

Overview of the MIPv6 Implementation

Tunneling

Tunneling support was added as it is necessary for MIPv6.

Interfaces have interfaceIDs that uniquely identify them. Similarly, every tunnel has a virtual interfaceID. These virtual IDs start at INT_MAX and are constantly decremented for every tunnel.

IPv6.cc – routePacket():

This method checks whether there is a tunnel for the destination address by calling the getVifIndexForDest() method from the tunnelling module.

In case there is a tunnel (interfaceID is larger than the number of available “real” interfaces), the packet is sent to the tunnelling module. The tunnelling has upper and lower input and output gates as recommended by RFC 2473. Due to this, to some extent it behaves like a normal interface from the IPv6 module point of view.

IPv6Tunneling.cc – encapsulate():

The new source and destination addresses are determined and a new controllInfo object added that contains this information. The datagram is sent back to the IPv6 module...

IPv6.cc – handleMessageFromHL():

...where it will be treated as a higher layer input and therefore be encapsulated with the new addresses from the controllInfo.

When the IPv6 module receives an encapsulated datagram from the lower layers, it will be automatically end up in...

IPv6.cc - localDeliver():

...where decapsulation takes place and the protocol ID of the next header will indicate IP. The packet is therefore sent to the Tunneling module.

IPv6Tunneling.cc – decapsulate():

The datagram will be sent back to the IPv6 module.

IPv6.cc – localDeliver():

The tunnelling header is now stripped from the datagram and the “original non-encapsulated” datagram is now processed.

Within the currently called method, a second decapsulation takes place and reveals the transport layer header (or whatever header was transported).

Tunnels are created and destroyed by direct method calls of the Tunneling module.

Pseudo-Tunnels

Mobile IPv6 provides the feature of Route Optimization that allows for direct routing between the MN and CN. In this case, from the MN point of view the tunnel to the HA has to be bypassed and the packet directly sent to the CN together with the Home Address Option contained in the Destination Option extension header.

This would require a binding cache lookup from within the IPv6 module. In order to not add a dependency on xMIPv6.h within IPv6.cc, this lookup is realized with help of the tunnelling module.

When creating a tunnel, the MIPv6 module indicates the tunnel type:

- IPv6Tunneling::NORMAL
- IPv6Tunneling::T2RH
- IPv6Tunneling::HA_OPT

When performing a lookup for tunnels (e.g. from within the IPv6.cc – routePacket()), it is checked whether the destination is covered by any of the three types of tunnels. Tunnels of type T2RH and HA_OPT are looked up first, followed by NORMAL tunnels. In case a T2RH or HA_OPT path is used for encapsulation, the

IPv6Tunneling.cc – encapsulate()

decapsulates the provided datagram and appends the appropriate extension header to the control info. This control info is processed in

IPv6.cc – encapsulate()

where the extension headers are copied to the datagram itself.

Mobile IPv6

The INET IPv6 core class (“IPv6.cc”), the Neighbor Discovery protocol (“Network/ICMPv6/IPv6NeighborDiscovery.cc”) and the Routing Table (“Network/IPv6/RoutingTable6.cc”) have been modified.

The main parts of the MIPv6 implementation are the files in the “Network/xMIPv6” folder.

Mobile Node

NeighborDiscovery.cc - processRAPacket()

When the MN receives a Router Advertisement (RA), its content is checked as follows.

If the Home Agent Flag is set, the prefix advertised in the RA is taken as Home Network Prefix, a Home Address (HoA) based on this prefix formed and the MN's Home Agent set to the source address of the RA (as global unicast address).

processRAPrefixInfoForAddrAutoConf():

In every other case (s.a. the MN visiting a foreign network), a Care-of address (CoA) is formed and Duplicate Address Detection (DAD) is initiated.

processDADTimeout():

As soon as DAD has been successfully completed, the MIPv6 protocol is initiated.

xMIPv6.cc - initiateMIPv6Protocol() and subsequent methods:

The very first step of the MIPv6 protocol is the Home Registration; hence a timer is created that points to an object of the data structure BUTransmitIfEntry that contains all necessary information.

The timer is scheduled and immediately fires.

sendPeriodicBU():

The information contained in the BUTransmitIfEntry associated to the timer is extracted, the BU created and sent and the timer rescheduled. Retransmission of BUs takes place until a Binding Acknowledgement (BA) has been received.

processBAMessage():

Upon receiving a positive BA to a home registration, a tunnel to the HA is established. Route Optimization to already known Correspondent Nodes (CNs) is triggered. As bindings have a limited lifetime, an appropriate expiration timer is scheduled.

triggerRouteOptimization():

Route Optimization is triggered from within the Tunneling module as soon as a packet from a CN that has been reverse tunnelled via the HA is received by the MN. If there is no entry in the Binding Update List (BUL) for this CN or the MN has moved to a new network and therefore configured a new CoA, the Return Routability procedure to this CN is triggered.

initReturnRoutability():

In case no home or care-of tokens are available, the HoTI and CoTI messages are created and sent. In case valid tokens are already available or have become available after having received HoT and/or CoT, a BU is sent to the CN.

sendBUtoCN() and subsequent methods:

A timer structure for sending a BU to the CN is initialized; the operation is equal to the one for performing the home registration.

processBAMessage():

As soon as the MN receives a valid BA from the CN, a “pseudo-tunnel” is created. A routing path that uses the Home Address Option path is created via the tunnelling module for direct communication with the CN. The timer for the expiration of the Binding Update List (BUL) entry is scheduled as well.

Home Agent

xMIPv6.cc - processBUMessage():

Having verified that the BU is valid, the Binding Cache Entry (BCE) for this MN is created or updated, a BCE expiration timer set and a BA returned. In addition a tunnel to the MN is established.

Correspondent Node

xMIPv6.cc - processBUMessage():

Operations are similar to those of the HA. The difference is that the CN establishes a “pseudo-tunnel” to create a routing path with the Type 2 Routing Header for direct communication with the MN.

Returning Home

NeighborDiscovery.cc – processRAPrefixInfoForAddrAutoConf():

The MN is considered of having returned to the Home Network based on the following algorithm - after receiving a RA the following conditions are verified:

- The prefix in the RA matches the one of the HoA
- The MN also has a CoA

If both conditions are true, the returning home procedure of the MIPv6 module is called.

xMIPv6.cc – returningHome():

All retransmission and expiry timers are cancelled and the tunnels destroyed that are related to the HA. This is also true for CNs for which Route Optimization (RO) has been performed. For the HA a deregistration BU is sent immediately whereas for the CNs it has to be verified whether it is possible to send a deregistration BU right now. In any case, the mobility state in the BC for the CNs is set to Deregister.

checkForBUtoCN():

In the case the binding cache indicates a Deregister state, the following actions are taken. If a home token is available a deregister BU is immediately sent. Otherwise, a HoTI message is sent.

As soon as a HoT message with a valid token is received the BU for deregistration is sent.

Retransmission Timers

Retransmission timers are used for BU, HoTI and CoTI. These timers are scheduled as normal messages, but have a context pointer to an object of class BUTransmitIfEntry or TestInitTransmitIfEntry. All information that is necessary to construct a new message (BU, HoTI or CoTI) is contained within these objects.

The base class for timers is the TimerIfEntry class.

The Binding Refresh Request (BRR) is available in the source code but not “activated”.

Expiration Timers

Similar to the Retransmission timers we also have timers for the expiration of Binding Cache (BC) entries, Binding Update List (BUL) entries and home & care-of keygen tokens.

They are also derived from the TimerIfEntry class.

The BUL expiry timer is scheduled shortly before the “real” expiration time (difference defined by PRE_BINDING_EXPIRY) within processBAMessage(). This allows renewing the binding before it is completely lost and the tunnels are destroyed.

While the BUL expiry is at the MN, the same holds for the BC expiration at HA and CN. In case a BCE expires, the entry is removed and the tunnels or “pseudo-tunnels” destroyed.

Datagram Handling enhancements for mobility

Modifications to IPv6.cc were necessary as the old code was not suited for mobility.

First, if DAD is still in progress and the appropriate address is therefore marked as tentative, datagrams with this source address can not be sent. These datagrams are therefore rescheduled as an object of class ScheduledDatagram.

As soon as the endService() method is called again with one of the rescheduled datagrams and DAD has finished, these datagrams can be sent to the lower layer.

IPv6.cc – routePacket():

This method was modified to include a lookup in the tunnelling module before the interface lookup.

After interface determination, if the source address of the datagram is tentative, rescheduling as ScheduledDatagram takes places.

localDeliver():

This method has been updated to also handle extension headers (T2RH and Home Address Option).

In addition, datagrams that contain the mobility header are sent to the xMIPv6 module for further processing.

encapsulate():

controllInfo also carries information related to extension headers. If one is present, then this header is copied to the datagram.

Additional Notes

The code was written with multihoming in mind. Therefore many methods contain the interfaceld as parameter; lookup keys and timers contain the ID as well to allow separation between different interfaces and for an “object-per-interface” policy.

DHAAD and Mobile Prefix Solicitation are not implemented.

The Home Agent does not perform DAD for the HoA during first registration; the delay of 1s that results from this procedure is hardcoded inside the implementation.

The current implementation is limited to:

- Only a single HoA
- Only a single CoA per interface
- Single prefix (only one prefix per RA processed)

Neighbor Discovery should be further improved for:

- Better mobility support
- Better Movement Detection (currently movement is considered to have taken place as soon as a new prefix is detected that is different from the one the CoA(s) are based on)
- “Nicer” way of purging the Routing Table and the Neighbor Cache after a movement has taken place and the old entries therefore become invalid (new default router, etc.)