Prototypical Implementation and Experimental Testbed Setup of a QoS-Enabled Mobility Concept Based on HMIPv6

Diplomarbeit
am Fachgebiet Telekommunikationsnetze
Prof. Dr.-Ing. A. Wolisz
Institut für Telekommunikationssysteme
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

vorgelegt von
Axel Neumann

Betreuer: Prof. Dr.-Ing. A. Wolisz
Dr. H. Karl
Dr. X. Fu

Contents

List of Figures

List of Tables

Abstract

Future internetworks will include large numbers of portable devices moving among small, wireless cells. In order to support new real-time applications, the users demand for seamless mobility and QoS provisioning.

One approach towards a more flexible, customizable and scalable mobility architecture that also reduces signaling load and handover latency results from the introduction of micro mobility. This is essentially achieved by moving the point of re-routing from a Corresponding Node (CN) closer to the Mobile Node (MN).

By coupling the Quality of Service (QoS) signaling messages to the mobility management messages, QoS requirements can be negotiated without incurring a significant additional signaling latency. This approach is even more promising through adapting to micro mobility.

This thesis describes the prototypical implementation QoCoo (QoS-Conditionalized Handover) to provide QoS in micro mobility environments. QoCoo extends the existing MIPv6 implementation for the Linux OS to provide functionalities for micro mobility based on the basic mode of HMIPv6, and QoS based on the QoS-Conditionalized BU .

Open issues and possible solutions according to the current specifications have been identified. Finally, several test cases and performance experiments have been conducted to demonstrate that the QoS-Conditionalized BU enables a MN with the capability to signal the MN's QoS requirements to the new path of an access network while performing a handover.

Chapter 1

Introduction

1.1 Motivation

The widespread popularity of mobile communication and Internet Protocol (IP)-based applications indicates an increasing demand for mobile access to the internet suitable for common IP applications as well as Quality of Service (QoS) critical real-time applications.

Traditionally, the Internet employs Internet Protocol version 4 (IPv4) as the crucial protocol for packet delivery in fixed networks. In order to overcome the need for mobility support in IP networks Mobile IP (MIP) was designed to provide transparent mobility to higher layers. However, due to the difficulty of introducing new protocol extensions to the existing IPv4 infrastructure, MIP[30] has never been deployed on a wide basis. The successor of IPv4, the Internet Protocol version 6 (IPv6)[21], has been designed to solve many urgent problems caused by new requirements to the meanwhile historical roots of IPv4. Moreover, IPv6 aims to facilitate integration and deployment of later extensions.

One of these extensions is the Mobile IPv6 (MIPv6) protocol [25] which indeed makes heavy use of the opportunities offered by the new version of IP. The MIPv6 protocol provides the necessary extensions to enable IPv6 with true and transparent mobility support to the application layer. However, with MIPv6 few efforts are undertaken to reduce the duration for re-establishing of connectivity and optimize handover operations regarding latency and signaling overhead.

One approach towards a more flexible, customizable and scalable mobility architecture that also reduces signaling load and handover latency results from the introduction of micro mobility. By way of this, mobility management, related to an MN's handover within a certain region, is hidden from the network part outside that region. The Hierarchical Mobile IPv6 (HMIPv6) protocol [38] provides a solution for support of micro mobility in the MIPv6 protocol.

Regarding the support of QoS sensitive real-time applications, the possibility to maintain an existing QoS assurance while moving between different Access Router (AR)s is required. Since an QoS interruption due to QoS re-establishment on a new path can cause significant service reduction to the application side, one important chal-

lenge for QoS provisioning in mobile IP networks is to employ a mechanism, which keeps handover latency caused by QoS management as small as possible. By coupling the QoS signaling messages to the micro-mobility management messages, QoS requirements can be negotiated with the new data path while performing a handover. This approach has the advantage, that besides the signaling delay and overhead caused by micro-mobility management anyway, no significant additional overhead and delay is introduced.

1.2 Contribution of this Work

In this thesis a prototypical implementation with the name QoCoo (QoS-Conditionalized Handover) to provide QoS in micro mobility has been developed.

QoCoo extends the existing MIPv6 implementation for Linux OS to provide functionalities for support of micro mobility. This is based on the basic mode of HMIPv6 which is specified in [38]. Regarding QoS support, QoCoo is based on the proposal of the QoS-Conditionalized BU, specified in [17].

To achieve this, the following intermediate steps have been conducted: The leveraged architectures IPv6 and MIPv6 have been reviewed and their relevant procedures for this work have been summarized. The underlying concepts by means of descriptional specifications in the HMIPv6 and QoS-Conditionalized BU proposal have been analyzed. Their behavior has been re-examined in a more formal way to provide a basis for the implementation design. Open issues and possible solutions according to the protocol's specifications have been identified and integrated in the implementation design. This implementation design has been made with respect to the selected operating system and MIPv6 implementation environment.

The QoCoo implementation has been setup in a testbed, which serves as a basis for emulating test cases and perform measurements.

1.3 Organization of this Document

The remainder of this thesis is organized as follows:

- In the first part of Chapter 2 general aspects of mobility and QoS in IP networks are reviewed. In the second part, the leveraged architectures for the QoCoo protocol, the IPv6 and MIPv6 protocols are summarized.

- A complete description of the QoS-Conditionalized Handover (QoCoo) scheme is given in Chapter 3. Since QoCoo is based on the architectures of HMIPv6 and the QoS-Conditionalized BU, this includes an informal and a formal description for each of the underlying protocols. At the end of the chapter open issues and possible solutions according the specification of the QoS-Conditionalized BU are summarized.

- In Chapter 4 an overview of the selected implementation environment, the Linux OS, is given and relevant features are explained.

- Chapter 5 provides the implementation design of the QoCoo scheme. It describes how the concepts discussed in Chapter 3 have been applied to the surrounding implementation environment of Chapter 4. A detailed function and data structure reference is given in the appendix of this document.
- A testbed has been setup in order to emulate representative scenarios and demonstrate the key capabilities of the underlying protocols. This has been accomplished by conducting a number of test cases and performance experiments and is explained in Chapter 6.
- Chapter 7 contains the conclusion of this work.

The following parts are contained in the appendix:

- Appendix A lists all the new and modified functions and data structures that have been introduced to the existing implementation environment.
- Appendix B contains additional experimental results and performance measurements which are only summarized in Section 6.3.2 and 6.3.3.
- An installation manual, describing the necessary steps how to setup and use the QoCoo implementation in a testbed, is given in Appendix C.
- Appendix D shows the slides presented in the defense talk of this diploma thesis.
- Appendix E contains the list of acronyms used in the work.

Chapter 2

Mobility and QoS

This chapter first reviews general aspects on mobility and Quality of Service (QoS) support. The focus here lies on IP networks and in particular on the leveraged architectures of this thesis: the IPv6 and MIPv6 protocol. In the second part features of the new IPv6 and MIPv6 protocols are summarized and related signaling procedures are explained.

2.1 Mobility and QoS Aspects in IP Networks

IP has traditionally been designed for fixed networks. Connected hosts were assumed to be located always on the same link and reachable via their IP address. While a human is often referred to by name and date of birth and its location by an affiliated address, the IP address merges identification and location in one single key. This makes it impossible to distinguish the different context of location and identification. With hosts that act only as clients, this is usually no problem since they only have to obtain a new IP address after having moved to a new location. For true mobile hosts there is a dilemma: when sessions are open, the IP address must be constant to preserve session identifiers and to be addressable via one well known IP address. On the other hand, to assure correct routing to the mobile host even when it has moved from one subnetwork to another, the IP address must be changed to reflect the new location.

Regarding the support of service guaranty, QoS support has been well investigated only for fixed IP networks. From a QoS point of view, mobility introduces challenges of retaining a once negotiated and established QoS level for the MN when it changes its point of attachment to the network. Explicit QoS provisioning requires to reserve resources along the data path in every node from the source towards the destination. In general, end hosts make resource reservation in these nodes by way of signaling. The signaling procedure needs to initiate a QoS request from the end host, install the reservations in network nodes and finally to give a response whether the requested operation succeeds. Then, an end host can be informed about whether its traffic can be QoS-guaranteed. The duration between an end host initiating its QoS request and the arrival of the response is called *signaling latency*.

In a mobility scenario, this latency might be of less importance for initial connection to a Corresponding Node (CN) since the QoS initiator can probably accept a few

milliseconds delay before the session is established. Therefore, a minimal service interruption for ongoing connections caused by QoS and mobility re-registration for the new path during a Handover (HO) is certainly desired.

While this also indicates some further challenges, the main problem remains is the re-establishment of QoS on a dynamic path. The discussion of mobility, QoS, and the interaction of both and related aspects is split into the Sections 2.1.1, 2.1.2 ,and 2.1.3. By classifying the contrasting properties of different schemes, related pros and cons are identified.

2.1.1 Classification of Mobility

IP Mobility versus Support in Other Layers

If applications desire transparent mobility, this can potentially be provided anywhere below the application layer. For IP networks, this might be the transport layer (e.g. Transmission Control Protocol (TCP), User Datagram Protocol (UDP)), the network layer (IP itself) or even lower layers.

If a layer lower than IP is expected to be responsible for mobility provisioning, it must assure that the logical IP location of a host does not change. A first example would be an IP connectivity over a GSM modem line. In that case, the mobility support is provided by the GSM network, leading to rather high overhead and long delay. It also contradicts the IP philosophy of packet switched networking if packets are routed over a circuit switched channel which must be paid for even when no traffic occurs. A second example is the IP connectivity via different Access Point (AP)s located in the same logical link. This case has certainly its legitimation but is usually only suitable for local coverage. e.g. when providing wireless access in one building. Since the IP layer is devoted for packet delivery anyway and is also the lowest layer in the protocol stack before functionality is distributed to different link layer properties it is widely regarded as a reasonable candidate for transparent mobility support.

Nomadic versus True Mobility

The term “nomadic mobility” refers to the ancient habit of nomads who regularly moved and left behind whatever they build just to rebuild everything from scratch wherever they rest. Transferring this notion to network applications, it is of course always possible to close all running sessions, change the location and access point, acquire a new IP address and finally start a new session. This might somehow work out for applications like WWW browsing where the communication relies on a variety of independent short sessions. However, it blocks the possibility to be reachable via one well known IP and prevents the user from retaining sessions like voice calls or File Transfer Protocol (FTP) downloads while moving.

True mobility overcomes this constraint by hiding all effects of mobility from the application layer. In particular, it “prepends” a fixed network connection by ensuring transparent reachability. A typical example for true mobility provisioning (in a context

different to IP networking) is the current GSM network. In this architecture the client is always reachable via a static telephone number. Ideally, the client does not even recognize a handover while performing a voice call. The MIPv6 protocol provides the necessary extensions to enable IPv6 for true mobility support. However, with MIPv6 few efforts are undertaken to reduce the duration for re-establishing of connectivity and optimize handover operations. A micro-mobility solution is discussed in the next paragraph to partly resolve this problem. The IPv6 and MIPv6 protocol will be discussed in Section 2.2.

Micro-Mobility versus Macro-Mobility

This aspect takes care of the network topological distance of a movement. Every movement can be distinguished between a micro (local) and a macro (regional) movement. In general, a micro movement considers a movement within the same region, typically covered by one sub network, access network or administrative domain. The opposite, a macro movement, identifies a movement to a new region.

In a typical micro-mobility management, a movement is transparent to the network part beyond the visited region, and the need for related signaling is also limited to that region. The common approach to hide local mobility to the outside world is to involve a dedicated router in the visited region into the mobility management. Whenever an MN performs a macro movement and enters a new region, it communicates only the location of the dedicated router to the CNs. As long as the MN moves between different access routers but within the same region, the dedicated router is responsible to deliver packets to the MN's current location. Hence, only the dedicated router must be updated with the MN's current location. Since in most cases movements are local [9], an architecture that is enabled for micro-mobility can avoid a reasonable amount of mobility signaling to potentially far away CNs. Additionally, handover latency can be reduced because micro mobility management involves only the relatively close, dedicated router.

MIPv6 does not differentiate between micro and macro mobility. Thus, every movement is treated as a macro movement. The HMIPv6 protocol proposes a solution for micro mobility in MIPv6. Because HMIPv6 is yet only a developing protocol and its implementation design and implementation is part of this thesis, it will be described in the next chapter in Section 3.3.

2.1.2 Classification of QoS

The existing Internet Engineering Task Force (IETF) QoS architectures for IP layer can be classified into two types according to their fundamentals. The first type is known as Integrated Services (IntServ) [5] and provides together with Resource Reservation Protocol (RSVP) [6, 45] a solution for explicit reservations end-to-end. The alternative is referred to as Differentiated Services (DiffServ) and offers hop-by-hop differentiated treatment of packets.

The IntServ model aims to provide a kind of circuit-switched service in packet-switched networks. To describe the attributes of the desired data flow and signal the

specific resource requirements along the data path, IntServ uses RSVP. Briefly summarized, the IntServ/RSVP model provides unidirectional resource reservation on a per-flow basis. The sender of a flow first sends a "PATH" message to the receiver, which is updated at every router on the path. The receiver responds with a "RESV" message to indicate the required resources at every hop for the forthcoming flow. Any hop along the end-to-end path may deny the flow if resources are scarce. If the sender finally receives the "RESV" message, the resources have been successfully reserved.

The idea of DiffServ [3] is to map multiple flows with similar a service level into a few aggregate flows. Packets are classified as belonging to a specific aggregate flow by marking them with a DiffServ Code Point (DSCP). The DSCP is a well known value which determines the conditions to which a packet should be forwarded by the nodes inside the domain. In this model, explicit signaling to reserve resources is not necessary. This architecture can only differentiate in relative service levels. Theoretically, a minimum level of assurance for service classes can only be guaranteed if every router inside the domain is overprovisioned with resources so that an overload cannot happen.

In summary, compared with the IntServ model, differentiated service classes results in a more scalable service but also in service behavior only guaranteed at a higher granularity level. In addition to enhancing the level of guaranteed service through the differentiated service model, the network provider may increase the overall resource provisioning towards overprovisioning of resources. Both architectures have their advantages and disadvantages, especially when considering the environment for their deployment. For example, in backbones of commercial networks it might be much easier to guarantee a high probability of resource overprovisioning rather than requiring explicit reservations and all related signaling and soft states management. On the other hand, in wireless access networks bandwidth is always a scarce resource. Temporary hot spots like rush hours additionally cause a high variance in resource requirement and would make reliable overprovisioning even more expensive. As a consequence, enabling an architecture to support signaling for both technologies would be promising.

2.1.3 Interaction of Mobility and QoS Management

There are two schools of supporting QoS in mobile IP networks. One is to treat QoS mechanisms independently of mobility mechanisms, another is to couple QoS mechanisms with mobility mechanisms. The latter approach has an advantage over the former since it tends to cause a lower latency of re-establishment of QoS states but at the cost of higher protocol complexity. Since QoS resources in mobile networks, especially wireless access networks, are very precious, the amount of signaling traffic in the network should be kept as small as possible. If the utilization of QoS resources outweighs the additional complexity, imposed by coupling with QoS management of a mobility protocol, the coupling approach would be more desirable. The various possibilities, advantages, and disadvantages for simultaneous deployment of mobility and QoS support in IP based networks is discussed in [28]. Schemes and proposals regarding the inter-operation of RSVP and MIPv6 are given in [36, 37, 14]

2.2 Leveraged Architectures

2.2.1 IPv6 Protocol

Internet Protocol version 6 (also referred to as IP Next Generation) was designed as the successor to IPv4 [21]. This section briefly describes the important innovations of IPv6 over IPv4. Because IPv6 is a wide and complex field, my focus lies on its features used in this work and functionalities essential for understanding of the rest of this thesis.

The main need for a new version of IP arose with the increasing scarceness of address space in IPv4. IPv6 increases the IP address size from 32 bits (in IPv4) to 128 bits to support more levels of addressing hierarchy, a much larger number of addressable nodes, and simpler auto-configuration of addresses. On the other hand, some IPv4 header fields have been purged or made optional to reduce the common-case processing cost of packet handling and to limit bandwidth cost of the IPv6 header. Issues regarding header format and addressing capabilities are explained in Section 2.2.1.

Changes in the way IP header options are encoded allow for more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future. In fact, this property provides the fundamentals on which MIPv6, HMIPv6 and eventually the QoS-Conditionalized BU rely. The improved support for extensions and options is explained in Section 2.2.1.

Neighbor Discovery describes the process in IPv6 to discover nodes on the same link, determine each others' link-layer addresses, find routers and learn about further link properties. Neighbor Discovery (ND) together with stateless address discovery provides a possibility to auto-configure an interface that is switched on. This is explained in Section 2.2.1

Header and Addressing Capabilities

The IPv6 header [21] consists of two parts, the basic IPv6 header (as illustrated in Figure 2.1) and optional IPv6 extension headers, which will be examined in Section 2.2.1. The usage of the fields conveyed by the IPv6 header is summarized in Table 2.1. It should be noted that additional header fields besides the source and destination address in IPv6 were reduced to only 6 scattered over 64 bytes compared to the IPv4 header, which carries 10 fields in 96 bytes. In particular, the *Header Length* (the length of the IPv6 header is fixed to 320 bytes), the *Identification*, the *D- and M- flags*, the *Fragment Offset* and the *Header Checksum* field from IPv4 header were dropped or made optional in IPv6 extension headers. Even though the IPv6 addresses are four time longer than the IPv4 addresses, the header itself is only twice the size.

The format and semantics of IPv6 addresses are specified in [22]. There are three types of IPv6 addresses. These are unicast, anycast, and multicast addresses. An IPv6 addresses identifies one interface (or several interfaces in case of multicast addresses). A single interface may have several IPv6 addresses assigned to it. Unicast addresses (and their various sub-forms described in the following) and multicast addresses can be distinguished by the leading bits of the addresses. Anycast addresses are allocated out of the unicast address space and can semantically not be distinguished from them.

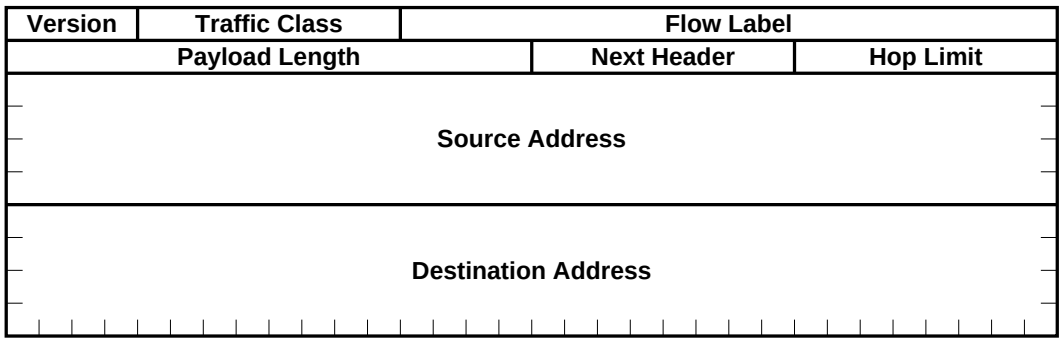


Figure 2.1: IPv6 header format

Unicast addresses identify a single interface and occupy the most of the so far assigned address space. A packet sent to an unicast address is delivered to the interface identified by that address. These are further split into several forms, where the relevant ones for this document are the provider-based unicast addresses, the site-local, and the link-local addresses. The **provider-based unicast addresses** are used for global communication and are similar in functionality to IPv4 addresses. Therefore, common routing in IPv6 is almost identical to IPv4 routing under Classless Inter Domain Routing (CIDR). **Link-local addresses** are designed to be used for addressing on a single link for purposes such as auto-address configuration, neighbor discovery, or when no routers are present. This will be discussed in more detail in Section 2.2.1. A packet carrying a link-local address as destination or source address is not forwarded by routers.

Site-local addresses are addresses that have only local routability scope and are guaranteed to be unique only on the specific site without the need for a global prefix. A packet carrying a site-local address as destination or source address is not forwarded by routers outside the site.

Anycast addresses identify a set of interfaces while a packet sent to an anycast address will be delivered only to the nearest one.

Multicast addresses identify a group of interfaces, such that a packet sent to a multicast address is delivered to all of the interfaces in the group. Since there are no broadcast addresses in IPv6, their functionality is superseded by multicast addresses. Therefore a node must join several multicast groups in order to recognize the corresponding addresses as identifying itself. For a host this is:

- The all-nodes multicast address
- The solicited-node multicast address for each of its assigned unicast and anycast addresses

A router is additionally required to recognize the following addresses as identifying itself:

- The subnet-router anycast address for the interface it is configured to act as a router on.

Name of Field	Meaning of Field
Version	4-bit Internet Protocol version number = 6.
Traffic Class	8-bit traffic class field.
Flow Label	20-bit flow label. Together with the source address a router can identify whether a packet belongs to certain flow and requests for special handling. This aspect of IPv6 is, at the time of writing, still experimental and subject to change as the requirements for flow support in the Internet become clearer.
Payload Length	16-bit unsigned integer payload length, in octets. Determines the size of the rest of the packet following this IPv6 header (note that any optional extension headers as described on 2.2.1 is considered part of the payload).
Next Header	8-bit selector. Identifies the type of header immediately following the IPv6 header. The same values as in the corresponding IPv4 Protocol field [32] plus some new values to identify specific extension headers (see 2.2.1) are used here.
Hop Limit	8-bit unsigned integer. Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.
Source Address	128-bit address of the originator of the packet. (This is not completely true for micro mobility but will be explained in the corresponding Section 3.3
Destination Address	128-bit address of the intended next recipient of the packet. (If a routing header is used (see: 2.2.1) the destination address might change several times on the way to the final destination)

Table 2.1: Meanings of IPv6 header field

- The all-routers multicast addresses

There are conventional notations for representing IPv6 addresses as text strings. The preferred form is x:x:x:x:x:x:x:x where the 'x's are the hexadecimal values of the eight 16-bit pieces of the address. It is not necessary to write the leading zeros in an individual field, but there must be at least one numeral in every field. Since many addresses contain long strings of zero bits, a special syntax is available to compress the zeros and make writing easier. The use of "::" indicates multiple groups of 16-bits of zeros. It can only appear once in an address and because IPv6 addresses are always 128 bit long the use of "::" always expands to one specific address. The text representation of IPv6 address prefixes is written in the notation IPv6-address / prefix-length where the prefix-length is a decimal value specifying how many of the leftmost contiguous bits of the address comprise the prefix. Table 2.2 gives some examples for valid IPv6 addresses and address prefixes in long and compressed form.

Long form	Compressed form	Purpose of the address
3ffe:b80:1111:0:0:0:0:0/64	3ffe:b80:1111::/64	A unicast link prefix
3ffe:b80:1111:0:0:0:0:1234/64	3ffe:b80:1111::/64	A unicast address combined with it's 64-bit link prefix
FF01:0:0:0:0:0:0:101	FF01::101	A multicast address
0:0:0:0:0:0:0:1	::1	The loopback address
0:0:0:0:0:0:0:0	::	The unspecified address

Table 2.2: Examples for valid IPv6 addresses in long and compressed form

Support for Extensions and Options

IPv6 options are placed in separate extension headers that are located between the IPv6 header and the transport-layer header in a packet. Most extension headers are not examined by any router along the packet's delivery path until arrival at its final destination. This reduces processing requirement and improves router performance for packets containing options. The basic outline of an extension header is depicted in Figure 2.2.

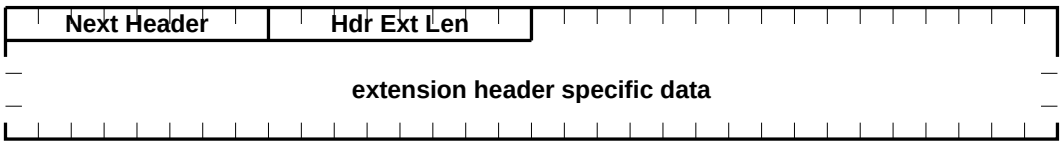


Figure 2.2: Basic IPv6 extension header format

Every extension header is identified by the Next Header field of the preceding header and its length (the fragmentation header is an exception) by the Hdr Ext Len field of its own header.

Currently, the following extension headers are obligatory for IPv6:

- A **Hop-by-Hop options header** conveys special options, which require processing by every passed router of the packet. If a Hop-by-Hop option header is present in a packet it must always be the first header behind the IPv6 header. (Note: Only if the next header field in the IPv6 header indicates that a Hop-by-Hop header is following, a router which is not the destination of a packet is required to examine the next header.)

- A **Routing header** allows extended routing (similar to IPv4 loose source routing). This is achieved by always storing the next (intermediate) destination as destination address in the IPv6 header. Once a packet arrives at the (intermediate) destination the router must check for a routing header. If one exists and more destinations to visit are present in the routing header, the destination address in the IPv6 header is modified to the next destination and the packet is forwarded accordingly. The extension headers following the routing header are only examined if the current router is the final destination of the packet.

- A **Destination options header** conveys special options that require processing only at the destination of the packet. Depending on whether the destination header appears before or behind the routing header, it is examined at every intermediate destination or only at the final destination.
- A **Fragment header** is responsible for fragmentation and reassembly.
- An **Authentication header** ensures integrity and authentication (security) for the packet.
- An **Encapsulation header** guarantees confidentiality (between source and destination) for the remainder of the packet.

Nevertheless, also another IPv6 header may appear in the extension header order of an IPv6 packet. This allows IP encapsulation or tunneling of packets. A node encountering another inner IPv6 header while processing the outer IPv6 headers processes this and potential further inner extension headers like an independent packet.

The encoding of one or several "opaque" options in either a Hop-by-Hop or Destination option header requires further conventions. The format of these so called Type-Length-Value (TLV) encoded options is depicted in Figure 2.3.

Figure 2.3: Type-Length-Value (TLV) encoded option format

The 8-bit *Option Type* identifies the type of the option where the three highest-order bits specify the action that must be taken if the processing IPv6 node does not recognize the Option Type and whether the Option Data might change on the way to the packet's final destination. The 8-bit *Opt Data Len* specifies the variable length of the *Option Data* field.

Functionality in the IP infrastructure is often distributed over different, potentially distant, entities. If the functionality is also controlled by external data, the involved entities in general require the exchange of some data in order to cooperate with each other. The exchange of such control data is also referred to as signaling. The concept of TLV-encoded options provides a powerful and flexible basis for signaling in the IP infrastructure. By placing options in an adequate option header and position in the extension header order, options can perfectly adapt to their functional purpose. Therefore, extension headers can be used to actively control the processing of a certain packet. On the other hand, by simply piggybacking signaling messages on packets which travel to the same destination anyway, overhead can be reduced.

Neighbor Discovery and Address Auto Configuration

The Neighbor Discovery [40] and Auto Configuration [42] protocols are summarized in one paragraph because they are strongly incorporated with each other. For example, the Address Auto Configuration uses Neighbor Discovery mechanisms to verify the

uniqueness of an assigned address. However, the basis protocol for ND and Address Auto-Configuration is the ICMP version 6 (ICMPv6).

ICMPv6 is actually used by IPv6 nodes not only to report errors encountered while packet processing, but also to perform other Internet-layer functions such as diagnostics and signaling. ICMPv6 is an integral part of IPv6 and must be fully implemented by every IPv6 node. The error messages defined in ICMPv6 specification [12] are concerned with packet delivery and parameter problems. The first group of problems address scenarios where the destination of a packet is unreachable, the packet is too big to be forwarded further, or the maximum hop limit of a packet has been reached. The second group handles problematic fields in the IPv6 header or extension header. In all cases, an appropriate error report is sent back to the source of the packet. By taking such error report into account the source can correct its behavior. An ICMPv6 header is identified by the Next Header value of the preceding header. However, since an ICMPv6 header does not have a Next Header field it is always the last message in a packet.

The general message format is illustrated in Figure 2.4. The message type is indicated by the *Type* field. Its value also determines the format of the remaining data. The *Code* field depends on the message type. It is used to create an additional level of message granularity. To detect data corruption in the ICMPv6 message and parts of the IPv6 header the *checksum* field is used. The *Message Body* finally conveys the message specific informations.

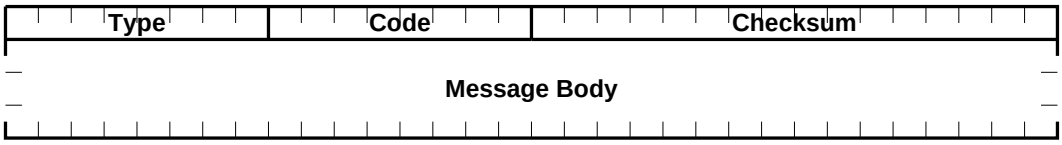


Figure 2.4: General ICMPv6 message format

The **Neighbor Discovery** (ND) protocol [40] defines mechanisms for interaction between nodes attached to the same link. It corresponds to the IPv4 protocols Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP) Router Discovery and ICMP Redirect [13, 31]. In IPv6 these mechanisms are based on five new ICMPv6 message types, which are discussed in the following. Routers use **Router Advertisement (RtAdv)** messages to advertise their presence together with various link and Internet parameters. This is done either periodically or in response to a Router Solicitation (RtSol) message. The message format is shown in Figure 2.5. The *Type*, *Code* and *Checksum* fields are part of the general ICMPv6 header as described above. The remainder of the packet conveys the RtAdv specific data. The meaning of the fields is summarized in Table 2.3

When an interface becomes enabled, hosts may send out a **RtSol** message, which requests routers to generate Router Advertisements immediately rather than at their next scheduled time. **Neighbor Solicitation (NbSol)** messages are sent by a node to determine the link-layer address of a neighbor or verify that a neighbor is still reachable via a cached link-layer address. NbSol are also used for Duplicate Address Detection (DAD). The response to a NbSol is a **Neighbor Advertisement (NbAdv)** message. It may also

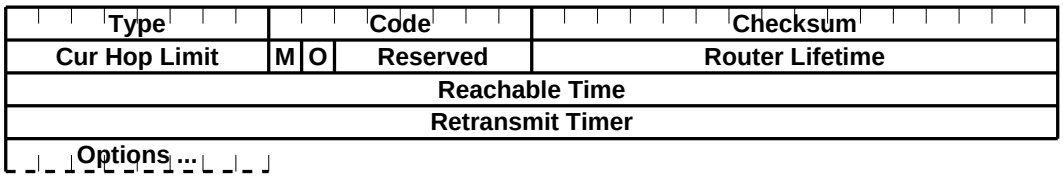


Figure 2.5: Router Advertisement message format

Name of Field	Meaning of Field
Cur Hop Limit	8-bit unsigned integer, specifying the hop limit of the RA.
M and O Flags	Must be set for Managed or other stateful address configuration, which is out of scope of this work
Router Lifetime	16-bit unsigned integer indicating the lifetime of the RA in seconds.
Reachable Time	32-bit unsigned integer. The time, in milliseconds, that a node assumes a neighbor is reachable after having received a reachability confirmation.
Retransmit Time	32-bit unsigned integer. The time in milliseconds, between retransmitted NS messages.
Options	The RA message may contain additional options. For address auto configuration the Prefix Information option is required in order to specify the prefix that is on-link. New options may be specified later. Indeed HMIPv6 introduces a new RA option for MAP discovery which will be discussed in Section 3.3

Table 2.3: Meaning of Router Advertisement fields

be sent out unsolicited by a node to announce a link-layer address change. A **Redirect** message is used by routers to inform a host of a better first hop for a destination.

Stateless address auto configuration [42] specifies the steps an host takes in acquiring a IPv6 address on a specific link. The procedure starts by creating a link-local address for the network interface. This is done whenever a network interface becomes enabled on a new link, by applying the IPv6 interface identifier [22] to the well known link-local prefix. The IPv6 interface identifier can be derived from the IEEE EUI-64 [23] address or the IEEE 802 addresses. Before link-local addresses are finally assigned to their respective network interface, they must be tested for uniqueness. This is done with the DAD procedure as part of the ND protocol explained later.

2.2.2 Mobility in IPv6

The IETF has been developing the MIPv6 protocol [25] for true mobility support in IPv6. Its stage is (at time of writing) still work in progress but is already very close to standardization. This section gives an overview of the protocol's basic operations, the related signaling messages and conceptual data structures.

A general system model is given in Figure 2.6. It shows typical subnetworks and access networks interconnected through the Internet. A subnetwork consists of one ore several links connected via a router, and one Edge Router (ER) for interconnection to the Internet. Each Access Router (AR) represents a router enabled for router discovery and stateless address auto-configuration as described in Section 2.2.1. The MN can move between its home link and different foreign links by being attached to any of the depicted Access Router. A general communication peer of the MN is represented as the Corresponding Node (CN). It may be located anywhere in the Internet.

By utilizing the new features of IPv6 in MIPv6, most of the mobility problems discussed in Section 2.1 are solved. The main issue of differentiating between identification and location of a Mobile Node (MN) was overcome by splitting the IP address into Home Address (HoA) and Care-of Address (CoA). Each Mobile Node (MN) is always identified by its HoA, regardless of its current point of attachment to the Internet. If an MN is located away from home (e.g., connected to another link as identified by its home link), it is also associated with a CoA to identify its currently visited location. Such a CoA can be acquired through ND and address auto-configuration whenever an MN becomes attached to the link served by a new AR (see Section 2.2.1) . Since an MN may have several valid CoAs, the one currently used for sending packets is referred to as its primary CoA. The MIPv6 protocol enables nodes to cache the binding of an MN's HoA with the MN's affiliated primary CoA. Any IPv6 packet addressed to the MN's HoA is then transparently routed to its CoA. To support this operation, MIPv6 defines one new entity, the Home Agent (HA), responsible for providing the link between the MN and yet (into MN's HoA – CoA relation) uninitiated CNs. It intercepts packets on the MN's home link destined for the MN and tunnels them to the MN's primary CoA. Additionally, several new requirements for MN and CN operations, four new IPv6 destination options, and some extensions to the Neighbor Discovery protocol are introduced. Because the scope of this thesis is limited to mobility, issues regarding authentication and security are completely neglected here.

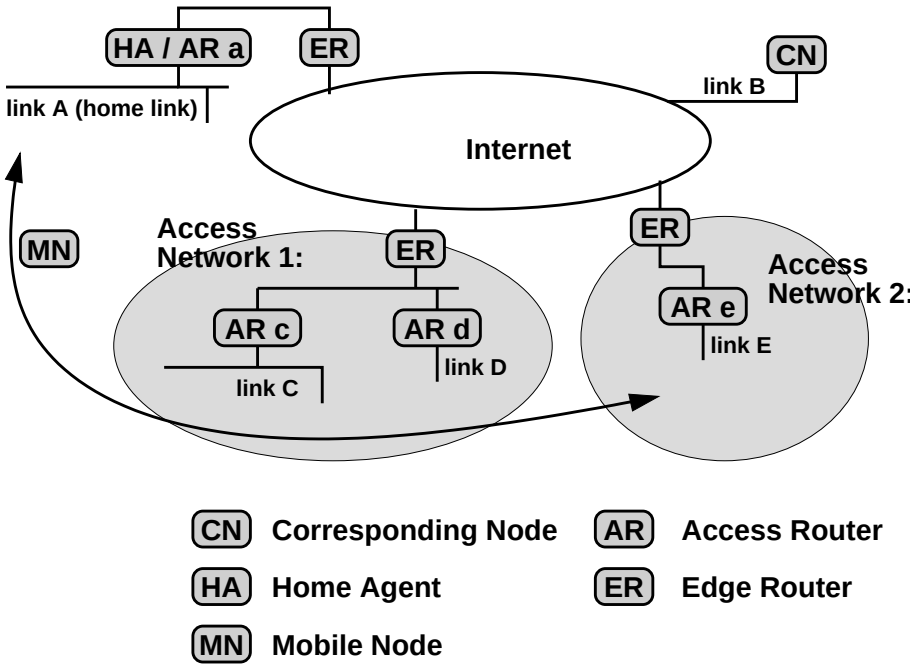


Figure 2.6: General MIPv6 system model

Conceptual Data Structures

The conceptual data structures defined by MIPv6 are the Binding Cache (BC) and the Binding Update List (BUL). The Binding Cache (BC) is a list that is maintained by every node for temporary bindings of other node's HoA to their currently used CoA. A lifetime value supplied with every entry indicates the remaining lifetime for this binding. When a node is sending a packet, it searches the BC list for the node's destination address (HoA) and if an entry for this destination exists the packet is routed via the corresponding CoA. Every MN also maintains a BUL to record information about active CNs whose BC must be updated before the lifetime of the particular binding expires.

Destination Options

The four new destination options added in MIPv6 are used for signaling mobility-management relevant information to the HA and CNs. They are called Home Address Option (HAO), Binding Update (BU), Binding Acknowledgement (BA) and Binding Request (BR). All of them are encoded in the TLV format as described in Section 2.2.1. Additionally, various sub-options exist for carrying optional information associated with a specific destination option and can be found in [25].

A *Home Address* destination option is used in a packet sent by an MN to inform the recipient of the packet of the MN's HoA. For packets sent by an MN while away from home, the MN uses its CoA as source address in the packet's IPv6 header. By including a HAO in the packet, the CN receiving the packet is able to substitute the MN's HoA

for this CoA when processing the packet, thus making the use of CoA transparent to the CN. Since the HAO is only relevant for the final destination of a packet it must be placed after the Routing Header but before the Fragment Header if any of those headers exist.

A *Binding Update (BU)* destination option is used together with a HAO option by an MN to notify a CN or the MN's Home Agent of its current binding. Therefore, the BU always refers to the CoA conveyed as source address in the IPv6 header and the HoA field in the HAO option. Some flags in the BU indicate whether:

- the destination of a BU is requested to act as the MN's HA,
- such requested HA should also perform DAD on behalf of the MN or
- a Binding Acknowledgement should be returned upon receipt of the BU.

A 32-bit lifetime field indicates the remaining lifetime of a BU before the binding must be considered expired. A value of zero indicates that the corresponding binding cache entry must be deleted immediately. An 8-bit sequence number field is used by the receiving node to sequence BUs and by the sending MN to match a returned Binding Acknowledgement with the corresponding BU.

A *Binding Acknowledgement (BA)* destination option is used to acknowledge receipt of a BU if this was required by the BU. A 32-bit Lifetime field indicates the granted interval in seconds for which the node will retain the binding in its binding cache. If the node sending this BA is also serving as the MN's HA, this value also indicates for how long this node will continue this service. A second 32-bit Refresh field indicates the recommended interval in seconds at which the MN should refresh the binding with a new BU. A BA is also used to inform the sender of a BU about the reasons why a specific BU was rejected. Therefore, an 8-bit Status field is included. Another 8-bit Sequence field is used to identify the corresponding BU to which this BA was sent in return.

Finally, a *Binding Request (BR)* destination option is used by a node to request an MN to send or update its binding in the nodes binding cache. An MN should reply to a BR with a BU and HAO destination option.

Basic Operations

The basic operations of MIPv6 will be described by a paradigm covering movement detection, HA registration, triangle routing and route optimization. It presumes that all nodes support MIPv6. A timeline for illustrating the message flow in the scenario is given in Figure 2.7.

The depicted timeline conforms to the system model in Figure 2.6. The first line identifies the involved nodes and their IPv6 address or Access Routers with the link prefix they are serving for. Whenever a router is required to process a traversing packet for which it is not the source or destination of the packet, this is indicated by an enlarged dot. Every circle with an index indicates a specific event. The grey vertical arrows at the left side of the figure indicate the lifetime of a specific Router Advertisement

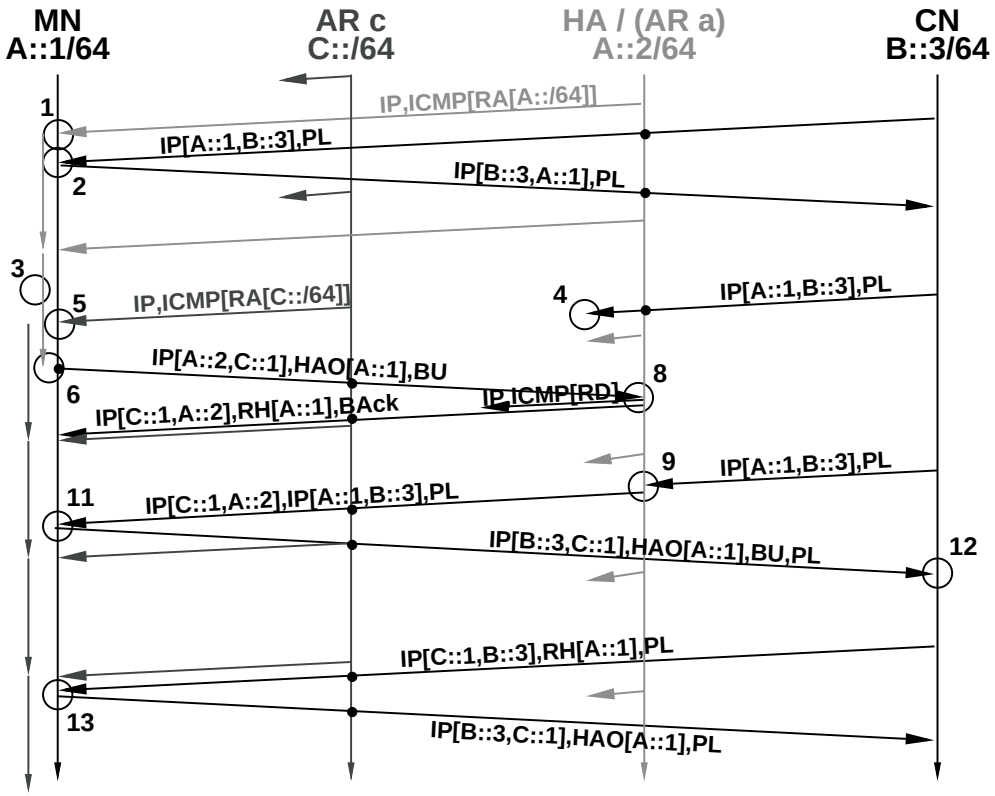


Figure 2.7: Timeline: MIPv6 signaling

(RtAdv). The content of every new message is roughly summarized by the text heading the corresponding message arrow.

The HoA of the MN can be formed by way of stateful or stateless address auto-configuration but can also be mannually configured. As long as the MN receives RtAdvs (event 1) which advertised prefix match the prefix of it own unicast address it can deduce to be still connected to its home link. While the MN resides on its home link, the CNs use its regular HoA for communication (event 2). For such ordinary packets a simple IPv6 header with destination and source address as identifying the actual payload is sufficient.

When the MN undergoes a movement (event 3) it depends on the movement detection capabilities to indicate this event to the IP layer. Otherwise (if movement detection only relies on IP layer), the MN can not recognize the loss of connectivity to the old AR until the lifetime of its last received RtAdv has expired. Packets destined for the MN HoA and routed via the old AR a will consequently get lost (event 4) because the last hop – AR a – has no possibility to forward them successfully. When the MN receives the next scheduled RtAdv from the AR b (event 5) it acquires a new IPv6 address as CoA. This is accomplished based on ND and address auto-configuration as described in Section 2.2.1. At this time it depends on the movement detection and handover policy whether an MN starts immediately with the handover operations or prefers to wait until the old default router finally becomes invalid. In some scenarios, e.g., areas with over-

lapping coverage by ARs, the latter case might be preferable, because the reception of a new RtAdv does not necessarily mean that the connection to the old AR was lost. If so called “eager handover processing” is performed, it might lead to unnecessarily frequent handover.

This scenario assumes “lazy handover processing”. No registration message is sent until expiration of the old default router (event 6). Then, AR C is selected as the new default router and the CoA, deduced from its RtAdv, becomes the MN’s primary CoA.

The next 3 events describe the procedures necessary for HA registration. In order to retain connectivity, at least the HA must be initiated with the binding of the MN’s HoA to the new primary CoA. Therefore, a Binding Update (BU) with the HA and BA flag is sent to the HA. Because the source address of the packet is now the MN’s CoA, also a HAO option must be present in the packet to identify the MN’s HoA. The reception of the MN’s BU (event 7) triggers the HA to modify its BC, intercept packets destined for the MN (routed to the home link of the MN) and tunnel them to the MN’s CoA. To ensure this requirement the HA performs proxy neighbor advertisement on behalf of the MN by advertising an ICMP Redirect (RD) message to the nodes sharing the same link. By receiving the ICMP RD message neighboring nodes recognize the link layer address change of the MN’s HoA to that of the HA. Finally, the HA acknowledges the successful registration by replying to the MN with a BA. Instead of sending the BA directly to the HoA of the MN, an Routing Header (RH) is used to route it via the MN’s CoA first.

From now on, a packet sent to the MN’s well known HoA is intercepted by the HA of the MN (event 8). The HA then tunnels the packet, by attaching an additional IP header before the original IP header, to the MN’s registered CoA. When the MN receives the packet (event 9), it can deduce from the encapsulating second IP header from its HA that the CN does not know about its current CoA. By either piggybacking a BU and HAO option to the next packet or sending them in an individual packet to the CN, the MN can avoid triangular routing and initialize route optimization. The CN now updates its BC and sends the following packets (event 10) with a routing header directly to the MN’s CoA. No BA is required since the MN can deduce from the use of a routing header instead of an encapsulating IP header (event 11) that the BU sent to CN was successfully registered. For further communication with the CN only a HAO option is required to identify the source of a packet. A new BU to the HA or CN is not necessary until the lifetime of the corresponding binding is about to expire.

Chapter 3

Description of QoCoo (QoS-Conditionalized Handover) Scheme

This chapter describes the general architecture and system behavior of the QoS-Conditionalized Handover (QoCoo) protocol. The goal of QoCoo is to provide a concept for efficient QoS signaling during handoff in Mobile IPv6 (MIPv6). This is achieved by enabling MIPv6 for micro-mobility support and allow a close interaction of micro-mobility and QoS management as discussed in Section 2.1.

For micro-mobility support, QoCoo is based on the architecture of HMIPv6 [38]. In HMIPv6, a new entity, the Mobile Anchor Point (MAP) is introduced that allows an MN to send only one Binding Update (BU) to the MAP when changing its AR within one Access Network (AN). For local movements, mobility can be completely hidden to all nodes outside the AN.

In order to allow a close interaction between mobility and QoS management, QoS messages are coupled with mobility messages as proposed in the QoS-Conditionalized BU [18]. It is accomplished by wrapping the QoS message as an IPv6 hop-by-hop option and piggybacking it with the BU and/or the BA for signaling purposes along every new path when performing a handover. This concept enables an MN to conditionalize a BU on the QoS requirements it expects from handover. The coupling with mobility signaling limits the additional latency for QoS provisioning to a level required for mobility management anyway[1]. Especially through adapting the QoS-Conditionalized BU to micro-mobility scenarios the handoff latency can be further reduced.

3.1 Assumptions and Limitations of QoCoo

QoCoo is a prototypical implementation with the purpose to prove the general feasibility of interaction between different technologies. In particular, QoCoo does not claim to provide a full implementation of any of the addressed underlying architectures. It merely

[1]Increased signaling latency only depends on additional processing in the affected router in order to check for QoS requirements but not due to new packets or additional transmissions.

focuses on the functionality necessary for realizing the core idea of the scheme. A list of the essential assumptions and thereby legitimate simplifications and limitations to the underlying architectures is:

- **Only essential basic mode functionalities supported from HMIPv6:**
  In HMIPv6, two different MAP modes are proposed, called "extended" and "basic" mode. The extended mode can be used to support mobile networks and other features with the disadvantage of introducing much more complexity. Since basic mode is sufficient for QoS support based on the QoS-Conditionalized BU only the latter is supported.

- **IPv6-routing header used to route BUs to MAP via new, tentative path:**
  Sending a packet via a new, tentative path which does not contain the MN's current default router is necessary, because a new path (access router) should not be established until the corresponding registration with the new MAP (reachable via this path) is acknowledged. Since the current Linux IPv6 stack does not support the "Strong ES model" (as suggested in [4] and providing a host with the capability to select the next hop of a packet with respect to its source address) an IPv6 routing header is used.

- **HA implementation used for provisioning of HMIPv6-MAP functionality:**
  According to the HMIPv6 specification, a MAP BU distinguishes from a HA BU only by the M flag in the registration message. However, regarding the actual binding operations of a MAP and a HA, the only difference in basic HMIPv6 mode is the context in which it is used by the MN as described in detail in Section 3.3. For the offered service from the MAP of HA point of view nothing is different. Therefore, an HA implementations is employed in the MAPs and also a common HA BU is used by the MN for registration with the MAP.

- **No overlapping of MAP domains:**
  The HMIPv6 architecture utilizes a MAP to offer micro-mobility to the MN. HMIPv6 conceptually also allows one AR to be served by several MAPs. Hence, overlapping of adjacent access networks can be achieved but may also introduce indeterministic routes to a CN outside the AN, which must be strictly avoided for transparent QoS provisioning. This aspect and a possible solutions is further discussed in Section 3.7. Currently, QoCoo enables an MN to register only with one MAP at a time and assumes a strict tree structure with the MAP, located in the Edge Router (ER), as the root and the ARs as the leafs of the AN.

- **QoS signaling supported only for micro-mobility:**
  QoCoo focuses only on micro-mobility scenarios. Therefore, QoS signaling is limited to the path within a visited AN up to the ER where the old and the new path converge. In the remainder, this is referred to as local QoS signaling.

- **Flow identification limited to flow label and RCoA:** According to the specification in [21] a flow is identified by its source address, destination address and

flow label. Because no end-to-end QoS signaling is supported in the QoCoo implementation, the CN's IPv6 address, as the MN's upstream destination address, is always initiated with zero. This indicates that any packet containing the specified flow label and MN's RCoA should be treated according the negotiated QoS guarantees for his flow.

- **Reduced QoS parameter set:**
  Various QoS parameters are proposed in [10] to represent a certain QoS level, such as jitter, delay, bandwidth, packet loss, etc. Without loss of generality, the QoCoo implementation only takes one important QoS parameter into account: bandwidth. Consequently, the QoS satisfaction is simply evaluated by the relationship among the available bandwidth along the affected path and the desired or minimal acceptable bandwidth requirements for the MN's flow.

3.2 Overview of System Model and Entities

Figure 3.1 illustrates the general system model of QoCoo. It shows basically the same model as illustrated in Figure 2.6 extended only with the necessary functionalities and entities to support micro mobility and the local QoS-Conditionalized BU. This includes a MAP located in the ER of every AN, the extension of every node in the access networks to be enabled for the QoS-Conditionalized BU and the functionality required in the MN to support micro mobility.

3.3 HMIPv6 – Informal Specification

HMIPv6 introduces one new entity, the MAP, which is responsible for assisting MNs while moving between ARs below its radio and hide local mobility of the MN from outside the AN. To accomplish this, an MN's CoA is split into Regional Care-of Address (RCoA) and Local Care-of Address (LCoA). The RCoA identifies the currently visited region[2] of an MN and the LCoA its exact location, similar to the MN's CoA in the MIPv6 architecture. Only the RCoA is communicated outside the region to HA and CNs. The MAP essentially serves as a local HA for the MN. Its purpose is to deliver packets, sent to the MN's RCoA, to the MN's current exact location as identified by its LCoA. In order to notify the MN of the availability of a MAP, the Neighbor Discovery protocol is extended for MAP Discovery. Whenever the MN moves to a new location inside the region, it requests local mobility support from the MAP by registering with its regional and local CoA.

In the following, the new signaling messages for MAP Discovery and MAP registration are explained. Then, an example scenario describes MAP Discovery, regional, and local registration procedures in detail.

The Sections 3.4.1 to 3.4.2 explicitly describe and specify the extensions required for MAP, intermediate routers and the MN to support HMIPv6 on top of MIPv6.

[2] in this model a region is equivalent to an AN.

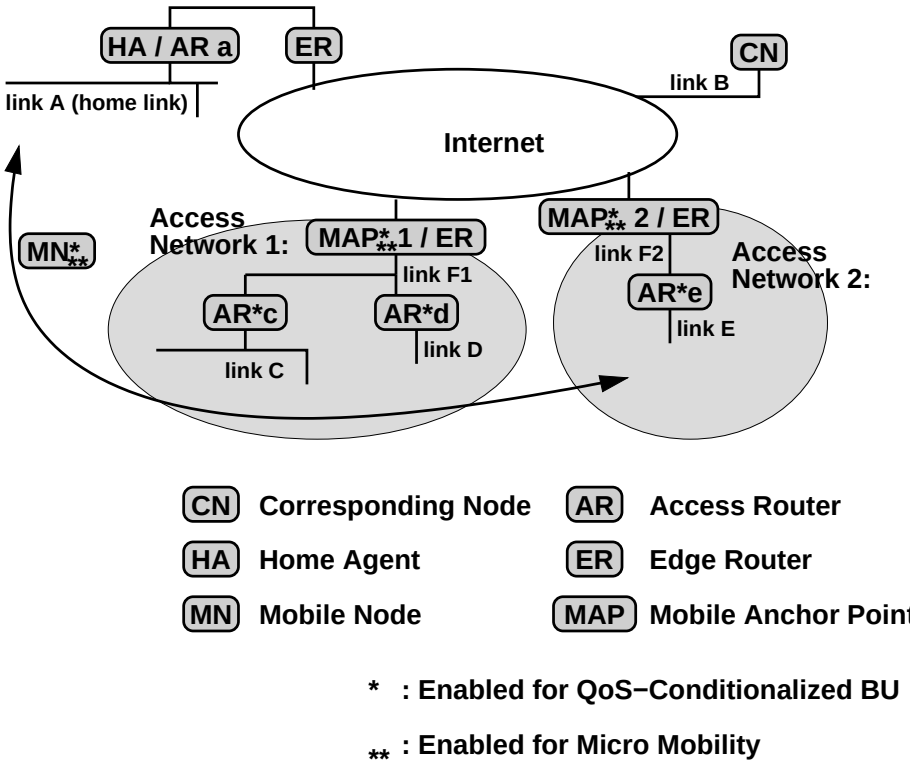


Figure 3.1: General QoCoo system model

New Signaling Messages for HMIPv6

HMIPv6 defines two new message options, one for Dynamic MAP Discovery and another for registration with a MAP. For MAP Discovery, an MAP Option (MAO) is conveyed as an additional suboption in the RtAdv message. A MAP (which always serves also as a router) uses the RtAdv messages to advertise its availability and additional parameters to neighboring nodes. Routers that reside on the same link receive the MAP option and propagate them with their own RtAdv down the AN towards the ARs and attached MNs. The format of a MAP option is illustrated in Figure 3.2. The meanings of the fields and assignments specific for the QoCoo implementation are given in Table 3.1.

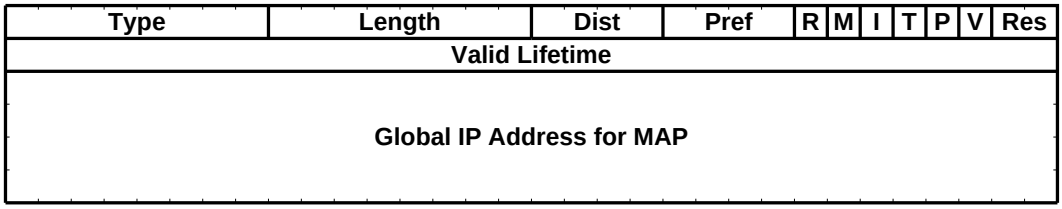


Figure 3.2: MAP option format

Name of Field	Meaning of Field
Type	8-bit unsigned integer to identify the MAP option. In the specification of HMIPv6 is is referred as TBA. In the QoCoo implementation it is set to a yet unused arbitrary value of ND_OPT_MAP_INFORMATION = 22.
Length	8-bit unsigned integer defining the length of the option in units of 8 octets and is set to 3.
Dist	A 4-bit unsigned integer indicating the distance or number of hops between receiver and originating MAP of the option. This value is mainly relevant for overlapping MAP domains, where a MN might select a particular MAP based on its distance. The evaluation of this value is not supported by QoCoo.
Pref	4-bit unsigned integer representing the preference of a MAP. A value of 15 means the lowest preference of a MAP. This value can be used to indicate an overloaded MAP or path. Default value is 3.
R flag	Indicates basic mode and that MN's RCoA must be formed based on 64 bit prefix of Global Address field. Since QoCoo only supports basic mode it is set to 1.
M flag	Indicates extended mode, set to 0.
I flag	When set indicates that the MN may use its RCoA as source address for outgoing packets. In order to keep flow identification transparent for local movement, the MN even must do so. Therefore set to 1.
T flag	Reserved feature for extended mode, set to 0.
P flag	Indicates that the MN must use its RCoA as source address for outgoing packets as discussed above. Set to 1.
V flag	When set indicates that reverse tunneling of outbound tunneling, to the MAP, is allowed, which is not supported and therefore set to 0.
Valid Lifetime	This value indicates the validity of the MAP's address and consequently the time for which an RCoA, based on this address prefix, is valid.
Global Address	One of the MAP's global addresses to be used by the MN for registration. The higher ordered 64 bit of the address provide the prefix for construction of the MN's RCoA

Table 3.1: Meaning of MAP option fields

For registration with the MAP while moving within its covered domain, a BU message with a HAO, as discussed in Section 2.2.2, is used. To distinguish the MAP BU from the HA BU, one of the original reserved flags is reassigned as the M flag to indicate a MAP registration. The only thing that makes the service from a MAP different from that of an HA is the context in which it is used by the MN. When sending a BU to the MAP, the MN stores its RCoA as the HoA in the HAO and the LCoA as source address in the IPv6 header[3]. The MAP interprets the content of the address fields in the same way as a HA does. Packets, destined for the address as indicated by the HAO field, are intercepted and tunneled to the CoA represented by the source address field of the IPv6 header.

HMIPv6 Signaling Paradigm

Figure 3.3 illustrates the procedure of MAP discovery and MAP-, HA- and CN- registration for a macro movement and micro movement. The timeline corresponds to the system model in Figure 3.1. The relevant events are marked with an index which is used for reference in the following. The syntax and tokens used to describe the behavior and content of a message are the same as in Figure 2.7. To simplify, some interim procedures like DAD or link layer address change are neglected in this example.

The first 3 events are part of the MAP discovery process. It overlaps with the actual MAP registration procedure covered by event 3 to 5. Every ER which is also serving as a MAP advertises its availability by attaching a MAP option to its RtAdv messages (event 1). This RtAdv is received by all other nodes sharing link F1, in particular AR c and d (event 2). Each of them now maintains a record of the MAP option for further propagation. The lifetime of the received MAP option is indicated by the vertical arrows on the right side of the AR.

The reception of the RtAdv message (event 3) triggers the MN to initiate the ND process. In particular establish a new CoA and select AR c as its default router. The supplied MAP option additional indicates the availability of a MAP, willing to assist the MN for micro-mobility. By combining the 64-bit MAP address prefix with its own 64-bit interface identifier, the MN first constructs a tentative RCoA. To verify the RCoA and request for the offered service, a BU message is sent to the MAP. The MAP then checks for uniqueness of the requested RCoA, arranges the necessary entry in its Binding Cache (BC) and acknowledges the successful operation with a BA to the MN (event 4).

The following 3 events illustrate the additional mobility signaling that is only necessary for a macro movement. The now verified RCoA (event 5) is signaled to the HA and CN[4] with an appropriate BU message, revealing only the new RCoA as the MN's current primary CoA. The corresponding BA from the HA (event 6) is consequently replied to the MN's RCoA on link F1. There it is intercepted by the MAP (event 7) and tunneled to the MN's LCoA. Common packets sent by the CN (event 8) undergo the same treatment. Because the re-routing via the MAP (event 9) is the desired effect

[3]It should be noted that the case of a MAP registration is the only exception to the general requirement of always using the RCoA as source address for outgoing packets.

[4]This scenario assumes that the CN is already maintained in the Binding Update List (BUL) of the MN and route optimization is initiated immediately.

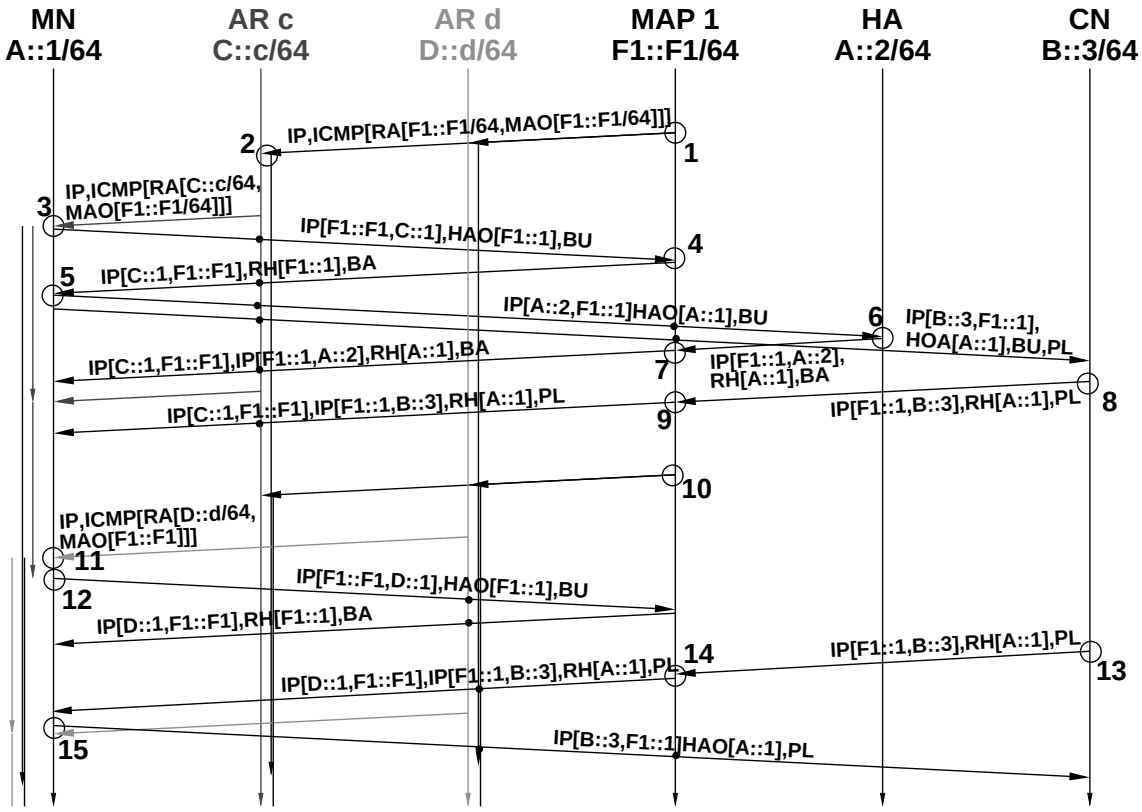


Figure 3.3: Timeline: HMIPv6 signaling

for micro-mobility provisioning in HMIPv6, route optimization is only performed to avoid triangle routing via the HA but suppressed for the MAP. Event 10 simply indicates the regularly requirement for a MAP to refresh its availability by re-initiating the MAP discovery process.

The gain for a local movement is illustrated by the next two events. In event 11, the MN receives the RtAdv from AR d informing it with the availability of a new default router but covered by the old MAP. When the lifetime of the old AR finally expires (event 12), it must only update the binding entry in the MAP with the new LCoA acquired via AR d. For HA and CN the RCoA has not changed. Further packets by the CN (event 13) are still routed to the MN's HoA via its RCoA, where they are intercepted (event 14) and tunneled to the new LCoA.

3.4 HMIPv6 – Formal Description of Implementation Model

Approaching a concrete implementation design, this section represents a more formal description of the required functionality and expected behavior of the HMIPv6 part of QoCoo. Actually, the implementation modeling went in parallel with the elaboration of

the selected implementation environment. This was necessary to get a feeling for what is feasible at all and adapt my implementation model to the concept and potential entries of the surrounding implementation. For a straightforward development from concept to praxis, the formal description of the implementation model is discussed before the selection and elaboration of the implementation environment (Chapter 4).

The description of the implementation model is done by means of Specification and Description Language (SDL) inspired notation. Such pseudo-SDL slang simply offers a suitable method to transform the underlying informal specifications towards a model that is reasonable as a basis for an implementation design. I believe that violations of a formally correct description in SDL are justifiable since the purpose of this model is to provide an intermediate step between the conceptual specification and the final implementation design. The aim is **not** to prove the correctness of the underlying protocols or this implementation design.

In the following sections the distinct entities and their behavior regarding the support of the HMIPv6 protocol is described.

3.4.1 Mobile Node

The most new requirements are necessary for the MN and are identified here. The new behavior of a MIPv6 MN to be enabled for HMIPv6 is illustrated in Figure 3.4 a-c. To save space and provide a better overview of the described behavior certain conditions and actions are specified in individual tables and are only referenced in the Figures 3.4 a-c. The conditions for determining the required actions according to the current context are given in Table 3.3. Table 3.2 lists related registration messages and the carried IPv6 addresses that must be present in the IPv6 source field and Home Address Option (HAO). The corresponding line of the table and the message's destination is indicated by the name of the message[5]. The related column is indicated by the argument of the message.

Table 3.4 provides an additional context summary of every leaf branch that leads to a specific action.

BU-DST	HOME mode (H)		ANCHORED mode (A)		MIP mode (M)	
	SRC	HAO	SRC	HAO	SRC	HAO
BUxMAP			x!MAP!LCoA	x!MAP!RCoA		
BU_HA	HoA	HoA	x!MAP!RCoA	HoA	CoA	HoA
BU_CN	HoA	HoA	x!MAP!RCoA	HoA	CoA	HoA
BUpMAP	pRCoA	pRCoA	pRCoA	pRCoA	pRCoA	pRCoA

Table 3.2: Composition of MN registration messages for Figure 3.4.

[5]BUxMAP represents a BU to be sent to the MAP identified through x, BUpMAP represents a BU to be sent to the previously used MAP, and BU_HA and BU_CN represent BUs to be sent to HA and CNs

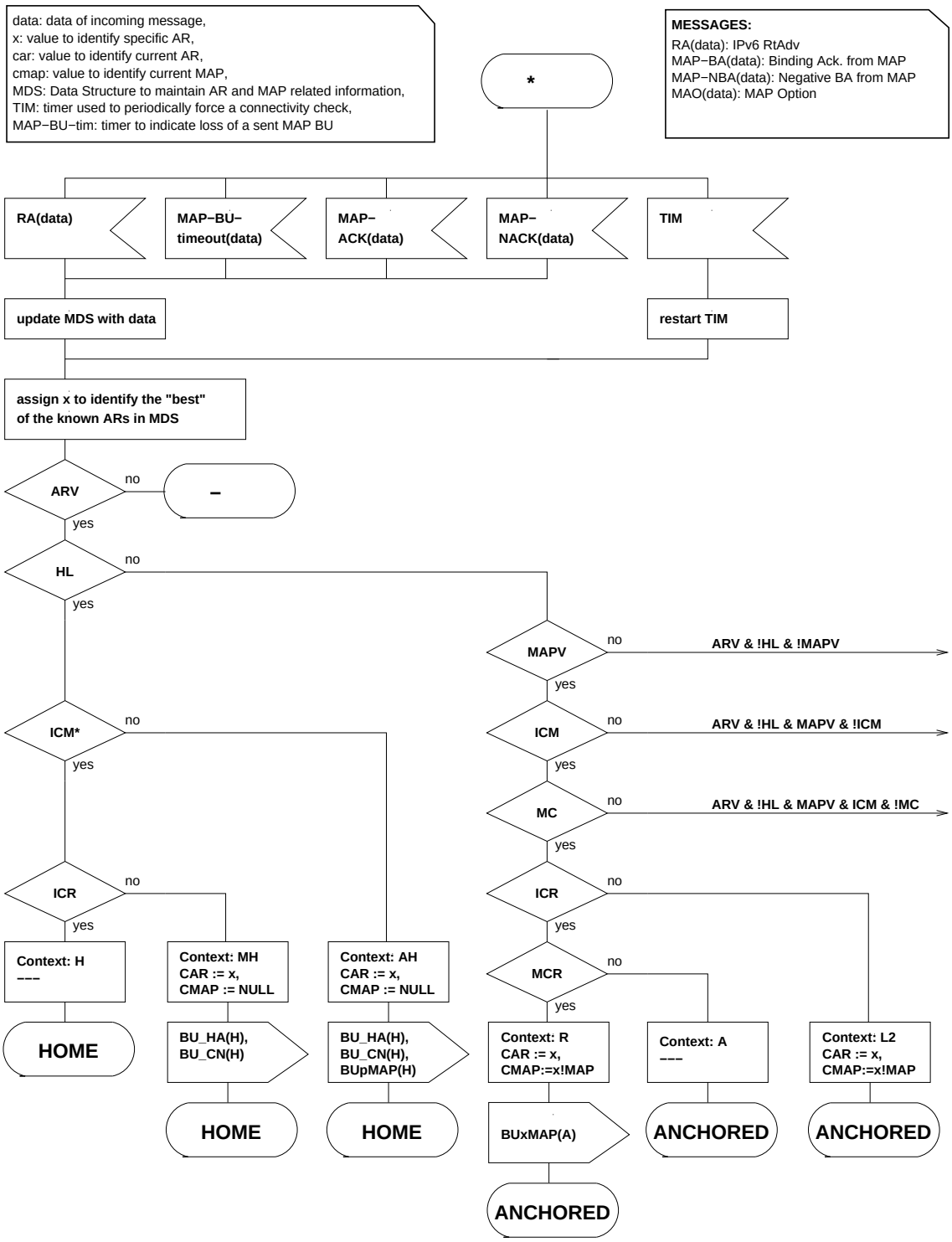


Figure 3.4: Pseudo-SDL description of extensions required for HMIPv6 support in the MN (1/3)

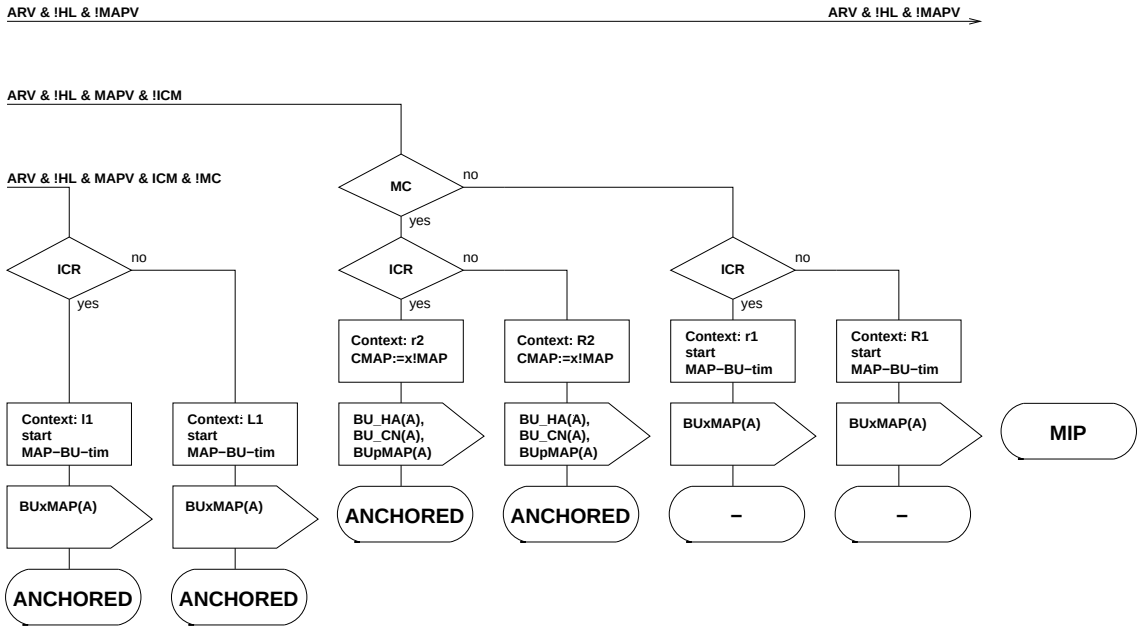


Figure 3.5: Continued pseudo-SDL description of extensions required for HMIPv6 support in the MN (2/3)

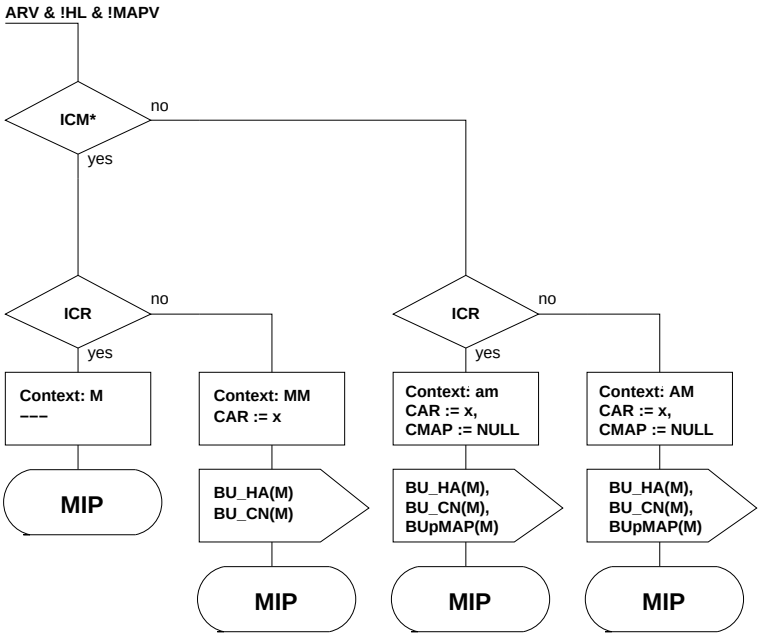


Figure 3.6: Continued pseudo-SDL description of extensions required for HMIPv6 support in the MN (3/3)

Abbreviation of condition	Description of condition
ARV (AR valid)	AR X is valid as indicated by its lifetime.
HL (Home link)	AR X is located in home link as indicated by the advertised prefix.
MAPV (MAP valid)	AR X is covered by a valid MAP as indicated by the advertised MAP option and lifetime.
ICM (Is current MAP)	The advertised MAP option of AR X belongs to the current MAP (CMAP). This is indicated by the conveyed MAP address in the MAP option.
ICM* (Not ANCHORED)	The state of the MN is **not** ANCHORED (not using a Mobile Anchor Point, not in HMIPv6 mode). It is in HOME or MIP mode.
MC (MAP confirmed)	A recently sent BU to the MAP has already been confirmed by a corresponding BA.
ICR (Is current router)	AR X is identical to the current default router.
MCR (MAP needs refresh)	MAP is the current MAP and registration with the MAP has already been acknowledged but lifetime of the registration is about to expire and needs to be refreshed.

Table 3.3: Meaning of abbreviations for the conditions in Figure 3.4 regarding the selected AR X.

Abbrev.	Description of Context
H	MN has **not** moved and is still in HOME environment.
MH	MN has moved from MIP environment to HOME environment.
AH	MN has moved from ANCHORED environment to HOME environment.
R	MN is in ANCHORED environment and has not moved but MAP binding lifetime is about to expire and requires to be refreshed.
A	MN has **not** moved and is still in ANCHORED environment.
L2	MN has undergone a local movement in ANCHORED environment and the new LCoA registration with the MAP has already been confirmed.
l1	MN is in ANCHORED environment and has not moved but MAP binding is currently not confirmed or has expired (Note: MAP registration must have expired without being refreshed in time.)
L1	MN has undergone local movement in ANCHORED environment.
r2	MN has **not** moved but the availability of a new MAP has been advertised by the current AR, resulting in an new ANCHORED environment for the MN. A registration with that MAP has already been confirmed by a BA. (Note: Advertised MAP option must have changed with this AR.)
R2	MN has undergone a regional movement to a new ANCHORED environment. LCoA registration with the MAP has already been confirmed by a BA.
r1	MN has **not** moved but the availability of a new MAP has been advertised by the current AR, resulting in an ANCHORED environment for the MN. However, the MN keeps assuming the previous environment until registration with the MAP is confirmed. (Note: Advertised MAP option must have changed with this AR.)
R1	MN has undergone a regional movement to a new ANCHORED environment. The MN keeps assuming the previous environment until registration with the MAP is confirmed.
M	MN has **not** moved and is still in MIP environment.
MM	MN has moved from MIP to MIP environment.
am	MN has **not** moved but the recently availability of a MAP is no more advertised via the AR, resulting from an ANCHORED to a MIP environment. (Note: Advertisement of MAP option has ceased with this AR.)
AM	MN has moved from ANCHORED to MIP environment.

Table 3.4: Meaning of abbreviations in Figure 3.4 for current context of the MN regarding the selected AR X.

0

3.4.2 Mobile Anchor Point (MAP)

This section explicitly describes the operations required from a MAP. It is divided into MAP discovery and the binding operation.

MAP Discovery

A router serving as MAP is responsible for initiating the MAP discovery process. Therefore, the MAP must be configurable to advertise its own MAP options and propagate MAP options from potential, topological higherMAPs along with router advertisements downstream the AN to the MN. For MAP discovery, every interface must be configurable for the following properties:

- To receive or propagate MAP options received from other interfaces. To avoid loops only one should be the case.
- Whether or not its own MAP options shall be attached to an already received MAP options.
- If own MAP options are configured to be advertised, the parameters carried by the MAP option as described in the previous section.

Figure 3.7 describes the behavior for MAP discovery on process level.

MAP Binding Operations

Because the main difference between a MAP and an HA is only the context in which it is used by the MN, an HA existing implementation can be used. One advantage of this approach is, that no new functionality for the binding operations is required.

3.4.3 Intermediate Routers

Intermediate routers only have to receive and propagate MAP options in router advertisements. They perform essentially the same behavior for MAP discovery as the MAP itself, except that they don't emit their own MAP options. They only propagate received MAP options with router advertisements.

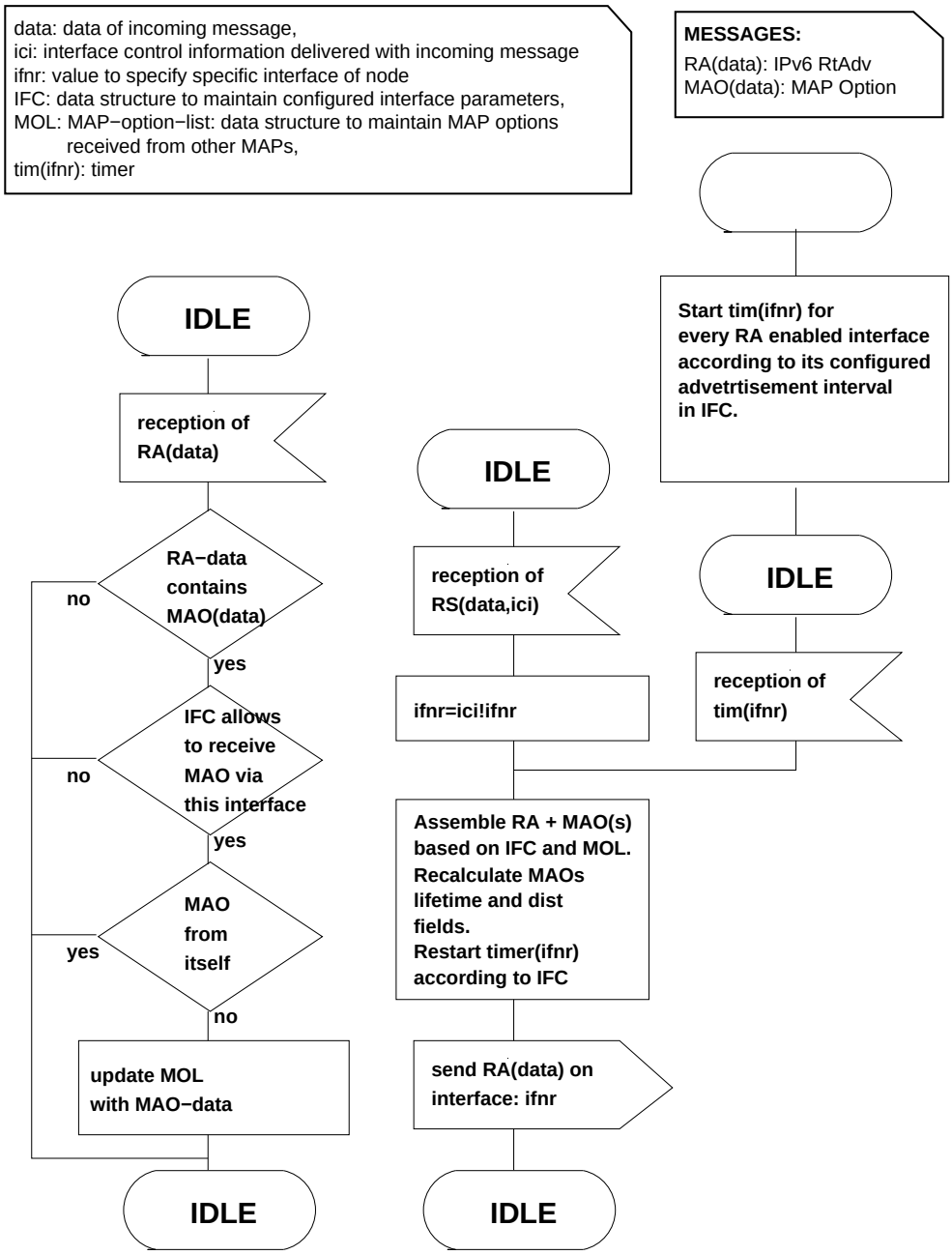


Figure 3.7: Pseudo-SDL description of extensions required for MAP discovery in MAP and IR

3.5 QoS-Conditionalized BU – Informal Specification

In this scheme an IPv6 QoS hop-by-hop option (see Section 2.2.1), to represent the MN's QoS requests, is piggybacked on the message containing the BU to the MAP. This message is called BU+QoS message. Each node concerned with QoS management between the MN and the MAP conceptually passes the QoS request to internal QoS mechanisms in order to check for resource availability. If desired resources are available, they are reserved and the message will be forwarded along its route. Further, the QoS message allows the distinction into desired and minimum acceptable QoS requirements of the MN to the new data path. If an IR decides that the MN's desired QoS request can not be guaranteed but a level covering the specified acceptable amount, the QoS option is updated respectively. If not even the acceptable QoS request can be guaranteed, no reservations are made and the QoS option is modified to indicate this fact to the other involved nodes on the path and to the MN.

Upon reception of the BU+QoS message, finally the MAP checks on the requested resource availability. In case of success the binding status of the MN is updated and the MAP responds with a positive BA+QoS message, representing the actual amount of reserved resources. Otherwise, no binding update is performed and a negative BA+QoS message is returned to the MN. While the BA+QoS message travels back to the MN intermediate routers check whether higher nodes have decreased the desired QoS level or even rejected the MN's QoS request completely. Then also the amount of previously reserved resources is corrected. When the MN receives the BA+QoS message it can determine whether the MAP registration was a success and how much resources have been reserved.

By way of this scheme, QoS (re-)establishment for micro movements is managed locally and transparently to the CNs. In case of a macro movement QoS re-establishment might be managed with MIPv6 and [11], however, this is not supported by the QoCoo implementation. Only if all routers along the new path find that sufficient resources are available a handoff is performed. In this sense, the handoff process to a new AR is conditional on the availability of QoS resources and the scheme can take advantage of HMIPv6. If there are multiple AR to choose from another one can be tried in succession. Alternatively, the MN may decide to reduce the minimum acceptable QoS level of its QoS request to the path and send a new BU+QoS message via the selected AR to the MAP.

New Signaling Messages for QoS-Conditionalized BU

The specification of the QoS option used in the QoS-Conditionalized BU [17] is essentially taken from a proposal by H. Chaskar and R. Koodli called "Framework for QoS Support in Mobile IPv6" [10]. In the QoCoo implementation some reorganizations were made to the compositions of a QoS option. Figure 3.8 illustrates the differences between the proposal in [17] and the composition used in QoCoo.

The reasons for the modifications are explained in the following text. The QoS options is wrapped as a TLV encoded option in an IPv6 hop-by-hop extension header where one QoS option may contain one or several QoS objects. Each QoS object is

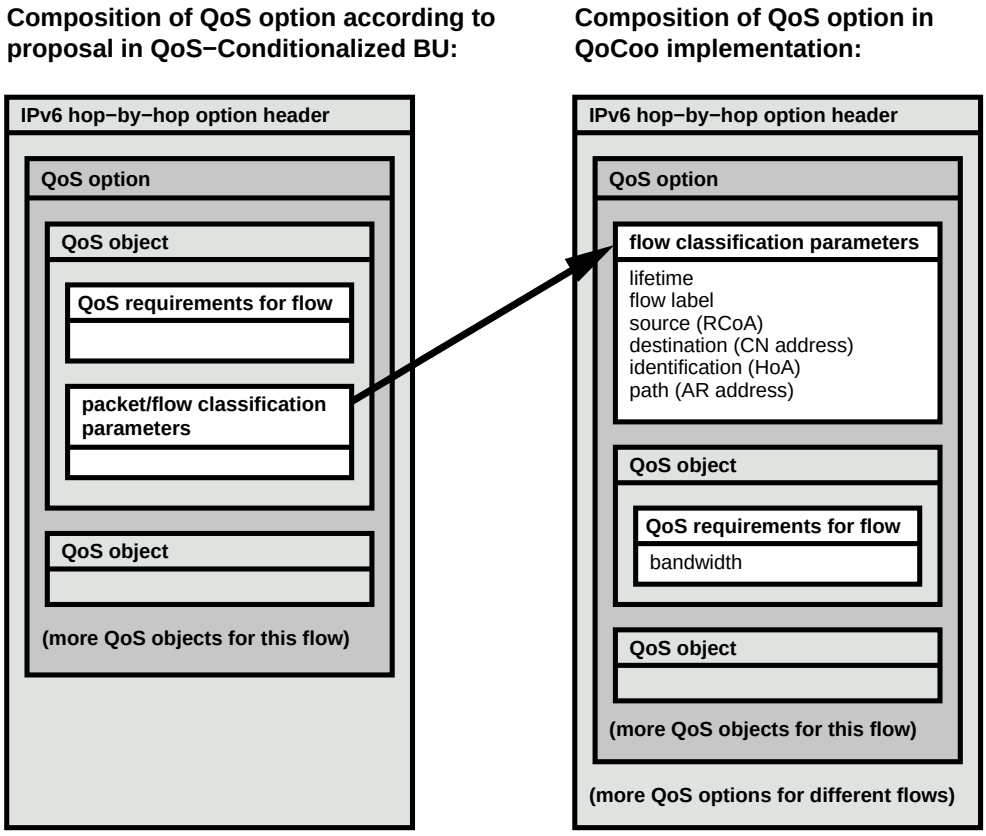


Figure 3.8: Composition of QoS options in QoS-Conditionalized BU and QoCoo

capable of holding QoS descriptions and packet-classification parameters (flow identifications) for one specific data flow of the MN. According to the definition for the use of flow labels to label sequences of packets [21], it is not allowed to use the same flow label value for up and downstream packets between two communicating nodes. The QoCoo implementation does not conform to this rule, but since the IPv6 specification [21] also states, that the use of the flow label field is still experimental and subject to change, I believe this violation to be reasonable. On the other hand, specifying the QoS requirements for the up and downstream part of a flow separately is still possible by using one QoS option for each direction.

The D- and A-flag introduced in [17] indicate which of the possible four combinations for a flow (a – upstream and acceptable, b – downstream and acceptable, c – upstream and desired, d – downstream and desired) is contained in the QoS option. Thereby the QoS objects which are present in one QoS option can only describe QoS requirements for flows that belong to one single CN or one group of CNs. Whether several QoS options may exist within one hop-by-hop extension header to express QoS requirements for flows to different (groups of) CNs is not stated explicitly in [17]. However, the "values of packet classification parameters" field, carrying the necessary information to let routers identify packets that need special handling, is present in every QoS object. This is even the case when the QoS objects belong to the same flow and only express

the difference for its acceptable and its desired QoS requirements. Another potential problem with the proposed QoS object composition in [10, 17] is its resulting length. The maximum option data length of a TLV encoded option is limited by the 8-bit option length field (see Section 2.2.1) that can represent the option data length only up to 255 octets (bytes). If the "Values of Packet Classification Parameters" field contains two (16-byte) IPv6 address, a (4-byte) lifetime value, and a (3-byte) flow label the total size of one QoS object would result in 66 bytes length and limits the amount of QoS objects to only three per option.

Therefore, the multiple times of presence for the same packet classification parameters result in unnecessary overhead, especially when one or several (16 byte long) IPv6 addresses are included to identify the source and/or destination of a flow. To overcome this weakness, the packet classification parameters are moved to the QoS option header, providing flow identification parameters for all QoS objects in this option. More QoS options may follow the first option in the hop-by-hop extension header if QoS reservations for different flows are desired.

Figure 3.9 illustrates the detailed format of the QoS option used in the QoCoo implementation. Table 3.5 and 3.6 describes the parameters carried in the "packet classification parameters" field and other fields of the QoCoo-QoS option.

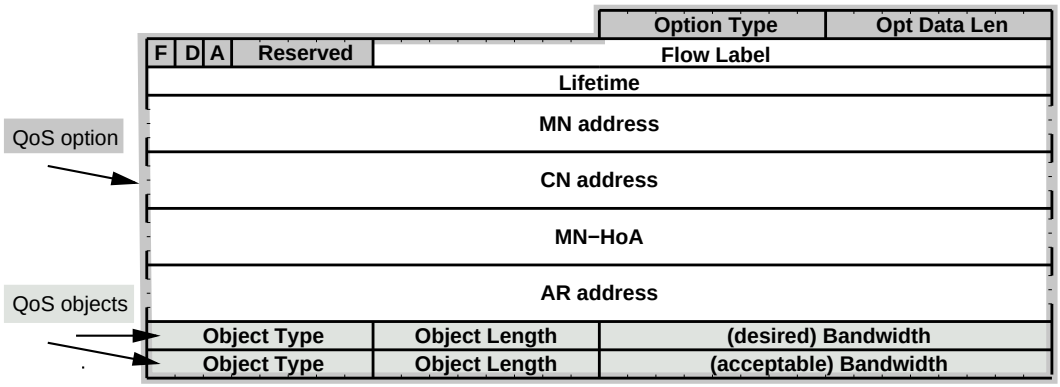


Figure 3.9: QoCoo-QoS option format

QoS-Conditionalized BU Signaling Paradigm

The example shown in Figure 3.10 illustrates the procedure of a QoS-conditionalized handover within a micro movement.

For the beginning of the timeline it is assumed that the MN performs a QoS sensitive communication with the CN B::3 (see Figure 3.1) . QoS related resources have already been reserved in all involved routers on the data path between MN and CN, particularly in AR c, the MAP, and further nodes between the visited AN and the CN. In event 1 the MN receives the first RA from AR d which is covered by the same MAP as the currently-in-use AR c. When the lifetime of the last received RA from AR c exceeds (event 2), a BU+QoS message is sent via AR d to MAP 1. In this example, the QoS

Name of field	Meaning of field
Option Type	Identifies the hop-by-hop option type. The highest three bits must be 001 to state that a processing IPv6 node, which does not know the option type must skip over this option and continues processing the packet header and that the option data may change on en-route. The lower five bits are set to 11001
Opt Data Len	Length of the option data field in octets. Length of this option is: 80.
F flag	If the "F" bit is set, this means "QoS can not be met", otherwise "(up to current node) QoS can be met".
D flag	If set indicates that “both upstream and downstream QoS requirements are specified separately, otherwise QoS specifications express requirements for up and downstream. (In this example not set.)
A flag	If set indicates that “both desired QoS and acceptable QoS are specified”, otherwise only desired is specified. (In this example set to 1.)
Flow Label	24-bit value to identify packets that must be treated according the expressed QoS requirements.
Lifetime	Indicates the lifetime for which QoS reservations for this flow are valid. If needed for a longer period, the reservation must be refreshed by a new QoS message before the lifetime elapses, if requirement ends prematurely a QoS message for this flow with the F bit set must be send.
MN address	Indicates the upstream source of the flow. This value could theoretically be inferred from the HAO of the MAP BU that also holds the MN’s RCoA. However, by this way the possibility is given to send QoS options independent of a MAP BU. This might be beneficial when a MN wants to increase or decrease its QoS requirements for an already established flow. Also, since the HAO is an destination option it is normally not supposed to be processed by an intermediate router.
CN address	Indicates the downstream source of the flow.
MN HoA	Represents an global unique identification of the MN. This is actually only relevant for charging.
AR address	Indicates the used path of the MN to the MAP. This becomes important when the MAP is requested to release reserved resources on behalf of the MN when the MN looses connectivity to the AR of the path. Otherwise reserved resources could only timeout and would lead to “expensive” unused reservations whenever a MN moves.

Table 3.5: Meaning of QoCoo QoS option fields (1/2)

Name of field	Meaning of field
Object Type	Identifies the QoS object type that specifies the QoS requirements. This one only specifies the amount of bandwidth. If multiple objects of this type exist, each of them express the different requirements for up- and/or down-stream combined with desired or acceptable bandwidth.
Object Length	Length of this object in octets. (Set to 4 in this example)
Bandwidth	16 bit unsigned integer indicating the maximum bandwidth requirements for this flow in kilobit/second (Kb/s).

Table 3.6: Meaning of QoCoo QoS option fields (2/2)

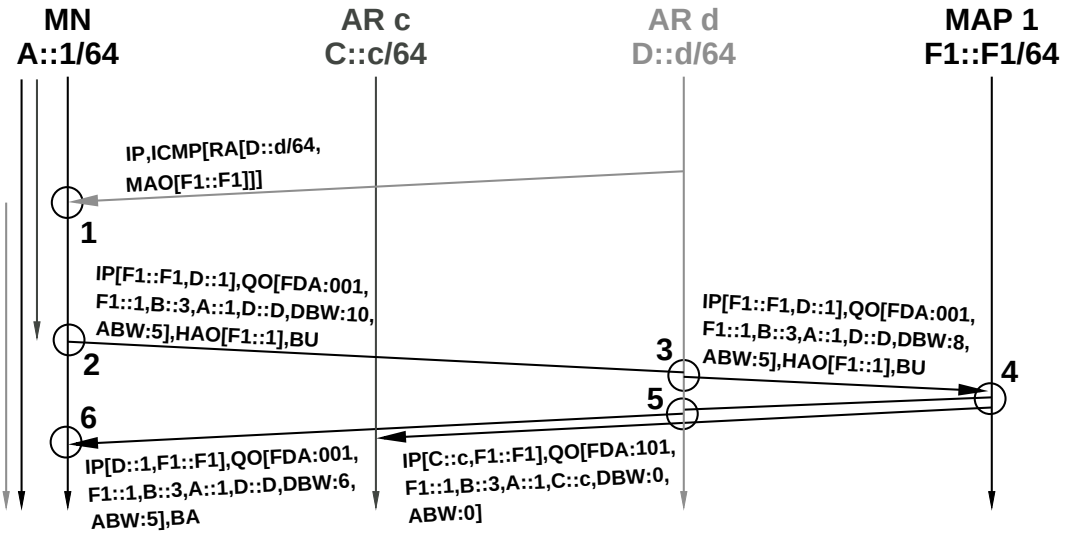


Figure 3.10: Timeline: QoS signaling

option (QO) requests all routers on the new data path in the AN to reserve 10 Kb/s bandwidth for upstream traffic and 10 Kb/s bandwidth for downstream traffic to the CN B::3. However, if the requested 10 Kb/s could not be guaranteed by any of the affected routers, the MN would also accept a smaller bandwidth as long as a minimum of 5 Kb/s can be guaranteed.

When the BU+QoS message passes AR d (event 3), the router checks its available resources. The check results in only 8 Kb/s instead of the desired amount of 10. Because the amount of available bandwidth even though fits into the acceptable requirement of the MN, the QO is modified accordingly and 8 Kb/s of bandwidth are reserved for the MN's flow. Then the packet is forwarded towards its destination the MAP. In this example the MAP reduces the desired bandwidth of the MN's QO another time since it can only guarantee the amount of 6 Kb/s, which is still within the MN's acceptable range (event 4). The MAP updates the MN's binding status, reserves the accepted amount of bandwidth, and replies to the MN with the corresponding BA+QoS message to acknowledge the partly success of the operation. Additionally, a hop-by-hop-QoS message, with the F bit set, is sent to AR c to trigger the release of reserved resources on the previously used data path of the MN.

When the QoS option, now piggybacked by a BA, passes the AR d a second time (event 5), it checks the option's F flag and compares the contained DBW value with the amount of previously reserved resources for this flow. In this example, 2 Kb/s of the reserved bandwidth is released and the message is forwarded to the MN's LCoA. By receiving the BA+QoS message, the MN can recognize whether the requested reservation was a success (F flag = 0) and if yes, how much of the desired QoS could be satisfied by the new path.

3.6 QoS-Conditionalized BU – Formal Description of Implementation Model

This section provides a more formal description of the behavior for the nodes involved in the QoS-Conditionalized BU procedure. The description is conducted in the same way as for HMIPv6 requirements in Section 3.4. An SDL inspired notation is used with the aim to provide an intermediate step between the informal specification of the underlying protocol and the implementation design. In the following sections the distinct entities and their behavior regarding the QoS-Conditionalized BU is described.

3.6.1 Mobile Node

This section describes the behavior required from a HMIPv6-enabled MN to support the QoS-Conditionalized BU. The MN is responsible to initiate the reservation and release of resources on the current data path. As long as no handover occurs, related messages are sent directly from the MN. However, if the MN loses connectivity to its current AR, there is no possibility for the MN to directly release resources on the old data path. Then the MN instructs the MAP, located at the other end of the previous data path,

to release unused resources on its behalf. Essentially two variants of such desired, but indirectly triggered, resource de-registrations via the MAP exist. How the messages sent by the MN for resource de- and re-registration are interpreted by the MAP is described in Section 3.6.2.

- In case of a local handover QoS reservations must be released on the data path below the MAP and reserved on the new path. Therefore a MN simply sends an appropriate BU+QoS message, requesting the reservation of resources on the new path, to the MAP.

- In case of a regional handover the MN must send a binding release message (BU with lifetime = 0, A-flag = 0, and HoA (RCoA) = IPv6-header source address (LCoA)) to the previous MAP.

If none of these variants fit, eg when the MN incurs a total loss of connectivity, installed resources on the data path are reserved until their lifetime elapses.

A description of the extensions required for a MN to perform this behavior is given in Figure 3.11. To illustrate the modifications regarding the description for a HMIPv6-enabled MN (Figure 3.4), the extensions for support of the QoS-Conditionalized BU are represented in grey color. An arrow indexed with a star indicates that no further extensions to Figure 3.4 are necessary for this branch.

3.6.2 MAP and Intermediate Router

This section describes the behavior for entities serving as MAP or IR to support a MN's QoS-Conditionalized BU. Both entities are required to install, maintain, and release resources as requested by a traversing QoS-hop-by-hop option from a MN. A description of the new operations required from an IR to perform this behavior is given in Figure 3.12.

The MAP is additionally responsible to initiate the release of resources on a MN's previous data path. Therefore the MN instructs the MAP, located at the other end of the previous data path, to release unused resources on its behalf. Essentially three variants of resource de-registrations exist for a MAP. Which of the following three cases fits can be deduced from the content of the received BU(+QoS) message.

- In case of a local handover QoS reservations must be released on the data path below the MAP. Reservations in the MAP must be updated for the new path.

- In case of a regional handover the previous MAP only receives a binding release message (BU with lifetime = 0, A-flag = 0, and HoA = IP-header source address) form the MN. This message must trigger the MAP to initiate the release of reserved resources on the whole data path including the MAP.

- In case of binding release because of insufficient resources, the MAP must reply to the MN with a negative BA+QoS message. All previously reserved resources for the MN must be released.

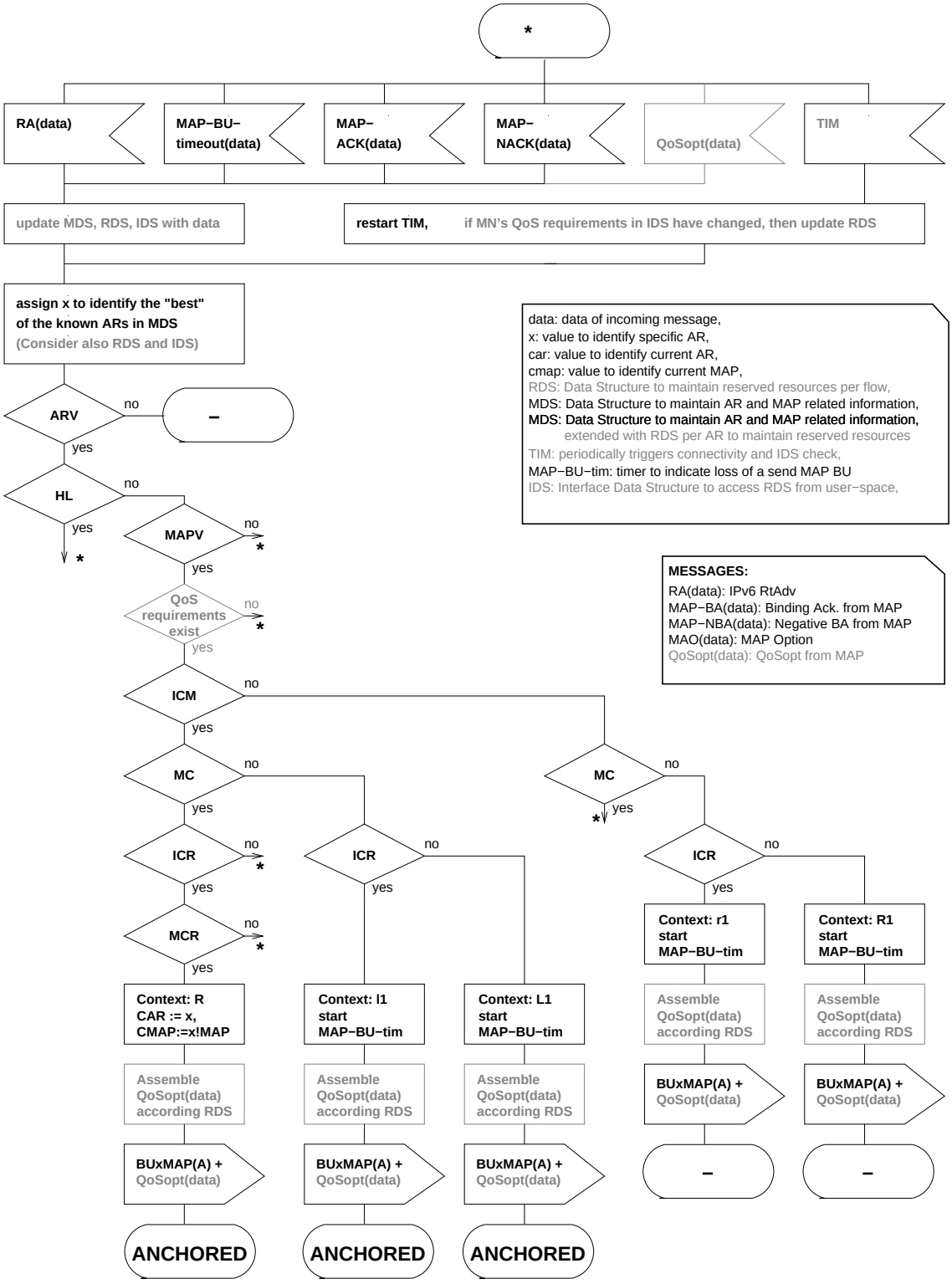


Figure 3.11: Pseudo-SDL description of extensions required to enable a MN for the QoS-Conditionalized handover

If none of these variants fit, eg when the MN incurs a total loss of connectivity, installed resources in the MAP and IRs are reserved until their lifetime expires.

A description of the new operation required from a MAP to perform this behavior is given in Figure 3.13.

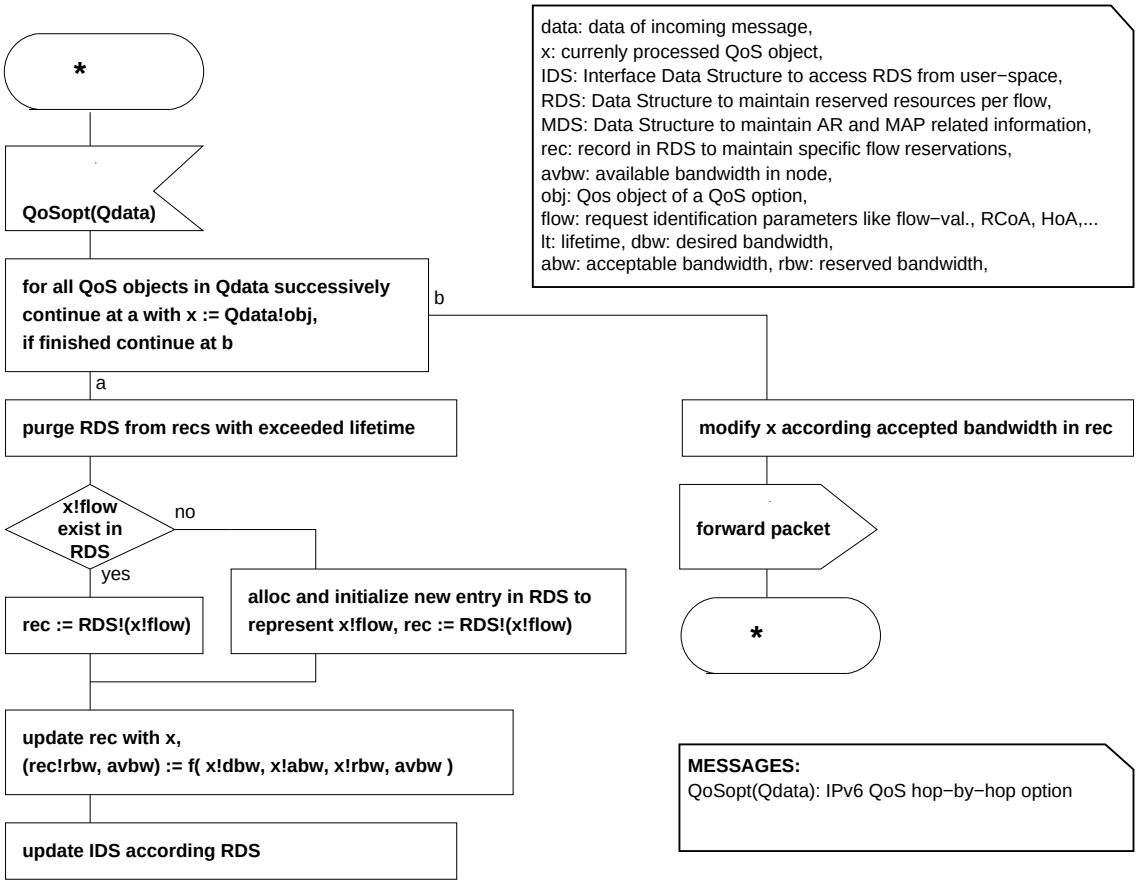


Figure 3.12: Pseudo-SDL description of extensions required to enable an IR for the QoS-Conditionalized handover

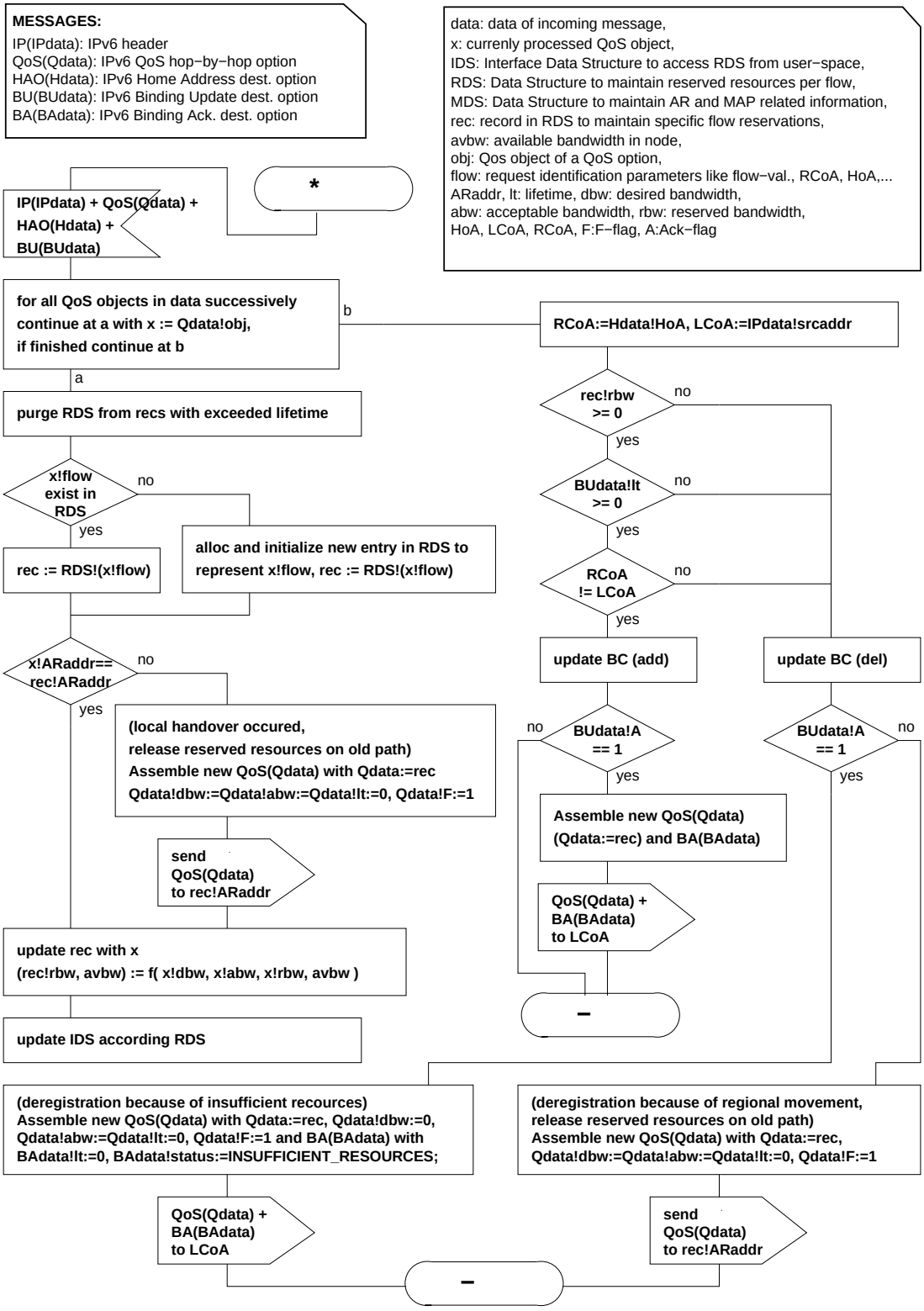


Figure 3.13: Pseudo-SDL description of extensions required to enable a MAP for the QoS-Conditionalized handover

3.7 Summary, Open Issues, and Possible Solutions

In this Chapter the underlying concepts by means of informal specifications of the HMIPv6 and QoS-Conditionalized BU proposal have been analyzed and significant procedures for this work have been summarized. Thereby, the following shortcomings for support of the QoS-Conditionalized BU have been identified and discussed with a possible solution.

- In an access network, enabled for the QoS-Conditionalized BU, the use of the RCoA as source address should be forced by the MAP by setting the P-Flag in the MAP Option (MAO) used for MAP discovery. This is necessary if a MN performs a local movement but the the MN's flow identification values (eg its source address) must be transparent for nodes outside the visited region.
- The reorganization of a QoS option was necessary in order to allow a less overhead driven signaling of QoS parameters to the network.
- Significant information for efficient maintenance of QoS reservations have been identified. A new "flow classification parameters" field has been proposed to signal these informations to the involved nodes on the path.
- In particular the requirement to release stale reservations on an old data path of the MN has been neglected in the QoS-Conditionalized BU specification. A solution has been developed how this operation can be superseded by the MAP of that data path.

Finally, the resulting behavior of the distinct entities has been re-examined in a more formal way to provide a basis for an implementation design.

Chapter 4

From Concept to Praxis

4.1 Selection of Environment

In order to implement the necessary functionalities for the QoS-Conditionalized BU in a HMIPv6 environment, an existing Operating System (OS) environment needed to be found first. The Microsoft Research (MSR) , Linux-2.4, and FreeBSD-3.4 OS have been identified for further evaluation. All of these OSs provided an IPv6 and MIPv6 implementation, where the FreeBSD-3.4 OS even offered an HMIPv6 implementation. Therefore, the obvious candidate would have been the only available HMIPv6 implementation from INRIA, based on the FreeBSD 3.4 OS.

To backup this prematurely decision, a testbed based on the INRIA HMIPv6 implementation has been setup. After intensive preoccupation with the INRIA implementation, this option was finally dropped due to its unstable code, an outdated version of the underlying HMIPv6 proposal and OS, and because no longer maintenance support was provided, neither for the OS itself nor for the HMIPv6 implementation.

Finally, the decision was to take the Mobile IP for Linux (MIPL) [20] implementation from Helsinki University of Technology (HUT) and it's Telecommunication and Multimedia Lab. The following pros could be assigned to this decision. The Unix like Linux OS, a free and well documented OS proved to be an excellent option for development of prototype network functionalities in various cases. A reasonable amount of experience with the Linux OS already exist and was assumed to facilitate the development process. Together with the MIPL implementation an open, very stable, and well maintained IPv6 and MIPv6 implementation is available. A very responsive MIPL developer and user mailing list indicated the wide usage, acceptance and opportunity for discussion of potential future problems.

At last, a new MIPL-based testbed has been setup to gather experience with the implementation and its configuration. The MIPL source code has been explored and possible hooks for the necessary extensions have been identified.

4.2 Linux Operation System

The Linux operating system and in its kernel is an evolutionary project which has been developed since 1991 by hundreds of developers around the world. As a consequence of this process, no complete design documentation exist that could be used as a basis for planning the introduction of new functionality. Information covering the Linus OS and related features is available by means of books, docs, howtos, manuals, mailing lists, and FAQs (frequently asked questions). Information I found to be useful for this thesis is given in [2, 27, 43].

An alternative approach to understand the functionality of the Linux OS and related programs is of course given through the source code itself. This is available for most of the programs contained in the common Linux distributions. Anyway, in order to extend an existing implementation with new functionality an intensive study of its source code is indispensable. Regarding the Linux kernel, analyzing its sources seems to be the only kind of available documentation for many parts of its implementation, especially that of the IPv6 network layer.

In order to simplify this process, the usage of some kind of a source browser tool is recommended. The tool I selected for this purpose is called Linux Cross Reference (LXR) [19] and allows versatile cross referencing for relatively large code repositories (such as the Linux kernel). The main feature of the indexer is of course the ability to easily jump to the declaration and reference(s) of any identifier such as a variable, structure, and function declarations as well as preprocessor macro definitions.

In the following selected features of the Linux OS will be summarized and their significance according the QoCoo implementation will be stated.

Kernel Modules
The Linux kernel has the possibility to separate part of its functionality in modules that can be loaded when needed and unloaded when becoming obsolete. This provides the running operating system with the capability to potentially offer a lot of functionality and at the same time keep the amount of code and processes currently running in kernel space relative small. Another advantage of loadable kernel modules emerge when developing new kernel functionality. By way of that, the possibility is given to load, test, unload, and modify some kind of prototype implementation in kernel space even multiple times without the constraint to recompile the complete kernel or reboot the whole system. Once the module is loaded into the kernel also other kernel functionality can be accessed.

Regarding the QoCoo implementation all the new kernel functionality required is either provided as an individual kernel module or applied as an extension to an already existing one. But also many of the features available for the Linux system that are used in the QoCoo implementation are provided as kernel modules.

Proc-Filesystem
The proc-filesystem is a special filesystem, which only exist in the memory of the Linux system. The kernel and kernel modules can use this filesystem as an interface to the

userspace. The kernel module can create a special directory tree and files in the filesystem, which can be read with normal user tools like cat. The other way around, it is possible to write a file to the proc-filesystem in order to pass parameters to the kernel.

In the QoCoo implementation, the proc-filesystem is used to define requested resources for an MN's flow and provide feedback (about whether the operation succeeded) to the user space. In the intermediate routers and the MAP it is used to access available resource information.

4.2.1 IPv6 Module

The IPv6 kernel module provides all the functionality necessary for processing of an IPv6 packets between the network device and the final application in user space. This includes the basic routing, ND, ICMPv6, and extension header processing as well as functionality related to the transport layer protocols: UDP and TCP. To allow the dynamic control to the various IPv6 protocol parameters several new entries are made to the proc-filesystem. Thereby, for example auto configuration or forwarding via specific interfaces can be enabled or disabled. In order to provide limited access to the use of destination headers from user space via RAW sockets, the "Advanced Sockets API" as suggested by [39] is at least partly supported. If IPv6 packets and extensions headers have to be sent from kernel space, an IPv6 RAW socket can be allocated and passed with the appropriate prepared data structures (to hold the extension header data) to the ip6_build_xmit() function. Then the ip6_build_xmit() function performs the assembling of the packet and ensures further processing.

In the MIPv6 and QoCoo implementation, functionality such as the ip6_build_xmit() function is utilized for generation and sending of signaling messages.

4.2.2 Netfilter Module

Netfilter [34] is a module for packet mangling, outside the normal Berkeley socket interface. It introduces a series of hooks at five well-defined points while a packet traverses across the IPv4 and/or IPv6 protocols stack of an IP node. Whenever a packet being processed is passed along any of these hooks, it is handled to the netfilter module. The position of the IPv6 related hooks and their names are illustrated in Figure 4.1.

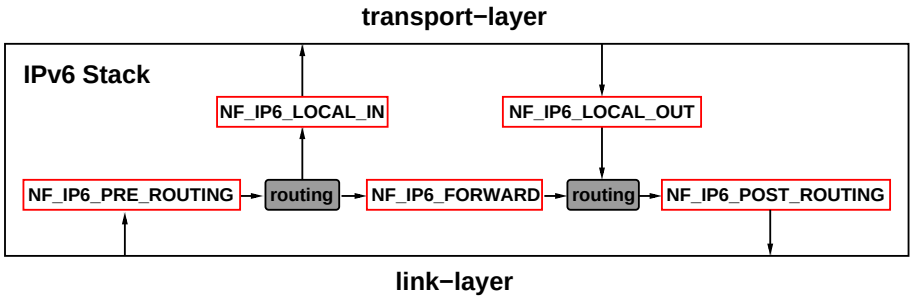


Figure 4.1: Position of netfilter hooks in IPv6 stack

Every incoming IPv6 packet which is delivered from the link-layer to the IPv6 layer first encounters the NF_IP6_PRE_ROUTING hook. Then it enters the first routing part of the IPv6 stack, which decides whether the packet is destined for another node, or a local process. If it is destined for the node itself, it is hooked by NF_IP6_LOCAL_IN before being passed to the next higher layer processing. Otherwise, if it is destined to another node instead, the netfilter module is called again via the NF_IP6_FORWARD hook. The packet then passes a final hook, the NF_IP6_POST_ROUTING hook, before being passed to the link layer again. The NF_IP6_LOCAL_OUT hook is called for packets that are created locally before being processed by the routing part of the IPv6 stack. Other kernel modules can register to access to any of the defined hooks and are thereby provided with the possibility to modify or even drop such intercepted packets.

The traditional purpose of the netfilter was to provide a framework for firewalling. With the further developing it proved to be a very flexible basis for various network control functions such as traffic scheduling, network address translation (NAT) or to enforce other advanced packet treatment. In the QoCoo implementation, the netfilter module is utilized to access the QoS hop-by-hop option of traversing packet.

4.3 Mobile IP for Linux (MIPL) Environment

The previous sections of this chapter summarized the reasons for using MIPL and Linux as the basis for the QoCoo implementation and provided an overview of the underlying Linux operating system In continuation this section gives a brief introduction to the selected MIPv6 implementation environment and how it is embedded in the IPv6 stack of the Linux operating system.

Because a close interaction between IPv6 stack and MIPv6 stack is believed to be mandatory, the MIPL developers decided to implement all related MIPv6 functionality in kernel space [26]. Otherwise splitting the code into user and kernel level would require new interfaces and lead to extra complexity and overhead. The system is provided as a single kernel module, which can be dynamically inserted and removed from the kernel. The module provides functionality for HA, CN and MN. Whether a node running the MIPL system is serving as HA, CN or MN is defined via module load parameter. To allow dynamic (un)loading of the module into the kernel, stubs for MIPL callable functions were introduced into the IPv6 code. This is provided through the mipglue module. If during the processing of a MIPv6 destination option the IPv6 stack passes a stub, it checks whether the MIPL module is loaded and if yes the MIPL related function is called. The advantage of this approach is that the module can be unloaded, modified and reloaded into the kernel of a running system. Figure 4.2 gives a rough overview of the MIPL general architecture.

Several hooks were patched to the IPv6 stack allowing the mipglue module to forward certain calls from IPv6 code to the MIPv6 code. The other way around, once the module is loaded into the kernel, other functionality from the kernel code (in particular the IPv6 code) can be called directly. This avoids the need for re-implementing functionality in the MIPL module which actually already exist in the IPv6 module. Additionally, for interception and manipulation of certain packets the netfilter module is

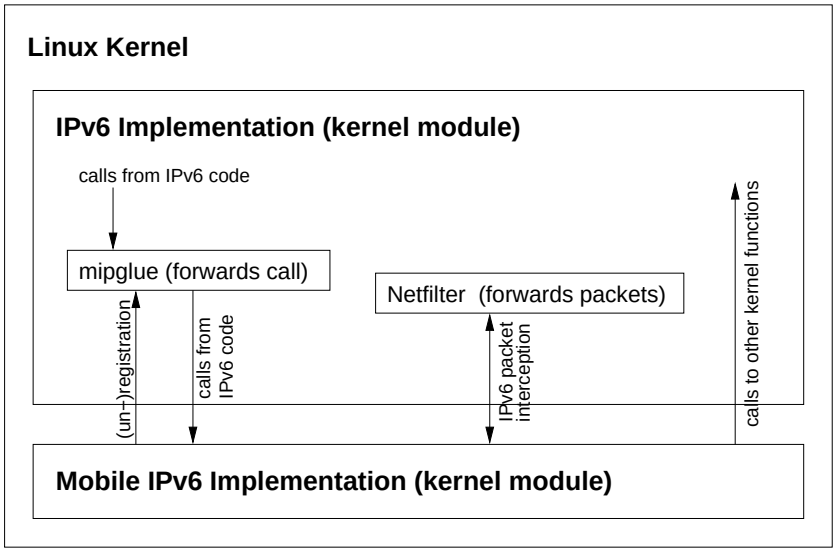


Figure 4.2: MIPL general architecture

used. For example when the HA intercepts packets destined for the MN and when the MN modifies its IPv6 source address to its current CoA. The extensions made to the MIPL module are described in detail in Section 5.1 for support of HMIPv6 and Section 5.2 for the QoS-Conditionalized BU.

Chapter 5

Implementation Design

This chapter describes how the concepts discussed in Chapter 3 were applied to the surrounding implementation environment in Chapter 4. In particular, the Linux operating system and related features, the MIPL version 0.9 MIPv6 implementation, and the router advertisement daemon (radvd) is either applied or extended. Where the gap between abstraction levels in the previous section and the final description on implementation level is still considerable, another intermediate overview will be provided for a better understanding. In conformance with Chapter 3, the HMIPv6 and the QoS-Conditionalized BU part is discussed independently.

5.1 Implementation Design of HMIPv6 Functionalities

The implementation design of HMIPv6 requires extensions to the nodes serving as MN, MAP, or Intermediate Router (IR).

5.1.1 Mobile Node

All required features for an MN to support HMIPv6 are extended into the MIPL code. The following text provides a detailed description to support the behavior outlined in Section 3.4.

Overview of Data Structures
The MIPL implementation manages the known ARs in a list where it records AR-specific information like the AR address, its lifetime and the assigned CoA. In order to maintain the available MAPs and related information, the MN data structure must be extended. Figure 5.1 gives an overview of the data structure related to movement detection. The extensions made for HMIPv6 are highlighted.

The pointer curr_router identifies the AR serving currently as default router. Another pointer next_router is temporarily assigned while performing a handoff. A list of MAPs is added, to hold MAP-relevant information such as its RCoA and lifetime. In addition, the AR structure is extended with several new fields, particularly an LCoA field, a context field and a pointer to a potentially affiliated MAP available via this AR.

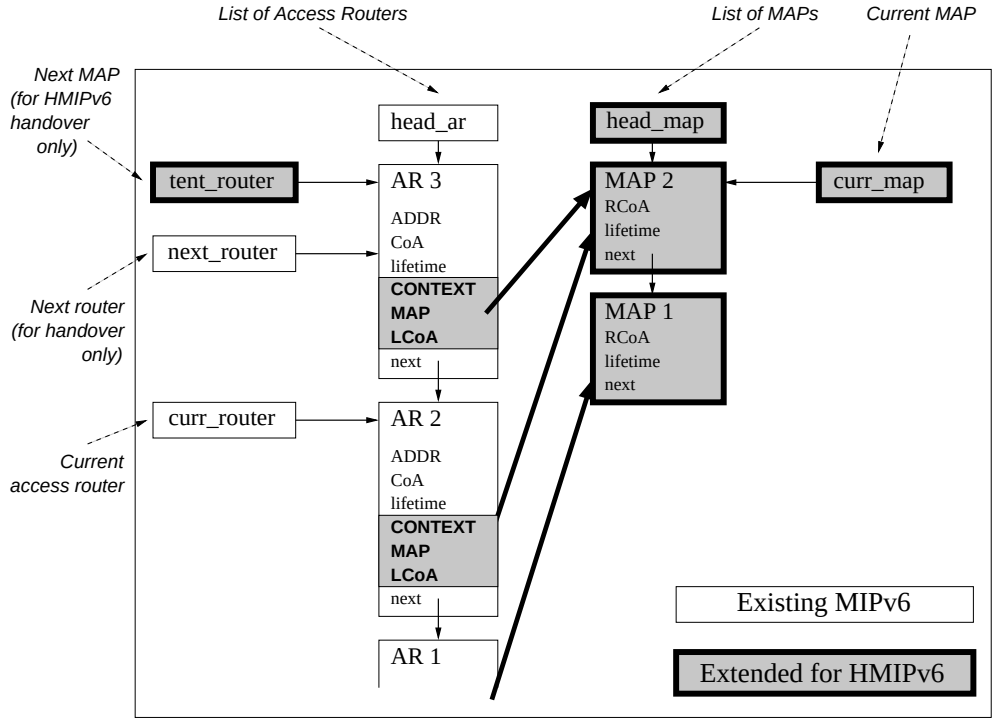


Figure 5.1: Overview of MN data structure for mobility environment

The “context field” is introduced to simplify the evaluation of the available ARs in order to select the best-suited default AR among all known ARs. It is explained in more detail later in this section.

Two new pointers are introduced to reflect the current relation of the MN to the available MAPs. The pointer tent_router is only temporarily assigned while performing a handover in HMIPv6 mode. While emitting the necessary MAP registration message, it points via the AR selected for MAP registration to the next MAP. The identified router of the pointer remains tentative until registration with the MAP of this AR is acknowledged. After acknowledgement by the next MAP, the pointer curr_map is reassigned to identify the new, currently in use, MAP.

Overview of Handover Processing

Handover processing of the MN involves a series of procedures. The focus of the design lies in the strict isolation for handling incoming events, updating the data structure, making a decision and finally executing the decision. Figure 5.2 illustrates the function call sequence related to MN event handling.

The function router_event() is called if any of the following events occurs:

- a RtAdv handled by the function ra_rcv_ptr().
- a positive or negative BA in response to a BU which is handled by the function ack_rcvd().

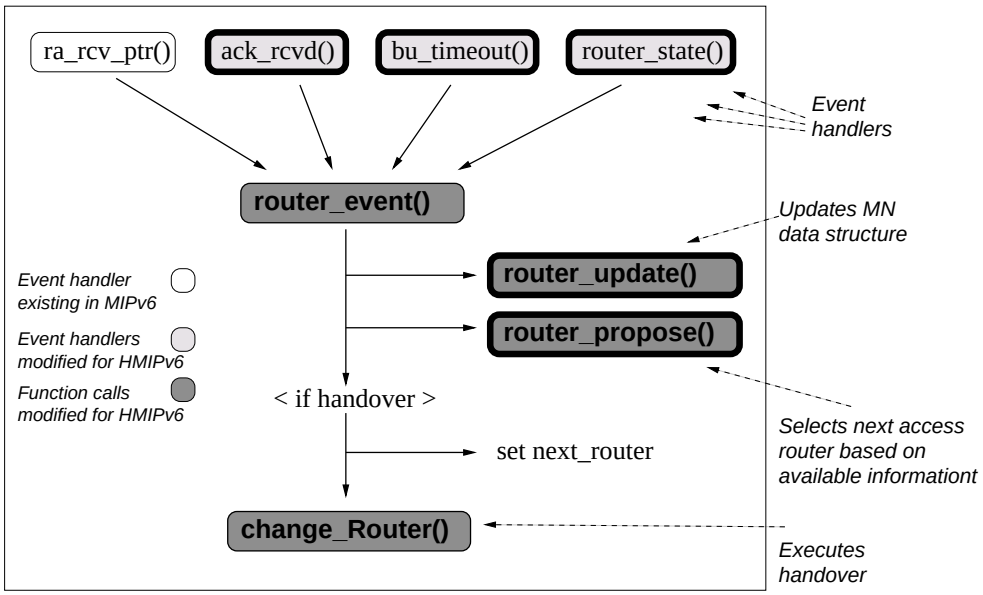


Figure 5.2: Overview of MN event handling

- a timeout indicating the loss of a previously sent BU handled by the function bu_timeout().

- a periodic timer calling the function router_state() in order to check the current AR and MAP binding for expired lifetime.

After that, if new information is available, router_update() is called to store them in the MN data structure. The function router_propose() is responsible for making a "just-at-the-moment valid" judgment based on the currently available information and a certain policy. The method and rules used to enforce this policy is explained in the next paragraph. If the judgment results in selecting a new AR as default router, a handover is performed. This is indicated by the pointer next_router that is assigned to identify the next router in the AR list. Eventually change_router() is called to execute the relevant registration procedures according to the latest judgment.

The method used in router_propose() to select "the best" next router and to identify the next steps to be executed is illustrated in Figure 5.3.

This is achieved by using the "context field" that has been introduced in Figure 5.1. The function parses the complete AR list and sets for every AR specific property bits in its context field. Each property bit reflects a certain property of the AR. The identified properties partly consist of the conditions used in Figure 3.4 and described in Table 3.3 to determine the registration actions of a MN regarding a specific AR. A property-bit to reflect the recent behavior of an AR has been added. Where the overall and quite complex connection state of the MN is represented through a number of pointers, address and lifetime fields, the context field summarizes the "just-in-the-moment valid" context in only one field for every single AR.

Additionally, these flags are arranged in an order (depending on the policy) such

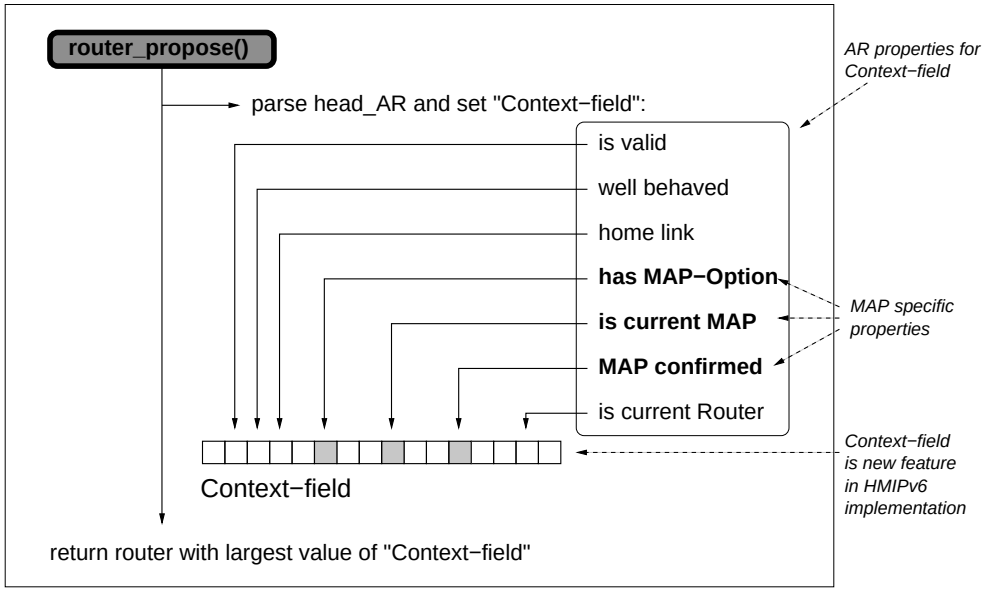


Figure 5.3: MN policy method

that the router with the largest value in the context field is regarded as the currently most promising router. The reasons for the selected policy in this implementation is given in the following, ordered from the highest towards the lowest assumed priority:

- The most important property of an AR is the fact whether it is valid or not (depending on its lifetime and timestamp of the last received RtAdv).
- Also important is the fact whether any problems occurred recently with that AR or its related data path in the AN. If several MAP registrations sent via this AR are not acknowledged before the MAP-BU-tim timer indicates the unsuccessful registration, the population of this bit is denied. Other doubts regarding the path may be coupled to this property-bit later. An affiliated timestamp and default lifetime assures that the judgement of an once as “not well behaved” convicted AR can elapse.
- If the evaluated AR is located in the home link the third highest priority is assigned. By accepting this AR as the MN’s default router, all kind of mobility management becomes unnecessary at all.
- When the MN has lost connectivity to the home link but is within the coverage of one or several foreign AR, MIPv6 or HMIPv6 mobility mechanisms are used. In this case an AR which also advertises the availability of a MAP and thereby offering support for micro-mobility is preferred.
- Even more desirable for an HMIPv6 handover is the capability for a local movement. This is possible if the AR advertises the availability of the same MAP, with which the MN is currently registered.

- Once a MAP registration via a certain path is already acknowledged, another tiny benefit is given with the corresponding AR since the MN does not need to send a new registration message.
- Finally, to avoid superfluous handover, the current router is preferred as long as another available AR can not come up with any additional benefits.

In rare cases, if the evaluation of several routers result in the same context-field value, then ties are broken in preference of earlier added routers.

This method provides a good means for the evaluation of the known AR and simplifies the identification of necessary registration operations to be executed. Moreover, this selection method allows to easily apply different movement policies. This can be achieved by changing the order in which properties are reflected in the context field.

Finally, the identified registration operations are executed through the change_router() function which disseminates functionality into further procedures. Figure 5.4 gives an overview to the change_router() related functions.

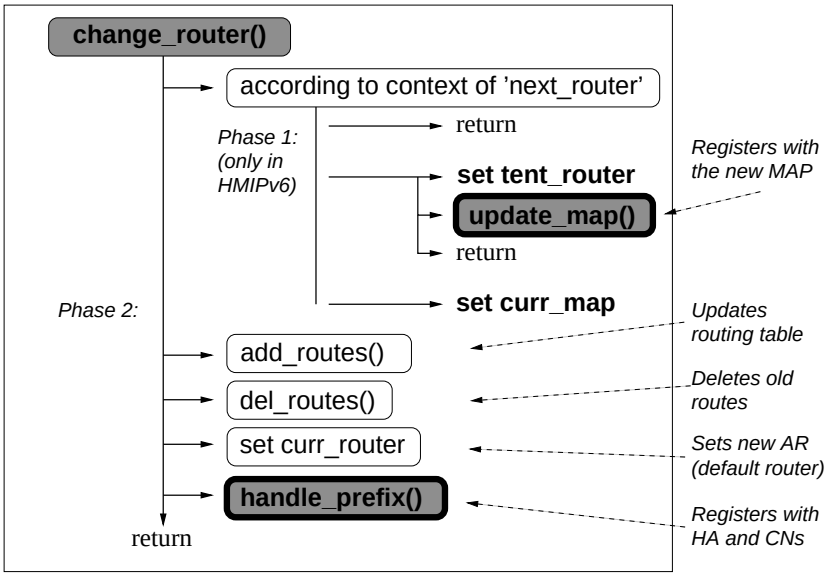


Figure 5.4: MN handover execution

The procedure consists of two phases. The first is concerned with the tentative MAP registration, while the second is responsible for completion of default router establishment and possible HA/CN registrations. According to the context field of next_router, essentially three options exits for the first phase of registration. If the next_router points to the same router as current_router, processing is only continued if a MAP binding needs to be refreshed because the lifetime of the binding is about to expire. Then, if the context field indicates the availability of a valid MAP via the next router, the pointer tent_router is assigned the next_router value and the function update_map() is called to coordinate sending of the MAP BU. It should be noted that according to the current router nothing has changed yet. The router selected for MAP registration is yet only tentative. Thus, a routing header must be used before the HAO and BU destination option

to assure that the packet does not travel via the default router towards the MAP (see also Section 3.1). Eventually, the function returns.

If the context field indicates that the registration with the MAP has already been confirmed, the third option of phase 1 is given. Then, the pointer curr_map is assigned to indicate the currently in use new MAP and processing continues with phase two. In case of a movement in MIPv6 mode phase 1 is omitted and processing continues directly in phase 2. It is concerned with the actual movement. This requires to add the new default router to the routing table (function add_routes()) and purge it from entries related to the old AR (function del_routes()). The pointer curr_router is reassigned to indicate the currently in use AR and eventually the function handle_prefix() is called to coordinate the sending of BUs to HA and CNs. However, registration with the latter is only performed in MIPv6 mode or if the MN has undergone a macro movement in HMIPv6 mode. To send and keep track of current bindings, existing functionality was used and required only minor modifications. For example, the function get_coa() is used to access the current CoA. It is modified to always provide the correct CoA for HA and CN. This would be the CoA (LCoA) in MIPv6 mode and the RCoA in HMIPv6 mode.

Description of Extended Modules, Functions and Data Structures

The MIPL system consists of a bunch of modules (.c and .h files) in order to group functionality. Modules, which have been extended are summarized in Table 5.1.

A detailed reference of the extended data structures and functions is given in the Appendix A.1.

5.1.2 Mobile Anchor Point and Intermediate Router

An HA implementation can be used for a MAP but requires extensions for MAP discovery. Since MAP discovery is an extension to the Neighbor Discovery protocol and in particular to the part responsible for router discovery, extensions to the code providing this service is sufficient. In the Linux operating system, sending RtAdvs is performed by the router advertisement daemon (radvd) [33]. This is a program running in userspace of a Linux IPv6 router and can be configured to initiate router discovery via its interfaces. The MAP discovery service includes advertising of a MAP's own MAP option as well as the receiving and propagation of options from other MAPs. Therefore, the described functionality in this section fits also for intermediate routers by assuring the propagation of received MAP options. The desired behavior of such entity eventually depends only on the configuration.

For the QoCoo implementation the source code of the radvd version 0.7 has been extended for MAP discovery as described in Section 3.4.2. The source files from radvd that needed modifications are given with a brief description in Table 5.2.

After initializing and reading the configuration file, the radvd eventually enters two loops for every configured interface. One is responsible for advertising multicast RtAdvs at the configured advertisement rate. The other one waits for incoming RtAdv or RtSol messages and replies with the corresponding RtAdv in case of RtSol. A detailed reference of the extended data structures and functions is given the Appendix A.1.

Module name	Description
mipglue.c	Handles integration between IPv6 module and MIPL module. Nothing has changed here. The only new suboption that needed to be processed is the MAP Option (MAO) piggybacked by a RtAdv message. This option can accessed when parsing the contained sub-options of a received ICMPv6-RtAdv message.
mipv6.c/.h	Main source file for MIPv6 module. Handles shared resource allocation and some general purpose functions. Also contains the structure definitions for the various extension headers. This module has been extended with data structures for new HMIPv6 related suboptions. General purpose functions have been modified when necessary to consider HMIPv6 related processing.
mdetect.c/.h	Encapsulates the used movement detection algorithm. New data structures are introduced for maintenance of known AR and MAPs. The processing of movement detection has been reorganized to support micro mobility in a HMIPv6 environment.
mn.c/.h	Contains Mobile Node (MN) specific functions for managing registration procedures with HA. Has been extended with functions execute registration and de-registration operations of a MN with a MAP.
router.c	Helper functions for mdetect. Various new helper functions according to the requirement of a HMIPv6 aware MN have been introduced and stored here. Existing ones have been adapted to the new functional requirements.
sendopts.c/.h	Handles outgoing options management. Extended for management of HMIPv6 related outgoing options such as the BU to the MAP, the use of a routing header with tentative registrations, and the start and handling of timers to indicate the loss of a previously sent registration message.

Table 5.1: MIPL modules extended for HMIPv6

Module name	Description
gram.y	Grammar description for configuration file in form of a LARL(1) context-free grammar. It serves as an input files for bison [15], a tool to generate c program to parse that grammar. The grammar has been extended for syntax to configure interfaces for MAP discovery. For the syntax see also QoCoo manual.
scanner.l	Describes lexical patterns of grammar. It serves as an input file for flex[16], a tool for generating "scanners": programs which recognize lexical patterns in text. Together with bison the generated .c files provide the functionality to process the configuration file of radvd and store the information in data structures accessible while running radvd. The scanner.l file has been extended with the rules and expression necessary for configuration of MAP discovery.
interface.c	Contains functions to initialize the default RA values for every configured interface and check values assigned by the configuration file. Is extended with functions for MAP advertisement defaults.
process.c	Contains functions for processing of received RAs and RSs. Has been extended with the capability to understand and process MAP options in received RAs and update the MAP list with new MAP information.
radvd.c	Main source file for radvd. Essentially reads in parameters and configuration file, opens ICMPv6 socket and enters loop. Holds root pointer to interface data structure. The only modification to this file was to extend functions calls with new parameters and add a root pointer for data structure to maintain received MAP options.
radvd.h	Contains structure definitions for interface configuration. Is extended with structure definitions to maintain the own configured and received MAP information. Also holds structure definition for MAP option header, some macros definitions and constants.
send.c	Contains function to assemble and send RAs on an interface based on the given interface configuration. Has been extended to attach MAP option to sending RA
timer.c	Contains timer and alarm handler. A function for returning the absolute system time was added.

Table 5.2: Radvd modules, extended for MAP discovery

5.2 Implementation Design of QoS-Conditionalized Binding Update Functionalities

In order to integrate functionality for support of the QoS-Conditionalized BU in the QoCoo-HMIPv6 implementation, further extensions are required for the MN, the MAP, and the IRs of an AN. Common for all entities is the requirement to process received QoS-hop-by-hop options. This is accomplished by introducing a new netfilter module called qos-monitor which is responsible for intercepting of traversing QoS options and managing of further processing.

I call the receiving, processing, maybe modifying and forwarding of a QoS option passive processing because it does not require to generate any new QoS options. In addition to this passive processing, the MN and the MAP must be capable to generate new QoS options to be attached to certain BU and BA messages. Utilizing netfilter for generation of new extension headers is much more complicated and can cause new problems (For example, if a QoS option is to be added to an existing packet it might not fit into the maximum transfer unit of the path). Therefore the active processing of QoS options is done directly in the mobility module when assembling the total packet. Responsibilities of the netfilter based qos-monitor module and the MIPL based mobility module for passive and active processing of a QoS option is illustrated in Figure 5.5.

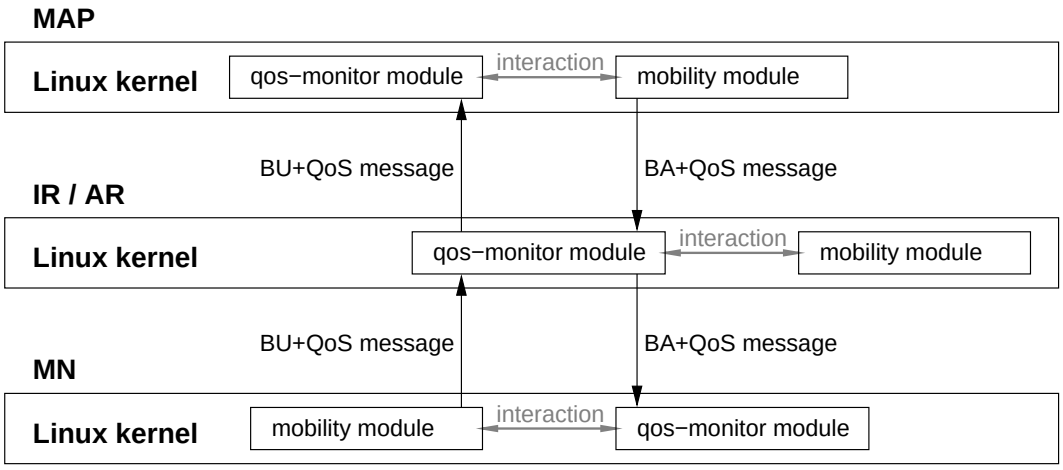


Figure 5.5: Responsibilities of QoS option processing

Further handling of the information conveyed in a QoS option then depends on whether the node is an MN, an IR or a MAP. It is performed according to the description given in Section 3.6. The necessary extensions were applied to the HMIPv6-extended-MIPL implementation. Table 5.3 gives an overview of the modules introduced or extended for support of the QoS-Conditionalized BU. A detailed reference of the extended data structures and functions is given in the Appendix A.2.

5.2.1 Mobile Node

For the MN, every initiated QoS request relies on a specific data path in the visited AN. In the QoCoo implementation a path is represented by its leaf node the corresponding AR. Therefore, the maintenance of requested and successfully reserved resources is coupled with the data structure maintaining the known AR information. The MN's AR data structure (introduced in Section 5.1.1 and Figure 5.1) has been extended respectively (See MDS and RDS in Figure 3.11).

Additionally, the MN's proc-file system has been extended to dynamically pass flow and resource-request parameters from user-space (introduced as IDS in Figure 3.11) to the related functionalities in the processing kernel module.

5.2.2 Mobile Anchor Point and Intermediate Routers

A new data structure (RDS in Figure 3.13 and 3.12 is also introduced for nodes employed as MAP or IR to maintain the reserved resources of a MN. This data structure consists of a list, which is identified by the pointer head_resr. Each list element holds flow identification parameters, the affiliated QoS requirements and the lifetime status as deduced from corresponding QoS objects. To allow access to the router's total available and reserved bandwidth and provide information about the currently maintained QoS flows the proc-file system has been extended with new entries (IDS in Figure 3.13 and 3.12).

For IRs the processing of received QoS-hop-by-hop options mainly consists of checking available resources, manage required QoS reservations, and maybe modify the traversing QoS option. All necessary processing management is controlled by the qos-monitor daemon. However, several helper functions from other IPv6 related modules are called if required.

For a MAP the initial processing of a received QoS-hop-by-hop option is identical to that of an IR. Regarding the actual registration the main difference is that the MAP, instead of forwarding a maybe modified QoS option, has to reply this option to the MN, coupled to the corresponding BA. Therefore the existing functions for managing the MAP's binding operations have been modified. Whenever the MAP is processing a BU and modifying its BC, it consults its Resource Data Structure (RDS) for entries that are related to this binding. If this is the case, two new options exist. If the BU has been received with an attached QoS option and indicates a MAP registration, a QoS option is also attached to corresponding BA. If the BU actually indicates a Binding release or no QoS option is attached, a de-registration due to a regional handover is assumed. Then the sending of a QoS de-registration message to the MN's old AR is initiated to release resources on its previous path.

Another issue is the de-registration of reserved resources on the MN's previous path. In case of a local handover, the MAP receives a new BU+QoS message. By comparing the content of the new QoS objects with the maintained information for that flow, the qos-monitor module in the MAP deduces the MN's movement to a new AR and initiates the sending a QoS-release message towards the old AR.

Module name	Description
qos-monitor.c	Contains functions for initializing of QoS-hop-by-hop daemon which is responsible to monitor traversing QoS options. According to the node's purpose the qos-monitor daemon is running on (MAP, IR or MN) it manages the core processing of received QoS options and required reservations. For processing several helper functions are called from other IPv6 related modules.
mipv6.h	Extended with new data structure declarations for QoS signaling and maintaining requested resource reservations in the IRs.
mipv6.c	Extended with pointer and variables to maintain requested resources in the IRs.
mdetect.h	Extended with new data structure declarations for coupling the maintenance of QoS requests with the maintenance of known ARs in the MN.
mdetect.c	Movement-even-handling functions have been extended with capability to consider desired, acceptable and finally successfully reserved QoS requirements
router.c	Provided with new helper functions for mdetect.c
procrcv.c	Modified with hooks in the MAP function sequence to consider QoS related operations when processing an received BU+QoS message.
sendopts.c	Extended with Functions for managing sending and coupling of QoS-hop-by-hop options with BU destination options. Additionally several helper functions for maintaining reserved resources and flow parameters are stored in this file.
dstopts.c	Extended with functions to assemble QoS-hop-by-hop extension headers.

Table 5.3: Modules, extended for QoS support

Chapter 6

Testing

This chapter discusses the process of testing the new functionalities provided through the QoCoo implementation and the approaches being made for accomplishing performance results in order to prove the benefits of the QoCoo scheme. Section 6.1 describes the testbed that has been setup as a basis for emulating test cases and perform measurements. The emulated scenarios are summarized in Section 6.2. Performance result are given in Section 6.3.

6.1 Testbed

A testbed has been setup from the beginning of implementing and was gradually adapted to new emulation requirements. This way, the testbed has been developed from a rather simple one for MIPv6 functionality tests to a much more complex version capable of emulating micro- and macro-mobility movements as well as QoS-conditionalized handover.

The aim of the testbed is to provide a network environment for emulation of certain scenarios. On one hand, the network environment is required to behave similar to significant properties of the real world; on the other hand, insignificant and indeterministic behavior should be avoided. Especially it is very important and desirable to eliminate indeterministic behavior in order to allow the reproduction of once performed observations. The following network properties were identified as significant for testing of QoCoo functionality and are integrated in the testbed.

- The testbed must include a HA, CN, and a MN.
- To emulate a macro movement at least two ANs covered by a MAP should be available.
- Each AN must contain one or several ARs to allow the MN to access the foreign link. To emulate a micro movement at least one of the ANs should advertise the availability of several links.
- To trigger the movement of the MN, it must be possible to enable and disable the connection of the MN to any of the available ARs and its advertised links.

- The transmission of a message always causes a certain delay until it is received by its destination node. This transmission delay in real wide area networks (WANs) as well as in real access networks (ANs) is typically much longer than that in a testbed, built up in a single room. To emulate the effects of the QoCoo scheme that can be expected from its deployment in a real WAN/AN environment, some transmission delay (for packets traversing the system model's WAN part and AN part) must be enforced artificially.

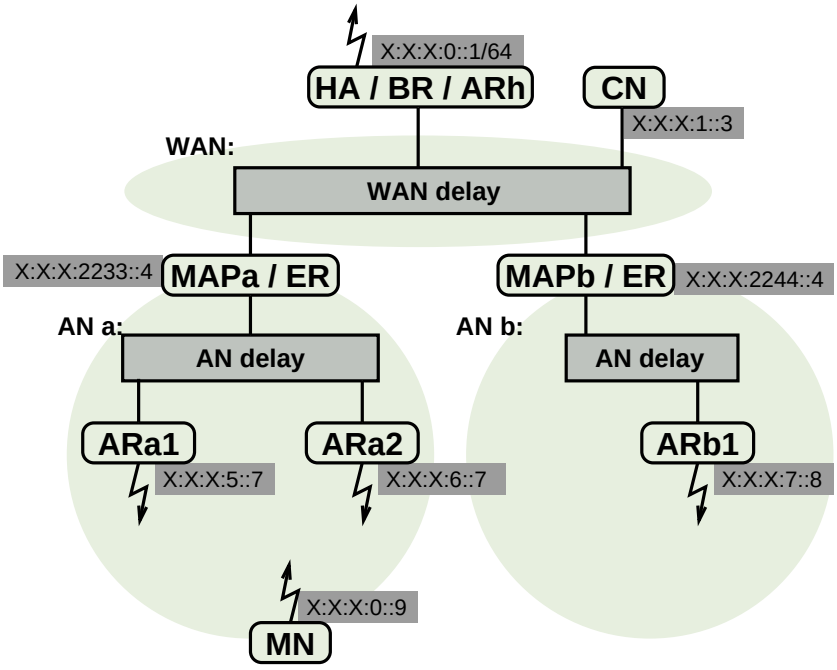


Figure 6.1: Testbed setup from IPv6 point of view

In Figure 6.1 a simplified testbed setup from a IPv6 point of view is illustrated. It conforms mostly to the system model given in Figure 3.1. The relevant IPv6 address of the different entities are given in the dark grey boxes. For the HA, ARa1, ARa2, and ARb this is the address which is advertised by their RtAdvs. MN and CN are depicted with their HoA. The address depicted next to the MAPs is the address which is carried by means of MAP discovery to the MNs. An artificial but controlled WAN-delay is enforced for all IPv6 packets traveling between a MAP and the HA or CN. Another controlled transmission delay is introduced in the two AN of the model and is referred in the following text as the AN-delay. This affects all packets traveling between any AR and the responsible MAP of each AN. Since no IPv6 enabled WAN emulator, capable to enforce some controllable delay to traversing packets, could be found the Nist Net Delay tool [8] for IPv4 was used. To accomplish the delay of IPv6 packets, they are routed through an IPv4 tunnel via an additional PC serving as IPv4- WAN- and AN-emulator.

A detailed description of the testbed setup is given in Figure 6.2. The Figure shows 6 physical Linux PCs where some of these machines virtually serve for multiple purposes.

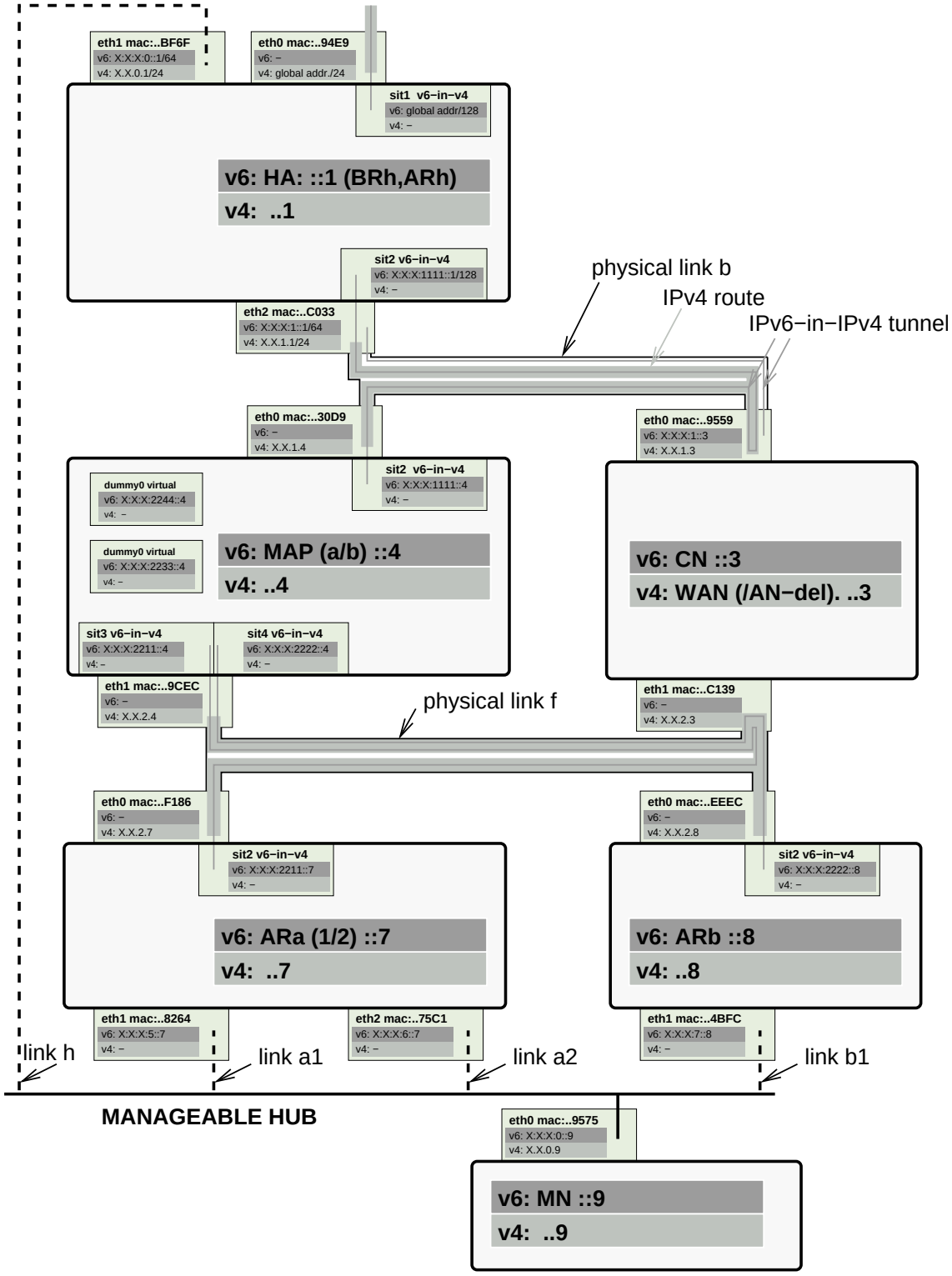


Figure 6.2: Concrete testbed setup

These machines are named as HA, MAP, CN/WAN, ARa, ARb, and MN. The smaller, black-bordered boxes represent network interfaces, where those attached from the outside represent real physical interfaces and those in the inside virtual, tunnel or dummy interfaces. All interfaces are illustrated with an abbreviation for its type. If the interface is IPv6-enabled, its most relevant IPv6 address is depicted in the dark grey box. If the interface is IPv4-enabled, its IPv4 address is given in the light grey box.

All computers are Intel Pentium II machines with a CPU speed between 300 and 500 MHz. All Network Interface Card (NIC)s are 100BaseT 3Com cards. Because link b and f employ only a 10Mbps hub, the bandwidth of these links is reduced to 10 Mbps.

The WAN-delay is enforced to all IPv4 packets, which are sent or received from interface eth2 of the HA or eth0 of the MAP and which are routed via interface eth0 of the machine WAN. From IPv6 point of view, the tunnel route provided by the sit2 interfaces on the HA and MAP is the only possible packet exchange route between HA and CN on one side and the MAP on the other side of the modeled wide area network. In this setup the HA also serves as ER for the CN which is physically located in the same machine as WAN and AN emulator. However, since only the CN's interface eth0 is IPv6 enabled and the HA is the its only known IPv6 default router, all IPv6 packets leaving the CN are routed via the HA. The IPv6 layer of the CN does not know about the IPv6-in-IPv4 tunneled packets.

The machine MAP serves as MAP with two different address spaces and tunnel interfaces virtually like two MAPs for AN a and b. Therefore the QoCoo-modified router advertisement daemon (radvd) is configured to advertise the availability of MAP (1) with the RCoA-address space X:X:X:2233::/64 via interface sit3 and the availability of MAP (2) with the RCoA address space X:X:X:2244::/64 for AN b via interface sit4.

The AN-delay between the MAP and the AR a and b is achieved in the same way as the WAN-delay described above. None of the physical interfaces connected to link f is enabled for IPv6. For AN a, all packets traveling between the ARa and the MAP are IPv6-in-IPv4 tunneled and routed via the interface eth1 of machine WAN. The machine WAN then delays the packets for a certain amount of time (AR-delay). Finally the ARa serves as AR for its two interfaces eth1 and eth2.

The connection of the MN to the access links of HA, AR a, or AR b is controlled via an Small Network Management Protocol (SNMP) manageable hub. All ports of the hub can be disabled and enabled through certain SNMP commands. When, for example, an overlapping movement from AR a eth1 to AR a eth2 is desired the corresponding ports are turned on and off respectively. In the following the MN's link-connection state will be given as a vector where the terms h, a1, a2, and b1 stand for the home, ARa1, ARa2 and ARb1 link. These terms are followed by a ":0/1" indicating whether the connection to this link is disabled or enabled by the controlling hub. One disadvantage of the manageable hub used in this testbed is the indeterministic duration (between 2 and 6 seconds) it takes from sending the SNMP command until the desired action is finally executed by the hub.

An alternative approach to emulate movements is provided by the Netfilter module. The basic idea is to let the MN physically always be connected to all potential AR interfaces, but drop packets from certain interfaces based on their Medium Access Control (MAC) address. If the packets are dropped before they are further processed by the

IPv6 layer, the MN has no means to deduce the connection to this particular link of the AR. By way of this, tools utilized for performance evaluation can immediately trigger a handoff and have the possibility to consider for example the time when a handoff was executed with a much better accuracy. Also it becomes possible to increase the maximal HO rate from only one HO in 6 seconds (possible with the SNMP manageable hub) to more than one HO per second. This concept of a "virtual manageable hub" employed as a software module in the MN is discussed in more detail in Section 6.3.

Simple connectivity test between the MN and the HA or CN were performed with the diagnostic tool ping6. For evaluation of application packet loss a simple tool called udp6traffic was developed for sending and receiving UDP messages between MN and CN. The transmitted messages and their content were monitored via selected interfaces with the tool tcpdump. If monitoring of certain processing parameters of a node was desired this was achieved by inserting appropriate debug messages in the QoCoo kernel code. The amount of debug message provisioning could be accessed via an entry in the proc-file system, ranging from an error level where only serious problems are reported to an informal level where all interesting function calls and variables are logged.

6.2 Functionality Tests

This section discusses several representative scenarios which were emulated in the testbed in order to show the desired functionality of the QoCoo scheme. It should be noted, that the scope of the selected cases covers by no means all possible cases that might happen within this scheme. It should not be regarded as any prove for the correctness of neither the underlying protocols nor its implementation.

6.2.1 HMIPv6 Test Cases

Regarding HMIPv6 support in the QoCoo implementation the following test cases were performed and proved to work in the described environment.

Regional handover from home link to foreign AN
Scenario Description: MN has connection to the home link (Manageable Hub Configuration (MHC): h:1, a1:0, a2:0, b1:0). At time t1 link a1 is enabled, at time t2 link a is disabled.
Observation: After time t1, the MN receives a RtAdv from the new AR a1. When, after time t2, the lifetime of the last received RtAdv of the HA has expired, a BU is sent via AR a1 to the MAP a. As soon as the MAP BA is received by the MN, the new AR a1 is selected as default router and a BU is sent to the MN's HA.
Conclusion: Works

Local handover
Scenario Description: MN maintains connectivity to HA and CN via ARa1 (MHC: h:0, a1:1, a2:0, b1:0). At time t1 link a2 and b1 is enabled, at time t2 link a1 is disabled.
continues on next page

continued from previous page

Observation: After time t1, the MN receives RtAdv from new AR a2 and b1. When the lifetime of the last received RtAdv from AR a1 has expired a BU is sent to the MAP. The MN's binding status in the MAP is updated and a BA is returned to the MN. In the following, packets destined to the CN use the AR a2 as default router but contain the same RCoA as source address used before the handover.
Conclusion: Works

Regional handover from AN a to AN b

Scenario Description: MN maintains connectivity to HA and CN via ARa2 (MHC: h:0, a1:0, a2:1, b1:1). At time t1 link a2 is disabled.
Observation: After time t1, when the lifetime of AR a2 has expired, a BU is sent to the MAP b. When the corresponding BA is received by the MN another BU containing the MN's new RCoA is sent to the HA and CN. Additionally a BU is sent to the old MAP a to release the old registration.
Conclusion: Works

Regional handover from foreign AN b to home link

Scenario Description: MN maintains connectivity to HA and CN via ARb1 (MHC: h:0, a1:0, a2:0, b1:1). At time t1 link h is enabled.
Observation: When the MN (after time t1) received a valid RtAdv from its HA, a BU is sent to the HA, CN, and MAP b for deregistration. In the following no more mobility management is involved for communication between CN and MN.
Conclusion: Works

Failed MAP registration

Scenario Description: MN is connected via AR a1 (MHC: h:0, a1:1, a2:0 b1:0) and receives RtAdv+MAO messages from AR a1. At time t1 the MAP binding service performed at MAP a is disabled but advertisement of MAOs is still performed.
Observation: When (after time t1) the refresh timer of the MAP's Binding Cache (BC) entry in the MN has expired, a BU is sent to the MAP a. After four unacknowledged registration tries the MN switched back to MIPv6 mode and sent a BU containing the MN's HoA and CoA (LCoA) to the HA and CN. After 20 seconds the registration with the MAP a was retried.
Conclusion: Works

Loss of MAP availability

Scenario Description: MN is connected via AR a1 (MHC: h:0, a1:1, a2:0 b1:0) and receives RtAdv+MAO messages from AR a1. From time t1 on, no more MAO is advertised with the RtAdv emitted by AR a1.
Observation: With the reception of the first RtAdv from AR a1 not containing a MAO, the MN switched back to MIPv6 mode and sent a BU containing the MN's HoA and CoA (LCoA) to the HA and CN.

continues on next page

continued from previous page

Conclusion: Works

6.2.2 QoS-Conditionalized BU Test Cases

This section summarizes several representative test cases regarding the QoS-Conditionalized BU support in the QoCoo implementation. In the QoCoo implementation, this affects only the registration messages exchanged with the MAP of the visited AN. Therefore only the MN's data path in the visited AN network up to the MAP is considered for QoS signaling. In the following description the abbreviation avbw stands for available bandwidth in the data path of an AN. Dbw and abw, respectively, stand for the MN's desired and acceptable bandwidth requirements to the path and rbw for the currently reserved bandwidth on the path according to the feedback provided through the BA+QoS message from the MAP.

QoS request via current path (avbw $\geq$ dbw $\geq$ abw)

Scenario Description: MN uses AR a1 as default router (MHC: h:0, a1:1, a2:1, b1:1). The avbw in MAP and AR a1 is 10Kb, rbw = dbw = abw = 0Kb. At time t1 the dbw is set to 5Kb.

Observation: When the dbw is changed a BU+QoS message is sent towards the MAP. Required reservations for the MN's specified flow are installed in AR a1 and MAP a. A BA+QoS message is returned to the MN. The MN's rbw proc-file entry confirms that 5Kb have been successfully reserved.

Conclusion: Works

QoS request via current path (dbw $>$ avbw $\geq$ abw)

Scenario Description: MN uses AR a1 as default router (MHC: h:0, a1:1, a2:1, b1:1). The avbw in AR a1 is 10Kb and in the MAP 8Kb, rbw = dbw = abw = 0Kb. At time t1 the dbw is increased to 15Kb.

Observation: When the MN's dbw is changed a BU+QoS message is sent towards the MAP. While processing of the QoS option in AR a 10Kb of the AR's available bandwidth is reserved for the MN, the dbw value of the message is reduced to 10 Kb and the message is forwarded towards the MAP. At the MAP all 8Kb of the avbw is reserved for the MN. The returned BA+QoS message's dbw value is set to 8Kb. When the QoS option is processed at the AR a two of the previously reserved Kb for the MN are released and the message is forwarded to the MN. The MN's rbw proc-file entry confirms that 8Kb of the desired 15Kb have been successfully reserved.

continues on next page

continued from previous page

Conclusion: Works

Local HO with QoS request via new path (avbw $\geq$ dbw $\geq$ abw)

Scenario Description: MN uses AR a1 as default router (MHC: h:0, a1:1, a2:1, b1:1). The avbw in MAP and AR a1 and AR a2 is 10Kb, rbw = dbw = abw = 5Kb. At time t1 the link a1 is disabled.

Observation: When the lifetime of the last received RtAdv from AR a1 has expired a BU+QoS message is sent via AR a2 to the MAP. The desired resources are reserved on the new path, a positive BA+QoS message is returned to the MN. The MN's rbw proc-file entry confirms that 5Kb have been successfully reserved.

Conclusion: Works

Regional HO due to unsatisfied QoS requirements in current AN

Scenario Description: MN uses AR a1 as default router (MHC: h:0, a1:1, a2:1, b1:1). The reserved bandwidth for the MN in AR a1 and the MAP is 5Kb, the avbw in the MAP is 10Kb, in AR a1 and a2 it is 5 Kb, and in AR b1 it is 10Kb. The MN's rbw = dbw = abw is also 5Kb. At time t1 the MN's dbw and abw is increased to 15Kb.

Observation: After the dbw and abw of the MN have been increased a BU+QoS message is sent to the MAP via AR a1. AR a1 set the F flag of the QoS option, released all previously reserved resources of the MN, and forwarded the message towards the MAP. The MAP then returned a negative BA+QoS message to the MN. In a second try, the MN sent a BU+QoS message via AR a2 to the MAP which is also negatively acknowledged, no QoS reservations were installed. The third BU+QoS message is sent via AR b1 and is positively acknowledged by the MAP. The desired bandwidth has been reserved on the path via AR b1 up to the MAP. When the MN has received the BA+QoS message a BU is sent to HA and CN.

Conclusion: Works

QoS request via current path (abw $>$ avbw)

Scenario Description: MN uses AR a1 as default router (MHC: h:0, a1:1, a2:0, b1:0). The avbw in AR a1 and the MAP is 5Kb, rbw = dbw = abw = 0Kb. At time t1 dbw and abw is increased to 15Kb.

Observation: BU+QoS message is sent towards the MAP and negatively returned to the MN. No resources are reserved. The MN switches to MIPv6 mode. A BU is sent to HA and CN.

Conclusion: Works

6.3 Performance Results

To demonstrate the benefits of HMIPv6 and the QoS-Conditionalized BU various performance experiments have been performed and are explained in this section. In a first step the registration latency (signaling latency) for different mobility protocols (e.g. MIPv6, HMIPv6, QoS-Conditionalized BU) was measured. In a second step the effect of the registration latency (caused by the different mobility mechanisms and network environments) on the application layer has been investigated.

6.3.1 Registration Latency

For measuring the registration latency the testbed described in Section 6.1 and Figure 6.2 was used. The WAN delay was configured to 30 ms and the AN delay to 5 ms. The scenarios were performed according to the test case description given in in Section 6.2:

- HMIPv6 local and regional handover scenarios were performed according to the test case description given for "Local handover" and "Regional handover from AN a to AN b".
- QoS-Conditionalized handover scenarios according to description for "Local HO with QoS request via new path" but with dbw $\geq$ avbw $\geq$ abw. Thereby it was assured that a QoS option is processed and modified by every node along the path to the MAP.
- MIPv6 scenarios were performed similar to the "Local handover" description but with disabled MAP discovery. Thereby the MN could not learn about the availability of a MAP and was forced to use MIPv6 mode.

The MN's sent and received messages were logged with a timestamp with the tool tcpdump. The registration latency is specified as duration between sending of the BU (to MAP in HMIPv6 and to HA in MIPv6) and reception of the BA (from MAP in case of a local HO in HMIPv6 and from HA in case of a regional handover in HMIP6 or a MIPv6 handover). The latency caused by ICMPv6 neighbor discovery for next hop address resolution of a BU or BA messages is considered as part of the registration latency. Table 6.1 summarizes the average, minimum, and maximum recorded registration latency from 10 handover for each scenario. Additionally, the total controlled transmission delay applied to the registration messages by the WAN and AN simulator is given in the column delay, the variance is given in the column s^2, and a 99 percent confidence interval is given in the column $\pm\Delta$[1].

The experimental results achieved with this setup show that the measured signaling latencies mostly depend on the delays enforced by the WAN and AN simulator. Delay caused by neighbor discovery and packet processing in the involved nodes is only of minor importance. The results also demonstrate that HMIPv6 outperforms MIPv6 in local movements and causes only a small increase of signaling latency in a global movement. Additionally, it could be shown that a local QoS-Conditionalized handover compared to a normal local HMIPv6 handover introduces only a small increase of processing delay. However, depending on the distance between AN and CN, it can be much faster than a MIPv6 handover.

6.3.2 Packet Loss During Handover

From the user's point of view, the main concern lies in the packet loss during a handover. To measure this, a tool called udp6traffic has been developed for generation

[1]Confidence interval, given as $\pm\Delta$ value to the given average value. How the confidence interval is calculated and an discussion on how its output can be interpreted is given in [7, 24].

	delay	min	max	average	s^2	$\pm\Delta$
MIPv6 HA :	70 ms	70.2 ms	70.4 ms	70.26 ms	0.007	0.09 ms
local HMIPv6 HO:	10 ms	10.3 ms	10.5 ms	10.42 ms	0.004	0.07 ms
QoS-Cond. HO:	10 ms	11.4 ms	12.2 ms	11.8 ms	0.033	0.19 ms
reg. HMIPv6 HO:	80 ms	81.1 ms	82.8 ms	81.4 ms	0.257	0.52 ms

Table 6.1: Experimental results for registration latency for different handover scenarios (WAN delay: 30ms, AN delay: 5ms).

and capturing of UDP up- and downstream traffic between two nodes. The minimum, maximum, and average packet loss of 100[2] handover measurements (meas.) for each mobility scenario is presented in Table 6.2.

The metric used for the measurements is as follows: udp6traffic was configured to send packets up- and downstream between MN and CN with an interval of 10 ms and a UDP payload size of 1000 bytes for a duration of 20 seconds. Within each 20 seconds slot one overlapping handover was forced by enabling and disabling related ports of the manageable hub. A detailed graph illustrating the packet loss for every single handover scenario is given in the Figures B.1 – B.4.[3]

The experimental results (column "meas.") are given with a theoretically expected value (column "theor.") in Table 6.2. The theoretical values are calculated based on the following minimum, maximum, or assumed average[4] components.

- Movement detection delay: Since the RtAdv period and lifetime is 1.5 seconds, the duration between physical loss of connectivity and the expire of the last RtAdv's lifetime received by the MN depends on how much earlier this RtAdv was sent before the link is disabled; this could be between 0 ms and 1500 ms.

- Function router_state() period: Because the expiration of the current AR lifetime is controlled and triggered by the function router_state(), the internal detection also depends on the period with which this function is called. This was set to 100 ms, thus resulting in a RtState delay between 0 and 100 ms.

- Controlled transmission delay caused by WAN/AN simulator:

[2] In some cases the script responsible for starting the udp6traffic tool and the execution of enabling and disabling certain ports on the manageable hub did not synchronize properly (see disadvantage of manageable hub described in Section 6.1). This resulted in multiple or null handover within some of the 20 seconds slots. These measurements have been removed with the consequence that for the MIPv6 and the local HMIPv6 handover emulation not all of the 100 handover could be used.

[3] The acronym(s) like 025_RL-RMN_30-05 given at the top of the figures indicate: a test number (025); real (R) or virtual link (X); local (L) or global/regional movement (G); real (R) or virtual (V) movement detection; the used mobility protocol MIPv6 (M), or HMIPv6 (H), or QoS-Conditionalized BU (Q); with background traffic or without background traffic (N); the artificial WAN transmission delay (30) in ms; and the artificial AN transmission delay (05) in ms.

[4] The assumed average is simply calculated as the $((minimum + maximum)/2)$ (this might not fit the average of the actual probability distribution function for this particular delay).

  - For local HMIPv6 and QoS-Conditionalized handover this is the round trip time (RTT) MN – MAP – MN: 10 ms.
  - For regional HMIPv6 this is the RTT MN–MAP–MN + MN–CN–MN: 80 ms.
  - For a MIPv6 handover this is the RTT MN–CN–MN: 70 ms.

- Extra CN delay: BUs that need to be sent to a CN (which is not the HA or MAP), are delayed by the MN in order to wait for a packet which travels to the same destination anyway. By way of this, bandwidth can be saved but can also result in additional packet loss. Since udp6traffic in this metric sends a packet every 10 ms this results in a extra delay for CNs between 0 and 10 ms.

- Delay due to ND address resolution and packet processing: According to the experimental results given in Table 6.1 this delay is something between 0.2 and 2.8 ms. However, since these values are rather small compared to the former ones, they are neglected for this calculation.

- The total number of lost packets finally depend on the sum of the time values given above and how this time range fits into the pattern of the CN's packet emission. For example, if packets are sent every 10 ms and the total above sum results in 25 ms, this should cause at least two total lost packets but might also result in three total lost packets. Regarding the metric of this experiment this effect can cause zero or one additional packet losses; in conformance to the previous components this equals to something between 0 and 10 ms measured interruption time.

	minimum		maximum		average		s^2
	meas.	theor.	meas.	theor.	meas.	theor.	meas.
MIPv6 HO:	18	7	163	169	104.4	88	1354
local HMIPv6 HO:	13	1	155	162	87.6	81.5	1222
QoS-Cond. HO:	18	1	158	162	92.1	81.5	1569
reg. HMIPv6 HO:	19	8	165	170	97.9	88.5	1653

Table 6.2: Experimental results and theoretical values for downstream packet losses (for a 10 ms emission period) or interruption time in 10 ms units for different handover scenarios. Summary of Figures B.1 – B.4 (movement detection based on RtAdv period of 1.5 seconds, WAN delay: 30 ms, AN delay: 5 ms).

The results given in Table 6.2 show that as long as the MN's movement detection relies only on the RtAdvs' lifetime and retransmission time of 1.5 seconds, no significant benefits regarding the application packet loss can be measured or expected. However, (with focus on the minimum theoretical values and the knowledge that most of the resulting interruption time can be assigned to the movement detection delay) it also shows that using HMIPv6 and QoS-Conditionalized BU in combination with an improved movement detection mechanism potentially allows much fewer packet losses compared to MIPv6.

6.3.3 Ideal Link Layer Trigger Case

The previously described experiments and its discussion regarding the application packet loss during a handover indicated: as long as the execution of the MN's registration activities relies on a movement detection algorithm that is much more time-intensive than the actual protocol-dependent registration latency, no significant differences could be expected. Therefore, finding an (even proprietary) option to by-pass this obstacle in further measurements would be interesting. Another interesting issue is how application packet loss depends on the MN's handover frequency (rate) and the transmission delay (WAN/AN delay) between the mobile and correspondent node. However, the investigation of the packet-loss – handover-rate interdependence by using the manageable hub, described in the beginning of this chapter, is not possible. In order to investigate this interdependencies despite the described difficulties, a number of assumptions and thereby legitimated modifications were made to the metric of the following measurements.

The main modification consists of using the netfilter module as a basis for a virtual switch. This concept has already been mentioned at the end of Section 6.1 and is discussed in more detail here. The virtual switch provides an entry in the proc-file system as an interface to control the treatment (drop or accept) of certain IPv6 packets from user space (More precisely: Via this entry it could be controlled which of the MN's *received* IPv6 packets are to be dropped or accepted for further processing – depending on their MAC source address. This is reasonable since only the downlink traffic – udp packets sent from CN and *received* by the MN – is considered for evaluation and also the availability of a new AR is learned by the MN through the *reception* of a new RtAdv.). Additionally, a change in this proc-file entry is used to immediately trigger the MN's mobility module to consider this "virtual" movement (More precisely: If the MAC address of a specific AR becomes marked as "to be dropped" the lifetime value of the related AR data structure is set to zero and the MN sends a router solicitation message. In the following the MN's event-handling processing could only react on such RtAdvs which MAC address is not marked as "to be dropped".).

Using this concept of a virtual switch essentially provides three advantages for the performance evaluation.

- Because the virtual switch is actually a software module in the MN's kernel space, virtual handover are executed very fast, conceptually allowing even several handover per second.

- The immediate trigger to the MN's event handling provides a proprietary possibility to by-pass the indeterministic and undesired delay caused by the formerly used movement detection mechanism. The current approach of triggering the mobility module directly from the tool executing the handover can later easily be replaced by a mechanism based on a real link layer trigger.

- The execution of a handover can be controlled by the same tool responsible for traffic generation and capturing. As a consequence, the tool can much better control the granularity of the handover frequency during a certain measurement slot. This is accomplished by extending the tool udp6traffic with the capability of writ-

ing specific values to the virtual switch's proc-file entry. The virtual switch module then interprets this value as a vector indicating the "to be dropped" and "to be accepted" packets.

Two more necessary modification have been discovered during the first experiments with the new setup.

- The Neighbor Discovery for IPv6 specification [40] states that RtAdv sent in response to a RtSol must be delayed by a random time between 0 and MAX_RA_DELAY_TIME (where MAX_RA_DELAY_TIME is defined as 500 ms). This is an issue which is also discussed in the Mobile-IP working group and related mailing lists [29] regarding its consequences for mobile networks. To avoid this potential long response time as a new reason for delayed execution of necessary registration operations, the router advertisement daemon (radvd) is modified to respond immediately to a received RtSol.

- The MIPv6 specification [25] defines a mechanism to limit the maximum rate of sending RtSol by the MN. This mechanism is disabled to allow the MN to send RtSol with a minimum interval of less than one RtSol per second depending on the current handover rate.

The experimental results illustrated in Figure 6.3 show the measured handover loss dependent on the handover frequency. The handover loss is represented in percent divided through its current handover frequency. However, for this metric it could also be regarded as the absolute packet loss per handover or as the transmission interruption time per handover in units of 10 ms. These results are given for the mobility scenarios and protocols of a MIPv6, local and regional HMIPv6, and QoS-Conditionalized handover.

For this set of measurements the WAN / AN delay was set to 30 ms / 5 ms. Unlike the previous conducted handover measurements, the handover performed here were non-overlapping handover, also referred to as hard handover. For a handover in this setup it means that at the very moment when the link from the old AR is virtually disabled the link from the new one is enabled. Each illustrated point in the graph represents the average packet loss, measured during one 100-seconds slot. One complete measurement lasts for 25 slots where the handover rate has been gradually increased by 5 handover per slot, starting with 5 handover per slot (0.05 HO / sec) to finally 125 handover per slot (1.25 HO / sec). The unit [number of HO per sec] is also referred to as handover rate or handover frequency (HOf).

This measurement has been repeated for various WAN and AN delays. The Figures B.5 to B.11, given in the appendix, illustrate the corresponding results in the same format as used in Figure 6.3. The applied WAN delays between 10 and 40 ms have been selected with respect to typical delays known from commercial IP backbones like BT, WorldCom, and AT&T [41, 44, 1]. The controlled delay of 5 and 10 ms used for the AN was chosen to adapt to values used in other performance studies like [35].

Table 6.3 provides additional significant values of the conducted measurements like the minimum, average, and maximum packet loss per handover, the calculated variance,

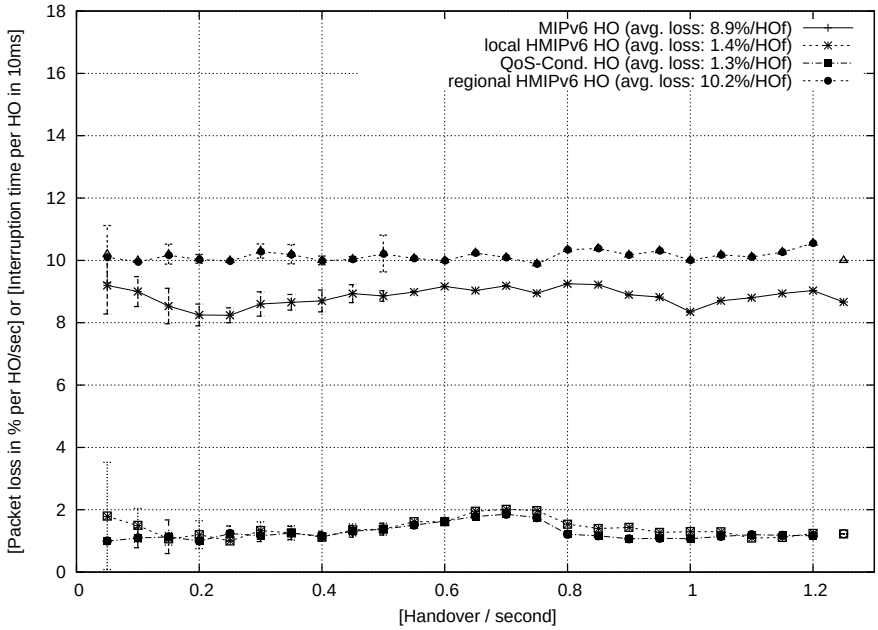


Figure 6.3: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 30 ms, AN delay: 5 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

and the 99 percent confidence interval[5] for the average packet loss. All presented values are based on the observations made during the first 275 handover of the total 1625 handover, meaning only the handover rates between 0.05 and 0.5 HO/s are considered for this statistical analysis. Detailed analysis for handover rates higher than 0.5 HO/s were not possible because the resolution of the output files produced by udp6traffic made it impossible to distinguish to which of two successive handover a certain lost packet belongs. Regarding the calculation of the confidence interval, a normal distribution of observed packet losses per handover was assumed.[6] This is also the reason why confidence intervals in the Figure 6.3 and Figures B.5 – B.11 are given only for handover rates up to 0.5 HO/s.

Figure 6.4 and 6.5 provide a summary of all measurements conducted with the virtual switch. The graphs now represent the total average packet loss dependent on the transmission delay between the MAP of the visited AN and the CN.

Figure 6.6 eventually shows the reduction of the packet loss of different mobility schemes over MIPv6 dependent on the WAN delay and for a fixed AN delay of 5 ms. The reduction is indicated as the difference between a given scheme and MIPv6; this difference is scaled with respect to MIPv6 (formally: $(1-(X/MIP))*100$ in percent).

[5]The confidence interval is given as $\pm\Delta$ value to the given average value. How the confidence interval is calculated and an discussion on how its output can be interpreted is given in [7, 24].

[6]The attempt to back up this assumption by finding binomial distribution functions for comparation with the numerical distributions given in Table 6.3 (by using the χ^2- test) did not succeed.

scenario	min.	max.	avg.	var.	$\pm\Delta$	o_{-2}	o_{-1}	o_0	o_{+1}	o_{+2}	o_{+3}	o_{+4}
MIP 10/5	4	9	5.05	0.19	0.07	0	11	243	18	2	0	1
MIP 20/5	6	9	7.11	0.15	0.06	0	6	235	33	1	0	0
MIP 30/5	7	12	8.69	0.45	0.11	0	1	110	142	19	2	1
MIP 40/5	10	13	10.89	0.43	0.10	0	0	68	176	24	7	0
MIP 10/10	5	9	6.16	0.27	0.08	0	11	216	42	5	1	0
MIP 20/10	7	11	8.11	0.41	0.10	0	34	185	49	6	1	0
MIP 30/10	9	38	10.53	4.12	0.32	0	48	134	90	3	0	0
MIP 40/10	11	25	11.93	1.07	0.16	0	0	72	170	26	7	0
l. HMIP 10/5	1	3	1.30	0.23	0.07	0	0	194	9	2	0	0
l. HMIP 20/5	0	8	1.68	0.44	0.10	0	1	99	170	2	2	0
l. HMIP 30/5	0	4	1.27	0.24	0.08	0	1	204	67	2	1	0
l. HMIP 40/5	1	12	1.27	0.61	0.12	0	0	214	58	2	0	0
l. HMIP 10/10	1	8	2.32	0.67	0.13	0	45	102	126	1	0	0
l. HMIP 20/10	1	13	3.23	2.38	0.24	8	95	100	16	19	34	0
l. HMIP 30/10	2	12	2.35	0.56	0.12	0	0	188	86	9	0	0
l. HMIP 40/10	2	4	2.54	0.26	0.08	0	0	128	145	2	9	0
QoS-C. 10/5	0	4	1.65	0.33	0.09	0	2	101	164	6	2	0
QoS-C. 20/5	9	3	1.56	0.33	0.09	0	4	120	143	8	0	0
QoS-C. 30/5	1	3	1.23	0.19	0.07	0	0	213	60	2	0	0
QoS-C. 40/5	1	3	1.20	0.17	0.06	0	0	221	53	1	0	0
QoS-C. 10/10	2	5	2.39	0.26	0.08	0	0	171	103	0	1	0
QoS-C. 20/10	1	4	2.70	0.36	0.09	0	1	99	156	19	0	0
QoS-C. 30/10	1	4	2.62	0.33	0.09	0	4	105	157	9	0	0
QoS-C. 40/10	1	7	3.09	1.52	0.19	2	94	126	6	25	20	2
r. HMIP 10/5	5	8	6.10	0.43	0.10	0	37	183	46	9	0	0
r. HMIP 20/5	7	11	8.11	0.19	0.07	0	4	244	22	3	2	0
r. HMIP 30/5	9	21	10.12	0.56	0.12	0	2	252	18	2	1	0
r. HMIP 40/5	11	14	11.89	0.46	0.11	0	0	76	157	38	4	0
r. HMIP 10/10	7	12	8.53	0.71	0.13	0	4	161	82	20	3	5
r. HMIP 20/10	9	16	10.57	1.82	0.21	0	54	107	63	22	17	9
r. HMIP 30/10	11	19	11.98	0.77	0.14	0	0	78	143	45	8	1
r. HMIP 40/10	13	35	14.28	1.76	0.21	0	1	220	53	0	1	0

Table 6.3: Additional significant values of experimental results given in the Figure 6.3 and Figures B.5 – B.11 (Only handover rates up to 0.5 HO/s were considered for evaluation of presented values).

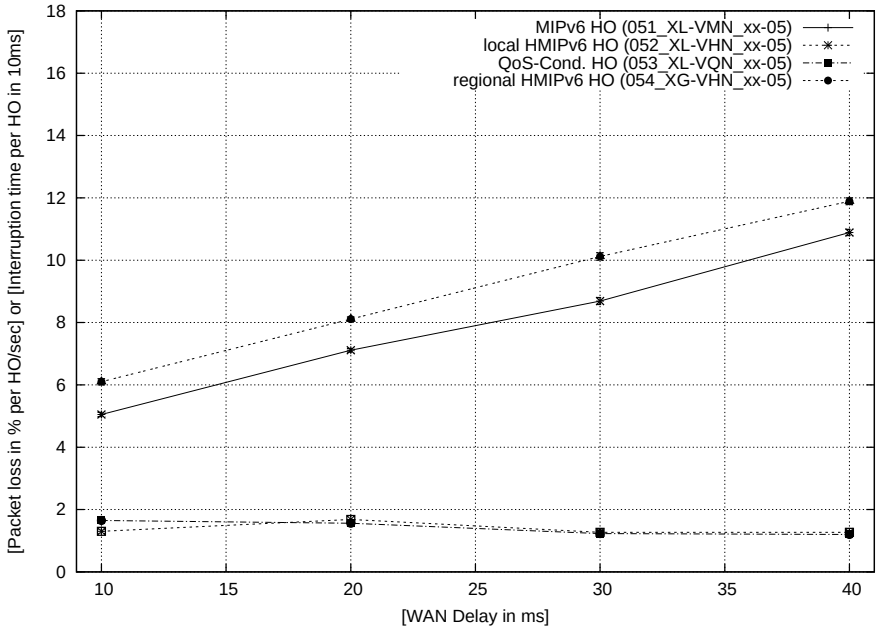


Figure 6.4: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Conditionalized handover scenarios over WAN delay (AN delay: 5 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

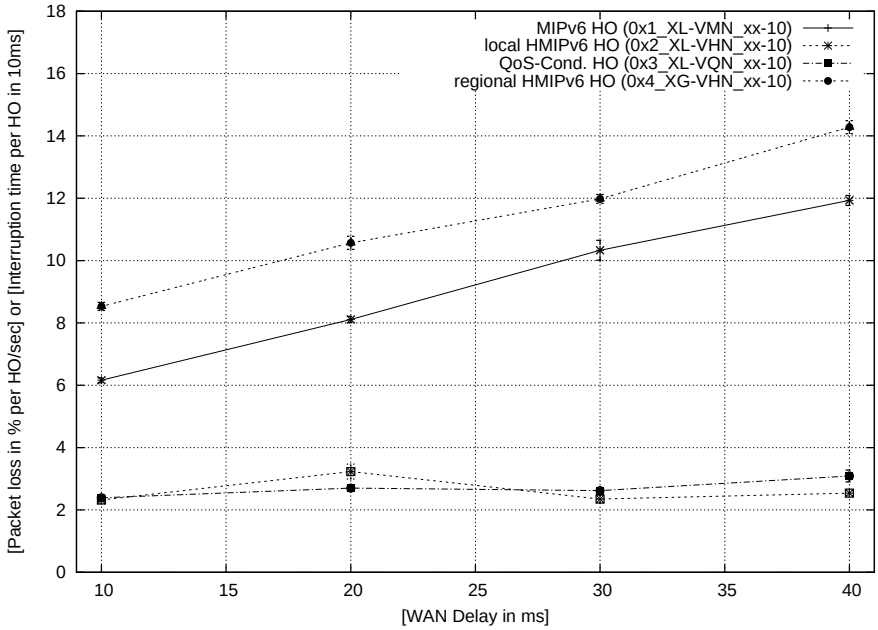


Figure 6.5: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Conditionalized handover scenarios over WAN delay (AN delay: 10 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

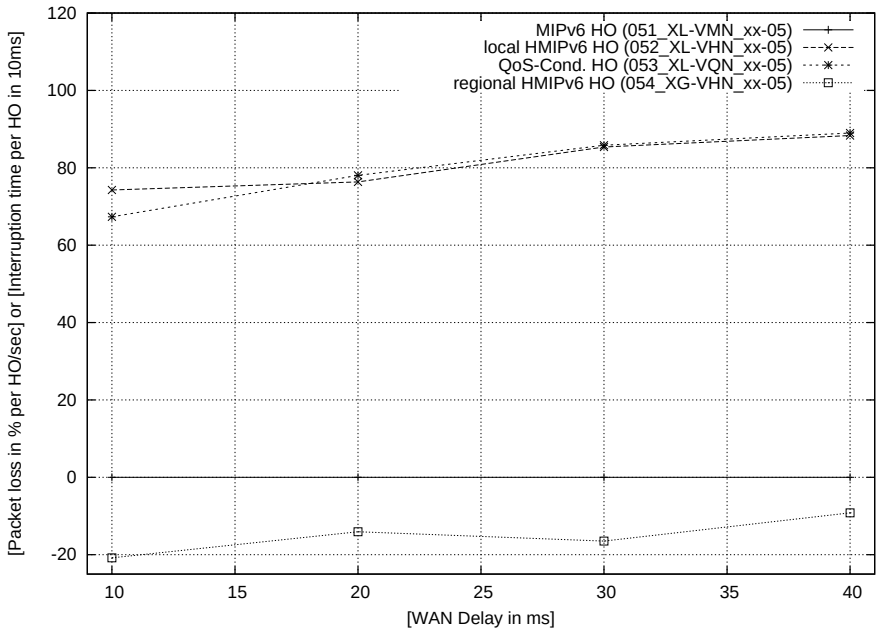


Figure 6.6: Experimental results for reduction of downstream packet loss of local and regional HMIPv6 and QoS-Conditionalized handover over MIPv6 handover dependent on the WAN delay. (AN delay: 5 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

On closer examination of the results accomplished with the virtual switch it could be noted that some of the measurements to a certain degree do not match their theoretical expectation. This is even the case when taking related confidence intervals into account. For example when comparing the average packet loss caused by a local HMIPv6 and a QoS-Conditionalized handover in Figure 6.5. One would expect (based on the theoretical consideration listed in Section 6.3.2 and the measured registration latencies given in Table 6.1) that a local HMIPv6 handover compared with a QoS-Conditionalized handover causes the same or less packet losses. However the corresponding average losses measured for 20 and 40 ms indicate even contradicting results.

One possible explanation for this result is that the number of performed measurements could not compensate potential existing systematic mistakes. For example, many of a Linux-i386 machine's process scheduling is triggered by the internal clock (so called jiffies). This clock is a value periodically increased by one with a 10 ms interval. Some of the known processing influenced by the jiffies pattern and relevant for the measured packet loss is: the scheduling of a to be sent UDP packet (MN's BUs to a CN are piggybacked by such packets and UDP packets sent by the CN to the MN are considered for evaluation) and the scheduling of a movement (via the virtual switch) that has to be executed. The consequences of such patterns for packet processing on several machines and especially on their interaction is not known and might be a reason for a systematic mistake that could not be represented by the calculated confidence interval. If, as a consequence of this knowledge, the interpretation of the results given above hap-

pens using a confidence interval extended by only 10 ms (or 1 percent), it would turn all unsatisfied expectations into reasonable results but would still provide valuable information. In addition, while these small differences can be regarded as being statistically significant, their practical relevance is rather questionable. The impact of small modifications (e.g. in implementations) is likely to have a much larger impact than differences in the underlying approaches themselves.

6.4 Summary

In this chapter the realized QoCoo implementation has been successfully tested in an experimental testbed. Additionally, a number of performance experiments have been conducted to investigate the advantages and disadvantages (regarding the registration latency and the packet loss during a handover) of HMIPv6 [38] and the QoS-Conditionalized BU [17] over MIPv6 [25].

The observation of the various test cases showed that the QoS-Conditionalized BU in combination with HMIPv6 provides a MN with micro mobility support and with the capability to negotiate a level of QoS provisioning with the routers along the new data path in the AN. Thereby, the MN can conditionalize a handover on a minimum (acceptable) level of QoS requirements. Moreover, the QoCoo implementation manages the release of reserved resources on a stale data path (a path to which the MN has lost connectivity) before the reservation's actual lifetime elapses. QoS (re-)establishment for micro movements is managed locally and transparent to the CN with the consequence that the number of registration messages for micro movements is reduced from several BUs to (potentially distant) HA and CNs to only one BU to the relative close MAP.

The performance experiments showed that the registration latency caused by a local HMIPv6 and QoS-Conditionalized handover is much smaller than that caused by MIPv6. On the other hand, HMIPv6 causes only a small increase of registration latency for regional movements.

Regarding the application packet loss, it could be concluded that an improved movement detection mechanism (improved over the simple reliance on IPv6 RtAdvs) is obligatory to make use of the enhanced registration latency. By simulating an ideal link layer trigger, it has been demonstrated that the downlink packet loss for micro movements could be reduced by over 80 percent per handover (depending on the transmission delay between AN and CN) when using a protocol based on HMIPv6 instead of MIPv6. This is even the case when performing a QoS-Conditionalized handover. Measurements, comparing a macro movement between HMIPv6 and MIPv6, resulted in an increase of the packet loss per handover between 20 and 8 percent using the hierarchical approach.

Chapter 7

Conclusion

In this work, a prototypical implementation and experimental testbed setup of a QoS-enabled mobility concept based on HMIPv6 [38] and the QoS-Conditionalized BU [17] has been accomplished. To achieve this, the following intermediate steps have been conducted. The leveraged architectures such as IPv6 and MIPv6 have been reviewed and relevant procedures for this work have been summarized.

The underlying concepts – descriptional specifications of HMIPv6 and QoS-Conditionalized BU – have been analyzed. Their behavior has been re-examined in a more formal way to provide a basis for the implementation design. Open issues and possible solutions according to the protocols' specifications have been identified. Especially the composition of a QoS option needed to be reorganized and the possibilities to release reserved resources on a stale data path has been neglected in the specification and required additional investigation.

The implementation design has been adapted to the selected operating system and existing MIPv6 implementation environment. Thereby, it was discovered that especially the proper management of movement detection, selection of a suitable new default router, and the execution of related registration operations proved to be a difficult task. A new method, based on a so-called "context field" (essentially summarizing the overall context of a path, represented by an AR, in a single value), has been designed to overcome this challenge. This selection method can easily be extended to consider new information and different movement policies.

Based on the implementation design, a prototype implementation called QoCoo has been realized to provide support for HMIPv6 and the QoS-Conditionalized BU. QoCoo has been setup in an experimental testbed. Various representative scenarios have been successfully tested, with the objective to demonstrate the key capabilities of the underlying protocols. Performance experiments showed that the packet loss per handover for micro movements can be reduced by more than 80 percent (depending on the transmission delay between access network and CN) when using HMIPv6 or the QoS-Conditionalized BU instead of MIPv6. However, it was also shown that an enhanced movement detection mechanism is necessary to let the application layer benefit from the reduced signaling latency introduced by the use of HMIPv6.

Nevertheless, a number of open issues does still exist. In particular, as a local QoS signaling protocol, QoS-Conditionalized BU needs to interact with end-to-end QoS sig-

naling solution, while the latter is currently still under investigation, e.g., in the IETF Next Steps in Signaling (NSIS) working group. Additional challenges emerge when considering QoS support for multiple, independent flows or when allowing some of these flows traversing via different MAPs, which might also lead to asymmetric paths, in the access network.

Finally, it can be concluded that the QoCoo implementation presented here provides a flexible, extensible and easy-to-configure prototype for testing and evaluating of the HMIPv6 and the QoS-Conditionalized BU protocol. Moreover, the deployment of these protocols can provide significant performance advantages over MIPv6 even then when QoS provisioning is desired.

Bibliography

[1] AT&T. Backbone Dealy and Loss, September 2002. http://ipnetwork.bgtmo.ip.att.net/delay_and_loss.shtml.

[2] M. Beck, H. Böhme, M. Dziadzka, U. Kunitz, R. Magnus, C. Schröter, and D. Verworner. *Linux Kernelprogrammierung, Algorithmen und Strukturen der Version 2.4*. Addison-Wesley, 2001.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998.

[4] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, October 1989.

[5] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, September 1994.

[6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification . RFC 2205, September 1997.

[7] I. N. Bronstein, K. A. Semendjajew, G. Grosche, V. Ziegler, and D. Ziegler. *Teubner Taschenbuch der Mathematik*. B.G. Teubner Stuttgart - Leipzig, 1986.

[8] Mark Carson. Application and Protocol Testing through Network Emulation, September 1997. http://is2.antd.nist.gov/itg/nistnet/.

[9] Claude Castelluccia. Toward a Hierarchical Mobile IPv6. In *HPN*, pages 275–290, 1998.

[10] H. Chaskar. Requirements of a QoS Solution for Mobile IP. Internet draft, work in progress, February 2002.

[11] H. Chaskar and R. Koodli. QoS Support in Mobile IP version 6. In *IEEE Broadband Wireless Summit (Networld+Interop)*, May 2001.

[12] A. Conta and S. Deering. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6). RFC 2463, December 1998.

[13] S. Deering. ICMP Router Discovery Messages. RFC 1256, September 1991.

[14] G. Fankhauser, S. Hadjiefthymiades, N. Nikaein, and L. Stacey. Interworking RSVP and Mobile IP Version 6 in Wireless Environments. ACTS Mobile Communications Summit, 1999.

[15] Bison. Free Software Foundation, 1998. http://www.gnu.org/software/bison/bison.html.

[16] Flex. Free Software Foundation, 1999. http://www.gnu.org/software/flex/.

[17] X. Fu, H. Karl, A. Festag, G. Schaefer, C. Fan, C. Kappler, and M. Schramm. QoS-Conditionalized Binding Update in Mobile IPv6. Internet draft, work in progress, January 2002.

[18] X. Fu, H. Karl, and C. Kappler. Qos-conditionalized handoff for mobile ipv6. In *Proc. of the Second IFIP-TC6 Networking Conf. - Networking2002*, Pisa, Italy, May 2002.

[19] A. G. Gleditsch and P. K. Gjermshus. The Linux Cross-Reference, August 2001. http://lxr.sourceforge.net/.

[20] GO/Core project at HUT Telecommunication and Multimedia Lab. Mobile IP for Linux (MIPL) mipv6-0.9-v2.4.7. http://www.mipl.mediapoli.com/.

[21] R. Hinden and S. Deering. Internet Protocol, Version 6 (IPv6). RFC 2460, December 1998.

[22] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 2373, July 1998.

[23] IEEE. Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority. IEEE Standards Tutorials. http://standards.ieee.org/regauth/oui/tutorials/EUI64.html.

[24] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[25] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. Internet-Draft, work in progress, May 2002.

[26] Sami Kivisaari. MIPL Technical Specification Tik-76.115, March 2000. url:http://www.soberit.hut.fi/tik-76.115/99-00/palautukset/groups/MIPL/su/palautus3.html.

[27] The Linux Documentation Project, 2002. http://www.tldp.org/docs.html.

[28] Alberto Lopez, Jukka Manner, Andrej Mihailovic, Hector Velayos, Eleanor Hepworth, and Youssef Khouaja. A study on qos provision for IP-based radio access networks. In *IWDC*, pages 139–157, 2001.

[29] Mobile-IP. IETF Mailing list, 2002. http://playground.sun.com/mobile-ip/.

[30] C. Perkins. IP Mobility Support for IPv4. RFC 3344, August 2002.

[31] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.

[32] J. Reynolds. Assigned numbers, on-line database. RFC 3232, January 2002.

[33] Pedro Roque and Lars Fenneberg. RADVD – Router Advertisement Daemon, November 2001. http://v6web.litech.org/radvd/.

[34] Rusty Russell. Linux netfilter Hacking HOWTO, 2001. http://www.netfilter.org/unreliable-guides/netfilter-hacking-HOWTO/.

[35] Q. Shen, A. Lo, and W. Seah. Performance Evaluation of Flow Transparent Mobile IPv6 and RSVP Integration. In *Proc. SCI/ISAS 2001, Vol. IV*, volume 4, pages 515–520, July 2001.

[36] Q. Shen, A. Lo, W. Seah, and C.C. Ko. On Providing Flow Transparent Mobility Support for IPv6-based Real-Time Services. In *Proc. MoMuC 2000*, pages 2B–4–1 – 4, Tokyo, Japan, October 2000.

[37] Q. Shen, W. Seah, and A. Lo. Flow Transparent Mobility and QoS Support for IPv6-based Wireless Real-time Services. Internet draft, work in progress, February 2001.

[38] H. Soliman, C. Castelluccia, K. El-Malki, and L. Bellier. Hierarchical MIPv6 mobility management (HMIPv6). Internet-Draft, work in progress, July 2002.

[39] W. Stevens and M. Thomas. Advanced Sockets API for IPv6. RFC 2292, February 1998.

[40] E T. Narten, Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461, December 1998.

[41] British Telecom. IP Performace, September 2002. http://ippm.ignite.net/.

[42] S. Thomson and T. Narten. IPv6 Stateless Address Autoconfiguration. RFC 2462, December 1998.

[43] Harald Welte. SKB - Linux Network Buffers, October 2000. http://www.gnumonks.org/ftp/pub/doc/skb-doc.html.

[44] Worldcom. Latecy statistics, September 2002. http://www.worldcom.com/global/about/network/latency/.

[45] J. Wroclawski. The Use of RSVP with IETF Integrated Services, RFC 2210. RFC 2210, September 1997.

Appendix A

Function and Data Structure Reference

A.1 HMIPv6 Related Functions and Data Structures

Functions and Data Structures in RAdvD Code (MAP and IR Related)

radvd

Type (declaration): struct **Interface**		(extended)
Data Fields:		
struct Interface *	next	Pointer to next element in interface list.
Router discovery related parameters:		
char	Name[ ]	Name of interface (e.g. eth0).
struct in6_addr	if_addr	IPv6 link local address of that interface
...		
Mobile IPv6 extensions:		
struct AdvPrefix *	AdvPrefixList	Pointer to list containing valid prefixes to be advertised for that link.
struct timer_lst	tm	Data structure to be linked in the timer list for scheduling of next RA.
unsigned long	last_multicast	Timestamp when last multicast RA was send.
...		
Hierarchical Mobile IPv6 extensions (MAP discovery, see [38]):		
int	MaxRADelTime	ReDefines MAX_RA_DELAY_TIME in ms, default is 500, see [40]. This is only useful for faster Movement Detection and violates rfc 2461.

continues on next page

continued from previous page

int	MapOptRecv	Defines whether MAP options received on this interface should be recorded (in other_map_list) for further propagation.
int	MapOptProp	Defines whether recorded MAP options (in other_map_list) should be further propagated via this interface.
struct MapAddr *	MapAddrList	Pointer to first element of list containing configured MAP discovery parameters for this interface.

Description:
Data structure to contain configured parameters for router discovery on specific interface. Extended to hold also parameters for configuration of MAP discovery for this interface.

Type (declaration): (new)
struct **MapAddr**

Data Fields:

struct MapAddr *	next	Pointer to next element of list.
struct in6_addr	Prefix	Global IP-Address of MAP and Prefix/64 to be used by MNs to allocate RCoA.
int	MapOptAdv	Advertise this MAP-Opt, only valid if MapOptProp is 1
unsigned int	MapOptPref	Preference value of MAP option. Only the highest order 4 bits will be used.
int	MapOptFlagR	Indicates Basic Mode (1), MapOptFlagR != MapOptFlagM, default: 1.
int	MapOptFlagM	Indicates Ext. Mode (1), MapOptFlagM != MapOptFlagR, default: 0.
int	MapOptFlagI	Indicates whether RCoA may be used as source address, default: 1.
int	MapOptFlagT	Indicates potential topological incorrect address, default: 0.
int	MapOptFlagP	Indicates whether RCoA must be used as source, default: 1.
int	MapOptFlagV	Indicates whether reverse tunneling is allowed, default: 0.
uint32_t	MapOptLifetime	Indicates lifetime in seconds of MAP.

Description:
Data structure to contain parameters for configuration of MAP discovery.

Type (declaration): (new)
struct **nd_opt_map_info**

Data Fields:

continues on next page

continued from previous page

uint8_t	nd_opt_mi_type	Type field in MAP option.
uint8_t	nd_opt_mi_len	Length field in MAP option.
uint8_t	nd_opt_mi_dist_pref	Distance and Pref field (each 4-bit) are merged to one 8-bit field.
uint8_t	nd_opt_mi_flags_res	Flags and reserved fields in MAP option merged to one 8-bit field.
uint32_t	nd_opt_mi_valid_time	Valid lifetime field of MAP option.
struct in6_addr	nd_opt_mi_mapaddr	MAP address field of MAP option.

Description:
Structure to exactly conform to the defined alignment of a MAP option as specified in [38] and illustrated in Figure 3.2.

Type (declaration): (new)
struct **other_map**

Data Fields:

struct nd_opt_map_info	map_opt	Structure to hold the received MAP option.
unsigned long	arrival	Timestamp in seconds of arrival.
struct other_map *	next	Pointer to next element of other_map list.

Description:
List element of other_map_list to keep records of MAP options received from other MAPs.

Type (definition): (extended)
struct Interface * **IfaceList**

Description:
Root pointer to the first element of interface list. This is where all configured parameters for router discovery on the particular interfaces are maintained. Is extended to maintain also parameters for its own MAP discovery if this is configured. See also radvd.h: struct Interface.

Type (definition): (new)
struct other_map * **other_map_list**

Description:
Root pointer to first element of received MAP option list. This list maintains one entry for every available MAP as learned via received MAP options. It does not maintain the own MAP parameters. See also radvd.h: struct other_map.

process

Function: (modified)
void **process(** int sock, struct Interface *ifacel, unsigned char *msg, int len, struct sockaddr_in6 *addr, struct in6_pktinfo *pkt_info, int hoplimit, struct other_map **other_map_LP **)**
Description:
Function to process incoming ICMPv6-RS or ICMPv6-RA messages. Performs some preliminary checks on message and finally calls process_rs or process_ra respectively for further processing.

Function: (modified)
static void **process_rs(** int sock, struct Interface *iface, struct sockaddr_in6 *addr, struct other_map **other_map_LP **)**
Description:
Function for final processing of incoming ICMPv6-RS. Checks whether to respond to multicast or to unicast source address of RS. Eventually calls send_ra for advertising requested RA on the given interface and to the selected address.

Function: (extended)
static void **process_ra(** struct Interface *iface, unsigned char *msg, int len, struct sockaddr_in6 *addr, struct other_map **other_map_LP **)**
Description:
Function for final processing of incoming ICMPv6-RA. Explicitly checks the link parameters of received RA with own configured parameters for link. Extended with capability to recognize MAP options in RA and update or refresh other_map_list with new information.

send

Function: (extended)
void **send_ra(** int sock, struct Interface *iface, struct in6_addr *dest, struct other_map **other_map_LP **)**
Description:
Assembles and sends RA based on given arguments and via socket sock. The common RA part is assembled by use of iface and dest parameters. MAP optios are only assembled and send if MapOptProp value of interface is thru. The own MAP options are generated based on the MapAddrList of the interface. Received MAP options are assembled based on the other_map_list. Its new lifetime is scheduled via the difference of passed time since RA was received (now - timestamp in other_map_list element) and lifetime field of received RA.
continues on next page

continued from previous page

timer

Function: (new)
unsigned long **jiftime(** void **)**
Description:
Function to obtain system time. Returns system time in seconds. This is used to time stamp received MAP options in order to schedule the new lifetime when later propagating it.

interface

Function: (new)
void **map_init_defaults(** struct MapAddr *map **)**
Description:
Function to set default values (as given in radvd.h: MapAddr) in the MapAddr structure pointed to by map.

Functions and Data Structures in MIPL Code (MN Related)

mipv6

Type (declaration): (new)		
struct **nd_opt_map_info**		
Data Fields:		
__u8	type	Type field in MAP option.
__u8	len	Length field in MAP option.
__u8	dist_pref	Distance and Pref field (each 4-bit) are merged to one 8-bit field.
__u8	flags_res	Flags and reserved fields in MAP option merged to one 8-bit field.
__u32	valid_time	Valid lifetime field of MAP option.
struct in6_addr	mapaddr	MAP address field of MAP option.
Description:		
This structure wraps the MAP option that is carried in RtAdv ICMPv6 ND messages to perform MAP discovery. It has the correct alignement as specified in HMIPv6.		

continues on next page

continued from previous page

Function: (extended)

struct ipv6_txoptions * **mipv6_modify_xmit_packets (** struct sock *sk, struct sk_buff *skb, struct ipv6_txoptions *opts, struct flowi *fl, struct dst_entry **dst, void *allocptrs[] **)**

Description:

Function to modify outgoing packets by way of appending required extension headers. This could be a routing header, home address destination option or binding update destination option.

Function: (new)

static unsigned int **modify_xmit_addrs (** unsigned int hooknum, struct sk_buff **skb, const struct net_device *in, const struct net_device *out, int (*okfn) (struct sk_buff *) **)**

Description:

This Function is called from netfilter NF_IP6_LOCAL_OUT hook to finally put the correct address in the IPv6 header's source address field of a packet sent by the MN. It must be the CoA (LCoA) in MIPv6, the RCoA in HMIPv6 mode when sending a message to a CN or HA and the LCoA when sending a packet to the MAP.

Function: (new)

void **h_assign_if_rcoa (** struct anchor *nan, struct net_device *currdev **)**

Description:

This function assigns the given RCoA IPv6 address in the anchor structure to the given interface (net_device).

Function: (new)

void **h_do_map (** struct router *nrt, struct anchor *nan, struct nd_opt_map_info *map_opt **)**

Description:

This function is called when a MAP option (MAO) is received. It stores MAP relevant informations from the MAO in map_opt in the structures nrt and nan. Figures out the RCoA that can be used with this MAP and assigns it to the MN's interface.

Function: (extended)

continues on next page

continued from previous page

int **mipv6_ra_rcv_ptr (** struct sk_buff *skb, struct icmp6hdr *msg **)**

Description:
Processes incoming RtAdv and conveyed sub options. If node is MN it calls function h_router_event for further processing (See also Figure 5.2).

mdetect

Type (declaration): (new)
struct **bu_reply**

Data Fields:

int	type	Type of the reply to a BU or QoS message, could be of: BU_BACK , BU_BNACK , BU_TIMEOUT , QOS_ACK
struct in6_addr	raddr	Address of replying node, in general HA or MAP. Also contains prefix
struct in6_addr	coa	CoA the MN registered with this CN
unsigned long	now	Timestamp when reply was received in jiffies
unsigned long	lifetime	Lifetime of reply, Calculated by: (bulentry->expire - now)/HZ
unsigned long	refresh	Refresh time of binding, calculated by: 4*lifetime/5
struct in6_addr	ho_addr	Home address of MN taken from QoS reply: hmipv6_hopopt_qosrq structure
struct in6_addr	r_coa	RCoA of MN, taken from QoS reply: hmipv6_hopopt_qosrq structure
struct in6_addr	cn_addr	CN address of MN taken from QoS reply: hmipv6_hopopt_qosrq structure
__u8	resr_FDA_flags	FDA-flags taken from QoS reply: hmipv6_hopopt_qosrq structure
unsigned long	resr_fl	Flow label taken from QoS reply: hmipv6_hopopt_qosrq structure
unsigned int	resr_dbw	Desired bandwidth taken from QoS reply: hmipv6_hopopt_qosrq structure
unsigned int	resr_abw	Acceptable bandwidth taken from QoS reply: hmipv6_hopopt_qosrq structure
struct bu_reply *	next_reply	Pointer to next bu_reply record if more that one QoS option is contained in a reply message

continues on next page

continued from previous page

Description:
Data structure to handle the reply event to a previously send BU(+QoS) message. This could represent a positive or negative BA, a QoS option, or a timeout indicating the loss of a message.

Type (declaration): (new)
struct **anchor**

Data Fields:

struct in6_addr	raddr	Global address of the MAP.
struct in6_addr	r_coa	RCoA the MN has registered with this MAP.
struct in6_addr	lr_l_coa	Last registered LCoA the MN has registered with this MAP.
__u32 map_lifet	map_lifet	Lifetime of this MAP in seconds at the time the related MAP option was received. A timestamp of that event is in the related AR data structure that points to this anchor (MAP) data structure.
struct anchor *	next_map	Pointer to the next element in the MAP data structure or NULL.

Description:
Element of the MAP data structure used in the MN to maintain known MAPs and related information.

Type (declaration): (extended)
struct **router**

Data Fields:

struct router *	prev_router	Pointer to the data structure of the AR previously used as default router.
struct in6_addr	ll_addr	Link local address of the AR.
struct in6_addr	raddr	Address prefix or global address of the AR.
__u8 [MAX_ADDR_LEN]	link_addr	Link layer address of the AR.
__u8	link_addr_len	Link layer address len.
int	ifindex	Interface index of the MN's interface on which the related RA was received.
struct router *	next	Pointer to next element (router structure) in the AR data structure.
int	pfix_len	Prefix length of the link served by this AR.
int	state	State of the router, not used anymore.
unsigned long	lifetime	Lifetime value taken from the last received RA.

continues on next page

continued from previous page

__u32	last_ra_rcvd	Timestamp when last RA was received in jiffies.
__u32	interval	Interval value taken from the last received RA.
int	glob_addr	Indicates whether the address stored in raddr is also the global address of the AR.
__u8	flags	flags set in the received RA.
HMIPv6 related extensions:		
struct in6_addr	CoA	CoA used with this router. If the MN is in HMIPv6 mode this field contains the MN's RCoA.
__u8 [ETH_ALEN]	mac_addr	Source MAC address of received RA.
__u32	last_mac_rcvd	Timestamp of last received packet that contained the above MAC address as source address.
struct in6_addr	l_coa	LCoA the MN uses with this AR.
__u32	last_map_rcvd	Timestamp of last received MAP option contained in a RA of this AR.
__u32	map_ack_expiery	MAP binding expiration time in jiffies registered via this AR.
__u32	map_ack_refresh	MAP binding refresh time in jiffies for registration via this AR.
int	map_unacked_bu_send	Number of unacked BU send to MAP via this AR.
__u32	last_map_bu_send	Timestamp of last send BU to MAP via this AR.
unsigned	long context	Context field of this AR (See also Figure 5.3).
__u32	bad_path_expiery	This is set to jiffies + BAD_PATH_LIFETIME when map_unacked_bu_send has reached MAX_MAP_UNA_BU_SEND. It results in the denial of the well behaved flag of the AR's context field.
__u32	last_map_nack_rcvd	Set to jiffies when NBACK received or when bad_path_expiery is set because path to MAP caused trouble. This field is used to let the well behaved property of a path elapse after some time.

continues on next page

continued from previous page

QoS-Conditionalized BU related extensions:

struct qos_flow	pfl1	Structure used to store reserved and requested resources on the path represented by this AR. (MN's RDS)

Description:
Element of the AR data structure used in the MN to maintain known AR and information related to the data path represented by that AR. See also Figure 5.1. Regarding QoS support the data structure is extended to maintain also the reserved resources on the path up to the MAP.

Type (definition): (original)
static struct router * **curr_router**

Description:
Pointer to the data structure of the currently used AR regarded as default router (See also Figure 5.1 and 5.4).

Type (definition): (original)
static struct router * **next_router**

Description:
Pointer to the data structure of the AR selected as handover target (See also Figure 5.1 and 5.4).

Type (definition): (new)
static struct router * **tent_router**

Description:
Pointer to the data structure of the AR selected as HMIPv6 handover target. It directly indicates the tentative next AR and indirectly the tentative next MAP used for required registration messages (See also Figure 5.1 and 5.4).

Type (definition): (new)
static struct anchor * **curr_map**

Description:
Pointer to data structure of the currently used MAP (See also Figure 5.1 and 5.4). If this pointer is NULL the MN is in MIPv6 mode.

Function: (extended)
static int **h_handle_prefix (** struct router *old, struct router *new **)**

Description:
Called by mipv6_change_router (See also Figure 5.4 to initiate sending of BUs to HA and CNs but also sending of binding releases to old MAPs. Decides what to do based on the new prefix. Keeps track of CoAs and deletes the old CoA. Also notifies mn of movement. Finally function h_mn_moved() is called to coordinate the sending of related messages.)

continues on next page

continued from previous page

Function: (new)
int **h_make_up_add_rt (** struct router *tmp_nrt **)**
Description:
Function to add new router as default router to the routing table and purge the routing table from old routers and entries related to the old link.

Function: (extended)
void **mipv6_change_router (** void **)**
Description:
Function to initiate execution of necessary registration operations for selected AR (as identified by the pointer next_router and the routers context field). See also Figure 5.2 and 5.4 and the description given in 5.1.1.

Function: (new)
static void **h_router_state (** unsigned long foo **)**
Description:
Function periodically called (The period with which it is called can be dynamically adjusted with the proc-file-entry: /proc/sys/net/ipv6/mobility/mn_rst_p. Default is 10 == 100ms.) to check for: - The lifetime or interval value of the current router has expired. Then h_router_event() is called for further processing. - hmipv6_mn_mac_port to simulate movement has changed (This is a feature to trigger a movement from user space). See also Figure 5.2. Regarding QoS support: extended to also check whether MN's resource requirements for the current path have changed or reserved resources are about to expire.
continues on next page

continued from previous page

Function: (new)
int **h_get_tent_lcoa (** struct in6_addr *saddr, struct in6_addr *daddr, struct in6_addr *l_coa **)**

Description:
Function called from modify_xmit_addrs() (from NF_HOOK) to exchange the source address of a packet send by a MN with :
a) LCoA of curr_router when sending a refresh to MAP.
b) Tentative LCoA (of tent_router) when sending a "tentative" BU to new MAP in case of HMIPv6 handover. In this case the old AR is still assumed as the MNs default router. Therefore a routing header is used to assure that a BU packet travels via the new path.

Function: (new)
void **h_assign_ar_lcoa (** struct router *curr **)**

Description:
Generates LCoA based on the raddr (AR address prefix) and the MN's link local address of the interface on which the RtAdv was received on and stores it in the given router structure curr.

Function: (extended)
void **h_get_coa (** struct in6_addr *homeaddrINVALID, struct in6_addr *coaddr **)**

Description:
This function returns the MN's CoA (LCoA) in MIPv6 mode it's RCoA if it is in HMIPv6 mode to the address where coaddr points to.

Function: (modified)
void **mipv6_get_old_care_of_address (** struct in6_addr *homeaddr, struct in6_addr *prevcoa **)**

Description:
Returns the LCoA of the previous used AR to the address where prevcoa points to. If no previous router exist or the previous router does not support HMIPv6 NULL is returned.

Function: (new)
int **h_router_event (** struct router *nrt, struct bu_reply *reply, int *is_map **)**

continues on next page

continued from previous page

Description:
This function handles the events significant for the context of the known ARs. Such an event should be the reception of a RtAdv, BA, QoS option, timeout function that indicates the loss of a BU, and a timer function which periodically triggers the check for expired lifetimes or refresh times of existing MAP registrations. If new information is available by the handled event the AR data structure is updated. Additionally the function h_router_propose() is called to identify the best of the available AR and judge whether a handover to the identified AR is reasonable. Finally the function change router is called to initiate the execution of required registration operations. See also Figure 5.2.

router

Type (definition): (new)
struct router * **head_ar**

Description:
Pointer to first element of AR data structure. See also Figure 5.1.

Type (definition): (new)
struct anchor * **head_map**

Description:
Pointer to first element of MAP data structure. See also Figure 5.1.

Function: (new)
struct anchor * **h_find_map (** struct in6_addr *search_raddr **)**

Description:
Function to find MAP in the MAP data structure by its IPv6 address.

Function: (modified)
struct router * **h_find_router (** struct in6_addr *search_ll_addr **)**

Description:
Function to find AR in the AR data structure by its link local address.

Function: (new)
struct router * **h_find_router_mac (** int search_ifindex, __u8 *search_mac **)**

Description:
Function to find AR in the AR data structure by its MAC address.

Function: (new)
struct router * **h_find_router_lcoa (** struct in6_addr *search_coa **)**
Description:
Function to find AR in the AR data structure by the LCoA the MN has deduced from the RtAdv of the AR.

Function: (new)
void **h_release_ar2map (** struct router *rptr **)**
Description:
Function to release a link (relation) in the AR and MAP data structure between a given AR and its affiliated MAP. See also Figure 5.1.

Function: (extended)
int **h_delete_router (** struct router *search_ar, struct router **curr_router_p, struct router **tent_router_p **)**
Description:
Function to delete an AR element from the AR data structure and free the allocated memory of its router structure.

Function: (new)
void **h_add_map (** struct anchor *new **)**
Description:
Function to link a given element (anchore structure) in the MAP data structure.

Function: (original)
void **h_add_ar (** struct router *new **)**
Description:
Function to link a given element (router structure) in the AR data structure.

Function: (new)
struct anchor * **h_new_map (** struct anchor* nmap **)**
Description:
Function to create a new MAP entry in the MAP data structure. Checks maximum number of entries, allocates memory, and copies information from the given temporary anchor structure to the element it finally links in the MAP data structure.
continues on next page

continued from previous page

Function: (original)
struct router * **h_new_router (** struct router* nrt **)**
Description:
Function to create a new AR entry in the AR data structure. Checks maximum number of entries, allocates memory, and copies information from the given temporary router structure to the element it finally links in the AR data structure.

Function: (new)
int **h_router_update_reply (** struct bu_reply *reply, int *is_map, struct router *curr_router, struct anchor *curr_map **)**
Description:
Function to update the MN's AR and MAP data structure with new information given by a specific event. This event could be the reception of a positive or negative BA or QoS message or the trigger of a timeout function indicating the loss of a previously sent BU to a MAP. If the event was the reception of a QoS reply message, the MN's MDS, IDS, and RDS is updated.

Function: (new)
int **h_router_update_ra (** struct router *nrt, struct router **curr_router_p, struct router **tent_router_p **)**
Description:
Function to update the MN's AR and MAP data structure with new information from a received RtAdv.

Function: (extended)
void **h_list_free (** struct router **curr_router_p, struct anchor **curr_map_p **)**
Description:
Function to delete all entries in the AR and MAP data structure when the mobility module is unloaded.

Function: (new)
void **h_router_purge (** struct router **curr_router_p, struct anchor **curr_map_p **)**
Description:
continues on next page

continued from previous page

Function to purge the AR and MAP data structure from unused and elapsed entries.

Function: (new)

struct router * **h_router_propose (** struct router **curr_router_p, struct anchor **curr_map_p, struct in6_addr *home_prefix, int home_plen **)**

Description:

Function to select "the best" AR from the list of maintained AR in the AR data structure and identify the required steps to be executed for establishment of the selected AR as the MN's default router. The method applied in this function is illustrated in Figure 5.3. The position of this function in the MN's event handling process is illustrated in Figure 5.2.

mn

Function: (new)

int **h_update_map (** struct in6_addr *r_coa, struct in6_addr *l_coa, struct in6_addr *map, struct in6_addr *im_ar, int plen, __u32 map_opt_lifetime, __u32 pfl1_fl, __u32 pfl1_lifet, __u16 pfl1_dbw, __u16 pfl1_abw, struct in6_addr *pfl1_homeaddr, struct in6_addr *pfl1_rcoa, struct in6_addr *pfl1_cn **)**

Description:

This function simply calls h_send_upd_option_map_ph1 which then coordinates the sending of a BU to a MAP. See also Figure 5.4.

Function: (new)

void **h_bu_to_prev_map (** struct in6_addr *prev_map, struct in6_addr *prev_rcoa **)**

Description:

Function initiates sending of a binding release message to the given previous MAP's IPv6 address. It simply calls mipv6_send_upd_option() with corresponding parameters.

Function: (extended)

int **h_mn_moved (** struct in6_addr *coa, struct in6_addr *prev_router, int plen, unsigned long ps, struct in6_addr *prev_map, struct in6_addr *prev_rcoa **)**

Description:

This function is called by the function handle_prefix() (See also Figure 5.4 to coordinate sending of BUs to HA (by calling function mipv6_send_upd_option()) and CN (by calling function bul_iterate(mipv6_cn_bu_send())) and sending of binding release messages to the previously used MAP (by calling function h_bu_to_prev_map()).

continues on next page

continued from previous page

Function: (extended)
int **mipv6_mn_returned_home (** struct in6_addr *prev_router, int plen, struct in6_addr *prev_map, struct in6_addr *prev_rcoa **)**
Description:
This function is similar to the function h_mn_moved() but is called when the MN has moved back to its home link. It coordinates the sending of binding release messages to HA, potential CNs and a previously used MAP.

sendopts

Function: (new)
int **h_send_thrq_packet (** struct in6_addr *map, struct in6_addr *r_coa, struct in6_addr *l_coa, struct in6_addr *im_ar, __u32 pfl1_fl, __u32 pfl1_lifet, __u16 pfl1_dbw, __u16 pfl1_abw, struct in6_addr *pfl1_homeaddr, struct in6_addr *pfl1_rcoa, struct in6_addr *pfl1_cn **)**
Description:
Function used by MN to manage composition and sending of a packet containing a QoS-hop-by-hop option header and/or a routing header to assure that the packet travels a specified data path which does not contain the MN's default router. Such a packet is sent to be combined with a BU to the MAP when performing a handover in HMIPv6 mode. A QoS option is only appended if related arguments of this function calls are valid.

Function: (extended)
int **h_callback_function_Xack (** struct mipv6_bul_entry *bulentry, __u32 now, __u32 lifetime, __u32 refresh, int status **)**
Description:
This function is called by the function mipv6_ack_rcvd() when the MN has received a positive or negative BA from HA, CN or MAP. It calls h_router_event with related reply parameters to express this information and trigger further event handling. See also Figure 5.2.

Function: (new)
static int **h_map_callback_bu_timeout (** struct mipv6_bul_entry *bulentry **)**
Description:
continues on next page

continued from previous page

This function is registered with a timer function to indicate the loss of a BU send to a MAP. When the timer function expires before the corresponding BA from the MAP is received by the MN, the function h_map_callback_bu_timeout() calls h_router_event with related reply parameters to express this information and trigger further event handling. See also Figure 5.2.

Function: (extended)

struct ipv6_opt_hdr * **h_add_2_dst1opts (** struct in6_addr *saddr, struct ipv6_opt_hdr *hdr, int append_ha_opt **)**

Description:

This function is used to add the destination headers home address option (HOA) and BU to packets sent by the MN.

Function: (new)

int **h_send_upd_option_map_ph1 (** struct in6_addr *saddr, struct in6_addr *l_coa, struct in6_addr *daddr, struct in6_addr *im_ar, long maxdelay, __u32 initdelay, __u32 maxackdelay, __u8 exp, __u8 flags, __u8 plength, __u32 lifetime, struct mipv6_subopt_info *sinfo, __u32 pfl1_fl, __u32 pfl1_lifet, __u16 pfl1_dbw, __u16 pfl1_abw, struct in6_addr *pfl1_homeaddr, struct in6_addr *pfl1_rcoa, struct in6_addr *pfl1_cn **)**

Description:

Function manages sending of a BU to a MAP. Also starts timer to indicate the potential loss of that BU and makes an entry in the MN's BUL. To assure that the packet travels the tentative new path while the old AR is till considered as default router a routing header via the new AR is added to the packet to the MAP by calling function h_send_thrq_packet(). In case of a QoS request also a QoS option is added by this function.

Function: (extended)

int **mipv6_ack_rcvd (** int ifindex, struct in6_addr *cnaddr, __u16 sequence, __u32 lifetime, __u32 refresh, int status **)**

Description:

This function is called when a BA is received. It updates the BUL with new information like binding lifetime and refresh time and calls h_callback_function_Xack() to also update the AR and MAP data structure with new information and trigger further event handling.

A.2 QoS-Conditionalized BU Related Functions and Data Structures

qos-monitor

Function: (new)
static void **h_purge_update_show_fl1 (** void **)**
Description:
Function that is periodically called to purge the resource data structure from exceeded or invalid entries. When finished the function shows the first valid entry of RDS.

Function: (new)
static int **qoutfn (** struct sk_buff *skb, struct nf_info *info, void *data **)**
Description:
Queue processing function, called (after preprocessing), by the netfilter module. This function performs further processing of QoS option depending on whether the node it is running on is a HA, MAP or the MN. In case of a MAP or IR the resource data structure is updated according to the MN's desired and acceptable QoS bandwidth requirements and the currently available bandwidth in this node. In case of a local handover stale resources on the old data path of the MN below this MAP are released by initiating the sending of a negative QoS-message to the MN's previous AR. If the node is a MN the its RDS is updated by calling h_router_event(). The function always returns the verdict accept to the netfilter module indicating to proceed processing with the packet.

Function: (new)
static unsigned int **qos_hook (** unsigned int hook, struct sk_buff **pskb, const struct net_device *indev, const struct net_device *outdev, int (*okfn) (struct sk_buff *) **)**
Description:
This is the registered preprocessing function with netfilter. It checks whether the intercepter packet contains a QoS-hop-by-hop option and if yes returns a NF_QUEUE verdict indicating that further processing of the packet is required.

Function: (new)
static int __init **init (** void **)**
Description:
Initializing function for qos-monitor module. This function register the preprocessing function qos_hook() and queue processing function qoutfn() with the hook NF_IP6_PRE_ROUTING. Additionally it initializes the timer queue for periodically purging stale entries in the RDS and the available bandwidth DS.
continues on next page

continued from previous page

Function: (new)
static void __exit **fini (** void **)**

Description:
Function for unregistering of qos-monitor module. This function unregisters the pre-processing function qos_hook() and queue processing function qoutfn() with the hook NF_IP6_PRE_ROUTING. Additionally it destroys the timer queue for periodically purging stale entries in the RDS and frees he available bandwidth DS.

mipv6

Type (declaration): (new)
struct **res_record**

Data Fields:

struct in6_addr	resr_mn_rcoa	RCoA of the MN, used as source or destination address for the flow
struct in6_addr	resr_mn_hoaddr	Home address of the MN (for identification)
struct in6_addr	resr_cn_addr	CN address of the MN, used as source or destination address for the flow
struct in6_addr	resr_ar_addr	AR address of the path where resources a reserved
__u8	resr_FDA_flags	FDA flags of the QoS option
unsigned long	resr_fl	flow label of the flow
unsigned int	resr_dbw	Desired bandwidth for the flow
unsigned int	resr_abw	Acceptable bandwidth for the flow
unsigned int	resr_rbw	Actually reserved bandwidth for the flow
unsigned long	resr_timestamp	Timestamp of the reservation in jiffies
unsigned long	resr_lifetime	Total lifetime of the reservation in seconds
struct res_record *	resr_next	Pointer to next element of resource date structure list

Description:
Record of the resource data structure to maintain reserved resources.

Type (declaration): (new)
struct **h_qos_bw_object**

Data Fields:

__u8	qo_type	Type field of the bw option
__u8	qo_len	Length field of the bw option
__u16	qo_bw	Field to indicate desired or acceptable bandwidth for a flow containing this option

continues on next page

continued from previous page

Description:
Bandwidth option to indicate the QoS requirements of a flow.

Type (declaration): (new)
struct **hmipv6_hopopt_qosrq**

Data Fields:

__u8	type	Type-code of the option
__u8	length	Option length of the option
__u8	FDA_flags	FDA and reserved flags, initiated with 00100000
__u8	flow_lbl[3]	Flow label of this flow
__u32	lifetime	Lifetime of reservations for this flow
struct in6_addr	r_coa	RCOA of the MN and flow source and/or destination address
struct in6_addr	cn_addr	CN address of the MN and flow source and/or destination address
struct in6_addr	ho_addr	Home address of the MN, only required for accounting and global unique identification
struct in6_addr	ar_addr	AR address to identify path, required by MAP to release resource on path when MN moves to new AR
struct h_qos_bw_object	dbw	QoS requirement option to indicate desired bandwidth of the flow
struct h_qos_bw_object	abw	QoS requirement option to indicate acceptable bandwidth of the flow

Description:
This structure wraps a QoS object, which is carried in a IPv6-hop-by-hop option to signal the MN's bandwidth requirements to the data path up to the MAP.

Type (definition): (new)
struct res_record * **head_resr**

Description:
Root pointer of the resource data structure.

mdetect

Type (declaration): (new)
struct **bu_reply**

Data Fields:

int	type	Type of the reply to a BU or QoS message, could be of: BU_BACK , BU_BNACK , BU_TIMEOUT , QOS_ACK

continues on next page

continued from previous page

struct in6_addr	raddr	Address of replying node, in general HA or MAP. Also contains prefix
struct in6_addr	coa	CoA the MN registered with this CN
unsigned long	now	Timestamp when reply was received in jiffies
unsigned long	lifetime	Lifetime of reply, Calculated by: (bulentry->expire - now)/HZ
unsigned long	refresh	Refresh time of binding, calculated by: 4*lifetime/5
struct in6_addr	ho_addr	Home address of MN taken from QoS reply: hmipv6_hopopt_qosrq structure
struct in6_addr	r_coa	RCoA of MN, taken from QoS reply: hmipv6_hopopt_qosrq structure
struct in6_addr	cn_addr	CN address of MN taken from QoS reply: hmipv6_hopopt_qosrq structure
__u8	resr_FDA_flags	FDA-flags taken from QoS reply: hmipv6_hopopt_qosrq structure
unsigned long	resr_fl	Flow label taken from QoS reply: hmipv6_hopopt_qosrq structure
unsigned int	resr_dbw	Desired bandwidth taken from QoS reply: hmipv6_hopopt_qosrq structure
unsigned int	resr_abw	Acceptable bandwidth taken from QoS reply: hmipv6_hopopt_qosrq structure
struct bu_reply *	next_reply	Pointer to next bu_reply record if more that one QoS option is contained in a reply message

Description:
Data structure to handle the reply event to a previously send BU(+QoS) message. This could represent a positive or negative BA, a QoS option, or a timeout indicating the loss of a message.

Type (declaration): (new)
struct **qos_flow**

Data Fields:

__u32	pfl_fl	Flow label value applied to last QoS request, taken from MN's IDS (hmipv6_mn_fl1_fl).
struct in6_addr	pfl_cn	CN address value applied to last QoS request, taken from MN's IDS (hmipv6_mn_fl1_cn) (requires further investigation since no end-to-end signaling is supported yet).

continues on next page

continued from previous page

__u16	pfl_dbw	Desired bandwidth value applied to last QoS request, taken from MN's IDS (hmipv6_mn_fl1_dbw).
__u16	pfl_abw	Acceptable bandwidth value applied to last QoS request, taken from MN's IDS (hmipv6_mn_fl1_dbw).
__u16	pfl_rbw	Finally reserved bandwidth taken from corresponding QoS reply (struct bu_reply) and is copied to MN's IDS (hmipv6_mn_fl1_rbw).
__u32	pfl_lifetime	Lifetime of QoS reservation, taken form corresponding QoS reply.
__u32	pfl_timestamp	Timestamp of received QoS reply, necessary to calculate remaining lifetime.
struct qos_flow *	next_qos_flow	Pointer to next qos_flow entry if resources for multiple flows were reserved via this path (AR).

Description:
Entry contained in every known AR record of the MN to maintain reserved resources on the path indicated by that AR. This data structure represents the resource data structure (RDS) of the MN.

Type (declaration): (extended)
struct **router**

Data Fields:

struct router *	prev_router	Pointer to the data structure of the AR previously used as default router.
struct in6_addr	ll_addr	Link local address of the AR.
struct in6_addr	raddr	Address prefix or global address of the AR.
__u8 [MAX_ADDR_LEN]	link_addr	Link layer address of the AR.
__u8	link_addr_len	Link layer address len.
int	ifindex	Interface index of the MN's interface on which the related RA was received.
struct router *	next	Pointer to next element (router structure) in the AR data structure.
int	pfix_len	Prefix length of the link served by this AR.
int	state	State of the router, not used anymore.
unsigned long	lifetime	Lifetime value taken from the last received RA.
__u32	last_ra_rcvd	Timestamp when last RA was received in jiffies.

continues on next page

continued from previous page

__u32	interval	Interval value taken from the last received RA.
int	glob_addr	Indicates whether the address stored in raddr is also the global address of the AR.
__u8	flags	flags set in the received RA.
HMIPv6 related extensions:		
struct in6_addr	CoA	CoA used with this router. If the MN is in HMIPv6 mode this field contains the MN's RCoA.
__u8 [ETH_ALEN]	mac_addr	Source MAC address of received RA.
__u32	last_mac_rcvd	Timestamp of last received packet that contained the above MAC address as source address.
struct in6_addr	l_coa	LCoA the MN uses with this AR.
__u32	last_map_rcvd	Timestamp of last received MAP option contained in a RA of this AR.
__u32	map_ack_expiery	MAP binding expiration time in jiffies registered via this AR.
__u32	map_ack_refresh	MAP binding refresh time in jiffies for registration via this AR.
int	map_unacked_bu_send	Number of unacked BU send to MAP via this AR.
__u32	last_map_bu_send	Timestamp of last send BU to MAP via this AR.
unsigned	long context	Context field of this AR (See also Figure 5.3).
__u32	bad_path_expiery	This is set to jiffies + BAD_PATH_LIFETIME when map_unacked_bu_send has reached MAX_MAP_UNA_BU_SEND. It results in the denial of the well behaved flag of the AR's context field.
__u32	last_map_nack_rcvd	Set to jiffies when NBACK received or when bad_path_expiery is set because path to MAP caused trouble. This field is used to let the well behaved property of a path elapse after some time.
QoS-Conditionalized BU related extensions:		
struct qos_flow	pfl1	Structure used to store reserved and requested resources on the path represented by this AR. (MN's RDS)

continues on next page

continued from previous page

Description:
Element of the AR data structure used in the MN to maintain known AR and information related to the data path represented by that AR. See also Figure 5.1. Regarding QoS support the data structure is extended to maintain also the reserved resources on the path up to the MAP.

Function: (new)
static void **h_router_state (** unsigned long foo **)**

Description:
Function periodically called (The period with which it is called can be dynamically adjusted with the proc-file-entry: /proc/sys/net/ipv6/mobility/mn_rst_p. Default is 10 == 100ms.) to check for:
- The lifetime or interval value of the current router has expired. Then h_router_event() is called for further processing.
- hmipv6_mn_mac_port to simulate movement has changed (This is a feature to trigger a movement from user space).
See also Figure 5.2. Regarding QoS support: extended to also check whether MN's resource requirements for the current path have changed or reserved resources are about to expire.

Function: (new)
int **h_router_event (** struct router *nrt, struct bu_reply *reply, int *is_map **)**

Description:
This function handles the events significant for the context of the known ARs. Such an event should be the reception of a RtAdv, BA, QoS option, timeout function that indicates the loss of a BU, and a timer function which periodically triggers the check for expired lifetimes or refresh times of existing MAP registrations. If new information is available by the handled event the AR data structure is updated. Additionally the function h_router_propose() is called to identify the best of the available AR and judge whether a handover to the identified AR is reasonable. Finally the function change router is called to initiate the execution of required registration operations. See also Figure 5.2.

continues on next page

continued from previous page

router

Function: (new)
struct qos_flow * **h_find_qos_flow (** unsigned long resr_fl, struct in6_addr *cn_addr, struct qos_flow *head_qos_flow **)**
Description:
Function to find an entry in the MN's RDS according to given flow label and CN address. If entry exists pointer to RDS entry is returned, otherwise NULL.

Function: (new)
void **h_update_mn_flx_rb (** struct qos_flow *tmp_qos_flow **)**
Description:
Function to update the MN's IDS with the given QoS flow parameters.

Function: (new)
int **h_router_update_reply (** struct bu_reply *reply, int *is_map, struct router *curr_router, struct anchor *curr_map **)**
Description:
Function to update the MN's AR and MAP data structure with new information given by a specific event. This event could be the reception of a positive or negative BA or QoS message or the trigger of a timeout function indicating the loss of a previously sent BU to a MAP. If the event was the reception of a QoS reply message, the MN's MDS, IDS, and RDS is updated.

procrcv

Function: (extended)
static void **mipv6_bu_delete (** struct in6_addr *saddr, struct in6_addr *daddr, struct in6_addr *haddr, struct in6_addr *coa, int ack, int home, int router, int plength, __u16 sequence, __u32 lifetime **)**
Description:
Function to delete an entry in the BC of a node. Has been extended to consider the processing of QoS reservations when delaying an entry in the BC of the MAP. If a de-registration occurred due to a regional movement of the MN, resources on the old data path are released by initiating the sending of a negative QoS message to the MN's old AR.
continues on next page

continued from previous page

Function: (extended)
int **mipv6_handle_bindupdate (** struct sk_buff *skb, int optoff **)**

Description:
Function to handle a received BU of a MN. The function has been extended to consider potentially existing entries in the MAP's RDS for that binding. If that is the case and the MN's QoS request has been assured, the sending of a positive BA+QoS option is prepared otherwise the sending of a negative BA+QoS option to the MN is prepared and the binding entry is deleted.

sendopts

Function: (new)
int **h_send_thrq_packet (** struct in6_addr *map, struct in6_addr *r_coa, struct in6_addr *l_coa, struct in6_addr *im_ar, __u32 pfl1_fl, __u32 pfl1_lifet, __u16 pfl1_dbw, __u16 pfl1_abw, struct in6_addr *pfl1_homeaddr, struct in6_addr *pfl1_rcoa, struct in6_addr *pfl1_cn **)**

Description:
Function used by MN to manage composition and sending of a packet containing a QoS-hop-by-hop option header and/or a routing header to assure that the packet travels a specified data path which does not contain the MN's default router. Such a packet is sent to be combined with a BU to the MAP when performing a handover in HMIPv6 mode. A QoS option is only appended if related arguments of this function calls are valid.

Function: (new)
int **h_send_rsrp_packet (** struct in6_addr *map_addr, struct in6_addr *mn_lcoa, struct res_record *first_resr **)**

Description:
Function used by a MAP to attach a QoS option to a BA to the MN.

Function: (new)
struct res_record * **h_get_next_resr_rcoa (** struct in6_addr *search_rcoa, struct in6_addr *search_ar_addr, struct res_record *last_resr **)**

Description:
Function to return a pointer to the next entry after a given entry in the node's RDS that fits the given arguments for the flows RCoA and AR addr.

continues on next page

continued from previous page

Function: (extended)
int **mipv6_send_ack_option (** struct in6_addr *saddr, struct in6_addr *daddr, long maxdelay, __u8 status, __u16 sequence, __u32 lifetime, __u32 refresh, struct mipv6_subopt_info *sinfo, struct in6_addr *hoaddr **)**

Description:
Function to manage sending of a BA to the MN. The function is extended with the capability to attach a QoS option to the BA depending on the MN's QoS requirements that could be assured by a MAP.

Function: (new)
int **h_send_upd_option_map_ph1 (** struct in6_addr *saddr, struct in6_addr *l_coa, struct in6_addr *daddr, struct in6_addr *im_ar, long maxdelay, __u32 initdelay, __u32 maxackdelay, __u8 exp, __u8 flags, __u8 plength, __u32 lifetime, struct mipv6_subopt_info *sinfo, __u32 pfl1_fl, __u32 pfl1_lifet, __u16 pfl1_dbw, __u16 pfl1_abw, struct in6_addr *pfl1_homeaddr, struct in6_addr *pfl1_rcoa, struct in6_addr *pfl1_cn **)**

Description:
Function manages sending of a BU to a MAP. Also starts timer to indicate the potential loss of that BU and makes an entry in the MN's BUL. To assure that the packet travels the tentative new path while the old AR is till considered as default router a routing header via the new AR is added to the packet to the MAP by calling function h_send_thrq_packet(). In case of a QoS request also a QoS option is added by this function.

Function: (new)
void **h_update_resr_flow (** struct hmipv6_hopopt_qosrq *qosrq_p, struct res_record *tmp_resr **)**

Description:
Function to update a given entry in the MAP's or IR's RDS according to QoS request of a MN.

Function: (new)
void **h_show_fl1_resr (** struct res_record *tmp_resr **)**

Description:
Function to update the MAP's or IR's IDS with current content of a given RDS entry.

continues on next page

continued from previous page

Function: (new)
void **h_apply_qos_req (** struct res_record *tmp_resr **)**

Description:
Function to judge on a given RDS entry representing a MN's QoS request, if and how much of the requested bandwidth for that flow can be assured. Eventually updates the MAP's or IR's RDS and available bandwidth data structure according to the currently reserved bandwidth for that flow.

Function: (new)
void **h_mangle_qos_opt (** struct hmipv6_hopopt_qosrq *qosrq_p, struct res_record *tmp_resr **)**

Description:
Function used by the IRs to modify a traversing QoS option if the desired resources indicated in the QoS option can not be fully satisfied.

Function: (new)
int **h_validate_address (** struct sk_buff *skb, struct in6_addr *addr **)**

Description:
Function used by a MAP to validate the contained destination address of a received BU+QoS message.

Function: (new)
struct hmipv6_hopopt_qosrq * **h_parse_qos (** struct sk_buff *skb **)**

Description:
Function to check whether a given IPv6 message contains a valid QoS option.

Function: (new)
struct res_record * **h_get_resr_flow (** struct in6_addr *search_r_coa, struct in6_addr *search_cn_addr, struct in6_addr *search_ar_addr, unsigned long search_fl **)**

Description:
Function used by MAP's and IR's to find a specific entry in the node's RDS according to the given RCoA, CN addr, AR addr and flow label. If the AR address is NULL this value is not considered for the finding process.

continues on next page

continued from previous page

Function: (new)
truct res_record * **h_purge_resr_list (** void **)**
Description:
Function used by MAP's and IR's to purge stale entries in their RDS.

Function: (new)
struct res_record * **h_free_resr_list (** void **)**
Description:
Function used to destroy all entries of a node's RDS when the qos-module is unloaded.

Function: (new)
struct res_record * **h_new_resr_flow (** struct in6_addr *search_ho_adddr, struct in6_addr *search_r_coa, struct in6_addr *search_cn_addr, struct in6_addr *search_ar_addr, unsigned long search_fl **)**
Description:
Function to allocate a new entry in the RDS for maintenance of a new QoS request.

Function: (new)
int **h_check_resr_rcoa (** struct in6_addr *search_rcoa, struct in6_addr *search_ar_addr **)**
Description:
Function to chech whether a valid entry in the RDS for a given RCoA exists.

dstopts

Function: (extended)
static __inline__ int **mipv6_calculate_option_pad (** __u8 type, int offset **)**
Description:
Returns how many bytes of padding must be appended before (sub)option (alignment requirements can be found from the mobile-IPv6 draft, sect5). This function has been extended for calculation of a QoS option.
continues on next page

continued from previous page

Function: (new) struct ipv6_opt_hdr * **h_generate_qos_hop_header (** __u32 pfl1_fl, __u32 pfl1_lifet, __u16 pfl1_dbw, __u16 pfl1_abw, struct in6_addr *pfl1_homeaddr, **)**
Description: Function to assemble an IPv6 QoS-hop-by-hop option based on the given parameters.

Appendix B

Additional Performance Results

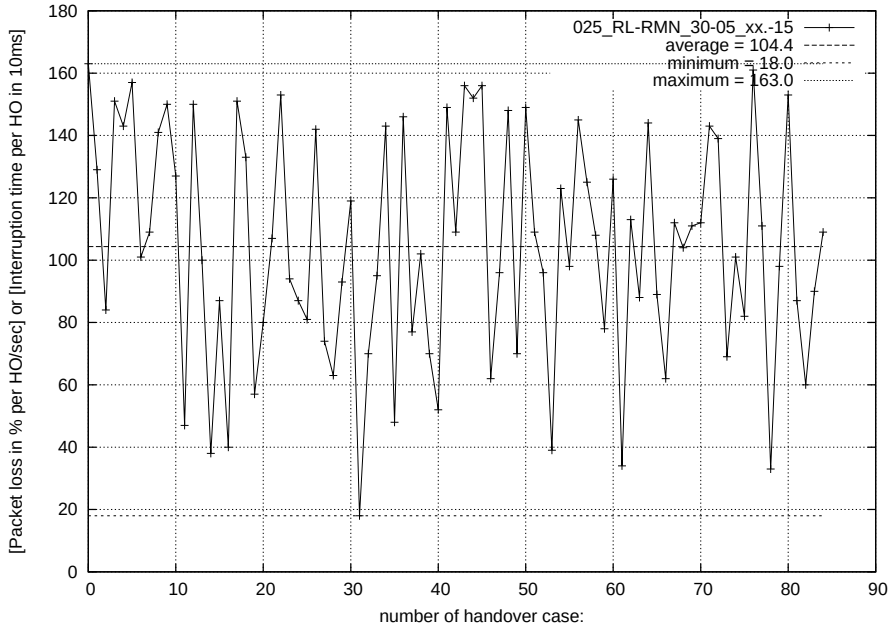


Figure B.1: Experimental results for downstream packet loss for several handover emulations with MIPv6 (WAN delay: 30 ms, AN delay: 5 ms, lazy movement detection based on RtAdv period of 1.5 sec).

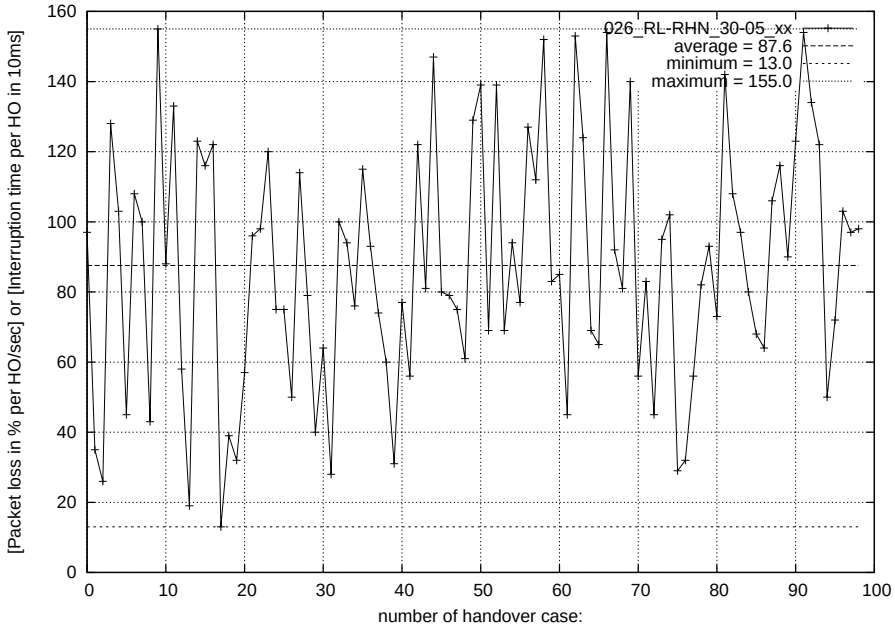


Figure B.2: Experimental results for downstream packet loss for several local handover emulations with HMIPv6 (WAN delay: 30 ms, AN delay: 5 ms, lazy movement detection based on RtAdv period of 1.5 sec).

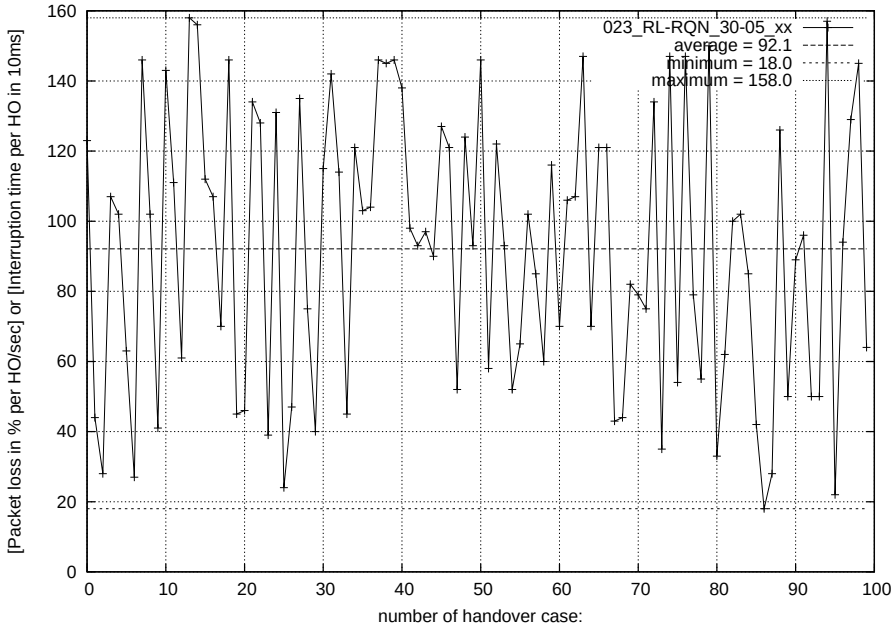


Figure B.3: Downstream packet loss for several QoS-conditionalized local handover emulations with HMIPv6 (WAN delay: 30 ms, AN delay: 5 ms, lazy movement detection based on RtAdv period of 1.5 sec).

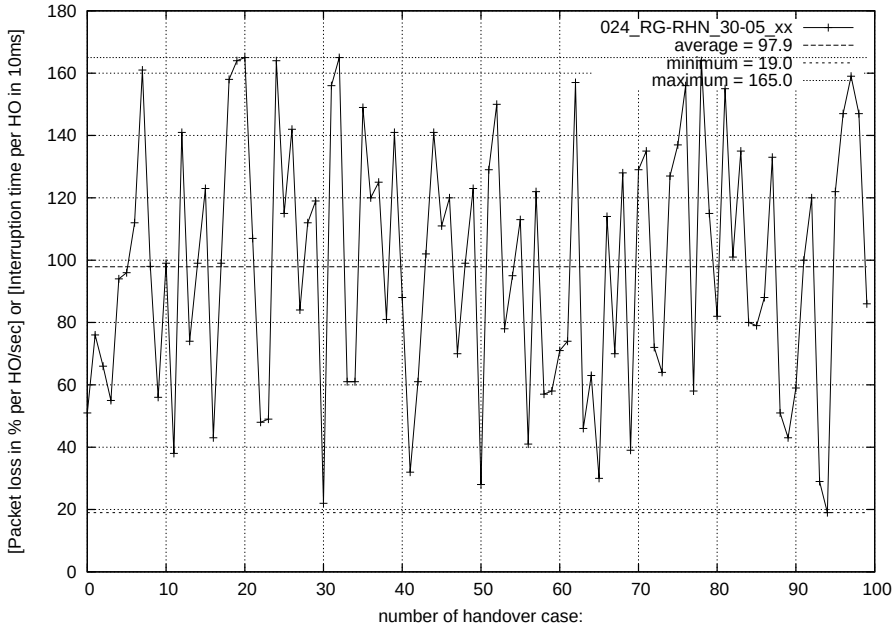


Figure B.4: Downstream packet loss for several regional handover emulations with HMIPv6 (WAN delay: 30 ms, AN delay: 5 ms, lazy movement detection based on RtAdv period of 1.5 sec).

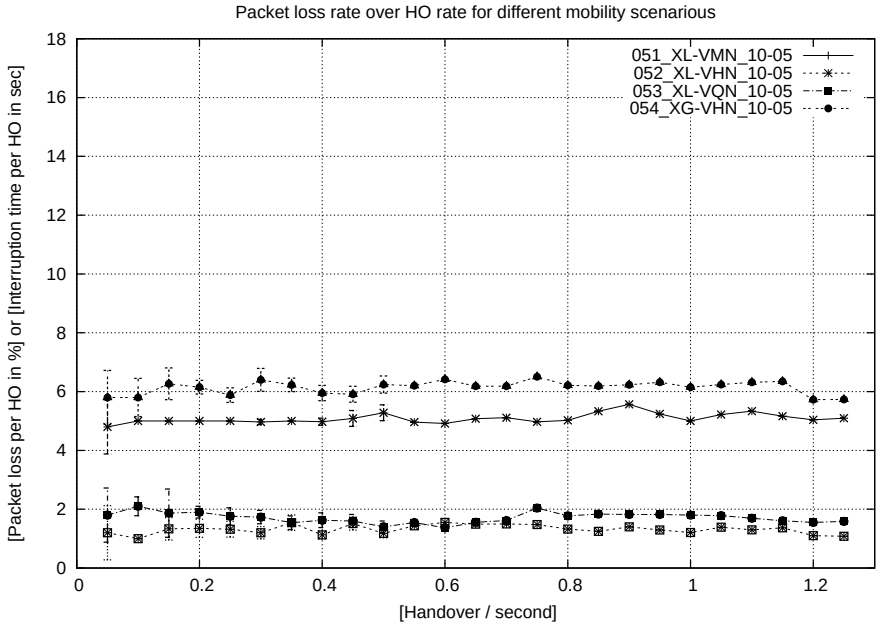


Figure B.5: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 10 ms, AN delay: 5 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

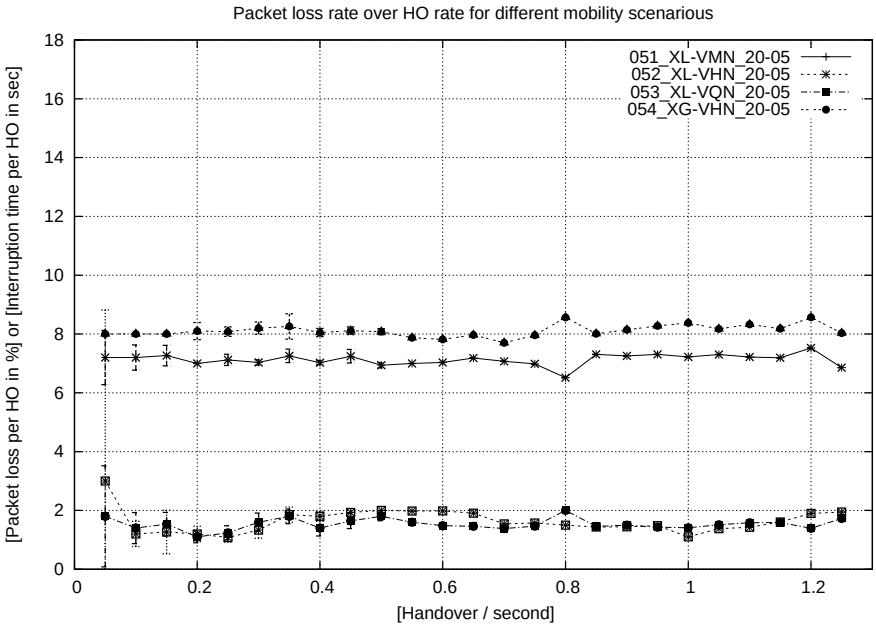


Figure B.6: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 20 ms, AN delay: 5 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

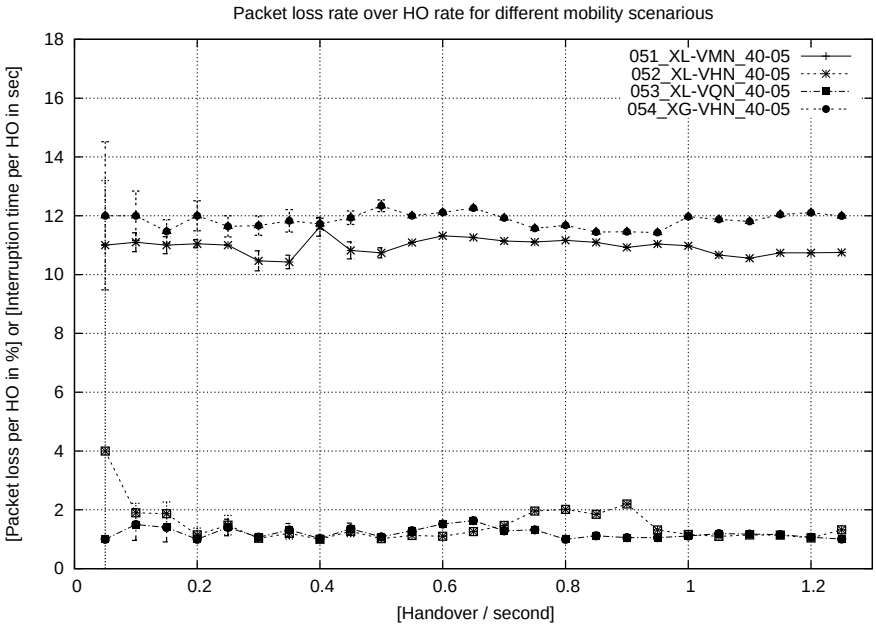


Figure B.7: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 40 ms, AN delay: 5 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

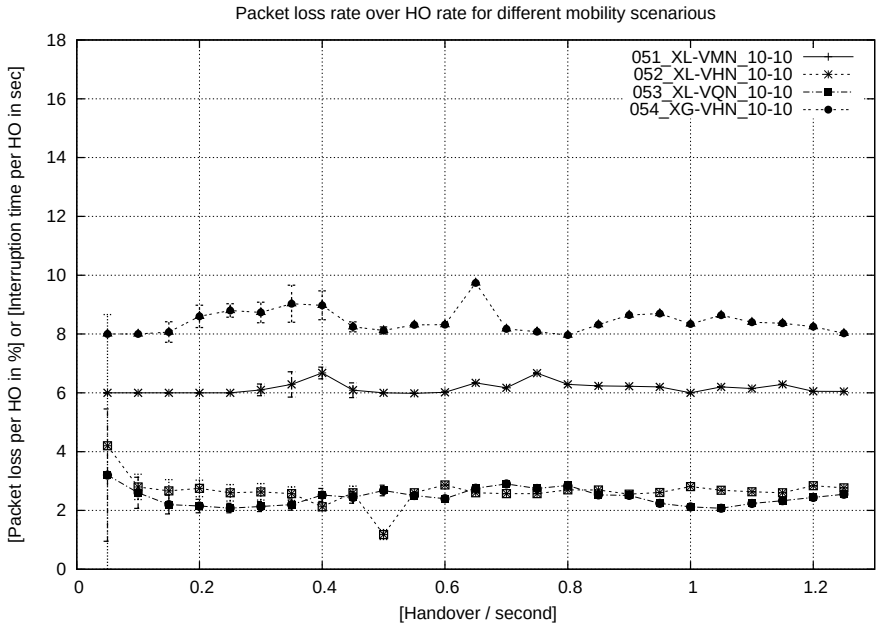


Figure B.8: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 10 ms, AN delay: 10 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

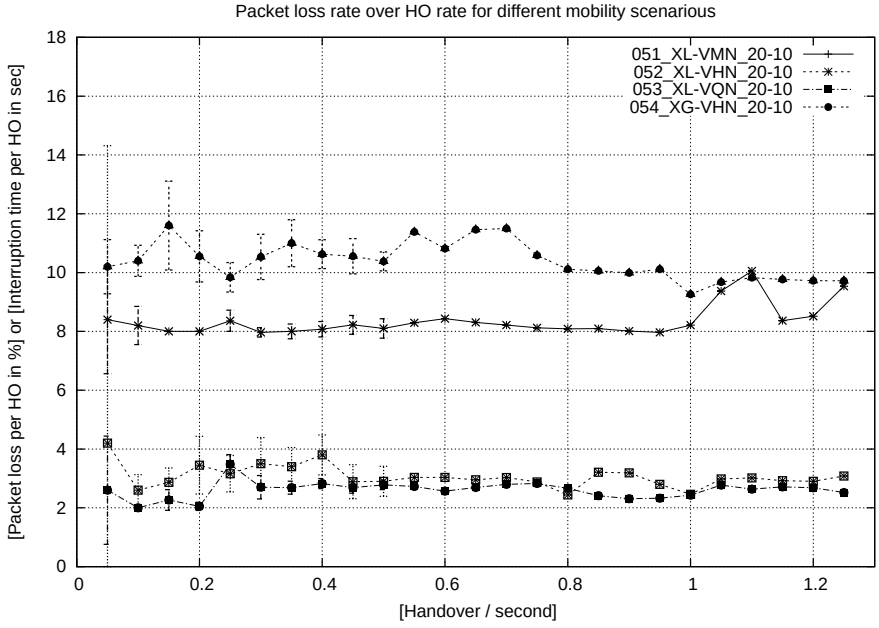


Figure B.9: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 20 ms, AN delay: 10 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

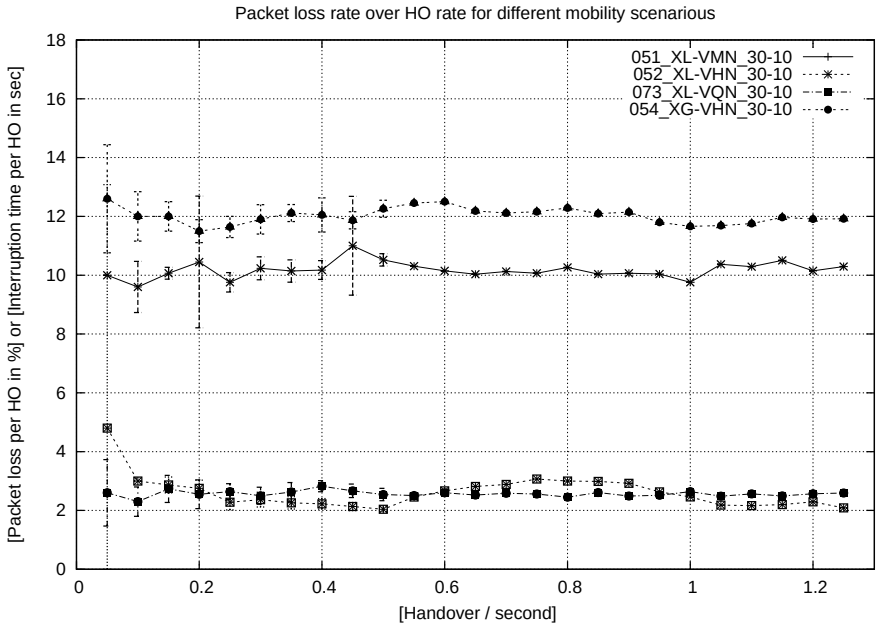


Figure B.10: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 30 ms, AN delay: 10 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

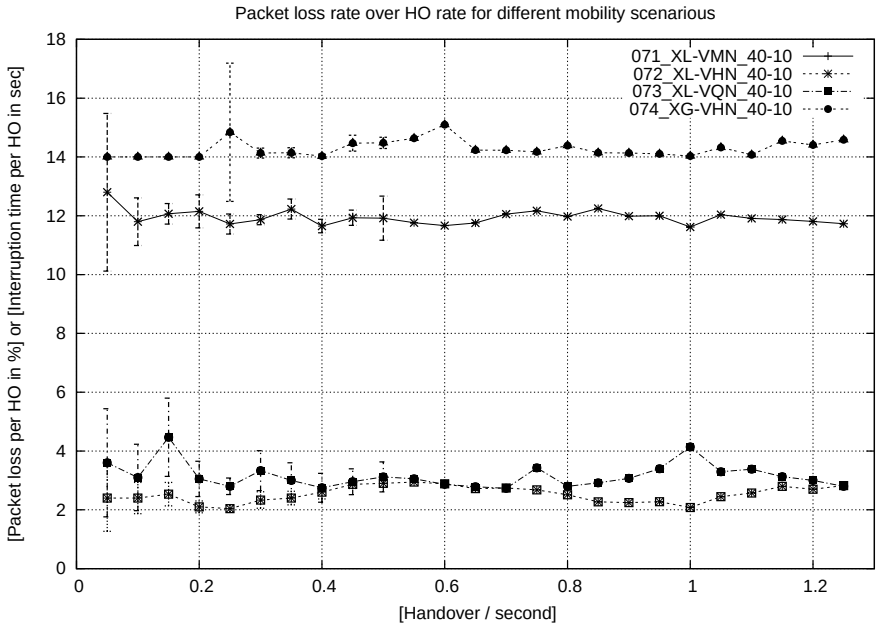


Figure B.11: Experimental results for downstream packet loss in MIPv6, local and regional HMIPv6, and the QoS-Cond. BU handover scenarios over handover rate (WAN delay: 40 ms, AN delay: 10 ms, movement detection delay reduced to less than 1 ms by using a virtual switch).

Appendix C

Installation Manual

This chapter contains installation instructions for the experimental QoCoo implementation Version 0.2

This software is provided without any explicit or implicit warranties or guarantees of any functionality. USE AT YOUR OWN RISK!

The installation procedure is described in sequence of the following steps:

- Hardware and Network Topology Setup
- Nodes OS Configuration
- MIPL MIPv6 Setup
- QoCoo-HMIPv6 Setup
- QoCoo-QoS-Conditionalized BU Setup

Hardware and Network Topology Setup

The setup given in Figure C.1 can be regarded as a "classic" experimental network topology. It is capable to simulate all relevant scenarios such as local and regional handover, QoS or not QoS-conditionalized depending on resource availability in the visited AN.

The implementation has been successfully tested in several setups. However, for beginning it is strictly recommended to conform to the given example setup and node configuration. A much more advanced example setup is given in Section 6.1 and Figure 6.2 of this Work.

Note: Multiprocessor machines probably do not work.

Setup recommendation:

All machines are i386 Pentium II 300MHz. The machines network cards are 3Com 100BaseT Network Interface Card (NIC)s. A manageable HUB is used for emulating the movement between the links 0, 6, 7 and 8. Links 1 and 4 are connected each with a simple 10M HUB. The terms i0 and i1 refer to interface 0 (eth0) and interface 1 (eth1).

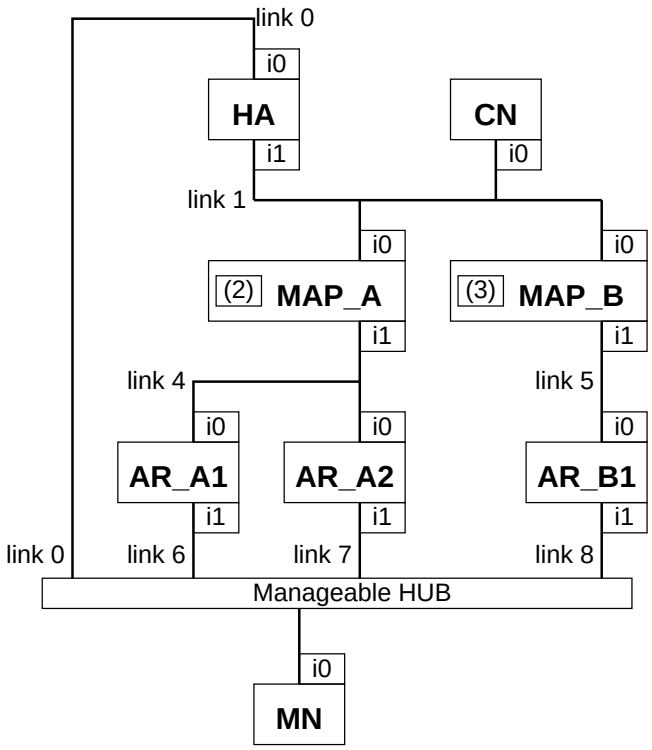


Figure C.1: Example network topology setup

Node OS Configuration:

Redhat 7.2 is used as Linux distribution on all machines. The sources provided with this package only match Linux kernel version 2.4.7. MIPL package mipv6-0.9-v2.4.7 and router advertisement daemon radvd-0.7.1.

MIPv6 Setup

Before coming to the HMIPv6 setup, the MIPv6 setup should work fine. A detailed documentation how to do that is provided with the MIPL and radvd package. Assuming the recommended setup:

- At least HA, CN and MN should be MIPv6 enabled (NO IPSec, NO DHAAD) and configured for their purpose. Some example configuration files are given in the Tables C.1, C.2 (for the HA) and C.3 (for the MN).

- HA, AR_A1, AR_A2 and AR_B1 must have the radvd installed from source. Configure each radvd to advertise the router's availability on the link connected with the manageable HUB. An example configuration file for HA is located in Table C.4

- MAP_A and MAP_B until now simply act as routers between link 1 and links 4 and 5.

- When testing the MIPv6 setup, the MN should properly handover between home link 0 and foreign links 6, 7 and 8.

QoCoo-HMIPv6 Setup:

MN Configuration:

- Copy all files from mobile_ip6_qocoo-0.2/ to the mobile_IP6 source directory of the MN (in general /usr/src/linux/net/ipv6/mobile_ip6/).
- Recompile and restart the mobile_ip6 module. (In general simply type as root:
  - cd /usr/src/linux
  - make modules
  - make modules_install
  - /etc/init.d/mobile-ip6 restart

  should work)
- Since MAP discovery is not enabled yet, regarding the behavior of the MN, yet nothing should have changed to the MIPL implementation (except for a different movement detection and handover processing).

MAP Configuration:

- Configure a virtual link (dummy0 interface) on each MAP (The virtual links are indicated in Figure C.1 by (2) and (3)). AN example configuration file is given in Table C.5. The new dummy0 interface becomes enabled by typing: /etc/init.d/network restart .
- Enable both MAPs as MIPv6 HAs, just like the HA in MIPv6 setup, but serving for their virtual link. Example configuration files for MAP_A are given in Table C.6 and C.7

MAP Discovery Configuration:

In order to perform MAP discovery, MAP options must be attached to the RtAdvs emitted by the MAP. Intermediate routers (e.g. AR) in an access network between MAP and access link must be capable of receiving and propagating the received MAP options with their own RtAdvs. The RtAdv daemon has been modified to be capable of emitting its own MAP options and receiving and propagating other MAP's MAP options on certain links. The configuration language of radvd-0.7.1 was extended to control this behavior. The new interface specific options are listed in Table C.8. The syntax is the same as the original radvd options syntax described by "man radvd". Examples, how to use the new configuration options, are given in Table C.10 (for MAP) and C.11 (for IR).

To enable the two access networks of the example setup in Figure C.1 for HMIPv6 (enable MAPs and MAP discovery) the following configurations steps are required:

- Install original radvd-0-7-1 from source also on MAP_A and MAP_B.

- Copy all files in radvd-0.7.1_hmipv6-0.2/ to the radvd-0.7.1 source directory of MAP_A, MAP_B, AR_A1, AR_A2, AR_B1.
- Go to each modified radvd source directory (at MAPs and ARs) and rerun:
  - ./configure # only necessary if not already run before (alternative: "./configure –prefix=/usr/ –sysconfdir=/etc/ –mandir=/usr/share/man/ –with-pidfile=/var/run/radvd/radvd.pid")
  - make
  - whereis radvd # This should tell you where radvd used to be.
  - cp ./radvd /usr/sbin/radvd # Copy it where 'whereis' told you.
- Configure /etc/radvd.conf in MAP_A and MAP_B for emission of MAP discovery options. An example configuration file for MAP_A is given in Table C.10.
- Configure /etc/radvd.conf in AR_A1 AR_A2 AR_B1 for propagation of MAP discovery options. An example configuration file for AR_A is given in Table C.11.
- (Re)start radvd as background process by typing: "/etc/init.d/radvd restart" or directly from shell in the radvd directory by typing: "./radvd -C /etc/radvd.conf -d4 -m stderr". In order to make the MAP options perceptible for the AR before they are advertised by the MAPs, the last step should be performed at the ARs first.

Finally, when starting the router advertisement daemon on a MAP, the MAP options should be propagated down the access network to the access links. An HMIPv6 enabled MN currently visiting that access link should learn about the availability of the MAP and register with the MAP. Once the binding update (BU) to the MAP is acknowledged, BUs are sent to HA and CN.

QoCoo-QoS-Conditionalized BU Setup:

To enable the AN of a testbed for the QoS-Conditionalized BU all involved nodes in the AN must run the qos-monitor deamon located in mobile_ip6_qocoo-0.2/daemon/.

The QoS-Conditionalized BU involved nodes of Figure C.1 would be MAP_A, AR_A1, and AR_A2 in AN A, MAP_B and AR_B1 in AN B, and the MN.

MAP (MAP_A and MAP_B) Configuration:

- Copy all files and subdirectories from mobile_ip6_qocoo-0.2/ to the mobile_IP6 source directory of the MAP (in general /usr/src/linux/net/ipv6/mobile_ip6/).
- Recompile and restart the mobile_ip6 module. (In general simply type as root:
  - cd /usr/src/linux
  - make modules
  - make modules_install

– /etc/init.d/mobile-ip6 restart

should work)

- Compile and load the QoS-monitor module. Note that the mobile_ip6 module must be started (loaded as indicated by the command: “lsmod”) before the qos-monitor module. In general simply type as root:

  – cd /usr/src/linux/net/ipv6/mobile_ip6/daemon/

  – make

  – insmod qos-monitor.o # This loads the qos-monitor module.

  – lsmod # Check that module is loaded.

  When unloading modules, the qos-monitor module must always be unloaded before the mobile_ip6 module therefore type:

  – rmmod qos-monitor

  before stopping (unloading) the mobile_ip6 module:

  – /etc/init.d/mobile_ip6 stop

No new configuration scripts are required. The possibilities to access available and monitor reserved resources is described in the following.

To specify a routers (MAP, IR, AR) theoretically available resources and monitor currently reserved resources of a MN, a number of new entries have been made to router’s proc-file-system. These entries can be read by the cat command (eg ”cat

/proc/sys/net/ipv6/mobility/debuglevel”) and modified with the echo command (eg echo 2 > /proc/sys/net/ipv6/mobility/debuglevel). Once the QoCoo-modified mobile_ip6 and qos-monitor module is loaded into the kernel of the node, the following new files can be used for QoS maintenance:

- /proc/sys/net/ipv6/mobility/ir_av_bw
  This entry contains the currently available bandwidth that is controlled by this node. When a MN successfully reserves bandwidth for a flow, this value is reduced respectively. When a reservation is explicitly released or a reservation’s lifetime exceeds it is increased to its former value. Its default value is 10.

- /proc/sys/net/ipv6/mobility/ir_res_bw
  This value contains the currently reserved bandwidth that is controlled by this node. It actually represents the node’s total bandwidth - its currently available bandwidth.

The currently reserved resources for a flow as well as certain flow identification parameters can be monitored with the following entries. Note: With this QoCoo version, only the first valid flow of the nodes resource data structure can be accessed from user-space.

- /proc/sys/net/ipv6/mobility/ir_fl1_fl
  This entry indicates the flow label of the flow fl1.

- /proc/sys/net/ipv6/mobility/ir_fl1_lt
  This entry indicates the remaining lifetime of the flow fl1.

- /proc/sys/net/ipv6/mobility/ir_fl1_dbw
  This entry indicates the desired bandwidth requirements of the MN as they are negotiated with other QoS entities on the path, up to this node.

- /proc/sys/net/ipv6/mobility/ir_fl1_abw
  This entry indicates the minimum acceptable bandwidth requirements of the MN that it‘s affiliated BU is conditionalized on.

- /proc/sys/net/ipv6/mobility/ir_fl1_rbw
  This entry indicates the node’s currently reserved bandwidth for flow fl1.

IR / AR (AR_A1, AR_A2 and AR_B1) Configuration:
To enabled an IR/AR for QoS-Conditionalized BU (besides the MAP discovery configuration requirements for IRs/ARs given in the Paragraph: QoCoo-HMIPv6 Setup) the node must run the QoCoo modified mobile_ip6 and qos-monitor modules:

- Configure the nodes like a MIPL CN. Read the MIPL INSTALL for related procedures. An example configuration file is given in Table C.12.

- Copy all files from mobile_ip6_qocoo-0.2/ to the mobile_IP6 source directory of the IR/AR (in general /usr/src/linux/net/ipv6/mobile_ip6/).

- Recompile and restart the mobile_ip6 module. (In general simply type as root:

  – cd /usr/src/linux
  – make modules
  – make modules_install
  – /etc/init.d/mobile-ip6 restart

  should work)

- Compile and load the QoS-monitor module (this step is identical to the above for MAP (MAP_A and MAP_B) Configuration). Note that the mobile_ip6 module must be started (loaded as indicated by the command: “lsmod”) before the qos-monitor module. In general simply type as root:

  – cd /usr/src/linux/net/ipv6/mobile_ip6/daemon/
  – make
  – insmod qos-monitor.o # This loads the qos-monitor module.
  – lsmod # Check that module is loaded.

The possibilities Access to the available and reserved resources parameters is possible via the proc-file-system as described above for MAP (MAP_A and MAP_B) configuration in this text.

MN Configuration:

- Compile and load the QoS-monitor module (this step is identical to the above for MAP or IA/AR Configuration). In general simply type as root:
  - cd /usr/src/linux/net/ipv6/mobile_ip6/daemon/
  - make
  - insmod qos-monitor.o # This loads the qos-monitor module.
  - lsmod # Check that module is loaded.

To specify the MN's resource requirements and monitor the successful reserved resources on the currently used data path, the following new entries in the MN's proc-file-system can be used:

- /proc/sys/net/ipv6/mobility/mn_fl1_fl
  A MN's QoS request must be specified for specific flow value. This entry can be used to specify the flow value, used as the flow label of the MN's fl1-QoS object. The remaining QoS requirement parameters of that flow are not valid until the mn_fl1_fl entry contains a positive integer value. Only one flow (fl1) per MN is allowed.

- /proc/sys/net/ipv6/mobility/mn_fl1_lt
  Indicates the remaining lifetime in seconds of the previous successfully performed reservation of low fl1.

- /proc/sys/net/ipv6/mobility/mn_fl1_dbw
  Can be used to specify the desired bandwidth for flow fl1.

- /proc/sys/net/ipv6/mobility/mn_fl1_abw
  Can be used to specify the acceptable bandwidth for flow fl1. This value should be smaller or equal to the desired bandwidth, otherwise the mn_fl1_dbw value is increased respectively.

- /proc/sys/net/ipv6/mobility/mn_fl1_rbw
  Can be read to monitor the currently reserved bandwidth of flow fl1.

QoS signalling should work whenever the MN is visiting a foreign QoS-Conditionalized BU - and HMIPv6-enabled AN. For comfortable monitoring of all QoS and binding parameters the mn_info_loop tool can be used with the MN and the map_info_loop tool with the MAPs, IRs, and ARs.

Good Luck!!!

If there are any questions feel free to contact neumann@ft.ee.tu-berlin.de

```
###
# Example:
# Allow all nodes with 3ffe:2620:6:1234:: prefix to register except
# nodes 3ffe:2620:6:1234:abcd::.
#
# DENY 3ffe:2620:6:1234:abcd::/80
ALLOW 3ffe:e:e:0::/64
```

Table C.1: Example configuration file for HA:/etc/mipv6_acl.conf

```
FUNCTIONALITY=ha
DEBUGLEVEL=1
TUNNEL_SITELOCAL=yes
MOBILENODEFILE=/etc/mipv6_acl.conf
```

Table C.2: Example configuration file for HA:/etc/sysconfig/network-mip6.conf

```
FUNCTIONALITY=mn
DEBUGLEVEL=2
TUNNEL_SITELOCAL=yes
HOMEADDRESS=3ffe:e:e:0::9/64
HOMEAGENT=3ffe:e:e:0::1/64
RTR_SOLICITATION_INTERVAL=1
RTR_SOLICITATION_MAX_SENDTIME=5
```

Table C.3: Example configuration file for MN:/etc/sysconfig/network-mip6.conf

```
interface eth0
{
AdvSendAdvert on;
MinRtrAdvInterval 1;
MaxRtrAdvInterval 1.5;

AdvHomeAgentFlag off;
AdvIntervalOpt on;

prefix 3ffe:e:e:0::1/64
{
AdvOnLink on;
AdvAutonomous on;
AdvRouterAddr on;
};

};
```

Table C.4: Example configuration file for HA:/etc/radvd.conf

```
DEVICE=dummy0
BOOTPROTO=static
BROADCAST=192.168.233.255
IPADDR=192.168.233.4
NETMASK=255.255.255.0
NETWORK=192.168.233.0
ONBOOT=yes

IPV6INIT="yes"
IPV6ADDR="3ffe:e:e:2::4/64"
```

Table C.5: Example configuration file for MAP:/etc/sysconfig/network-scripts/ifcfg-dummy0

```
ALLOW 3ffe:e:e:2::/64
```

Table C.6: Example configuration file for MAP:/etc/mipv6_acl.conf

```
FUNCTIONALITY=ha
DEBUGLEVEL=2
TUNNEL_SITELOCAL=yes
MOBILENODEFILE=/etc/mipv6_acl.conf
```

Table C.7: Example configuration file for MAP:/etc/sysconfig/network-mip6.conf

APPENDIX to INTERFACE SPECIFIC OPTIONS:

MapOptRecv on | off
This defines whether MAP options advertised by other router on the link connected to this interface should be ignored or received and considered for further propagation.
Note: To receive MAP options on a certain interface, the interface must be configured for sending router advertisements. However, they can be sent with a very low rate.

MapOptProp on | off
This defines whether MAP options, received on this or other interfaces or defined for emission on this interface by mapaddr, should be propagated.
mapaddr address {
list of mapaddr specific options
};
This defines the MAP option emitted via this interface. The address should be the global address of the virtual dummy0 interface the MAP is serving for. ALL possible mapaddr specific options are described below. Each option must be terminated by a semicolon.

MAPADDR SPECIFIC OPTIONS:

MapOptAdv on | off
This defines whether this specific MAP options should be advertised.

MapOptPref integer
This is the value advertised in the preference field of the MAP option. Its must be between 0 and 15.

MapOptLifetime seconds
This defines the lifetime of this MAP option. It is reduced by further intermediate routers that buffer and re-propagate this MAP option until the lifetime is exceeded. Eventually, it limits the lifetime of the BU sent to the MAP.

Table C.8: New radvd options for configuration of MAP option advertisement and propagation (1/2)

The following options should be left untouched because currently only basic mode and the use of RCoA as source address is allowed. For further details for the flags see the draft. **MapOptFlagR on | off** A flag indicating basic mode. (Default on) **MapOptFlagM on | off** A flag indicating extended mode. (Default off) **MapOptFlagI on | off** A flag indicating maybe use of RCoA as SRC. (Default on) **MapOptFlagT on | off** A flag indicating topological incorrect addr. (Default off) **MapOptFlagP on | off** A flag indicating must use of RCoA as SRC. (Default on) **MapOptFlagV on | off** A flag indicating reverse tunneling. (Default off)

Table C.9: New radvd options for configuration of MAP option advertisement and propagation (2/2)

```
interface eth1
{
AdvSendAdvert on;

MinRtrAdvInterval 20;
MaxRtrAdvInterval 30;

AdvHomeAgentFlag off;
AdvIntervalOpt on;

MapOptRecv off;
MapOptProp on;

prefix 3ffe:e:e:4::4/64
{
AdvOnLink on;
AdvPreferredLifetime 200;
AdvValidLifetime 300;
AdvAutonomous on;
AdvRouterAddr on;
};

mapaddr 3ffe:e:e:2::4
{
MapOptAdv on;
MapOptPref 5;
MapOptLifetime 500;
};

};
```

Table C.10: Example configuration file for MAP:/etc/radvd.conf

```
interface eth0
{
AdvSendAdvert on;
MinRtrAdvInterval 400;
MaxRtrAdvInterval 600;
AdvDefaultLifetime 600;

AdvHomeAgentFlag off;
AdvIntervalOpt on;

MapOptRecv on;

prefix 3ffe:e:e:4::7/128
{
AdvOnLink on;
# AdvValidLifetime 300;
AdvAutonomous on;
# AdvRouterAddr on;
# AdvIntervalOpt on;
};

};
interface eth1
{
AdvSendAdvert on;
MinRtrAdvInterval 1;
MaxRtrAdvInterval 1.5;

AdvHomeAgentFlag off;
AdvIntervalOpt on;

MapOptRecv off;
MapOptProp on;

prefix 3ffe:e:e:6::7/64
{
AdvOnLink on;
AdvPreferredLifetime 2;
AdvValidLifetime 2;
AdvAutonomous on;
AdvRouterAddr on;
};

};
```

Table C.11: Example configuration file for AR:/etc/radvd.conf

```
FUNCTIONALITY=cn
DEBUGLEVEL=2
TUNNEL_SITELOCAL=yes
```

Table C.12: Example configuration file for AR:/etc/sysconfig/network-mip6.conf

Appendix D

Slides

Prototypical Implementation and Experimental Testbed Setup of a QoS-Enabled Mobility Concept Based on HMIPv6

Diplomarbeit von:
Axel Neumann <neumann@ft.ee.tu-berlin.de>
Betreuer: Holger Karl, Xiaoming Fu <karl,fu@ft.ee.tu-berlin.de>
TKN, TU-Berlin

November 27, 2002

1

Overview

- **Motivation & Objective**
- **Description of QoCoo** (**Qo**S-**Co**nditionalized hand**o**ver)
  - Basics of HMIPv6 and QoS-Conditionalized BU
  - Open Issues and Possible Solutions
- **Implementation**
  - Implementation Background
  - Design of Selected Functionality
- **Testing**
  - Testbed Setup and Test Cases
  - Performance Experiments
- **Summary and Open Issues**

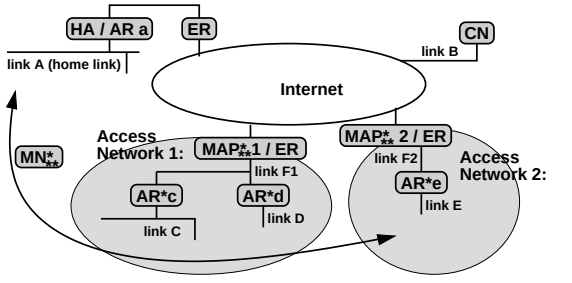
2

Motivation & Objective

- User demand for seamless mobility and QoS provisioning
- One approach: Micro mobility and the close interaction between mobility and QoS management
- HMIPv6 and the QoS-conditionalized BU provide these concepts
- Goal of this prototype implementation:
  - Demonstrate the basic idea of HMIPv6 and QoS-cond BU
  - Show the integration of both schemes
  - Identify potential open issues not covered by the protocol descriptions
  - Provide a testbed for experiments and to validate the key capabilities of underlying protocols

3

Description: System Model



CN Corresponding Node
AR Access Router
HA Home Agent
ER Edge Router
MN Mobile Node
MAP Mobile Anchor Point

* : Enabled for QoS−Conditionalized BU

** : Enabled for Micro Mobility

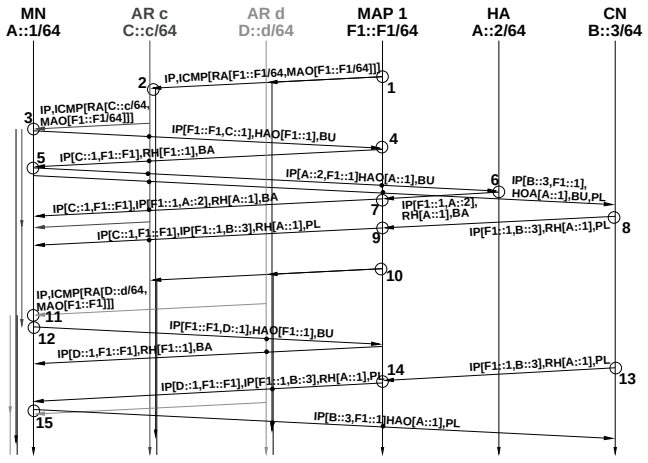
4

Description: Basics

- HMIPv6 implemented based on: draft-ietf-mobileip-hmipv6-06.txt
  - Only basic mode
  - Dynamic MAP discovery
  - Some small details not implemented like:
    - RCoA is always used as source address
  - MAP implementation is based on home agent (HA) implementation
- QoS-cond. BU implementation based on: draft-tkn-nsis-qosbinding-mipv6-00.txt
  - Some open issues and possible solution identified:
    - Release of stale reservations after handover
    - Composition of QoS signaling messages
  - Reduced QoS parameter set for this implementation

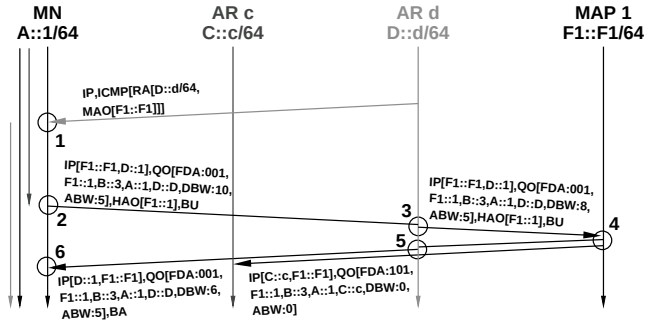
5

Description: HMIPv6 Signaling



6

Description: QoS Signaling



7

Open Issues: Release of Stale Resources

- Local handover: MAP has to release resources in old path where a MN has no connection.
  - But: MAP only gets QoS request along the new path
- Regional handover: MN must send an explicit release message to the old MAP (via the new MAP)
  - Release does not contain a QoS (hop-by-hop) option
  - Triggers old MAP to release all reserved resources for this MN
  - MAP sends a negative QoS option (on behalf of MN) along the old path towards the old AR
- Timers are used as fallback

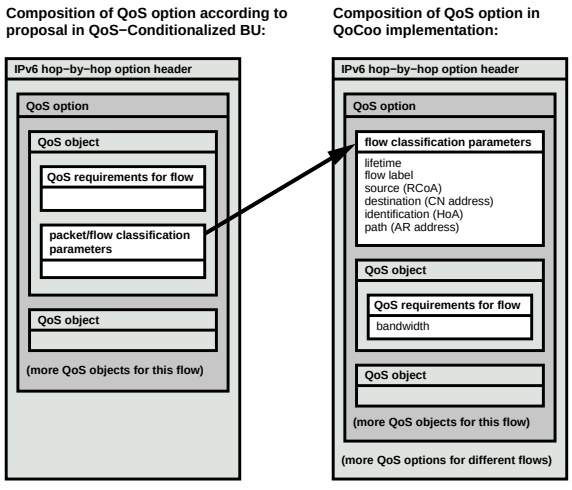
8

Open Issues: Expressing QoS Options and Objects

- QoS objects not initially intended to express/merge requirements for ”desirable/acceptable” (for ”up-/downstream”)
- QoS-cond BU draft uses two (or more) QoS objects to express this
  - Not an optimal solution, as flow identification is replicated
- Solution: Extract the flow identification into the QoS option itself
  - Reduces overhead; gives clearer semantics to the QoS signaling messages
- Gives a definition of a possible ”flow classification parameters”
  - Referred to in related drafts, but always empty ”for further study”

9

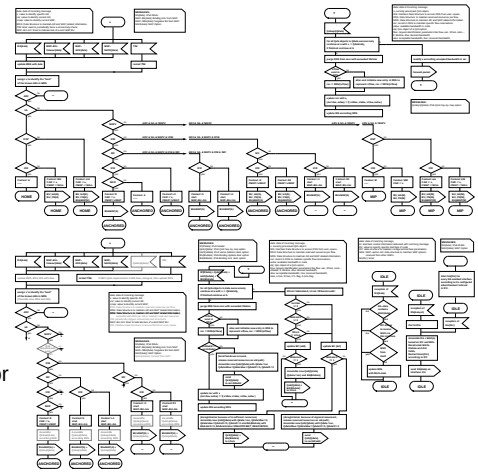
Open Issues: QoS Options/Objects Reorganization



10

Formal Description as Basis for Implementation Design

- Informal description restated more formally
- Flow chart
- Provides basis for implementation design
- Aim is NOT to prove correctness of underlying protocols or implementation



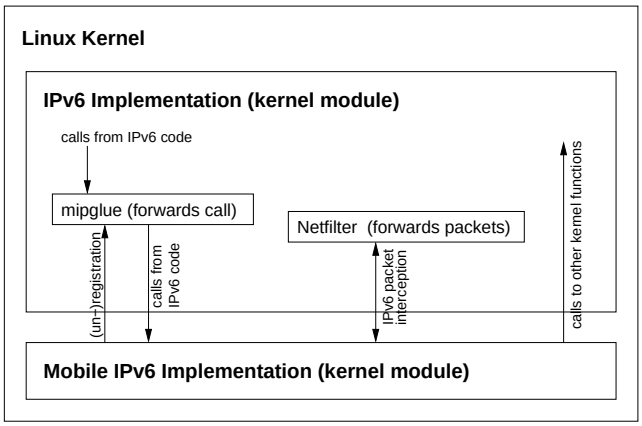
11

Implementation Background

- Implementation based on ”Mobile IP for Linux” (MIPL), Helsinki University of Technology
  - Kernel-space implementation
- Ideas for extensions to HMIPv6
  - Leverage basic agent functionalities and data structures
  - Restrict changes to the MIPL modules; do not change interfaces to kernel
  - New or modified data structures:
    - Table for available MAPs (modified AR table)
  - Main changes happen to the handover processing in the MN
  - Router discovery code extended for MAP discovery (user-space)

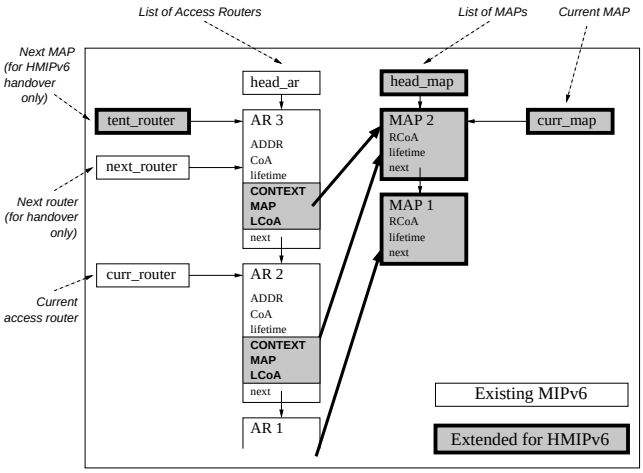
12

Implementation Background: Original MIPL General Architecture



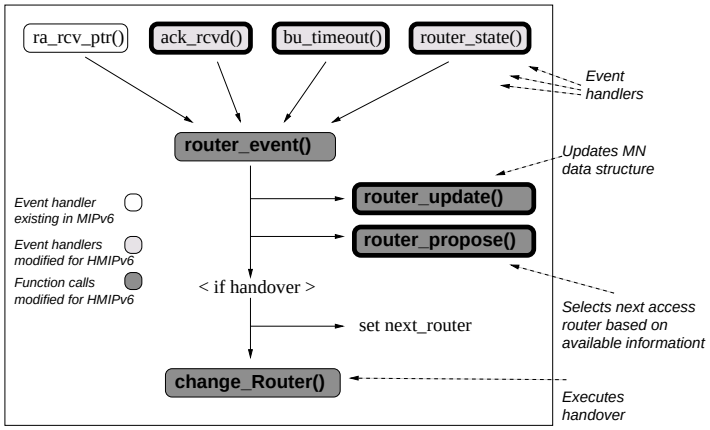
13

Implementation: MN data structure



14

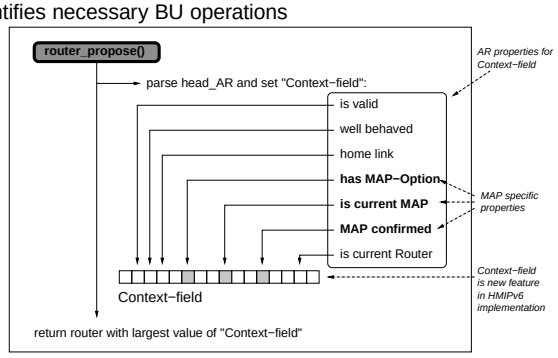
Implementation: MN event handling



15

Implementation: MN "Context field" Processing

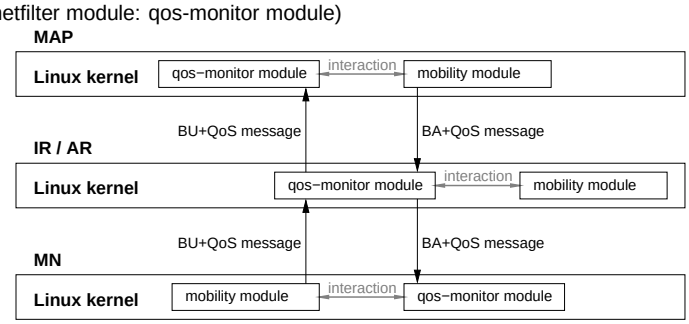
- MN's guideline to evaluate available ARs
  - Context field summarizes relevant information per AR
  - Simple to evaluate, yields priority in which to attempt handoff to AR
- Also identifies necessary BU operations



16

QoS-Cond. BU Implementation

- QoS-cond. BU implementation based on this HMIPv6 implementation
- MAP, intermediate router (IR), and MN are extended with resource data structure
- Registration messages are extended with QoS options (mobility module)
- Received and forwarded registrations are examined for QoS options (netfilter module: qos-monitor module)



17

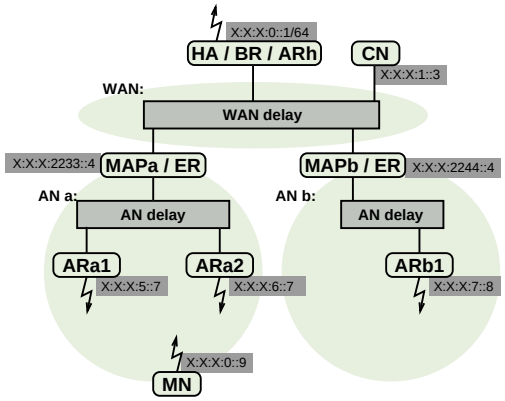
QoS-Cond. BU Implementation (contd.)

- MN is configured with a certain required bandwidth (set as parameter)
  - Whenever BU is to be sent, check for desired reservations is performed
- IRs intercept and forward QoS options (qos-monitor module)
  - Perform simple check on available resources; may modify the QoS options; record the permitted reservations in database
  - Reinject BU packet into the kernel's packet processing
- MAP:
  - Receives QoS options like IR: check QoS option,check available resources, modifies database entries (qos-monitor module)
  - HA stack conditionalizes binding on data base result (mobility module)
- HA not modified, QoS processing only implemented in access network

18

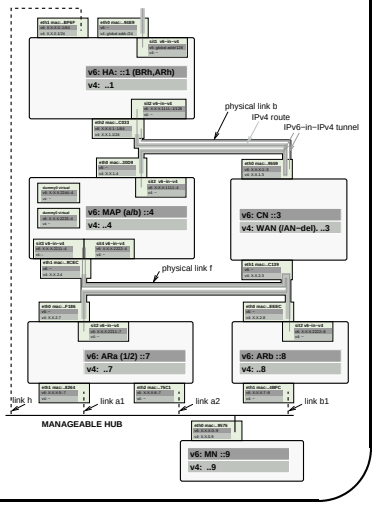
Testing: Logical Testbed Setup

- Testbed used to validate prototype implementation and perform measurements
- Test scenarios defined and tested
  - Examples: Simple local and regional handover, with or without adequate amounts of resources,...
  - Implementation finally passed all tests ;-)



Testing: Actual Testbed Setup

- Goal: Provide network environment with significant properties of real networks (foreign access networks, delay, loss)
- Problem: Limited in available machines, and IPv6 tools
  - Machines are used for several purposes
  - No applicable WAN simulator available for IPv6
    - IPv4 WAN simulator used;
    - IPv6 packets are tunnelled via IPv4
- Movement simulated by using ”manageable hub” or netfilter based ”virtual switch”



Experiments: Registration Latency

- Metric: Registration latency as duration of entire registration process (BU to BA)
  - Tcpdump used to monitor registration messages
  - Movement controlled via manageable hub; WAN delay: 30ms, AN delay: 5ms; 10 experiments per scenario

	minimum	maximum	average	s^2
MIPv6 HA :	70.2 ms	70.4 ms	70.26 ms	0.007
local HMIPv6 HO:	10.3 ms	10.5 ms	10.42 ms	0.004
QoS-Cond. HO:	11.4 ms	12.2 ms	11.8 ms	0.033
reg. HMIPv6 HO:	81.1 ms	82.8 ms	81.4 ms	0.257

- Registration latency for local handover could be significantly reduced

21

Experiments: UDP Packet Loss

- Metric: Downstream packet loss per HO for 10ms emission rate
  - Udp6traffic used to send, receive, and measure losses between CN and MN
  - Movement controlled via manageable hub; detection based on RtAdv period of 1.5 sec; WAN delay: 30ms, AN delay: 5ms; 100 experiments per scenario

	minimum	maximum	average	s^2
MIPv6 HO:	18	163	104.4	1354
local HMIPv6 HO:	13	155	87.6	1222
QoS-Cond. HO:	18	158	92.1	1569
reg. HMIPv6 HO:	19	165	97.9	1653

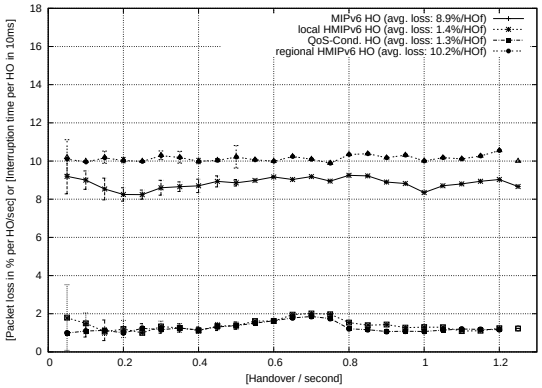
- Regarding packet loss, no relevant benefits could be achieved

22

Experiments: Ideal Link Layer Trigger Case (1)

- Metric: Downstream packet loss for different handover rates

- Movement controlled by udp6traffic via virtual switch
- Movement detection reduced to less than 1ms
- WAN del. 30ms, AN del. 5ms

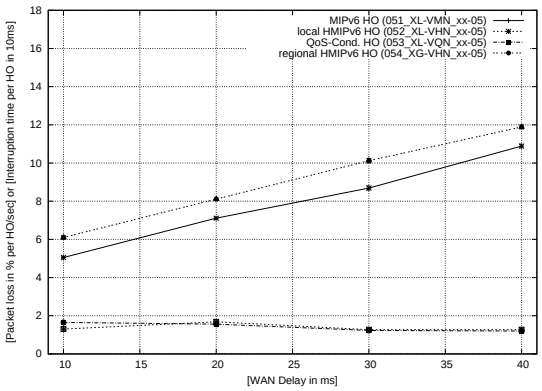


23

Experiments: Ideal Link Layer Trigger Case (2)

- Downstream packet loss for different transmission delays between MAP and CN (WAN delays)

- Movement controlled by udp6traffic via virtual switch
- Movement detection reduced to less than 1ms
- AN del. 5ms



24

Summary and Open Issues

- Underlying concepts (HMIPv6 and QoS-cond. BU) have been analyzed and re-examined in a more formal description
- Open issues and possible solutions have been identified:
  - Release of stale reservations; Composition of QoS option reorganized
- Prototypical Implementation called QoCoo has been realized and setup in a testbed
- Key capabilities of underlying concepts have been successfully tested
- Performance experiments showed that the packet loss for micro movements can be significantly reduced even in combination with QoS signaling
- Remaining open issues:
  - End-to-end signaling? Multiple, independent flows via different paths?

25

Appendix E

Acronyms

2G Second Generation

3G Third Generation

AN Access Network

AP Access Point

API Application Program Interface

AR Access Router

ARP Address Resolution Protocol

BA Binding Acknowledgement

BC Binding Cache

BdR Border Router

BR Binding Request

BU Binding Update

BUL Binding Update List

CIDR Classless Inter Domain Routing

CN Corresponding Node

CoA Care-of Address

DAD Duplicate Address Detection

DHCP Dynamic Host Configuration Protocol

DiffServ Differentiated Services

DLL Data Link Layer

DoS Denial of Service

DSCP DiffServ Code Point

ER Edge Router

FTP File Transfer Protocol

GSM Global System for Mobile Communication

HA Home Agent

HAO Home Address Option

HMIPv6 Hierarchical Mobile IPv6

HO Handover

HOf handover frequency

HoA Home Address

ICI Interface Control Information

ICMP Internet Control Message Protocol

ICMPv6 ICMP version 6

ID identification

IDU Interface Data Unit

IETF Internet Engineering Task Force

IntServ Integrated Services

IP Internet Protocol

IPng IP Next Generation

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IR Intermediate Router

LAN Local Area Network

LCoA Local Care-of Address

LXR Linux Cross Reference

MAC Medium Access Control

MAP Mobile Anchor Point

MAO MAP Option

MD Movement Detection

MHC Manageable Hub Configuration

MIP Mobile IP

MIPL Mobile IP for Linux

MIPv6 Mobile IPv6

MN Mobile Node

NbAdv Neighbor Advertisement

ND Neighbor Discovery

NbSol Neighbor Solicitation

NIC Network Interface Card

NSIS Next Steps in Signaling

OS Operating System

PDU Protocol Data Unit

QoS Quality of Service

QoCoo QoS-Conditionalized Handover

QCB QoS-Conditionalized Binding Update

RA Router Advertisement

radvd router advertisement daemon

RCoA Regional Care-of Address

RD Redirect

RDS Resource Data Structure

rfc request for comment

RH Routing Header

RS Router Solicitation

RtAdv Router Advertisement

RtSol Router Solicitation

RSVP Resource Reservation Protocol

RTT round trip time

SAP Service Access Point

SDL Specification and Description Language

SNMP Small Network Management Protocol

SP Service Primitive

TBA To Be Added

TCP Transmission Control Protocol

TLV Type-Length-Value

UDP User Datagram Protocol

VoIP Voice over IP

WAN Wide Area Network

WLAN Wireless Local Area Network