Unified Hierarchical Mobility Management Architecture
Spécifications Techniques d'Implantation dans FreeBSD
Révision novembre 1999

C. CASTELLUCCIA - L. BELLIER - PROJET PLANETE - INRIA Rhône-Alpes
claude.castelluccia@inrialpes.fr

16 Novembre 1999

Ce document décrit l'implantation du protocole Unified Hierarchical Mobility Management Architecture et Micro Mobile IPv6 dans la souche FreeBSD de l'INRIA, incluant la pile IPv6 et Mobile IPv6. Ces deux protocoles ont été décrits en détail dans le document "Unified Hierarchical Mobility Management Architecture Spécifications fonctionnelles". Ce travail s'inscrit dans le cadre du projet RNRT MobiSecV6. Le projet RNRT MobiSecV6 est réalisé conjointement par BULL, l'INRIA et le CNET.

Table des matières

1 Architecture

1.1 Généralités

La mobilité dans FreeBSD est incluse en partie dans les démons NDP, l'autre partie est du code noyau. Il existe deux démons NDP, ndpd-router et ndpd-host, à lancer respectivement sur les routeurs et sur les stations. Le code se trouve dans les répertoires :

```
/usr/src/usr.sbin/ndpd-router
/usr/src/usr.sbin/ndpd-host
```

ndpd-router et ndpd-host sont constitués de fichiers sources qui portent les même noms, mais ne contiennent pas le code. Par exemple il existe un icmp.c dans ndpd-host et un dans ndpd-router. Lorsque nous énoncerons des spécifications d'implémentation icmp pour le router, il faudra apporter les modifications dans icmp.c de ndpd-router.

ndpd-router et ndpd-host produisent chacun un binaire exécutable. Ces deux programmes incluent une partie du code de la mobilité : la gestion des listes des CH et des bindings. L'autre partie, l'ajout des entête et des options de Mobile IPv6, est faite directement dans le noyau. Les démons NDP communiquent avec le noyau en utilisant des sockets. Ces communications peuvent être des paquets pour envoi sur le réseau mais aussi des consignes, comme par exemple l'enregistrement d'un CH pour lequel il faut ajouter une option H@ dans les paquets.

La première partie des sources de la mobilité, celle incluse dans les démons NDP, se trouve dans /usr/src/usr.sbin/ndpd-mobility Les fichiers sont classés comme suit :

- m-* code de la mobilité
- ha-* code spécifique au HA
- mh-* code spécifique au MH
- mm-* code spécifique pour le multi-homed
- mc-* code commun aux différentes entités

L'autre partie des sources est comprise dans le code du noyau, principalement dans /usr/srs/sys/netinet/.

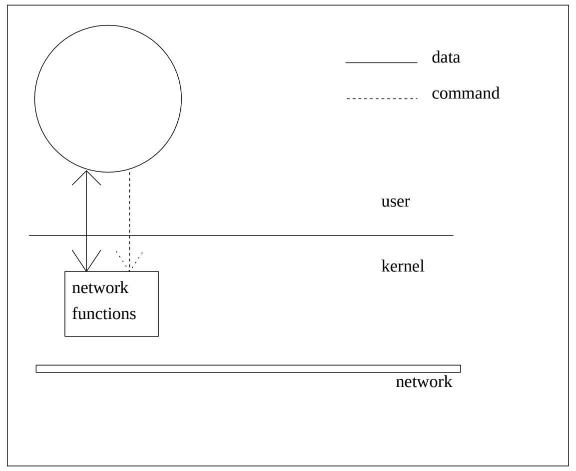

FIG. 1 – *La mobilité dans FreeBSD*

1.2 Schéma

1.3 Rôle des parties

Le noyau ajoute les entêtes utiles à la mobilité, comme les entête de routage, procède l'échange des adresses COA / H@ et dans le cas d'un HA intercepte les paquets à destination des MH et les fait suivre vers leur position courante. Le démon NDP s'occupe du reste ; la gestion des correspondants, des bindings, timeouts, etc.

1.4 Algorithme

Nous emploierons un langage algorithmique de haut niveau pour décrire le code à ajouter aux démons ndpd. Nous adopterons une notation style pascal pour accéder au champ des structures. Par exemple considérons une liste Router. Nous accéderons au divers champ d'un élément de cette liste par Router(x).champy

Nous traiterons souvent des informations issues d'un paquet IP, composé de Destination Option (Destination Option). Nous emploierons une notation semblable à celle ci-dessus pour accéder aux informations du paquet. Nous noterons toujours le paquet traité par P. Supposons que nous travaillons sur un paquet de BU. Nous accédons par exemple à la Destination Option BU de P par P.BU, au champ de la BU par P.BU.flag.P, etc.

1.5 Implémentations

Toutes les fonctionnalités décrites dans le document "Unified Hierarchical Mobility Management Architecture Spécifications fonctionnelles" seront à inclure dans la souche IPv6 du sous projet 1. Actuellement le travail s'effectue sous FreeBSD 3.2 mais devrait bientôt migrer vers 3.3.

L'installation de la souche IPv6 pour FreeBSD est un fichier tar.gz Il suffit de télécharger cette archive sur le site FTP de l'INRIA, et de le décompresser depuis la racine, par un tar xzf News.tar.gz Ensuite un petit script va finir l'installation des sources modifiées. Il se trouve dans /usr/src/IPv6. Il s'appelle inst-new. Une fois son exécution terminée, il ne reste plus qu'à recompiler le noyau et le système. Le système se recompile depuis le répertoire /usr/src/ par un make world. Pour la compilation du noyau, se référer à la documentation livrée avec celui-ci. Il faut simplement ajouter dans le fichier de configuration les lignes suivantes :

```
options DEBUG
options DIAGNOSTIC
options "INET6"
options IPSEC_AUTH
options "MD5"
pseudo-device    sit    X
pseudo-device    cti    Y
pseudo-device    llink
pseudo-device    tug 2
```

(with X >= 1 and X >= "number of used LAN physical interfaces")
(with Y >= 1, you add more at run time)

Par la suite, les sources de la mobilité hiérarchiques seront inclues dans la souche IPv6. Il n'y aura donc pas de manipulation supplémentaire à effectuer pour en disposer. Il suffira de recompiler le noyau, ndpd-host et ndpd-router avec les flags uMIP, uMIPBS et/ou UHMMA.

Voici la liste des fonctionnalités devant être réalisées, avec leur dépendance et leur état de réalisation. L'état de réalisation peut être non implémenté, implémenté et testé, partiellement implémenté ou implémenté non testé. Le terme test désigne les tests de fonctionnalité, les tests de fonctionnement étant obligatoirement fait au préalable.

Note : toutes les fonctionnalités sont évidemment dépendantes de la souche IPv6 de l'INRIA.

1.5.1 Mobile

- Découverte du MA : implémenté.
- Enregistrement du mobile auprès du MA : implémenté.
- Maintien de position : implémenté.
- Gestion des mouvements : implémenté, partiellement testé (en attente WLAN).
- Sécurité : non implémenté, dépendant de IPSEC dans FreeBSD.
- Confidentialité de position : non implémenté, jugé non prioritaire.
- Basculement du mode Mobile IP vers mode UHMMA (et inversement) : partiellement implémenté, jugé non prioritaire.

1.5.2 Base Station

- Apprentissage des informations sur le MA : implémenté.
- Communication des informations sur le MA : implémenté.

1.5.3 Mobility Agent

- Enregistrement des mobiles : implémenté, jugé prioritaire.
- Smooth handoff : implémenté.
- Gestion des mobiles : implémenté.
- Gestion des correspondants : partiellement implémenté (smooth handoff implémenté ; filtre apprentissage de nouveaux correspondants non implémenté, jugé non prioritaire.
- Mode proxy : implémenté.
- Communication d'adresse : implémenté.
- Sécurité : non implémenté, dépendant de IPSEC dans FreeBSD.

Tout nouveau code apporté aux sources des deux démons NDP et au noyau sera encadré par un couple define / endif. Il sera alors possible d'inclure ou non ce code lors de la compilation de ndpd-host / ndpd-router. Il existera trois define ; chacun désignant le code:

- spécifique de UHMMA, il s'agit donc de code dans le mobile $->$ UHMMA.
- d'une Base Station (BS), une BS peut être un correspondant classique, et donc un Home Agent (HA) $->$ uMIPBS.
- d'un Mobility Agent(MA), le MA ne peut être un Home Agent, en effet la plus part des fonctions du HA sont "hackées" pour faire tourner Micro Mobile IPv6 $->$ uMIP.

1.6 Tests

Pour les tests nous utilisons notre plate-forme IPv6. Cette plate-forme est composée de routeurs FreeBSD et de machine IPv6 ; FreeBSD, Windows NT et Solaris, et bientôt Linux. Nos réseaux sont des réseaux ethernets et des réseaux hertziens (WaveLAN bientôt très bientôt). Nos mobiles sont des PC portables munis de carte réseau Ethernet. Les tests consisteront à ouvrir des sessions FTP depuis ou vers des mobiles. Les mobiles seront alors déplacés. La simple observation de l'état du transfert FTP permettra de valider les protocoles.

1.7 Distribution

Les sources de l'implémentation seront disponibles dès que le prototype sera jugé stable. Elles seront ajoutées dans la distribution de la souche IPv6 du sous projet 1. La décompression et l'installation de la souche suffira donc pour posséder UHMMA et Micro Mobile IPv6. Le choix de l'utiliser ou non sera fait lors de la compilation des ndpd et du noyau en utilisant des "flags" de compilation UHMMA et uMIP. Une version pourra également être distribuée incluse dans le code de Mobile IPv6 du sous-projet 1. Un patch sera alors nécessaire pour le noyau.

1.8 Mode d'emploi

Il y a plusieurs paramètres à communiquer au démon NDP. Nous utiliserons pour cela la ligne de commande. Tous les détails seront décrit dans la MAN PAGE des démons NDP.

2 Protocole UHMMA

Le mobile doit pouvoir fonctionner dans trois modes distincts ; Mobile IPv6 et UHMMA mode 1 et 2. Nous utilisons donc un flag pour refléter le mode de fonctionnement ;

```
int mobility_mode
```

Ce flag sera déclaré globalement. Il pourra prendre trois valeurs, constantes symboliques, il sera initialisé à Mobile IPv6 par défaut. Un paramètre de la ligne de commande le modifiera :

- -m : Mobile IPv6
- -U 1 : UHMMA mode 1
- -U 2 : UHMMA mode 2

Pour nos variables globales nous allons utiliser une structure nommée *global*. Cette structure incluera toutes nos variables globales. Ainsi pour accéder a *mobility_mode*, nous emploierons *global.mobility_mode*.

Nous ajoutons un case dans la boucle de décodage des arguments de la ligne de commande, ceci est fait dans main (main.c) :

```
case 'U' :
  if (*optarg == '1')
  {
    global.mobility_mode = UHMMA_MODE1;
    global.mobility_mode_init = UHMMA_MODE1;
    mobile =1;
  }
  else if (*optarg == '2')
  {
    global.mobility_mode = UHMMA_MODE2;
    global.mobility_mode_init = UHMMA_MODE2;
    mobile =1;
  }
  else
    exit(3);
break ;
```

Etant donné que le mobile, lors de ses multiples déplacement à travers l'Internet sera peut être amené à basculer entre les différents modes (selon les dispositions des sites visités). Il est donc bon de garder en mémoire, le mode que l'utilisateur a demandé lors du lancement de la mobilité : *mobility_mode_init*.

2.1 Variables globales de ndpd-host

Voici la liste des variables que nous utiliserons, elles seront donc des éléments de notre structure *global* :

- *seq_num* ; un entier pour comptabiliser les paquets d'enregistrement / maintien de position
- *uMIP_timer*, *reemis_timer*, *back_timer* ; des timers
- *HomeAgent* ; un pointeur sur une structure contenant les info du HA
- *MA_address*, *MA_old*, des adresses IPv6 pour mémoriser le MA courant, et l'ancien

2.2 Déplacements

Les déplacements dans ndpd-host sont gérés dans les fonctions *major_move* et *minor_move*. *major_move* traite les déplacements depuis ou vers la Home Address (H@). *minor_move* traite tous les autres déplacements. *major_move* et *minor_move* ne traitent donc pas la macro mobilité et la micro mobilité telle que nous les avons définies dans les spécifications fonctionnelles [3]. Nous aurons besoin de connaître l'ancien MA et le MA courant ; pour cela nous définissons deux nouvelles adresses globales : *MA_address* et *MA_old*. *MA_address* contient en permanence l'adresse du MA du site sur lequel se trouve le MH.

Lorsque le mobile est sur son réseau mère, l'adresse de l'ancien MA à sauvegarder doit être l'adresse indéterminée (unspecified address). L'initialisation de cette adresse est faite dans *mobility_init*. Nous initialisons à unspecified quelque soit le cas de démarrage. Le MH peut démarrer à l'extérieur, l'adresse ne doit pas être unspecified dans ce cas. L'affectation de *MA_old* avec la bonne valeur se fera après la phase d'enregistrement.

Pour les paquets d'enregistrement et les paquets de confirmation de position nous avons besoin d'un compteur de numéro de séquence ; cela sera int *seq_num*. Ce compteur sera initialisé par défaut à 0 dans la fonction *mobility_init*. Si le MH démarre sur un réseau extérieur, ce compteur sera modifié à une valeur aléatoire dans *major_move*. Le fait de démarrer à l'extérieur revient à effectuer un déplacement, un major move.

Nous allons modifier le code de *major_move* et *minor_move* pour suivre notre protocole.

2.3 *major_move*

major_move est appelée lorsque le MH effectue un déplacement vers ou depuis sa H@. Ce n'est pas obligatoirement un cas de macro mobilité. Ce type de mouvement peut aussi bien être inter site que intra site. Le code distingue deux cas :

2.3.1 "from home"

Nous conservons tout le code (qui modifie les structures de données ; comme les flags at home ...) et nous plaçons devant la boucle

```
FORALLCOR(cop)
    mobile_move(....)
```

une conditionnelle. Elle a pour but de déterminer le bon traitement en fonction du mode de mobilité. Si le MH tourne Mobile IPv6 le traitement est l'appel à mobile_move, sinon il sera l'appel à une nouvelle fonction, send_uhmma_major_reg. Le rôle de cette fonction sera d'enregistrer le MH auprès du MA.

```
if (global.mobility_mode == UHMMA_MODE2)
{
    send_uhmma_major_reg(&cp->addr)
    FORALLHA(hap)
       FORALLHP(happ, hap)
          if (pp == happ)
          {
              hap->capp = cp
              hap->state = HAST_OUT
          }
    return;
}
else if (global.mobility_mode == UHMMA_MODE1)
    send_uhmma_major_reg(&cp->addr)

/* suite du code MIPv6 */
FORALLCOR
    mobile_move(cop, &pp->addr) ;
```

Nous notons donc que pour le cas 1, nous allons exécuter le code de UHMMA (pour enregistrement au MA) et celui de MIPv6 (pour envoi de BU vers les CH). Le code de MIPv6 sera néanmoins modifié pour tenir compte qu'il ne faut pas envoyé de BU aux CH situés à l'extérieur du site en cas de micro mouvement.

2.3.2 "to home"

même commentaires que plus haut.

```
if (global.mobility_mode == UHMMA_MODE2)
{
    send_uhmma_major_reg(&pp->addr);
    FORALLHA(hap)
       FORALLHP(happ, hap)
          if (pp == happ)
          {
              if (hap->capp != NULL)
                  hap->capp->mflags &= ~MOB_ISCA;
              hap->capp = NULL;
              hap->state = HAST_IN;
          }
   return;
}
else if (global.mobility_mode == UHMMA_MODE1)
{
    send_uhmma_major_reg(&pp->addr);
}

/* suite  du code MIPv6 */
FORALLCOR
    mobile_move(cop, &pp->addr) ;
```

2.3.3 *send_uhmma_major_reg()*

Cette fonction construit le paquet d'enregistrement et le transmet au noyau pour envoi sur le réseau vers le MA.

Il est inutile de communiquer une PCOA dans le paquet car la PCOA principale est utilisée pour initialiser le champ source du paquet IPv6.

Note : nous ne réalisons ici que du code pour le protocole UHMMA. L'envoi de BU vers les correspondants du mobile s'il utilise le mode 1, est réalisé par le code de Mobile IPv6.

Départ depuis la H@ Si *MA_old* est indéterminée, le MH vient de quitter son réseau mère. Lorsqu'il sera enregistré correctement, le MH recevra des paquets encapsulés depuis le MA courant. Il doit pour cela préparer un tug. Un tug est un pseudo device FreeBSD utilisé pour encapsuler décapsuler des paquets IPv6 entre deux machines. Nous utilisons la fonction *madd_tunnel* (code existant de MIPv6 cd 2.3.3), pour préparer le tug du coté du MH. Ce tug existera du coté du MH tant que le MH reste à l'intérieur de ce site. Les mouvements intra site n'entraîne pas sa modification car le MA reste le même. Il sera détruit par la procédure qui traitera le prochain mouvement inter site. Puis il sera remplacé par un tug vers le MA du nouveau site. Lorsqu'il sera enregistré (reception des BACK) le mobile pourra initialiser *MA_old* avec la valeur de *MA_address*.

Retour sur le réseau mère Si *MA_old* n'est pas indéterminée, le MH revient sur son réseau mère. Le MH va recevoir des paquets encapsulés depuis le MA de sa précédente position. Il devrait pour cela préparer un tug. Mais en fait le tug existe déjà, en effet le mobile avait ouvert ce tug avec son précédent MA lorsqu'il était dans son site. Du coté mobile, ce tug sera toujours ouvert vers la même adresse, celle du MA. *MA_old* va faire suivre les paquets pour le MH vers le MH lui-même. Cette redirection est limitée dans le temps. Lorsque la redirection sera achevée, il sera inutile de conserver le tug. Il faudrait que ce coté du tunnel se ferme. Nous utiliserons un timer *back_timer* pour limiter la durée de vie du tug. Le timer sera lancé en lui communiquant la fonction *clear_back_tug*. Cette fonction fermera le tug. Nous indiquons en paramètre le tug correspondant. Plus tard dans ce document nous verrons en détail le fonctionnement des différents timers2.6.1. Pour l'instant rappelons simplement qu'un MH doit confirmer sa position au MA régulièrement. Pour créer la périodicité nous employons un timer et une fonction. Ce timer s'appelle *uMIP_timer*, la fonction enverra les maintien de position. Le MH vient de rentrer sur son réseau mère, il n'aura plus besoin de confirmer sa position, le timer n'est plus utile. Il doit donc être détruit.

BACK Nous devons déterminer si un ou deux BACK seront attendus en retour de l'enregistrement, cela dépend du type de mobilité. Dans le cas présent, le HA sera obligatoirement averti quelque soit le type de déplacement.

Il acquittera comme l'impose Mobile IPv6, le MH doit être en attente de deux BACK, puisque le BACK du MA est obligatoire. Nous allons utiliser une variable *nbBACK* pour symboliser les BACK en attente. La politique sera la suivante :

– pour un BACK attendu, *nbBACK* prend la valeur -1, le BACK est celui du MA. Il est conséquence d'un enregistrement pour un déplacement intra site.

– pour deux BACK attendus, *nbBACK* est positionné à 3. Ensuite si le BACK du HA arrive, nbBACK passe à 1. Si le BACK de MA arrive, *nbBACK* passe à 2.

– aucun BACK attendu, *nbBACK* vaut 0.

Tug Le code pour ouvrir un tug existe. Il a été écrit pour Mobile IPv6, dans la fonction *madd_tunnel*. Cette fonction est utilisée par un MH pour ouvrir un tug vers son HA. Nous l'utiliserons pour faire de même vers le MA local. Cependant dans MIPv6 cette fonction porte obligatoirement sur le tug numéro 1. En effet dans MIPv6 le tunnel vers le HA est toujours le même, inutile d'implémenter une liste de tug, comme dans un MA. Pour notre code UHMMA nous allons ouvrir un second tug, vers le MA local. Nous allons donc utiliser le tug numéro 2.[1] La fonction *madd_tunnel* nécessite une *struct corr* en paramètre pour identifier l'autre extrémité du tug. Dans Mobile IPv6 la *struct corr* contient des informations utiles sur un CH. Ici la plupart de ces informations (qui sont des champs dans une structure C) nous sera inutile. Car nous utiliserons simplement l'adresse du MA. Nous créons une structure avec le champ addr initialisé par l'adresse du MA pour la passer en paramètre à *madd_tunnel*. Nous allons également modifier cette fonction pour quelle porte sur le tug numéro 1 ou numéro 2 selon si l'on traite le tug vers le HA ou le MA. Pour cela en début de fonction nous testons si l'adresse contenue dans la structure *corr* reçue en paramètre est celle du HA ou non.

Le code pour fermer un tug est naturellement dans une fonction *mdel_tunnel*. Cette fonction s'utilise de la même manière que *madd_tunnel*.

Construction du paquet d'enregistrement Le code de l'envoi du paquet est une copie du code de *send_bnd_upd* ; l'adresse de destination est

1. L'emploi de deux tug impose que le noyau du mobile soit compilé avec l'option "pseudo-device TUG 2".

celle du MA et le code de la composition des options est Destination Option BU avec VCOA et *MA_old* et HA en sub-option + Destination Option H@. Nous pouvons utiliser les primitives SET8, SET16 ... pour construire le paquet. Les informations H@ et HA peuvent être trouvées en parcourant la liste des correspondants en recherchant le HA (*flags == OPT6_BNDU_HR — OPT6_BNDU_ACK*). Nous nous apercevrons vite que les informations sur le HA sont souvent utilisées. Pour éviter de parcourir la liste de CH à de nombreuses reprises, nous utiliserons un pointeur HA. Ce pointeur indiquera qui est le HA dans la liste des CH : *struct corr * HomeAgent*. Le compteur *seq_num* est incrémenté d'un pas. Puis il est copié dans le champ sequence number du paquet d'enregistrement à envoyer.

Réémission d'enregistrement Le paquet d'enregistrement pourra être réémis si le(s) BACK n'arrive(nt) pas avant un temps fini. Ce temps est mesuré en utilisant un timer ; *reemis_timer* (cf. 2.6.1). Lorsqu'il envoie le paquet d'enregistrement, le MH lance le timer *reemis_timer* en lui donnant une fonction à exécuter s'il expire : *resend_uhmma_reg*. Le paquet d'enregistrement sera alors réémis. Pour éviter d'avoir à le reconstruire à chaque réémission, nous le conservons en mémoire. Le paquet sera pointé par une variable *reemit_reg_packet*, cette variable sera passée en paramètre à *resend_uhmma_reg* au moment du lancement du timer.

Algorithme

```
seq_num <-- seq_num +1
P <-- construire_paquet(REG, MA_address, seq_num)
P' <-- P
émettre (P)
reemis_timer <-- lancer_timer(resend_uhmma_reg, P')
si MA_old == adresse indéterminée alors
    tug_2 <-- ouvrir_tug(MA_address)
sinon
    back_timer <-- lancer_timer (clear_back_tug)
    détruire_timer (uMIP_timer)
```

2.3.4 *resend_uhmma_reg (paquet P)*

Le BACK de l'enregistrement n'est pas parvenu avant l'expiration du timer *reemis_timer*. La fonction *resend_uhmma_reg* est donc lancée. Peut être le BACK est perdu ou bien un nouveau MA a été mis en place suite à un crash. Le paquet d'enregistrement doit être réémis. Avant nous vérifions si l'adresse du MA n'a pas changé. Si elle a changé il faut modifier le paquet d'enregistrement pour le diriger vers le nouveau MA.

Algorithme

```
si P.destination == MA_address alors
    émettre(P)
sinon
    P.destination <-- MA_address
    émettre(P)
```

2.3.5 *clear_back_tug*

Après un retour sur sa H@, un MH est le destinataire temporaire des paquets redirigés par son ancien MA. Pour cela il ouvre un tug vers ce MA. La durée de vie du tug est finie, après il doit être fermé. *clear_back_tug* est appelée du fait de l'expiration d'un timer qui mesure la durée de vie du tug. Elle ferme le tug.

Algorithme

```
fermer_tug(tug_2)
détruire_timer (back_timer)
MA_old <-- unspecified address
```

2.4 *minor_move*

minor_move est appelée lorsque le MH effectue un déplacement dans lequel n'intervient pas la H@. Ce déplacement peut être un déplacement inter site ou intra site.

Même démarche que pour *major_move* paragraphe 2.3, nous plaçons une conditionnelle devant la boucle

```
if((global.mobility_mode == UHMMA_MODE2)||
   (global.mobility_mode == UHMMA_MODE1))
{
    send_uhmma_minor_reg(&newpp->addr);
    if (global.mobility_mode == UHMMA_MODE2)
       return;
}
FORALLCOR(cop)
    mobile_move(    )
```

Le reste de la fonction reste inchangé.

2.4.1 *send_uhmma_minor_reg*

Cette fonction est très similaire à *send_uhmma_major_reg*. Le format du paquet d'enregistrement est le même : Destination Option BU et *MA_old* en sub-option et HA en sub-option et Destination Option H@. La mécanique de gestion du sequence number est la même. Le compteur est augmenté d'une unité, puis copié dans le champ sequence number du paquet.

Timers Le MH vient d'un réseau qui n'était pas son réseau mère, sinon le traitement aurait été fait dans *major_move*. Le timer *uMIP_timer* est en activité. Nous allons l'arrêter pour l'instant, car le MH doit passer par une phase d'enregistrement acquittée avant de pouvoir commencer à confirmer sa position. Nous allons donc attendre d'avoir reçu le(s) BACK pour relancer ce timer.

BACK Selon le type de déplacement que le MH vient d'effectuer, il doit attendre un ou deux BACK ; un du MA et éventuellement un du HA. Pour connaître le type de mouvement, le MH peut comparer *MA_old* et *MA_address* [2]. Si les préfix sont différents, le MH vient de changer de site, il lui faudra attendre deux BACK. La comparaison doit porter sur les préfix, et non sur les adresses elles-même. Dans le cas d'un remplacement de MA, les adresses changent, mais les préfix restent les mêmes. Dans le cas, où un MH se déplacerait à l'intérieur d'un site en même temps que le remplacement du MA, le MH ne doit pas interpréter le changement comme un déplacement inter site, c'est pourquoi il teste les préfix et non les adresses. C'est lors du traitement de réception des BACK que le MH devra vérifier de qui proviennent les BACK. Nous devons ici indiquer si un ou deux BACK seront attendus. Nous positionnons la variable *nbBACK* à -1 pour un BACK ou à 3 pour deux.

Tug Si le MH vient d'opérer un mouvement intra site, le tug vers le MA ne change pas pour le MH (bien qu'il change du coté du MA pour prendre en compte la nouvelle PCOA du mobile). Si le MH change de site alors le tug vers le MA doit être changé. Sur son précédent site le MH avait ouvert un

2. rappel : *MA_address* est initialisé par la fonction recv_rtadv lorsqu'elle décode le rt adv.

tug vers le MA local. Ce tug doit maintenant être fermé pour être remplacé par celui vers le nouveau MA, le MA local.

Réémission du paquet d'enregistrement Nous utilisons ici la même mécanique que pour *send_uhmma_major_reg*. Nous conservons une copie du paquet en mémoire, pointée par un pointeur. Ce pointeur est transmit au timer *reemis_timer* lors de son armement. Lorsque le(s) BACK sera(ont) arrivé(s), la mémoire sera rendu au système et le timer détruit.

Algorithme

```
seq_num <-- seq_num +1
P <-- construire_paquet (REG, MA_address)
P' <-- P
émettre (P)
reemis_timer <-- lancer_timer (resend_uhmma_reg, P')
si MA_old != MA_address alors
    détruire_tug(tug_MA)
    tug_MA <-- ouvrir_tug(MA_address)
détruire_timer (uMIP_timer)
```

2.4.2 *send_old_position*

Périodiquement le MH doit confirmer sa position au MA qui le gère, sans quoi sa mobilité ne sera plus prise en charge par le MA. Ceci est fait pas envoi de BU (Destination Option BU + Destination Option H@) avec le flag P pour le distinguer d'un BU MIPv6. Nous utilisons le timer *uMIP_timer* pour générer la périodicité des envois. Chaque fois que le timer expire, un BU est envoyé par la fonction *send_old_position*. Puis elle relance le timer. Le compteur *seq_num* est augmenté d'un point, puis copié dans le champ sequence number du paquet.

Algorithme

```
seq_num <-- seq_num +1
Pbu <-- construire_paquet(BU, MA_address)
émettre (Pbu)
détruire_timer(uMIP_timer)
uMIP_timer <-- lancer_timer(send_old_position)
```

2.5 Plusieurs VCOA

Lorsqu'un MH change de site, il peut retirer l'ancienne VCOA de son interface réseau. Mais nous devons la conserver temporairement pour pouvoir recevoir des paquets encapsulés du MA précédent. Ces paquets sont adressés à l'ancienne VCOA. Le MA old les encapsule vers la nouvelle VCOA. Le nouveau MA fait suivre vers la PCOA. Les paquets décapsulés seront jetés si l'ancienne VCOA n'est plus une adresse du MH. La durée de vie de l'ancienne VCOA doit donc être la même que la durée de forwarding du MA précédent. Lorsque le MH change de réseau, le préfix associé au réseau précédent n'est pas détruit instantanément, il est déprécié. L'adresse reste donc sur l'interface réseau. Elle sera retirée lorsque le préfix sera retiré lui-même(lorsque le temps de dépréciation sera passé).

2.6 Fonctionnement

2.6.1 Timers

Le MH utilise plusieurs timers. Nous reprenons ici un par un le rôle de chacun d'entre eux. Les timers sont des variables globales.

Gestion des timers Le code de ndpd contient des fonctions pour gérer des timers. Une liste de timers est implémentée. Le fait d'insérer un timer dans la liste, assume que le timer fonctionne et exécute un traitant s'il vient à expirer. Le traitant est une fonction qu'il faut communiquer au timer lors de son armement. Il est également possible de préciser des arguments à donner au traitant lors de son lancement. Nous utilisons ces fonctions codées dans util.c :

- void add_timeout(struct timeout *)
  ajoute un timer à la liste
- void set_timeout(struct timeout *, int, int, func_to, void *)
  initialise le timer donné avec la durée en sec, en micro sec, la fonction à exécuter et ses paramètres
- struct timeout * new_timeout(int, int, func_to, void *)
  créer, initialise et ajoute un timer dans la liste
- void kill_timeout(struct timeout *)
  retire un timer de la liste

Réémission d'un enregistrement Le premier timer sert à la réémission des paquets d'enregistrement pour lesquelles le MH n'a pas encore reçu de BACK ; reemis_timer. Lorsque le timer est armé dans *send_uhmma_major_reg* ou *send_uhmma_minor_reg*, nous lui communiquons la fonction *resend_uhmma_reg(char *)*. Le paramètre est un pointeur sur le paquet d'enregistrement. Cette fonction sera exécutée en cas d'expiration du timer. Elle est chargée de réémettre l'enregistrement et de relancer le timer. Elle repositionne le flag *nbBACK* à la bonne valeur ; -1 si le MH attend un BACK, 3 pour deux BACK. Le nombre de BACK dépend du type de mouvement que vient d'effectuer le MH. Deux BACK pour un déplacement inter site. Pour déterminer le type de mouvement, nous comparons *MA_address* et *MA_old*. Si leur préfix sont identiques, le MH vient d'effectuer un déplacement intra site. Lors de la première émission du paquet d'enregistrement, ce paquet a été sauvegardé en mémoire, pointé par *char * reemit_reg_packet*. Ainsi la fonction de rémission n'aura plus qu'à le retourner au noyau pour envoi sur le réseau sans avoir à le reconstituer.

Lorsque le(s) BACK arrive(nt), la mémoire utilisée pour la sauvegarde de ce paquet peut être libérée et le timer peut être arrêté.

Ce timer sera :

- lancé à chaque mouvement,
- relancé chaque fois qu'il expire en même temps que la réémission de l'enregistrement,
- détruit par l'arrivée des BACK attendus,

Confirmation de position Le deuxième timer *uMIP_timer* sert au MH pour confirmer sa position au MA. Il est armé quand le MH quitte sa H@ et qu'il a reçu les BACK, il sait alors qu'il est correctement enregistré. Ceci est fait dans la fonction *recv_bnd_ack*. Une fonction, *send_old_position*, est communiquée au timer. Le rôle de cette fonction est d'envoyer un BU au MA courant, puis de relancer le timer. Le timer est détruit lorsque le MH revient sur son réseau mère, dans la fonction *major_move*.

Ce timer sera :

- lancé après réception des BACK de l'enregistrement,
- relancé chaque fois qu'il expire en même temps que l'émission d'un BU vers le MA,
- détruit par un retour à la H@,

– temporairement arrêté pendant la phase d'enregistrement du MH,

Direct forward Le dernier timer, *back_timer* sert pour mesurer la durée de vie du tug vers le MA précédant dans le cas d'un retour sur le réseau mère. Dans ce cas précis, le MA précèdent ne redirige pas au MA courant mais directement vers la H@. Cette redirection étant limitée dans le temps, la durée de vie du tug doit l'être également. Ce timer est donc lancé dans *major_move* en lui communiquant la fonction *clear_back_tug*. Cette fonction fermera le coté du tunnel.

Ce timer sera :

– lancé lors du retour sur le réseau mère,

– détruit de fait de son expiration,

2.6.2 ACKnowledgment

Lorsqu'un paquet contenant une destination option arrive sur une interface, le noyau le remonte vers le démon ndp. Il est alors réceptionné dans la fonction *recv_destopt*. Selon le type de l'option, un switch aiguille vers la bonne procédure en fonction de la Destination Option. Dans le cas d'un BACK le traitement sera fait dans *recv_bnd_ack*. Nous allons modifier le code de cette fonction. Nous conservons la partie de décodage et de traitement d'erreur, par contre juste avant le FORALLCOR, nous plaçons notre conditionnelle pour vérifier dans quel mode fonctionne le MH ; Mobile IPv6 ou UHMMA_MODE mode 1 ou 2.

Type de mouvement Selon le type de mouvement qui a provoqué l'attente de BACK, le MH doit attendre un ou deux BACK :

– inter site : BACK du MA et BACK du HA

– intra site : BACK du MA

Nous utilisons une variable *nbBACK* pour représenter le nombre de BACK attendus. Elle est positionnée à 0 si le MH n'est pas en attente de BACK. La fonction d'enregistrement l'initialise à -1 pour un BACK attendu ou à 3 pour deux BACK.

Sequence number Un BACK du MA doit contenir un champ sequence number égal au compteur *seq_num*. Sans quoi le paquet est rejeté et le MH est toujours en attente de BACK du MA. Si le mobile donctionne selon le mode 2 il ne connait pas le numéro de séquence associé à son HA. Nous avons vu dans les Spécifications Fonctionnelles [3] que le MA va procéder à une synchronisation des numéro de séquence. Donc si le numéro de séquence reçu dans le BACK du MA est supérieur au numéro de séquence que le mobile avait mis dans son paquet d'enregistrement, le mobile attend un BACK du HA avec ce numéro de séquence. Sinon le numéro de séquence du HA sera le même que celui du MA. Donc les numéros de séquence du BACK du MA et de celui du HA doivent toujours être les même.

Exemple :

```
soit un MA qui gère M, et possède un numéro de séquence
a pour le mobile
b pour le HA du mobile

                    a      x
enregistrement : M --> MA --> HA

                     y
acquittement :   M <-- MA
                     z
                 M <-- HA


si a > b alors x == a
               y == a
               z == a
si a < b alors x == b
               y == b
               z == b
```

Traitement Si tous les BACK sont arrivés et valides, il faut détruire le timer qui sert à réémettre l'enregistrement et libérer la mémoire utilisée pour sauvegarder le paquet d'enregistrement. Pour savoir si tous les BACK sont arrivés nous utilisons une variable *nbBACK*. Cette variable est initialisée par la procédure de mouvement, *major_move* ou *minor_move*. Si elle est positionnée à -1, le MH doit attendre un seul BACK, celui du MA. Si nbBACK est positionnée à 3, le MH est en attente de deux BACK, celui du

HA et celui du MA. Il y a eu déplacement inter site.

Une fois tous les BACK attendus reçus, le MH sait qu'il est enregistré. Le mécanisme de confirmation de position peut commencer ; le timer *uMIP_timer* est armé. L'adresse du MA courant est copiée dans *MA_old*.

Algorithme

```
si mobility_mode == UHMMA_MODE 1 ou 2 alors
    si P.source == MA_address courant alors
        si P.sequence_number != seq_num alors
            finir
        sinon
            cas nbBACK parmi
            -1 :
                détruire_timer (reemis_timer)
                MA_old <-- MA_address
                rendre_mémoire (reemit_reg_packet)
                uMIP_timer <-- lancer_timer (send_old_position)
                nbBACK <-- 0
            3 :
                nbBACK <-- 1
                VCOA <-- P.BACK.VCOA
                ajouter_adresse (VCOA)
            2 :
                détruire_timer (reemis_timer)
                MA_old <-- MA_address
                rendre_mémoire (reemit_reg_packet)
                uMIP_timer <-- lancer_timer (send_old_position)
                nbBACK <-- 0
    sinon
si P.source == HA.addr alors
            cas nbBACK parmi
            3 :
                nbBACK <-- 2
            1 :
                détruire_timer (reemis_timer)
                MA_old <-- MA_address
                rendre_mémoire (reemit_reg_packet)
                uMIP_timer <-- lancer_timer (send_old_position)
                nbBACK <-- 0
```

```
        sinon
            finir
sinon
    Mobile IPv6 classique
```

2.6.3 Binding Request (BR)

Nous savons que les CH peuvent envoyer des BR lorsque la durée de vie de leur binding dans leur cache est proche de la fin. Pour les CH externes au site où se trouve le MH actuellement, il n'y a pas de problème. Les BR seront interceptés par le MA et jetés. Par contre les CH se trouvant dans le même site que le MH possèdent un binding H@ / PCOA. Nous avons fait en sorte que le MA envoie des BU plus fréquemment aux CH locaux, mais il peut toujours survenir le cas où un CH local émet un BR vers le MH. Dans ce cas, dans la procédure *recv_bnd_req* nous devons tester si le MH tourne Mobile IPv6 ou UHMMA. Dans le cas de UHMMA mode 2, le BR doit tout simplement être ignoré.

2.6.4 Annonce des routeurs

Chaque MH doit être capable de décoder une annonce de routeur contenant l'adresse du MA. Ceci est fait dans la fonction *recv_rtadv* (icmp.c). Il existe une boucle qui décode toutes les options contenues dans un paquet router advertisement. Nous ajoutons un case dans le switch pour traiter une nouvelle option ; MA address. Si l'adresse lue dans l'option n'est pas la même que *MA_address* nous sommes dans la situation où un nouveau MA vient d'être déployé sur le site. Son adresse se propage à travers tous les routeurs du site. Le MH doit s'enregistrer auprès de ce nouveau MA. Le changement de MA s'apparente à un *minor_move*.

Algorithme

```
case NDX6_MA :
    A <-- P.opt.MA_address
    si A != MA_address alors
        MA_address <-- A
        send_uhmma_minor_reg()
break;
```

3 Protocole uMobile IPv6

3.1 Le BS

Nous appellerons le Base Station BS la machine qui route les paquets vers des réseaux capables / autorisés à accueillir des MH. Le routeur doit avoir connaissance qu'il est un BS. Nous utiliserons pour cela un flag, initialisé par défaut à 0. Un argument de la ligne de commande permettra de le modifer. Nous proposons -B. Le flag se nomme *BS_op*, il s'agit d'une variable globale.

3.1.1 Réception de l'annonce du MA

Un BS connaît les informations sur le MA grâce aux annonces de celui-ci : le MA emet périodiquement des paquet avec une destination option mobility agent annonce. Nous utilisons une structure pour concerver les informations du MA :

- adresse
- mode
- longueur de préfix

Réception Les annonces seront traitées dans la fonction *recv_destop* car ce sont des Destination Option. Un switch détermine le type de la Destination Option et aiguille sur le bon traitement. Nous ajoutons un case pour le cas mobility agent annonce.

```
case OPT6_MA_ANNONCE :
    cp++ ;
    len = *cp++ ;
    recv_ma_ann() ;
    cp +=len ;
break ;
```

recv_ma_ann Cette fonction décode la destination option reçue. Si l'adresse lue n'est pas la même que la dernière mémorisée, il faut initialiser la variable MA avec cette nouvelle valeur.

Algorithme

```
si BS_op == 1 alors
    A <-- P.Destination Option.MA
```

```
    si A != MA_address alors
        MA_address <-- A
        mode <-- P.Destination Option.mode
        plen <-- P.Destination Option.plen
```

3.1.2 Annonce du MA

Un BS doit ajouter l'adresse du MA aux annonces de routeur qu'il envoie sur le réseau (router advertisement). Ces paquets peuvent véhiculer des options (MTU, préfix, source). Nous créons une nouvelle option : MA address.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     type      |   longueur    |      flags    | pref. length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                       MA address                              |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

type : type de l'option, nous proposons 6. length : longueur de l'option, ici 20. MA address : adresse du MA local.

La fonction *send_rtadv* (icmp.c) construit un paquet ICMPv6 et le transmet au noyau pour envoi sur le réseau. Nous allons ajouter dans le paquet construit une option MA address, définie ci-dessus. Nous insérons notre option après le MTU et les préfix ; donc après la boucle *LOOPALLPP ()* et avant le calcul du checksum.

3.2 Le MA

Un MA est une machine qui va tourner ndpd-router. Pour se différencier des autres BS, il doit avoir conscience de sa condition de MA. Nous utilisons un flag *MA_op*, initialisé par défaut à 0. Un argument de la ligne de commande permet de la changer : -M. Ce paramètre doit être suivi du mode de fonctionnement que le MA peut autoriser. Le mode 2 inclut le mode 1.

On ajoute un case dans la boucle de décodage des options :

```
case 'M' :
if (*optarg == '1')
    MA_op = 1;
else if (*optarg == '2')
```

```
    MA_op =2;
else
    exit(3);
```

3.2.1 Structures de données

Le MA va gérer une liste de MH. Nous utiliserons une liste de structures mobile, en y apportant de nouveaux champs :

```
struct mobile {
    TAILQ_ENTRY(mobile) mb_list;    /* mobile list */
    struct in6_addr home;           /* home address */
    struct in6_addr careof;         /* care-of address */
    struct in6_addr me;             /* my address */
    struct timeout *tp;             /* timeout */
    u_long refresh;                 /* last refresh time */
    u_int lifetime;                 /* life time */
    u_short seq;                    /* sequence number */
    u_char flags;                   /* flags (from update) */
    u_char plen;                    /* prefix length */
    caddr_t sap;                    /* pointer to SA */
#ifdef HOME_AGENT
    struct prefix *pp;              /* associated prefix */
    struct anycast *ap;             /* proxy entry */
    struct tunnel *tup;             /* tunnel to client */
#endif
#ifdef UHMMA
    struct anycast *ap;             /* proxy entry */
    struct tunnel *tup;             /* tunnel to client */
    TAILQ_HEAD(, corr) CH;                 /* CH list */
    struct in6_addr VCOA;
    struct corr * HomeAgent;
#endif
};
```

Nous utiliserons les primitives TAILQ* pour gérer notre liste de CH. Nous ne prévoyons pas de flag pour spécifier le mode selon le mobile désire être pris en charge. Il suffit de consulter le pointeur CH, s'il est NULL, le mobile tourne le mode 1, sinon il y aura au minimum le HA du mobile.

Chaque MH est associé à un timer. Ce timer indique la durée pendant laquelle le MA doit gérer le MH. Le MH peut bien sûr prolonger la durée de cette gestion (relancer le timer) par simple envoi d'un paquet de maintien de position.

3.2.2 Interception

Le MA doit intercepter tous les paquets à destination des VCOA des MH qu'il gère. Pour cela nous allons utiliser le mécanisme de proxy créé pour Mobile IPv6. Ce mécanisme permet d'indiquer au noyau d'intercepter des paquets à destination d'une adresse A et de les tunneler vers une adresse B. Ceci est utilisé par les HA pour rediriger les paquets à destination de MH en déplacement. Nous ajoutons une conditionnelle pour vérifier si les paquets interceptés sont déjà encapsulés, auquel cas le paquet vient d'un CH non informé de la nouvelle position du MH. En plus de faire suivre ce paquet, il est nécessaire de le remonter à ndpd, pour que celui-ci fasse parvenir un BU au CH.

C'est également à ce niveau qu'il faut détecter les binding request (BR). Un BR doit être rejeté et non routé. Par contre il doit être remonté à ndpd pour analyse et envoi de BU.

Le code pour mettre en place le proxy se trouve dans *recv_home_reg*. Nous nous baserons dessus pour mettre en place notre proxy. Le proxy en lui même est implémenté dans le noyau. Nous devrons donc apporter nos tests (BR ou encapsulation) dans le noyau.

3.2.3 Gestion des CH ; mode 2

Il y a deux façons d'apprendre des CH.

1. smooth handoff
2. interception de paquets encapsulés

Dans les deux cas, un envoi périodique de BU est à mettre en place. Dès qu'un nouveau CH est appris, nous l'enregistrons dans la liste des CH. Il est important de remplir tous les champs de la structure corr car nous les utiliserons par la suite. Nous emploierons le champ careof pour contenir la PCOA du MH. Nous aurons également besoin d'un nouveau champ dans la structure corr, *struct in6_addr VCOA*. Nous créons un timer pour cadencer les envois de BU. Pour cela nous utilisons du code écrit pour Mobile IPv6. Des fonctions ont été implémentées pour gérer les retransmissions. Il est utilisé par les MH pour l'envoi de BU périodique vers ses CH. Lorsque nous insérons un CH dans une liste, nous appelons la fonction *start_retrans*. Nous passons à cette fonction un pointeur sur le CH, et elle va s'occuper de lancer la retransmission des BU périodiquement. Pour cela elle créait un timer et lui donne *schedule_ret* comme traitant et le CH en paramètre. La fonction *schedule_ret* appelle soit *send_bnd_upd* soit *send_home_reg* selon la nature du CH (HA ou non). Ces deux fonctions construisent un paquet de type :

- src = corr.careof
- dst = corr.me
- BU
- H@, corr.home

Nous allons placer une conditionnelle dans ces deux fonctions. Si *MA_op == 2* alors le paquet construit devra être du type :

- src = MA
- dst = corr.me
- BU avec flag C, corr.careof si corr.me (site local, corr.VCOA sinon
- H@, corr.home

Pour arrêter la retransmission, par exemple après avoir reçu un smooth handoff, il suffit d'appeler la fonction *cancel_retrans*.

Il est ici important de différencier les CH externes et internes au site. D'une part pour savoir quelle adresse placer dans le BU, et d'autre part pour calculer la fréquence de l'envoi de BU vers ces CH. Nous savons en effet que pour éviter au CH internes d'envoyer des BR vers les MH, nous devons envoyer des BU à une plus grande fréquence que pour les CH externes. Le timer associé aux CH internes sera donc initialisé avec un temps plus petit.

Lorsqu'un MA reçoit la liste des CH d'un MH après un smoth handoff, il doit parcourir cette liste et mettre à jour les champs careof et VCOA.

3.2.4 Enregistrement des MH

Les paquets d'enregistrement des MH sont des BU. Ils contiennent des Destination Option, ils seront donc traités dans un premier temps par *recv_destop*. Une boucle décode toutes les options une par une. Si elle tombe sur une Destination Option BU, elle n'est pas traitée de suite car une Destination Option H@ est nécessaire. Il faut donc déférer et faire un tour de plus dans la boucle pour décoder la H@. En possession du BU et de la H@ nous pouvons lancer la fonction *recv_bnd_upd*. Il est bon de noter que le paquet est passé par le noyau pour arriver jusqu'ici. Le noyau a déjà échangé la H@ de la Destination Option avec l'adresse source du paquet ipv6.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     COA       |       MA       |  BU +  H@    |    data       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     H@        |       MA       |  BU +  COA   |    data       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Nous devons décoder les sub-options du paquet.

Algorithme

```
tant que P.Destination Option != null
    D <-- P.Destination Option
    si D.type == H@ alors
            COA <-- D.H@
            si D.HA != null alors
                HA <-- D.HA

si D.type == BU alors
    ... (reste de la boucle inchangée)

si HA != null alors
    si MA_op == 1 ou 2 alors recv_uhmma_reg (P, COA, HA)
    sinon
        B <-- construire_paquet (BACK, P.source, "refused, not a MA")
        émettre (B)
sinon
    si P.BU.flag == HR alors recv_home_reg(P, COA)
    sinon
        si P.BU.flag == BU alors recv_bnd_upd(P, COA)
```

recv_uhmma_reg (P, COA, HA) Cette fonction va finir de décoder le paquet. Elle va ensuite traiter la demande d'enregistrement. Plusieurs cas sont alors possible, cf. Enregistrement des MH dans la spécification fonctionnelle.

Les BACK à retourner au MH doivent avoir un sequence number égal au sequence number lu dans le paquet d'enregistrement.

Algorithme

```
Mo <-- P.BU.MA
H@ <-- P.BU.H@
x <-- parcourir (listeMH)
si listeMH(x).H@ == H@ alors
    si listeMH(x).PCOA == COA alors
        si Mo == MA_address alors
            A <-- construire_paquet (BACK, P.source,
                    P.sequence_number, "accepted")
            émettre (A)
            listeMH(x).timer <-- relancer_timer(kill_mobile,
                                        listeMH(x))
        sinon
            A <-- contruire_paquet (BACK, P.source,
                    P.sequence_number, "accepted")
            émettre (A)
            si listeMH(x).HA != NULL alors
                B <-- construire_paquet (BU, listeMH(x).HA.addr,
                        listeMH(x).HA.sequence_number,
                        listeMH(x).VCOA)
                émettre (B)
            listeMH(x).timer <-- relancer_timer (kill_mobile, MH)
    sinon
        listeMH(x).PCOA <-- COA
        fermer_tug (listeMH(x).tug)
        listeMH(x).tug <-- ouvrir_tug (COA)
        A <-- construire_paquet (BACK, P.source,
                P.sequence_number, "accepted")
        émettre (A)
        listeMH(x).timer <-- relancer_timer (kill_mobile,
                                    listeMH(x))
        si listeMH(x).HA != NULL alors
          y <-- parcourir listeMH(x).listeCH
            si listeMH(x).listeCH(y).prefix == prefix_local()
            alors
                B <-- construire_paquet (BU,
                       listeMH(x).listeCH(y).addr,                                                                       listeMH(x).listeCH(y).sequence_number,                                              listeMH(x).PCOA)
            émettre (B)
sinon
```

```
    si MA_old == adresse indeterminée alors
        new <-- créer_enregistrement (listeMH, COA, VCOA, H@, HA)
        listeMH(new).ap <-- créer_proxy (H@, COA, VCOA)
        A <-- construire_paquet (BACK, P.source, P.sequence_number,
                        "accepted")
        émettre (A)
        listeMH(new).timer <-- lancer_timer (kill_mobile, MH)
        listeMH(new).tug <-- ouvrir_tug (PCOA)
        if pkt.BU.flags.mode == 2
            B <-- construire_paquet (BU, listeMH(x).HA.addr,
                listeMH(x).HA.sequence_number, listeMH(x).VCOA)
        émettre (B)
    sinon
        si H@ == PCOA alors
            B <-- construire_paquet (BU, Mo, P.sequence_number,
                                H@)
        sinon
            B <-- construire_paquet (BU, Mo, P.sequence_number,
                                VCOA)
        émettre (B)
        smooth_timer <-- lancer_timer (resend_smooth, B)
```

Si une opération de l'enregistrement n'aboutit pas, il faut retourner à l'expéditeur un paquet BACK avec un status refusé.

La fonction *kill_mobile(x)* retire l'entrée x de la liste des MH gérés. Elle détruit tous les timers associés au émission de BU vers les CH (dans le cas du mode 2)et termine l'activité de proxy pour ce MH.

3.2.5 Confirmation de position

Les MH gérés par le MA envoie périodiquement des BU(COA/H@) pour confirmer leur position, sans quoi leur gestion ne sera plus assurée. Ces BU sont des BU Mobile IPv6 classiques avec un flag P. Ce flag permet de les différentier des autres BU. Ils seront traités dans *recv_destop*. Nous allons modifier les appels de fonctions (situés après la boucle de décodage) pour prendre en compte les BU avec flag P. Nous appellerons alors une fonction, *recv_confirm_pos*, qui traitera le paquet.

recv_destop (boucle de décodage ...)

```
si HA != null alors
```

```
    si MA_op == 1 ou 2 alors recv_uhmma_reg (P, COA, HA)
    sinon
        B <-- construire_paquet (BACK, P.source, "refused, not a MA")
        émettre (B)
sinon
    si P.BU.flag == HR alors recv_home_reg(P, COA)
    sinon
        si P.BU.flag == BU alors
            si P.BU.flga.P == 1 alors
                si MA_op == 1 ou 2 alors
                     recv_confirm_pos(P, COA)
                sinon
                     B <-- construire_paquet (BACK, P.source,
                                  "refused, not a MA")
                     émettre (B)
            sinon recv_bnd_upd(P, COA)
```

recv_confirm_pos (P, COA C) Le MH émet régulièrement un paquet de confirmation de position.

```
x <-- parcourir listeMH
si P.source == listMH(x).H@ alors
    listeMH(x).timer <-- relancer_timer
    listeMH(x).seq_num <-- P.sequence_number
sinon
    B <-- construire_paquet (BACK, P.source, "refused, not registered")
    émettre (B)
```

3.2.6 Smooth handoff

Il y a ici plusieurs aspect à aborder. Le smooth handoff étant une négociation entre deux MA, nous verrons dans ce même paragraphe le déroulement des opérations des deux cotés, coté initiateur et coté esclave.

Réception d'une initiation Une initiation de smooth handoff est un paquet de type BU, contenant une Destination Option BU avec VCOA et une Destination Option H@, le flag S est positionné. Ce paquet sera remonté par le noyau à ndpd, car il contient des Destination Option. Il sera ensuite traité dans *recv_destop*. Nous avons déjà vu cette fonction à plusieurs reprises, et nous connaissons bien sa structure. Une boucle décode les Destination Option puis aiguille sur le bon traitement. Nous avons déjà modifié cette fonction dans le paragraphe Enregistrement des MH. Nous apportons ici de nouvelles modifications dans les appels de fonction suivant la sortie de la boucle de décodage. Il existera à présent une nouvelle sorte de BU, les initiateurs de smooth handoff. Ils seront reconnaissables par le flag S. Nous les traiterons dans une fonction, *recv_smooth*.

Algorithme

```
si HA != null alors
    si MA_op == 1 ou 2 alors recv_uhmma_reg (P, COA, HA)
    sinon
        B <-- construire_paquet (BACK, P.source, "refused, not a MA")
        émettre (B)
sinon
    si P.BU.flag == HR alors recv_home_reg(P, COA)
    sinon
        si P.BU.flag == BU alors
            si P.BU.flag.P == 1 alors
                si MA_op == 1 alors
                    recv_confirm_pos(P, COA)
                sinon
                    B <-- construire_paquet (BACK, P.source,
                                 "refused, not a MA")
                    émettre (B)
            sinon
                si P.BU.flag.S == 1 alors
                si MA_op == 1 alors
                        recv_smooth(P, COA)
                    sinon
                        B <-- construire_paquet (BACK, P.source,
                                  "refused, not a MA")
                        émettre (B)
                sinon
                    sinon recv_bnd_upd(P, COA)
```

recv_smooth (P, COA) Cette fonction doit vérifier que le MH pour lequel est demandé le smooth handoff est bien dans sa liste de MH gérés, ensuite que le séquence number du paquet est bien supérieur au compteur associé au MH (dans la liste des MH gérés). Si cela n'est pas le cas, le paquet est rejeté, un BACK est retourné avec le status correspondant.

Si le séquence number est valide, le MA construit un paquet d'acquittement à envoyer à l'initiateur du smooth handoff. Ce paquet sera constitué d'une Destination Option BACK avec en sub-option l'ensemble de la liste des CH du MH concerné, si une telle liste existe ; cela signifie que ce mobile est géré selon le mode 2. Dans la liste des MH gérés localement, le MA doit mettre à jour le champ PCOA du MH avec l'adresse VCOA sortie du BU. Cela permet de faire suivre les paquets vers la nouvelle position du MH. Ce mécanisme de redirection est temporaire. Au bout d'un temps fini, nous devons le terminer. Pour cela nous n'avons pas besoin d'implémenter de fonctionnalité supplémentaire. Le mécanisme de maintien de position que nous avons mis en place, va arrêter automatiquement cette redirection. Le timer associé au MH est toujours en fonctionnement. Lorsqu'il expirera, son traitant est chargé de supprimer l'entrée du MH de la liste des MH gérés. C'est exactement ce que nous recherchons ici. Nous devons tout de même prévoir une mécanique pour arrêter le fonctionnement du proxy mis en place pour le MH. Dans la fonction *kill_mobile* nous ajoutons l'appel à une fonction qui terminera le proxy. Afin d'obtenir une durée toujours égale pour la redirection, nous relançons le timer associé au MH une dernière fois à l'arrivée d'un BU initiateur. L'envoi de BU pour un MH fonctionnant en mode 2 doit être terminé à partir de cet instant, sauf si le mobile vient de revenir sur son réseau mère. Auquel cas, il faut envoyer un BU (H@/H@) à tous les CH de la liste avant de la supprimer. Le nouveau MA est chargé de ce rôle. Nous devons détruire tous les timers associés aux envois périodiques de BU vers les CH de ce MH. Si l'ancien MA intercepte des paquets contenant des BR ou étant des paquets encapsulés, ce MA ne doit plus envoyer de BU. C'est pourquoi avant envoi d'un BU le MA doit s'assurer que le MH est toujours dans son site. Il peut vérifier cette information en analysant le préfix du champ PCOA dans sa liste de MH gérés. Si le préfix est celui du site, alors le MH est présent, sinon il est parti.

Nous allons ajouter une conditionnelle juste après le décodage du paquet et avant le début du traitement de la Destination Option.

Algorithme

```
x <-- parcourir listeMH
si listeMH(x).H@ == P.source alors
    si P.sequence_bumber > listeMH(x).seq_num alors
        listeMH(x).PCOA <-- COA
        listeMH(x).timer <-- relancer_timer (kill_mobile,
```

```
                                        listeMH(x))
            y <-- parcourir listeMH(x).listeCH
                détruire_timer listeMH(x).listeCH(y).timer
            B <-- construire_paquet (BACK, P.source,
                            P.sequence_number, "accepted",
                            listeMH(x).listeCH)
            émettre (B)
        sinon
            B <-- construire_paquet (BACK, P.source,
                              P.sequence_number,
                       "refused, sequence number too small")
            émettre (B)
    sinon
        B <-- construire_paquet (BACK, P.source,
                         P.sequence_number,
                         "refused, not registered")
        émettre (B)
```

Emission d'un smooth handoff Lorsqu'un MA reçoit un paquet d'enregistrement d'un MH qui vient d'un autre site (qui n'est pas le réseau mère), il doit initier un smooth handoff. Pour cela il compose un paquet avec une Destination Option BU avec VCOA et H@. Le sequence number de ce paquet est le même que celui lu dans le paquet d'enregistrement du MH. Ce paquet est donné au noyau pour émission sur le réseau. Il est obligatoire d'obtenir la liste des CH du MA précédent pour pouvoir assurer correctement la mobilité du MH. Nous armons un timer pour réémission du paquet si le BACK n'arrive pas. Nous utilisons la même mécanique que pour l'émission du paquet d'enregistrement d'un MH. Nous conservons le paquet en mémoire tant que le BACK n'est pas arrivé. Nous utilisons un pointeur *reemis_paquet_ptr*, un timer *reemis_timer* avec son traitant *resend_smooth*. L'algorithme est le même que pour l'enregistrement d'un MH, nous ne le répétons pas ici.

Réception du BACK Le MA initiateur d'un smooth handoff recevra un paquet d'acquittement. Ce paquet contiendra une Destination Option BACK avec en sub-option la liste complète des CH du MH. Le sequence number du BACK doit être le même que celui du BU initiateur. Si le BACK contient une liste de CH, celle-ci est décodée et insérée dans la liste des MH gérés, à l'entrée correspondante. Les BU peuvent maintenant être envoyés pour signifier au CH la nouvelle position du MH. Un parcours des adresses des CH détermine quel type de BU envoyé. Si le CH est local au site, le BU est H@/PCOA. Si le CH est extérieur, le BU est H@/VCOA. Note : le HA doit toujours recevoir un BU avec la VCOA pour ce type d'opération. Nous rappelons ici qu'il faut synchroniser les numéros de séquence du mobile (a) et de son HA (x). Si a>x alors le BU vers le HA doit contenir a et non x. Si a<x alors le BACK vers le mobile doit contenir x et non a. Le MH peut être acquitté. Le MA construit un paquet contenant une Destination Option BACK avec une sub-option VCOA. Le sequence number de la Destination Option est le même que celui qui a servi dans la demande d'enregistrement, puis dans l'initiation du smooth handoff et enfin dans le BACK du smooth handoff.

Le problème est qu'un MA peut avoir initier plusieurs smooth handoff, pour plusieurs mobiles qu'il aurait dans sa liste de gestion. Il est donc en attente de plusieurs BACK. Lorsqu'un BACK arrive, il doit savoir à quelle initiation il correspond. Nous allons implémenter une liste d'initiation de smooth handoff dans le MA. Cette liste s'appellera waitingBack. Nous la manipulerons avec les primitives TAILQ. Elle sera composée de structure de type :

```
struct WBACK
{
    struct mobile * MH;  /* le MH en question */
    struct timeout * tm; /* timer associé à la retransmission */
    struct in6_addr MA;  /* adresse du MA esclave */
    char * paquet;       /* pointeur sur le paquet */
    int seq_num;         /* numéro de séquence
                            du paquet d'enregistrement */
}
```

Lorsque le MA émet son initiation de smooth handoff, il remplit une structure WBACK et l'insère dans la liste. Lorsqu'un BACK arrive, il suffit de parcourir la liste pour vérifier la validité de ce paquet.

Si le paquet contient une liste de CH, celle-ci est sortie et décodée. Elle est affectée au MH. Puis un parcours de cette liste s'impose pour mettre à jour les champs careof et VOCA. Nous nous servons du pointeur sur le MH de la structure WBACK. Ses champ ont déjà été initialisés dans la procédure d'enregistrement du MH.

Les BACK passent par la fonction *recv_destop*, puis sont aiguillés vers *recv_bnd_ack*. Nous allons donc modifier *recv_bnd_ack* pour tenir compte des smooth handoff.

Algorithme

Nous conservons le décodage du paquet et juste avant le traitement spécifique à Mobile IPv6, nous plaçons une conditionnelle.

```
si MA_op == 1 ou 2 alors
    si waitingBack != null alors
        x <-- parcourir waitingBack
        MHc <-- waitingBack(x)
        si P.source == MHc.MA alors
            si P.sequence_number > MHc.seq_num alors
                détruite_timer (MHc.tm)
                rendre_mémoire (MHc.paquet)
                retirer_élément (MHc, waitingBack)
                listeMH(MHc.MH).listeCH <-- P.BACK.listeCH
                A <-- construire_paquet (BACK, P.source,
                          P.sequence_number, "accepted")
                émettre (A)
                y <-- parcourir listeMH(MHc.MH).listeCH
                    listeMH(MHc.MH).listeCH(y).careof <--
                               MHc.MH.careof
                    listeMH(MHc.MH).listeCH(y).VCOA <--
                               MHc.MH.VCOA
            sinon
                A <-- construire_paquet (BACK, P.source,
                              P.sequence_number,
                   "refused, sequence number too small")
                émettre (A)
        sinon
            Mobile IPv6 ...
```

Références

[1] David B. Johnson et Charles Perkins. Mobility Support in IPv6. Internet Draft, draft-ietf-mobileip-ipv6-07.txt. Novembre 1998.

[2] Claude Castelluccia. A hierarchical Mobile IPv6 Proposal. Technical Report. Novembre 1998.

[3] Claude Castelluccia, Ludovic Bellier. Unified Hierarchical Mobility Management Architecture - Spécifications Fonctionnelles. Novembre 1999.