7 References

[1] A. O. da Silva, S. Colcher, "Evolução da Otimização do Handoff no Mobile IP", MCC 27/08, Pontifícia Universidade Católica, Departamento de Informática, July 2008.

[2] A. O. da Silva, L. F. G. Soares, S. Colcher, "Aprimoramentos no RSVP para o Mobile IPv6", MCC 28/08, Pontifícia Universidade Católica, Departamento de Informática, July 2008.

[3] A. O. da Silva, M. Endler, S. Colcher, "Otimização do Handover na Camada de Rede (L3) utilizando o Media Independent Handover (MIH)", MCC 29/08, Pontifícia Universidade Católica, Departamento de Informática, July 2008.

[4] J. Sachs, "A Generic Link Layer for Future Generation Wireless Networking", IEEE, 2003, p. 834-838.

[5] J. Stein, "Survey of 802.21 Media Independent Handover Services", April 2006, based on "Draft IEEE Standard for Local and Metropolitan Area Networks: Media Independent Handover Services", IEEE P802.21/ D01.00, March 2006.

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

[7] D. Johnson, C. Perkins, J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

[8] H. Soliman, C. Castelluccia, K. El Malki, L. Bellier, "Hierarchical Mobile IPv6 Mobility Management (HMIPv6)", RFC 4140, August 2005.

[9] R. Koodli, "Fast Handovers for Mobile IPv6", RFC 4068, July 2005.

[10] H. Jung et al., "Fast Handover for Hierarchical MIPv6 (F-HMIPv6)", draft-jung-mobileip-fastho-hmipv6-04, Internet draft, June 2004.

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

[12] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, December 1998.

[13] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C. and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[14] Wright, D.J., "Maintaining QoS During Handover Among Multiple Wireless Access Technologies", International Conference on Mobile Business, Toronto, 2007.

[15] Velayos, H., Karlsson, G., "Techniques to Reduce IEEE 802.11b MAC Layer Handover Time", Laboratory for Communication Networks, IMIT, ISSN 1651-7717, 2003.

[16] Lo, S., Lee, G., Chen, W., “Architecture for Mobility and QoS Support in All-IP Wireless Networks”, IEEE Journal, Vol. 22, No. 4, 2004.

[17] Maltz, D. A., and Bhagwat, P., MSOCKS: An Architecture for Transport Layer Mobility, IEEE INFOCOM, 1998.

[18] Zandy, V. C., and Miller, B. P., Reliable Network Connections, ACM MOBICOM, 2002.

[19] Snoeren, A. C., and Balakrishinan, H., An End-to-End Approach to Host Mobility, ACM MOBICOM, 2000.

[20] Moskowitz, R., and Nicander, P., Host Identity Protocol (HIP) Architecture, IETF, RFC 4423, 2006.

Appendix A
Mobility manager handlers

This appendix details the logic and code of the handlers of the MM of the GMA entities. There are three types of handlers: (i) trigger handler; (ii) time handler; and (iii) message handler. The first handles the L2 triggers fired by the link layer and notified as MIES events. The second handles the system timer scheduled events. And the latter, handles the signaling messages of the GMP.

These handlers use the functional procedures briefly described as follows:

CHECK_PARTNERSHIP(lrs, hrs)

Description:	Checks whether or not there is a partnership between the LRS and the HRS of the MN. This checking is done to determine if the MN is allowed to move to the GAN of the MN.
Entity:	LRS

CHECK_DB_RECORD(hrs, lrs, mn, coa)

Description:	Checks whether or not there is a record for the (hrs, lrs, mn, coa) key in the database of the LRS.
Entity:	LRS

CHECK_MBC_RECORD(hrs, lrs, mn, coa)

Descrip-	Checks whether or not there is a record for the (hrs, lrs, mn, coa)

tion: key in the mobility binding cache of the AR.

Entity: AR

CHECK_TEMPORARY_CHECKING_RECORD(mn, coa)

Description: Checks whether or not there is a temporary checking record for the (mn, coa) pair in the temporary table.

Entity: LRS

CHECK_TEMPORARY_REGISTERING_RECORD(hrs, lrs, mn, coa)

Description: Checks whether or not there is a temporary registering record for the (hrs, lrs, mn, coa) key in the temporary table.

Entity: AR, LRS

COA_BELONGS_TO_HRS_GAN(coa)

Description: Checks whether or not the CoA belongs to the GAN of the HRS.

Entity: LRS

CREATE_CN_ADDRESS_LIST(void)

Description: Creates the address list of the CNs currently communicating with the MN.

Entity: MN

CREATE_DB_RECORD(hrs, lrs, mn, coa, validity)

Description: Creates a record for the MN and writes it to the database of the LRS.

Entity: LRS

CREATE_MBC_RECORD(hrs, lrs, mn, coa, validity)

Description: Creates a record for the MN and writes it to the mobility binding cache.

Entity: AR

CREATE_TEMPORARY_CHECKING_RECORD(mn, coa, ar, validity)

Description: Creates a temporary checking record for the request done by the AR and writes it to the temporary table.

Entity: LRS

CREATE_TEMPORARY_REGISTERING_RECORD(hrs, lrs, mn, coa, validity)

Description: Creates a temporary registering record for the MN and writes it to the temporary table.

Entity: AR, LRS

DELETE_DB_RECORD(hrs, lrs, mn, coa, validity)

Description: Deletes the record of the MN from the database of the LRS.

Entity: LRS

DELETE_MBC_RECORD(hrs, lrs, mn, coa, validity)

Description: Deletes the record of the MN from the mobility binding cache.

Entity: AR

DELETE_TEMPORARY_CHECKING_RECORD(mn, coa, ar)

Description: Deletes the temporary checking record of the AR from the temporary table.

Entity: LRS

DELETE_TEMPORARY_REGISTERING_RECORD(hrs, lrs, mn, coa, validity)

Description: Deletes the temporary registering record of the MN from the temporary table.

Entity: AR, LRS

GET_AR_OF_POA(selectedPoA)

Description: Returns the address of the AR that currently offers service to the PoA.

Entity: LRS

GET_LRS_OF_CN(cn)

Description: Returns the address of the LRS that currently offers service to the CN.

Entity: LRS

GET_LRS_OF_AR(ar)

Description: Returns the address of the LRS that currently offers service to the AR.

Entity: AR

GET_LRS_OF_COA(coa)

Description: Returns the address of the LRS that currently offers service to the CoA.

Entity: LRS

GET_LRS_OF_POA(selectedPoA)

Description: Returns the address of the LRS that currently offers service to the AR of the PoA.

Entity: LRS

MN_IS_ALLOWED_AT_LRS_GAN(lrs, mn)

Description: Checks whether or not the MN is allowed at the GAN of the LRS.

Entity: LRS

POA_ID_BELONGS_TO_GAN(selectedPoA-ID)

Description: Checks whether or not the PoA-ID belongs to the current GAN.

Entity: AR/LRS

PREPARE_HANDOVER_CONTEXT(hrs, lrs, mn, coa, cnlist, hc)

Description: Prepares the handover context of the MN. Reads the record of the MN and the records of all the CNs of the CNLIST that are mobile nodes from the mobility binding cache. Appends the addresses of all the stationary CNs to the context.

Entity: AR

READ_DB_RECORD(hrs, lrs, mn, coa, validity)

Description: Reads the record for the MN from database of the LRS.

Entity: LRS

READ_MBC_RECORD(hrs, lrs, mn, coa, validity)

Description: Reads the record of the MN from the mobility binding cache.

Entity: AR

READ_TEMPORARY_CHECKING_RECORD(mn, coa, ar)

Description: Reads temporary checking record of the AR from the temporary table.

Entity: LRS

READ_TEMPORARY_REGISTERING_RECORD(hrs, lrs, mn, coa, validity)

Description: Reads the temporary record of the MN from the temporary table.

Entity: AR, LRS

REGISTER_SUCCESS_HANDLER(statusFlag)

Description: Executes the procedure related to a successful registering at the GAN.

Entity: MN

REGISTER_ERROR_HANDLER(statusFlag)

Description: Executes the procedure related to an unsuccessful registering at the GAN.

Entity: MN

SELECT_POA_FROM_CANDIDATES(list_of_the_candidate_poas)

Description: Selects the best PoA to handover from a list of candidate PoAs.

Entity: MN

SEND(destination_address, source_address, message_pdu)

Description: Sends a GMP message.

Entity: MN, AR and LRS

SEND_BINDING_NOTIFICATION(mn, coa, cnlist)

Description: Starts the anticipated binding notification procedure to notify the ARs of the CNs in behalf of the MN during handover process.

Entity: AR

SEND_REGISTRATION_REQUEST(mn, coa)

Description: Starts the anticipated registration request procedure in behalf of the MN during handover process.

Entity: AR

SET_TIMER_WAKE-UP_TRIGGER(awake_after)

Description: Sets the timer wake-up trigger to be fired after *awake_after* seconds.

Entity: MN

START_DOWNSTREAM_FLOW_BUFFERING_MECHANISM(void)

Description: Starts the downstream flow buffering mechanism to store in-flight

tion: packets during handover process.

Entuty: AR

START_DOWNSTREAM_FLOW_FORWARDING_MECHANISM(void)

Description: Starts the downstream flow forwarding mechanism to forward in-flight packets during handover process.

Entity: AR

HO_ERROR_PROCEDURE(void)

Description: Starts the handover error procedure to execute the appropriate counter measure against the problem.

Entity: MN, AR and LRS

START_DOWNSTREAM_FLOW_BUFFERING_MECHANISM(buffer_length)

Description: Starts the downstream flow buffering mechanism to store in-flight packets during handover process.

Entity: AR

START_DOWNSTREAM_FLOW_FORWARDING_MECHANISM(mn, coa)

Description: Starts the downstream flow forwarding mechanism to forward all in-flight packets buffered during handover process.

Entity: AR

START_UPSTREAM_FLOW_BUFFERING_MECHANISM(buffer_length)

Description: Starts the upstream flow buffering mechanism to store in-flight packets during handover process.

Entity: MN

START_UPSTREAM_FLOW_FORWARDING_MECHANISM(void)

Description: Starts the upstream flow forwarding mechanism to forward all in-flight packets buffered during handover process.

Entity: MN

STOP_DOWNSTREAM_FLOW_BUFFERING_MECHANISM(void)

Description: Stops the downstream flow buffering mechanism used to store in-flight packets during handover process.

Entity: AR

STOP_DOWNSTREAM_FLOW_FORWARDING_MECHANISM(void)

Description: Stops the downstream flow forwarding mechanism used to forward in-flight packets during handover process.

Entity: AR

STOP_UPSTREAM_FLOW_BUFFERING_MECHANISM(void)

Description: Stops the upstream flow buffering mechanism used to store in-flight packets during handover process.

Entity: MN

UPDATE_DB_RECORD(hrs, lrs, mn, coa, validity)

Description: Updates the record of the MN in the database of the LRS.

Entity: LRS

UPDATES_MBC_RECORD(hrs, lrs, mn, coa, validity)

Description:	Updates the record of the MN in the mobility binding cache.
Entity:	AR

A1 – MN_L2-Trigger_Handler(Trigger)

```
CASE Trigger.Event IN
        L2-UP:  /* registers MN at the GAN or finishes L3-handover */
                SELECTED_POA_ID=Trigger.PoA-ID

                IF oldPoA-ID = NULL
                THEN    /* registers MN at the GAN */
                        SEND(GMAP_LRS, GMAP_MN, C2N_RegistrationRequest(
                                newPoA-ID=SELECTED_POA_ID, hrs=HRS_ADDR,
                                mn=GMAP_MN))
                ELSE    /* finishes L3-Handover */
                        CN_ADDRESS_LIST=CREATE_CN_ADDRESS_LIST()
                        SEND(GMAP_AR, GMAP_MN, C2N_HandoverFinish(
                                oldPoA-ID=OLD_POA_ID,
                                newPoA-ID=SELECTED_POA_ID,
                                hrs=HRS_ADDR, mn=GMAP_MN,
                                cnlist=CN_ADDRESS_LIST))
                FI

        L2-GoingDown:   /* initiates L3-Handover */
                SELECTED_POA_ID=SELECT_POA_FROM_CANDIDATES(
                                        poa-list=Trigger.PoAList)

                CN_ADDRESS_LIST=CREATE_CN_ADDRESS_LIST()

                SEND(GMAP_AR, GMAP_MN,
                        C2N_HandoverInitiate(oldPoA-ID=CURRENT_POA_ID,
                        newPoA-ID=SELECTED_POA_ID, hrs=HRS_ADDR,
                        mn=GMAP_MN, cnlist=CN_ADDRESS_LIST))

        L2-Down:        /* starts upstream flow buffering mechanism */
                START_UPSTREAM_FLOW_BUFFERING_MECHANISM(
                                        bufferLength=MAX_BUFFER_LENGTH)
ESAC
```

A2 – MN_Message_Handler(Message)

```
CASE Message.Type IN
        N2C_RegistrationResponse:        /* checks registering status */
                STATUS_FLAG=Message.statusFlag

                CASE STATUS_FLAG IN
                        APPROVED:    /* successful registerig */
                                SET REGISTERING_STATE=APPROVED
                                REGISTERING_SUCCESS_PROCEDURE(
                                        statusFlag=APPROVED)
                        DENIED_BY_AR:   /* unsuccessful registering: denied byAR */
                                SET REGISTER_STATE=DENIED_BY_AR
                                REGISTERING_ERROR_PROCEDURE(
                                        statusFlag=DENIED_BY_AR)
                        DENIED_BY_LRS:   /* unsuccessful registering: denied by LRS */
                                SET REGISTER_STATE=DENIED_BY_LRS
                                REGISTERING_ERROR_PROCEDURE(
                                        statusFlag=DENIED_BY_LRS)
                        DENIED_BY_HRS:   /* unsuccessful registering: denied by HRS */
                                SET REGISTER_STATE=DENIED_BY_HRS
                                REGISTERING_ERROR_PROCEDURE(
                                        statusFlag=DENIED_BY_HRS)
                ESAC

        N2C_HandoverAcknowledge:        /* checks handover status */
                STATUS_FLAG=Message.statusFlag

                CASE STATUS_FLAG IN
                        OK:     /* successful handover */
                                STOP_UPSTREAM_FLOW_BUFFERING_MECHANISM()
                        FAIL:   /* unsuccessful handover */
                                HO_ERROR_PROCEDURE()
                ESAC
ESAC
```

A3 – MN_Timer_Handler(Time)

```
IF Time.ElapsedTime >= REFRESH_PERIOD
        THEN        /* starts refresh cycle */
                CN_ADDRESS_LIST=CREATE_CN_ADDRESS_LIST()
                SEND(GMAP_LRS, GMAP_MN, C2N_RegistrationRefresh(
                        currentPoA-ID=SELECTED_POA, hrs=HRS,
                        mn=GMAP_MN, cnlist=CN_ADDRESS_LIST,
                        validity=DEFAULT_EXPIRING_TIME))
                SET_TIMER_WAKE-UP_TRIGGER(awake_after=REFRESH_PERIOD)
FI
```

A4 – AR_Message_Handler(Message)

```
CASE Message.Type IN
    C2N_RegistrationRequest:        /* requests the registering of the MN at the GAN */
        SELECTED_POA_ID=Message.newPoA-ID
        HRS=Message.hrs
        MN=Message.mn

        IF POA_ID_BELONGS_TO_GAN(selectedPoA-ID=SELECTED_POA_ID)
        THEN  /* requests the registering of the MN at the LRS of the GAN */
            AR=MY_EXTERNAL_IP_ADDRESS
            LRS=GET_LRS_OF_AR(ar=AR)
            CREATE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                    lrs=LRS, mn=MN, coa=AR,
                                    validity=TEMP_EXPIRING_TIME)
            SEND(LRS, AR, N2N_RegistrationRequest(hrs=HRS, lrs=LRS,
                                                  mn=MN, coa=AR))
        ELSE  /* sends registering failure status */
            SEND(MN, GMAP_LRS, N2C_RegistrationResponse(
                    statusFlag=DENIED_BY_AR))
        FI

    N2N_RegistrationResponse:       /* checks registering status */
        STATUS_FLAG=Message.statusFlag
        HRS=Message.hrs
        LRS=Message.lrs
        MN=Message.mn
        COA=Message.coa
        AR=MY_EXTERNAL_IP_ADDRESS

        IF COA = AR
        THEN  /* accepts message only if this AR is the COA of the MN */
          IF CHECK_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                              lrs=LRS, mn=MN, coa=COA)
          THEN  /* temporary registering record still valid */
            CASE STATUS_FLAG IN
            APPROVED:  /* successful registering: creates MBC record for MN */

        READ_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                lrs=LRS, mn=MN, coa=COA)
```

```
                DELETE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                lrs=LRS, mn=MN, coa=COA)
                        CREATE_MBC_RECORD(hrs=HRS, lrs=LRS, mn=MN,
                                coa=COA, validity=DEFAULT_EXPIRING_TIME)
                    DENIED_BY_LRS:   /* unsuccessful registering: denied by LRS */

                DELETE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                lrs=LRS, mn=MN, coa=COA)
                    DENIED_BY_HRS:   /* unsuccessful registering: denied by HRS */

                DELETE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                lrs=LRS, mn=MN, coa=COA)
                    ESAC

                    SEND(MN, GMAP_LRS,
                            N2C_RegistrationResponse(statusFlag=STATUS_FLAG))
                ELSE   /*temporary registering record expired:  ignores message */
                FI
            ELSE   /* ignores message */
            FI

        C2N_RegistrationRefresh: /* starts the refresh cycle of the MN */
            CURRENT_POA_ID=Message.currentPoA-ID
            HRS=Message.hrs
            MN=Message.mn
            CN_ADDRESS_LIST=Message.cnlist
            VALIDITY=Message.validity
            AR=MY_EXTERNAL_IP_ADDRESS
            LRS=GET_LRS_OF_AR(ar=AR)

            IF POA_ID_BELONGS_TO_GAN(selectedPoA-ID=CURRENT_POA_ID)
            THEN   /* checks the current mobility binding cache record */
                IF CHECK_MBC_RECORD(hrs=HRS, lrs=LRS, mn=MN, coa=AR)
                THEN   /* updates the record of the MN */
                        UPDATE_MBC_RECORD(hrs=HRS, lrs=LRS,
                                        mn=MN, coa=AR, validity=VALIDITY)
                        SEND(LRS, AR, N2N_RegistrationRefresh(hrs=HRS,
                                mn=MN, coa=AR, cnlist=CN_ADDRESS_LIST,
                                validity=VALIDITY))
```

```
                ELSE    /* ignores refresh cycle */
                FI
        ELSE    /* ignores refresh cycle */
        FI

N2N_BindingNotification:        /* refreshes or creates a record for the MN */
        HRS=Message.hrs
        LRS=Message.lrs
        MN=Message.mn
        COA=Message.coa
        VALIDITY=Message.validity

        /* checks the current mobility binding cache record */
        IF CHECK_MBC_RECORD(hrs=HRS, lrs=LRS, mn=MN, coa=COA)
        THEN    /* updates the record of the MN */
                UPDATE_MBC_RECORD(hrs=HRS, lrs=LRS, mn=MN,
                                coa=COA, validity=VALIDITY)
        ELSE    /* creates the record of the MN */
                CREATE_MBC_RECORD(hrs=HRS, lrs=LRS, mn= MN,
                                coa=COA, validity=VALIDITY)
        FI

C2N_HandoverInitiate:   /* starts the L3 handover process in behalh of the MN */
        OLD_POA_ID=Message.oldPoA-ID
        NEW_POA_ID=Message.newPoA-ID
        HRS=Message.hrs
        MN=Message.mn
        CN_ADDRESS_LIST=Message.cnlist
        AR=MY_EXTERNAL_IP_ADDRESS
        LRS=GET_LRS_OF_AR(ar=AR)

        IF NOT POA_ID_BELONGS_TO_GAN(selectedPoA-ID=NEW_POA_ID)
        THEN    /* checks the current mobility binding cache record */
                IF CHECK_MBC_RECORD(hrs=HRS, lrs=LRS, mn=MN, coa=AR)
                THEN    /* continue the L3 handover process: anticipated mode */
                        PREPARE_HANDOVER_CONTEXT(hrs=HRS, lrs=LRS,
                                mn=MN, coa=AR, cnlist=CN_ADDRESS_LIST,
                                 hc=HAND_OVER_CONTEXT)
                        SEND(LRS, AR, N2N_HandoverIndication(
                                                mode=ANTICIPATED,
```

```
                                                    oldPoA-ID=OLD_POA_ID,
                                                    newpoa-id=NEW_POA_ID,
                                                    mn=MN, coa=AR,
                                                    cnlist=CN_ADDRESS_LIST,
                                                    hc=HANDOVER_CONTEXT))

                        /* starts to buffer downstream flow in behalf of the MN */
                        START_DOWNSTREAM_FLOW
                                _BUFFERING_MECHANISM(
                                buffer_length=MAX_BUFFER_LENGTH)

                ELSE    /* no MN record in the mbc: ignores L3 handover initializa-
tion */
                FI
        ELSE    /* no AR changing:  ignores L3 handover initialization */
        FI

C2N_HandoverFinish:     /* finishes the L3 handover process in behalh of the MN */
        OLD_POA_ID=Message.oldPoA-ID
        NEW_POA_ID=Message.newPoA-ID
        HRS=Message.hrs
        MN=Message.mn
        CN_ADDRESS_LIST=Message.cnlist
        AR=MY_EXTERNAL_IP_ADDRESS
        LRS=GET_LRS_OF_AR(ar=AR)

        IF NOT POA_ID_BELONGS_TO_GAN(selectedPoA-ID=NEW_POA_ID)
        THEN    /* checks the current mobility binding cache record */
                IF CHECK_MBC_RECORD(hrs=HRS, lrs=LRS, mn=MN, coa=AR)
                THEN    /* finishes the L3 handover process: anticipated mode */
                        /* starts to forward the downstream flow buffered */
                        START_DOWNSTREAM_FLOW
                                _FORWARDING_MECHANISM(mn=MN)

                        /* NAR acks the end of the handover of the MN */
                        SEND(MN, AR,
                                N2C_HandoverAcknowledge(statusFlag=OK,
                                                        mn=MN))

                ELSE    /* no MN record in the mbc: starts reactive mode */
```

```
                    SEND(LRS, AR, N2N_HandoverIndication(
                                        mode=REACTIVE,
                                        oldPoA-ID=OLD_POA_ID,
                                        newpoa-id=NEW_POA_ID,
                                        mn=MN, coa=AR,
                                        cnlist=CN_ADDRESS_LIST,
                                        hc=NULL))

                FI
            ELSE    /* no AR changing:  ignores L3 handover finishing */
            FI

        N2N_HandoverIndication:/* sends/receives handover context of the MN */
            MODE=Message.mode
            OLD_POA_ID=Message.oldPoA-ID
            NEW_POA_ID=Message.newPoA-ID
            MN=Message.mn
            COA=Message.coa
            CN_ADDRESS_LIST=Message.cnlist
            HANDOVER_CONTEXT=Message.hc

            AR=MY_EXTERNAL_IP_ADDRESS

            CASE MODE IN
                ANTECIPATED:/* anticipated mode */
                    IF POA_ID_BELONGS_TO_GAN(selectedPoA-ID=
                                                        NEW_POA_ID)
                    THEN        /* receives handover context of the MN */
                        UPDATE_MBC_FROM_HC(mn=MN, coa=AR,
                                            hc=HANDOVER_CONTEXT)

                                /* notifies PAR to start the forwarding mechanism */
                        SEND(COA, AR, N2N_HandoverAcknowledge(
                                            Mode=MODE,
                                            statusFlag=OK,
                                            mn=MN, coa=COA, hc=NULL))

                                /* anticipates MN registering at the GAN */
                        SEND_REGISTRATION_REQUEST(mn=MN, coa=AR)
```

```
                        /* Notifies the ARs of the CNs to update their bindings */
                        SEND_BINDING_NOTIFICATION(mn=MN, coa=AR,
                                                  cnlist=CN_ADDRESS_LIST)

                    ELSE        /* error:  ignores message */
                    FI
                REACTIVE:               /* reactive mode */
                    IF POA_ID_BELONGS_TO_GAN(
                                                  selectedPoA-ID=OLD_POA_ID)
                    THEN        /* receives handover context request from NAR */
                        READ_MBC_RECORD(hrs=HRS, lrs=LRS,
                                mn=MN, coa=COA, validity=VALIDITY)
                        PREPARE_HANDOVER_CONTEXT(hrs=HRS, lrs=LRS,
                                mn=MN, coa=AR, cnlist=CN_ADDRESS_LIST,
                                 hc=HAND_OVER_CONTEXT)

                                /* sends handover context to NAR */
                        SEND(COA, AR, N2N_HandoverAcknowledge(
                                                  Mode=MODE,
                                                  statusFlag=OK,
                                                  mn=MN, coa=COA, cnlist=NULL,
                                                   hc=HANDOVER_CONTEXT))
                    ELSE        /* error:  ignores message */
                    FI
            ESAC

        N2N_HandoverForward:    /* requests/confirms the forwarding of in-flight packets */
            MODE=Message.mode
            MN=Message.mn

            AR=MY_EXTERNAL_IP_ADDRESS

            CASE MODE IN
                ANTICIPATED:    /* anticipated mode: message received by PAR */
                    NAR=AR
                    PAR=Message.source_address

                    START_DOWNSTREAM_FLOW
                                _BUFFERING_MECHANISM(mn=MN)
```

```
                SEND(PAR, NAR, N2N_HandoverAcknowledge(
                                                Mode=MODE,
                                                statusFlag=FORWARD,
                                                mn=MN, coa=NAR,
                                                cnlist=NULL,
                                                hc=NULL))

            REACTIVE:       /* reactive mode: message received by PAR */
                PAR=AR
                NAR=Message.source_address

                SEND(NAR, PAR, N2N_HandoverAcknowledge(
                                                Mode=MODE,
                                                statusFlag=FORWARD,
                                                mn=MN, coa=NULL,
                                                cnlist=NULL,
                                                hc=NULL))

                START_DOWNSTREAM_FLOW
                        _FORWARDING_MECHANISM(mn=MN, coa=NAR)

        ESAC

N2N_HandoverAcknowledge:    /* checks handover acknowledge status */
        MODE=Message.mode
        STATUS_FLAG=Message.statusFlag
        MN=Message.mn
        COA=Message.coa
        CN_ADDRESS_LIST=Message.cnlist
        HANDOVER_CONTEXT=Message.hc

        AR=MY_EXTERNAL_IP_ADDRESS

        CASE MODE IN
            ANTICIPATED:    /* anticipated mode */

                CASE STATUS_FLAG IN
                        OK:     /* handover context received by NAR */
                                /* PAR sends forward request */
                                PAR=AR
```

```
                                NAR=COA
                                SEND(NAR, PAR,
                                        N2N_HandoverForward(
                                        mode=ANTICIPATED,
                                        mn=MN))

                        FORWARD:    /* forward notification received by NAR */
                                    /* PAR starts forwarding mechanism */
                                START_DOWNSTREAM_FLOW
                                _FORWARDING_MECHANISM(mn=MN,
coa=COA)
                    ESAC

                REACTIVE:       /* reactive mode */
                    CASE STATUS_FLAG IN
                        OK:     /* handover context request received by PAR */
                                /* NAR sends forward request */
                                NAR=AR
                                PAR=COA
                                SEND(PAR, NAR,
                                    N2N_HandoverForward(mode=REACTIVE,
                                        mn=MN))

                        FORWARD:    /* forward request received by NAR */
                                    /* PAR starts forwarding mechanism */
                                /* buffering mechanism is not started in this mode */
                                /* NAR acks the end of the handover of the MN */
                                SEND(MN, AR,
                                    N2C_HandoverAcknowledge(statusFlag=OK,
                                        mn=MN))
                    ESAC
ESAC
```

A5 – LRS_Message_Handler(Message)

```
CASE Message.Type IN
    N2N_RegistrationRequest:        /* requests the registering of the MN at the GAN */
        HRS=Message.hrs
        LRS=Message.lrs
        MN=Message.mn
        COA=Message.coa

        IF CHECK_LRS_IS_HRS(hrs=HRS)
        THEN  /* lrs is hrs: checks GAN */
            IF COA_BELONGS_TO_HRS_GAN(coa=COA)
            THEN  /* mn is at the GAN of the HRS: registers MN at the GAN */
                CREATE_DB_RECORD(hrs=HRS, lrs=LRS, mn=MN,
                    coa=COA, validity=DEFAULT_EXPIRING_TIME)
                SEND(COA, MY_LRS_ADDR, N2N_RegistrationResponse(
                    statusFlag=APPROVED, hrs=HRS, lrs=LRS,
                    mn=MN, coa=COA,))
            ELSE  /* mn is at a foreign GAN */
                IF MN_IS_ALLOWED_AT_LRS_GAN(mn=MN)
                THEN  /* allows registering of the MN at the foreign GAN */
                    CREATE_DB_RECORD(hrs=HRS, lrs=LRS,
                        mn=MN, coa=COA,
                        validity=DEFAULT_EXPIRING_TIME)
                    SEND(LRS, HRS, N2N_RegistrationResponse(
                        statusFlag=APPROVED, hrs=HRS,
                        lrs=LRS, mn=MN, coa=COA,))
                ELSE  /* denies registering of the MN at the foreign GAN */
                    SEND(LRS, HRS, N2N_RegistrationResponse(
                        statusFlag=DENIED_BY_HRS, hrs=HRS,
                        lrs=LRS, mn=MN, coa=COA,))
                FI
            FI
        ELSE  /* lrs is not hrs: checks partnership between lrs and hrs */
            IF CHECK_ PARTNERSHIP(hrs=HRS, lrs=LRS)
            THEN  /* requests the registering of the MN at the HRS */

        CREATE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                        lrs=LRS, mn=MN, coa=COA,
                        validity=TEMP_EXPIRING_TIME)
```

```
                    SEND(HRS, LRS, N2N_RegistrationRequest(hrs=HRS,
                                        lrs=LRS, mn=MN, coa=COA))
                ELSE    /* sends registering failure status */
                    SEND(COA, LRS, N2N_RegistrationResponse(
                                        statusFlag=DENIED_BY_LRS,
                                        hrs=HRS, lrs=LRS,
                                        mn=MN, coa=COA,))
                FI
            FI

        N2N_RegistrationResponse:       /* checks registering status */
            STATUS_FLAG=Message.statusFlag
            HRS=Message.hrs
            LRS=Message.lrs
            MN=Message.mn
            COA=Message.coa

            IF CHECK_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                        lrs=LRS, mn=MN, coa=COA)
            THEN    /* temporary registering record still valid */

                CASE STATUS_FLAG IN
                    APPROVED:   /* successful registering: creates DB record for MN
*/

            READ_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                lrs=LRS, mn=MN, coa=COA)

            DELETE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                lrs=LRS, mn=MN, coa=COA)
                        CREATE_DB_RECORD(hrs=HRS, lrs=LRS, mn=MN,
                                coa=COA, validity=DEFAULT_EXPIRING_TIME)
                    DENIED_BY_HRS:      /* unsuccessful registering: denied by HRS
*/

            DELETE_TEMPORARY_REGISTERING_RECORD(hrs=HRS,
                                        lrs=LRS, mn=MN, coa=COA)
                ESAC

                SEND(COA, LRS, N2N_RegistrationResponse(
```

```
                                statusFlag=STATUS_FLAG, hrs=HRS, lrs=LRS,
                                mn=MN, coa=COA,))
        ELSE    /* temporary registering record expired: ignores message */
        FI

N2N_RegistrationRefresh:/* starts the refresh cycle of the MN */
        HRS=Message.hrs
        LRS=Message.lrs
        MN=Message.mn
        COA=Message.coa
        CN_ADDRESS_LIST=Message.cnlist
        VALIDITY=validity

        /* checks if there is a DB record for the MN */
        IF CHECK_DB_RECORD(hrs=HRS, lrs=LRS, mn=MN, coa=COA)
        THEN    /* refreshes the DB record of the MN */
                UPDATE_DB_RECORD(hrs=HRS, lrs=LRS, mn=MN,
                                coa=COA, validity=VALIDITY)

                /* checks if LRS is not HRS */
                IF NOT CHECK_LRS_IS_HRS(hrs=HRS)
                THEN    /* lrs is not hrs: refreshes the HRS and LRSs of the CNs*/
                        SEND(HRS, LRS, N2N_RegistrationRefresh(hrs=HRS,
                                        lrs=LRS, mn=MN, coa=COA,
                                        validity=VALIDITY))

                        /* refreshes CNs through their respective LRSs */
                        FOR EACH CN IN CN_ADDRESS_LIST DO
                                LRS_CN=GET_LRS_OF_CN(cn=CN)
                                SEND(LRS_CN, LRS,
                                        BindingNotification(hrs=HRS,
                                        lrs=LRS, mn=MN, coa=COA, cnlist=CN,
                                        validity=VALIDITY)
                        DONE
                FI
        ELSE    /* ignores refresh cycle */
        FI

N2N_BindingNotification:        /* refreshes the MN record at the AR of the CN */
        HRS=Message.hrs
```

```
        LRS=Message.lrs
        MN=Message.mn
        COA=Message.coa
        CN_ADDRESS_LIST=Message.cnlist
        VALIDITY=Message.validity

        IF COA_BELONGS_TO_GAN(coa=COA)
        THEN   /* LRS is the LRS of the CoA: notifies the LRSs of each CN */
               FOR EACH CN IN CN_ADDRESS_LIST DO
                      LRS_CN=GET_LRS_OF_CN(cn=CN)
                      SEND(LRS_CN, LRS, BindingNotification(hrs=HRS,
                                    lrs=LRS, mn=MN, coa=COA, cnlist=CN,
                                    validity=VALIDITY)
               DONE
        ELSE   /* LRS is not the LRS of the CoA: checks if the CN belongs to GAN */
               IF CN_BELONGS_TO_GAN(cn=CN)
               THEN   /* relays the message to the AR of the CN */
                      AR = GET_AR_OF_CN(cn=CN)
                      SEND(AR, MY_LRS_ADDR, BindingNotification(hrs=HRS,
                             lrs=LRS, mn=MN, coa=COA, cnlist=NULL,
                             validity=VALIDITY))
               ELSE   /* ignores notification */
               FI
        FI

N2N_BindingCheckRequest:      /* checks the CoA of a MN */
        MN=Message.mn
        COA=Message.coa

        IF COA_BELONGS_TO_GAN(coa=COA)
        THEN   /* LRS is the LRS of the CoA */
               AR=Message.source_address

               /* checks if there is a DB record for the MN */
               IF CHECK_DB_RECORD(hrs=NULL, lrs=NULL,
                                                  mn=MN, coa=COA)
               THEN   /* sends the DB record of the MN to the AR_LRS */
                      READ_DB_RECORD(hrs=HRS, lrs=LRS,
                                                  mn=MN, coa=COA)
```

```
                    SEND(AR, MY_LRS_ADDR, N2N_BindingCheckResponse(
                        statusFlag=CONFIRMED, hrs=HRS, lrs=LRS,
                        mn=MN, coa=COA,
                        validity=DEFAULT_EXPIRING_TIME))
                ELSE /* sends the unconfirmed status to the AR_LRS */
                    SEND(AR, LRS, N2N_BindingCheckResponse(
                        statusFlag=UNCONFIRMED, hrs=NULL,
                        lrs=NULL, mn=MN, coa=COA, validity=NULL))
                FI
            ELSE    /* relays the message to the LRS of the COA */
                AR=Message.source-address

                /* creates a temporary checking record to control the resquest */
                CREATE_TEMPORARY_CHECKING_RECORD(mn=MN,
                    coa=COA, ar=AR, validity=TEMP_EXPIRING_TIME)

                COA_LRS = GET_LRS_OF_COA(coa=COA)
                SEND(COA_LRS, MY_LRS_ADDR,
                        N2N_BindingCheckRequest(mn=MN, coa=COA))
            FI

N2N_BindingCheckResponse:   /* checks binding status */
            STATUS_FLAG=Message.statusFlag
            HRS=Message.hrs
            LRS=Message.lrs
            MN=Message.mn
            COA=Message.coa
            VALIDITY=Message.validity

            IF CHECK_TEMPORARY_CHECKING_RECORD(mn=MN, coa=COA)
            THEN    /* there is a record that matches the data of the response message */
                READ_TEMPORARY_CHECKING_RECORD(mn=MN, coa=COA,
                                                    ar=AR)
                DELETE_TEMPORARY_CHECKING_RECORD(mn=MN,
                                                    coa=COA, ar=AR)
                SEND(AR, MY_LRS_ADDR, BindingCheckResponse(
                        statusFlag=STATUS_FLAG, hrs=HRS, lrs=LRS,
                        mn=MN, coa=COA, validity=VALIDITY))
            ELSE    /* ignores the response */
            FI
```

```
N2N_HandoverIndication:/* relays handover context of the MN */
        MODE=Message.mode
        OLD_POA_ID=Message.oldPoA-ID
        NEW_POA_ID=Message.newPoA-ID
        MN=Message.mn
        COA=Message.coa
        HANDOVER_CONTEXT=Message.hc

        CASE MODE IN
                ANTECIPATED:/* anticipated mode */
                        DEST_POA_ID=NEW_POA_ID
                REACTIVE:               /* reactive mode */
                        DEST_POA_ID=OLD_POA_ID
        ESAC

        IF POA_ID_BELONGS_TO_GAN(selectedPoA-ID=DEST_POA_ID)
        THEN    /* relays the messaget to the AR of the PoA */
                AR=GET_AR_OF_POA(selectedPoA-ID=DEST_POA_ID)
                SEND(AR, MY_LRS_ADDR, N2N_HandoverIndication(
                                                oldPoA-ID=OLD_POA_ID,
                                                newPoA-ID=NEW_POA_ID,
                                                mn=MN, coa=COA,
                                                hc=HANDOVER_CONTEXT))
        ELSE    /*tries to relay the message to the LRS of the PoA */
                LRS_POA=GET_LRS_OF_POA(selectedPoA-ID=DEST_POA_ID)

                IF LRS_POA <> NULL
                THEN    /* relays the message to LRS_POA */
                        SEND(LRS_POA,                   MY_LRS_ADDR,
N2N_HandoverIndication(
                                                oldPoA-ID=OLD_POA_ID,
                                                newPoA-ID=NEW_POA_ID,
                                                mn=MN, coa=COA,
                                                hc=HANDOVER_CONTEXT))
                ELSE    /* notifies the sender about the error */
                        SENDER=Message.souce_address
                        SEND(SENDER, MY_LRS_ADDR,
                                                N2N_HandoverAcknowledge(
                                                Mode=MODE,
```

statusFlag=LRS_UNKNOWN,
mn=MN, coa=COA))

FI

FI

ESAC

Appendix B
GMP message format

This appendix presents the message format of the messages exchanged between Mobility Manager (MM) entities.

B1 – GMP message structure

Each GMP message includes a message header and a PDU. The message structure is illustrated in Figure 24.

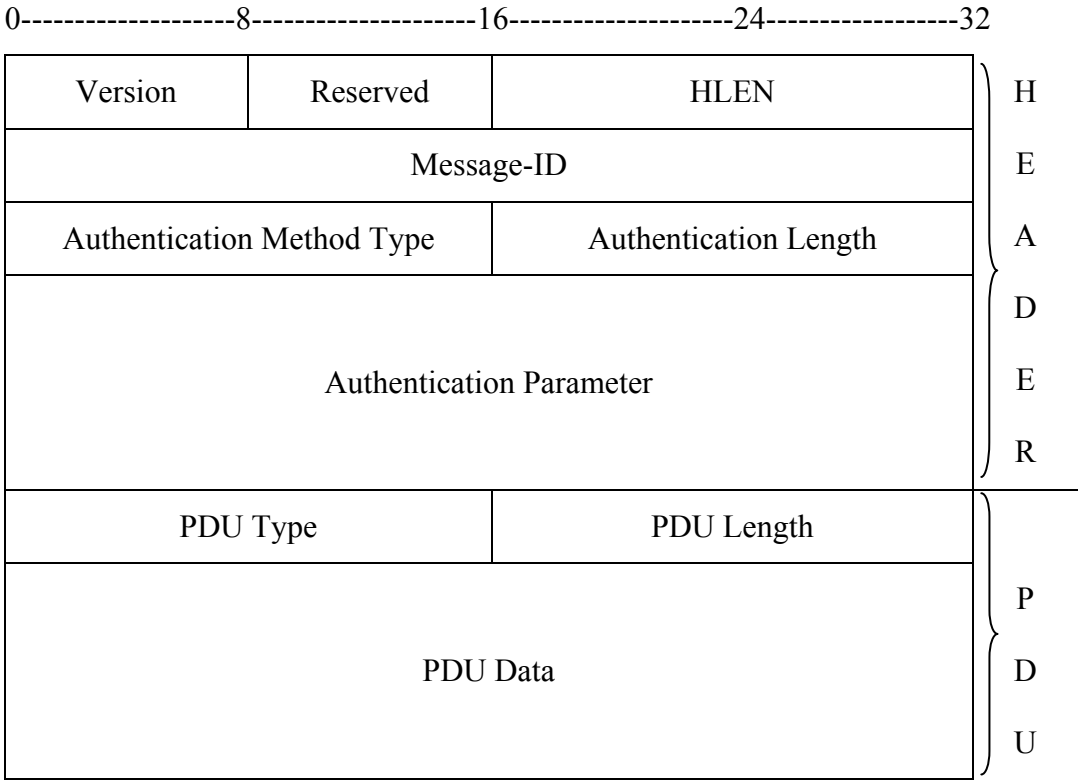

Figure 24 – GMP message structure

B1 – GMP message fields

Each field of the GMP message is described in Table 9.

Table 9 – Description of the GMP message fields

Field name	Field description
Version	Currently set to version 1.
Reserved	Reserved for future use.
HLEN	Indicates the length of the header of the message.
Message-ID	A unique identifier used between two MM entities to coordinate request and response messages. The range of this ID is 0 through $2^{31} – 1$.
Authentication Method Type	An identifier in the range of 0 through $2^{16} – 1$ that indicates which authentication method was used by the sender to prepare this message and therefore which authentication method must be used by the receiver to process this message. Reserved values are 1 for HMAC/MD5, 2 for HMAC/SHA1, 3 for MD5/RSA, 4 for SHA1/RSA and 5 for SHA1/DSA.
Authentication Length	Indicates the number of octets of the authentication parameter field (padding not included).
Authentication Parameter	An octet string that contains parameters generated by the authentication method in the sending MM entity and processed by the receiving MM entity. This field is a multiple of 4 octets and a padding of zeros must be used to complete the field when necessary.
PDU Type	An identifier in the range of 0 through $2^{16} – 1$ that indicates which PDU is carried by this message. Reserved values are: 0 – C2N_RegistrationRequest 1 – N2C_RegistartionResponse

	2 – N2N_RegistrationRequest 3 – N2N_RegistrationResponse 4 – C2N_RegistrationRefresh 5 – N2N_RegistrationRefresh 6 – N2N_BindingNotification 7 – N2N_BindingCheckRequest 8 – N2N_BindingCheckResponse 9 – C2N_HandoverInitiate 10 – C2N_HandoverFinish 11 – N2C_HandoverAcknowledge 12 – N2N_HandoverIndication 13 – N2N_HandoverForward 14 – N2N_HandoverAcknowledge
PDU Length	Indicates the number of octets of the PDU data field (padding not included).
PDU Data	An octet string that contains the data of the PDU. This field is a multiple of 4 octets and a padding of zeros must be used to complete the field when necessary.

Appendix C
GMP Protocol Data Unit (PDU) Types

The GMP actually defines 15 types of PDUs. This appendix presents the functional description of each PDU and their fields.

C1 – C2N_RegistrationRequest

PDU-Type: 0 (zero)

Description: Used to request the registering of the MN at the GAN.

From/To: GMAP_MN/GMA_LOCAL_RENDEZVOUS

Fields:

- newPoA-ID: Indicates the identification of the selected PoA.
- hrs: Indicates the IPv6 address of the MN's HRS.
- mn: Indicates the GMA address of the MN.

C2 – N2C_RegistrationResponse

PDU-Type: 1

Description: Used to inform the result of the registering request.

From/To: GMA_LOCAL_RENDEZVOUS/GMAP_MN

Fields:

- statusFlag: Indicates the result of the registering requested by the MN. The current valid values are:

  1 (APPROVED), 2 (DENIED_BY_AR),
  3 (DENIED_BY_LRS) and
  4 (DENIED_BY_HRS).

- hrs: Indicates the IPv6 address of the MN's HRS.
- lrs: Indicates the IPv6 address of the current MN's LRS. This field may be NULL.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the CoA that was bound to the address of the MN.

C3 – N2N_RegistrationRequest

PDU-Type: 2

Description: Used to request the registering of the MN at the GAN in behalf of the MN.

From/To: AR/LRS, LRS/HRS

Fields:

- hrs: Indicates the IPv6 address of the MN's HRS.
- lrs: Indicates the IPv6 address of the current MN's LRS. This field may be NULL.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the CoA that was bound to the address of the MN.

C4 – N2N_RegistrationResponse

PDU-Type: 3

Description: Used to inform the result of the registering request.

From/To: HRS/LRS, LRS/AR

Fields:

- statusFlag: Indicates the result of the registering requested in behalf of the MN. The current valid values are:

  1 (APPROVED), 3 (DENIED_BY_LRS)
  and
  4 (DENIED_BY_HRS).

- hrs: Indicates the IPv6 address of the MN's HRS.
- lrs: Indicates the IPv6 address of the current MN's LRS.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the CoA that was bound to the address of the MN.

C5 – C2N_RegistrationRefresh

PDU-Type: 4

Description: Used to execute the refresh cycle of the MN record.

From/To: GMAP_MN/GMA_LOCAL_RENDEZVOUS

Fields:

- currentPoA-ID: Indicates the identification of the current PoA.
- hrs: Indicates the IPv6 address of the MN's HRS.
- mn: Indicates the GMA address of the MN.
- cnlist: Indicates the address list of the CNs the MN is communicating with.
- validity: Indicates the new expiring value of the record.

C6 – N2N_RegistrationRefresh

PDU-Type: 5

Description: Used to execute the refresh cycle of the MN record in behalf of the MN.

From/To: AR/LRS, LRS/HRS

Fields:

- currentPoA-ID: Indicates the identification of the current PoA.
- hrs: Indicates the IPv6 address of the MN's HRS.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the CoA that was bound to the address of the MN.
- cnlist: Indicates the address list of the CNs the MN is communicating with.
- validity: Indicates the new expiring value of the record.

C7 – N2N_BindingNotification

PDU-Type: 6

Description: Used to notify the CoA of the MN to both the LRSs and ARs that currently offer service to the CNs, as part of the MN's re-fresh cycle.

From/To: LRS/LRS, LRS/AR

Fields:

- lrs: Indicates the IP address of the LRS of the MN's current GAN.
- hrs: Indicates the IPv6 address of the MN's HRS.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the CoA that was bound to the address of the MN.
- cnlist: Indicates the address list of the CN the MN is communicating with. This field is NULL when the LRS sends this PDU to the AR of the CN.
- validity: Indicates the new expiring value of the record.

C8 – N2N_BindingCheckRequest

PDU-Type: 7

Description: Used to request the validation of the binding between a CoA and the address of a MN.

From/To: AR/LRS, LRS/LRS

Fields:

- mn: Indicates the GMA address of the MN to be checked.
- coa: Indicates the CoA to be checked.

C9 – N2N_BindingCheckResponse

PDU-Type: 8

Description: Used to inform the result of the checking between a CoA and the address of a MN.

From/To: LRS/LRS, LRS/AR

Fields:

- statusFlag: Indicates the result of the checking. The current valid values are:

  1 – CONFIRMED and

  2 - UNCONFIRMED
- lrs: Indicates the IP address of the LRS of the MN's current GAN. May be NULL.
- hrs: Indicates the IPv6 address of the MN's HRS. May be NULL.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the CoA that was bound to the address of the MN.
- validity: Indicates the new expiring value of the record.

C10 – C2N_HandoverInitiate

PDU-Type: 9

Description: Used to initiate de L3 handover of the MN.

From/To: GMAP_MN/GMAP_DEFAULT_ROUTER

Fields:

- oldPoA-ID: Indicates the identification of the old PoA.
- newPoA-ID: Indicates the identification of the new PoA.
- hrs: Indicates the IPv6 address of the MN's HRS.
- mn: Indicates the GMA address of the MN.
- cnlist: Indicates the address list of the CNs the MN is communicating with.

C11 – C2N_HandoverFinish

PDU-Type: 10

Description: Used to finish de L3 handover of the MN.

From/To: GMAP_MN/GMAP_DEFAULT_ROUTER

Fields:

- oldPoA-ID: Indicates the identification of the old PoA.
- newPoA-ID: Indicates the identification of the new PoA.
- hrs: Indicates the IPv6 address of the MN's HRS.
- mn: Indicates the GMA address of the MN.
- cnlist: Indicates the address list of the CNs the MN is communicating with.

C12 – N2C_HandoverAcknowledge

PDU-Type: 11

Description: Used to confirm the end of the handover process.

From/To: GMAP_DEFAULT_ROUTER/GMAP_MN

Fields:

- statusFlag: Indicates the status of the L3 handover process. The current valid values are:

  1 – OK and 2 – FAIL

C13 – N2N_HandoverIndication

PDU-Type: 12

Description: Used to relay/request the handover context of the MN during handover process.

From/To: AR/LRS, LRS/LRS

Fields:

- mode: Indicates the mode of operation. The current valid values are:

  1 – ANTECIPATED and

  2 - REACTIVE

- oldPoA-ID: Indicates the identification of the old PoA.
- newPoA-ID: Indicates the identification of the new PoA.
- mn: Indicates the GMA address of the MN.
- coa: Indicates the IPv6 address of the MN's COA. Set to the IP address of the PAR if mode is ANTECIPATED. Set to the IP address of the NAR if mode is REACTIVE.
- hc: Indicates the handover context of the MN. Set to NULL if mode is REACTIVE.

C14 – N2N_HandoverForward

PDU-Type: 13

Description: Used to request/confirm the forwarding of the in-flight packets buffered during handover process.

From/To: AR/AR

Fields:

- mode: Indicates the mode of operation. The current valid values are:

  1 – ANTECIPATED and

  2 - REACTIVE

- mn: Indicates the GMA address of the MN.

C15 – N2N_HandoverAcknowledge

PDU-Type: 14

Description: Used to relay/confirm the handover context of the MN during handover process.

From/To: AR/AR, LRS/AR

Fields:

- mode: Indicates the mode of operation. The current valid values are:

  1 – ANTECIPATED and

  2 - REACTIVE

- statusFlag: Indicates the status of the L3 handover process. The current valid values are:

  1 – OK and 2 – LRS_UNKNOWN

- mn: Indicates the GMA address of the MN.

- coa: Indicates the IPv6 address of the MN's COA. Set to the IP address of the NAR if mode is ANTECIPATED. Set to the IP address of the PAR if mode is REACTIVE.

- hc: Indicates the handover context of the MN. Set to NULL if mode is ANTECIPATED.