto-configured link-local scope address should be available for use on the home network as the source address of the Neighbor Solicitation.
2. When the MN has active mobility bindings in the HA, it must not perform DAD on its home addresses. Otherwise, if it does not have active mobility bindings, it should perform DAD. In some cases (i.e. after the MN has been power-cycled) the MN is not aware that it still has active mobility bindings in the HA, so it seems impossible to implement in all cases. Note that if the MN performs DAD on its own home address, if the HA is defending that address, it will respond, and will include its link-layer address in the Neighbor Advertisement.

To add further complication, the MN must use IPsec authentication when sending the BU to deregister with the HA, so it either must use existing security associations that it established previously with the HA (i.e. before returning home) for this purpose, or if that is not possible (i.e. the MN has rebooted, and has forgotten the security association information), then the MN creates new security associations with the HA, but the difficult part of this is that the MN cannot use its home addresses until it has successfully deregistered. In this case, if dynamic keying (i.e. IKE) is used, the MN may be able to use its auto-configured link-local scope address on its home network to create security associations with the HA as it would when operating on a foreign network. The simpler case is where manual keying is used, because then the security associations can be immediately created without the need for any messages to be exchanged with the HA.

Worst case (i.e. if the MN is not able to successfully deregister), the mobility bindings in the HA will eventually timeout if they are not explicitly deleted by the MN, provided that they do not have an infinite lifetime, but it could still take hours or even a few days for this to happen so we must make this deregistration process work reliably.

---

**NOTE: we do not support the MN creating mobility bindings with an infinite lifetime.**

---

### 2.2.3.7 MN Initializes (i.e. Cold Boot) on Home Network

When the MN initializes on its home network, it is possible that the HA is defending its home addresses from a previous registration, and the MN may not be aware of that fact.

### 2.2.3.8 MN Initializes (i.e. Cold Boot) on Foreign Network

When the MN initializes on a foreign network, it needs to register with the home agent a mobility binding associating its home address with a primary care-of address, however if there is a different home agent maintaining an active mobility binding for that same home address, the MN will not be allowed to register the mobility binding. In this scenario, the MN had previously registered the conflicting mobility binding with the other home agent before it was powered off on the foreign network, and then once powered back on has since forgotten which home agent it was that is still maintaining and defending that active mobility binding.

Because of this scenario it is important that the MN continues to use the same home agent while it is operating on a foreign network.

“Mobile nodes that use dynamic home agent address discovery should be careful with long lifetimes. If the mobile node loses the knowledge of its binding with a specific home agent, registering a new binding with another home agent may be impossible as the previous home agent is still defending the existing binding. Therefore, mobile nodes that use home agent address discovery SHOULD ensure information about their bindings is not lost, de-register before losing this information, or use small lifetimes.”

## 2.3 Mobile IPv6 CN

The CN creates a Binding Cache Entry (i.e. mobility binding) in response to a received Binding Update message from a MN, which maps the MN's home address to its CoA, and then uses the Binding Cache Entry when sending packets to the MN to perform Route Optimization, which consists of sending packets directly to the MN's CoA using a Type 2 Routing header containing the MN's home address. The CN uses the Return Routability procedure to authenticate the BUs that it receives from the MN.

## 2.4 Integration with IPsec

We fully support the (K) Key Management Mobility Capability bit used in the MN-to-HA BU/BA exchange. Our mobile node updates the IKE endpoint to the new primary care-of address whenever it moves, provided that the BA from the HA has the (K) bit set to 1. To use this functionality, the user calls **tf6MnStartEnhancedIke** immediately prior to the first call to **tf6MnRegisterBinding** for a specific home address. Upon receipt of a BA from the HA with the (K) bit set to 0, Treck IKE discards and renegotiates all IKE phase 1 SAs associated with the peer address, which in this case is the home agent address.

When the MN is away from home, the control traffic between the HA and MN must be secured using IPsec. In order to do this, security policies must be created in the Security Policy Database, and then IPsec security associations (SA) can be established either manually (using manual keying) or dynamically (using IKE). Whenever the MN sends a packet (including control traffic), it searches the SPD to find a matching security policy, and if there is a match but no SAs, then it uses IKE to establish the SAs. Whenever the MN receives a packet, again it searches the SPD to find a matching security policy.

A security policy is comprised of a selector and at least one policy content. The sent and received packets are matched against the selector to choose the correct security policy to apply – then, the policy content tells IPsec what to do with the packet (i.e. authenticate, encrypt, bypass IPsec, discard, etc.).

To secure the BU/BA exchange between the MN and HA, the MN creates a normal security policy in the SPD with the selector matching the MN's home address. This is possible because in this case the MN's care-of address is actually not inserted into the IPv6 packet header until after applying IPsec. To protect payload traffic tunneled by the MN through the HA, the MN specifies a new IPsec policy flag `TM_PFLAG_MIPV6_HA_TUNNEL` when creating this security policy in the SPD. Please refer to the code example as well as the IPsec/IKE user documentation for more information.

It seems like it would be possible for Mobile IPv6 to use other authentication and privacy mechanisms between the MN and the HA besides IPsec. For example, in cellular wireless network, the MN is already authenticated by the network using the SIM and a different mechanism besides IPsec. Whether or not to use IPsec is entirely a function of the application configuring the Security Policy Database correctly. We will permit IPsec to be used between the MN and the HA, but we will not require it. It is the responsibility of the user application to setup the IPsec Security Policy Database correctly to make this happen. Typically, there would be two security policies in effect between the MN and the HA: direct communication between the MN and the HA using the Mobility Header (new protocol ID is *TBD*) must be authenticated for protection of BUs from the MN to the HA (this also covers the Prefix Discovery process when the MN discovers its home network prefix information while it is attached to a foreign network). The other security policy is for all other traffic tunneled between the HA and the MN, and in this case ESP with authentication should be used.

Currently, [MIPV6\_18++] seems somewhat broken as far as how it handles subnet renumbering when the MN is away from home. IPsec SAs must be established between the MN and the HA, before the MN can discover its home prefixes. The currently defined procedure for doing this with IKE assume that the home address of the MN does not change, or if it does (and also if the HA address changes), that there are already new digital certificates in place which can be used to establish a security association using the new home address. Refer to section 14.3 of [MIPV6\_18++] which says:

*"While certificate-based automatic keying alleviates this problem to an extent, it is still necessary to ensure that a given mobile node can not send Binding Updates for the address of another mobile node. In general, this leads to the inclusion of home addresses in certificates in the Subject AltName field. This again limits the introduction of new addresses without either manual or automatic procedures to establish new certificates. Therefore, this specification restricts the generation of new home addresses (for any reason) to those situations where there already exists a security association or certificate for the new address."*

For this specific scenario, we can discover our home prefixes using DHAAD even though we cannot first establish IPsec SAs with the HA.

---

## 2.5 PPPv6 and Mobile IPv6 Interaction

Mobile IPv6 default move detection uses NUD to determine when the current IPv6 default router is unreachable. However, when using Mobile IPv6 over PPPv6, in one scenario IPv6 Router Advertisement messages are not sent to the mobile node over that link, and then some other method must be used for Mobile IPv6 move detection. PPPv6 link reestablishment can be used as the indication that movement has occurred. If the user application registers a PPP notification function, PPPv6 gives the user application a `TM_LL_IP6_OPEN_COMPLETE` link notification after a PPPv6 link reestablishment. If the PPP link supports IPv6 Router Discovery, in the context of the `TM_LL_IP6_OPEN_COMPLETE` callback the user can call **`tf6MnMoveNotify`** to notify Mobile IPv6 that a L3 move has occurred. Otherwise, the user should not call **`tf6MnMoveNotify`** but instead should wait until the mobile node has obtained a new care-of address, and then call **`tf6MnRegisterBinding`** to update the mobility bindings it previously created in its home agent and correspondent nodes with the new care-of address.

The local and peer addresses used by PPPv6 are link-local scope addresses, formed from the interface IDs negotiated by PPPv6 during link establishment. When sending a packet, if we fail to find a default gateway entry when we do the routing lookup in **`tf6IpDestToPacket`**, there is special logic that looks up the routing entry for the PPPv6 peer address (added by **`tfDeviceStart`**) and then we use that to send the packet out the PPPv6 interface.

## 2.6 New Public APIs

When the user wants to enable Mobile IPv6 MN functionality, they call **tf6MnStartMobileIp**. When they want to disable Mobile IPv6 MN functionality (which also causes deregistration of existing mobility bindings with the HA), they call **tf6MnStopMobileIp**.

When the user calls **tf6MnStartMobileIp**, they can optionally specify the interface ID to use for stateless address auto-configuration on the virtual home interface. To do this, the user can first open the mobile interface (if the link-layer is Ethernet or PPP, the interface ID is set after the interface is opened), call the API **tf6Eui64GetInterfaceId** to get the interface ID of the mobile interface, and then pass a pointer to that interface ID to **tf6MnStartMobileIp** to set the same interface ID on the virtual home interface. This way, the same home addresses will be auto-configured on the virtual home interface as would normally be auto-configured when the mobile node is operating on its home network (provided it uses the same interface ID when attached to its home network).

There is only a single home agent supported by the MN, and that is specified by the user when **tf6MnStartMobileIp** is called.

The user calls **tf6MnGetHomeAgentAddress** to discover the address of a home agent on their home network.

To enable support for the (K) Key Management Mobility Capability bit used in the MN-to-HA BU/BA exchange, the user calls **tf6MnStartEnhancedIke** immediately prior to the first call to **tf6MnRegisterBinding** for a specific home address. Refer to the code example to see how this is done.

When operating on a foreign network, the user calls **tf6MnRegisterBinding** to create a mobility binding associating a primary CoA with a home address. To deregister an existing MN mobility binding without disabling Mobile IPv6, the user can call **tf6MnRegisterBinding** for a specific mobility binding, specifying the new care-of address to be the same as the home address of the mobility binding or specifying a binding lifetime of 0. Mobility bindings are automatically deregistered by the mobile node when it returns home, when the home address is unconfigured from the virtual home interface (which could happen due to Home Prefix Discovery prefix invalidation), or when the user calls **tf6MnStopMobileIp** to disable Mobile IPv6 mobile node functionality.

The user calls **tf6MnMoveNotify** to notify the stack that the mobile node has moved to a new link and/or possibly a new subnet. The stack will automatically perform movement detection using Neighbor Unreachability Detection if that functionality is enabled. However this method is fairly slow and L2 triggers for move detection are expected to be significantly more responsive. We also support Eager Cell Switching which is one of the fastest methods of movement detection (refer to Treck option `TM_6_OPTION_MN_EAGER_CELL_SWITCH`, disabled by default), however this is only appropriate for some Mobile IPv6 deployment scenarios, specifically when there is only a single default router per link.

The user calls **tf6MnGetStatus** to determine if the MN is operating on the home network or on a foreign network.

The user can call **tf6Eui64GetInterfaceId** to determine what the previous value of the interface ID is on the mobile interface prior to calling **tf6SetRandomInterfaceId**. These APIs make it easier for the mobile node to change the interface ID on the mobile interface to recover from a DAD failure when operating on a foreign link. After the mobile node moves to a different link with a different default router, it should normally restore the previous setting of the interface ID by calling **tf6Eui64SetInterfaceId** (if **tf6SetRandomInterfaceId** was used).

## 2.7 Code Example

```

#define TM_MAX_IPS_PER_IF 4 /* #define this in trsystem.h */
#include <trsocket.h>

/* declare some global variables to use in this example */

#ifdef TM_USE_IKE
/* pre-shared key used by IKE */
char ikePreSharedKey[] = "a shared secret";
char mobileNodeIdentity[] = "river2222.treck.com";
char homeAgentIdentity[] = "river1111.treck.com";
char inKeyMat[] = "mobilenodetunnelkeybetweenMNandHA";
char outKeyMat[] = "AHdnaNMneewtebyeklennutedonelibom";
char homeAddrString[INET6_ADDRSTRLEN];
char homeAgentString[INET6_ADDRSTRLEN];
int startedIkeStatus = 0;
#endif /* TM_USE_IKE */

/* The mobile interface is the Treck stack interface that you create
   by calling tfAddInterface, which is associated with a real
   physical device. When the mobile node is at home, this interface
   is attached to the home link and is used to configure home
   addresses. When the mobile node is operating on a foreign network,
   this interface is attached to the foreign link and is used to
   auto-configure care-of addresses. */
ttUserInterface      mobileInterface;
char                 mobileInterfaceId[8];

/* tf6MnStartMobileIp returns a pointer to a special interface that
   the stack creates for the purpose of maintaining your home
   addresses when the MN is operating on a foreign network. This
   special interface is called the virtual home interface. The
   virtual home interface cannot be opened or closed by the user,
   and is automatically opened by the stack when the mobile node
   detects that it has moved from home, then later automatically
   closed by the stack when the mobile node detects that it has
   returned to home or when the user calls tf6MnStopMobileIp to
   disable Mobile IPv6 mobile node functionality. */
ttUserInterface      virtualHomeInterface;

/* The mobile node's home address, which is global scope. This is
   the local IP address that we will register with the home agent
   when the mobile node is operating on a foreign network. In this
   example, we assume that this home address is stored in NVRAM.
   In general, it is difficult to change the mobile node's home
   address, because of the requirement that an IPsec security
   association must be used to protect the Binding Update message
   that the mobile node sends to its home agent to register a
   mobility binding ([MIPV6_18+].R5.1:10) and the fact that
   configuration of this security association is in many cases
   dependent on the specific home address. For this example,
   we will require that this home address is always manually
   configured by the user on the mobile interface in slot 0
   whenever the mobile node returns home. Note that this example
   does not attempt to handle the scenario where the home address
   is invalidated. */
struct sockaddr_storage mnHomeAddr;
int haveMnHomeAddrStatus = 0;

```

```

/* We attempt to register our home address mobility binding with
the home agent. At some point, this mobility binding we have
created in the home agent will expire, and then our home
address is no longer registered. The mobile node will
automatically re-register to refresh this mobility binding and
keep it from expiring, but it is possible that it is not able
to contact the home agent for an extended period of time, or
the home agent fails our (re)registration attempt in which
case we become unregistered. */
int                mnHomeAddrRegistered = 0;

/* The mobile node's primary care-of address, which is global
scope. This is the address associated with the mobile node's
home address in the mobility binding that we register with the
home agent. In this example, the care-of address is dynamically
obtained by the mobile node when it starts operating on a
foreign network. */
struct sockaddr_storage  mnCareOfAddr;
int                haveMnCareOfAddrStatus = 0;

/* The home agent address. This can be dynamically obtained using
Dynamic Home Agent Address Discovery (DHAAD), however once a
specific home agent is located and successfully registered with
by the mobile node, the mobile node should continue to use that
same home agent as long as there is the possibility that that
home agent could still be maintaining active mobility bindings
for it. So, it is strongly recommended that the mobile node
persist the home agent address in NVRAM. */
struct sockaddr_storage  homeAgentAddr;
int                haveHomeAgentAddrStatus = 0;

/* The user can notify the mobile node of a change in IP subnet,
which is referred to as a L3 handover because it requires the
mobile node to configure a new primary care-of address (if the
move is to a new foreign link) or to deregister its active
mobility bindings (if the mobile node is returning home). This
is done by calling tf6MnMoveNotify, specifying
TM_6_MNME_L3_NEW_SUBNET for moveType. Other types of movement
notifications (that do not necessarily indicate a change of
subnet) are also possible using this API. This use of
tf6MnMoveNotify assumes that the user has access to some
information available via the link-layer and/or MAC layer
that they can use to detect link (L2) and subnet (L3) movement.
The mobile node will also notify the user when it detects L3
movement, specifically when its current default router on the
mobile interface becomes unreachable per Neighbor
Unreachability Detection ([MIPV6_18++].R11.5.1:40). In this
example, we keep track of L3 movement notifications so that
we can manage the change of primary care-of address. */
int                l3MovePendingStatus = 0;

/* it is assumed that you persist certain things such as the home
address and home agent address in NVRAM. This example expects
you to implement the following APIs which do this. These APIs
return 0 for success, and non-0 if an error occurs. */

/* read the mobile node local IP address from NVRAM. */
int myEEReadMnHomeAddr(struct in6_addr * sin6_addr);

```

---

```

/* read the home agent address from NVRAM. */
int myEEReadHomeAgentAddr(struct in6_addr * sin6_addr);

/* write the home agent address to NVRAM. */
int myEEWriteHomeAgentAddr(const struct in6_addr * sin6_addr);

/* myMnNotifyFunc is the callback function for Mobile IPv6 mobile
 * node events (see the API tf6MnStartMobileIp). These events
 * consist of Mobile IPv6 Home Agent reachability and IPv6
 * default router discovery and reachability.
 */
void myMnNotifyFunc(
    ttUserInterface interfaceHandle,
    struct sockaddr_storage * routerAddrPtr,
    tt6MnEvent event)
{
    switch(event)
    {
        case TM_6_MNE_HA_UNREACHABLE:
/* This event more likely indicates that there is a transient loss
of network connectivity. We do not handle it
in this example, mainly because you generally do not want to
change the home agent you have registered with. */

            break;

        case TM_6_MNE_RTR_UNREACHABLE:
/* Our current default router has been determined by the mobile node
to be unreachable, and this means that we have probably moved to
a new subnetwork. */
            l3MovePendingStatus = 1;
            break;

        case TM_6_MNE_MOVE_FROM_HOME:
/* When the mobile node detects that it has moved from the home
network to a foreign network, or when Mobile IPv6 initializes
on a foreign network, we need to configure our home address
on the virtual home interface. */
            errorCode=tfNgConfigInterface(
                virtualHomeInterface,
                &mnHomeAddr, /* home address */
                64, /* prefix length */
                configFlags,
                0, /* IPv6-specific flags */
                scatteredBufferCount,
                TM_6_DEV_ADDR_NOTIFY_FUNC_NULL_PTR,
                0); /* slot 0 */
            break;

#ifdef TM_6_MN_DISABLE_HOME_DETECT
        case TM_6_MNE_RETURN_TO_HOME:
/* Whenever we return to our home network, or when Mobile IPv6
initializes on the home network, we need to configure our
home address on the mobile interface. */
            errorCode=tfNgConfigInterface(
                mobileInterface,
                &mnHomeAddr, /* home address */
                64, /* prefix length */
                configFlags,

```

```

        TM_6_DEV_OPTIMIZE_DAD | TM_6_USE_AUTO_IID, /* IPv6 flags */
        scatteredBufferCount,
        myMobileAddrNotifyFunc,
        0); /* slot 0 */
    break;
#endif /* TM_6_MN_DISABLE_HOME_DETECT */

    default:
        break;
}

return;
}

/* myMnBindingNotifyFunc is the callback function for registration/
 * deregistration status associated with our home address mobility
 * binding in the home agent (see the API tf6MnRegisterBinding).
 */
void myMnBindingNotifyFunc(
    struct sockaddr_storage * homeAddrPtr,
    struct sockaddr_storage * careOfAddrPtr,
    tt6MnBindingEvent event,
    int status )
{
    switch(event)
    {
    case TM_6_MNBE_HA_REG_SUCCESS:
/* We successfully registered our home address mobility binding
with the home agent. */
        mnHomeAddrRegistered = 1;
        break;

    case TM_6_MNBE_HA_DEREG_SUCCESS:
        mnHomeAddrRegistered = 0;
/* If we just successfully deregistered our home address when at
home (i.e. the home agent was defending it), then reconfigure
it on the mobile interface which will recover from a previous
Duplicate Address Detection failure on that address. */
        if (tf6MnGetStatus() == TM_6_MN_HOME_NETWORK)
        {
            errorCode=tfNgConfigInterface(
                mobileInterface,
                &mnHomeAddr, /* home address */
                64, /* prefix length */
                configFlags,
                TM_6_DEV_OPTIMIZE_DAD | TM_6_USE_AUTO_IID, /* IPv6 flags */
                scatteredBufferCount,
                myMobileAddrNotifyFunc,
                0); /* slot 0 */
        }
        break;

    case TM_6_MNBE_HA_REG_FAILED:
    case TM_6_MNBE_HA_REG_TIMEOUT:
    case TM_6_MNBE_HA_DEREG_FAILED:
    case TM_6_MNBE_HA_DEREG_TIMEOUT:
    case TM_6_MNBE_HA_BINDING_TIMEOUT:
        mnHomeAddrRegistered = 0;
        break;

```

```

    default:
        break;
    }

    return;
}

/* myMobileAddrNotifyFunc is the IPv6 address configuration event
 * callback function for the mobile interface (See the APIs
 * tfNgOpenInterface/tfNgConfigInterface).
 */
void myMobileAddrNotifyFunc(
    ttUserInterface interfaceHandle,
    unsigned int multiHomeIndex,
    struct sockaddr_storage * ipv6AddrPtr,
    int event )
{
    int errorCode;

    switch(tf6MnGetStatus())
    {
    case TM_6_MN_FOREIGN_NETWORK:
        if (l3MovePendingStatus)
        {
            /* Whenever we detect that we have moved, if we have moved to
             * a new foreign link we need to discover a new primary care-of
             * address and (re)register our mobility binding. Only global
             * scope care-of addresses can be used ([SCOPE_02].R11:30). */
            if (tm_6_is_addr_global(
                &(ipv6AddrPtr->addr.ipv6.sin6_addr)))
            {
                if (event == TM_6_DEV_ADDR_CONFIG_COMPLETE)
                {
                    /* We found a new primary care-of address */
                    tfMemCpy(
                        &mnCareOfAddr,
                        ipv6AddrPtr,
                        sizeof(struct sockaddr_storage));
                    haveMnCareOfAddrStatus = 1;

#ifdef TM_USE_IKE
                    if (!startedIkeStatus)
                    {
                        startedIkeStatus = 1;
                    }
#endif

                    /* Start IKE between the MN and the HA for a specific home address.
                     * When the MN moves and changes its care-of address, the IKE end point
                     * will be replaced with new care-of address, and all previous phase 1
                     * SA will be transferred to that new care-of address too. */
                    errorCode = tf6MnStartEnhancedIke(
                        &mnHomeAddr,
                        &mnCareOfAddr,
                        TM_DOI_ID_FQDN,
                        strlen(mobileNodeIdentity),
                        mobileNodeIdentity);

                    /* Add preshared key for the Home Agent. At the HA side, user needs to
                     * add this same preshared key for mobile node's home address. */
                    errorCode = tfPresharedKeyAdd(
                        homeAgentIdentity, ikePreSharedKey, 0);
                }
            }
        }
    }
}

```

```

    }
#endif /* TM_USE_IKE */

/* Attempt to register a mobility binding for our home address with
the home agent to notify it of our new primary care-of address.
Note that the mobile node will automatically re-register with
the home agent before the granted lifetime expires. */
    errorCode = tf6MnRegisterBinding(
        &mnHomeAddr, /* home address */
        &mnCareOfAddr, /* care-of address */
        3600, /* request 1-hour lifetime */
        myMnBindingNotifyFunc,
        0, /* no timeout */
        0); /* no flags */

    l3MovePendingStatus = 0;
}
}
}
else if (haveMnCareOfAddrStatus)
{
    if (IN6_ARE_ADDR_EQUAL(
        &(ipv6AddrPtr->addr.ipv6.sin6_addr),
        &(mnCareOfAddr.addr.ipv6.sin6_addr))
        && (event == TM_6_DEV_ADDR_INVALIDATED))
    {
        haveMnCareOfAddrStatus = 0;

/* Our primary care-of address has been invalidated. We need to
pick a new primary care-of address on the mobile interface
and re-register. */
        errorCode = tf6GetLocalIpAddress(
            mobileInterface,
            &mnHomeAddr,
            TM_6_LOCAL_ADDR_CURSOR_NULL_PTR,
            1, /* start the search */
            &mnCareOfAddr);

        if (errorCode == TM_ENOERROR)
        {
            haveMnCareOfAddrStatus = 1;

/* register our new primary care-of address. */
            errorCode = tf6MnRegisterBinding(
                &mnHomeAddr, /* home address */
                &mnCareOfAddr, /* care-of address */
                3600, /* request 1-hour lifetime */
                myMnBindingNotifyFunc,
                0, /* no timeout */
                0); /* no flags */
        }
    }
}
break;

#ifdef TM_6_MN_DISABLE_HOME_DETECT
/* If your home network is virtual (i.e. the mobile node will
never be directly attached to it), this can significantly
simplify Mobile IPv6 processing. In that case you should
enable the macro TM_6_MN_DISABLE_HOME_DETECT. */

```

```

case TM_6_MN_HOME_NETWORK:
    if (IN6_ARE_ADDR_EQUAL(
        &(mnHomeAddr.addr.ipv6.sin6_addr),
        &(ipv6AddrPtr->addr.ipv6.sin6_addr)))
    {
/* This address configuration event is for our home address */
        switch(event)
        {
/* When operating at home, we need to pay attention to Duplicate
Address Detection failures for our home address because this
may mean that the home agent is defending that address from
a previous registration, in which case we should try to
deregister. This can happen if the mobile node is powered down
on a foreign link while it still has active mobility bindings
in the home agent, and then powered back up on its home link
while those mobility bindings are still active. In this case,
the mobile node is probably not aware that it still has active
mobility bindings. */
            case TM_6_DEV_ADDR_CONFIG_FAILED:
/* If you know that your network controller on the mobile
interface does not loopback its own multicasts, then you can
make this deregistration logic more robust by enabling the
following code. */
#if 0
                case TM_6_DEV_ADDR_DUP_DETECTED:
#endif
                    if (haveHomeAgentAddrStatus)
                    {
/* attempt to deregister our home address with the home agent */
                        errorCode = tf6MnRegisterBinding(
                            &mnHomeAddr, /* home address */
/* deregistration: care-of address is home address */
                            &mnHomeAddr, /* care-of address */
                            0, /* deregistration: 0 lifetime */
myMnBindingNotifyFunc,
                            0, /* no timeout */
                            0); /* no flags */
                    }
                    else
                    {
/* The mobile node should know the IP address of its home agent
since it should persist that in NVRAM. If it does not know the
IP address of its home agent, it can do DHAAD on its home network
to attempt to find that, but if there are multiple home agents,
it may not pick the right one to deregister with. Note that
tf6MnGetHomeAgentAddress is non-blocking, and therefore if you
call it here you will need to set a global flag and finish up
the deregistration logic elsewhere (i.e. you cannot go into a
loop in this callback waiting for tf6MnGetHomeAgentAddress to
complete, because that will hang the receive task, and then
DHAAD will never complete). */
                        tfKernelWarning(
                            "myMobileAddrNotifyFunc",
                            "cannot block for DHAAD in callback.");
                    }
                    break;

                default:
                    break;
            }
        }
    }
}

```

```

    }
    break;
#endif / ! TM_6_MN_DISABLE_HOME_DETECT */

    default:
        break;
}

return;
}

#ifdef TM_USE_IPSEC
/* myAddIpsecPolicy is called by the mobile node to add the
 * security policies used to secure messages sent between the mobile
 * node and the home agent. Note that similar configuration must also
 * be done on the home agent.
 */
int myAddIpsecPolicy(struct sockaddr_storage TM_FAR * homeAddrPtr,
                     struct sockaddr_storage TM_FAR * homeAgentPtr)
{
    int                errorCode = 0;

    inet_ntop(
        PF_INET6,
        &(homeAddrPtr->addr.ipv6.sin6_addr),
        homeAddrString,
        INET6_ADDRSTRLEN);

    inet_ntop(
        PF_INET6,
        &(homeAgentPtr->addr.ipv6.sin6_addr),
        homeAgentString,
        INET6_ADDRSTRLEN);

    {

/* Setup the selector table */

        ttIpsecSelectorInString plcySelector[] =
        {
/* selector #0 ANY to ANY */
            {
/* local IP address and its mask (or prefix length, or IP address
range) */
                (char*)0,          0,
/* remote IP address and its mask (or prefix length, or IP address
range)*/
                (char*)0,          0,
/* IP address type flags. */
                (ttUser16Bit)0,
/* local port */
                TM_SELECTOR_WILD_PORT,
/* remote port */
                TM_SELECTOR_WILD_PORT,
/* protocol */
                TM_SELECTOR_WILD_PROTOCOL
            },

/* selector Home Address ->

```

```

    Home Agent Address, Mobile Header protocol */
    {
/* local IP, i.e. the Mobile Node's home address */
    (char*)homeAddrString,    0,
/* remote IP, i.e., the Home Agent */
    (char*)homeAgentString,  0,
/* both the local IP and remote IP are of type host
(not subnet nor IP address range) */
    TM_SELECTOR_BOTHIP_HOST,
/* local port, don't care */
    TM_SELECTOR_WILD_PORT,
/* remote port, don't care */
    TM_SELECTOR_WILD_PORT,
/* protocol must be Mobility Header */
    IPPROTO_MH,
    },

/* NOTE: you may also want to define
selector Home-address -> Home-Agent, ICMPv6 protocol
to protect home prefix discovery messages */

/* selector Mobile Node -> ANY, Mobile Header protocol */
    {
/* local IP, i.e. the Mobile Node's home address */
    homeAddrString,    0,
/* remote IP, can be any Correspondent Node */
    (char*)0,          0,
/* local IP is a host (not a subnet nor IP address range) */
    TM_SELECTOR_LOCIP_HOST,
    TM_SELECTOR_WILD_PORT,
    TM_SELECTOR_WILD_PORT,
    IPPROTO_MH,
    }

/* NOTE: you may also want to define
selector Mobile Node -> ANY, any protocol
to protect payload packets. */
    };

/* Setup the policy content table */

    ttPolicyContentInString plcyContent[] =
    {
/* the default policy: typically bypass IPsec, or discard */
    {
/* local tunnel end */
    (char*)0,
/* remote tunnel end */
    (char*)0,
/* policy content: bypass IPsec. TM_PFLAG_DISCARD will discard */
    TM_PFLAG_BYPASS,
/* authentication algorithm, 0 = NULL */
    0,
/* encryption algorithm, 0 = NULL */
    0,
/* lifetime for the security association, in seconds. Use default
value */
    0,
/* lifetime for the security association, in kbytes. Use default

```

```

value */
    0
},

/* The policy content used to protect the BU/BA exchange between the
mobile node and the home agent. You may want to use this same
policy content to protect ICMPv6 home prefix discovery messages. */
{
/* local tunnel endpoint, not used for transport mode */
(char*)0,
/* remote tunnel endpoint, not used for transport mode */
(char*)0,
/* policy content: ESP, Transport */
TM_PFLAG_ESP | TM_PFLAG_TRANSPORT,
/* authentication algorithm: MD5 in HMAC mode */
SADB_AALG_MD5HMAC,
/* encryption algorithm: 3DES in CBC mode */
SADB_EALG_3DESCBC,
/* rekey the IPsec SA every 2 hours, i.e. 7200 seconds */
7200,
/* rekey the IPsec SA every 0 kbytes. Use the default value. */
0
},

/* The policy content used to protect the HoTI/HoT messages tunneled
through the HA as part of the return routability procedure. You may
want to use this same policy content to protect payload packets. */
{
/* local tunnel endpoint, typically the MN's primary care-of address.
When you set this security policy at home, you don't know this
care-of address yet, so you can specify it as 0. Whenever the
primary care-of address changes, IPsec automatically updates this
tunnel endpoint because you have specifies the policy flag
TM_PFLAG_MIPV6_HA_TUNNEL. */
(char*)0,
/* remote tunnel end, has to be the Home Agent */
homeAgentString,
/* policy content: ESP, tunnel, supports Key Management Mobility
Capability (TM_PFLAG_MIPV6_HA_TUNNEL). */
TM_PFLAG_ESP | TM_PFLAG_TUNNEL | TM_PFLAG_MIPV6_HA_TUNNEL,
/* authentication algorithm: SHA1, HMAC mode */
SADB_AALG_SHA1HMAC,
/* encryption algorithm: AES, CBC mode */
SADB_EALG_AESCBC,
/* use default seconds value for rekeying */
120,
/* use default kbytes value for rekeying */
0
}
};

/* Setup the policy pair table that associates security policy
selectors with security policy contents */

ttIpsecPolicyPair plcyPair[] =
{

/* SELECTOR INDEX, POLICY CONTENT INDEX, DIRECTION APPLIES TO */

```

```

/* 3.0: any-any, bypass, the last resort, applies to both directions */
    {0, 0, TM_IPSEC_BOTH_DIRECTION},
/* 3.1: MN <-> HA, mobile header protocol, ESP transport, applies to
both directions */
    {1, 1, TM_IPSEC_BOTH_DIRECTION},
/* NOTE: if you want to protect ICMPv6 home prefix discovery
messages, you may define here:
    3.2: MN <-> HA, ICMPv6, ESP transport */

/* NOTE: If you want to protect payload packets, the policy matching
pair must be defined before the return routability one. So, you may
define here:
    3.3: MN <-> any, any protocol, on tunnel interface, use ESP tunnel
*/

/* 3.4: MN <-> any, Mobile header protocol, on tunnel interface, use
ESP tunnel, applies to both directions */
    {2, 2, TM_IPSEC_BOTH_DIRECTION}
};

/* When IPsec queries the Security Policy Database to find a security
policy matching on the selector values of a packet being sent or
received, it first tries to match policy pair 3.4, and then 3.3 and
3.2 (not implemented in this example), and then 3.1, and the last
resort match (i.e. default rule) is 3.0 which in this example says
to bypass IPsec (i.e. do not apply any IPsec to this packet). */

/* Initialize the Security Policy Database */
    errorCode = tfPolicyRestore(
        plcyPair,      /* matches selectors to policy content*/
        plcySelector, /* selector table */
        plcyContent,  /* policy content table */
        3); /* three entries in plcyPair <0,0>, <1,1>, and <2,2> */
    }
return errorCode;
}

/* We manually add SAs to protect the HoTI and HoT messages. Note that
we do not support the use of IKE for this purpose. */
int myAddManualSa(struct sockaddr_storage TM_FAR * homeAddrPtr)
{

    ttPolicyEntryPtr    plcyPtr;
    ttIpsecSelectorPtr  selPtr;
    ttChildSaInfo       sainfo;
    ttSadbRecordPtr     sadbPtr;
    int                 errorCode;

    errorCode = TM_ENOERROR;
    memset(&sainfo,0, sizeof(sainfo));

/* You need to know the policy according to which you are going to add
SA manually. We know it is policy index #2. */
    tfIpsecPolicyQueryByIndex(2, &plcyPtr);
    if(plcyPtr == (ttPolicyEntryPtr)0)
    {
        return -1;
    }

/* Assemble the selector */

```

```

selPtr = &sainfo.chdPacketSelector;

/* local IP == MN's home address, remote is any (all zeros). */
selPtr->selIpFlags = TM_SELECTOR_LOCIP_HOST;
memcpy(&selPtr->selLocIpl,
       homeAddrPtr,
       sizeof(struct sockaddr_storage));

/* specify that the protocol is Mobility Header */
selPtr->selProtocol = IPPROTO_MH;

/* Assemble the SA information, for example, the lifetime value, the
SPI number, the key length */

sainfo.chdAuthKeyBytes = 20 ; /* for SHA1, key length = 20 bytes*/
sainfo.chdEncryptKeyBytes = 16; /* for AES */
sainfo.chdLifetimeKbytes = 0 ; /* use default, 0x7fff kbytes */
sainfo.chdLifetimeSeconds = 8*3600; /* time value 8 hours */
sainfo.chdMySpi = 0x1111; /* peer -> I use this */
sainfo.chdPeerSpi = 0x2222; /* I -> peer use this */
sainfo.chdPlcyPtr = plcyPtr;

/* one policy may have multiple bundled policy contents */
sainfo.chdPlcyContentPtr = plcyPtr->plcyContentPtr;

/* Manually add SAs, one for each direction */
/* SA for inbound/receiving direction */
errorCode = tfSadbRecordGenerate(&sadbPtr,
                                &sainfo,
                                (unsigned char *)inKeyMat,
                                (unsigned short)
                                (sainfo.chdAuthKeyBytes +
                                 sainfo.chdEncryptKeyBytes),
                                TM_IPSEC_INBOUND);

if(errorCode != 0)
{
    goto COMMON_RETURN;
}

/* Tell IPsec don't use IKE to rekey this SA. If you don't use IKE at
all, you don't have to make this function call. */
tfSadbRecordSetOptions(sadbPtr, TM_IPSEC_SADB_NOREKEYING, 1);

/* SA for outbound/sending direction */
errorCode = tfSadbRecordGenerate(&sadbPtr,
                                &sainfo,
                                (unsigned char *)outKeyMat,
                                (unsigned short)
                                (sainfo.chdAuthKeyBytes +
                                 sainfo.chdEncryptKeyBytes),
                                TM_IPSEC_OUTBOUND);

if(errorCode != 0)
{
    goto COMMON_RETURN;
}
tfSadbRecordSetOptions(sadbPtr, TM_IPSEC_SADB_NOREKEYING, 1);

COMMON_RETURN:

return errorCode;

```

```
}
#endif /* TM_USE_IPSEC */

void main (void)
{
    struct sockaddr_storage    ipNgAddr;
    ttUserLinkLayer           linkLayerHandle;
    int                        errorCode;
    ...

    /* Start the Treck stack */
    tfStartTreck();

    /* We use the Ethernet link-layer in this example */
    linkLayerHandle = tfUseEthernet();

    /* Add the mobile interface */
    mobileInterface = tfAddInterface(
        "mobile.if0",
        linkLayerHandle,
        ...
    );

    /* Read our local IP address (mobile node home address) from NVRAM */
    tfMemSet(&mnHomeAddr, 0, sizeof(struct sockaddr_storage));
    mnHomeAddr.ss_family = AF_INET6; /* IPv6 address */
    mnHomeAddr.ss_len = sizeof(struct sockaddr_storage);
    errorCode = myEEReadMnHomeAddr(
        &(mnHomeAddr.addr.ipv6.sin6_addr));
    if (errorCode != TM_ENOERROR)
    {
        /* This example expects that you already know the home address.
        IPsec must be configured to protect the Binding Update message
        that the mobile node sends to the home agent, and in many cases
        this configuration is dependent on the specific home address. */
        tfKernelError("main", "Failed to get home address.");
    }
    haveMnHomeAddrStatus = 1;

    /* Initialize global variable used to keep track of our primary
    care-of address (we have not discovered any care-of address
    yet). */
    tfMemSet(&mnCareOfAddr, 0, sizeof(struct sockaddr_storage));
    mnCareOfAddr.ss_family = AF_INET6; /* IPv6 address */
    mnCareOfAddr.ss_len = sizeof(struct sockaddr_storage);

    /* Open the mobile interface for use with IPv6. This will cause the
    interface ID to be set on the interface. We might be attached to
    a foreign network, in which case we don't want to configure
    our home address on the mobile interface until we have first
    called tf6MnStartMobileIp to determine if we are operating at
    home or not. */
    tfMemSet(&ipNgAddr, 0, sizeof(struct sockaddr_storage));
    ipNgAddr.ss_family = AF_INET6; /* IPv6 address */
    ipNgAddr.ss_len = sizeof(struct sockaddr_storage);

    /* start Stateless Address Auto-Configuration on the mobile
    interface */
    errorCode=tfNgOpenInterface(
        mobileInterface,
```

```

    &ipNgAddr, /* auto-configuration: IPv6 address all 0's */
    0, /* auto-configuration: prefix length not applicable */
    configFlags,
    TM_6_DEV_OPTIMIZE_DAD | TM_6_USE_AUTO_IID, /* IPv6 flags */
    scatteredBufferCount,
    myMobileAddrNotifyFunc);

/* Wait for auto-configuration to complete */
do
{
    tfKernelTaskYield(); /* allow other tasks to run */
    errorCode = tfCheckOpenInterface(mobileInterface, AF_INET6);
}
while (errorCode == TM_EINPROGRESS);

/* Read the home agent address from NVRAM. Once a specific home
agent is located and successfully registered with by the mobile
node, the mobile node should continue to use that same home agent
as long as there is the possibility that that home agent could
still be maintaining active mobility bindings for it. */
tfMemSet(&homeAgentAddr, 0, sizeof(struct sockaddr_storage));
homeAgentAddr.ss_family = AF_INET6; /* IPv6 address */
homeAgentAddr.ss_len = sizeof(struct sockaddr_storage);
errorCode = myEEReadHomeAgentAddr(
    &(homeAgentAddr.addr.ipv6.sin6_addr));
if (errorCode == TM_ENOERROR)
{
    haveHomeAgentAddrStatus = 1;
}
else
{
/* Attempt DHAAD to discover a home agent on our home network */
while(!haveHomeAgentAddrStatus)
{
    errorCode = tf6MnGetHomeAgentAddress(
        &mnHomeAddr, /* home subnet prefix */
        64, /* home prefix length */
        &homeAgentAddr, /* store home agent address here */
        sizeof(struct sockaddr_storage),
        TM_BLOCKING_ON); /* blocking mode, preemptive RTOS */

    if (errorCode > 0)
    {
        errorCode = myEEWriteHomeAgentAddr(
            &(homeAgentAddr.addr.ipv6.sin6_addr));

        haveHomeAgentAddrStatus = 1;
    }
}
}

/* Setup the IPsec security policy database based on the
combination of mobile node home address and home agent address */
#ifdef TM_USE_IPSEC
/* Initialize IPsec global variables */
errorCode = tfUseIpsec();
if(errorCode != TM_ENOERROR)
{
    return;
}

```

```
    }
/* Add the security policy */
    errorCode = myAddIpsecPolicy(
        &mnHomeAddr, &homeAgentAddr);
    if(errorCode != TM_ENOERROR)
    {
        return;
    }

/* manually add the SA to protect HoTI and HoT (we use IKE to negotiate
SAs used between MN and HA to protect BU/BA exchange) */
    errorCode = myAddManualSa(&mnHomeAddr);
    if(errorCode != TM_ENOERROR)
    {
        return;
    }
#endif /* TM_USE_IPSEC */

/* Set the interface ID on the virtual home interface to be the
same as the interface ID on the mobile interface. This ensures
that any auto-configured home addresses on the virtual home
interface will match what the mobile node would normally
auto-configure when it is at home. */
    errorCode = tf6Eui64GetInterfaceId(
        mobileInterface, mobileInterfaceId);

/* Enable Mobile IPv6 mobile node functionality */
    virtualHomeInterface = tf6MnStartMobileIp(
        mobileInterface,
        mobileInterfaceId,
        &homeAgentAddr,
        &mnHomeAddr, /* home address prefix */
        64, /* home address prefix length */
        TM_6_MN_NOTIFY_FUNC_NULL_PTR,
        myMnNotifyFunc, /* mobile node event notification */
        0, /* no flags */
        &errorCode);

/* Configuration of our home address on the appropriate interface
(i.e. mobile interface if we are at home, otherwise virtual
home interface) will occur in the callback function
myMnNotifyFunc. */

/* Your application loop starts here */
    while(1)
    {
/* Do some application stuff using your home address. To use your
home address, you bind it to a socket as the local address.
Note: while you are operating on a foreign network, it is
generally acceptable for short-term communication that can
easily be retried if it fails (i.e. a DNS resolver query) to
use a care-of address auto-configured on the mobile interface
instead of your home address ([MIPV6_18++].R11.3.1:40). */
        ;
    }

/* When you shutdown, remember to deregister, otherwise you might
find yourself back on your home network with your home agent
still defending your home address causing a Duplicate Address
Detection failure when you try to configure it. */
```

```
/* Attempt to deregister our home address with the home agent. By
default (see Treck option TM_6_OPTION_MN_DEREG_TIMEOUT), the
mobile node will timeout after 10 seconds if it hasn't
successfully deregistered. */
errorCode = tf6MnStopMobileIp(0); /* no flags */
if (errorCode == TM_ENOERROR)
{
    while (mnHomeAddrRegistered)
    {
/* wait 10 seconds for deregistration to complete or to time-out */
        tfKernelTaskYield(); /* allow other tasks to run */
    }
}

return;
}
```

### 3 Limitations

We currently do not implement any mobile node support for the “Fast Handovers for Mobile IPv6” and “Hierarchical MIPv6 mobility management (HMIPv6)” draft RFCs.

We are not implementing optimistic DAD for care-of addresses configured on the mobile interface. However, we already do a form of optimistic DAD with the link-local scope IPv6 address auto-configured on the interface, so we can start using that address as a source address as soon as we complete the move process. Also, we provide the IPv6 interface flag `TM_6_DEV_OPTIMIZE_DAD` which, when specified when opening the mobile interface for use with IPv6 by calling `tfNgOpenInterface` (see the example code), bypasses performing DAD on a stateless auto-configured care-of address, which per RFC-2462 we can do since that address is formed from the same interface ID as the auto-configured link-local scope address. By specifying this IPv6 interface flag whenever you call `tfNgOpenInterface/tfNgConfigInterface` on the mobile interface, you will essentially achieve the same fast handover performance as optimistic DAD.

We do not support the MN sending a BU with the (L) bit set.

We restrict the MN to use only a single home agent.

[IPSECMIPV6\_05] requires the MN to support the use of IKE run between the MN and HA to secure the return routability HoT/HoTI message exchange. However, this cannot be reliably implemented with per-interface SPDs on the HA, therefore we chose not to support this feature. Instead, IPsec manually keyed SAs can be used between the MN and HA to secure this message exchange.

## 4 Function Reference

### 4.1 Changes to Existing Public APIs

#### 4.1.1 `tfInitTreckOptions`

Refer to `tfSetTreckOptions` below.

#### 4.1.2 `tfSetTreckOptions`

---

Please See `tfSetTreckOptions` in Chapter 5 'Programmer's Reference of the Turbo Treck TCP/IP User's Manual

The following are additional configuration options for Mobile IPv6 Support.

---

<code>TM_6_OPTION_MIP_RO_ENABLE</code>	A boolean used to globally enable/disable Mobile IPv6 route optimization, applies to both Correspondent Node and Mobile Node. To disable, set to 0. <b>Default: 1, enabled</b>
<code>TM_6_OPTION_MAX_BINDING_LIFETIME</code>	The maximum lifetime (in seconds) that a Mobile IPv6 correspondent node supports for a binding cache entry. The Mobile IPv6 mobile node also uses this as an inactivity timeout to deregister route optimization binding update list entries that aren't being actively used for exchanging application data. <b>Default: 420 seconds (7 minutes)</b>
<code>TM_6_OPTION_MN_REG_LT_BIAS</code>	The amount of time (in seconds) that the mobile node will downward bias (for the purpose of re-registration) the binding registration lifetime granted by the home agent. This can be used to force Mobile IPv6 to re-register more frequently than required by the home agent. This does not affect the lifetime used to expire/delete the mobility binding, which is still the original lifetime returned by the home agent. <b>Default: 300 seconds (5 minutes)</b>
<code>TM_6_OPTION_MN_DEREG_TIMEOUT</code>	The default timeout period (in seconds) for a mobile node attempt to automatically deregister a mobility binding. <b>Default: 10 seconds</b>
<code>TM_6_OPTION_RR_MAX_BCE_ENTRIES</code>	The maximum number of correspondent node binding cache entries. <b>Default: 8</b>
<code>TM_6_OPTION_RR_MAX_BUL_ENTRIES</code>	The maximum number of mobile node route optimization binding update list entries used to track whether or not a specific correspondent node we are communicating with supports route optimization, as well as to keep track of the state of route optimization when it does. Note that this does not apply to home registration binding update list entries, which are limited by the maximum number of home addresses that can be configured on the virtual home interface (i.e. <code>TM_MAX_IPS_PER_IF * 2</code> ). <b>Default: 16</b>
<code>TM_6_OPTION_MN_BEACON_THRESHOLD</code>	The maximum number of consecutive Router Advertisement "beacons" from its current default router that the mobile node can tolerate dropping before initiating a L3 handover. This method of move detection is enabled only when the current default router includes the Advertisement Interval option in the Router Advertisement message. Set to 0 to disable this functionality. <b>Default: 4</b>
<code>TM_6_OPTION_MN_1ST_REG_INIT_TIMEOUT</code>	The amount of time to delay in milliseconds before the initial retransmission of the home registration Binding Update when a mobile node does not already have an existing binding with the home agent. This corresponds to the <code>InitialBindackTimeoutFirstReg</code> Mobile IPv6 protocol configuration variable. <b>Default: 1500 (1.5 seconds)</b>
<code>TM_6_OPTION_MN_EAGER_CELL_SWITCH</code>	A boolean used to globally enable/disable Mobile IPv6 mobile node Eager Cell Switching movement detection algorithm. To enable, set to 1; this causes the mobile node to perform a L3 handover whenever it receives a Router Advertisement from a router that is different from its current default router. <b>Default: 0, disabled</b>

---

### 4.1.3 getsockopt

---

Please See `getsockopt` in *BSD 4.4 Socket API* section of Chapter 5 ‘Programmer’s Reference’ in the Turbo Treck TCP/IP User’s Manual. The following are additional configuration options for Mobile IPv6 Support.

---

Option Name	Description
TM_6_IPO_MN_USE_HA_TUNNEL	IPPROTO_IPV6 <i>protocolLevel</i> option. Enable/disable use of the MN-to-HA tunnel when the Mobile Node sends packet originating on a specific socket. Default is 0 (disabled).

### 4.1.4 setsockopt

---

Please See `setsockopt` in *BSD 4.4 Socket API* section of Chapter 5 ‘Programmer’s Reference’ in the Turbo Treck TCP/IP User’s Manual. The following are additional configuration options for Mobile IPv6 Support.

---

Option Name	Description
TM_6_IPO_MN_USE_HA_TUNNEL	IPPROTO_IPV6 <i>protocolLevel</i> option. Enable/disable use of the MN-to-HA tunnel when the Mobile Node sends packets originating on a specific socket. Default is 0 (disabled). Normally, the MN will try to use Mobile IPv6 route optimization when communicating with a peer. This socket option, when enabled (i.e. set to 1) on a specific socket, disables the use of route optimization for all packets that the MN sends on that socket.

## 4.2 New Public APIs

### 4.2.1 tf6MnGetHomeAgentAddress

```

int                tf6MnGetHomeAgentAddress
(
const struct sockaddr_storage * homePrefixPtr,
int homePrefixLen,
struct sockaddr_storage * homeAgentAddrPtr,
int addrBufferLen,
int flags
);

```

#### Function Description

*This function only supports IPv6.*

The user calls this function to discover the IPv6 address of a Mobile IPv6 Home Agent on the specified home network (i.e. *homePrefixPtr*). This function uses Mobile IPv6 Dynamic Home Agent Address Discovery (DHAAD) to attempt to discover home agent addresses. Note that once you register the MN with a specific home agent, you should continue to use that same home agent for all future registration and deregistration attempts.

The user must ensure that the user buffer pointed to by *homeAgentAddrPtr* is at least *addrBufferLen* bytes long, and must be at least `sizeof(struct sockaddr_storage)` bytes long to be able to hold at least one home agent address. When multiple home agent addresses are returned in the DHAAD reply, this function stores as many of those addresses as will fit in the buffer pointed to by *homeAgentAddrPtr* and returns the number of addresses stored.

#### Parameters

Parameter	Description
<i>homePrefixPtr</i>	This points to an IPv6 address prefix that is an on-link prefix on the home network, and it should be global scope and cannot be link-local scope.
<i>homePrefixLen</i>	This is the prefix length of the address prefix pointed to by <i>homePrefixPtr</i> . The value specified must be in the range of 4 to 128, and will typically be 64.
<i>homeAgentAddrPtr</i>	When this function returns a value greater than 0, <i>homeAgentAddrPtr</i> points to an array of home agent addresses on the specified home network. The memory that this points to must be allocated by the caller, e.g. a local variable in the caller's address space. The home agent addresses can be accessed as <i>homeAgentAddrPtr[0]</i> , <i>homeAgentAddrPtr[1]</i> , etc. for the number of home agent addresses returned.
<i>addrBufferLen</i>	Length (in bytes) of the user buffer pointed to by <i>homeAgentAddrPtr</i> . To be able to hold at least one home agent address, this buffer must be at least <code>sizeof(struct sockaddr_storage)</code> bytes long.
<i>Flags</i>	Flags for use with <b>tf6MnGetHomeAgentAddress</b> . The desired flags are OR'ed together. The following flags can be used with this API: <code>TM_BLOCKING_ON</code> When <i>flags</i> is specified as 0, operation is non-blocking.

#### Returns

Value	Meaning
>0	Success. The home agent address(es) discovered on the specified home network are stored in the user buffer pointed to by <i>homeAgentAddrPtr</i> .
=0	No home agent was discovered on the specified home network.
-1	An error occurred. The error code can be retrieved by calling <i>tfGetSocketError(-1)</i> .

---

**Error Codes****Error Code**

TM\_EINPROGRESS

**Meaning**

This function could not complete processing without blocking, and TM\_BLOCKING\_ON was not specified in *flags*. You should call this API again specifying the exact same parameters until it either returns success (TM\_ENOERROR) or an error.

TM\_EINVAL

One of the parameters was invalid:

1. *homePrefixPtr* was NULL or did not point to a valid `sockaddr_in6` structure holding an IPv6 global scope or site-local scope unicast address.
2. *homePrefixLen* was invalid for the address prefix specified by *homePrefixPtr*. In most cases, *homePrefixLen* should be set to 64.
3. *homeAgentAddrPtr* is NULL
4. *addrBufferLen* was less than `sizeof(struct sockaddr_storage)`

## 4.2.2 tf6MnGetStatus

```

tt6MnStatus      tf6MnGetStatus
(
void
);

```

### Function Description

*This function only supports IPv6.*

This function returns status indicating whether or not Mobile IPv6 Mobile Node functionality is enabled, and when it is enabled then this status also indicates whether or not the mobile node is operating on its home network. Note that it is possible during L3 handover processing that Mobile Node functionality is enabled, but since we are moving to a new link we do not know yet whether we are at home or away from home; this API returns `TM_6_MN_UNKNOWN_NETWORK` in that case.

### Parameters

None.

### Returns

#### Value

`TM_6_MN_DISABLED`

`TM_6_MN_HOME_NETWORK`

`TM_6_MN_FOREIGN_NETWORK`

`TM_6_MN_UNKNOWN_NETWORK`

#### Meaning

Mobile Node functionality is currently disabled. Call `tf6MnStartMobileIp` to enable.

The mobile node is operating on its home network.

The mobile node is operating on a foreign network.

The mobile node is in the process of determining whether it is at home or away from home (i.e. L3 handover processing).

### 4.2.3 tf6MnMoveNotify

```
int          tf6MnMoveNotify
(
ttUserInterface  mobileIfaceHandle,
tt6MnMoveType    moveType,
int              flags
);
```

#### Function Description

*This function only supports IPv6.*

This function is called by the user to notify Mobile IPv6 that a L2 or L3 handover has just completed. The primary goal of Mobile IPv6 move detection is to detect L3 handovers. Handover is the process of a MN moving from an old link to a new link. In some wireless network implementations, a change of physical link may not always require a change of care-of address (i.e. both the old link and the new link may be on the same subnet). This change of physical link is referred to as a L2 handover. If the L2 handover results in a change of subnet and thus requires a change of care-of address, then it is referred to as a L3 handover.

Note that if the mobile node has been “off channel” (i.e. in a RF null) for a period of time that would affect network protocol operation, and then comes back “on channel”, the user should call **tf6MnMoveNotify** specifying the appropriate move type. For example, if the user knows for sure that the MN has returned to the exact same link that it was operating on before it lost network connectivity, the user would notify the stack of a TM\_6\_MNME\_L2\_SAME\_SUBNET move.

#### Parameters

##### Parameter

*mobileIfaceHandle*

##### Description

The interface handle of the currently active mobile interface you are completing the handover on, which is a real physical interface that the MN is using to send/receive messages on. If your handover consists of changing the mobile interface (for example, closing the Bluetooth interface, and opening the 802.11b interface), then this must be the interface handle for the link that you are moving to. It is very important that you specify the correct mobile interface, because this is the interface that will be used to discover default routers, auto-configure care-of addresses, perform move detection, etc.

*moveType*

The type of handover that has just completed, one of:

TM\_6\_MNME\_L2\_UNKNOWN  
 TM\_6\_MNME\_L2\_SAME\_SUBNET  
 TM\_6\_MNME\_L3\_NEW\_SUBNET

*flags*

Flags for use with **tf6MnMoveNotify**. The desired flags are OR’ed together. Currently no flags are defined, so this parameter must be specified as 0.

#### Returns

##### Value

TM\_ENOERROR

TM\_EPERM

##### Meaning

Move notification was successful.

You must start the Mobile IPv6 Mobile Node first (i.e. call **tf6MnStartMobileIp**) before calling this API.

#### 4.2.4 tf6MnRegisterBinding

```

int                tf6MnRegisterBinding
(
const struct sockaddr_storage * homeAddrPtr,
const struct sockaddr_storage * careOfAddrPtr,
int                maxHaBindingLifetimeSec,
tt6MnBindingNotifyFuncPtr mnBindingNotifyFuncPtr,
int                timeoutSec,
int                flags
);

```

##### Function Description

*This function only supports IPv6.*

This function is called by the user to register a new mobility binding in the home agent that associates a home address with a (new) care-of address. The home address specified by the user in *homeAddrPtr* must have already been (manually or auto-) configured on the virtual home interface. Calling this function causes the mobile node to register the specified mobility binding with the home agent, and to attempt route optimization with correspondent nodes when the source address of the communication is the specified home address.

The user should choose the lifetime (i.e. *maxHaBindingLifetimeSec*) of mobility bindings that it registers with the home agent to be less than the lifetime of the security associations that the MN has with the home agent. Note that the mobile node will automatically re-register the specified mobility binding with the home agent before the granted lifetime expires (refer to the Treck option `TM_6_OPTION_MN_REG_LT_BIAS`).

This function can be called to deregister an existing mobility binding by specifying the new care-of address to be the same as the home address or by specifying the mobility binding lifetime (i.e. *maxHaBindingLifetimeSec*) to be 0. Note that it is not necessary for the user application to explicitly deregister active mobility bindings when the mobile node returns home, because that is automatically done by the mobile node.

If *mnBindingNotifyFuncPtr* is non-null, then the user wants to be notified of Mobile IPv6 binding events via a user-defined notify function. The function prototype for the user-defined notify function (specified by *mnBindingNotifyFuncPtr*) is:

```

void myMnBindingNotifyFunc
(
struct sockaddr_storage * homeAddrPtr,
struct sockaddr_storage * careOfAddrPtr,
tt6MnBindingEvent event,
int status
);

```

where *event* is the Mobile IPv6 binding event (i.e. `TM_6_MNBE_HA_REG_SUCCESS`, ...) for the specified mobility binding (i.e. *homeAddrPtr* associated with *careOfAddrPtr*), and *status* is either the value of the status field received in the Binding Acknowledgement message from the home agent or else `-1` if this event is not associated with a received Binding Acknowledgement message.

The Mobile IPv6 binding events you can be notified of are as follows:

Mobile IPv6 Binding Event	Description
TM_6_MNBE_HA_REG_SUCCESS	Registration with the home agent was successful for the specified mobility binding. The parameter <i>status</i> contains the status code from the home agent Binding Acknowledgement message.
TM_6_MNBE_HA_REG_FAILED	Registration with the home agent failed for the specified mobility binding. The parameter <i>status</i> contains the status code from the home agent Binding Acknowledgement message: 128 - Reason unspecified 129 - Administratively prohibited 130 - Insufficient resources 131 - Home registration not supported 132 - Not home subnet 133 - Not home agent for this mobile node 134 - Duplicate Address Detection failed
TM_6_MNBE_HA_REG_TIMEOUT	Registration with the home agent failed for the specified mobility binding due to not receiving the Binding Acknowledgement message from the home agent. You will not get this event if you specify 0 for <i>timeoutSec</i> .
TM_6_MNBE_HA_REG_REFRESH	The mobile node is initiating the procedure to refresh the specified active mobility binding with the home agent.
TM_6_MNBE_HA_DEREG_SUCCESS	Deregistration with the home agent was successful for the specified mobility binding.
TM_6_MNBE_HA_DEREG_FAILED	Deregistration with the home agent failed for the specified mobility binding. The parameter <i>status</i> contains the status code from the home agent Binding Acknowledgement message. This may not be a real failure, since it may just indicate that the home agent was not maintaining an active mobility binding (in which case there is no need for the mobile node to deregister the mobility binding).
TM_6_MNBE_HA_DEREG_TIMEOUT	Deregistration with the home agent failed for the specified mobility binding due to not receiving the Binding Acknowledgement message from the home agent. When you call <b>tf6MnStopMobileIp</b> the mobile node will automatically deregister your mobility bindings, in which case the timeout period is specified using the Treck option TM_6_OPTION_MN_DEREG_TIMEOUT and the callback function that will be called is the one you specified when you registered the mobility binding.
TM_6_MNBE_HA_BINDING_TIMEOUT	The granted lifetime of the specified mobility binding has expired and the mobility binding was removed on the mobile node.

## Parameters

### Parameter

*homeAddrPtr*

### Description

Pointer to the IPv6 unicast home address for which we want to register a mobility binding associating it with the specified care-of address. It is expected that most home agents will not support tunneling of site-local scope traffic to a site-local scope home address, therefore the home address should be global scope. NOTE: the home address cannot be link-local scope.

*careOfAddrPtr*

Pointer to the IPv6 global scope unicast primary care-of address for the mobility binding. If you want to deregister an existing mobility binding, set the primary care-of address to be the same as the home address, or alternately specify the mobility binding lifetime (i.e. *maxHaBindingLifetimeSec*) to be 0.

*maxHaBindingLifetimeSec*

The maximum binding lifetime (in seconds) that the MN can request when it sends a Binding Update message to the HA to register the specified mobility binding. The actual value sent in the BU may be less, depending on the valid lifetimes of the home address and primary care-of address associated with the mobility binding. Specifying a binding lifetime of 0 will deregister all active mobility bindings for the specified home address. Note that the mobile node will automatically re-register with the home agent before the granted lifetime expires. Note that you cannot specify an infinite binding lifetime using this API.

<i>mnBindingNotifyFuncPtr</i>	The function to call to notify the user of Mobile IPv6 binding events, or <code>TM_6_MNB_NOTIFY_FUNC_NULL_PTR</code> if notification is not desired. The events are: <code>TM_6_MNBE_HA_REG_SUCCESS</code> <code>TM_6_MNBE_HA_REG_FAILED</code> <code>TM_6_MNBE_HA_REG_TIMEOUT</code> <code>TM_6_MNBE_HA_DEREG_SUCCESS</code> <code>TM_6_MNBE_HA_DEREG_FAILED</code> <code>TM_6_MNBE_HA_DEREG_TIMEOUT</code> <code>TM_6_MNBE_HA_BINDING_TIMEOUT</code>
<i>timeoutSec</i>	The number of seconds to keep trying to register/deregister without receiving a response from the home agent. Specify 0 for no timeout (i.e. infinite retries). After the timeout period elapses with no response from the home agent, if you have registered a callback function to be notified of Mobile IPv6 binding events (see the parameter <i>mnBindingNotifyFuncPtr</i> ) you will be notified of the appropriate timeout event.
<i>flags</i>	Flags for use with <b>tf6MnRegisterBinding</b> . The desired flags are OR'ed together. Currently no flags are defined, so this parameter must be specified as 0.

**Returns****Value**

TM\_ENOERROR

TM\_EINVAL

TM\_EPERM

**Meaning**

Creation (or deregistration) of the specified mobility binding was successfully initiated.

One of the parameters was invalid:

1. *homeAddrPtr* was NULL, or did not point to a valid `sockaddr_in6` structure holding an IPv6 global scope or site-local scope unicast address which is currently configured on the virtual home interface.
2. *careOfAddrPtr* was NULL, or did not point to a valid `sockaddr_in6` structure holding an IPv6 global scope unicast address.

You must start the Mobile IPv6 Mobile Node first (i.e. call **tf6MnStartMobileIp**) before calling this API.

## 4.2.5 tf6MnStartEnhancedIke

```
int                tf6MnStartEnhancedIke
(
  const struct sockaddr_storage *  homeAddrPtr,
  const struct sockaddr_storage *  careOfAddrPtr,
  unsigned int    idType,
  unsigned int    idLength,
  ttUserVoidPtr  idPtr
);
```

### Function Description

*This function only supports IPv6.*

This function can be called by the user immediately prior to the first call to **tf6MnRegisterBinding** for a specific home address to start IKE for use between the MN and the HA. This is done to support the (K) Key Management Mobility Capability bit used in the MN-to-HA BU/BA exchange, which enables the MN to avoid renegotiation of IKE phase 1 SAs whenever the primary care-of address changes.

### Parameters

<b>Parameter</b>	<b>Description</b>
<i>homeAddrPtr</i>	Pointer to the IPv6 unicast home address for which we want to register a mobility binding associating it with the specified care-of address. It is expected that most home agents will not support tunneling of site-local scope traffic to a site-local scope home address, therefore the home address should be global scope. NOTE: the home address cannot be link-local scope.
<i>careOfAddrPtr</i>	Pointer to the IPv6 global scope unicast primary care-of address for the mobility binding.
<i>idType</i>	Refer to the user documentation for <b>tfStartEnhancedIke</b> .
<i>idLength</i>	Refer to the user documentation for <b>tfStartEnhancedIke</b> .
<i>idPtr</i>	Refer to the user documentation for <b>tfStartEnhancedIke</b> .

### Returns

<b>Value</b>	<b>Meaning</b>
TM_ENOERROR	IKE was started successfully for the specified home address.
TM_EINVAL	One of the parameters was invalid: <ol style="list-style-type: none"> <li>1. <i>homeAddrPtr</i> was NULL, or did not point to a valid <code>sockaddr_in6</code> structure holding an IPv6 global scope or site-local scope unicast address which is currently configured on the virtual home interface.</li> <li>2. <i>careOfAddrPtr</i> was NULL, or did not point to a valid <code>sockaddr_in6</code> structure holding an IPv6 global scope unicast address.</li> </ol>
TM_EPERM	You must start the Mobile IPv6 Mobile Node first (i.e. call <b>tf6MnStartMobileIp</b> ) before calling this API.

## 4.2.6 tf6MnStartMobileIp

```

ttUserInterface          tf6MnStartMobileIp
(
ttUserInterface          mobileIfaceHandle,
const ttUser8Bit *       virtHomeEui64IdPtr,
const struct sockaddr_storage * homeAgentAddrPtr,
const struct sockaddr_storage * homePrefixPtr,
int                      homePrefixLen,
tt6DevAddrNotifyFuncPtr virtHomeAddrNotifyPtr,
tt6MnNotifyFuncPtr      mnNotifyFuncPtr,
int                      flags,
int TM_FAR *            errorCodePtr
);

```

### Function Description

*This function only supports IPv6.*

The user calls this function to enable Mobile IPv6 Mobile Node functionality. Before calling **tf6MnStartMobileIp**, the user must first determine the global scope IPv6 address of the home agent, since that information is passed to **tf6MnStartMobileIp** using the *homeAgentAddrPtr* parameter. This may be discovered by calling **tf6MnGetHomeAgentAddress**.

Once you find a working home agent for your home address(es), you should store the home agent address that you are using in NVRAM, so that you can continue to use that same home agent. However, if for some reason that home agent is no longer reachable, and this is not due to the MN not being able to communicate with the network, you may need to change the home agent address.

When you change the home agent address, be very careful to first deregister all of your active mobility bindings with the original home agent by calling **tf6MnRegisterBinding**. Otherwise, you may have to wait until the lifetime expires for your previous registrations with your previous home agent. This is one argument against using long mobility binding lifetimes.

It is important that the MN always knows whether it is operating on its home network or on a foreign network. When the MN receives a solicited router advertisement from its IPv6 current default router that contains the on-link prefix specified by the parameters *homePrefixPtr* and *homePrefixLen*, it knows that it is operating on its home network, otherwise it assumes that it is operating on a foreign network. This detection is enabled by default, but can be disabled by #define'ing the macro `TM_6_MN_DISABLE_HOME_DETECT`, in which case the stack assumes that the mobile node is never attached to its home network.

**tf6MnStartMobileIp** returns a pointer to a special interface that the stack creates for the purpose of maintaining your home addresses when the MN is operating on a foreign network. This special interface is called the *virtual home interface*. The virtual home interface cannot be opened or closed by the user, and is automatically opened by the stack when the mobile node detects that it has moved from home, then later automatically closed by the stack when the mobile node detects that it has returned to home or when the user calls **tf6MnStopMobileIp** to disable Mobile IPv6 mobile node functionality. When `TM_6_MN_DISABLE_HOME_DETECT` is #define'd, the mobile node is assumed to never be operating at home, therefore the virtual home interface is opened immediately by **tf6MnStartMobileIp** in that case.

When the MN is not operating on its home network, it uses Mobile IPv6 Home Prefix Discovery to discover its home network address prefixes from the home agent. Based on this information, it uses Prefix Discovery (if you have #define'd `TM_6_USE_PREFIX_DISCOVERY`) to auto-configure home addresses on the virtual home interface. You may want to be notified when new home addresses are configured on this interface, in which case you can pass a pointer to your notification function using the *virtHomeAddrNotifyPtr* parameter. This works just the same as any IPv6 address configuration notification function that you would register using **tfNgConfigInterface**/**tfNgOpenInterface**, but it only notifies you of address configuration events for the virtual home interface. The interface ID used by the virtual home interface for stateless address auto-configuration is specified using the parameter *virtHomeEui64IdPtr*.

If *mnNotifyFuncPtr* is non-null, then the user wants to be notified of Mobile IPv6 Mobile Node events via a user-defined notify function. The function prototype for the user-defined notify function (specified by *mnNotifyFuncPtr*) is:

```

void myMnNotifyFunc(
ttUserInterface interfaceHandle,
struct sockaddr_storage * routerAddrPtr,
tt6MnEvent event);

```

where *event* is the Mobile IPv6 Mobile Node event (i.e. `TM_6_MNE_SET_NEW_RTR`, ...) for the specified IPv6 router address (i.e. *routerAddrPtr*) on the specified mobile interface (i.e. *interfaceHandle*).

Note that when Mobile IPv6 Mobile Node functionality is enabled, the stack filters Prefix Discovery and Parameter Discover to use the current IPv6 default router, and additionally filters Router Discovery to use the currently active mobile interface.

The Mobile IPv6 Mobile Node events you can be notified of consist of Mobile IPv6 Home Agent reachability and IPv6 default router discovery and reachability:

Mobile IPv6 Mobile Node Event	Description
<code>TM_6_MNE_HA_REACHABLE</code>	The home agent specified by the notify function parameter <i>routerAddrPtr</i> has been confirmed to be bi-directionally reachable, and is functioning as a Mobile IPv6 home agent. This event only occurs when reachability of the home agent is in question, i.e. after you call <b>tf6MnStartMobileIp</b> specifying a new home agent address, or after you are notified of the event <code>TM_6_MNE_HA_UNREACHABLE</code> .
<code>TM_6_MNE_HA_UNREACHABLE</code>	The home agent specified by the notify function parameter <i>routerAddrPtr</i> either did not respond to us, is not functioning as a Mobile IPv6 home agent, or is not allowing us to register. When you receive this event, if you suspect that the home agent is permanently unreachable (i.e. this is not just due to a transient failure of the mobile interface link), you should first call <b>tf6MnStopMobileIp</b> to disable Mobile IPv6 mobile node functionality, next determine a new home agent address (i.e. by doing a DNS query, and/or by calling <b>tf6MnGetHomeAgentAddress</b> ), finally restart Mobile IPv6 mobile node functionality by calling <b>tf6MnStartMobileIp</b> .
<code>TM_6_MNE_SET_NEW_RTR</code>	The stack has selected a new current IPv6 default router to use on the mobile interface. IPv6 Neighbor Unreachability Detection is used to determine the reachability of an IPv6 default router. This event is generated after the solicited Router Advertisement from the new IPv6 default router has been completely processed by the mobile node (including any advertised prefixes).
<code>TM_6_MNE_RTR_UNREACHABLE</code>	The current IPv6 default router on the mobile interface is not reachable per IPv6 Neighbor Unreachability Detection. When this event occurs, the stack will automatically call <b>tf6MnMoveNotify</b> specifying <code>TM_6_MNME_L3_NEW_SUBNET</code> for <i>moveType</i> .
<code>TM_6_MNE_MOVE_FROM_HOME</code>	This event indicates that the MN has moved from its home network to a foreign network, or that Mobile IPv6 has initialized on a foreign network. <i>routerAddrPtr</i> points to the address of the current IPv6 default router. When you receive this event, you should configure your home addresses on the virtual home interface, and care-of addresses will normally be auto-configured on your mobile interface.
<code>TM_6_MNE_RETURN_TO_HOME</code>	This event indicates that the MN has moved to its home network after having operated on a foreign network, or that Mobile IPv6 has initialized on the home network. Note that in the case of returning from a foreign network to the home network, the mobile node will already have attempted to deregister all active mobility bindings with the home agent before you receive this event. <i>routerAddrPtr</i> points to the address of the current IPv6 default router. When you receive this event, you should configure your home addresses on your mobile interface. You will not receive this event if you <code>#define TM_6_MN_DISABLE_HOME_DETECT</code> .

**Parameters**

<b>Parameter</b>	<b>Description</b>
<i>mobileIfHandle</i>	The interface handle of the currently active mobile interface, which is a physical device interface that the MN is using to send/receive messages on. It is very important that you specify the correct mobile interface, because this is the interface that Mobile IPv6 uses to discover default routers, auto-configure care-of addresses, perform move detection, etc.
<i>virtHomeEui64IdPtr</i>	Pointer to the 8-byte interface ID to set on the virtual home interface, in EUI-64 format (i.e. 64-bit IEEE global identifier). If you do not want to set an interface ID on the virtual home interface (i.e. you do not want to enable stateless address auto-configuration on the virtual home interface), then you can specify a NULL pointer for this parameter, however in that case you must manually configure a link-local scope address on the virtual home interface. In general, you should set the interface ID on the virtual home interface to be the same as the interface ID that the mobile node uses when operating on its home network (i.e. the interface ID set on the mobile interface). This way, the same home addresses will be auto-configured on the virtual home interface as would normally be auto-configured when the mobile node is operating on its home network. To do this, you can first open the mobile interface (if the link-layer is Ethernet or PPP, the interface ID is set after the interface is opened), call the API <b>tf6Eui64GetInterfaceId</b> to get the interface ID of the mobile interface, and then pass a pointer to that interface ID to <b>tf6MnStartMobileIp</b> using the <i>virtHomeEui64IdPtr</i> parameter.
<i>homeAgentAddrPtr</i>	This points to the global scope IPv6 address of the home agent. This address could be site local scope, if a global scope address is not available.
<i>homePrefixPtr</i>	This points to an IPv6 address prefix that the MN uses to detect when it is operating on the home network versus on a foreign network. This IPv6 address prefix must be an on-link prefix on the home network, and it should be global scope and cannot be link-local scope. If you know your global scope home address, you should specify that as your home prefix, otherwise your home agent address (if it is global scope) should work in most cases as your home prefix.
<i>homePrefixLen</i>	This is the prefix length of the address prefix pointed to by <i>homePrefixPtr</i> . The value specified must be in the range of 4 to 128, and will typically be 64.
<i>virtHomeAddrNotifyPtr</i>	If you want to be notified when a new IPv6 home address is auto-configured on the virtual home interface using Mobile IPv6 Home Prefix Discovery, you can specify a notification function for this purpose. This corresponds to the <i>dev6AddrNotifyFuncPtr</i> parameter of <b>tfNgOpenInterface</b> for the virtual home interface. Note that if you call <b>tfNgConfigInterface</b> on the virtual home interface to manually configure home addresses, you must specify this same function if you want to receive these notifications.
<i>mnNotifyFuncPtr</i>	The function to call to notify the user of Mobile IPv6 Mobile Node events, or <code>TM_6_MN_NOTIFY_FUNC_NULL_PTR</code> if notification is not desired. The events are: <pre> TM_6_MNE_HA_REACHABLE TM_6_MNE_HA_UNREACHABLE TM_6_MNE_SET_NEW_RTR TM_6_MNE_RTR_UNREACHABLE TM_6_MNE_MOVE_FROM_HOME TM_6_MNE_RETURN_TO_HOME </pre>
<i>flags</i>	Flags for use with <b>tf6MnStartMobileIp</b> . The desired flags are OR'ed together. Currently no flags are defined, so this parameter must be specified as 0.
<i>errorCodePtr</i>	A pointer to an int that will contain an error code if an error occurred.

**Returns****Value**

NULL

!=NULL

**Meaning**

Failed starting Mobile IPv6. *errorCodePtr* points to the error code (see below).

Pointer to the virtual home interface. This interface is created by the stack, and is managed (i.e. opened, closed) internally by the stack. When Mobile IPv6 is active (i.e. after **tf6MnStartMobileIp** is called but before **tf6MnStopMobileIp** is called) and the mobile node is operating away from home, the virtual home interface is open, and then the user can call most public APIs (with the exception of those that open and close the interface) on this interface to manually configure addresses (**tfNgConfigInterface**), to select home addresses to bind to sockets (**tf6GetLocalIpAddress**), etc. Additionally, when the MN is operating on a foreign network and `TM_6_USE_PREFIX_DISCOVERY` is `#define'd`, the stack auto-configures home addresses on this interface using Mobile IPv6 Home Prefix Discovery.

**Error Codes****Value**

TM\_ENOERROR

TM\_EALREADY

TM\_EINVAL

**Meaning**

Mobile IPv6 Mobile Node was successfully started.

Mobile IPv6 Mobile Node is already running.

One of the parameters was invalid:

1. *mobileIfaceHandle* did not point to a valid interface.
2. *homeAgentAddrPtr* was NULL or did not point to a valid `sockaddr_in6` structure holding an IPv6 global scope or site-local scope unicast address.
3. *homePrefixPtr* was NULL or did not point to a valid `sockaddr_in6` structure holding an IPv6 global scope or site-local scope unicast address.
4. *homePrefixLen* was invalid for the address prefix specified by *homePrefixPtr*. In most cases, *homePrefixLen* should be set to 64.

## 4.2.7 tf6MnStopMobileIp

```
int          tf6MnStopMobileIp
(
int          flags
);
```

### Function Description

*This function only supports IPv6.*

This function disables Mobile IPv6 mobile node functionality and closes the virtual home interface, which also causes auto-deregistration of existing mobility bindings with the home agent and with correspondent nodes. The timeout used for auto-deregistration is specified by the Treck option `TM_6_OPTION_MN_DEREG_TIMEOUT`. This API must be called if it is ever necessary to change the home agent address that Mobile IPv6 is using.

If you registered a callback function when creating mobility bindings (i.e. `mnBindingNotifyFuncPtr` parameter of **tf6MnRegisterBinding**), that same callback function will be called for notification of deregistration events when the stack auto-deregisters those bindings.

### Parameters

#### Parameter

*flags*

#### Description

Flags for use with **tf6MnStopMobileIp**. The desired flags are OR'ed together. The following flags can be used with this API: `TM_BLOCKING_ON` When *flags* is specified as 0, operation is non-blocking.

### Returns

#### Value

`TM_ENOERROR`

#### Meaning

Mobile IPv6 Mobile Node functionality has been stopped. Note that it is possible that some active mobility bindings are not successfully deregistered with the home agent if the home agent is unreachable or does not respond to our deregistration requests within the timeout period specified by the Treck option `TM_6_OPTION_MN_DEREG_TIMEOUT`. In this case, you could run into Duplicate Address Detection registration failures (`TM_6_MNBE_HA_REG_FAILED` binding event, *status* 134 "Duplicate Address Detection failed") trying to register those same home addresses with a different home agent until the mobility bindings finally expire in the original home agent. Because of this potential system issue, it is preferable to use short lifetimes when registering mobility bindings with the home agent, and it is preferable to avoid changing home agents once you have found one that works.

`TM_EINPROGRESS`

Initiated stop of Mobile IPv6 Mobile Node. The mobile node is currently in the process of deregistering active bindings with the home agent.

## 5 Macros

Macro Name	Value	Meaning
<b>TM_6_USE_MIP_CN</b>	(disabled)	Enable Mobile IPv6 Correspondent Node functionality.
<b>TM_6_USE_MIP_MN</b>	(disabled)	Enable Mobile IPv6 Mobile Node functionality.
<b>TM_6_USE_MIP_RO</b>	(enabled when TM_6_USE_MIP_CN is enabled)	Enable Mobile IPv6 route optimization (RO) (includes return routability procedure). ([MIPV6_18++].R14.4.5:10, [MIPV6_18++].R14.4.5:20)
<b>TM_6_MN_DISABLE_HOME_DETECT</b>	(disabled, so home detection/return to home is enabled)	Disable Mobile IPv6 Mobile Node home detection/return to home functionality. If you know that the MN never returns home, then you can define this macro to save some code space.
<b>TM_6_USE_MIP_RA_RTR_ADDR</b>	(enabled)	When enabled, the mobile node uses the Router Address option (when available) in received Router Advertisement messages to determine if a Router Advertisement is from the current default router.
<b>TM_6_IPO_MN_USE_HA_TUNNEL</b>	0x2000	<b>setsockopt/getsockopt IPPROTO_IPV6 protocolLevel</b> option used to disable Mobile IPv6 route optimization for a specific Mobile Node user application socket.
<b>TM_6_OPTION_MIP_RO_ENABLE</b>	110	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . A boolean used to globally enable/disable Mobile IPv6 route optimization, applies to both Correspondent Node and Mobile Node. ([MIPV6_18++].R14.4.5:10, [MIPV6_18++].R14.4.5:20)
<b>TM_6_OPTION_MAX_BINDING_LIFETIME</b>	111	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The maximum lifetime (in seconds) that a Mobile IPv6 correspondent node supports for a binding cache entry. The Mobile IPv6 mobile node also uses this as an inactivity timeout to deregister route optimization binding update list entries that aren't being actively used for exchanging application data.
<b>TM_6_OPTION_MN_REG_LT_BIAS</b>	112	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The amount of time, in seconds, that the mobile node will downward bias (for the purpose of re-registration) the Registration Lifetime returned by the home agent.
<b>TM_6_OPTION_MN_DEREG_TIMEOUT</b>	113	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The default timeout period (in seconds) for a mobile node attempt to automatically deregister a mobility binding.
<b>TM_6_OPTION_RR_MAX_BCE_ENTRIES</b>	114	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The maximum number of correspondent node binding cache entries.

TM_6_OPTION_RR_MAX_BUL_ENTRIES	115	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The maximum number of mobile node route optimization binding update list entries used to track whether or not a specific correspondent node we are communicating with supports route optimization, as well as to keep track of the state of route optimization when it does. Note that this does not apply to home registration binding update list entries, which are limited by the maximum number of home addresses that can be configured on the virtual home interface (i.e. TM_MAX_IPS_PER_IF * 2).
TM_6_OPTION_MN_BEACON_THRESHOLD	116	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The maximum number of consecutive Router Advertisement "beacons" from its current default router that the mobile node can tolerate dropping before initiating a L3 handover. This method of move detection is enabled only when the current default router includes the Advertisement Interval option in the Router Advertisement message. Set to 0 to disable this functionality.
TM_6_OPTION_MN_1ST_REG_INIT_TIMEOUT	117	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . The amount of time to delay in milliseconds before the initial retransmission of the home registration Binding Update when a mobile node does not already have an existing binding with the home agent. This corresponds to the InitialBindackTimeoutFirstReg Mobile IPv6 protocol configuration variable. (ISSUE181].R13:50)
TM_6_OPTION_MN_EAGER_CELL_SWITCH	118	Option for <b>tfSetTreckOptions/tfInitTreckOptions</b> . A boolean used to globally enable/disable Mobile IPv6 mobile node Eager Cell Switching movement detection algorithm.
TM_6_MNME_L2_UNKNOWN	(tt6MnMoveType) 0	Indicates that a L2 handover just completed, the subnet may or may not have changed. Refer to <b>tf6MnMoveNotify</b> .
TM_6_MNME_L2_SAME_SUBNET	(tt6MnMoveType) 1	Indicates that a L2 handover just completed but the subnet did not change. Refer to <b>tf6MnMoveNotify</b> .

TM_6_MNME_L3_NEW_SUBNET	<b>(tt6MnMoveType) 2</b>	Indicates that a L3 handover just completed, the subnet has changed. Refer to tf6MnMoveNotify. Normally, the MN will detect this type of move without this notification by detecting that the previous IPv6 default router has become unreachable using Neighbor Unreachability Detection and then discovering a new default router that is on a different subnet, but this method is slow. Indicates that the home agent is reachable. Refer to tf6MnStartMobileIp.
TM_6_MNE_HA_REACHABLE	<b>(tt6MnEvent) 0</b>	Indicates that the home agent is not responding to us. Refer to tf6MnStartMobileIp.
TM_6_MNE_HA_UNREACHABLE	<b>(tt6MnEvent) 1</b>	Indicates that we have set a new IPv6 default router. Refer to tf6MnStartMobileIp.
TM_6_MNE_SET_NEW_RTR	<b>(tt6MnEvent) 2</b>	Indicates that our current IPv6 default router is unreachable, which results in the mobile node calling tf6MnMoveNotify for move type TM_6_MNME_L3_NEW_SUBNET . Refer to tf6MnStartMobileIp.
TM_6_MNE_RTR_UNREACHABLE	<b>(tt6MnEvent) 3</b>	Indicates that we have completed a move from the home network to a foreign network. Refer to tf6MnStartMobileIp.
TM_6_MNE_MOVE_FROM_HOME	<b>(tt6MnEvent) 4</b>	Indicates that we have completed a move from a foreign network back to the mobile node's home network. Refer to tf6MnStartMobileIp.
TM_6_MNE_RETURN_TO_HOME	<b>(tt6MnEvent) 5</b>	Indicates registration with the home agent was successful for the specified mobility binding. Refer to tf6MnRegisterBinding.
TM_6_MNBE_HA_REG_SUCCESS	<b>(tt6MnBindingEvent) 0</b>	Indicates registration with the home agent failed for the specified mobility binding. Refer to tf6MnRegisterBinding.
TM_6_MNBE_HA_REG_FAILED	<b>(tt6MnBindingEvent) 1</b>	Indicates the home agent did not respond to our registration request. Refer to tf6MnRegisterBinding.
TM_6_MNBE_HA_REG_TIMEOUT	<b>(tt6MnBindingEvent) 2</b>	Indicates the mobile node is automatically reregistering before the mobility binding in the home agent expires. Refer to tf6MnRegisterBinding.
TM_6_MNBE_HA_REG_REFRESH	<b>(tt6MnBindingEvent) 3</b>	Indicates deregistration with the home agent was successful for the specified mobility binding. Refer to tf6MnRegisterBinding.
TM_6_MNBE_HA_DEREG_SUCCESS	<b>(tt6MnBindingEvent) 4</b>	

TM_6_MNBE_HA_DEREG_FAILED	<b>(tt6MnBindingEvent)</b> <b>5</b>	Indicates deregistration with the home agent failed for the specified mobility binding (which may not be a real failure). Refer to <code>tf6MnRegisterBinding</code> .
TM_6_MNBE_HA_DEREG_TIMEOUT	<b>(tt6MnBindingEvent)</b> <b>6</b>	Indicates the home agent did not respond to our deregistration request. Refer to <code>tf6MnRegisterBinding</code> .
TM_6_MNBE_HA_BINDING_TIMEOUT	<b>(tt6MnBindingEvent)</b> <b>7</b>	Indicates the specified mobility binding has expired and been removed. Refer to <code>tf6MnRegisterBinding</code> .
TM_6_MN_NOTIFY_FUNC_NULL_PTR	<b>(tt6MnNotifyFuncPtr)</b> <b>0</b>	NULL pointer value for the <code>mnNotifyFuncPtr</code> parameter of <code>tf6MnStartMobileIp</code> .
TM_6_MNB_NOTIFY_FUNC_NULL_PTR	<b>(tt6MnBindingNotifyFuncPtr)</b> <b>0</b>	NULL pointer value for the <code>mnBindingNotifyFuncPtr</code> parameter of <code>tf6MnRegisterBinding</code> .
TM_6_MN_DISABLED	<b>(tt6MnStatus)</b> <b>0</b>	Indicates mobile node functionality is currently disabled. Refer to <code>tf6MnGetStatus</code> .
TM_6_MN_HOME_NETWORK	<b>(tt6MnStatus)</b> <b>1</b>	Indicates the mobile node is operating on its home network. Refer to <code>tf6MnGetStatus</code> .
TM_6_MN_FOREIGN_NETWORK	<b>(tt6MnStatus)</b> <b>2</b>	Indicates the mobile node is operating on a foreign network. Refer to <code>tf6MnGetStatus</code> .
TM_6_MN_UNKNOWN_NETWORK	<b>(tt6MnStatus)</b> <b>3</b>	Indicates the mobile node is in the process of determining whether it is at home or away from home (i.e. L3 handover processing). Refer to <code>tf6MnGetStatus</code> .

## 6 Acronyms

This section lists some acronyms that are used in this document, and their meaning.

AP	The MN's link-layer access point to the network. The AP acts as a link-layer relay that relays link-layer frames to the subnet.
AR	Access router. An Access Router is the last router between the network and the mobile node, i.e., the MN has link-layer connectivity to the Access Router.
BA	Mobile IPv6 Binding Acknowledgement message.
BCE	Correspondent node Binding Cache Entry. This is also referred to as mobility binding.
BE	Mobile IPv6 Binding Error message.
BER	Bit error rate, can be used by L2 as a measure of signal quality. BER applies to either transmit or receive.
BRR	Mobile IPv6 Binding Refresh Request message.
BU	Mobile IPv6 Binding Update message. "BU" is the content of the Binding Update packet, excluding (1) the IP header, (2) any extension headers between the IP header the Mobility Header, and (3) the Authenticator field inside the Authorization Data mobility option appearing in the packet.
BUL	Mobile Node Binding Update List. This is used to keep track of the active bindings that the MN has created in the HA and in CNs for its home addresses.
CDPD	Cellular Digital Packet Data. This is a packet data overlay over AMPS cellular, technology circa 1996 and soon to become obsolete. CDPD was an early implementation of Mobile IPv4 over cellular wireless.
CN	Correspondent Node: a node with which a NM communicates at the application layer.
CoA	Care-of address: the temporarily assigned IPv6 address that a MN uses at the network layer to be able to send packets on a foreign subnetwork. This will be on the same IPv6 subnetwork as the subnetwork that the MN is physically attached to.
CoT	Mobile IPv6 Care-of Test message. Sent by a CN to a MN in response to a CoTI.
CoTI	Mobile IPv6 Care-of Test Init message. A mobile node uses the Care-of Test Init (CoTI) message to initiate the return routability procedure and request a care-of keygen token from a correspondent node.
DAD	Duplicate Address Detection
DHAAD	Dynamic HA Address Discovery
HA	Mobile IPv6 Home Agent. This is the Mobile IPv6-aware router on the Mobile Node's home network.
HoT	Mobile IPv6 Home Test message. Sent by a CN to a MN in response to a HoTI.
HoTI	Mobile IPv6 Home Test Init message. A mobile node uses the Home Test Init (HoTI) message to initiate the return routability procedure and request a home keygen token from a correspondent node.
Kbm	A binding management key (Kbm) is a key used for authorizing a binding cache management message (e.g., Binding Update or Binding Acknowledgement). Return routability provides a way to create a binding management key.
Kcn	Each correspondent node has a secret key, Kcn, called the "node key", which it uses to produce the keygen tokens sent to the mobile nodes.
L2	The link-layer part of the protocol stack, which can include the device driver.
MIPv6	Mobile IPv6
MN	Mobile Node: an IPv6 node capable of dynamically changing its point-of-attachment to the Internet in such a way as to allow its applications to seamlessly operate over/across a change of IPv6 address/subnetwork at the network layer.
NUD	IPv6 Neighbor Unreachability Detection.
PKI	Public Key Infrastructure, which makes a specific host's public key available for a MN to use to establish an IPsec security association with that host.
RA	ICMPv6 Router Advertisement
RF	Radio frequency (i.e. wireless) mode of operation.
RO	Route optimization

RR	Return routability
RS	ICMPv6 Router Solicitation
RSSI	Receive signal strength indicator. Can be used by L2 as a measure of signal quality.
SA	IPsec security association.
SADB	IPsec security association database.
SPD	IPsec security policy database.
ULP	Upper layer protocol (i.e. UDP, TCP).