JEAN LOUIS DE OLIVEIRA

PROPOSTA DE UM MECANISMO DE NOMES PARA SUPORTAR MACRO-MOBILIDADE COM IP MÓVEL

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: *Redes de Computadores e de Telecomunicações*

Orientador: Prof. Dr. Edgard Jamhour

Oliveira, Jean Louis
O48p Proposta de um mecanismo de nomes para suportar macro-mobilidade com
2005 IP móvel / Jean Louis Oliveira ; orientador, Edgard Jamhour. – 2005.
xi, 105 f. : il. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Paraná, Curitiba, 2005
Inclui bibliografia

1. Sistemas de comunicação móvel. 2. TCP/IP (Protocolo de rede de computação). 3. Telefonia celular. I. Jamhour, Edgard. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada. III. Título.

CDD 20. ed. 621.3845
004.62
384.64

Agradecimentos

Aos meus pais, João e Ailê, pelo apoio e compreensão durante toda a minha vida.

À minha esposa, Tatiane, e meu filho, Gabriel, pelo apoio durante toda a execução deste trabalho.

Ao orientador e amigo, Prof. Edgard Jamhour, pela paciência e dedicação nas longas discussões sobre os rumos deste trabalho.

Aos amigos conhecidos durante o Mestrado, e demais amigos que de alguma maneira colaboraram na realização desta dissertação.

Agradeço também aos colegas e professores do PPGIA que me acompanharam nestes mais de dois anos.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

3G	Terceira Geração
AAA	*Authentication, Authorization and Accouting*
ACK	*Acknowledgement*
AR	*Access Router*
BU	*Binding Update*
CDMA	*Code Division Multiple Access*
CDMA2000	*Code Division Multiple Access 2000*
CH	*Correspondent Host*
CIP	*Cellular IP*
CoA	*Care-of-Address*
CRL	*Certificate Revocation List*
CSR	*Certificate Signing Request*
DH	*Diffie - Hellman*
DHCP	*Dynamic Host Configuration Protocol*
DHMIP	*Dynamic Hierarchical MIP*
DNS	*Domain Name System*
DNSSEC	*DNS Security Extensions*
FA	*Foreign Agent*
GGSN	*Gateway GPRS Suport Node*
GPRS	*General Packet Radio Service*
HA	*Home Agent*
HMAC	*Keyed – Hashing for Message Authentication*
HMIP	*Hierarchical MIP*
HTTP	*Hyper Text Transmission Protocol*
IDMP	*Intra Domain Mobility Management Protocol*
IETF	*Internet Engineering Task Force*
MAC	*Message Authentication Code*
MAP	*Mobility Anchor Point*
MD5	*Message – Digest version 5*

MH	*Mobile Host*
MIP	*Mobile IP*
MN	*Mobile Node*
NS	*Name Server*
NXT	*Non-Existent*
PDSN	*Packet Data Serving Node*
PPP	*Point-to-Point Protocol*
RAN	*Radio Access Network*
RCODE	*Response Code*
RFC	*Request for Comments*
RNC	*Radio Network Controller*
RR	*Resource Record*
RRset	*Resource Record Set*
RTT	*Round Trip Time*
SHA	*Secure Hash*
SIG	*Signature*
SOA	*Start Of Authority*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TIMIP	*Terminal Independent Mobile IP*
TSIG	*Transaction Signature*
TLS	*Transport Layer Security*
TTL	*Time to Live*
UDP	*User Datagram Protocol*
UML	*User-mode Linux*
UMTS	*Universal Mobile Telecommunications System*
WAP	*Wireless Application Protocol*
WLAN	*Wireless LAN*
WWW	*World Wide Web*

Resumo

Nos últimos anos tem-se observado a popularização dos serviços de comunicação sem fio e o suporte para usuários móveis. Para esse fim, distinguem-se as tecnologias celulares (e.g., CDMA2000, GPRS e Edge), tecnologias para WLAN (e.g. 802.11 a,b,g e n) e WMAN (802.16). Essas tecnologias podem ser combinadas para formar uma única rede sem fio, a fim de obter uma ampla cobertura de serviços de comunicação em presença de mobilidade. A versão do protocolo IP com suporte à mobilidade (Mobile IP) é um importante elemento para construção dessa rede. A mobilidade ao longo de uma rede de comunicação sem fio, formada pela integração de várias redes administradas independentemente é denominada macro-mobilidade.

Um dos problemas encontrados na macro-mobilidade é a dificuldade de manter um identificador único para o usuário quando este comuta entre redes administradas independentemente. Nosso desafio é propor um mecanismo de nomes unificado para suporte a macro-mobilidade em redes móveis com tecnologias heterogêneas, através de alterações na implementação de entidades relacionadas ao serviço IP Móvel.

A nossa proposta para manter a rastreabilidade do nó móvel é usar um mecanismo de DNS (*Domain Name System*) dinâmico integrado ao processo de autenticação do serviço de IP Móvel. Permitir alterações de registros DNS pode levar à resultados indesejáveis, pois é necessário evitar a vinculação indevida do nome que identifica o nó móvel com um endereço IP que não lhe pertença num dado instante.

Em nossa proposta, consideramos dois cenários de aplicação: no primeiro cenário, a atualização da entrada DNS do nó móvel é disparada pela infra-estrutura de rede (mais especificamente a entidade HA, do padrão IP móvel); no segundo, pelo próprio nó móvel. O desenvolvimento dessas estratégias e sua respectiva avaliação são os principais componentes deste trabalho.

Palavras-Chave: Macro-mobilidade**,** Redes Móveis, IP Móvel, DNS Dinâmico

Abstract

During the last years, one observes a growing interest on mobile and wireless technologies. In this context, one can distinguish technologies based on cellular networks (e.g., GPRS, CDMA2000) and technologies for deploying WLAN (e.g. 802.11 a,b,g e n) and WMAN (802.16). These technologies can be combined to create a single wireless network, in order to achieve a wide coverage of mobile services. The Mobile IP, i.e., an IP-based mechanism for supporting mobility, is an important component for building such a network. The mobility across a network composed by the aggregation of independently managed networks is called macro-mobility.

An important problem faced for supporting macro-mobility is the difficulty in keeping a unique identifier for a user that moves across independently managed networks. Our motivation is to propose a unified resolution name mechanism for supporting macro-mobility by introducing new functionalities in the entities related to the Mobile IP mechanism. Our strategy for tracking the mobile node identity is to deploy a dynamic DNS mechanism integrated with the Mobile's IP authentication process.

Without safety mechanisms, permitting modifications in the DNS registers can lead to undesirable results. At any instant, it is necessary to assure that the name that identifies a mobile node is mapped to the current mobile node's IP address.

In our proposal, we have considered two application scenarios. In the first scenario, the update of the DNS entry is triggered by the network infra-structure (i.e., the HA entity defined by the Mobile IP standard). In the second scenario, the mobile node is responsible for triggering the update. Describing and evaluating both strategies are the main components of this work.

Key-mot: Macro-mobility**,** mobile networks, mobile IP, dynamic DNS

Capítulo 1 - Introdução

Nos últimos anos tem-se observado a popularização dos serviços de comunicação sem fio e o suporte para usuários móveis. Para esse fim, distinguem-se as tecnologias celulares (e.g., CDMA2000, GPRS e Edge), tecnologias para WLAN (e.g. 802.11 a,b,g e n) e WMAN (802.16). Essas tecnologias podem ser combinadas para formar uma única rede sem fio, a fim de obter uma ampla cobertura de serviços de comunicação em presença de mobilidade. A versão do protocolo IP com suporte a mobilidade (Mobile IP) é um importante elemento para construção dessa rede. Quando tecnologias de comunicação heterogêneas são integradas, elas podem ser administradas por uma única entidade (acoplamento forte) ou por entidades distintas (acoplamento fraco). A mobilidade ao longo de uma rede de comunicação sem fio, formada pela integração de várias redes administradas por entidades distintas é denominada macro-mobilidade. Nesse cenário, o nó móvel precisa se autenticar, eventualmente, numa nova entidade administradora durante seu processo de mobilidade. Quando isto acontece, o nó móvel muda o seu endereço IP.

Um dos problemas encontrados na macro-mobilidade com acoplamento fraco é a dificuldade de manter um identificador único para o usuário quando este comuta entre redes administradas independentemente.

1.1) Desafio

Nosso desafio é propor um mecanismo de nomes unificado para suporte a macro-mobilidade em redes móveis com tecnologias heterogêneas, através de alterações na implementação de entidades relacionadas ao serviço IP Móvel. A nossa proposta para manter a rastreabilidade do nó móvel é usar um mecanismo de DNS dinâmico integrado ao processo de autenticação do serviço de IP Móvel. Permitir alterações de registros DNS pode levar a resultados indesejáveis, pois é necessário evitar a vinculação indevida do nome que identifica o nó móvel com um endereço IP que não lhe pertença num dado instante. Também é necessário que o processo de atualização do DNS seja escalável e que não tenha impacto significativo na comunicação com o nó móvel.

1.2) Motivação

A construção de redes de comunicação sem fio com suporte à mobilidade é uma tendência importante para as telecomunicações. Projetos denominados "Redes Metropolitanas Móveis" estão sendo abordados tanto pela academia como pela iniciativa privada. Devido à dificuldade de garantir a cobertura usando uma única tecnologia, o processo de construção dessas redes freqüentemente utilizará uma combinação de tecnologias heterogêneas e poderá combinar esforços de entidades administradoras distintas. Por exemplo, já estão disponíveis no mercado dispositivos adaptadores para usuários móveis que oferecem conectividade para tecnologias do tipo WLAN (i.e., 802.11g) e celular (e.g., GPRS). Quando o usuário se encontra dentro de uma região com cobertura WLAN (*Wireless LAN*) com a qual ele tenha contrato, o adaptador utiliza o serviço WLAN. Caso o usuário não tenha cobertura WLAN, o adaptador comuta para GPRS (*General Packet Radio Service*). Caso essas redes sejam administradas por entidades distintas, o usuário receberá um novo endereço IP, mesmo que o protocolo IP com suporte a mobilidade (IP móvel) seja utilizado. Este cenário é caracterizado como "macro-mobilidade com acoplamento fraco".

Para muitos casos será necessário manter a rastreabilidade do usuário mesmo na presença de mobilidade. Por exemplo, para que o usuário móvel possa receber uma comunicação de voz sobre IP (VoIP). Dessa forma, caracteriza-se a necessidade de introduzir um mecanismo de nomes que permita localizar o usuário.

1.3) Proposta

Esta dissertação tem como objetivo propor um mecanismo de nomes unificado para *macro-mobilidade* com acoplamento fraco através de alterações na implementação de entidades relacionadas ao serviço IP Móvel. Em nossa proposta, consideramos dois cenários de aplicação: no primeiro cenário, a atualização da entrada DNS do nó móvel é disparada pela infra-estrutura de rede (mais especificamente a entidade HA, do padrão IP móvel); no segundo, pelo próprio nó móvel. O desenvolvimento dessas estratégias e sua respectiva avaliação são os principais componentes deste trabalho.

1.4) Contribuições

As contribuições trazidas por este trabalho de pesquisa são:

- Definição de um mecanismo de nomes unificado para usuários móveis na presença de macro-mobilidade com acoplamento fraco;
- Definição dos mecanismos necessários para evitar a vinculação indevida da entrada de DNS correspondente ao nó móvel com um endereço IP que não lhe pertença num dado instante;
- Definição e comparação de duas estratégias para implementação do mecanismo proposto. Uma baseada em alterações no software do nó móvel (suporte ao IP móvel), e outra na infra-estrutura de rede (mais especificamente na entidade *home agent* do padrão IP móvel);
- Implementação do mecanismo proposto utilizando um pacote de IP móvel disponível em fonte aberta (*Dynamics - HUT Mobile IP*);
- Avaliação das estratégias propostas, utilizando um ambiente de simulação em UML (User-Mode Linux), em termos de escalabilidade e da latência introduzida no processo de comunicação entre um *host* externo e um nó móvel.

1.5) Organização

A dissertação está estruturada em sete capítulos, organizados da seguinte forma:

- Capítulo 1 – Introdução: Faz uma apresentação geral do contexto deste trabalho.
- Capítulo 2 – Redes Móveis e IP Móvel: Descreve os conceitos básicos de redes móveis necessários para o entendimento da dissertação. São abordados a definição do problema que procuramos resolver, aspectos de mobilidade, definição de macro-mobilidade, macro-mobilidade com IP Móvel e a definição de micro-mobilidade.
- Capítulo 3 – Trabalhos relacionados à Macro-mobilidade: Apresenta um estado da arte com relação à macro-mobilidade. São apresentados o protocolo DHMIP

(*Dynamic Hierarchical MIP*), a definição de *Context Transfer,* uma arquitetura inter-redes WLAN / GPRS e uma arquitetura inter-redes WLAN / CDMA.

- Capítulo 4 – Atualização Dinâmica de DNS: Descreve os conceitos básicos de DNS necessários para o entendimento da dissertação. É abordada a definição do *DNS Update*.
- Capítulo 5 – Proposta: Propõe um novo algoritmo baseado em um mecanismo de nomes para suportar macro-mobilidade, através de alterações na implementação de entidades relacionadas ao serviço IP Móvel. São apresentadas as estratégias para implementação do mecanismo de nomes, o mecanismo de atualização baseado no HA (*Home Agent*), o mecanismo de atualização baseado no cliente e uma análise comparativa entre as duas estratégias.
- Capítulo 6 – Avaliação: Apresenta a avaliação dos testes do mecanismo de nomes para suportar macro-mobilidade com IP Móvel. São apresentados o ambiente de avaliação, a avaliação do tempo de resposta e a avaliação de escalabilidade.
- Capítulo 7 – Conclusão: São apresentadas as conclusões e propostas de trabalhos futuros.

Capítulo 2 – Redes Móveis e IP Móvel

2.1) Introdução

O objetivo desse capítulo é contextualizar nosso trabalho no ambiente de redes móveis e ressaltar que o IP Móvel é o protocolo padrão de macro-mobilidade. Este capítulo está estruturado da seguinte forma: a seção 2.2 apresenta a definição do problema que procuramos resolver; a seção 2.3 discute os aspectos de mobilidade; a seção 2.4 define macro-mobilidade; a seção 2.5 apresenta a macro-mobilidade com IP Móvel; a seção 2.6 define micro-mobilidade; e a seção 2.7 apresenta as considerações finais do capítulo.

2.2) Definição do problema

Com o avanço da tecnologia na área de computação móvel, várias redes móveis já estão utilizando o protocolo IP em suas implementações, outras estão próximas de necessitar de seu uso. Isto ocorre, porque novas aplicações estão surgindo e com elas surge também a necessidade de empregar novas infra-estruturas para suportá-las.

Grande parte das operadoras de redes celulares já tem adotado a tecnologia IP Móvel [1], algumas tecnologias como GPRS [2] e CDMA (*Code Division Multiple Access*) [3] já possuem implementações que empregam este protocolo. Um dos problemas enfrentado ao se implementar IP Móvel em redes celulares é a escassez de endereços IP versão 4 (IPv4) disponíveis para este fim, ou seja, o esquema de endereçamento IPv4 é limitado e não tem como atender a demanda crescente dessa área.

As operadoras de rede *GPRS European* mantêm uma discussão com a IANA (*Internet Assigned Numbers Authority*) e a RIPE (*European Registry*) sobre o estabelecimento de uma política para a distribuição de endereços IP para operação de transmissão de dados por pacotes em redes sem fio. Porém, se considerarmos que todos os endereços IPv4 ainda disponíveis fossem destinados à utilização dessas operadoras, mesmo assim não seria o suficiente para implementar IP Móvel com a escala requerida no mundo inteiro.

O objetivo primordial deste projeto é propor um mecanismo de nomes unificado para *macro-mobilidade* em redes móveis com tecnologias heterogêneas, como WLAN e GPRS, utilizando IPv4 e, mapeando uma entrada DNS do *host* a cada mudança de rede. No nosso trabalho, assumimos que o endereço IP atribuído ao nó móvel é público. O problema de escassez de endereços é um aspecto que será abordado em desenvolvimentos futuros deste projeto.

2.3) Aspectos de Mobilidade

O requisito básico na Internet sem fio é suportar mobilidade através de várias redes de acesso. Mobilidade significa a habilidade de um *mobile host* (MH) extrapolar a natureza dependente da localização dos endereços IP através de um mecanismo de tradução apropriado, e enviar e receber datagramas eficientemente de qualquer lugar.

Em [4], vemos que dois tipos de mobilidade, por comutação de circuitos e por comutação de pacotes, são amplamente considerados, baseado no tipo da comutação de datagramas na rede analisada. Uma vez que o IP é agora o protocolo de camada de rede dominante para redes comutadas por pacote, a mobilidade por comutação de pacotes é freqüentemente referenciada como mobilidade IP.

Vários protocolos de gerenciamento da mobilidade foram propostos [5 - 8, 9, 10, 11]. Apesar deles terem o objetivo comum da transparência de localização, eles diferem consideravelmente uns dos outros devido às escolhas feitas durante as fases de projeto e implementação.

A fim de chegar a uma solução de mobilidade compreensível e eficiente para Internet sem fio, acreditamos que é necessária uma perspectiva que leve em consideração as camadas nas quais operam essas soluções. Os protocolos de mobilidade existentes foram projetados principalmente para as camadas de rede, de transporte, e de aplicação, e a maioria dos estudos refere-se ao suporte à mobilidade inerentemente fornecido pela rede sem fio através da mobilidade de camada de sub-rede. [4]

Apenas os protocolos das camadas de transporte e de aplicação mantém a semântica fim-a-fim de uma conexão entre *hosts* que estão se comunicando. Uma classificação

camada-a-camada dos protocolos de comunicação existentes é mostrada na figura 1, que também mostra o escopo ou área de atuação destes protocolos. [4]

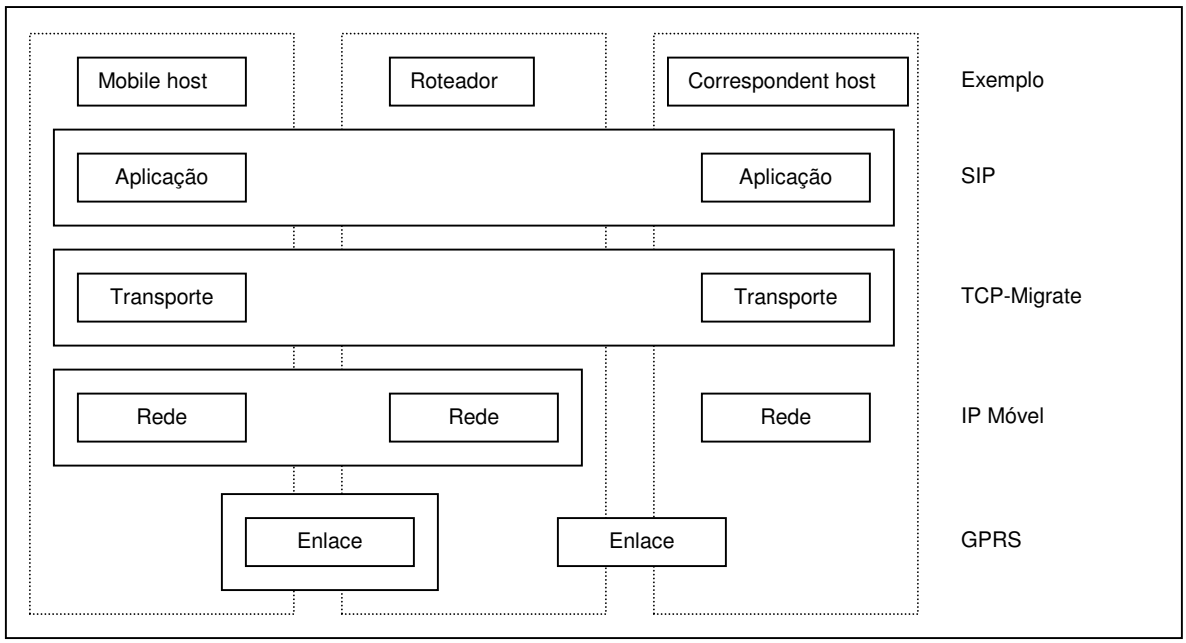

Figura 1 ***Uma classificação camada-a-camada dos protocolos de mobilidade***

A maioria dos protocolos de mobilidade está limitada a uma única camada, por isso eles são transparentes para as outras camadas. A mobilidade da camada de enlace é transparente para as camadas de rede e superiores porque um MH muda seu ponto de acesso usando apenas mecanismos da camada 2.

Isto faz com que o desenvolvimento destes mecanismos seja fácil das perspectivas da rede e dos *hosts* fim-a-fim, uma vez que todos os componentes afetados estão confinados no *mobile host* e em seu enlace imediatamente adjacente, sem alterações no roteamento ou nos *correspondent hosts*. Os protocolos da mobilidade de enlace fornecem mobilidade "invisível".

As entidades de protocolo das camadas de rede, de transporte, e de aplicação – não alteradas – continuam a funcionar sem conhecimento dos mecanismos da camada de enlace abaixo delas. A mobilidade da camada de enlace é suportada inerentemente por quaisquer sistemas de mobilidade sem fio, como a mobilidade intra-célula em uma rede celular comutada por circuitos.

O movimento intercelular em redes celulares é conseguido com sofisticados mecanismos de chaveamento nos centros de chaveamento móvel que controlam as estações base. O objetivo principal deste tipo de protocolo de mobilidade é fornecer mobilidade limitada apenas para os enlaces, que podem trabalhar em conjunto com os protocolos das camadas superiores para fornecer mobilidade fora do escopo do enlace.

A mobilidade da camada de rede encontra seu valor exatamente onde a mobilidade da camada de enlace falha com respeito ao escopo da operação. Assim, quando um protocolo de mobilidade da camada de rede está operando em uma rede baseada na arquitetura TCP (*Transmission Control Protocol*) / IP, as camadas superiores não têm que se incomodar com mudanças de endereço IP devidas à mobilidade do *host*.

Embora a noção de mobilidade da camada de rede tenha surgido com a proposta do IP Móvel, outros protocolos de mobilidade da camada de rede diferentes foram propostos, cada um tentando melhorar o desempenho no que diz respeito a certos parâmetros. Estes protocolos podem ser classificados em duas classes, micro-mobilidade e macro-mobilidade, baseado no seu escopo de operação no que diz respeito aos domínios administrativos na Internet.

Os protocolos de micro-mobilidade trabalham dentro de um domínio, enquanto os protocolos de macro-mobilidade operam entre domínios. As aplicações legadas ainda podem ser utilizadas sem alterações com um protocolo de mobilidade nesta camada.

Mas este protocolo requer modificações em nível de *kernel* no sistema operacional dos MH's para implementar as funções específicas de mobilidade da camada de rede. Do ponto de vista arquitetural, a camada de rede é a camada certa para fornecer mobilidade a todos os transportes e todas as aplicações.

Por várias razões, como filtragem da entrada de pacotes ou não desenvolvimento dos agentes de mobilidade, os mecanismos da camada de rede não foram amplamente desenvolvidos até agora. Quanto à camada de transporte, a interface entre a aplicação e a camada de transporte muda com a introdução de um protocolo de mobilidade desta camada.

Com relação à camada de aplicação, os esquemas de gerenciamento da mobilidade através da camada de aplicação superam o inconveniente das mudanças no *kernel* dos MH's. Nosso trabalho concentra-se na camada de rede, por isso, a seguir tratamos de macro-mobilidade e micro-mobilidade.

2.4) Macro-mobilidade

Macro-mobilidade refere-se à mobilidade de usuário que não é freqüente, mas que atravessa um espaço considerável, freqüentemente entre vários domínios administrativos. Podemos considerar o IP Móvel como a mais representativa solução de mobilidade da camada de rede uma vez que ela foi padronizada pelo IETF (*Internet Engineering Task Force*).

Observando a figura 2, um exemplo de macro-mobilidade aconteceria cada vez que o nó móvel se deslocasse da Rede de Acesso 1 para uma das outras duas redes.

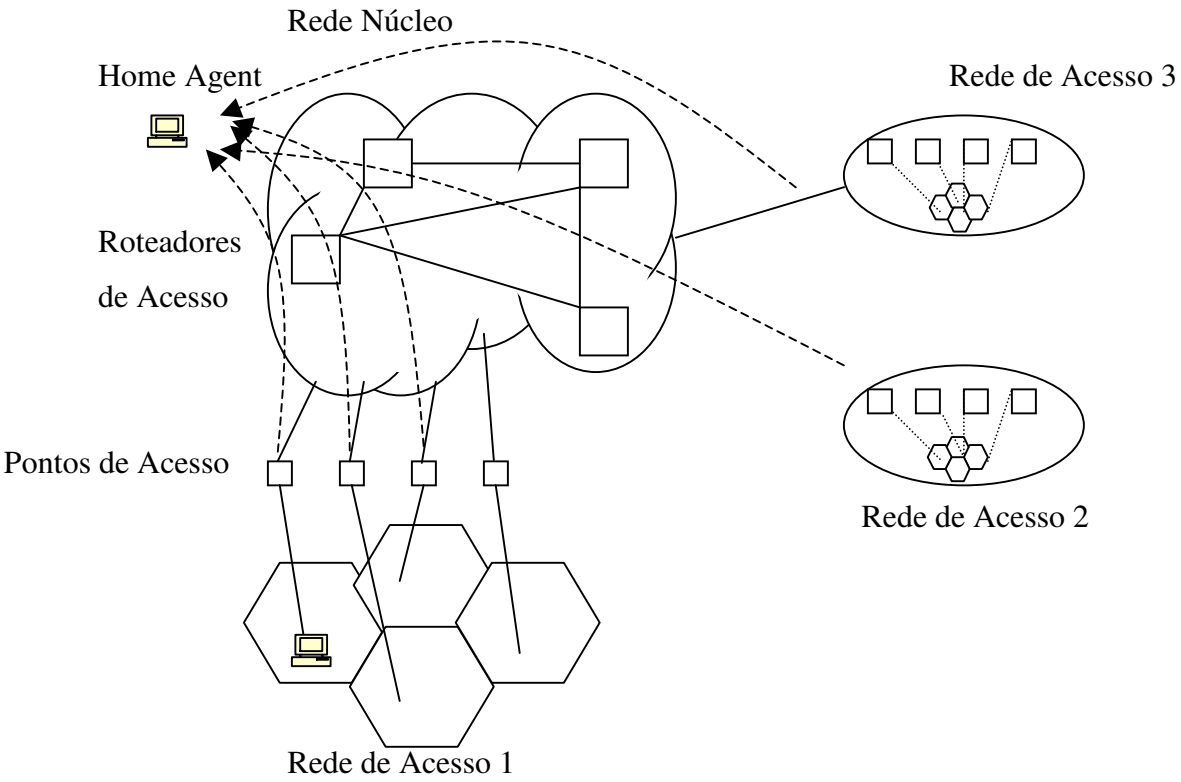

Figura 2 ***Exemplo de Macro-mobilidade***

Vale mencionar que um conjunto de redes de acesso administrado por uma única Autoridade Administrativa é denominado de Domínio Administrativo na Internet Móvel [12]. Observando a figura 3, um exemplo de macro-mobilidade aconteceria cada vez que o nó móvel se deslocasse da Rede de Acesso 1 para a Rede de Acesso 3, isto é, cada vez que o nó se deslocasse para uma rede de outro Domínio Administrativo.

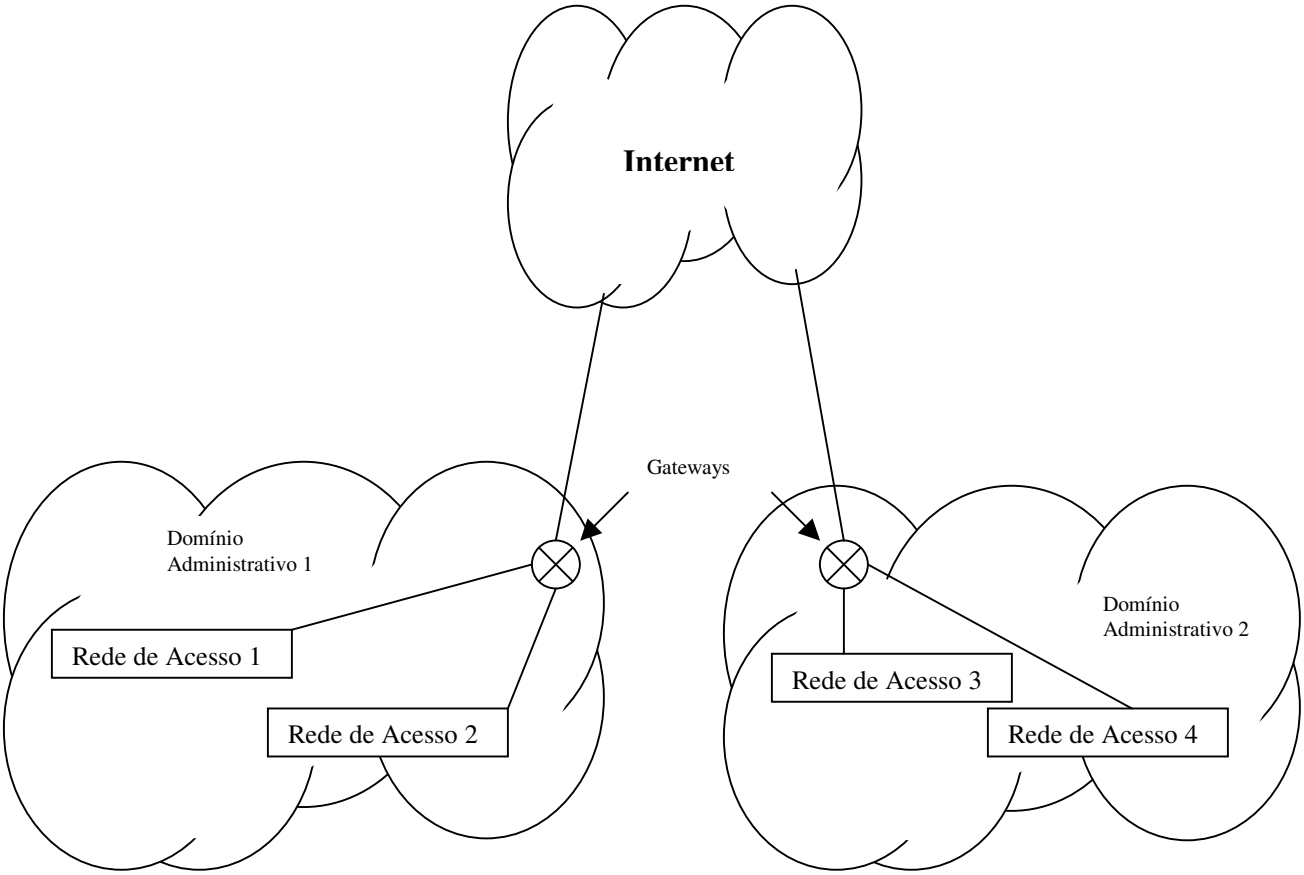

Figura 3 ***Exemplo de Macro-mobilidade – Domínios Administrativos Diferentes***

Como vemos em [13], um dos problemas associados à macro-mobilidade é que, para ambientes de alta mobilidade, o protocolo padrão (IP Móvel) se apresenta ineficiente, devido à freqüente necessidade do dispositivo móvel realizar registro em seu *Home Agent* (HA) toda vez que mudar de *Foreign Network* (FN). Isso implica em longos atrasos no processo de registro e grande carga de sinalização na Internet, principalmente quando um *Mobile Node* (MN) estiver distante do seu HA.

Este problema acontece porque o MIPv4 apresenta algumas falhas no protocolo, tais como o roteamento triangular e a perda de pacotes no filtro de entrada de uma rede. A primeira falha refere-se aos filtros de entrada encontrados nos roteadores de borda das redes visitadas.

Esses filtros possuem regras de filtragem que descartam pacotes destinados a *hosts* externos e cujo endereço de origem do pacote não tenha um prefixo igual ao prefixo da rede

em que se encontra o filtro. Pelo fato do MN utilizar seu endereço *Home Address* como o endereço de origem dos pacotes enviados para um CN, os pacotes do MN serão descartados pelo Filtro.

Para resolver este problema, uma solução, denominada de Tunelamento Reverso foi proposta, cujo objetivo é criar um túnel do MN para o HA, denominado de túnel reverso. Ao chegar no HA os pacotes são desencapsulados e entregues ao CN com o *Home Address* como seu endereço de origem.

A figura 4 ilustra o tunelamento reverso.

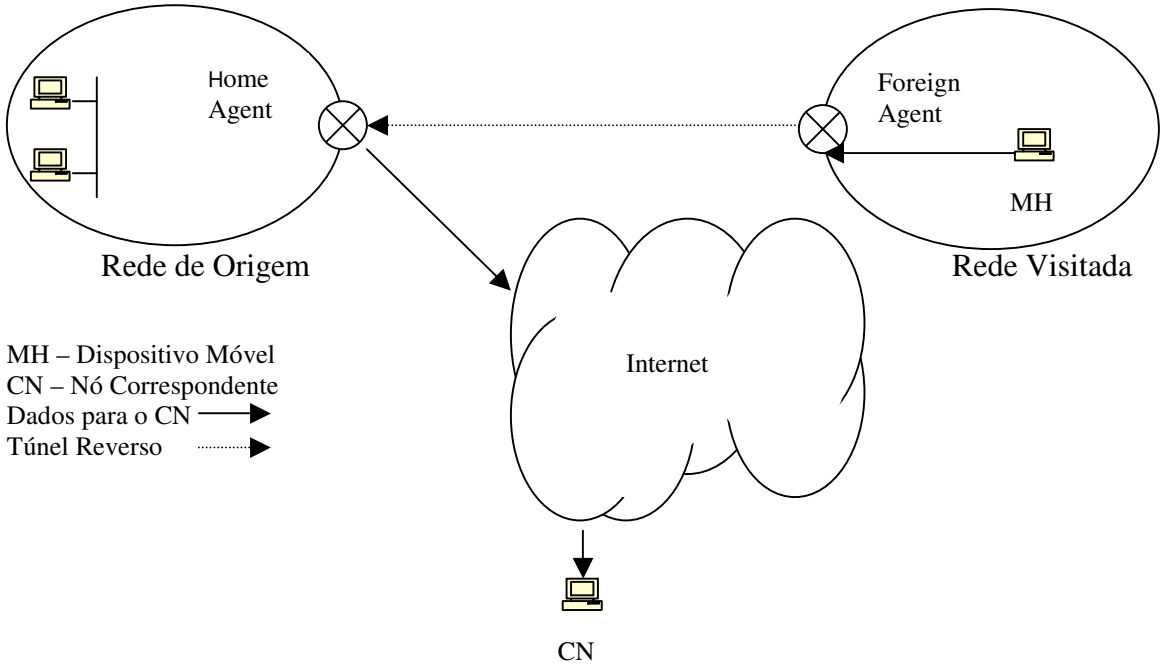

Figura 4 ***Tunelamento Reverso***

O segundo problema ocorre devido ao fato de que no MIPv4 todos os pacotes vindos dos CNs são entregues ao móvel via *Home Agent*. O MN por sua vez envia diretamente os pacotes destinados ao CN, formando assim um roteamento triangular que é ineficiente principalmente quando houver uma comunicação entre um terminal visitando uma rede distante e algum CN pertencente a tal rede.

Uma solução para esse problema foi proposta e denominada Otimização de rota. Através dela é possível transmitir pacotes oriundos dos CNs diretamente ao MN, sem passar pelo HA.

Isso é possível através de mensagens de *Binding Updates* (BUs) enviadas pelo *Home Agent* aos CNs informando a nova posição (*care-of-address*) do MN.

2.5) Macro-mobilidade com IP Móvel

O IP Móvel [1] foi proposto com o objetivo de suportar usuários móveis com transparência para a aplicação e a possibilidade de *roaming* sem interrupção da comunicação. Isto foi feito adicionando tunelamento IP para uma infra-estrutura de roteamento IP unificada.

O padrão IP Móvel tem por objetivo tratar o problema que ocorre quando um *host* muda seu endereço IP durante a comunicação. Esta mudança ocorre porque o protocolo IP [14] assume que cada identificador de rede IP é relacionado a uma rede física específica.

Em redes celulares, os *hosts* móveis são conectados a redes físicas através de *links* que conectam o *host* móvel a uma célula que representa a área de cobertura. O acesso à Internet dessa rede física se dá por meio de um roteador. Quando um *host* móvel muda sua posição, ele pode potencialmente conectar-se a outra célula que esteja mais próxima dele do que sua célula de origem. Caso a célula mais próxima não pertença à mesma rede física, o *host* móvel poderá potencialmente ser conectado a uma rede física diferente da sua rede de origem e conseqüentemente precisará mudar seu endereço IP.

Dependendo do tipo de conexão que o *host* móvel estiver executando, TCP ou UDP (*User Datagram Protocol*), problemas ocorrerão quando houver a troca de redes físicas. No caso de conexões TCP, quando o *host* mudar seu endereço IP imediatamente a conexão será cortada, isto porque conexões TCP são implementadas utilizando-se quatro referências, os endereços IP de origem e destino e das portas TCP utilizadas. Em conexões UDP, o caso é menos grave, porém um ou mais datagramas podem ser perdidos na replicação dos mesmos para um *host* móvel que já tenha mudado seu endereço IP. Para resolver estes problemas, o padrão IP Móvel utiliza uma técnica de tunelamento.

Segundo o padrão IP Móvel cada *host* móvel possui dois endereços IP. Um é relacionado à sua *home network* (onde o *host* é registrado) e não muda quando o *host* mudar sua posição. O segundo IP é relacionado a *foreign network* e muda cada vez que o *host*

conecta-se a uma rede física diferente. Este segundo endereço IP é denominado de CoA (*Care-of-Address*). A figura 5 ilustra, de forma simplificada, uma rede IP Móvel.

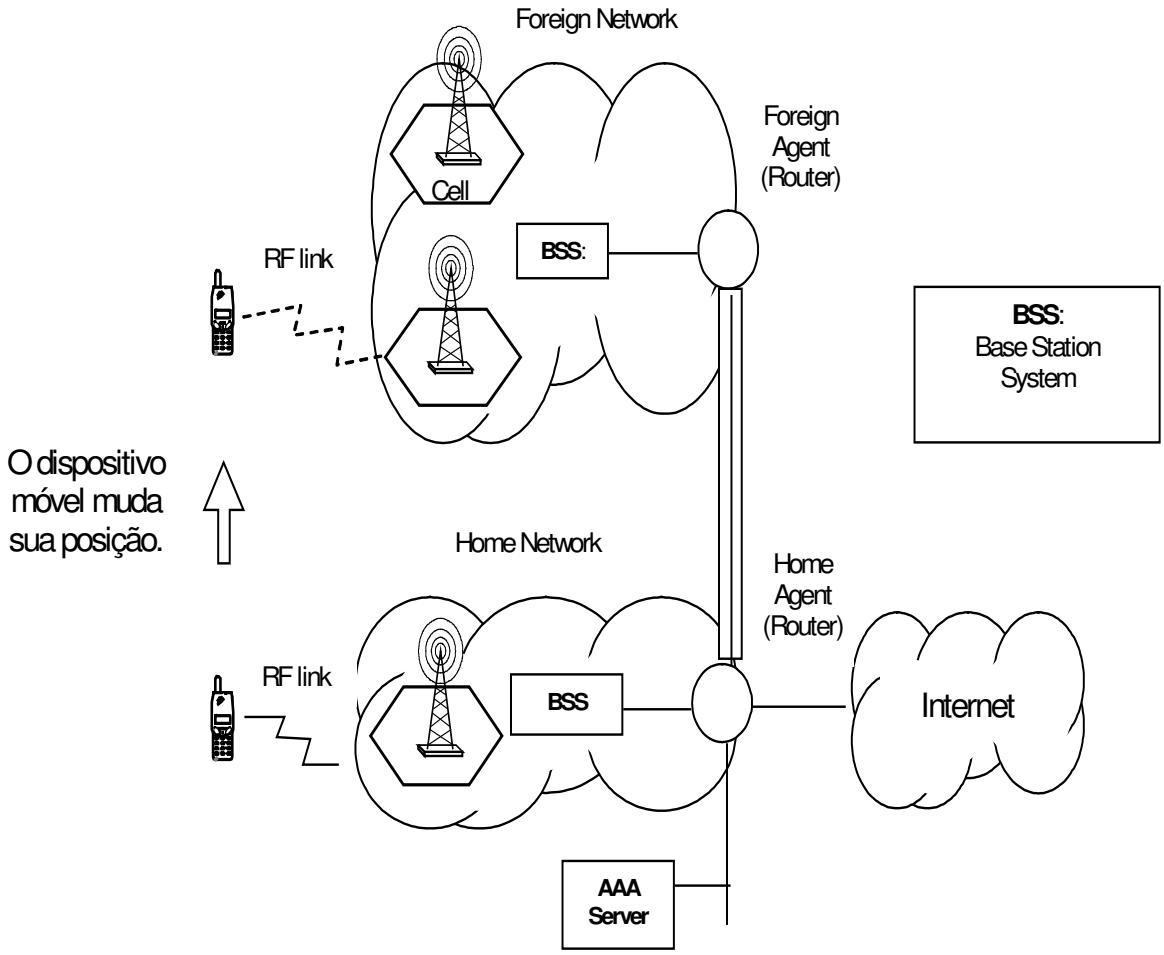

Figura 5 *Representação Simplificada de uma rede IP Móvel*

O roteador conectado ao *host* móvel na *foreign network* é denominado de *foreign agent*. O roteador da *home network* é denominado de *home agent*. O *home agent* é um roteador especial, responsável por autenticar o *host* móvel e manter uma tabela interna de mapeamento de endereços CoA para endereços IP da *home network,* ou seja, para cada *host* móvel existe este mapeamento. O padrão IP Móvel especifica que é de responsabilidade do *host* móvel informar ao *home agent* que seu CoA mudou. O *host* móvel envia esta informação através de uma mensagem *binding update* para o *home agent* toda vez que esta mudança ocorre, sendo que esta mensagem é enviada ao *home agent* através do *foreign agent*.

Para *hosts* não móveis da Internet, um *host* móvel é identificado pelo seu endereço IP da *home network,* isto é, pacotes vindos da Internet são enviados para o *host* móvel através de um túnel que o acompanha enquanto ele muda sua posição e se conecta a diferentes redes. Dois tipos de implementação de tunelamento podem ser empregados: o túnel pode ser criado entre o *home agent* e o *foreign agent*, ou entre o *home agent* e o *host* móvel. O túnel é criado pelo encapsulamento de pacotes vindos da Internet, endereçados ao *host* móvel através de seu *home address,* com um cabeçalho IP que endereça o *host* móvel através de seu *Care-Of-Address* (CoA). Se o túnel é criado somente até o *foreign agent*, então todos os *hosts* móveis servidos pelo mesmo *foreign agent* podem compartilhar o mesmo CoA. Se o túnel é criado até o *host* móvel, então todo *host* móvel precisa ter seu próprio CoA. A figura 6 mostra o tunelamento do IP Móvel.

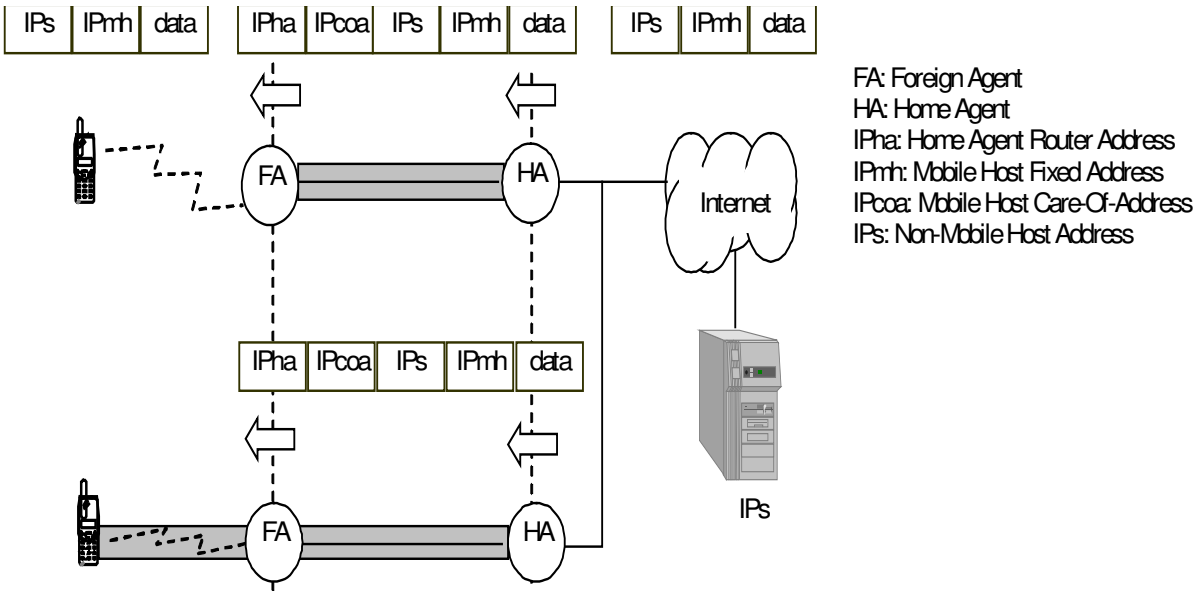

Figura 6 *Tunelamento IP Móvel*

Alguns dos aspectos mais importantes do IP Móvel estão relacionados à autenticação. A figura 7 apresenta um diagrama de seqüência ilustrando este processo de autenticação.

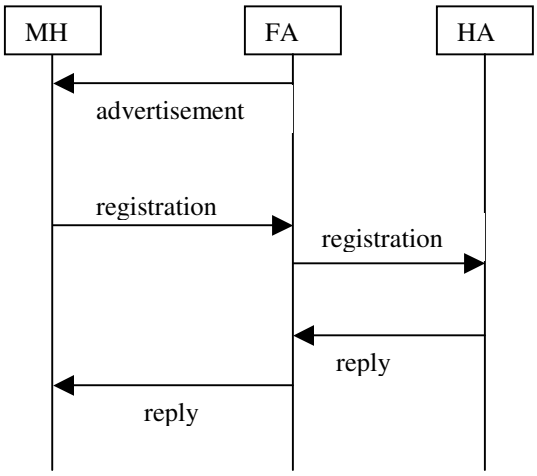

Figura 7 ***Autenticação IP Móvel***

O IP Móvel apresenta dificuldades como a necessidade de registro e de autenticação de um dispositivo móvel com o seu *Home Agent* cada vez que se movimenta para uma nova rede. Para resolver esse problema, algumas propostas surgiram para gerenciar o deslocamento dentro de uma área limitada, reduzindo o número de acessos ao *Home Agent*.

Como autenticação entende-se que, antes de encaminhar dados, ambas as partes devem ser capazes de autenticar a identidade da outra. Em geral, um servidor deve confirmar a identidade pessoal de um usuário antes de fornecer um serviço.

O processo de *login* do usuário pode determinar se este é um usuário legal do sistema. Se a pessoa conectando-se é verificada como um usuário legal, o sistema oferece o serviço.

O protocolo de autenticação do usuário de IP Móvel é diferente do protocolo de autenticação dos serviços comuns. Três níveis de autenticação são necessários no IP Móvel: o nó móvel deve autenticar-se com o *foreign agent,* o *foreign agent* com o *home agent* e o nó móvel com o *home agent*.

Os usuários devem ser verificados em todos os três níveis de autenticação a fim de receber o serviço. O processo da autenticação de identidade do nó móvel previne que usuários ilegais usem o ataque de repetição para adquirir serviços do sistema.

O IP Móvel impôs forte autenticação sobre todas as mensagens de registro que são trocadas durante o processo para a prevenção do ataque de DoS. Para este propósito cada nó móvel e cada *home agent* devem compartilhar uma associação de segurança.

2.5.1) Aplicações IP Móvel

Um aparelho móvel pode trocar dados com outros *hosts* através de duas formas: utilizando a transmissão de dados por pacotes ou por circuitos. O modo de transmissão de dados utilizado pelo padrão IP Móvel é por pacotes.

Para aparelhos móveis que operam neste tipo de transmissão, o padrão IP Móvel disponibiliza três tipos aplicações: as aplicações WAP (*Wireless Application Protocol*) [15], as que utilizam o aparelho móvel como um modem (*tethered*) e as aplicações embarcadas não-WAP (*non-WAP embedded*). A figura 8 apresenta estes três tipos.

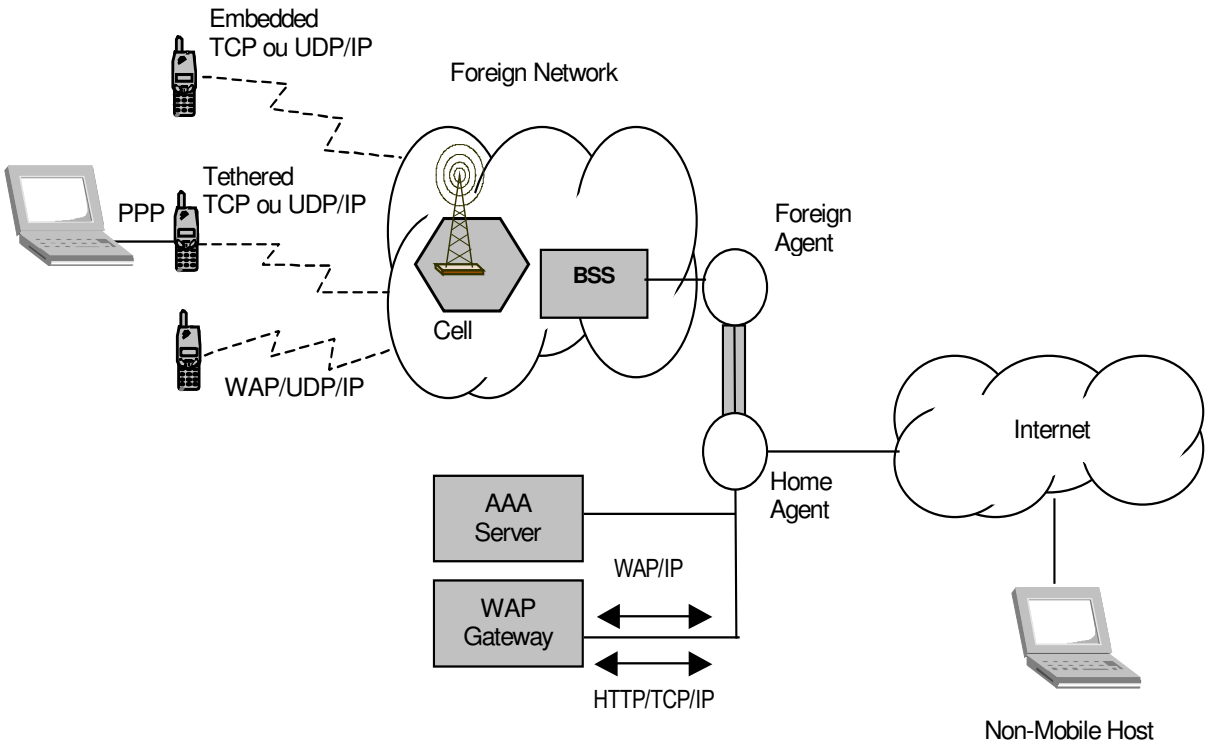

Figura 8 ***Aplicações IP Móvel.***

As aplicações WAP representam o meio mais comum do uso da capacidade de transmissão de dados por pacotes utilizando telefones celulares. Basicamente, WAP é uma versão *wireless* do modelo cliente/servidor da *World Wide Web (WWW).* Para acessar serviços WAP, um celular precisa ter um *microbrowser*, que é um *software* cliente responsável por enviar e receber pacotes IP com conteúdo WAP. Um elemento fundamental na comunicação WAP é o *WAP gateway* que é o elemento responsável por converter a pilha HTTP/TCP/IP na pilha WAP/IP para garantir a interoperabilidade entre clientes WAP e servidores http (*Hyper Text Transmission Protocol*).

As aplicações *tethered* são aquelas que utilizam o celular como se fosse um modem conectado a um computador. Neste caso, o celular é visto pelo sistema operacional do computador como um equipamento que se comunica com o computador utilizando PPP (*Point-to-Point Protocol*). O computador móvel pode comunicar-se diretamente com outros computadores utilizando a pilha de protocolos padrão TCP/UDP/IP.

As aplicações embarcadas não-WAP são empregadas em duas situações principais: para *handsets* que suportam um sistema operacional aberto para programação, tal como o Windows-CE [16] e, para telefones celulares com capacidades de programação específicas, tais como *Java 2 Platform, Micro Edition (J2ME)* [17]. Do ponto de vista de rede não existe diferença entre aplicações *tethered* e embarcadas, isto porque em ambos os casos, os pacotes IP são trocados com a Internet sem intermediação, utilizando-se a pilha de protocolos TCP/UDP/IP.

As aplicações IP Móvel funcionam muito bem, exceto quando a distância entre a *foreign network* visitada e a *home network* do MH é suficientemente grande, a ponto de introduzir um grande retardo de sinalização para os registros dos novos CoAs. Depois de um MH entrar em uma nova rede, nenhum tráfego do CH (***Correspondent Host***) alcançará o MH até o MH atualizar seu HA.

2.6) Micro-mobilidade

Devido às limitações do IP Móvel, alguns tipos de protocolos de mobilidade localizada são necessários.

Estes protocolos operam em um domínio administrativo restrito e atendem os MHs dentro daquele domínio com conexões à rede central, enquanto mantém o custo de sinalização, a perda de pacotes, e a latência do *handover* tão baixos quanto possível. Vários protocolos de micro-mobilidade foram propostos, incluindo HAWAII [5], *Cellular* IP (CIP) [6], *Terminal Independent Mobile IP* (TIMIP) [7], e *Intra Domain Mobility Management Protocol* (IDMP) [8].

Um *framework* de avaliação proposto em [18] pode ser usado para comparar qualitativamente o desempenho de vários protocolos de micro-mobilidade em termos de esquemas de roteamento, gerenciamento de *handover*, suporte a *hosts* móveis inativos, gerenciamento de endereços, e topologia de roteamento. Para esquemas de roteamento, HAWAII, CIP, e TIMIP substituem o protocolo de roteamento IP tradicional e implementam seus próprios esquemas de roteamento específicos, os quais acarretam algum *overhead* adicional para manter a informação do caminho corrente para cada *host*.

IDMP (*Intra Domain Mobility Management Protocol*), por outro lado, usa o roteamento IP normal para encaminhar pacotes. Uma vez que o IDMP não precisa atualizar a informação de caminho antes do *handoff*, a latência associada é menor que a dos outros protocolos.

Além disso, o IDMP é capaz de minimizar a perda de pacotes durante o *fast handoff*. No CIP, quando um MH executa um *fast handoff*, a perda de pacotes não é insignificante.

Todos os protocolos citados, exceto IDMP, fornecem algum tipo de suporte para *hosts* inativos na forma de um esquema de paginação. Quanto ao gerenciamento de endereços, somente o CIP pode usar seu endereço da rede de origem como seu identificador na rede visitada; todos os outros protocolos têm que obter um novo CoA no domínio visitado.

No caso do IDMP, o MH adquire dois CoAs. HAWAII, CIP, e TIMIP precisam organizar a rede de acesso em uma topologia em árvore com um nó *gateway,* no topo da árvore, comunicando-se com a Internet.

Para o IDMP, não há este requisito e um MH pode comunicar-se com a Internet através de múltiplos nós *gateways*. Considerando todos estes aspectos, o IDMP é uma solução mais completa que os outros protocolos de micro-mobilidade citados.

2.7) Considerações finais

O objetivo desse capítulo foi contextualizar nosso trabalho no ambiente de redes móveis e ressaltar que o IP Móvel é o protocolo padrão de macro-mobilidade. Com o avanço da tecnologia na área de computação móvel, várias redes móveis já estão utilizando o protocolo IP em suas implementações, outras estão próximas de necessitar de seu uso.

Macro-mobilidade refere-se à mobilidade de usuário que não é freqüente mas que atravessa um espaço considerável, freqüentemente entre vários domínios administrativos. O IP Móvel foi proposto com o objetivo de suportar usuários móveis com transparência para a aplicação e a possibilidade de *roaming* sem interrupção da comunicação.

Este trabalho considera o cenário em que existem redes que usam mais de um HA, isto é, uma integração de redes gerenciadas por entidades independentes. Um exemplo disto seria a integração entre uma rede WLAN e uma rede celular, sendo que hoje em dia isto é perfeitamente possível, uma vez que recentemente começaram os anúncios de lançamento de celulares capazes de se conectar a qualquer rede sem fio sob a qual estiver inserido, como os padrões WCDMA, GSM, GPRS, *Bluetooth* e WLAN.

Como no cenário considerado as redes usam mais de um HA, se o nó móvel mudar de HA ele mudará de IP, uma vez que a integração se dá entre redes gerenciadas por entidades independentes. Nesse caso – e esta é a nossa proposta - a alternativa mais lógica para manter a rastreabilidade do nó móvel é usar um mecanismo de DNS dinâmico.

Capítulo 3 – Trabalhos Relacionados à Macro-mobilidade

3.1) Introdução

O objetivo desse capítulo é apresentar trabalhos relacionados à macro-mobilidade. Ele está estruturado da seguinte forma: a seção 3.2 apresenta a definição do protocolo DHMIP; a seção 3.3 define *Context Transfer*; a seção 3.4 apresenta um estudo de caso de *handover* transparente em redes terrestres com acesso por rádio; a seção 3.5 define uma arquitetura de inter-redes WLAN / CDMA2000; e a seção 3.6 apresenta as considerações finais do capítulo.

3.2) DHMIP

Um dos maiores desafios para o projeto de redes sem fio é o gerenciamento eficiente da mobilidade, que pode ser tratado global e localmente. Em [19], a contribuição mais importante é a nova abordagem analítica na avaliação do desempenho das redes IP móvel.

A atual demanda - rapidamente crescente - por acesso sem fio às aplicações Internet é alimentada pelo sucesso reconhecido das redes de comunicação sem fio e pelo crescimento explosivo da Internet. O protocolo IP foi projetado para redes cabeadas.

O MIP (`Mobile IP`) permite aos terminais móveis manter as comunicações com a Internet em andamento enquanto se movem de uma sub-rede a outra. Entretanto, o MIP não é uma boa solução para usuários com alta mobilidade.

Em [19], os autores propõem um esquema de gerenciamento da mobilidade hierárquico e dinâmico para redes MIP. Além disso, enquanto a maioria das avaliações de desempenho dos esquemas de gerenciamento da mobilidade em redes MIP é executada através de simulações, o trabalho citado apresenta uma nova abordagem analítica para a avaliação de desempenho das redes MIP.

Como vimos anteriormente, a mobilidade de usuário em redes sem fio que suportam mobilidade IP pode ser classificada em macro-mobilidade e micro-mobilidade, sendo que para o escopo desta dissertação estamos mais interessados na macro-mobilidade.

Em [20], os autores propuseram um esquema de gerenciamento da mobilidade escalável que utiliza FAs hierárquicos para administrar a mobilidade de usuário dentro de uma sub-rede para a Internet sem fio, e a hierarquia de FAs neste esquema é pré-configurada.

Em [19], os autores propõem outro esquema de gerenciamento da mobilidade - hierárquico e dinâmico - para as redes MIP. Uma vez que este esquema reduz significativamente o tráfego de registro com os HAs, ele também pode reduzir grandemente a perda de pacotes.

Por este procedimento, uma hierarquia de localização dinâmica é construída para um usuário móvel específico. O esquema DHMIP pode ser descrito pelo pseudocódigo na tabela 1.

Tabela 1 ***DHMIP Protocol***

```
% Procedimentos de registro da localização
K_opt: o ponto inicial ótimo do nível da hierarquia;
Inicializa i = 0;
IF (MH entra em uma nova sub-rede)
        i = i + 1;
        IF (i ≤ K_opt)
                FA novo registra-se com o FA anterior;
        ELSE
                FA novo registra-se com o HA;
                i = 0;
                Calcula o novo K_opt;
        ENDIF
ENDIF
% Procedimentos da entrega de pacotes
IF (pacotes para o MH foram interceptados pelo HA)
        Tunela os pacotes para o primeiro FA;
        IF (O primeiro FA não é o que está servindo atualmente o MH)
                Retunela os pacotes para o FA corrente;
        ENDIF
        O FA corrente desencapsula os pacotes e envia-os para o MH;
ENDIF
```

Os autores desenvolvem um modelo analítico para derivar o custo das funções de atualização da localização e de encaminhamento dos pacotes para o esquema DHMIP.

Eles definem os seguintes parâmetros para sua análise: número de sub-redes atravessadas entre a chegada do último pacote e a última atualização da localização antes da chegada do último pacote; taxa de mobilidade do MH; ponto inicial do nível hierárquico de FA (*Foreign Agent*); custo médio da atualização de localização do MH com seu HA; custo de encaminhamento de pacote, para um pacote até um MH, em uma *foreign network* sob o esquema MIP; custo total da atualização de localização para um MH decorrido entre duas chegadas de pacotes consecutivas sob o esquema DHMIP; custo médio de encaminhamento de pacote, para um pacote até um MH, em uma *foreign network* sob o esquema DHMIP; custo de encaminhamento de pacote entre FAs no esquema DHMIP; custo de configuração da hierarquia no esquema DHMIP; e a distribuição probabilística do primeiro parâmetro. Com esses parâmetros, os autores desenvolvem o modelo analítico.

Para avaliar o desempenho do esquema DHMIP, é preciso conhecer o custo relativo de atualizar o HA e o FA anterior. O esquema MIP pode resultar em uma pesada carga de tráfego no sistema por causa das mensagens de atualização da localização dos MHs.

Com os resultados numéricos do modelo analítico, observamos que quando a taxa de mobilidade do MH é pequena, o esquema DHMIP gera menos tráfego que o esquema MIP mesmo sem utilizar os valores ótimos. Alguns usuários móveis costumam transitar apenas em uma certa “redondeza” quando acessam as aplicações Internet.

Antes de discutir a escolha do ponto inicial do nível hierárquico ótimo para um usuário móvel, os autores estudam o impacto do número de níveis da hierarquia sobre o desempenho do esquema DHMIP. Eles mostram o desempenho relativo do esquema DHMIP utilizando os valores ótimos sob diferentes taxas de mobilidade do MH e diferentes distribuições do número de sub-redes atravessadas entre a chegada do último pacote e a última atualização da localização antes da chegada do último pacote.

Após isto, os autores investigam a sensibilidade dos custos de desempenho e dos benefícios do esquema DHMIP de acordo com a variação no padrão da mobilidade do usuário móvel. Como resultado, observa-se que o DHMIP ainda pode conseguir desempenho similar sob diferentes taxas de mobilidade do MH, diferentes tempos de

permanência na *home network* e distribuições do número de sub-redes atravessadas entre a chegada do último pacote e a última atualização da localização antes da chegada do último pacote.

O protocolo HMIP (*Hierarchical MIP*) do IETF objetiva reduzir o tráfego de sinalização para as *home networks* enquanto minimiza o retardo de sinalização. Os autores comparam seu esquema DHMIP com o esquema HMIP do IETF.

O desempenho do esquema DHMIP pode ser melhorado futuramente, segundo seus idealizadores, pela adição de mecanismos que simplifiquem o processo de sinalização, através da paginação de nós móveis inativos (*IP paging)* e remoção da sinalização de usuários que revisitam frequentemente a mesma região de mobilidade (*loop removal).*

O *Cellular IP* já suporta IP *paging* [21]. Através do modelo analítico, os autores derivam o desempenho que o esquema DHMIP acrescido do IP *paging* teria.

Segundo os autores, no esquema DHMIP acrescido do IP *paging*, o custo total de sinalização do sistema nunca excederá o do esquema MIP, sob todas as condições. O desempenho do esquema DHMIP também pode ser melhorado removendo o *loop* formado durante o movimento do usuário móvel.

Novamente através do modelo analítico, podemos ver que o custo total de sinalização para o esquema DHMIP pode ser reduzido pelo mecanismo de *loop removal*. Concluindo, o trabalho analisado forneceu um novo esquema de gerenciamento da localização para redes MIP, a estratégia DHMIP.

Os autores também demonstraram que o DHMIP pode futuramente ser melhorado considerando *state activation (IP paging)* e *loop removal*.

3.3) Context Transfer

Fazer *handover* transparente é um dos assuntos chave em gerenciamento de mobilidade para as redes baseadas em IP da próxima geração. "*Context Transfer*" objetiva contribuir para o aprimoramento no desempenho do *handover*.

As informações de estado dos serviços candidatos à transferência de contexto podem ser transferidas, por exemplo, do roteador de acesso anterior ao novo roteador de

acesso de forma que os serviços possam ser restabelecidos rapidamente, como vemos na figura 9.

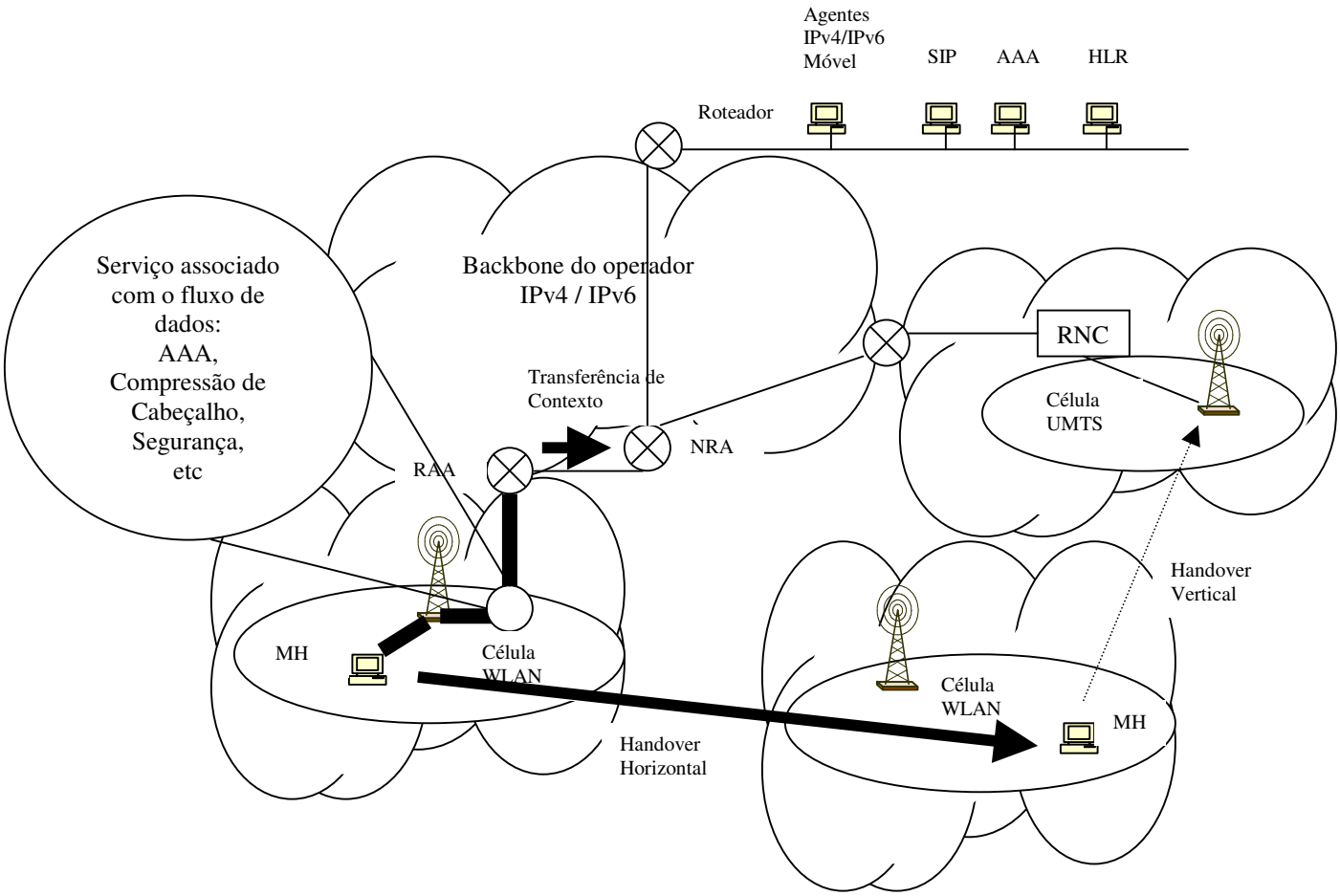

Figura 9 ***Transferência de Contexto em infraestruturas baseadas em IP***

O trabalho apresentado em [22] atua como um complemento do trabalho apresentado em [23]. Exemplos possíveis de serviços candidatos à transferência de contexto são AAA (`Authentication, Authorization and Accouting`) e Compressão de Cabeçalho.

Outros possíveis serviços candidatos à transferência de contexto podem incluir "*Multicast group membership*", onde o AR (`Access Router`) deve conhecer a quais grupos *multicast* o MH já faz parte; e "*IPsec state*", onde o AR pode atuar como um *gateway IPsec*, em cujo caso uma associação de segurança entre o MH e o AR permite que os

pacotes sejam criptografados e decriptografados entre os dois. As pesquisas anteriores relacionadas a permitir mobilidade transparente sobre infraestruturas IP eram focadas principalmente na melhora do procedimento de *handover* entre roteadores de acesso ou estações base.

Para a pesquisa em [22], os autores objetivam projetar e desenvolver um esquema de transferência de contexto para uma infra-estrutura de gerenciamento da mobilidade multicamada, um ambiente que diferencie entre os diferentes tipos de mobilidade. Mais especificamente, os seguintes protocolos de mobilidade serão considerados para extensão com capacidade de transferência de contexto: *Mobile IP* [9]*, Hierarchical Mobile IP* [24] e *Cellular-IP* [25].

Esses protocolos de mobilidade serão estendidos para oferecer funcionalidades extras possibilitando encaminhar as informações de estado desejadas para o novo roteador de acesso. Quanto ao *Mobile-IP,* quando um MN move-se de uma rede a outra, após estabelecer uma conexão na camada de enlace em sua nova rede, ele envia um pacote *Binding Update* para um *foreign agent* e para seu *Home Agent*.

Uma possível abordagem deve considerar que algum tipo de sinalização explícita é necessário para o novo FA requisitar características de contexto do FA anterior do MN. O procedimento para transferência de contexto descrito no trabalho analisado pode ser suficiente porque se o *Mobile IP* é aplicado para tratar da mobilidade de MN entre redes, que usualmente não é tão freqüente quanto a mobilidade entre roteadores de acesso ou entre estações base, o desempenho de *handoff* pode ser "desprezado".

Mas executar a transferência de contexto antes da operação de *handoff* certamente acarretará um melhor desempenho de *handoff*, e se isso for conseguido em um tempo suficientemente curto, os serviços usados pelo MN em sua rede anterior podem continuar sem nenhuma interrupção. Com relação ao *Hierarchical Mobile-IP,* ele introduz o *Mobility Anchor Point* como uma entidade local para auxiliar nos *Mobile IP handoffs*.

Conforme a proposta apresentada pelos autores, quando o nó móvel muda o ponto de acesso dentro de um domínio MAP (`Mobility Anchor Point`) local ele apenas registra seu novo *care-of-address* local com o MAP. O MAP poderia ser usado como uma entidade central para armazenar a informação de contexto e copiá-la para o novo roteador de acesso, usando um pacote de atualização do contexto, na recepção de um pacote BU.

A operação de transferência de contexto – independente de protocolo - descrita acima será futuramente estudada em maiores detalhes. A fim de avaliar, assim como refinar, a operação de transferência de contexto proposta pelos autores, modelagem de rede e simulação estão sendo feitos.

Um dos critérios para escolher a solução de transferência de contexto mais otimizada deve ser a quantidade de sinalização envolvida com a extensão e a carga extra adicionada ao *handoff*.

3.4) Arquitetura Inter-redes WLAN / GPRS

Em [26], os autores abordam o fato de que as redes da próxima geração utilizarão tecnologias de rede heterogêneas. Este artigo fornece uma análise quantitativa de um protótipo de *handover* WLAN – GPRS baseado em IPv4 Móvel, e apresenta o impacto deste *handover* sobre o tráfico de dados UDP e TCP assim como sobre a própria sinalização IP Móvel.

O advento das redes sem fio tem produzido uma alteração dramática na forma como a comunicação de dados é realizada. A comunicação transparente envolve a capacidade do equipamento móvel, sucessiva ou simultaneamente, conectar-se a diferentes pontos de acesso na infra-estrutura de rede, de uma forma que torne o movimento físico imperceptível e mantenha a conectividade inalterada em nível de aplicação.

Muitas tecnologias terrestres – bastante diferentes - sem fio já existem. O *handover* vertical entre GPRS e WLAN aparece como uma área de particular interesse porque combina o alto grau de mobilidade física oferecido pela tecnologia GPRS com a grande largura de banda que a tecnologia WLAN pode oferecer em áreas limitadas.

Quanto ao gerenciamento de *handovers* verticais com IP Móvel, deve-se observar que o *handover* vertical envolve a mudança da interface de acesso. O IP Móvel fornece mobilidade na camada IP, habilitando um nó móvel que se origina em sua *home network* a ser endereçável pelo mesmo endereço IP original, através das diferentes *foreign networks* que este nó estiver visitando.

A conseqüência mais importante de manter o endereçamento do nó móvel no mesmo endereço IP, é o fato de que a mobilidade entre diferentes domínios de

endereçamento IP é transparente, para as aplicações das camadas superiores. O IP Móvel é tipicamente integrado com infra-estruturas de autenticação, autorização e contabilização.

A figura 10 mostra um nó móvel movendo-se entre uma rede de acesso WLAN e uma rede de acesso GPRS enquanto comunica-se com um nó correspondente localizado em sua rede de origem.

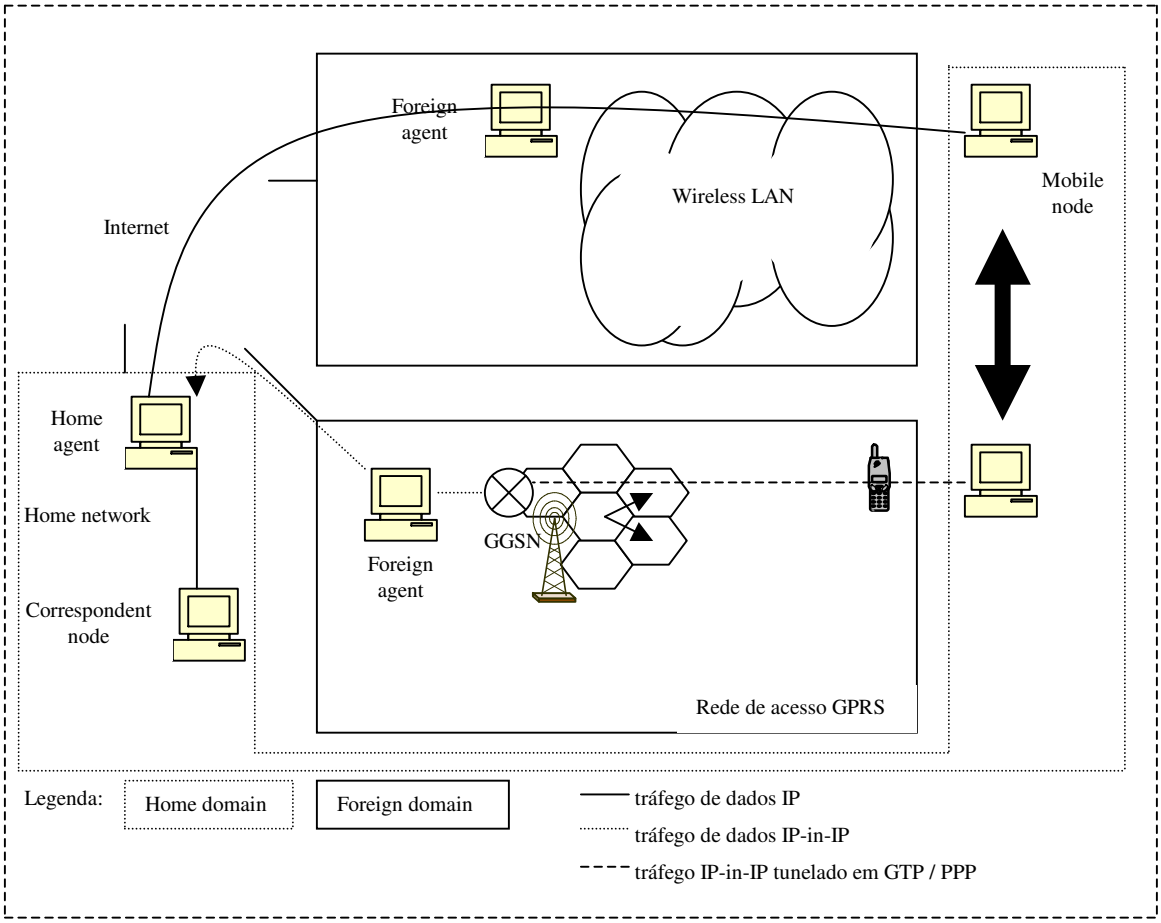

Figura 10 ***IP Móvel em um ambiente WLAN - GPRS***

Os datagramas enviados entre o nó móvel e o nó correspondente são redirecionados através dos segmentos *home agent – foreign agent* e *foreign agent – mobile node* usando tunelamento reverso e encapsulamento *IP-in-IP*. Um *handover* vertical dispara uma mensagem de registro IP Móvel que é enviada a partir da "nova" localização do nó móvel.

O IP Móvel requer mecanismos adicionais a fim de reduzir o retardo e a perda de dados, quando o *handover* ocorre entre redes IP diferentes, mas através do mesmo meio de

acesso. Com relação ao cenário de utilização alvo do trabalho em [2], deve-se observar que o IP Móvel suporta diferentes cenários de endereçamento, que cobrem várias alternativas de desenvolvimento.

Uma das funções primárias de um *foreign agent* é auxiliar os nós móveis a detectar movimento, enviando avisos periódicos, e respondendo às mensagens de solicitação explícitas dos nós móveis. A arquitetura proposta aplica-se especificamente ao cenário em que, tanto a rede origem como as redes visitadas, empregam o uso de esquemas de endereçamento IP privado.

O objetivo é habilitar um nó móvel a acessar entidades localizadas no domínio origem. O GGSN (`Gateway GPRS Suport Node`) é tipicamente assistido por um servidor *Radius* para autorização de usuário e por um servidor DHCP (`Dynamic Host Configuration Protocol`) para alocação de endereço IP.

Os autores decidiram não forçar a atualização da tabela de roteamento durante o experimento, a fim de identificar tanto quanto possível todos os efeitos colaterais que possam ocorrer, em conjunto com o *handover* vertical. Um desses efeitos é o *timeout* de registro, que é o intervalo de tempo que um nó móvel aguarda por uma resposta para a requisição de registro mais recente, antes de enviar uma requisição atualizada.

A fim de garantir a proteção contra ataques de reenvio, cada requisição tem um número seqüencial, que deve conferir com o número seqüencial na resposta. Para analisar o impacto do *handover* vertical no desempenho de aplicações TCP, arquivos de vários *megabytes* foram transferidos entre um nó móvel e um nó correspondente.

Primeiro, o *trace* foi obtido executando o utilitário *tcpdump* no nó correspondente, que atuava como destino. Foi presumido que as perdas de pacotes ocorreram em algum ponto no núcleo da rede GPRS.

Depois, uma transferência TCP foi iniciada a partir do nó correspondente. Um lapso na seqüência de ACKs (`Acknowledgements`) pode ser observada a cada *handover*.

Foi observado que a perda de desempenho das aplicações TCP - quando o nó móvel alterna para um meio de acesso caracterizado por uma latência significativamente maior - é devido tanto à perda de pacotes quanto ao crescimento repentino do RTT (`Round Trip Time`). O *home agent,* idealmente, deveria continuar encaminhando datagramas para

o *care-of address* antigo por um curto período de tempo após disparar o estabelecimento de uma rota para o *care-of address* novo.

O nó correspondente, como transmissor, apenas poderia ser informado da ocorrência de um *handover,* pelo fornecimento de uma opção específica nas mensagens ACK, que ainda assim não evitaria um possível *timeout*. Concluindo, o *handover* entre tecnologias de acesso diferentes, deverá representar um papel importante com relação à comunicação transparente em ambientes de rede heterogêneos.

Os experimentos conduzidos em [26], identificaram as fontes da degradação de desempenho em cenários reais, tanto em tráfego UDP quanto TCP. Finalizando, eliminar os efeitos de *timeouts* requereria que o TCP suportasse alguma forma de notificação de *handover*.

3.5) Arquitetura Inter-redes WLAN / CDMA2000

Em [27], os autores apresentam o argumento de que a combinação das tecnologias sem fio 3G (terceira geração) e WLAN, oferece a possibilidade da realização de acesso à Internet em qualquer lugar e a qualquer hora. O sistema por eles proposto suporta *handoff* transparente, na presença de cobertura de rádio sobreposta.

A Internet tem se apresentado como uma tecnologia amadurecida, que continua a experimentar um crescimento de popularidade tremendo. Os desenvolvedores CDMA [3] estão compelidos a combinar WLAN e CDMA2000 (*Code Division Multiple Access 2000*), para fornecer acesso contínuo sem fio.

A figura 11 ilustra uma visão conceitual das redes - sem fio e públicas – integradas que oferecerão serviços combinando WLAN e CDMA2000.

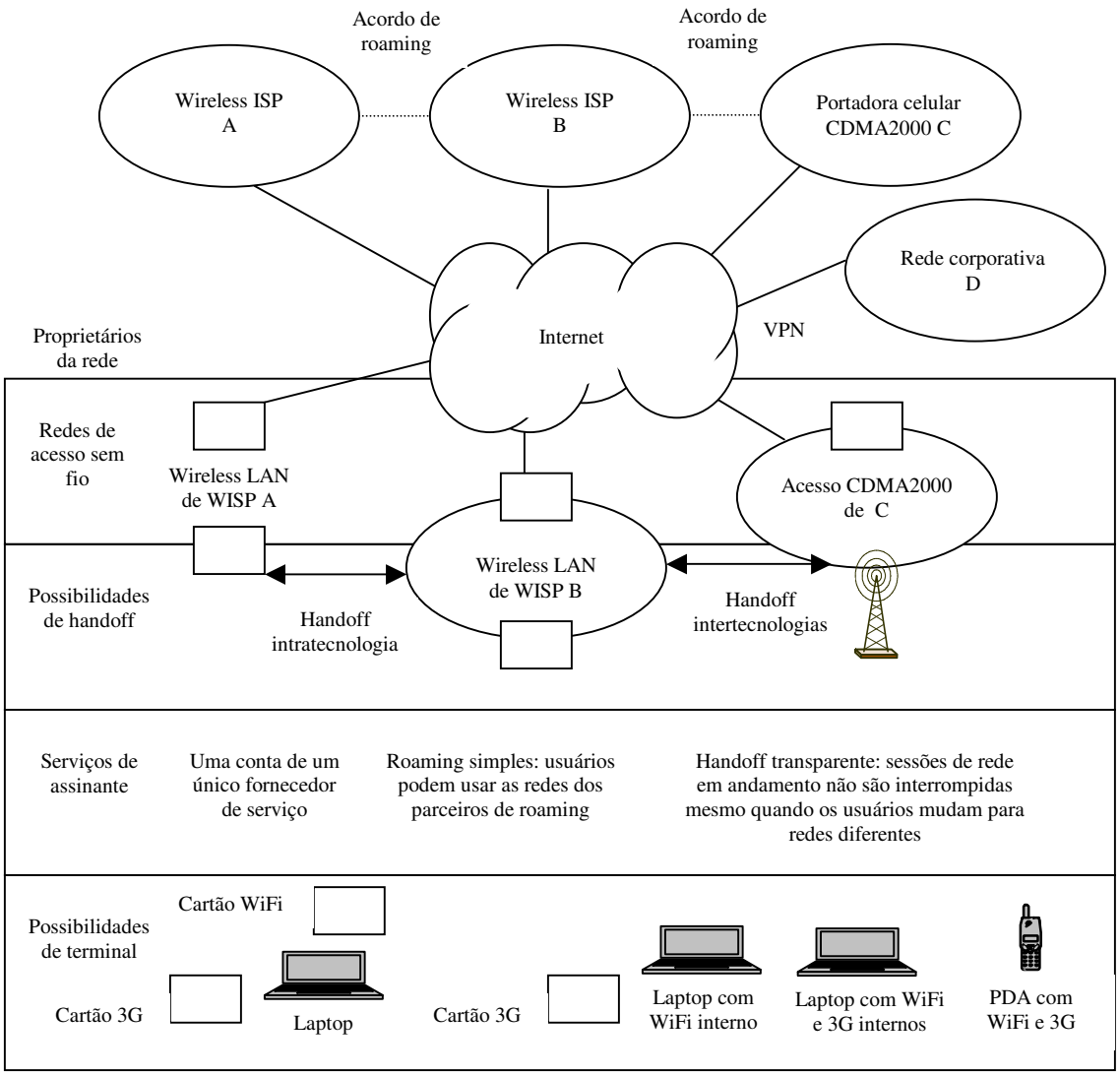

Figura 11 ***Integração CDMA2000 / WLAN***

Dispositivos de usuário final, tais como *laptops* e telefones, que podem acessar redes baseadas em ambas as tecnologias já estão começando a aparecer no mercado. As redes sem fio públicas integradas emergentes oferecerão dois serviços de *roaming*, IP simples, e IP móvel.

O primeiro objetivo do artigo publicado pelos pesquisadores do Bell Labs [27], é descrever opções de projeto, disponíveis para construir uma rede integrada. Após esta descrição, é recomendada a arquitetura de integração baseada em acoplamento fraco, sendo que a implementação desta abordagem e as escolhas de projeto para o *gateway* de integração e para o *software* cliente descritos aplicam-se também à integração de redes WLAN e UMTS (`Universal Mobile Telecommunications System`).

Quanto à rede CDMA2000, um componente chave é o PDSN (*Packet Data Serving Node*), que associa o tráfego de dados proveniente de múltiplos RNCs (*Radio Network Controller*) e relaciona a RAN (`Radio Access Network`) com a rede comutada por pacotes. O PDSN é uma das pontas da conexão PPP, e mantém o estado da sessão para cada nó móvel que esteja em sua área de serviço.

A integração apresentada foca-se na arquitetura básica de uma rede CDMA2000 e de uma rede 802.11. Uma integração entre 3G e WLAN, baseia-se tipicamente em uma arquitetura fortemente acoplada, ou em uma arquitetura fracamente acoplada.

Torna-se claro, pelo exposto no trabalho, que a abordagem baseada em acoplamento fraco oferece várias vantagens arquiteturais sobre a abordagem baseada em acoplamento forte. Por isso, ela tem-se apresentado como uma arquitetura preferencial, para a integração de redes WLAN com redes 3G.

No modelo proposto, um nó móvel não-802.11i pode conectar-se ao *access point* sem qualquer autenticação de camada 2. Porém, ele não pode conectar-se à Internet, a menos que tenha sido autenticado com sucesso pelo *gateway*.

Neste modelo, utiliza-se IP Móvel para efetuar o *handoff* entre as duas tecnologias quando o MN está na rede 3G, assim como quando ele está na WLAN. Neste tipo de integração, existem dois cenários, um com a cobertura de rádio sobreposta e outro com a cobertura de rádio não sobreposta.

O *gateway* WLAN proposto tem integrado também um cachê *Web*, minimizando assim a ineficiência do IP Móvel, em contraposição ao roteamento ineficiente que acontece sem o cachê *Web* integrado ao FA. Ao invés disto, propõem-se roteamento direto através do *gateway* WLAN, integrando assim o cachê *Web* e o FA.

Concluindo, os serviços integrados WLAN / CDMA2000 beneficiarão tanto os provedores de serviço quanto os usuários. Uma implementação típica para a arquitetura

baseada em acoplamento fraco requer um *gateway* de integração WLAN e um *software* cliente para fornecer mobilidade.

Finalizando, os autores acreditam que as tecnologias descritas em [27] podem promover a rápida distribuição de serviços integrados e o crescimento do acesso sem fio a dados, com alta velocidade, em qualquer lugar e a qualquer hora.

3.6) Considerações Finais

A análise de trabalhos relacionados à macro-mobilidade indicam que a evolução recente e o sucesso no desenvolvimento de sistemas WLAN ao redor do mundo têm alimentado a necessidade de mecanismos de integração entre WLANs e redes celulares de dados como GPRS. Vários fóruns e grupos de padronização em todo o mundo iniciaram várias atividades para explorar a tecnologia WLAN e integrá-la com as redes celulares de dados.

Os trabalhos relacionados a integração de redes WLAN e redes celulares podem ser classificados de acordo com duas abordagens genéricas, denominadas: acoplamento forte (*tight coupling)* e acoplamento fraco *(loose coupling)*. A abordagem *tight coupling* é basicamente concebida para WLANs pertencentes às operadoras celulares e não suporta facilmente WLANs de terceiros. Nessa abordagem, o gerenciamento da rede é feito de forma integrada, e existe a possibilidade de oferecer um serviço de mobilidade transparente para os usuários (isto é, sem troca de IP ou perda de conexão). Já a abordagem *loose coupling* é baseado principalmente nos protocolos do IETF. Ela corresponderá ao caso mais genérico, onde as redes WLAN e celulares são administradas de forma independente.

Nosso trabalho concentra-se na abordagem *loose coupling,* uma vez que um dos objetivos é a integração de redes administradas por entidades independentes. No nosso cenário, o usuário precisa refazer sua autenticação e mudará de IP durante o processo de *handover* vertical (mas não durante o *handover* horizontal).

Capítulo 4 – Atualização Dinâmica de DNS

4.1) Introdução

O objetivo desse capítulo é apresentar os conceitos da atualização dinâmica de DNS. Esse capítulo está estruturado da seguinte forma: a seção 4.2 apresenta a definição do *DNS Update*; e a seção 4.3 apresenta as considerações finais do capítulo.

4.2) *DNS Update*

Quando visitamos um *web site*, nosso navegador "solicita" uma busca do endereço IP daquele *site*. Após o DNS retornar o endereço, as informações posteriores trafegam através do protocolo HTTP.

A configuração desta busca da forma mais simples, e talvez a mais típica, é mostrada na figura 12.

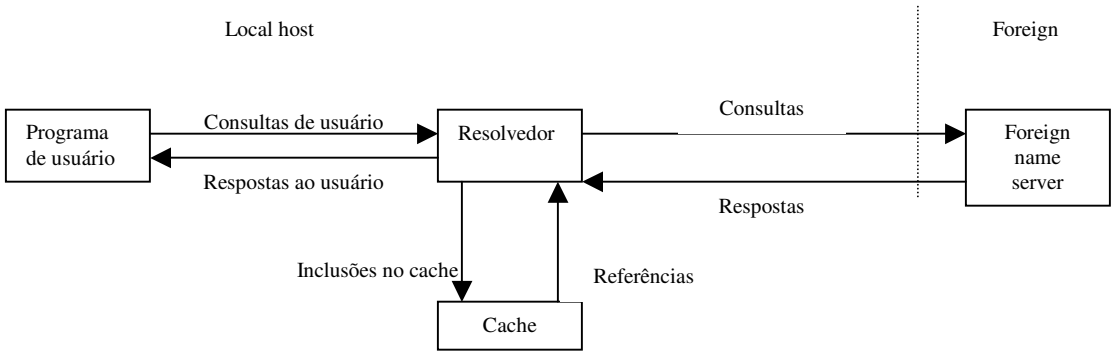

Figura 12 ***Configuração DNS***

Os programas de usuário interagem com o espaço de nomes de domínio através de resolvedores, o formato das consultas de usuário e das respostas ao usuário é específico do *host* e seu sistema operacional, ao contrário das consultas e respostas do resolvedor (que seguem o padrão DNS). O formato - em nível macro - das mensagens DNS é dividido nas 5 seções mostradas na figura 13.

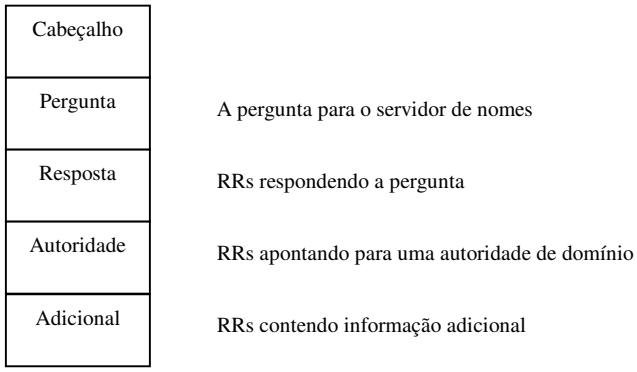

Figura 13 ***Seções das Mensagens DNS***

A RFC (*Request for Comments*) 3007 [28] propõe um método para executar atualizações dinâmicas de DNS seguras. Comunicação segura baseada em requisições e transações autenticadas é utilizada para fornecer autorização.

O formato completo de uma mensagem de atualização é mostrado na figura 14.

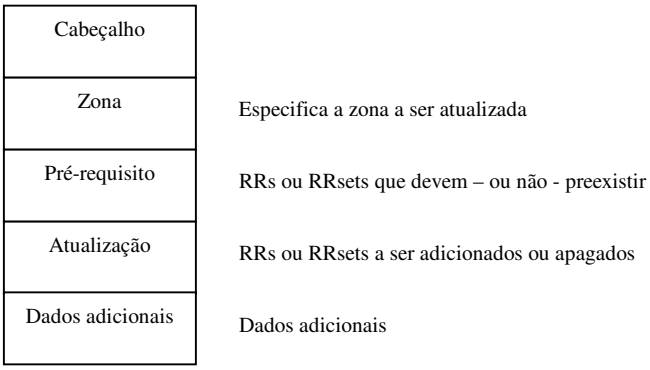

Figura 14 ***Mensagem de Atualização***

A seção Cabeçalho especifica que esta mensagem é uma atualização, e descreve o tamanho das outras seções. A seção Zona nomeia a zona que está sendo atualizada por esta mensagem.

A seção Pré-requisito especifica as premissas necessárias para esta atualização. A seção Atualização contem as alterações a serem feitas, e a seção Dados Adicionais contem dados que podem ser necessários para o complemento - mas que não fazem parte - desta atualização.

A atualização usa os mesmos campos do Formato das Mensagens DNS, e os mesmos formatos de seção, mas o nome e o uso destas seções difere como mostrado no cabeçalho modificado da figura 15.

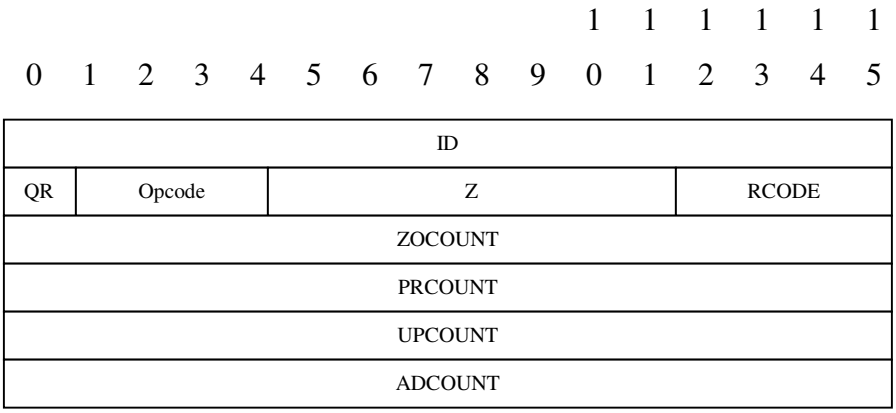

Figura 15 ***Cabeçalho da Atualização***

Estes campos são usados como segue: ID – Um identificador de 16 *bits* designado pela entidade que gera qualquer tipo de requisição; QR – Um campo de um *bit* que especifica quando esta mensagem é uma requisição, ou uma resposta; *Opcode* – Um campo de quatro *bits* que especifica o tipo de requisição nesta mensagem; Z – Reservado para uso futuro; RCODE – *Response Code*; ZOCOUNT – O número de RRs (*Resource Record*) na seção Zona; PRCOUNT – O número de RRs na seção Pré-requisito; UPCOUNT – O número de RRs na seção Atualização; e ADCOUNT – O número de RRs na seção Dados Adicionais.

A seção Zona tem o mesmo formato daquele especificado na RFC que define os nomes de domínio, com os campos redefinidos como na figura 16.

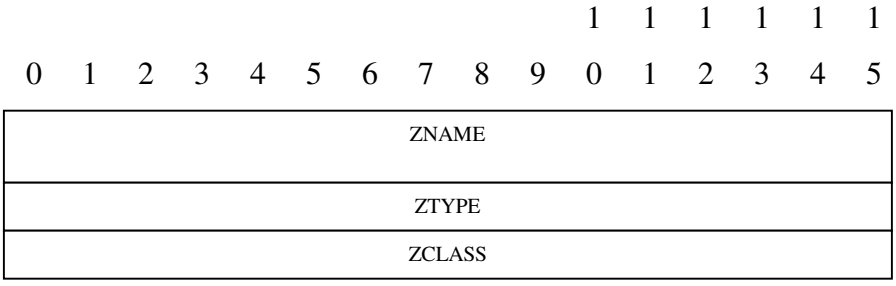

Figura 16 ***Seção Zona***

A Atualização usa esta seção para determinar a zona dos registros que estão sendo atualizados.

O servidor primário para uma zona dinâmica deve incrementar o número serial SOA (*Start Of Authority*) da zona, quando uma alteração ocorrer, ou antes da próxima recuperação do SOA. Uma sugestão de implementação aparece na figura 17.

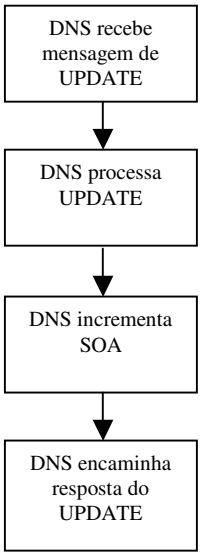

Figura 17 ***Sugestão de Implementação***

Com relação à segurança das transações DNS, a troca de mensagens DNS que incluam registros TSIG (*Transaction Signature*) ou SIG(0) permite que duas entidades DNS autentiquem requisições e respostas DNS enviadas entre elas.

A criptografia de chave pública é escalável porque as chaves públicas estão armazenadas no DNS. A título de comparação entre autenticação de dados e autenticação de mensagem, esta última fornece proteção para a mensagem inteira com uma única assinatura e uma única verificação, que são uma criação e checagem de MAC (*Message Authentication Code*) relativamente leve.

Registros DNSSEC (*DNS Security Extensions*) SIG (*Signature*) podem ser usados para proteger a integridade de RRs individuais, ou de RRsets (*Resource Record Sets*) em uma mensagem DNS, com a assinatura do administrador da zona. Entretanto, usar registros SIG para proteger RRsets em uma requisição de atualização é incompatível com o projeto da

atualização, e requereria de qualquer forma múltiplas e pesadas assinaturas e verificações de chave pública.

Além disto, os registros SIG não protegem o cabeçalho da mensagem. Ainda nesta linha, se os registros SIG fossem usados para proteger a seção de pré-requisitos, seria impossível determinar quando os SIGs seriam pré-requisitos e quando eles estariam sendo usados simplesmente para validação.

Na seção de alteração de uma requisição de atualização, requisições assinadas para adicionar um RRset são recomendáveis. Mas, se um RRset for deletado, não restará nenhum dado para o SIG proteger.

Quanto às assinaturas de dados e das mensagens, o processo de validação do DNSSEC executado por um resolvedor de nomes não deve processar nenhuma chave que não seja chave de zona, a menos que a política do servidor local afirme o contrário. A utilidade primária das chaves de *host* e de usuário é autenticar mensagens.

Quanto à autenticação, registros TSIG ou SIG(0) devem ser incluídos em todas as mensagens de atualização dinâmica segura.

As assinaturas SIG(0) não devem ser geradas por chaves de zona, uma vez que as transações são iniciadas por um *host* ou por um usuário. Registros DNSSEC SIG podem ser incluídos em uma mensagem de atualização.

Se uma atualização falhar porque foi assinada com uma chave não autorizada, o servidor deve indicar falha retornando uma mensagem com RCODE REFUSED. Outros erros TSIG ou de atualização dinâmica são retornados como especificado na descrição de protocolo apropriada.

Com relação às políticas, todas são configuradas pelo administrador da zona, e aplicadas pelo servidor de nomes primário da zona. As políticas do servidor definem os critérios que determinam se a chave usada para assinar a atualização pertence a alguma entidade autorizada a executar as alterações requisitadas.

4.2.1) Políticas

As políticas são completamente implementadas na configuração do servidor de zona primário por várias razões. Uma mudança nas políticas, ou um novo tipo de políticas não deve afetar o protocolo DNS, ou o formato de dados.

Quanto às políticas padrão, as implementações devem permitir às políticas de controle de acesso usar a chave utilizada para assinar as mensagens como um sinal de autorização. Uma prática comum poderia ser restringir as permissões do usuário de uma chave por nome de domínio.

Um servidor deve permitir alterações restritas pelo tipo do RR, a fim de que o usuário de uma chave contemplada nas políticas, possa modificar tipos de registros específicos em certos nomes. As implementações devem permitir controle de acesso por tipo, e devem fornecer representações concisas de todos os tipos, e de todos os tipos "de usuário".

Com relação a estes tipos "de usuário", eles incluem todos os tipos de dados exceto SOA, NS (*Name Server*), SIG e NXT (*Non-Existent*). Registros NXT não devem ser modificados através da alteração dinâmica.

Quanto às políticas adicionais, os usuários são livres para implementar quaisquer políticas. Elas podem depender da chave utilizada para assinar ou de qualquer outra característica da mensagem assinada.

Com relação à interação com o DNSSEC, existe uma série de pontos que devem ser esclarecidos. Uma requisição de atualização autorizada pode incluir registros SIG com cada RRset.

Os registros SIG são adicionados sem verificação. O servidor pode examinar registros SIG, e descartar os que tiverem um período de validade vencido.

Quanto à exclusão de SIGs o servidor pode optar por sobrepor sua autoridade e rejeitar a alteração. O servidor pode permitir que todos os registros SIG não gerados por uma chave de zona sejam deletados.

Com relação às alterações não explícitas dos SIGs, se a zona alterada é uma zona protegida pelo DNSSEC, o RRset afetado por uma operação de alteração deve ser assinado de acordo com a política de assinatura da zona. Quando os conteúdos de um RRset são alterados, o servidor pode deletar todos os registros SIG associados.

Quanto aos efeitos na zona, se quaisquer mudanças forem feitas, o servidor deve gerar um novo registro SOA e novos registros NXT. As alterações do registro SOA são permitidas.

Com relação às considerações de segurança, a chave da zona e possivelmente outros materiais criptográficos secretos serão mantidos em um computador, conectado à rede e sempre acessível. Permitir alterações de registros KEY pode levar a resultados indesejáveis, e possivelmente à mascaração do proprietário de uma chave.

4.3) Considerações finais

O objetivo desse capítulo foi apresentar os conceitos da atualização dinâmica de DNS. Quando visitamos um *web site*, nosso navegador "solicita" uma busca do endereço IP daquele *site*.

Permitir alterações de registros DNS pode levar a resultados indesejáveis, e possivelmente à mascaração do proprietário de uma chave, caso não se tenha os devidos cuidados com a segurança. Por isso a nossa proposta, que é de um mecanismo de nomes para suportar macro-mobilidade baseado em *DNS Update,* pretende não descuidar das questões relativas à segurança.

Capítulo 5 - Proposta

5.1) Introdução

O objetivo desse capítulo é apresentar a nossa proposta de um mecanismo de nomes para suportar macro-mobilidade com IP Móvel. Esse capítulo está estruturado da seguinte forma: a seção 5.2 apresenta uma visão geral da proposta definindo duas estratégias para sua implementação: atualização de nomes iniciada pelo HA e atualização de nomes iniciada pelo cliente; a seção 5.3 apresenta o detalhamento do mecanismo de atualização baseado no HA; a seção 5.4 apresenta o detalhamento do mecanismo de atualização baseado no cliente; a seção 5.5 apresenta uma análise comparativa entre as duas estratégias, indicando suas vantagens e desvantagens relativas; finalmente, a seção 5.6 apresenta as considerações finais do capítulo.

5.2) Estratégias para implementação do mecanismo de nomes

Conforme discutido nos capítulos anteriores, a implantação de redes móveis com tecnologias heterogêneas pode levar a um cenário de utilização no qual um usuário precisa se autenticar em HAs (*home agents*) distintos. Nesse cenário, o usuário muda o seu endereço IP de referência (*o home address*). A fim de manter a rastreabilidade do usuário móvel, esta dissertação propõe um mecanismo de atualização de nomes dinâmico que atualiza o endereço IP do usuário de maneira síncrona com o processo de autenticação no HA. Ao propor o mecanismo, os seguintes requisitos foram considerados:

1. O mecanismo deve evitar que agentes maliciosos alterem o mapeamento entre o nome e o endereço IP do nó móvel.
2. O processo de atualização do DNS deve estar sincronizado com o processo de autenticação do HA, pois é nesse momento que o endereço IP é efetivamente atribuído ao usuário.
3. A validade do registro DNS deve estar condicionada ao tempo de mapeamento (*binding* de um *home address* com o CoA) definido pelo HA. Isto é, ao expirar o tempo de mapeamento no HA, o registro deve ser eliminado do DNS.

4. Se o usuário se desregistrar do HA, o mapeamento do DNS deve ser eliminado imediatamente.
5. A implementação do mecanismo deve ser baseada em padrões existentes, e introduzir o mínimo de modificações possíveis nas implementações do nó móvel, FA (*foreign agent*) e HA (*home agent*).
6. A solução proposta deve ter um tempo de resposta que não provoque uma latência significativa no processo de comunicação entre um *host* externo e um nó móvel.
7. A solução proposta deve ser escalável, isto é, o mecanismo proposto deve ser capaz de gerenciar uma grande quantidade de redes móveis sem perda significativa de desempenho. A solução deve também gerar a menor quantidade possível de mensagens de atualização na rede.

Durante os nossos estudos, concluímos que duas estratégias podem ser utilizadas para manter o sincronismo entre o processo de autenticação de um nó móvel e o servidor de nomes DNS: atualização iniciada pelo HA e atualização iniciada pelo cliente.

A figura 18 mostra um diagrama de seqüência reduzido do primeiro cenário - em que a atualização é disparada pelo HA - onde aparece apenas o processo para um registro.

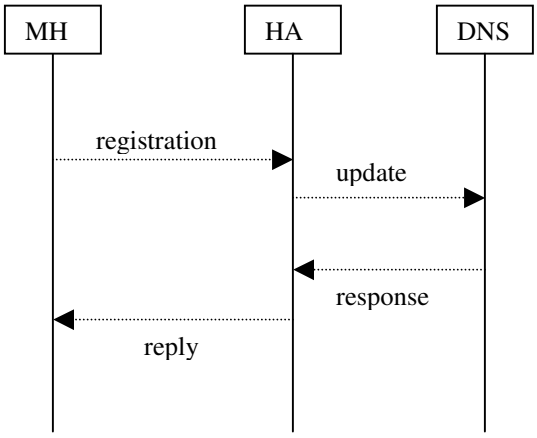

Figura 18 ***Cenário 1 (reduzido)***

Resumidamente temos que, após o HA processar com sucesso a solicitação de registro do MH, ele encaminha uma mensagem de atualização para o servidor DNS. Após receber a resposta do servidor DNS, ele responde ao MH, concluindo conjuntamente os processos de registro do IP Móvel e de atualização do DNS.

A figura 19 mostra um diagrama de seqüência reduzido do segundo cenário - em que a atualização é disparada pelo MH - onde aparece apenas o processo para um registro.

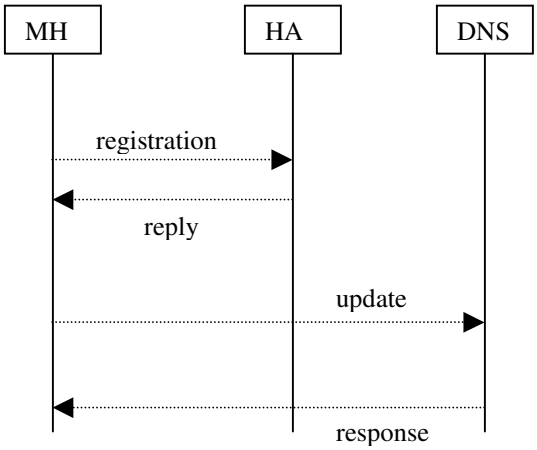

Figura 19 ***Cenário 2 (reduzido)***

Resumidamente temos que, após o MH encaminhar a solicitação de registro ao HA, ele aguarda a mensagem de resposta. Somente após receber a resposta do HA, ele envia a mensagem de atualização para o servidor DNS, havendo então um período em que ele já está registrado no HA mas ainda não atualizou o DNS.

5.3) Mecanismo de atualização baseado no HA

A figura 20 mostra o diagrama completo do primeiro cenário - em que a atualização é disparada pelo HA - onde aparece o processo para registro em mais de um HA.

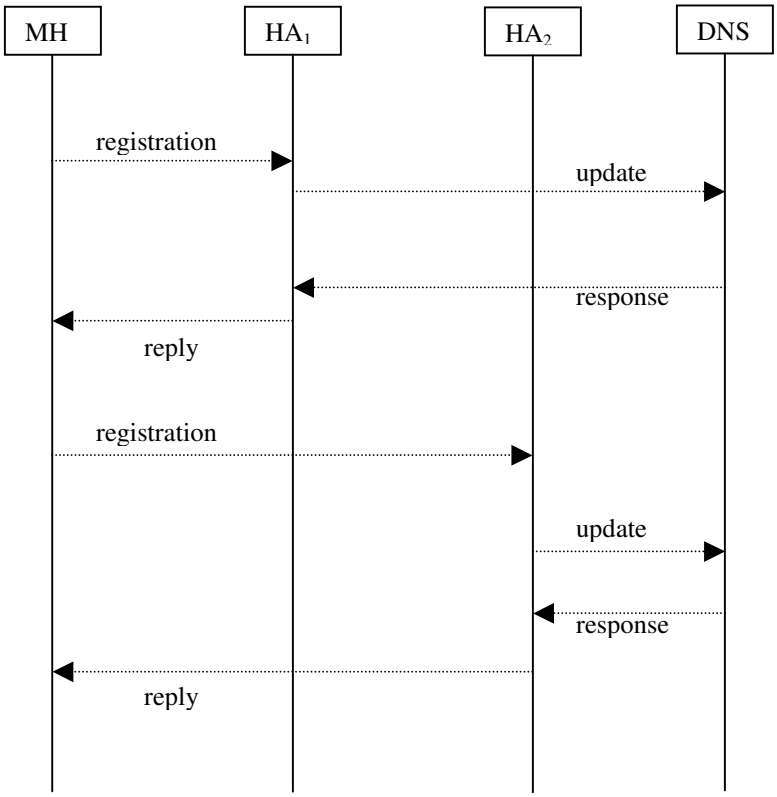

Figura 20 ***Cenário 1 (completo)***

Aqui temos que, após o HA_1 processar com sucesso a solicitação de registro do MH, ele encaminha uma mensagem de atualização para o servidor DNS. Após receber a resposta do servidor DNS, ele responde ao MH, concluindo conjuntamente os processos de registro do IP Móvel e de atualização do DNS.

Quando o MH entra na área de atuação do HA_2, após este processar com sucesso a solicitação de registro, ele também encaminha uma mensagem de atualização para o servidor DNS. Após receber a resposta do DNS, ele responde ao MH, concluindo a atualização da localização.

Um aspecto importante a ser considerado, no tocante à segurança, é a gerência das chaves envolvidas na autenticação das mensagens. Verifica-se que as técnicas para proteção

da comunicação de dados tem evoluído muito mais no sentido de desenvolvimento de artifícios para tornar ininteligível a um estranho ou intruso os dados que são transmitidos, do que no sentido de se proteger fisicamente os meios de comunicação para evitar o acesso indevido a eles.

Em outras palavras, não há maiores preocupações em impedir que um estranho "ouça" o que passa por um canal, mas são tomadas providências para que ele não consiga entender o que puder ouvir. E nem poderia ser diferente, já que os modernos canais de telecomunicações envolvem meios cuja proteção física em alto grau seria impraticável, por exemplo longos cabos passando por vias públicas.

No mecanismo de atualização baseado no HA, temos duas possibilidades de autenticação, gerar assinatura com chaves assimétricas ou simétricas. Utilizando chaves assimétricas, temos uma situação mais confortável com relação à gerência das chaves, uma vez que cada HA teria sua chave privada para assinar as mensagens de atualização, não precisando compartilhar chaves com o DNS, pois este desencriptaria as mensagens com a chave pública do respectivo HA.

Embora ideal quanto à gerência das chaves, esta abordagem não é recomendada para assinar mensagens de atualização dinâmica do DNS, uma vez que os algoritmos de chave assimétrica são computacionalmente "pesados" para que o servidor DNS os utilize para desencriptar um grande volume de mensagens de atualização, causando um problema de desempenho. Por outro lado, utilizando chave simétrica temos o problema da gerência das chaves, uma vez que teríamos que compartilhar a chave secreta entre os HAs e o servidor DNS.

Entretanto, com esta segunda abordagem não temos o problema de desempenho, já que os algoritmos de chave secreta são bem mais eficientes – computacionalmente falando – do que os de chaves pública / privada. Nossa proposta utilizou-se desta última abordagem, assumindo que as chaves secretas são configuradas estaticamente, contornando assim a dificuldade com a gerência das chaves.

Cabem, aqui, algumas palavras sobre o processo de autenticação. Autenticação é o processo de reconhecimento dos dados que são recebidos, comparando-se com os dados que foram enviados, e verificando se o transmissor que fez a requisição é, na verdade, o transmissor real.

Em nosso mecanismo, a autenticação é provida através da assinatura das mensagens de atualização, as quais são assinadas com a chave do transmissor. Gera-se um MAC, através do algoritmo HMAC-MD5, utilizando-se a mensagem DNS e a chave secreta do transmissor.

Adiciona-se a assinatura gerada ao pacote DNS, como o último RR da seção de dados adicionais da mensagem, possibilitando que o servidor DNS confira a identidade do transmissor por meio desta assinatura. O algoritmo com o processo para gerar a assinatura é mostrado na figura 21.

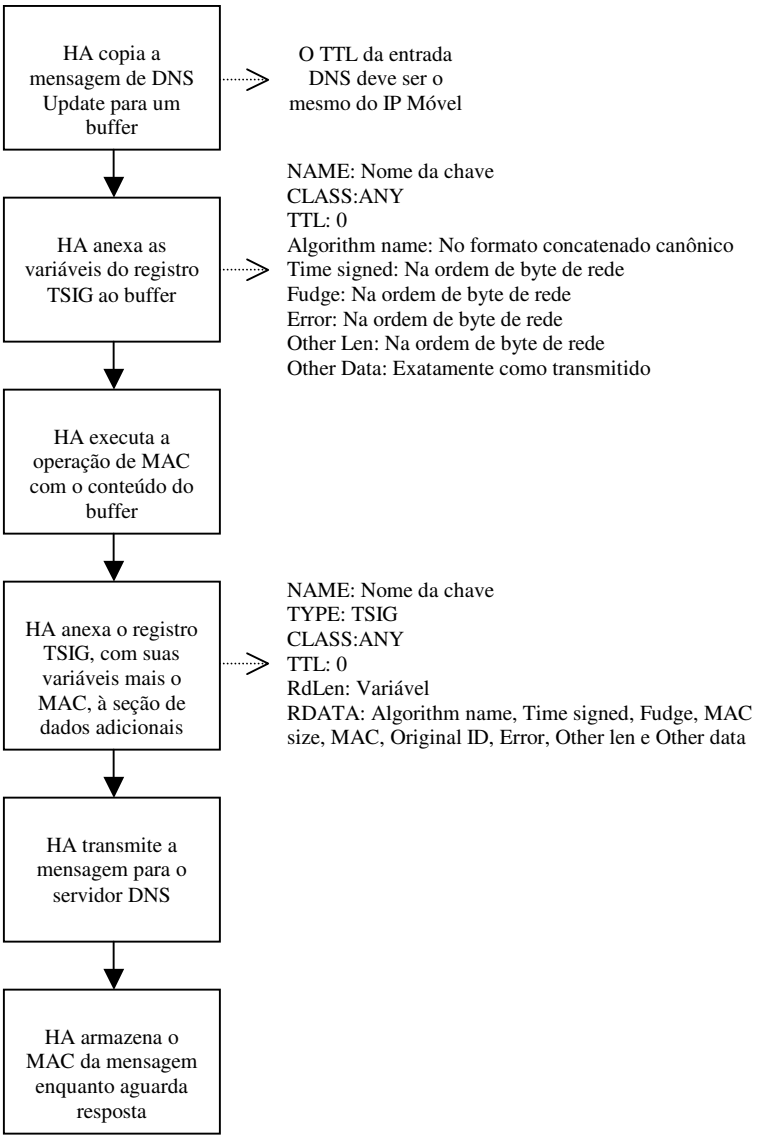

Figura 21 ***Algoritmo do processo para gerar assinatura no cenário 1***

No primeiro passo, o HA copia a mensagem de *DNS Update* inteira, sem incrementar o contador da seção de dados adicionais, e sem o registro TSIG, para um *buffer* destinado ao cálculo do MAC. A seguir, concatena com o conteúdo do *buffer* as variáveis do registro TSIG, que são: NAME, CLASS, TTL (*Time to Live*), nome do algoritmo, hora da assinatura, tolerância de tempo, código de erro, tamanho de outros dados, e outros dados.

No passo seguinte, o HA executa a operação de MAC com o conteúdo do *buffer,* operação esta que consiste em aplicar o algoritmo HMAC-MD5 (que é o único definido na RFC 2845) para calcular um *digest* com o conteúdo combinado com a chave secreta. Em seguida, acrescenta o registro TSIG, agora tendo o campo MAC preenchido com o resultado do passo anterior, à seção de dados adicionais da mensagem de *DNS Update,* e incrementa o contador de registros da seção de dados adicionais para refletir este acréscimo.

No passo seguinte, o HA transmite a mensagem para o servidor DNS, efetuando assim a atualização da entrada do MN. Concluindo, armazena o MAC da mensagem enquanto aguarda resposta, para poder autenticar a resposta recebida.

No DNS convencional, a alteração do valor de algum registro de um determinado nome, como por exemplo o endereço IP de uma máquina, é propagada com o auxílio do parâmetro TTL, que define o tempo de validade do mesmo. No entanto, este mecanismo requer a cooperação de todos os participantes do sistema, ou seja, todos os servidores devem periodicamente expurgar de seus *caches* registros que estejam com o tempo de validade vencido.

O TTL é um inteiro de 32 *bits* com sinal que especifica o intervalo de tempo que o *resource record* pode ficar armazenado antes que a fonte da informação deva novamente ser consultada. Valores zerados são interpretados como significando que o RR pode apenas ser usado para a transação em andamento, e não deve ser armazenado.

Por exemplo, registros SOA são sempre distribuídos com um TTL zero para proibir armazenamento. Valores zero podem também ser usados para dados extremamente voláteis.

Em outras palavras, o TTL especifica o intervalo de tempo que o *resource record* pode ficar armazenado antes que ele deva ser descartado. O TTL é um inteiro decimal.

Tanto os dados TTL para RRs quanto os dados de tempo para atividades de *refresh* dependem dos *timers* de 32 *bits* em unidades de segundos. Dentro da base de dados, *timers*

de *refresh* e TTLs para dados armazenados em *cache* estão conceitualmente em "contagem regressiva", enquanto dados na zona permanecem com TTLs constantes.

Como precisamos atualizar entradas DNS para MNs, é importante que o TTL destas entradas esteja sincronizado com o *Lifetime* do IP Móvel, caso contrário teríamos o intervalo de tempo expirando em momentos diferentes nos dois protocolos. Isto nos causaria alguns problemas, como no caso da entrada DNS ser descartada enquanto o registro do IP Móvel ainda é válido, ou vice-versa.

Números absolutos são considerados com respeito a alguma origem conhecida e convertidos para valores relativos quando colocados em uma resposta a uma consulta DNS. Quando um TTL absoluto fica negativo após a conversão para relativo, então o dado está expirado e deve ser ignorado.

O valor MINIMUM no SOA deve ser usado para configurar um limite inferior no TTL dos dados distribuídos por uma zona. Esta função de configurar um limite inferior deve ser executada quando o dado é copiado para o conteúdo de uma resposta.

Quando um TTL de um RR indica um tempo relativo, o RR não deve estar expirado, uma vez que ele é parte de uma zona. Quando o RR tem um tempo absoluto, ele é parte de um *cache,* e o TTL do RR é comparado com a estampa de tempo para o início da requisição.

Quando fazemos a atualização dinâmica, os RRs são adicionados à seção de atualização da mensagem DNS com os campos NAME, TYPE, TTL, RDLENGTH e RDATA povoados com o conteúdo a ser adicionado à zona, e o campo CLASS com o mesmo conteúdo do campo CLASS da zona. O conteúdo do campo TTL deve ser o mesmo do *Lifetime* do IP Móvel.

Quando apagamos um RR de um RRset, os RRs a serem excluídos são adicionados à seção de atualização da mensagem DNS. Os campos NAME, TYPE, RDLENGTH e RDATA devem ser iguais ao do RR a ser apagado.

O TTL deve ser especificado como zero e será ignorado pelo servidor DNS. O campo CLASS deve ser especificado como NONE para distinguir esta operação de uma inclusão.

Se não existir nenhum RR que atenda às condições, então este RR de atualização será ignorado silenciosamente pelo servidor DNS. Uma implementação possível para a

atualização dinâmica, é apagar o RR antigo e adicionar o novo, e esta foi a implementação utilizada para nossos testes.

5.4) Mecanismo de atualização baseado no cliente

A figura 22 mostra o diagrama completo do segundo cenário - em que a atualização é disparada pelo MH - onde aparece o processo para registro em mais de um HA.

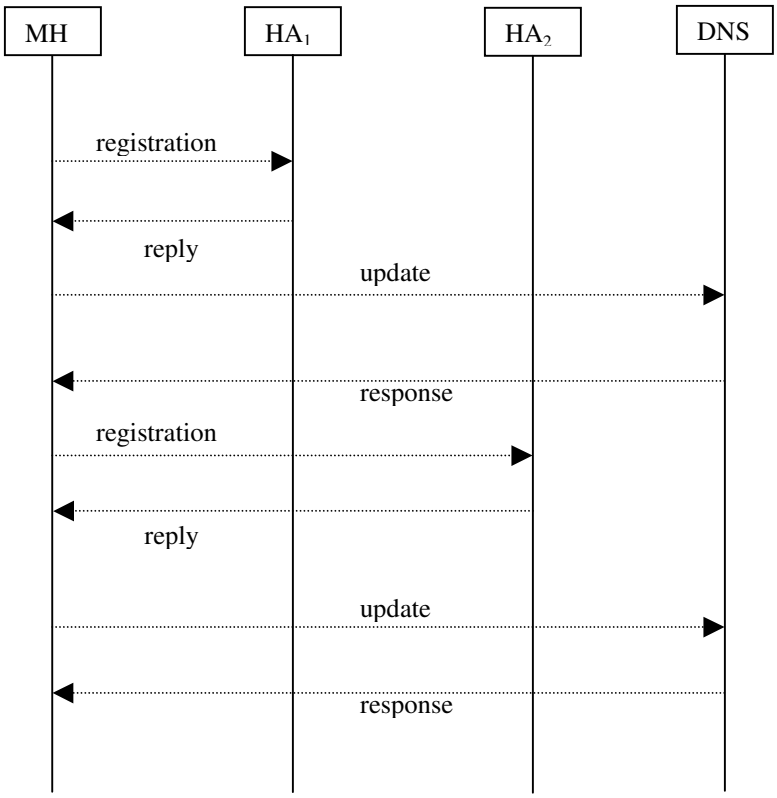

Figura 22 ***Cenário 2 (completo)***

Aqui temos que, após o MH encaminhar a solicitação de registro ao HA_1, ele aguarda a mensagem de resposta. Somente após receber a resposta do HA_1, ele envia a mensagem de atualização para o servidor DNS, havendo então um período em que ele já está registrado no HA_1 mas ainda não atualizou o DNS.

Quando o MH entra na área de atuação do HA_2, após ele encaminhar a solicitação de registro para este HA, ele novamente aguarda a mensagem de resposta. Somente após receber a resposta do HA_2, ele envia a mensagem de atualização para o servidor DNS,

havendo então um período em que ele já está registrado no HA_2 mas sua entrada DNS continua apontando para o HA_1.

Mais uma vez, a gerência das chaves envolvidas na autenticação das mensagens é um aspecto importante a ser considerado, no que se refere ao quesito segurança. O problema básico da utilização de autenticação é a gerência das chaves de cifragem, se as chaves forem descobertas o sistema estará comprometido e se forem perdidas as informações não poderão mais ser acessadas.

No nosso caso, se o MN perder a chave não conseguirá assinar as mensagens, e se o servidor DNS perder a chave não conseguirá conferir a assinatura.

No mecanismo de atualização baseado no MN, temos as mesmas duas possibilidades de autenticação, gerar assinatura com chaves assimétricas ou simétricas. Utilizando chaves assimétricas, temos uma situação mais confortável com relação à gerência das chaves, uma vez que cada MN teria sua chave privada para assinar as mensagens de atualização, não precisando compartilhar chaves com o DNS, pois este desencriptaria as mensagens com a chave pública do respectivo MN.

Isto é, nos deparamos com as mesmas considerações a respeito da gerência das chaves, sendo que não há alteração para a situação do mecanismo abordado na seção anterior.

Aqui, novamente, algumas palavras sobre o processo de autenticação precisam ser ditas. Autenticação utiliza o modelo cliente - servidor.

Um cliente faz a requisição para o servidor, que verifica se o cliente tem a permissão para acessar o servidor.

Em nosso mecanismo, a autenticação é provida através da assinatura das mensagens de atualização, as quais são assinadas com a chave do transmissor. Gera-se um MAC, através do algoritmo HMAC-MD5, utilizando-se a mensagem DNS e a chave secreta do transmissor.

Adiciona-se a assinatura gerada ao pacote DNS, como o último RR da seção de dados adicionais da mensagem, possibilitando que o servidor DNS confira a identidade do transmissor por meio desta assinatura. O algoritmo com o processo para gerar a assinatura é mostrado na figura 23.

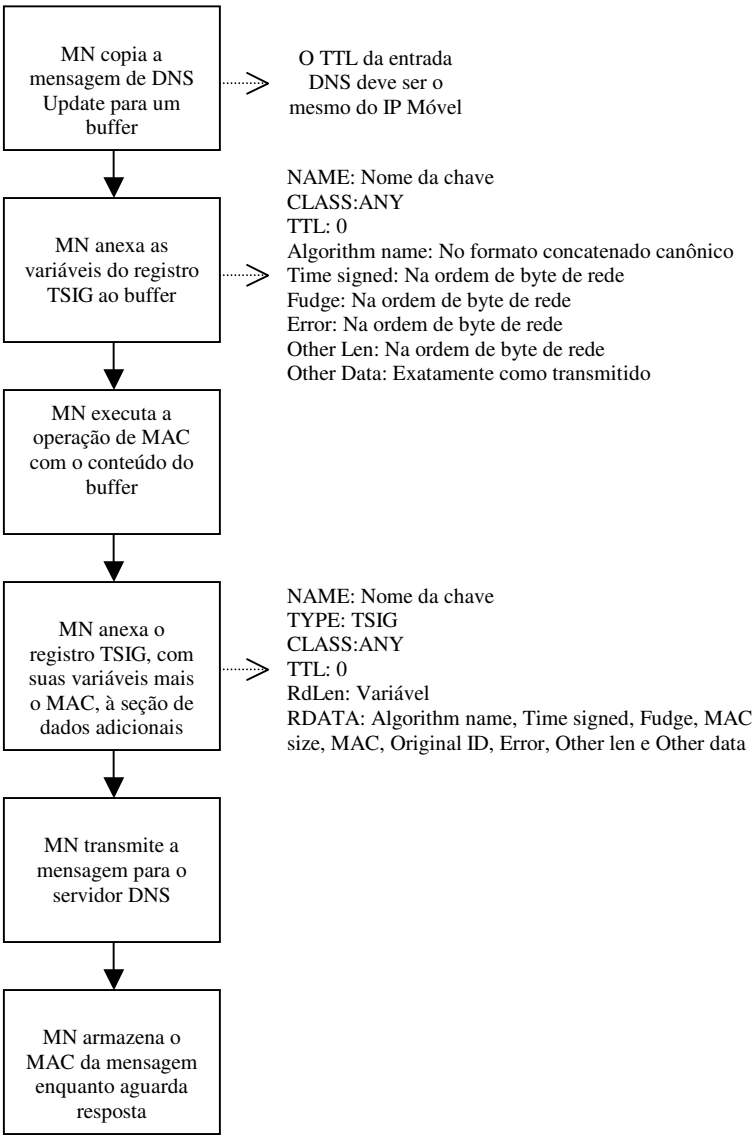

Figura 23 ***Algoritmo do processo para gerar assinatura no cenário 2***

No primeiro passo, o MN copia a mensagem de *DNS Update* inteira, sem incrementar o contador da seção de dados adicionais, e sem o registro TSIG, para um *buffer* destinado ao cálculo do MAC. A seguir, concatena com o conteúdo do *buffer* as variáveis do registro TSIG, que são: NAME, CLASS, TTL, nome do algoritmo, hora da assinatura, tolerância de tempo, código de erro, tamanho de outros dados, e outros dados.

No passo seguinte, o MN executa a operação de MAC com o conteúdo do *buffer,* operação esta que consiste em aplicar o algoritmo HMAC-MD5 (que é o único definido na RFC 2845) para calcular um *digest* com o conteúdo combinado com a chave secreta. Em

seguida, acrescenta o registro TSIG, agora tendo o campo MAC preenchido com o resultado do passo anterior, à seção de dados adicionais da mensagem de *DNS Update,* e incrementa o contador de registros da seção de dados adicionais para refletir este acréscimo.

No passo seguinte, transmite a mensagem para o servidor DNS*,* efetuando assim a atualização da sua entrada. Concluindo, o MN armazena o MAC da mensagem enquanto aguarda resposta, para poder autenticar a resposta recebida.

5.5) Análise comparativa entre as duas estratégias

Comparamos as duas estratégias na tabela 2.

Tabela 2 ***Análise comparativa entre as duas estratégias***

	HA	Cliente
Modificações	apenas no HA	em todos os clientes móveis
Gerência de Chaves	simples (poucas chaves)	complexa (muitas chaves)
Segurança	parcialmente satisfatória	satisfatória
Desempenho	depende do roteador	depende da máquina cliente
Escalabilidade	sobrecarga no HA	distribuída nos clientes

Comparando os dois mecanismos apresentados neste capítulo, estudamos cinco aspectos, apresentados na tabela 2. O aspecto *Modificações* diz respeito às alterações necessárias na infra-estrutura atual, o *Gerência de Chaves* refere-se ao gerenciamento das chaves criptográficas, *Segurança* trata da confiabilidade do mecanismo, *Desempenho* refere-se à velocidade de execução e *Escalabilidade* diz respeito ao desempenho *versus* a quantidade de acessos simultâneos.

Quanto ao aspecto *Modificações,* o mecanismo baseado no HA necessita apenas de alterações no próprio HA, sem a imposição de alterações nos FAs e nos clientes móveis. Já a atualização baseada no cliente móvel exige alterações em todos os clientes que forem

utilizar o mecanismo, mas não necessita de alterações na infra-estrutura de rede, seja nos HAs ou FAs.

Com relação ao aspecto *Gerência de Chaves,* o mecanismo baseado no HA é mais simples, uma vez que poucas chaves estão envolvidas. Por outro lado, a atualização baseada no cliente móvel é mais complexa, por que cada cliente partilha sua chave secreta com o servidor DNS.

Quanto ao aspecto *Segurança,* o mecanismo baseado no HA é parcialmente satisfatório, por que diferentes entidades alterarão a mesma entrada DNS. Já a atualização baseada no cliente móvel é completamente satisfatória, uma vez que cada cliente – e apenas ele – alterará sua entrada DNS.

Com relação ao aspecto *Desempenho,* o mecanismo baseado no HA depende do *hardware* desta entidade, que freqüentemente possui mais recursos que os clientes. Por outro lado, a atualização baseada no cliente depende do *hardware* dos clientes móveis, que na maioria das vezes possuem recursos bastante limitados.

Quanto à *Escalabilidade,* o mecanismo baseado no HA provoca uma sobrecarga no HA, visto que esta entidade pode ter de lidar com um grande número de acessos simultâneos, tendo que – além de lidar com a autenticação do IP Móvel – realizar as atualizações de DNS de todos os solicitantes. Já a atualização baseada no cliente distribui esta carga entre os clientes móveis, o que elimina o problema da sobrecarga, uma vez que o HA continua executando apenas a sua tarefa relativa ao IP Móvel.

5.6) Considerações finais

O objetivo desse capítulo foi apresentar a nossa proposta de um mecanismo de nomes para suportar macro-mobilidade com IP Móvel. Foi apresentada uma visão geral da proposta definindo duas estratégias para sua implementação: atualização de nomes iniciada pelo HA e atualização de nomes iniciada pelo cliente.

Também foram detalhados estes mecanismos, e comparadas as duas estratégias, indicando suas vantagens e desvantagens relativas. A grande contribuição que esta proposta pretende dar é a utilização de um mecanismo de nomes para resolver o problema da

rastreabilidade do nó móvel quando ele se conecta a redes administradas por entidades independentes.

Capítulo 6 - Avaliação

6.1) Introdução

O objetivo desse capítulo é avaliar os requisitos 6 e 7 conforme definidos no capítulo anterior, que serão repetidos aqui para comodidade do leitor: Req. 6 – Impacto no tempo de resposta; Req. 7 - Escalabilidade. Esse capítulo está estruturado da seguinte forma: a seção 6.2 apresenta o ambiente de avaliação; a seção 6.3 apresenta a avaliação do tempo de resposta; a seção 6.4 apresenta a avaliação de escalabilidade; e a seção 6.5 apresenta as considerações finais do capítulo.

6.2) Ambiente de Avaliação

A idéia de rodar uma versão virtual do Linux "dentro" do Linux, simulando a presença de um segundo computador, totalmente virtual e incluindo seus próprios discos, arquivos, configurações e processos, não parece nada fora do comum para quem quer que já tenha utilizado *softwares* como o *VMware*, sistema comercial que emula um computador PC virtual rodando totalmente em *software,* permitindo acrobacias como rodar uma sessão de *Windows* dentro do Linux ou vice-versa, sem interferir no funcionamento do computador hospedeiro. De fato, o conceito de máquinas virtuais é muito mais antigo que *softwares* como o *VMware,* e é lugar comum há muito tempo em arquiteturas de maior porte.

Como o Linux atualmente suporta uma série de arquiteturas diferentes, como os PCs, Macs, SPARCs e Alphas, o seu *kernel* inclui código específico para lidar com as particularidades de cada uma delas. O *User-mode Linux* é especial neste particular, pois ele é uma implementação do *kernel* do Linux onde a arquitetura suportada é outro *kernel* do Linux.

Normalmente, os programas aplicativos fazem suas requisições ao *kernel,* que por sua vez aciona os recursos de *hardware* da máquina conforme solicitado. Mas no caso do UML, os programas do usuário fazem requisições ao *kernel* e, embora eles não suspeitem disso, este *kernel* vai passar todas as requisições para o *kernel* "real" da máquina hospedeira, e este sim vai fazer a comunicação com o *hardware*.

O uso de uma máquina virtual pode ser útil em várias situações. Em nosso trabalho, ele foi essencial para que pudéssemos simular uma rede real, em apenas uma máquina Linux.

O ambiente UML foi produzido utilizando-se os *kernels* Linux-2.4.18.tar e Linux-2.4.24.tar, o *patch* uml-patch-2.4.18-40, e a comunicação entre as máquinas virtuais foi realizada através de *multicast,* configurando-se no mesmo domínio *multicast* as interfaces que simulam estar em uma mesma rede física. Na verdade, o *kernel* Linux-2.4.18.tar foi utilizado para testar o ambiente, que depois foi migrado para o Linux-2.4.24.tar.

Para reproduzir o ambiente de avaliação, precisa-se do *kernel* do UML e de um sistema de arquivos especialmente preparado no qual ele possa dar *boot*. O *site* oficial do projeto – *user-mode-linux.sourceforge.net* – contém este *kernel,* bem como sistemas de arquivos completos de várias distribuições, já preparados e prontos para fazer o *download* e instalar.

Para o trabalho desta dissertação, obtivemos o *kernel* 2.4.18 e um sistema de arquivos populado com a distribuição *Red Hat 9*, totalizando em torno de 220 MB. Após os testes iniciais, descobrimos que o *kernel* utilizado não havia sido compilado com as opções necessárias para o funcionamento do pacote *Dynamics – HUT Mobile IP,* descritas no anexo B.

Por este motivo, obtivemos o código fonte do *kernel* 2.4.24, e o compilamos com as opções necessárias. Após a instalação e configuração – com sucesso – dos pacotes *Dynamics – HUT Mobile IP* e BIND, procedemos à montagem do ambiente de testes propriamente dito.

Foram montadas as seguintes máquinas virtuais: um servidor DNS, para rodar o BIND, e receber as atualizações dinâmicas; uma rede A, composta por um HA e dois FAs; uma rede B, também composta por um HA e dois FAs; e um MN, para ser o cliente alvo, movendo-se entre as redes virtuais. A comunicação entre as máquinas virtuais foi realizada através de *multicast,* configurando-se no mesmo domínio *multicast* as interfaces que simulam estar em uma mesma rede física.

A figura 24 ilustra a rede A.

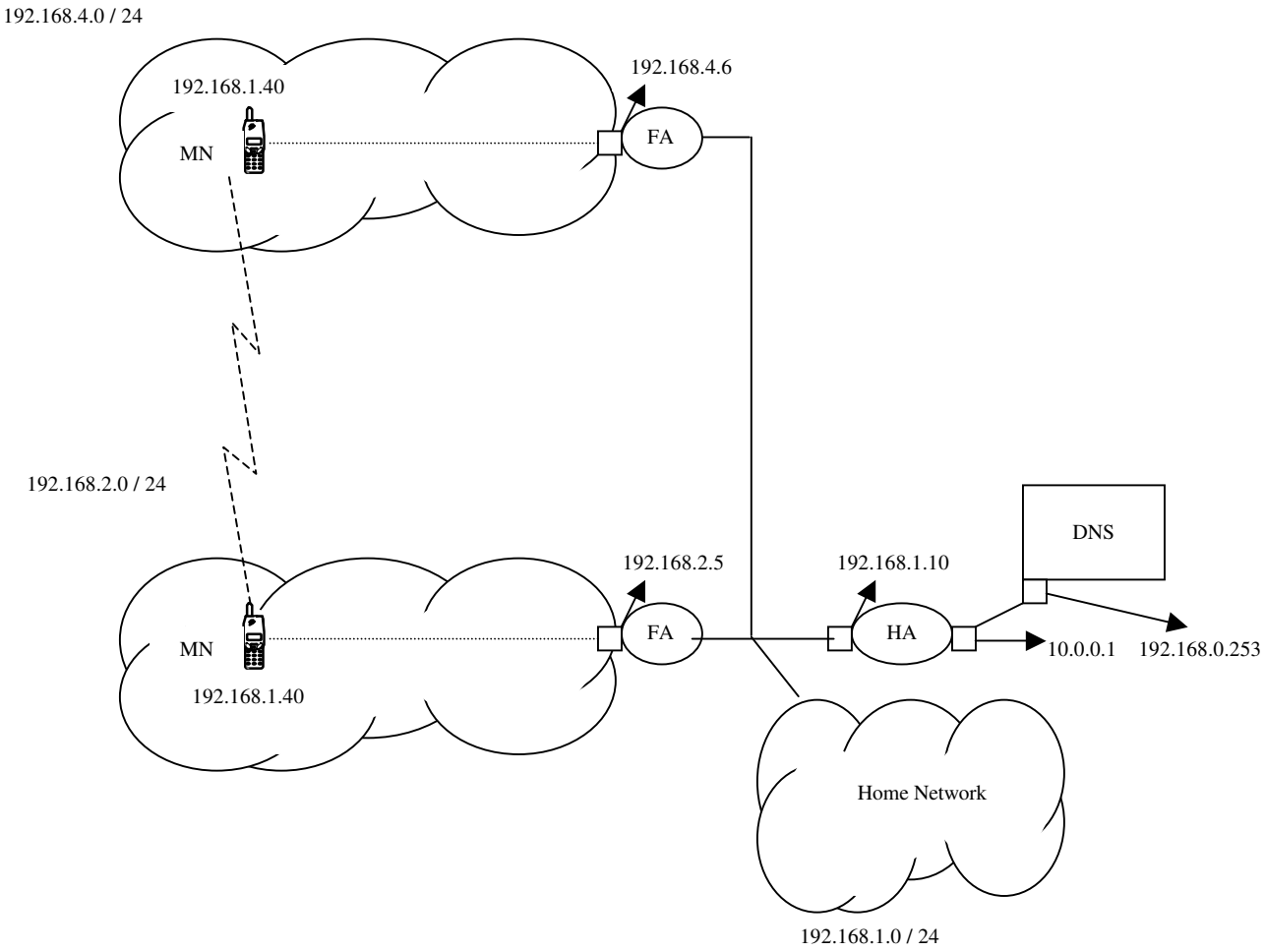

Figura 24 ***Rede A***

Quanto ao *multicast*, em um mesmo domínio foram colocados o servidor DNS e as interfaces externas dos dois HAs. Assim, simula-se o ambiente aberto da Internet, onde os roteadores externos dos domínios administrativos comunicam-se com o servidor DNS administrado por uma entidade independente.

Em outro domínio *multicast,* foram colocadas a interface interna do HA e as interfaces externas dos dois FAs da rede A. Assim, simula-se o ambiente da rede interna do domínio administrativo A, onde os roteadores internos do domínio administrativo comunicam-se, ligando suas sub-redes.

Em um terceiro domínio *multicast,* foram colocadas a interface interna do HA e as interfaces externas dos dois FAs da rede B. Assim, simula-se o ambiente da rede interna do domínio administrativo B, onde os roteadores internos do domínio administrativo comunicam-se, ligando suas sub-redes.

No último domínio *multicast,* foram colocadas as interfaces internas dos FAs e o MN. Assim, simula-se a conexão física do MN com os FAs, possibilitando a comunicação cada vez que o cliente entra em uma sub-rede.

6.3) Avaliação do tempo de resposta

Para a avaliação do tempo de resposta, capturamos os pacotes nas entidades que estavam disparando as atualizações dinâmicas, através da ferramenta *tcpdump*. Depois, para análise e exibição dos pacotes, utilizamos o utilitário *Ethereal*.

Na figura 25, vemos os pacotes capturados no primeiro cenário, onde a alteração do DNS é disparada pelo HA.

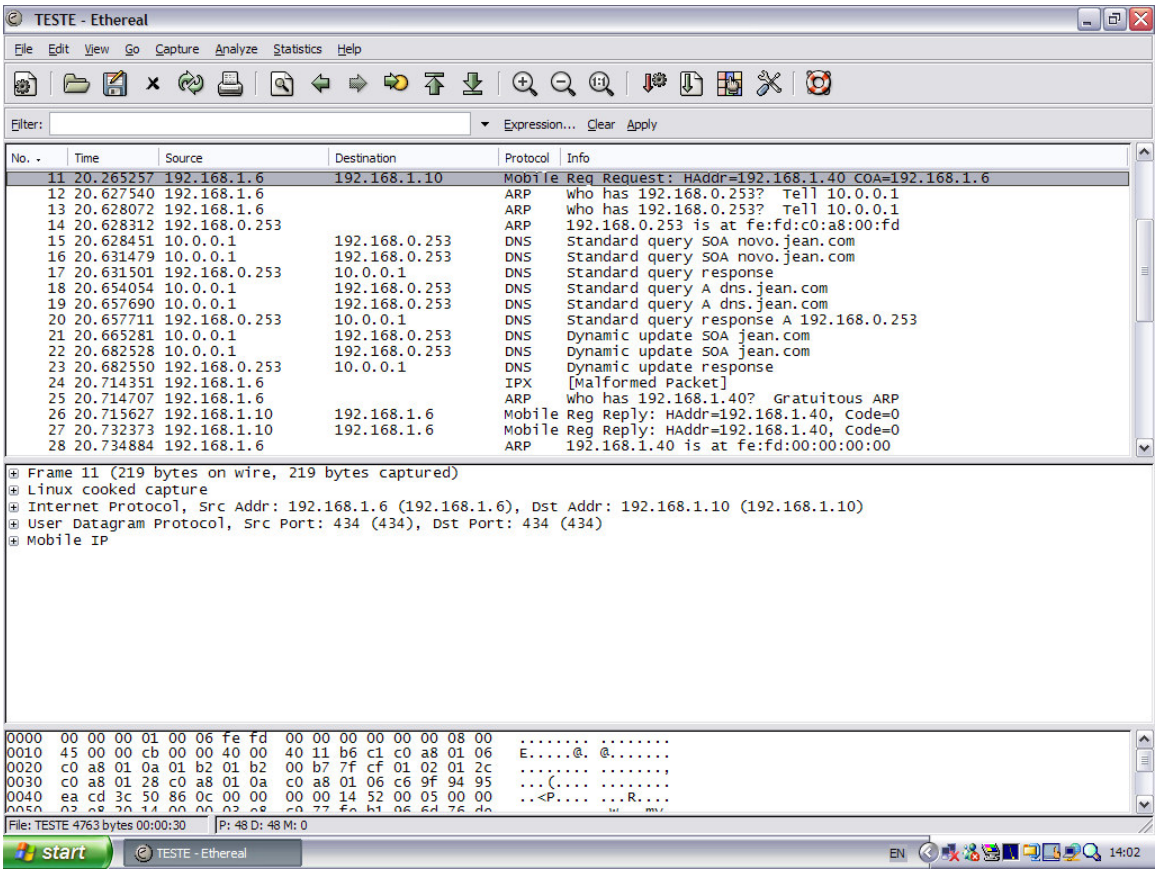

Figura 25 ***Primeiro cenário***

O primeiro pacote, na linha de número 11, e que está sublinhado, é o pacote da requisição de registro do MN. O pacote da linha número 21, cujo protocolo é o DNS, é o pacote da atualização dinâmica.

O pacote 23, cuja fonte é o endereço 192.168.0.253 (DNS), é o pacote com a resposta da atualização dinâmica do DNS. O pacote 26, cujo destino é o IP 192.168.1.6 (FA), é o pacote com a resposta do registro no IP Móvel.

Veremos agora, os tempos de cada pacote, para podermos avaliar o impacto no tempo de resposta global. A figura 26 mostra o tempo do primeiro pacote.

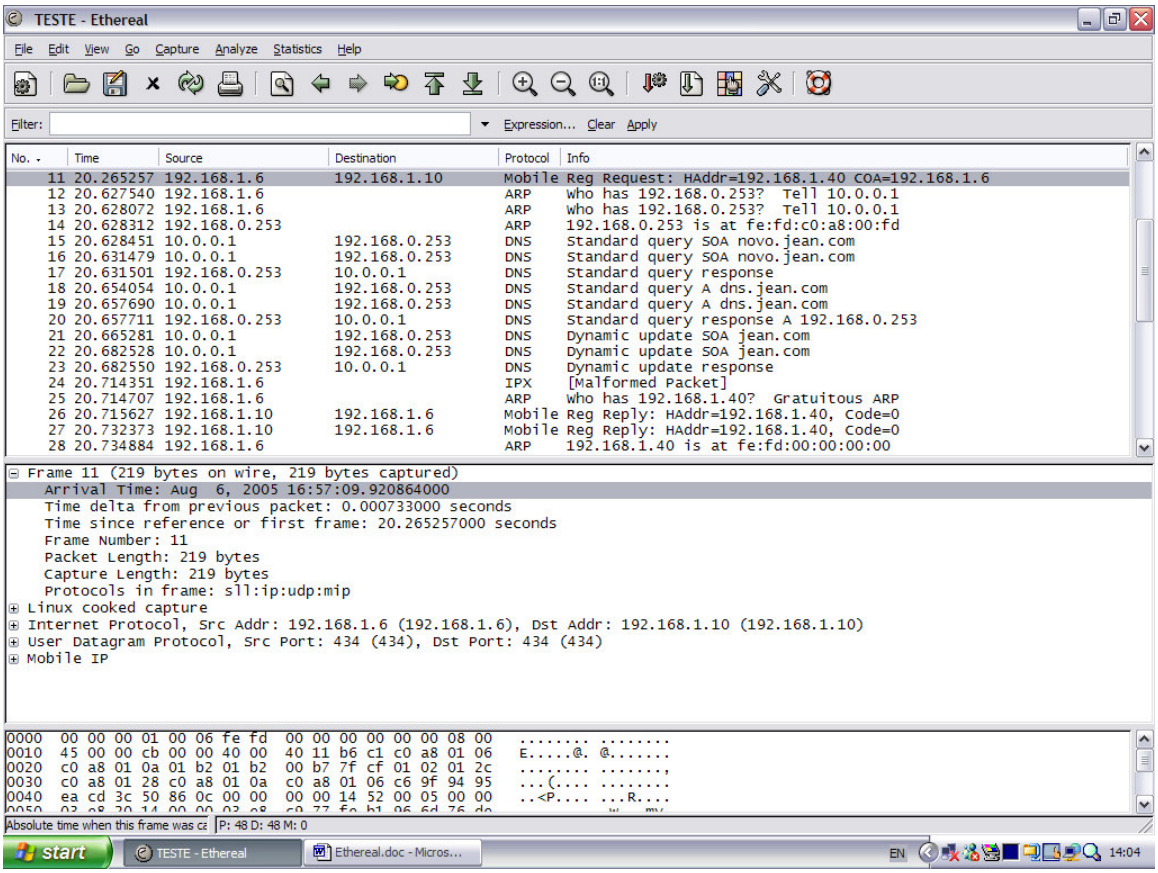

Figura 26 ***Registration Request***

No segundo quadro da figura, temos várias visões do pacote selecionado, visões estas relacionadas às camadas consideradas (rede, transporte, aplicação, etc). Na linha sublinhada neste quadro, vemos o tempo de chegada do pacote, com o horário 16:57:09.

A figura 27 mostra o tempo do segundo pacote.

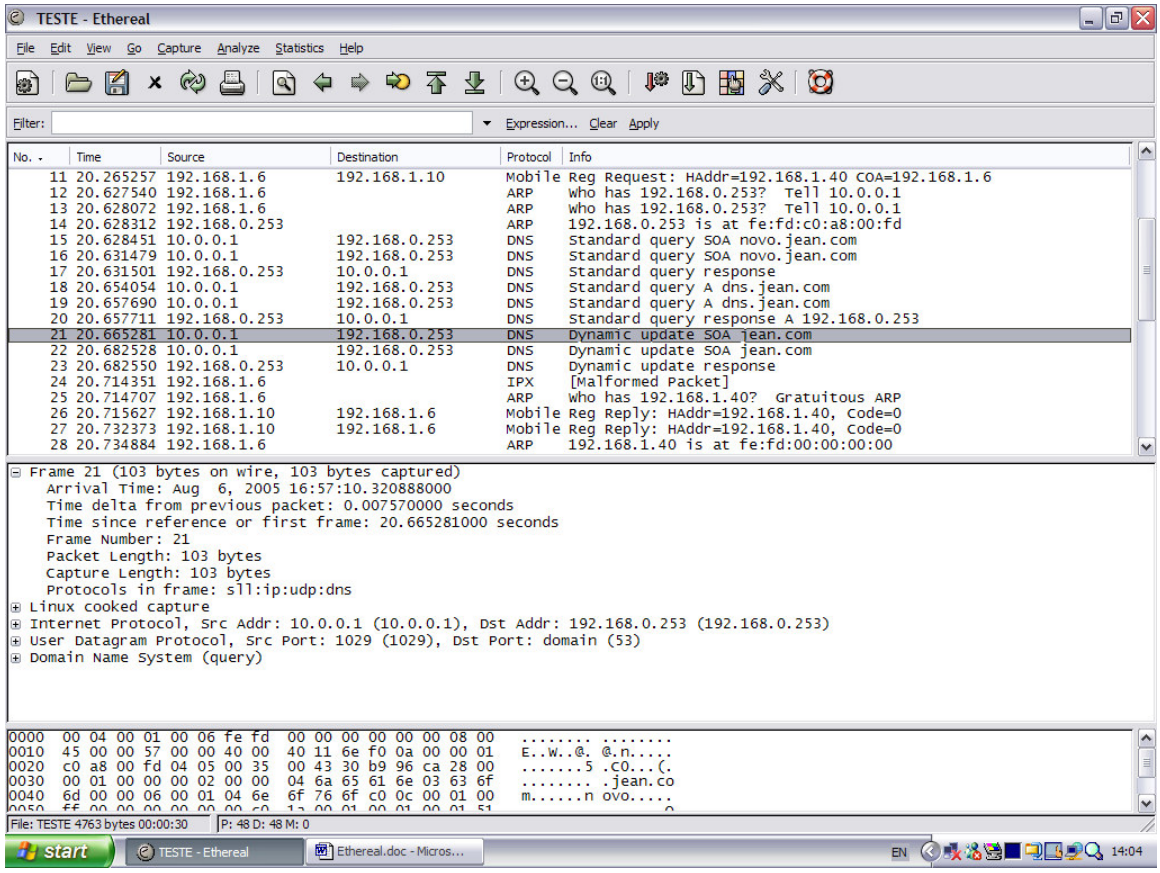

Figura 27 ***Dynamic update***

Na linha com a informação *Arrival Time* deste quadro, vemos o tempo de chegada do pacote à interface de rede, com o horário 16:57:10. A figura 28 mostra o tempo do terceiro pacote.

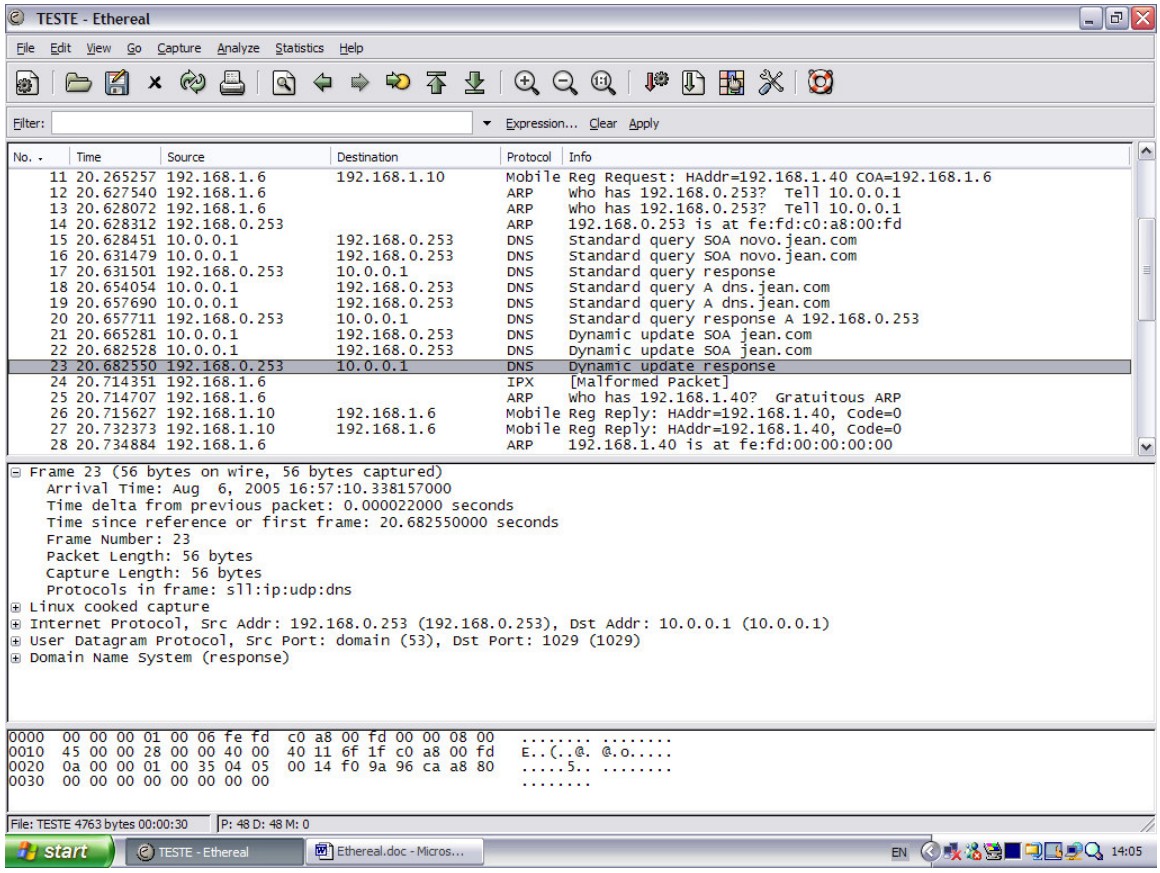

Figura 28 ***Dynamic response***

Na linha com a informação *Arrival Time* deste quadro, vemos o tempo de chegada do pacote com a resposta à atualização dinâmica, com o horário 16:57:10. A figura 29 mostra o tempo do último pacote.

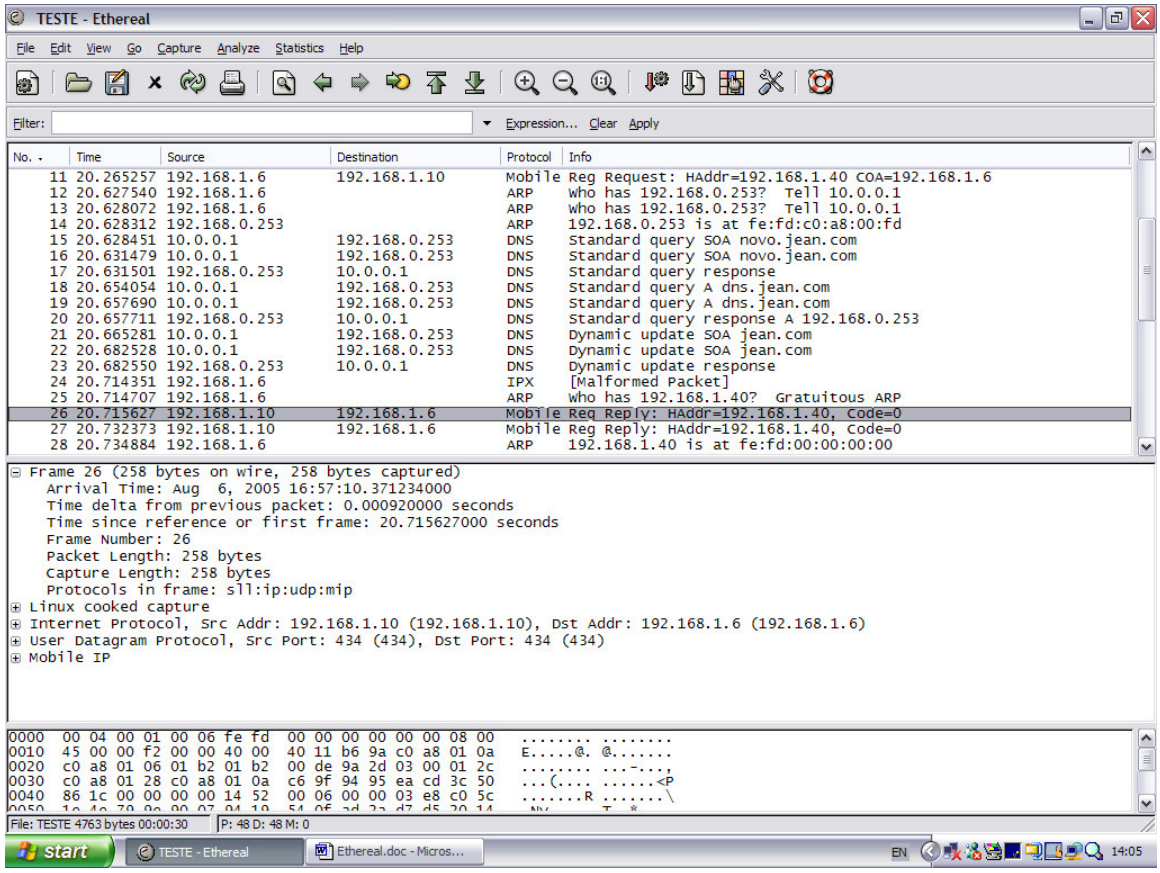

Figura 29 ***Registration Reply***

Na linha com a informação *Arrival Time* deste quadro, vemos o tempo de chegada do pacote à interface de rede, com o horário 16:57:10. Com isto, temos que, o tempo total da operação é de aproximadamente 1 segundo.

Percebemos que, apesar de introduzirmos a atualização do DNS no coração do registro IP Móvel, o impacto no tempo foi considerado desprezível. Isto pôde ser comprovado nos testes, uma vez que o registro do IP Móvel sem a atualização dinâmica consumia um tempo superior a meio segundo, em alguns casos próximo de 1 segundo.

Vamos, agora, ao segundo cenário. Na figura 30, vemos os pacotes capturados no cenário onde a alteração do DNS é disparada pelo cliente.

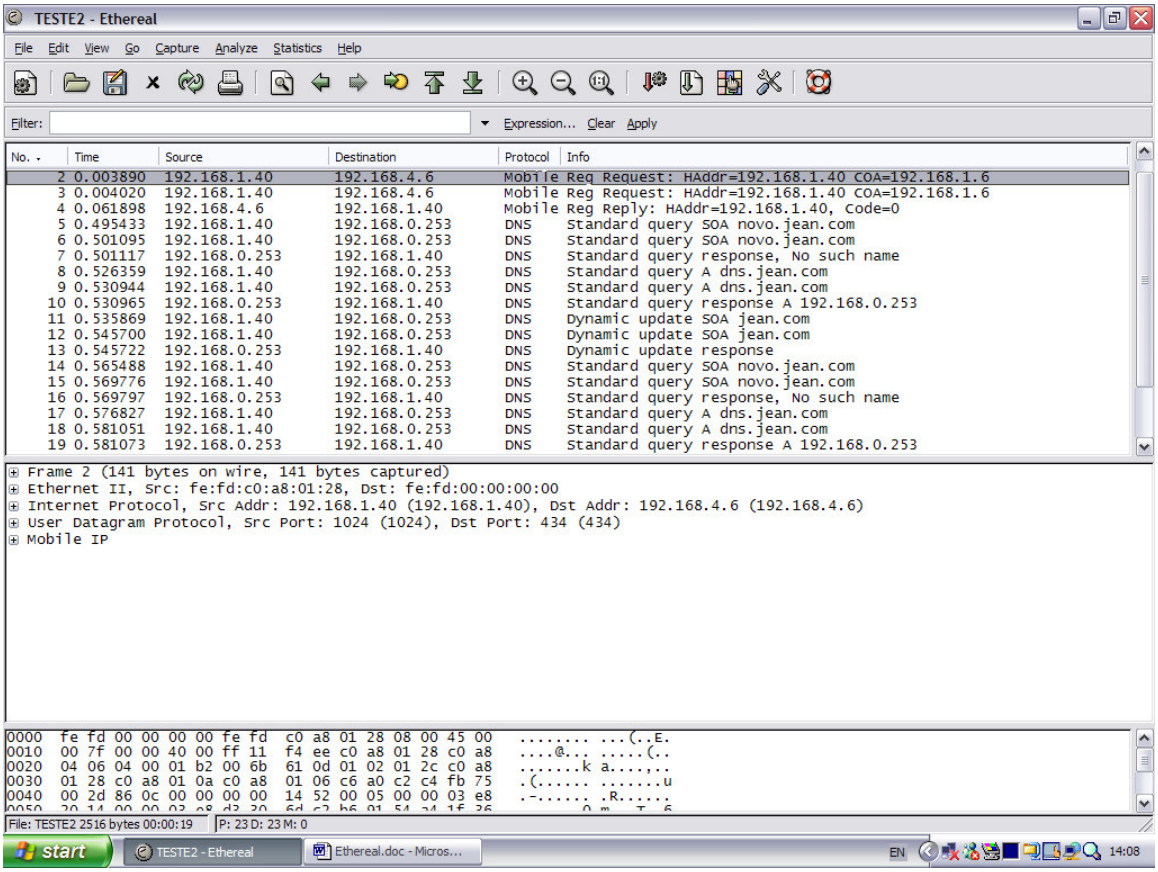

Figura 30 ***Segundo cenário***

O primeiro pacote, na linha de número 2, e que está sublinhado, é o pacote da requisição de registro do MN. O pacote da linha número 4, cuja fonte é o endereço 192.168.4.6 (FA), é o pacote com a resposta do registro no IP Móvel.

O pacote 11, cujo destino é o endereço 192.168.0.253 (DNS), é o pacote da atualização dinâmica do DNS. O pacote 13, cuja fonte é o IP 192.168.0.253 (DNS), é o pacote com a resposta da atualização dinâmica do DNS.

Veremos agora, os tempos de cada pacote, para podermos avaliar o impacto no tempo de resposta global. A figura 31 mostra o tempo do primeiro pacote.

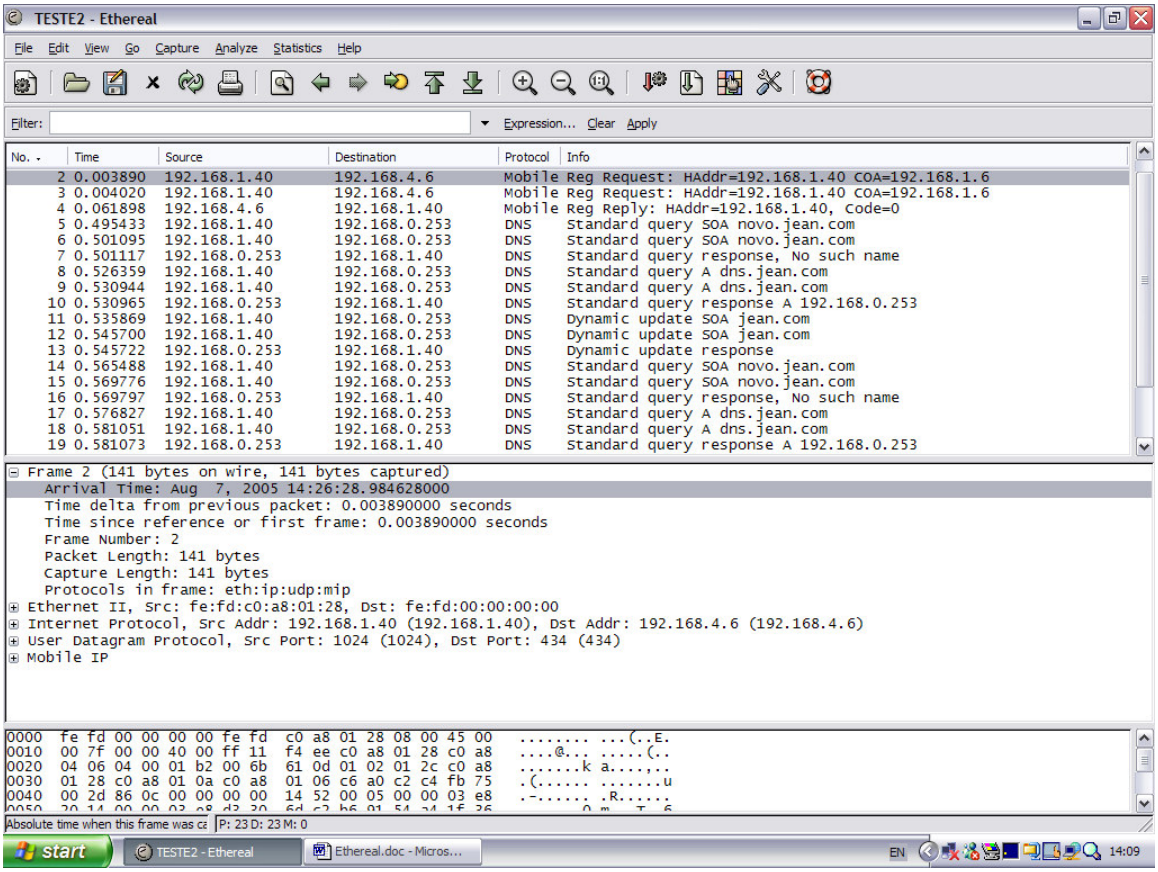

Figura 31 ***Registration Request***

No segundo quadro da figura, temos várias visões do pacote selecionado, visões estas relacionadas aos protocolos considerados (IP, UDP, IP Móvel, etc). Na linha sublinhada neste quadro, vemos o tempo de chegada do pacote à interface de rede, com o horário 14:26:28.

A figura 32 mostra o tempo do segundo pacote.

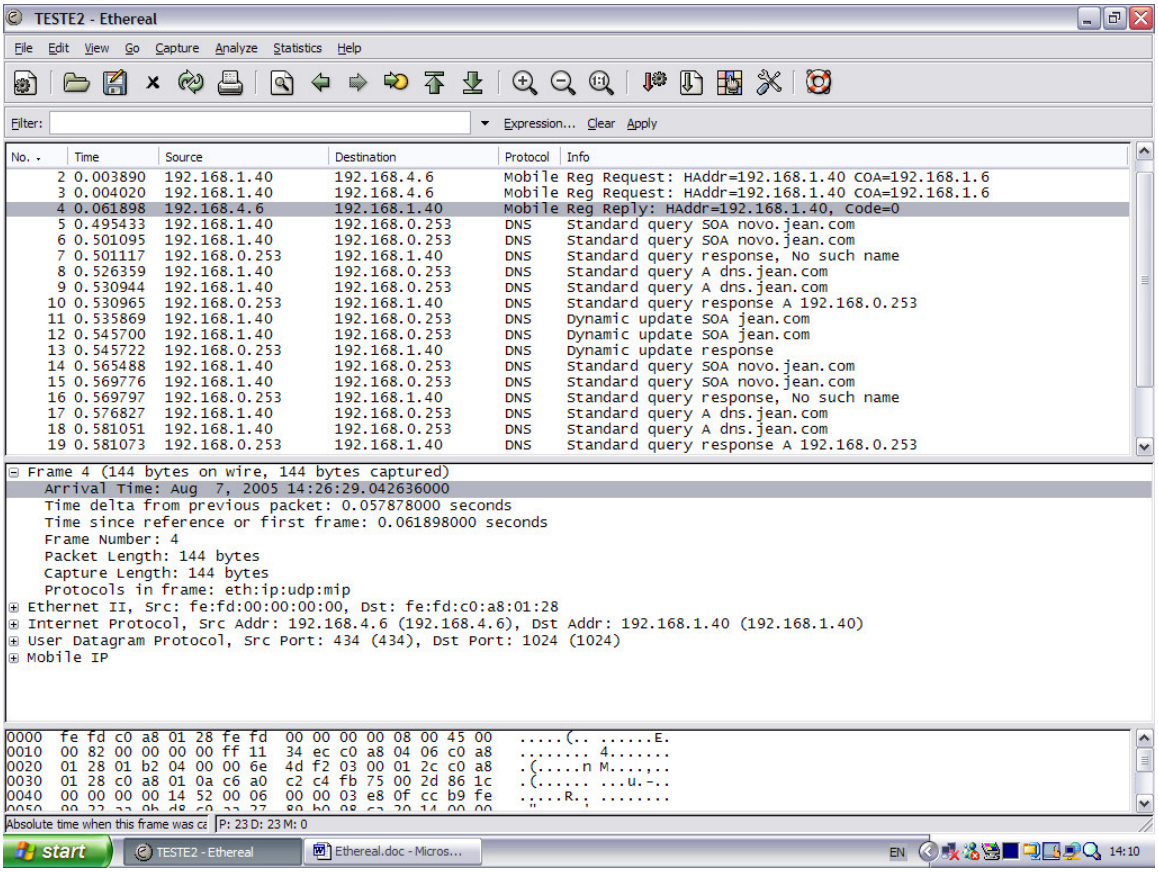

Figura 32 ***Registration Reply***

Na linha sublinhada do segundo quadro, vemos o tempo de chegada do pacote com a resposta ao registro do IP Móvel, com o horário 14:26:29. A figura 33 mostra o tempo do terceiro pacote.

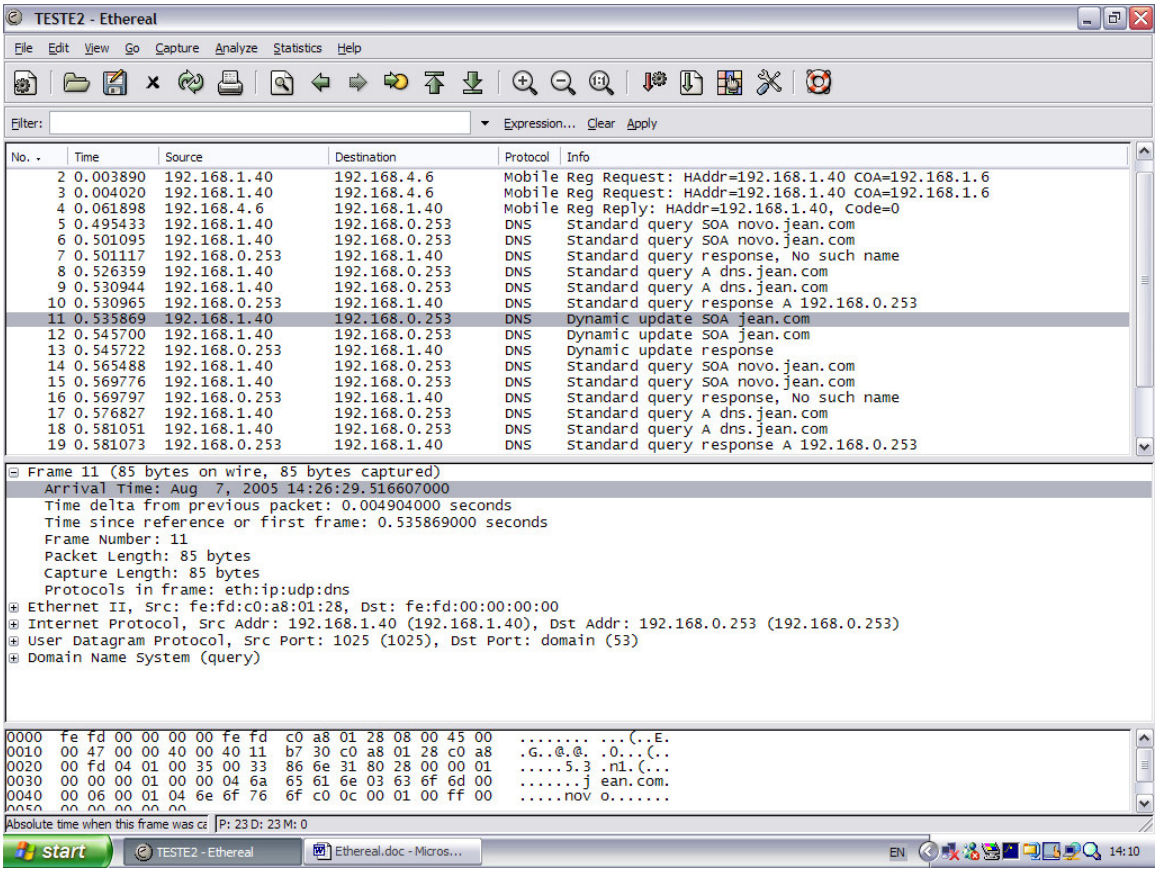

Figura 33 ***Dynamic update***

Na linha sublinhada do segundo quadro, vemos o tempo de chegada do pacote à interface de rede, com o horário 14:26:29. A figura 34 mostra o tempo do último pacote.

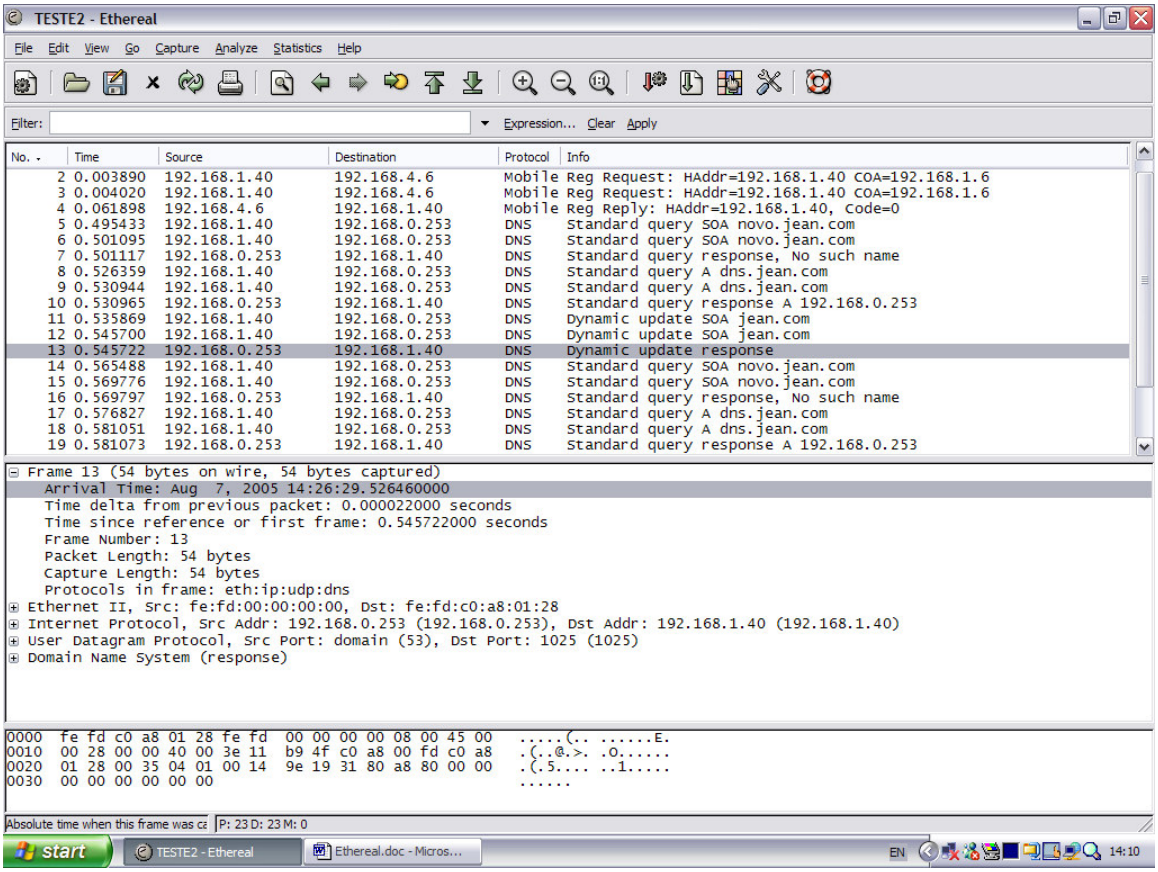

Figura 34 ***Dynamic response***

Na linha sublinhada do segundo quadro, vemos o tempo de chegada do pacote com a resposta à atualização dinâmica, com o horário 14:26:29. Com isto, temos que, o tempo total da operação é de aproximadamente 1 segundo.

Percebemos que, apesar de introduzirmos a atualização do DNS no coração do registro IP Móvel, o impacto no tempo foi considerado desprezível. Isto pôde ser comprovado nos testes, uma vez que o registro do IP Móvel sem a atualização dinâmica consumia um tempo superior a meio segundo, em alguns casos próximo de 1 segundo.

6.4) Avaliação de escalabilidade

Para a avaliação de escalabilidade, disparamos o processo do MN em quatro máquinas virtuais simultaneamente, para analisarmos como a arquitetura se comportaria com uma carga maior que um cliente por vez. Não foi possível realizar o teste simultâneo

com um número maior de máquinas virtuais, o que seria o ideal, devido aos recursos limitados da máquina hospedeira.

Devido a esta limitação, há que se considerar os resultados como podendo ter sido afetados pelo desempenho do *host*, uma vez que as máquinas virtuais compartilhavam os recursos da máquina hospedeira degradando seu desempenho. Na figura 35, vemos os pacotes capturados no primeiro cenário, onde a alteração do DNS é disparada pelo HA.

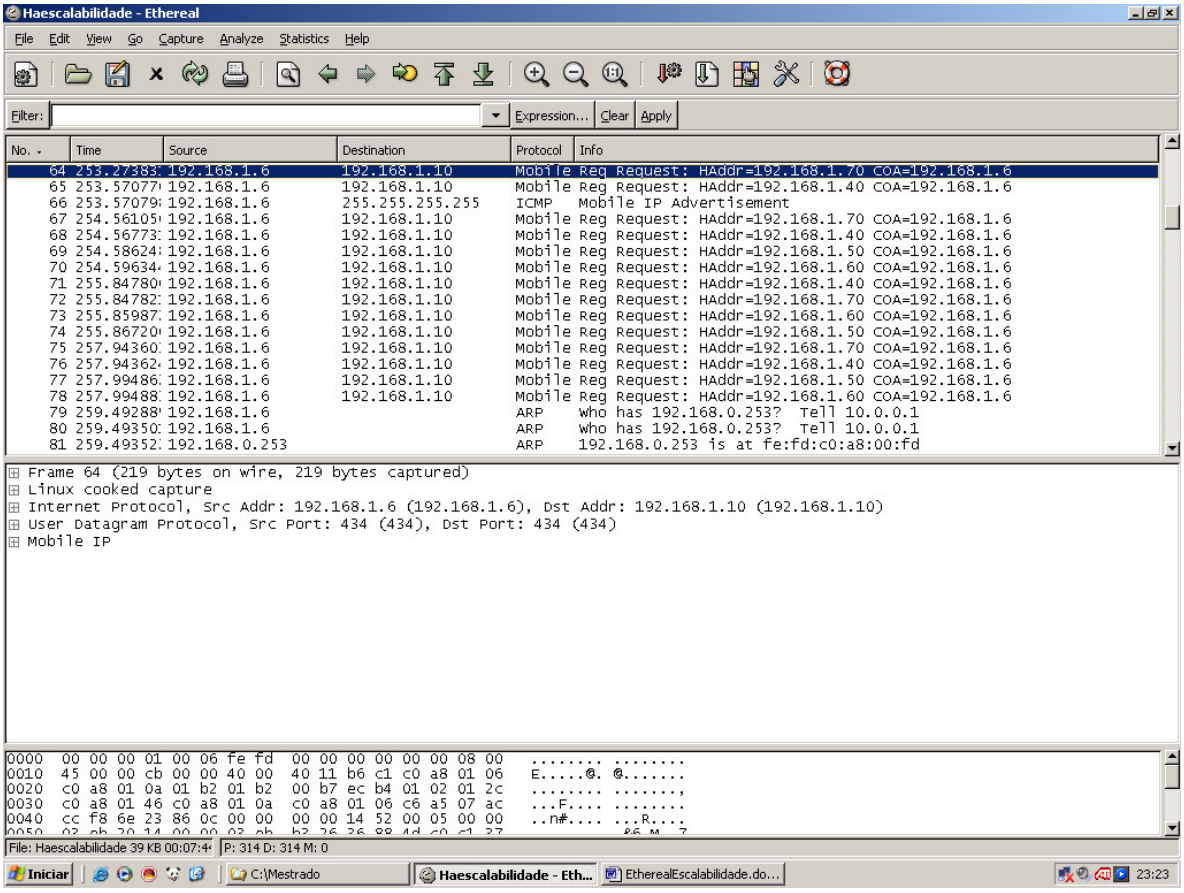

Figura 35 ***Primeiro cenário***

O primeiro pacote, na linha de número 64, e que está sublinhado, é o pacote da requisição de registro do primeiro MN. Aqui cabe ressaltar que, como a captura dos pacotes está sendo feita no HA, todas as requisições de registro aparecem com o mesmo endereço de origem.

Este endereço, 192.168.1.6, é o endereço do FA. Pelo mesmo motivo, todos os pacotes de requisição também têm o mesmo IP de destino, 192.168.1.10.

Este endereço é o do HA, isto é, da sua interface interna. O pacote da linha número 65, é o pacote da requisição de registro do segundo MN.

O pacote 69 é o pacote da requisição de registro do terceiro MN. Finalmente, temos o pacote 70, que é o pacote da requisição de registro do último MN.

Veremos agora, os tempos do primeiro e do último pacote, para podermos avaliar o impacto no tempo de resposta global. A figura 36 mostra o tempo do primeiro pacote.

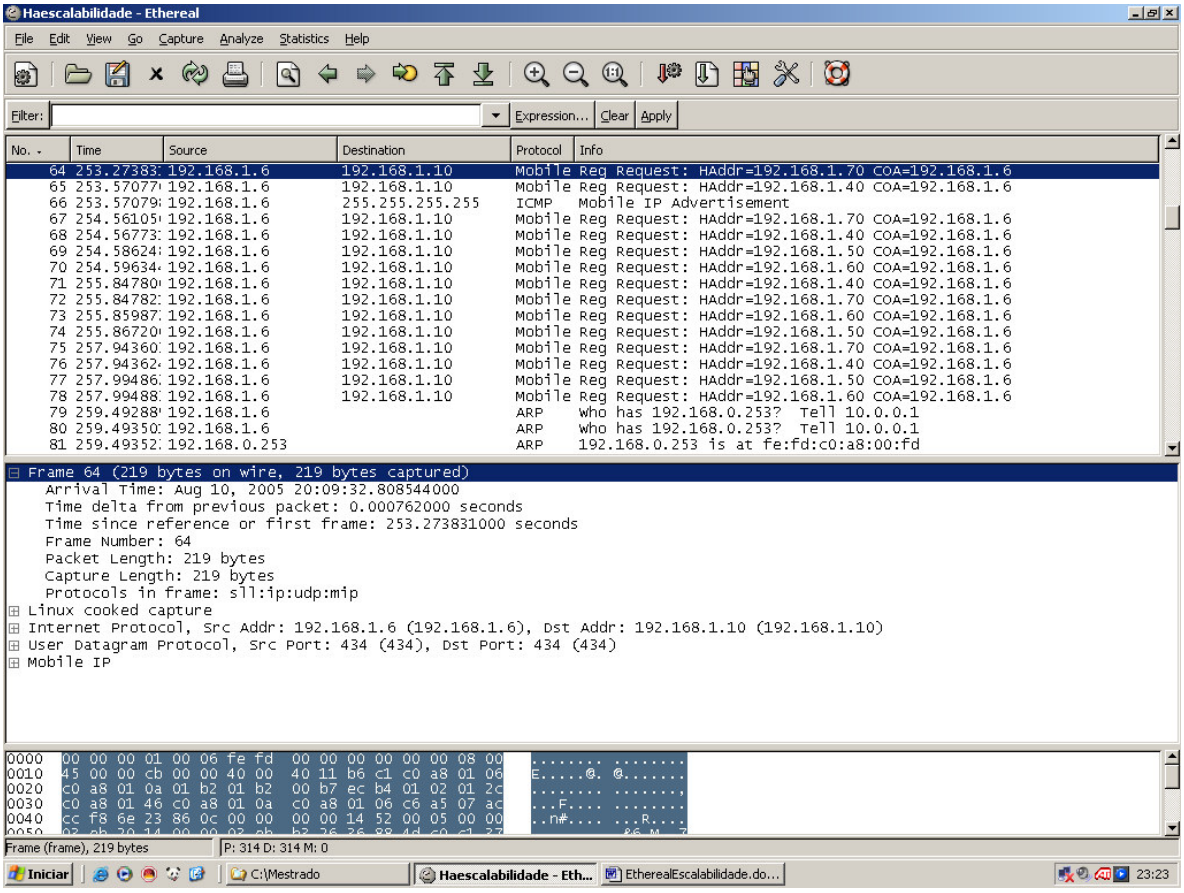

Figura 36 ***Primeiro cenário – primeiro pacote***

No segundo quadro da figura, temos várias visões do pacote selecionado, visões estas relacionadas às camadas consideradas (rede, transporte, aplicação, etc). Neste quadro, vemos o tempo de chegada do primeiro pacote, com o horário 20:09:32.

A figura 37 mostra o tempo do último pacote.

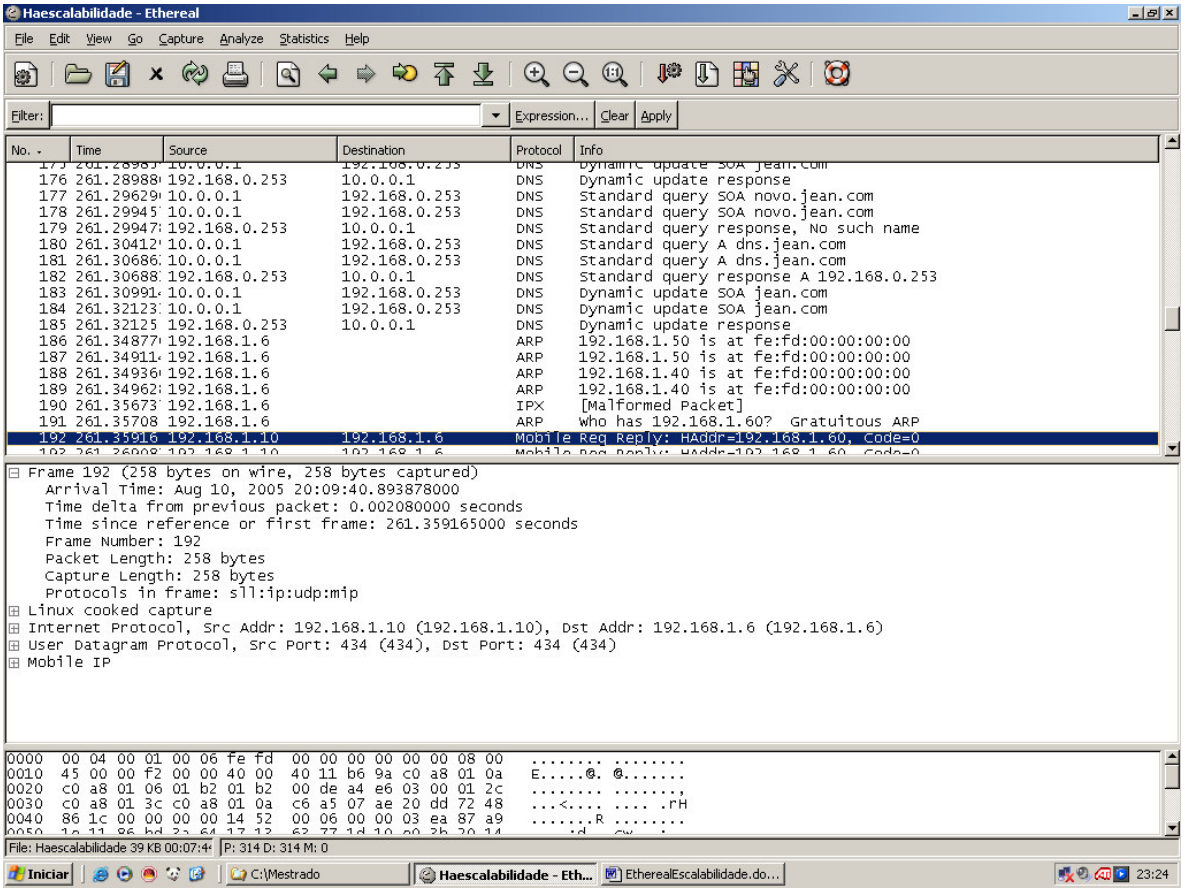

Figura 37 ***Primeiro cenário – último pacote***

Na linha com a informação *Arrival Time* deste quadro, vemos o tempo de chegada do pacote à interface de rede, com o horário 20:09:40. Com isto, temos que, o tempo total da operação é de aproximadamente 8 segundos.

Isto demonstra um impacto considerável, uma vez que uma atualização isolada demorou em média 1 segundo, e o tempo total para quatro atualizações simultâneas é bem superior à soma dos tempos individuais. Esta avaliação é ainda mais negativa, quando levamos em conta que o teste foi feito com apenas quatro clientes simultâneos, sendo que um cenário real poderia ter um número bem superior de requisições a serem manejadas.

Vamos, agora, ao segundo cenário. Na figura 38, vemos os pacotes capturados no cenário onde a alteração do DNS é disparada pelo cliente.

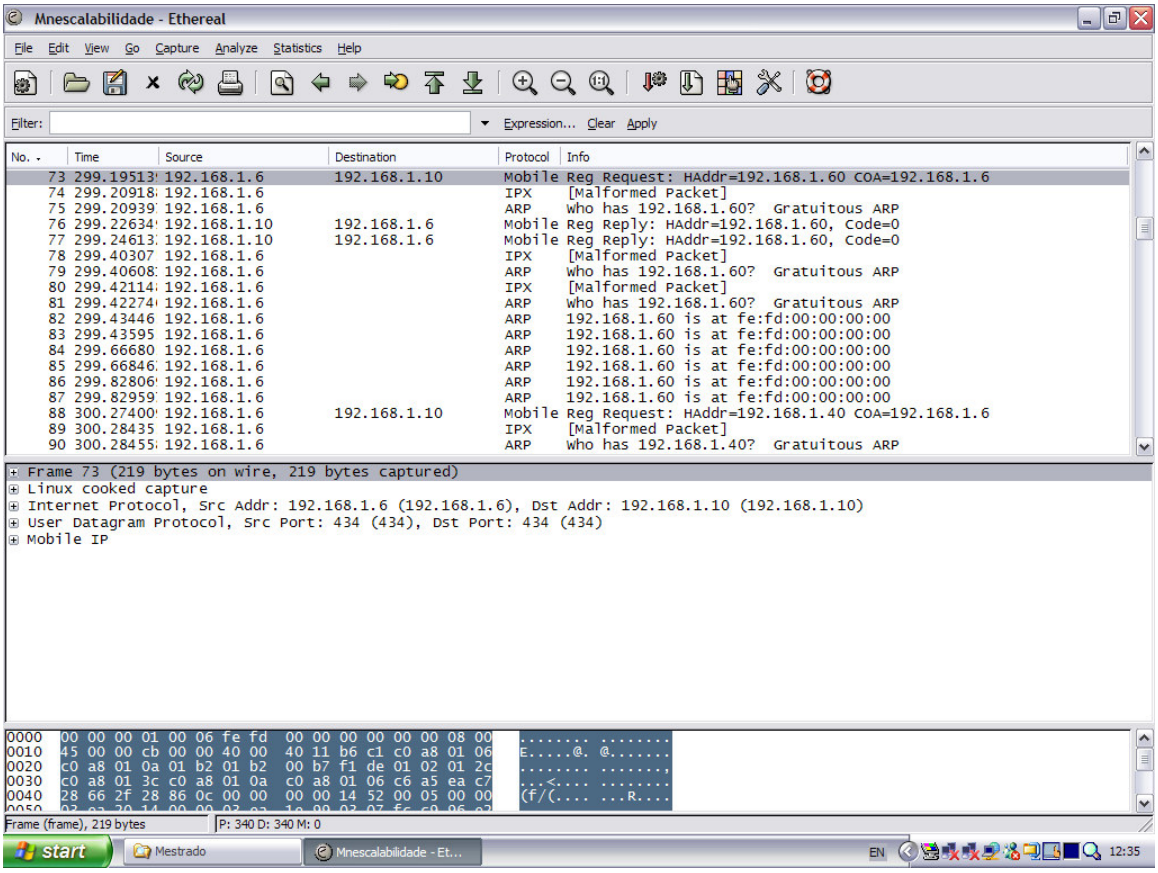

Figura 38 *Segundo cenário*

O primeiro pacote, na linha de número 73, e que está sublinhado, é o pacote da requisição de registro do primeiro MN. O pacote da linha número 76, cuja fonte é o endereço 192.168.1.10 (HA), é o pacote com a resposta do registro no IP Móvel do primeiro MN.

O pacote 88, cujo destino é o endereço 192.168.1.10 (HA), é o pacote da requisição de registro do segundo MN. E assim, sucessivamente, para os outros pacotes que não aparecem na imagem.

Veremos agora, os tempos do primeiro e do último pacote, para podermos avaliar o impacto no tempo de resposta global. A figura 39 mostra o tempo do primeiro pacote.

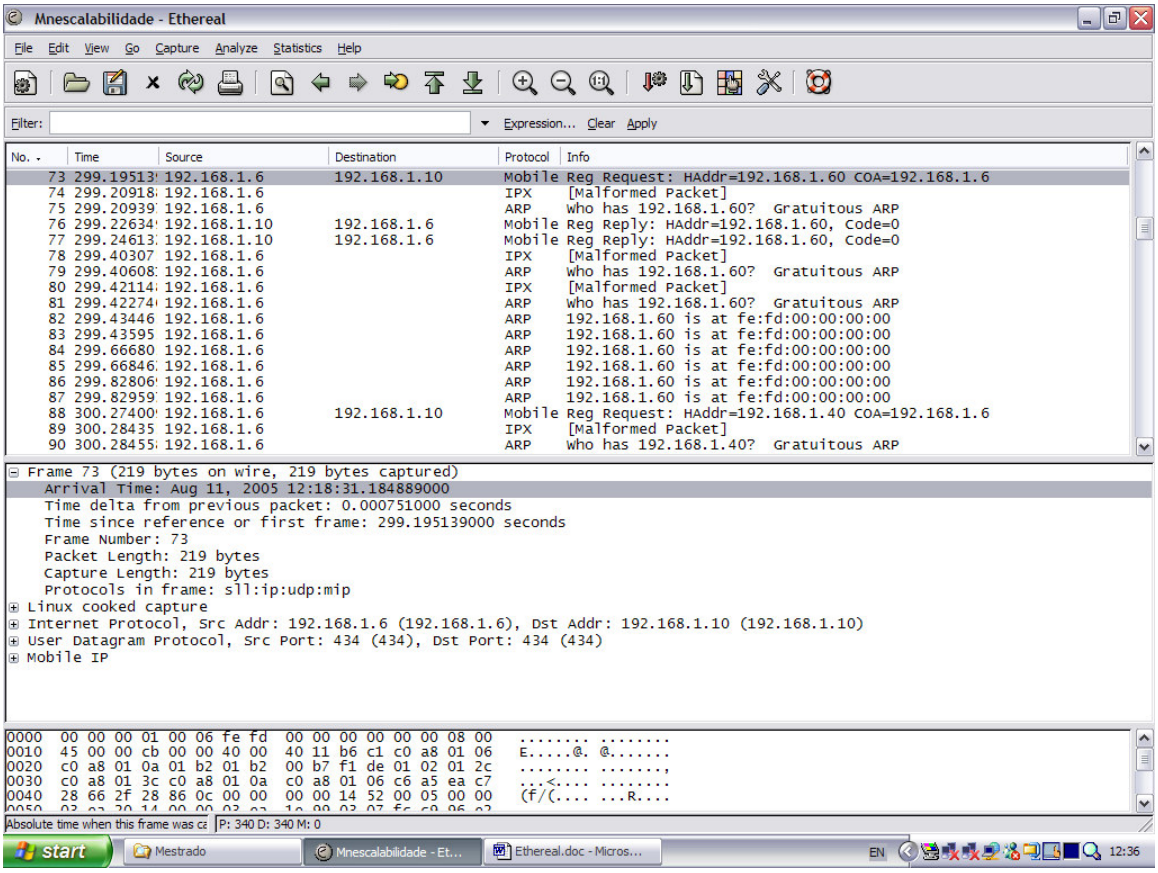

Figura 39 ***Segundo cenário – primeiro pacote***

No segundo quadro da figura, temos várias visões do pacote selecionado, visões estas relacionadas aos protocolos considerados (IP, UDP, IP Móvel, etc). Na linha sublinhada neste quadro, vemos o tempo de chegada do primeiro pacote, com o horário 12:18:31.

A figura 40 mostra o tempo do último pacote.

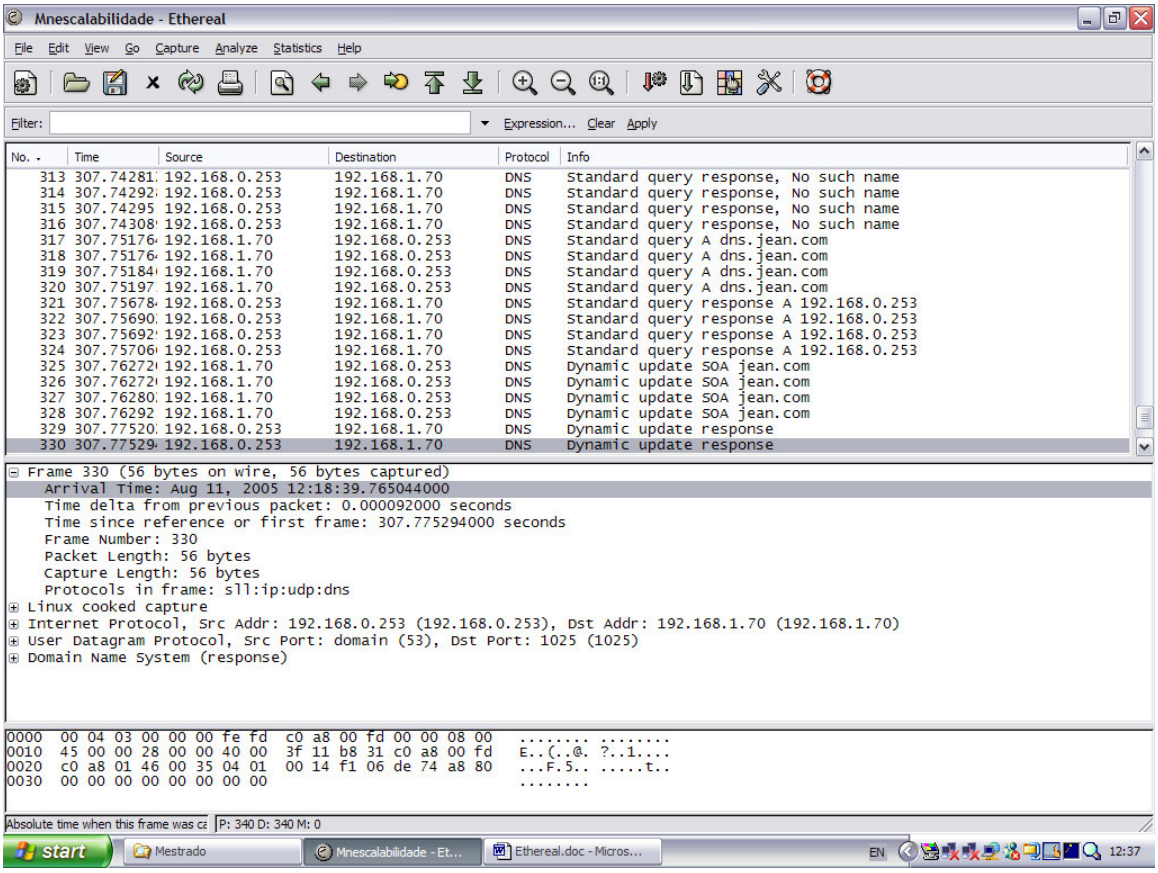

Figura 40 ***Segundo cenário – último pacote***

Na linha sublinhada do segundo quadro, vemos o tempo de chegada do pacote com a resposta à atualização dinâmica, com o horário 12:18:39. Com isto, temos que, o tempo total da operação é de aproximadamente 8 segundos.

Este cenário também demonstra um impacto considerável, uma vez que uma atualização isolada demorou em média 1 segundo, e o tempo total para quatro atualizações simultâneas é bem superior à soma dos tempos individuais. Devemos considerar que além da sobrecarga simultânea sobre o HA e o DNS (que é o que se buscou mensurar), o desempenho da máquina hospedeira pode ter comprometido os resultados.

6.5) Considerações finais

O objetivo desse capítulo foi apresentar a avaliação dos testes do mecanismo de nomes para suportar macro-mobilidade com IP Móvel. O ambiente UML foi produzido

utilizando-se um *kernel* compilado especificamente para este fim, e a comunicação entre as máquinas virtuais foi realizada através de *multicast,* configurando-se no mesmo domínio *multicast* as interfaces que simulam estar em uma mesma rede física.

O tempo de atualização foi obtido através do *log* das mensagens e, a escalabilidade, foi analisada a partir de um exercício de simulação. Foi feita, também, uma avaliação dos resultados do mecanismo.

Os resultados demonstraram que o impacto no tempo de resposta para uma operação isolada foi desprezível em ambos os cenários avaliados, mas também que a escalabilidade ficou comprometida quando operações simultâneas foram submetidas, mais uma vez para ambos os cenários. Outra conclusão é que, são necessários testes utilizando um servidor com mais recursos de *hardware,* sendo também interessante a execução de testes em uma rede física real.

Capítulo 7 – Conclusão e trabalhos futuros

Um dos problemas encontrados na macro-mobilidade é a dificuldade de manter um identificador único para o usuário quando este comuta entre redes administradas independentemente. Nosso desafio foi propor um mecanismo de nomes unificado para suporte a macro-mobilidade em redes móveis com tecnologias heterogêneas, através de alterações na implementação de entidades relacionadas ao serviço IP Móvel.

Foi apresentada uma visão geral da proposta definindo duas estratégias para sua implementação: atualização de nomes iniciada pelo HA e atualização de nomes iniciada pelo cliente. Também foram detalhados estes mecanismos, e comparadas as duas estratégias, indicando suas vantagens e desvantagens relativas.

Apresentamos - em anexo - os principais pacotes utilizados para implementação, e – no capítulo anterior – a avaliação de resultados. A implementação foi baseada no pacote *Dynamics - HUT Mobile IP,* como podemos ver no Anexo B, pacote *open source* disponível no *Source Forge*.

A avaliação dos resultados foi feita de forma quantitativa e qualitativa. Os resultados demonstraram que o impacto no tempo de resposta para uma operação isolada foi desprezível em ambos os cenários avaliados, mas também que a escalabilidade ficou comprometida quando operações simultâneas foram submetidas, mais uma vez para ambos os cenários.

Devemos salientar que, o impacto negativo na avaliação de escalabilidade pode dever-se ao ambiente simulado em UML, uma vez que as várias máquinas virtuais compartilhavam os recursos físicos da máquina hospedeira. Testes com um servidor dotado de mais recursos de *hardware* seriam interessantes, para tentar minimizar os efeitos desta limitação, além de testes com uma rede física real.

As maiores contribuições que esta proposta pretendeu dar são: do ponto de vista teórico, a utilização de um mecanismo de nomes para resolver o problema da rastreabilidade do nó móvel quando ele se conecta a redes administradas por entidades independentes; e, do ponto de vista prático, foram descritos os vários elementos que precisaram ser integrados a fim de implementar a estratégia de forma segura e confiável.

Propomos, como trabalho futuro, a integração com IPv6.

Esta integração será importante porque, o objetivo primordial deste projeto foi propor um mecanismo de nomes unificado para *macro-mobilidade* em redes móveis com tecnologias heterogêneas, como WLAN e GPRS, utilizando IPv4 e, mapeando uma entrada DNS do *host* a cada mudança de rede. Mas, no nosso trabalho, assumimos que o endereço IP atribuído ao nó móvel é público.

O problema de escassez de endereços é um aspecto que será abordado em desenvolvimentos futuros deste projeto. Por isso, a integração com o MIPv6 e com as entradas DNS no formato IPv6 é um caminho promissor, pois pode representar papel decisivo na solução do problema da escassez de endereços.

Referências

[1] C. Perkins, ed., "IP Mobility Support", IETF RFC 2002, Oct. 1996.

[2] GSM Association. An Overview of GPRS (General Packet Radio Service). http://www.gsmworld.com/technology/gprs.html

[3] CDMA Digital Wireless Technology
http://www.qualcomm.com/cdma/

[4] N. Banerjee, W. Wu, S. K. Das, S. Dawkins, J. Pathak, "Mobility Support in Wireless Intenet", IEEE Wireless Communications, Oct. 2003.

[5] R. Ramjee, T. La Porta, S. Thuel, K. Varadhan, e S. Y. Wang, "HAWAII: A Domain-based Approach for Supporting Mobility in Wide-area Wireless Networks", IEEE Pers. Commun., vol. 7, n. 4, Aug. 2000.

[6] C-Y. Wan, A. T. Campbell, e A. G. Valko, "Design, Implementation, and Evaluation of Cellular IP", IEEE Pers. Commun., vol. 7, n. 4, Aug. 2000, pp. 42-49.

[7] A. Grilo, P. Estrela, e M. Nunes, "Terminal Independent Mobility for IP (TIMIP)", IEEE Commun. Mag., Dec. 2001, pp. 34-41.

[8] S. Das *et al.*, "IDMP: An Intra-Domain Mobility Management Protocol for Next-Generation Wireless Networks", IEEE Wireless Commun., vol. 9, n. 3, Jun. 2003, pp. 38-45.

[9] C. Perkins Ed., "IP Mobility Support for IPv4", RFC 3344, Aug. 2002.

[10] M. Handley et al., "SIP: Session Initiation Protocol", IETF RFC 2543, Mar. 1999.

[11] A. C. Snoeren, H. Balakrishnan, "An End-to-End Approach to Host Mobility", Proc 6th Int'l. Conf. Mobile Comp. And Net., Boston, MA, Aug. 2000.

[12] D. Wisely, P. Eardley, L. Burness, "IP on 3G Networking Technologies for Mobile Communications", ed. John Wiley & Sons, 2002.

[13] Willian de Araújo Sales, "Uma Entidade Funcional para Autenticação de Dispositivos Móveis entre Áreas de Micromobilidade", disponível em http://great.lia.ufc.br/Teses/dissertacaoWilliamSales.pdf, acessado em abril 2005.

[14] J.B. Postel, ed., "Internet Protocol", IETF RFC 791, Sept. 1981.

[15] WAP Forum Technical Specifications. Wireless Application Protocol (WAP).
http://www.wapforum.org/what/technical.htm

[16] Windows CE. Windows CE web site.
http://www.microsoft.com/windows/embedded/ce

[17] Sun Microsystems. Java™ 2 Platform, Micro Edition.
http://java.sun.com/j2me/

[18] J. Manner *et al.*, "Evaluation of Mobility and Quality of Service Interaction", Comp. Networks, vol. 38, 2002, pp. 137-163.

[19] W. Ma e Y. Fang, "Dynamic Hierarchical Mobility Management Strategy for Mobile IP Networks", IEEE Commun. Mag., May 2004, pp. 664-676.

[20] R. Caceres e V. N. Padmanabhan, "Fast and scalable handoffs for wireless internetworks", in Proc. ACM MOBICOM'96, 1996, pp. 56-66.

[21] A. T. Campbell, J. Gomez, e A. G. Valko, “An overview of cellular IP”, Proc. IEEE Wireless Communications and Networking Conf., New Orleans, LA, Sep. 1999, pp. 606-610.

[22] M. Georgiades, K. Chew, C. Politis, e R. Tafazolli, “Context Transfer Extension to Mobility Protocols for All-IP Based Infrastructures”, Wireless World Research Forum (WWRF), 9th Meeting, Jul. 2003.

[23] T. Dagiuklas, D. Theofilatos, e D. Gatzounas, “Supporting Multimedia Services for Mobile Users in All-IP Based Networks: Requirements, Functions and Issues”, Wireless World Research Forum (WWRF), 6th Meeting, Jun. 2002.

[24] C. Castelluccia, “Hierarchical MIPv6 mobility management (HMIPv6)”, Internet Draft, Internet Engineering Task Force, draft-ietf-mobileip-hmipv6-06.txt.

[25] A. Campbell, “Cellular IP”, Internet Draft, Internet Engineering Task Force, Oct. 1999.

[26] John Williams Floroiu, Reinhard Ruppelt, Dorgham Sisalem, Jerome Voglimacci Stephanopoli, “Seamless Handover in Terrestrial Radio Access Networks: A Case Study”, IEEE Communications Magazine, Nov. 2003, pp. 110 – 116

[27] Milind M. Buddhikot, Girish Chandranmenon, Seungjae Han, Yui-Wah Lee, Scott Miller, Luca Salgarelli, “Design and Implementation of a WLAN / CDMA2000 Interworking Architecture”, IEEE Communications Magazine, Nov. 2003, pp. 90 – 100

[28] Wellington B., “Secure Domain Name System (DNS) Dynamic Update”, RFC 3007, Nov. 2000

[29] H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed – Hashing for Message Authentication”, IETF RFC 2104, Fev. 1997.

[30] R. Rivest, “The MD5 Message – Digest Algorithm”, IETF RFC 1321, Abr. 1992.

Anexo A – Função *handle_reg_msg*

É nesta função que fizemos as alterações de código necessárias para o mecanismo de atualização baseado no HA, e a listagem apresentada a seguir, mostra o seu código fonte antes das alterações.

```
/**
 * handle_reg_msg:
 * @s: registration message socket (UDP)
 *
 * Processes a registration request message that has arrived on socket @s.
 *
 * Returns:
 *   0 on success or -1 on failure
 */
static int
handle_reg_msg(struct interface_entry *from_iface, int s)
{
        char msg[MAXMSG];
        struct sockaddr_in cli_addr;
        struct in_addr dst_addr;
        struct msghdr mh;
        char cdata[256];
        struct iovec iov;
        struct cmsghdr *cmsg;
        struct bindingentry *binding;
        struct spi_entry *mn_spi;
        int n, res, rsock;
        struct msg_extensions ext;
        int killbinding;
        int code, auth_type;
        char mn_addrstr[17];
        char fa_addrstr[17];
        struct ha_tunnel_data *t_data = NULL;
        struct bindingkey bkey;
        struct node *node;
        struct interface_entry *force_iface = NULL;

        memset(&cli_addr, 0, sizeof(cli_addr));
        dst_addr.s_addr = 0;
        memset(&mh, 0, sizeof(mh));
        iov.iov_base = msg;
        iov.iov_len = MAXMSG;
        mh.msg_name = &cli_addr;
        mh.msg_namelen = sizeof(cli_addr);
        mh.msg_iov = &iov;
        mh.msg_iovlen = 1;
        mh.msg_control = &cdata;
        mh.msg_controllen = sizeof(cdata);

        n = recvmsg(s, &mh, 0);

        DEBUG(DEBUG_FLAG, "Received %d bytes from %s:%d\n", n,
            inet_ntoa(cli_addr.sin_addr), ntohs(cli_addr.sin_port));

        if (n < 0) {
                LOG2(LOG_ERR, "handle_reg_msg - recvfrom failed - %s\n",
                    strerror(errno));
                return -1;
        }

        for (cmsg = CMSG_FIRSTHDR(&mh); cmsg != NULL;
```

```
    cmsg = DYNAMICS_CMSG_NXTHDR(&mh, cmsg)) {
            if (cmsg->cmsg_level == SOL_IP &&
                cmsg->cmsg_type == IP_PKTINFO) {
                        struct _in_pktinfo *pkt;
                        pkt = (struct _in_pktinfo *) CMSG_DATA(cmsg);
                        dst_addr = pkt->ipi_addr;
                        DEBUG(DEBUG_FLAG, "\tIP_PKTINFO: ipi_ifindex=%i "
                            "ipi_spec_dst=%s ", pkt->ipi_ifindex,
                            inet_ntoa(pkt->ipi_spec_dst));
                        DEBUG(DEBUG_FLAG, "ipi_addr=%s\n",
                            inet_ntoa(pkt->ipi_addr));
            }
}

/* use the matching UDP socket for the registration reply */
rsock = -1;
for (node = list_get_first(&config.interfaces); node != NULL;
    node = list_get_next(node)) {
            struct interface_entry *iface =
                        (struct interface_entry *) node;
            if (iface->addr.s_addr == dst_addr.s_addr ||
                        (iface->ha_disc &&
                         iface->bcaddr.s_addr == dst_addr.s_addr)) {
                        rsock = iface->udp_sock;
            }
}
if (rsock < 0 && from_iface != NULL &&
   dst_addr.s_addr == (unsigned int) -1) {
            DEBUG(DEBUG_FLAG, "Broadcast 255.255.255.255 destination - "
                "using interface[%s] UDP socket[%i]\n", from_iface->dev,
                from_iface->udp_sock);
            rsock = from_iface->udp_sock;
}
if (rsock < 0) {
            LOG2(LOG_WARNING, "Could not find UDP socket for "
                "registration reply - trying recv. socket\n");
            rsock = s;
}

res = parse_msg(msg, n, &ext);

if (ext.req != NULL && ntohs(ext.req->lifetime) == 0 &&
   ext.req->home_addr.s_addr == cli_addr.sin_addr.s_addr &&
   ext.req->home_addr.s_addr == ext.req->co_addr.s_addr) {
            /* RFC 2002, Ch. 3.8.3.1:
             * When HA is replying to deregistration (lifetime == 0 &&
             * COA == home addr) whose srcIP is MN's home address, the
             * reply must be send directly to home network bypassing any
             * mobility bindings */
            DEBUG(DEBUG_FLAG, "Dereg. from home => bypassing mobility "
                "bindings in routing\n");
            force_iface = from_iface;
}

if (res == -3 || res == -4) {
            LOG2(LOG_WARNING, "Unknown critical vendor extension in a "
                "request from %s:%i\n", inet_ntoa(cli_addr.sin_addr),
                ntohs(cli_addr.sin_port));
            send_reg_failure(rsock, &cli_addr, NULL, 0,
                                    res == -3 ? REGREP_UNSUPP_VENDOR_ID_MN_HA :
                                    REGREP_UNSUPP_VENDOR_ID_FA_HA, NULL, &ext,
                                    force_iface, NULL, from_iface);
            stats.discarded_vendor_ext++;
            report_discarded_msg(msg, n, &cli_addr,
                                        "unknown vendor extension");
            return -1;
} else if (res != 0 || ext.req == NULL) {
            char *reason = "N/A";
            if (res == -1) {
                        reason = "unknown extension";
```

```
                stats.discarded_unknown_ext++;
        } else if (res == -2) {
                reason = "malformed message";
                stats.discarded_malformed_msg++;
        } else if (ext.req == NULL) {
                reason = "not a request";
                stats.discarded_not_request++;
        }
        LOG2(LOG_WARNING, "Message parser failed or not a request "
            "(from %s:%i): %s\n", inet_ntoa(cli_addr.sin_addr),
            ntohs(cli_addr.sin_port), reason);
        report_discarded_msg(msg, n, &cli_addr, reason);
        return -1;
}

memcpy(&bkey.mn_addr, &ext.req->home_addr, sizeof(bkey.mn_addr));
bkey.ha_addr = config.sha_addr.s_addr != 0 ? config.sha_addr :
        ext.req->ha_addr;
bkey.priv_ha = config.priv_ha;
binding = binding_fetch(bindings, &bkey);
if (binding != NULL)
        t_data = (struct ha_tunnel_data *) binding->data;
mn_spi = validate_request(binding, cli_addr.sin_addr, msg, n, &ext,
                                &code, &auth_type);
if (code > 2) {
        send_reg_failure(rsock, &cli_addr, mn_spi, 0, code,
                                (t_data != NULL ? &t_data->nonce : NULL),
                                &ext, force_iface, binding, from_iface);
        stats.req_rejected++;
        report_discarded_msg(msg, n, &cli_addr, "invalid message");
        return -1;
}
if (mn_spi == NULL) {
        stats.req_rejected++;
        report_discarded_msg(msg, n, &cli_addr, "MN SPI not found");
        return -1;
}

dynamics_strlcpy(mn_addrstr, inet_ntoa(ext.req->home_addr),
                sizeof(mn_addrstr));
dynamics_strlcpy(fa_addrstr, inet_ntoa(ext.req->co_addr),
                sizeof(fa_addrstr));

if (binding != NULL) {
        t_data->auth_type = (auth_type == AUTH_MAC_RFC2002 ?
                                AUTH_RFC2002BIS : AUTH_RFC2002);

        /* If the id field is identical the message was a duplicate.
         * In order to avoid breaking the tunnel drop this message as
         * a special case without sending a failure reply if the replay
         * protection is used
         */
        if (mn_spi->replay_method != REPLAY_PROTECTION_NONE &&
            binding->id[0] == ext.req->id[0] &&
            binding->id[1] == ext.req->id[1]) {
                DEBUG(DEBUG_FLAG,
                    "Received duplicate request - dropping it\n");
                stats.req_rejected++;
                return -1;
        }

        /* Make sure that the id field is incremented if the timestamp
         * protection is used */
        if (mn_spi->replay_method == REPLAY_PROTECTION_TIMESTAMP &&
            (ntohl(binding->id[0]) > ntohl(ext.req->id[0]) ||
             (ntohl(binding->id[0]) == ntohl(ext.req->id[0]) &&
              ntohl(binding->id[1]) >= ntohl(ext.req->id[1])))) {
                LOG2(LOG_NOTICE, "Timestamp did not increase "
                    "(MN=%s, FA=%s)\n", mn_addrstr, fa_addrstr);
                send_reg_failure(rsock, &cli_addr, mn_spi,
```

```
                                        t_data->auth_type,
                                        REGREP_ID_MISMATCH_HA,
                                        &t_data->nonce, &ext, force_iface,
                                        binding, from_iface);
                        stats.req_rejected++;
                        return -1;
                }

                /* remove binding from list (does not remove tunnels etc.) */
                binding_remove(binding);
        } else {
                LOG2(LOG_NOTICE, "MN %s registers, FA is %s\n",
                    mn_addrstr, fa_addrstr);
        }

        stats.req_accepted++;

        killbinding = 0;

        if (binding &&
           (binding->lower_addr.s_addr != (config.sha_addr.s_addr != 0 ?
                                        config.sha_addr.s_addr :
                                        ext.req->co_addr.s_addr) ||
            ext.req->lifetime == 0)) {
                DEBUG(DEBUG_FLAG, "Found existing binding for MN %s\n",
                     mn_addrstr);

                if (ext.req->co_addr.s_addr == ext.req->home_addr.s_addr &&
                   ext.req->lifetime == 0) {
                        DEBUG(DEBUG_FLAG, "MN is deregistering all its "
                             "simultaneous bindings\n");
                        /* Note: RFC 2002, 3.6.1.2, special case
                         * This should be taken into account if the
                         * simultaneous bindings are implemented.
                         * At the moment we just deregister the only possible
                         * binding below. */
                }

#ifdef USE_TEARDOWN
                /* if co_addr differs in request, send reply to old
                 * co_addr to purge old tunnel segment */
                if (config.sha_addr.s_addr == 0 &&
                   binding->lower_addr.s_addr != ext.req->co_addr.s_addr) {
                        binding->timeout = 0;
                        DEBUG(DEBUG_FLAG, "Sending purge msg to %s\n",
                             inet_ntoa(binding->lower_addr));
                        send_reg_repl(s, binding, mn_spi, NULL, force_iface,
                                      REGREP_ACCEPTED);
                }
#endif
```

```
        if (ext.req->lifetime > 0) {
                /* it was a location update */
                LOG2(LOG_INFO, "MN %s updating tunnel to FA %s\n",
                    mn_addrstr, fa_addrstr);
                if (switch_tunnel(binding, &ext, mn_spi) < 0)
                        return -1;
        } else {
                /* if it is a deregistration request,
                 * destroy the binding after the reply is sent,
                 * since the binding is needed when sending the
                 * reply */
                LOG2(LOG_NOTICE, "MN %s deregisters\n", mn_addrstr);
                remove_tunnel(binding);
                if (cli_addr.sin_addr.s_addr == binding->mn_addr.s_addr
                   && ext.req->co_addr.s_addr ==
                   ext.req->home_addr.s_addr) {
                        DEBUG(DEBUG_FLAG, "MN returned home and "
                            "deregistered all care-of addresses\n");
                        /* FIX: also HA should send gratuitous ARP
                         * telling that MN takes care of its own
                         * packets from now on;
                         * draft-ietf-mobileip-rfc2002-bis-03.txt, 4.6;
                         * This would require that HA would get the
                         * link-layer address of the MN from the
                         * deregistration message and use it in the
                         * gratuitous ARP packet */
                }
                killbinding = 1;
        }
}

if (binding == NULL) {
        if (ntohs(ext.req->lifetime) == 0) {
                DEBUG(DEBUG_FLAG, "Deregistration attempt, but binding"
                    " not found - creating temporary binding\n");
                binding = create_binding(&ext, mn_spi, 0);
                killbinding = 1;
        } else {
                binding = create_binding(&ext, mn_spi, 1);
        }

        if (binding == NULL) {
                LOG2(LOG_ERR, "Failed to create binding for MN: %s, "
                    "FA: %s\n", mn_addrstr, fa_addrstr);
                send_reg_failure(rsock, &cli_addr, mn_spi,
                                 (auth_type == AUTH_MAC_RFC2002 ?
                                  1 : 0),
```

```
                                        REGREP_NO_RESOURCES_HA, NULL, &ext,
                                        force_iface, binding, from_iface);
                return -1;
        }
        t_data = (struct ha_tunnel_data *) binding->data;
        t_data->auth_type = (auth_type == AUTH_MAC_RFC2002 ?
                                AUTH_RFC2002BIS : AUTH_RFC2002);
}
ASSERT(binding != NULL);
t_data = (struct ha_tunnel_data *) binding->data;
ASSERT(t_data != NULL);

binding->mod_time = time(NULL);
t_data->reverse_tunnel = (ext.req->opts & REGREQ_REVERSE_TUNNEL) != 0;
if (ext.req->opts & REGREQ_MINIMAL_ENCAPS)
        t_data->encapsulation = ENCAPS_MINIMAL;
else if (ext.req->opts & REGREQ_GRE_ENCAPS)
        t_data->encapsulation = ENCAPS_GRE;
else
        t_data->encapsulation = ENCAPS_IPIP;

if (ext.fa_keyreq)
        binding->fa_spi = ntohl(ext.fa_keyreq->spi);
else
        binding->fa_spi = 0;

/* If the request has a public key add it to the binding */
if (ext.fa_pubkey) {
        /* binding->fa_pubkey is NULL if we have generated a
         * new SK. If the public key sent by the FA is the same
         * as in a previous binding, we can reuse the encrypted
         * SK and save some CPU cycles */
        if (binding->fa_pubkey &&
           binding->last_sent_fa_pubkeyrep != NULL &&
           (memcmp(binding->fa_pubkey,
                   ext.fa_pubkey,
                   GET_KEY_EXT_LEN(ext.fa_pubkey)) == 0))
        {
                DEBUG(DEBUG_FLAG, "Reusing encrypted SK for FA %s\n",
                    fa_addrstr);
        } else {
                if (binding->fa_pubkey)
                        free(binding->fa_pubkey);
                if (binding->last_sent_fa_pubkeyrep)
                        free(binding->last_sent_fa_pubkeyrep);
                binding->last_sent_fa_pubkeyrep = NULL;
                binding->fa_pubkey = (struct msg_key *)
```

```
                        malloc(GET_KEY_EXT_LEN(ext.fa_pubkey));
                if (binding->fa_pubkey != NULL) {
                        memcpy(binding->fa_pubkey, ext.fa_pubkey,
                            GET_KEY_EXT_LEN(ext.fa_pubkey));
                } else {
                        LOG2(LOG_ERR, "Not enough memory "
                            "for fa_pubkey\n");
                        return -1;
                }
                binding->last_sent_fa_pubkeyrep =
                        dynamics_do_rsa_encrypt(binding->key,
                                                binding->keylen,
                                                binding->fa_pubkey);
                if (binding->last_sent_fa_pubkeyrep == NULL) {
                        LOG2(LOG_ERR, "RSA encryption failed (MN=%s, "
                            "FA=%s)\n", mn_addrstr, fa_addrstr);
                }
        }

} else {
        /* remove the (possible) old FA pubkey from the binding to make
         * sure that the next FA using RSA will get the correct key */
        if (binding->fa_pubkey)
                free(binding->fa_pubkey);
        binding->fa_pubkey = NULL;

        if (binding->last_sent_fa_pubkeyrep)
                free(binding->last_sent_fa_pubkeyrep);
        binding->last_sent_fa_pubkeyrep = NULL;
}

memcpy(&binding->id, ext.req->id, REG_REQ_ID_LEN);

/* store where the request came from */
t_data->lower_saddr.s_addr = cli_addr.sin_addr.s_addr;
binding->lower_port = cli_addr.sin_port;

binding->timeout = MIN(ntohs(ext.req->lifetime), mn_spi->max_lifetime);
binding->exp_time = time(NULL) + binding->timeout;

if (send_reg_repl(rsock, binding, mn_spi, &ext, force_iface,
                  code == REGREP_ACCEPTED_NO_SB ?
                  REGREP_ACCEPTED_NO_SB : REGREP_ACCEPTED) != 0) {
        LOG2(LOG_ERR, "sending of registration reply to MN: %s, "
            "FA: %s failed\n", mn_addrstr, fa_addrstr);
        remove_tunnel(binding);
        destroy_binding(binding);
```

```
                return -1;
        }

        if (killbinding) {
                destroy_binding(binding);
        } else {
                /* increase timeout with one second so that the one second
                 * timer resolution doesn't cause too early expirations */
                binding->timeout++;

                /* make sure that the bindings module gets timer update just
                 * before adding the binding */
                check_bindings();
                binding_add(bindings, binding);
        }

        return 0;
}
```

A listagem apresentada a seguir mostra o seu código fonte depois das alterações.

```
/**
 * handle_reg_msg:
 * @s: registration message socket (UDP)
 *
 * Processes a registration request message that has arrived on socket @s.
 *
 * Returns:
 *   0 on success or -1 on failure
 */
static int
handle_reg_msg(struct interface_entry *from_iface, int s)
{
        char msg[MAXMSG];
        struct sockaddr_in cli_addr;
        struct in_addr dst_addr;
        struct msghdr mh;
        char cdata[256];
        struct iovec iov;
        struct cmsghdr *cmsg;
        struct bindingentry *binding;
        struct spi_entry *mn_spi;
        int n, res, rsock;
        struct msg_extensions ext;
        int killbinding;
        int code, auth_type;
        char mn_addrstr[17];
        char fa_addrstr[17];
        struct ha_tunnel_data *t_data = NULL;
        struct bindingkey bkey;
        struct node *node;
        struct interface_entry *force_iface = NULL;

        memset(&cli_addr, 0, sizeof(cli_addr));
        dst_addr.s_addr = 0;
        memset(&mh, 0, sizeof(mh));
        iov.iov_base = msg;
        iov.iov_len = MAXMSG;
        mh.msg_name = &cli_addr;
```

```
mh.msg_namelen = sizeof(cli_addr);
mh.msg_iov = &iov;
mh.msg_iovlen = 1;
mh.msg_control = &cdata;
mh.msg_controllen = sizeof(cdata);

n = recvmsg(s, &mh, 0);

DEBUG(DEBUG_FLAG, "Received %d bytes from %s:%d\n", n,
     inet_ntoa(cli_addr.sin_addr), ntohs(cli_addr.sin_port));

if (n < 0) {
          LOG2(LOG_ERR, "handle_reg_msg - recvfrom failed - %s\n",
               strerror(errno));
          return -1;
}

for (cmsg = CMSG_FIRSTHDR(&mh); cmsg != NULL;
     cmsg = DYNAMICS_CMSG_NXTHDR(&mh, cmsg)) {
          if (cmsg->cmsg_level == SOL_IP &&
              cmsg->cmsg_type == IP_PKTINFO) {
                    struct _in_pktinfo *pkt;
                    pkt = (struct _in_pktinfo *) CMSG_DATA(cmsg);
                    dst_addr = pkt->ipi_addr;
                    DEBUG(DEBUG_FLAG, "\tIP_PKTINFO: ipi_ifindex=%i "
                         "ipi_spec_dst=%s ", pkt->ipi_ifindex,
                         inet_ntoa(pkt->ipi_spec_dst));
                    DEBUG(DEBUG_FLAG, "ipi_addr=%s\n",
                         inet_ntoa(pkt->ipi_addr));
          }
}

/* use the matching UDP socket for the registration reply */
rsock = -1;
for (node = list_get_first(&config.interfaces); node != NULL;
     node = list_get_next(node)) {
          struct interface_entry *iface =
                    (struct interface_entry *) node;
          if (iface->addr.s_addr == dst_addr.s_addr ||
                    (iface->ha_disc &&
                     iface->bcaddr.s_addr == dst_addr.s_addr)) {
                    rsock = iface->udp_sock;
          }
}
if (rsock < 0 && from_iface != NULL &&
    dst_addr.s_addr == (unsigned int) -1) {
          DEBUG(DEBUG_FLAG, "Broadcast 255.255.255.255 destination - "
               "using interface[%s] UDP socket[%i]\n", from_iface->dev,
               from_iface->udp_sock);
          rsock = from_iface->udp_sock;
}
if (rsock < 0) {
          LOG2(LOG_WARNING, "Could not find UDP socket for "
              "registration reply - trying recv. socket\n");
          rsock = s;
}

res = parse_msg(msg, n, &ext);

if (ext.req != NULL && ntohs(ext.req->lifetime) == 0 &&
    ext.req->home_addr.s_addr == cli_addr.sin_addr.s_addr &&
    ext.req->home_addr.s_addr == ext.req->co_addr.s_addr) {
          /* RFC 2002, Ch. 3.8.3.1:
           * When HA is replying to deregistration (lifetime == 0 &&
           * COA == home addr) whose srcIP is MN's home address, the
           * reply must be send directly to home network bypassing any
           * mobility bindings */
          DEBUG(DEBUG_FLAG, "Dereg. from home => bypassing mobility "
               "bindings in routing\n");
          force_iface = from_iface;
```

```
}

if (res == -3 || res == -4) {
        LOG2(LOG_WARNING, "Unknown critical vendor extension in a "
            "request from %s:%i\n", inet_ntoa(cli_addr.sin_addr),
            ntohs(cli_addr.sin_port));
        send_reg_failure(rsock, &cli_addr, NULL, 0,
                         res == -3 ? REGREP_UNSUPP_VENDOR_ID_MN_HA :
                         REGREP_UNSUPP_VENDOR_ID_FA_HA, NULL, &ext,
                         force_iface, NULL, from_iface);
        stats.discarded_vendor_ext++;
        report_discarded_msg(msg, n, &cli_addr,
                             "unknown vendor extension");
        return -1;
} else if (res != 0 || ext.req == NULL) {
        char *reason = "N/A";
        if (res == -1) {
                reason = "unknown extension";
                stats.discarded_unknown_ext++;
        } else if (res == -2) {
                reason = "malformed message";
                stats.discarded_malformed_msg++;
        } else if (ext.req == NULL) {
                reason = "not a request";
                stats.discarded_not_request++;
        }
        LOG2(LOG_WARNING, "Message parser failed or not a request "
            "(from %s:%i): %s\n", inet_ntoa(cli_addr.sin_addr),
            ntohs(cli_addr.sin_port), reason);
        report_discarded_msg(msg, n, &cli_addr, reason);
        return -1;
}

memcpy(&bkey.mn_addr, &ext.req->home_addr, sizeof(bkey.mn_addr));
bkey.ha_addr = config.sha_addr.s_addr != 0 ? config.sha_addr :
        ext.req->ha_addr;
bkey.priv_ha = config.priv_ha;
binding = binding_fetch(bindings, &bkey);
if (binding != NULL)
        t_data = (struct ha_tunnel_data *) binding->data;
mn_spi = validate_request(binding, cli_addr.sin_addr, msg, n, &ext,
                          &code, &auth_type);
if (code > 2) {
        send_reg_failure(rsock, &cli_addr, mn_spi, 0, code,
                         (t_data != NULL ? &t_data->nonce : NULL),
                         &ext, force_iface, binding, from_iface);
        stats.req_rejected++;
        report_discarded_msg(msg, n, &cli_addr, "invalid message");
        return -1;
}
if (mn_spi == NULL) {
        stats.req_rejected++;
        report_discarded_msg(msg, n, &cli_addr, "MN SPI not found");
        return -1;
}

dynamics_strlcpy(mn_addrstr, inet_ntoa(ext.req->home_addr),
                 sizeof(mn_addrstr));
dynamics_strlcpy(fa_addrstr, inet_ntoa(ext.req->co_addr),
                 sizeof(fa_addrstr));

if (binding != NULL) {
        t_data->auth_type = (auth_type == AUTH_MAC_RFC2002 ?
                             AUTH_RFC2002BIS : AUTH_RFC2002);

        /* If the id field is identical the message was a duplicate.
         * In order to avoid breaking the tunnel drop this message as
         * a special case without sending a failure reply if the replay
         * protection is used
         */
```

```
            if (mn_spi->replay_method != REPLAY_PROTECTION_NONE &&
                binding->id[0] == ext.req->id[0] &&
                binding->id[1] == ext.req->id[1]) {
                        DEBUG(DEBUG_FLAG,
                             "Received duplicate request - dropping it\n");
                        stats.req_rejected++;
                        return -1;
            }

            /* Make sure that the id field is incremented if the timestamp
             * protection is used */
            if (mn_spi->replay_method == REPLAY_PROTECTION_TIMESTAMP &&
                (ntohl(binding->id[0]) > ntohl(ext.req->id[0]) ||
                 (ntohl(binding->id[0]) == ntohl(ext.req->id[0]) &&
                  ntohl(binding->id[1]) >= ntohl(ext.req->id[1])))) {
                        LOG2(LOG_NOTICE, "Timestamp did not increase "
                             "(MN=%s, FA=%s)\n", mn_addrstr, fa_addrstr);
                        send_reg_failure(rsock, &cli_addr, mn_spi,
                                         t_data->auth_type,
                                         REGREP_ID_MISMATCH_HA,
                                         &t_data->nonce, &ext, force_iface,
                                         binding, from_iface);
                        stats.req_rejected++;
                        return -1;
            }

            /* remove binding from list (does not remove tunnels etc.) */
                        binding_remove(binding);
} else {
            LOG2(LOG_NOTICE, "MN %s registers, FA is %s\n",
                mn_addrstr, fa_addrstr);

            ret = system("nsupdate -k Knovo.jean.com.+157+15166.private configteste");

}

stats.req_accepted++;

killbinding = 0;

if (binding &&
   (binding->lower_addr.s_addr != (config.sha_addr.s_addr != 0 ?
                                   config.sha_addr.s_addr :
                                   ext.req->co_addr.s_addr) ||
    ext.req->lifetime == 0)) {
            DEBUG(DEBUG_FLAG, "Found existing binding for MN %s\n",
                 mn_addrstr);

            if (ext.req->co_addr.s_addr == ext.req->home_addr.s_addr &&
                ext.req->lifetime == 0) {
                        DEBUG(DEBUG_FLAG, "MN is deregistering all its "
                             "simultaneous bindings\n");
                        /* Note: RFC 2002, 3.6.1.2, special case
                         * This should be taken into account if the
                         * simultaneous bindings are implemented.
                         * At the moment we just deregister the only possible
                         * binding below. */
```

```
                }

#ifdef USE_TEARDOWN
                /* if co_addr differs in request, send reply to old
                 * co_addr to purge old tunnel segment */
                if (config.sha_addr.s_addr == 0 &&
                    binding->lower_addr.s_addr != ext.req->co_addr.s_addr) {
                        binding->timeout = 0;
                        DEBUG(DEBUG_FLAG, "Sending purge msg to %s\n",
                             inet_ntoa(binding->lower_addr));
                        send_reg_repl(s, binding, mn_spi, NULL, force_iface,
                                        REGREP_ACCEPTED);
                }
#endif

                if (ext.req->lifetime > 0) {
                        /* it was a location update */
                        LOG2(LOG_INFO, "MN %s updating tunnel to FA %s\n",
                            mn_addrstr, fa_addrstr);
                        if (switch_tunnel(binding, &ext, mn_spi) < 0)
                                return -1;
                } else {
                        /* if it is a deregistration request,
                         * destroy the binding after the reply is sent,
                         * since the binding is needed when sending the
                         * reply */
                        LOG2(LOG_NOTICE, "MN %s deregisters\n", mn_addrstr);
                        remove_tunnel(binding);
                        if (cli_addr.sin_addr.s_addr == binding->mn_addr.s_addr
                            && ext.req->co_addr.s_addr ==
                            ext.req->home_addr.s_addr) {
                                DEBUG(DEBUG_FLAG, "MN returned home and "
                                     "deregistered all care-of addresses\n");
                                /* FIX: also HA should send gratuitous ARP
                                 * telling that MN takes care of its own
                                 * packets from now on;
                                 * draft-ietf-mobileip-rfc2002-bis-03.txt, 4.6;
                                 * This would require that HA would get the
                                 * link-layer address of the MN from the
                                 * deregistration message and use it in the
                                 * gratuitous ARP packet */
                        }
                        killbinding = 1;
                }
        }

        if (binding == NULL) {
```

```
        if (ntohs(ext.req->lifetime) == 0) {
                DEBUG(DEBUG_FLAG, "Deregistration attempt, but binding"
                    " not found - creating temporary binding\n");
                binding = create_binding(&ext, mn_spi, 0);
                killbinding = 1;
        } else {
                binding = create_binding(&ext, mn_spi, 1);
        }

        if (binding == NULL) {
                LOG2(LOG_ERR, "Failed to create binding for MN: %s, "
                    "FA: %s\n", mn_addrstr, fa_addrstr);
                send_reg_failure(rsock, &cli_addr, mn_spi,
                                (auth_type == AUTH_MAC_RFC2002 ?
                                 1 : 0),
                                REGREP_NO_RESOURCES_HA, NULL, &ext,
                                force_iface, binding, from_iface);
                return -1;
        }
        t_data = (struct ha_tunnel_data *) binding->data;
        t_data->auth_type = (auth_type == AUTH_MAC_RFC2002 ?
                            AUTH_RFC2002BIS : AUTH_RFC2002);
}
ASSERT(binding != NULL);
t_data = (struct ha_tunnel_data *) binding->data;
ASSERT(t_data != NULL);

binding->mod_time = time(NULL);
t_data->reverse_tunnel = (ext.req->opts & REGREQ_REVERSE_TUNNEL) != 0;
if (ext.req->opts & REGREQ_MINIMAL_ENCAPS)
        t_data->encapsulation = ENCAPS_MINIMAL;
else if (ext.req->opts & REGREQ_GRE_ENCAPS)
        t_data->encapsulation = ENCAPS_GRE;
else
        t_data->encapsulation = ENCAPS_IPIP;

if (ext.fa_keyreq)
        binding->fa_spi = ntohl(ext.fa_keyreq->spi);
else
        binding->fa_spi = 0;

/* If the request has a public key add it to the binding */
if (ext.fa_pubkey) {
        /* binding->fa_pubkey is NULL if we have generated a
         * new SK. If the public key sent by the FA is the same
         * as in a previous binding, we can reuse the encrypted
         * SK and save some CPU cycles */
```

```
                if (binding->fa_pubkey &&
                    binding->last_sent_fa_pubkeyrep != NULL &&
                    (memcmp(binding->fa_pubkey,
                            ext.fa_pubkey,
                            GET_KEY_EXT_LEN(ext.fa_pubkey)) == 0))
                {
                        DEBUG(DEBUG_FLAG, "Reusing encrypted SK for FA %s\n",
                            fa_addrstr);
                } else {
                        if (binding->fa_pubkey)
                                free(binding->fa_pubkey);
                        if (binding->last_sent_fa_pubkeyrep)
                                free(binding->last_sent_fa_pubkeyrep);
                        binding->last_sent_fa_pubkeyrep = NULL;
                        binding->fa_pubkey = (struct msg_key *)
                                malloc(GET_KEY_EXT_LEN(ext.fa_pubkey));
                        if (binding->fa_pubkey != NULL) {
                                memcpy(binding->fa_pubkey, ext.fa_pubkey,
                                    GET_KEY_EXT_LEN(ext.fa_pubkey));
                        } else {
                                LOG2(LOG_ERR, "Not enough memory "
                                    "for fa_pubkey\n");
                                return -1;
                        }
                        binding->last_sent_fa_pubkeyrep =
                                dynamics_do_rsa_encrypt(binding->key,
                                                        binding->keylen,
                                                        binding->fa_pubkey);
                        if (binding->last_sent_fa_pubkeyrep == NULL) {
                                LOG2(LOG_ERR, "RSA encryption failed (MN=%s, "
                                    "FA=%s)\n", mn_addrstr, fa_addrstr);
                        }
                }

        } else {
                /* remove the (possible) old FA pubkey from the binding to make
                 * sure that the next FA using RSA will get the correct key */
                if (binding->fa_pubkey)
                        free(binding->fa_pubkey);
                binding->fa_pubkey = NULL;

                if (binding->last_sent_fa_pubkeyrep)
                        free(binding->last_sent_fa_pubkeyrep);
                binding->last_sent_fa_pubkeyrep = NULL;
        }

        memcpy(&binding->id, ext.req->id, REG_REQ_ID_LEN);
```

```
        /* store where the request came from */
        t_data->lower_saddr.s_addr = cli_addr.sin_addr.s_addr;
        binding->lower_port = cli_addr.sin_port;

        binding->timeout = MIN(ntohs(ext.req->lifetime), mn_spi->max_lifetime);
        binding->exp_time = time(NULL) + binding->timeout;

        if (send_reg_repl(rsock, binding, mn_spi, &ext, force_iface,
                          code == REGREP_ACCEPTED_NO_SB ?
                          REGREP_ACCEPTED_NO_SB : REGREP_ACCEPTED) != 0) {
                LOG2(LOG_ERR, "sending of registration reply to MN: %s, "
                    "FA: %s failed\n", mn_addrstr, fa_addrstr);
                remove_tunnel(binding);
                destroy_binding(binding);
                return -1;
        }

        if (killbinding) {
                destroy_binding(binding);
        } else {
                /* increase timeout with one second so that the one second
                 * timer resolution doesn't cause too early expirations */
                binding->timeout++;

                /* make sure that the bindings module gets timer update just
                 * before adding the binding */
                check_bindings();
                binding_add(bindings, binding);
        }

        return 0;
}
```

Anexo B – Implementação

Os seguintes aspectos serão descritos:

a) Pacotes utilizados
   - Atualizações do sistema operacional Linux
   - BIND
   - *Dynamics – HUT Mobile IP (source forge)*

b) Alterações de código
   - HA
   - Cliente

c) Opções de *kernel*

d) Algoritmo HMAC

e) Algoritmo MD5

f) Algoritmo HMAC-MD5

Pacotes utilizados

Os principais pacotes utilizados na implementação estão listados na tabela 3.

Tabela 3 ***Principais pacotes utilizados na implementação***

Pacotes	Funcionalidades que implementam
Openssl096B_0_9_6B_3_I386.RPM	Openssl
bind-9.3.0beta3.tar.gz	BIND
Bind_9_2_1_16_I386.RPM	BIND
Bind_Devel_9_2_1_16_I386.RPM	BIND
Bind_Utils_9_2_1_16_I386.RPM	BIND
dynamics-0.8.1-1.i386.rpm	Dynamics - HUT Mobile IP
dynamics-0.8.1-1.src.rpm	Dynamics - HUT Mobile IP
dynamics-0.8.1.tar.gz	Dynamics - HUT Mobile IP
dynamics-devel-0.8.1-1.i386.rpm	Dynamics - HUT Mobile IP
dynamics-fa-0.8.1-1.i386.rpm	Dynamics - HUT Mobile IP
dynamics-ha-0.8.1-1.i386.rpm	Dynamics - HUT Mobile IP
dynamics-mn-0.8.1-1.i386.rpm	Dynamics - HUT Mobile IP

Atualizações do sistema operacional Linux

O Openssl096B_0_9_6B_3_I386.RPM é o pacote que implementa o Openssl, que é a biblioteca que contem as características DNSSEC tais como SIG(0) e "zona assinada", e que pode ser encontrada em http://www.openssl.org/ . O Openssl é uma biblioteca completa, mas que é usada na maioria das vezes apenas para a geração de certificados.

Genericamente falando, o Openssl é uma "caixa de ferramentas" criptográfica que implementa os protocolos de rede *Secure Sockets Layer* e *Transport Layer Security* e os padrões de criptografia relacionados que são requeridos por estes protocolos. Ele pode ser usado, por exemplo, para: criação dos parâmetros de chave RSA, DH (*Diffie - Hellman*) e DSA; criação de certificados X.509, CSRs (*Certificate Signing Request*) e CRLs (*Certificate*

Revocation List); cálculo de MAC; criptografia e decriptografia; testes de servidor e cliente SSL (*Secure Sockets Layer*) / TLS (*Transport Layer Security*); e manuseio de *e-mails* criptografados ou assinados com formato S / MIME.

As funções padrão do Openssl são: interpretar um seqüência ASN.1; gerenciamento da Autoridade Certificadora; determinação de "cifradores" disponíveis; gerenciamento da Lista de Revogação de Certificados; conversão de CRL para PKCS#7; cálculo de MAC; gerenciamento de dados DSA; geração de parâmetros DSA; codificação com "cifradores"; conversão de Número de Erro para *String* de Erro; geração e gerenciamento de parâmetros Diffie – Hellman; geração de parâmetros DSA; geração de parâmetros RSA; Protocolo da Verificação de Certificados *Online*; geração de *hashed passwords*; gerenciamento de dados PKCS#12; gerenciamento de dados PKCS#7; geração de *bytes* pseudo-aleatórios; gerenciamento das Requisições de Assinatura dos Certificados X.509; gerenciamento de dados RSA; assinatura, verificação, criptografia e decriptografia RSA; implementação de um cliente SSL / TLS genérico que pode estabelecer uma conexão transparente com um servidor remoto através dos protocolos SSL / TLS; implementação de um servidor SSL / TLS genérico que aceita conexões de clientes remotos através dos protocolos SSL / TLS; medição de tempo da conexão SSL; gerenciamento de dados da sessão SSL; processamento de *e-mails* S / MIME; medição da velocidade de algoritmos; verificação de certificados X.509; e gerenciamento de dados dos certificados X.509. Destas funções utilizamos neste trabalho: o cálculo de MAC, para gerar *digests* MD5, que é o padrão para assinar mensagens de atualização dinâmica do DNS; codificação Base64, necessária para gravar e ler as chaves armazenadas no DNS, chaves estas que são utilizadas para autenticar as mensagens de atualização; geração de *hashed passwords*, através do protocolo HMAC, utilizadas para codificar os *digests* MD5; geração de *bytes* pseudo-aleatórios, utilizados nos cálculos; e medição da velocidade de algoritmos, para analisar o desempenho dos algoritmos utilizados.

BIND

Os pacotes *bind* implementam o BIND versão 9, que é uma das maiores reescritas de quase todos os aspectos da arquitetura BIND. Algumas das características importantes do BIND 9 são: Segurança DNS – DNSSEC e TSIG (requisições DNS assinadas); IP

versão 6 – Responde consultas DNS em *sockets* IPv6, aceita IPv6 *resource records*, e apresenta uma biblioteca experimental para um "resolvedor" IPv6; Avanços do Protocolo DNS – IXFR, DDNS, Notify, EDNS0, e aumento da concordância com os padrões; Visões – Um processo servidor pode fornecer múltiplas "visões" do espaço de nomes DNS, por exemplo uma visão "interna" para certos clientes, e uma visão "externa" para outros; Suporte multiprocessador; e Aumento da portabilidade arquitetural.

O desenvolvimento do BIND versão 9 tem sido executado pelas seguintes organizações: Sun Microsystems, Inc.; Hewlett Packard; IBM; Process Software Corporation; Silicon Graphics, Inc.; Network Associates, Inc.; U. S. Defense Information Systems Agency; USENIX Association; Stichting NLnet – NLnet Foundation; e Nominum, Inc. A versão 9 do BIND foi necessária para o nosso trabalho, por causa das suas novas características quanto à segurança DNS, uma vez que as requisições DNS assinadas com TSIG são uma das pedras angulares deste trabalho.

Dynamics – HUT Mobile IP

Os pacotes *dynamics* implementam o *Dynamics – HUT Mobile IP*, que é um sistema de tunelamento IP hierárquico e dinâmico, desenvolvido a partir de 1.998 por um grupo de pesquisadores. A distribuição utilizada é a versão protótipo 0.8 do sistema, que ainda tem erros e deficiências que podem resultar em resultados inesperados, como perda de dados.

O sistema operacional principal para o *Dynamics – HUT Mobile IP* é o Linux, mas partes da funcionalidade MN foram portadas para o Microsoft Windows; entretanto a implementação descrita nesta dissertação foi completamente desenvolvida em ambiente Linux. O programa é projetado para ser usado com redes IPv4 e não suporta IPv6, e é desenvolvido para ser usado com o sistema operacional Linux, sendo que todos os testes foram feitos com as versões de *kernel* 2.2.x e 2.4.x.

Se o suporte ao tunelamento IPIP é compilado como um módulo de *kernel* ele deve ser carregado antes do sistema poder ser usado, e se o suporte para endereços de origem do MN privados e não únicos é necessário, os FAs necessitam incluir túneis GRE. As rotinas RSA incluídas no programa usam a biblioteca GNU MP para precisão aritmética arbitrária, que deve estar instalada para que sejamos capazes de compilar os programas, sendo que a versão utilizada foi encontrada a partir de http://www.gnu.org/software/gmp/gmp.html .

O pacote *Dynamics – HUT Mobile IP* foi utilizado para simular o ambiente de redes móveis, fornecendo mobilidade IP, atuando como provedor de macro-mobilidade. A implementação focou-se em estender este último pacote, adicionando rotinas para atualização dinâmica de DNS, e dependendo do cenário a rotina é adicionada no *software* do HA ou do MN.

Alterações de código

Home Agent

O *Home Agent* trata do encaminhamento de datagramas para a localização corrente do *host*. No primeiro cenário apresentado no capítulo 5, a atualização da entrada DNS do MN é disparada pelo HA, e a seguir apresentaremos as alterações executadas no código desta entidade.

No pacote dynamics-0.8.1.tar.gz, temos o código fonte do *Dynamics – HUT Mobile IP,* que foi a implementação de IP Móvel utilizada. Em seu diretório *\src\ha*, encontramos os códigos relativos à implementação da entidade HA.

Dentro deste diretório, no arquivo *ha.c*, temos a implementação do HA propriamente dita. Neste arquivo, encontramos a função *handle_reg_msg,* que processa uma mensagem de requisição de registro que tenha chegado em um *socket* passado como parâmetro de entrada.

É nesta função que fizemos as alterações de código necessárias e, por isso, mostramos no anexo A o seu código fonte, antes e depois das alterações.

No código antes das alterações vemos que, caso a função não entre em nenhum dos desvios que conferem inconsistências, ela registra no histórico o *binding* do MN como efetuado. É justamente neste bloco de código, depois que todas as consistências foram feitas, que fizemos a alteração para atualizar a entrada DNS.

Na tabela 4 vemos a alteração no trecho de código citado.

Tabela 4 ***Alteração da função handle_reg_msg***

```
} else {
                LOG2(LOG_NOTICE, "MN %s registers, FA is %s\n",
                    mn_addrstr, fa_addrstr);

                ret = system("nsupdate -k Knovo.jean.com.+157+15166.private configteste");

        }
```

Aqui vemos que, apenas para fins de testes, acrescentamos uma chamada a um programa externo que faz a atualização dinâmica. O *nsupdate,* com a opção *–k*, recebe como parâmetro o arquivo contendo a chave secreta; além disto, recebe um segundo parâmetro, que é um arquivo de configuração contendo os comandos de atualização dinâmica.

No código depois das alterações vemos que, após a atualização dinâmica do DNS, a função segue seu curso normal. Nos blocos posteriores à atualização, a função executa tarefas correlatas (verificar *bindings* pré-existentes, apaga-los, iniciar o *timer* para o novo *binding*), concluindo assim o registro do MN.

Cliente

O cliente, em nosso caso, é o MN; ele é o dispositivo móvel que permite a um usuário manter uma comunicação ativa enquanto muda de rede de acesso. No segundo cenário apresentado no capítulo 5, a atualização da entrada DNS é disparada pelo próprio MN, e a seguir apresentaremos as alterações executadas no código desta entidade.

No pacote dynamics-0.8.1.tar.gz, temos o código fonte do *Dynamics – HUT Mobile IP,* que foi a implementação de IP Móvel utilizada. Em seu diretório *\src\mn*, encontramos os códigos relativos à implementação da entidade MN.

Dentro deste diretório, no arquivo *mn.c*, temos a implementação do MN propriamente dita. Neste arquivo, encontramos a função *connected,* que é onde a entidade entra no estado “conectado”.

É nesta função que fizemos as alterações de código necessárias e, por isso, mostramos na tabela 5 o seu código fonte, antes das alterações.

Tabela 5 ***Função connected***

```
void connected(int type, __u16 lifetime)
{
        DEBUG(DEBUG_INFO, "connected(%i, %i)\n", type, lifetime);
        ASSERT(lifetime > 0);
        if (mn.current_adv != NULL) {
                if (mn.current_adv->addr.s_addr != mn.fa_addr.s_addr) {
                        DEBUG(DEBUG_INFO, "fa_addr=%s != ",
                            inet_ntoa(mn.fa_addr));
                        DEBUG(DEBUG_INFO, "current_adv->addr=%s\n",
                            inet_ntoa(mn.current_adv->addr));
                }
                mn.current_adv->in_use = 1;
                /* Este FA funciona, então não degrade a prioridade: */
                mn.current_adv->prio_degrade_percent = 0;
        } else {
                LOG2(LOG_WARNING, "connected - current_adv == NULL\n");
                mn.warn_str = "connected - current_adv == NULL";
        }
        /* O HA não deve usar um Lifetime maior e os FAs  não podem mudar
         * o lifetime requisitado- força o lifetime a ser no máximo o
         * tempo configurado*/
        if (lifetime > config.mn_default_tunnel_lifetime) {
                LOG2(LOG_WARNING, "Lifetime in the reply (%i sec) larger than "
                    "expected (%i sec) - lowering it\n", lifetime,
                    config.mn_default_tunnel_lifetime);
                mn.warn_str = "lifetime in the reply larger than expected";
                lifetime = config.mn_default_tunnel_lifetime;
        }
        if (config.enable_fa_decapsulation &&
            mn.fa_addr.s_addr != mn.tunnel_addr.s_addr &&
            mn.tunnel_addr.s_addr != 0 &&
            mn.current_adv != NULL)
                remove_fa_host_routes(0);
    /* Se o fim do túnel mudou, ou o túnel caiu, cria túnel para o FA */
        if (mn.fa_addr.s_addr != mn.tunnel_addr.s_addr || !real_tunnel_up) {
              if (real_tunnel_up) {
                        DEBUG(DEBUG_STATES, "Restart tunneling\n");
                        restart_tunneling();
                } else {
                        DEBUG(DEBUG_STATES, "Start tunneling\n");
                        start_tunneling();
                }
        }
        /* configure o lifetime timer */
        timers[TIMER_LIFETIME].tv_sec = timers[TIMER_REQUEST].tv_sec +
                lifetime;
        timers[TIMER_LIFETIME].tv_usec = timers[TIMER_REQUEST].tv_usec;
          /* Se a conexão foi aprovada pelo HA,
           * reinicia o timer para re-registrar o túnel*/
        if (type == CON_HA) {
                set_reregistration_time(lifetime);
        }
        if (mn.state != MN_CONNECTED && type == CON_HA) {
                LOG2(LOG_INFO, "Connection established.\n");
                mn.info_str = "connection established";
        }
        DEBUG(DEBUG_TIMERS, "setting TIMER_GEN = TIMER_REREG %li.%06li (in %li"
             " sec)\n",
             timers[TIMER_REREG].tv_sec, timers[TIMER_REREG].tv_usec,
             timers[TIMER_REREG].tv_sec - time(NULL));
        timers[TIMER_GEN] = timers[TIMER_REREG];
        mn.state = MN_CONNECTED;
        mn.tunnel_up = 1;
        DEBUG(DEBUG_STATES, "State: Connected\n");
}
```

Vemos que, esta função registra no histórico o *binding* do MN como efetuado, além de consistir o *lifetime*. É no final desta função, depois que todas as consistências foram feitas, que fizemos a alteração para atualizar a entrada DNS.

Na tabela 6 vemos a alteração no trecho de código citado.

Tabela 6 ***Alteração da função connected***

```
timers[TIMER_GEN] = timers[TIMER_REREG];
mn.state = MN_CONNECTED;
mn.tunnel_up = 1;
DEBUG(DEBUG_STATES, "State: Connected\n");

ret = system("nsupdate -k Knovo.jean.com.+157+15166.private configteste");
```

Aqui vemos que, apenas para fins de testes, acrescentamos uma chamada a um programa externo que faz a atualização dinâmica. O *nsupdate,* com a opção *–k*, recebe como parâmetro o arquivo contendo a chave secreta; além disto, recebe um segundo parâmetro, que é um arquivo de configuração contendo os comandos de atualização dinâmica.

Opções de kernel

O *kernel* instalado pela distribuição Red Hat 6.x suporta a maioria das opções de *kernel* exigidas pelo pacote *Dynamics – HUT Mobile IP*, mas os *Foreign Agents* necessitam ainda do roteamento baseado em políticas avançado e os *Mobile Nodes* rodarão mais eficientemente com *Linux Socket Filters,* sendo que a configuração de *kernel* a partir do Red Hat 7.x parece incluir todas as opções necessárias. As seguintes opções de tempo de compilação do *kernel* Linux são necessárias para montar um *kernel* compatível: Suporte a módulos carregáveis; Opções de rede – CONFIG_PACKET, CONFIG_NETLINK, CONFIG_RTNETLINK, CONFIG_FILTER e CONFIG_NET_IPIP; além destas as seguintes opções são necessárias para os *Foreign Agents* – CONFIG_IP_ADVANCED_ROUTER e CONFIG_IP_MULTIPLE_TABLES; opção adicional para as extensões *wireless* de MN – CONFIG_NET_RADIO.

Algoritmo HMAC

O HMAC é um algoritmo de autenticação de mensagens usando funções de *hash*. Tem havido recentemente bastante interesse no assunto da autenticação de informações usando funções de *hash* criptográficas como MD5 e SHA (*Secure Hash*), particularmente para protocolos de segurança para Internet, porém geralmente estas funções são usadas em conjunto com o algoritmo HMAC.

A origem do interesse vem do fato de que duas máquinas comunicando-se através de um canal inseguro necessitam de um método pelo qual qualquer tentativa de modificar a informação enviada de um para o outro, ou falsificar sua origem, seja detectada. A abordagem mais comum era construir MACs a partir de cifradores de bloco como DES.

Mas, hoje em dia, as pessoas parecem concordar que construções de MACs baseadas em funções de *hash* valem mais a pena. A construção HMAC pretende preencher os requisitos das análises de segurança atuais.

Na verdade, o algoritmo HMAC nada mais é do que uma construção baseada em *hashing,* mas também em chaves, para autenticação de mensagens. O algoritmo é descrito na RFC 2104 [29], como um mecanismo para autenticação de mensagens usando funções de *hash* criptográficas, sendo que sua força criptográfica depende das propriedades da função de *hash* utilizada.

Sua utilidade reside no fato de que fornecer uma forma de conferir a integridade da informação transmitida através de um meio não confiável, ou armazenada nele, é uma necessidade primordial no mundo da computação e das comunicações abertas. O HMAC pode ser usado em conjunto com qualquer função de *hash* criptográfica iterativa.

Os objetivos principais desta construção são: utilizar, sem modificações, funções de *hash* disponíveis; preservar o desempenho original da função de *hash* sem incorrer em uma degradação significativa; usar e manipular chaves de uma forma simples; ter uma análise criptográfica bastante compreensível da força do mecanismo de autenticação baseada em suposições razoáveis sobre a função de *hash* utilizada; e permitir a fácil substituição da função de *hash* utilizada no caso de funções de *hash* mais rápidas ou mais seguras serem encontradas ou necessárias. A RFC 2104 [29] especifica o HMAC usando uma função de *hash* criptográfica genérica.

Na época da redação da RFC do HMAC [29], o MD5 e o SHA – 1 eram as funções de *hash* criptográficas mais amplamente utilizadas. De qualquer forma, programadores e usuários precisam estar cientes de possíveis desenvolvimentos criptanalíticos a respeito de alguma destas funções de *hash* criptográficas, e da eventual necessidade de substituir a função de *hash* utilizada.

A definição do HMAC requer uma função de *hash* criptográfica, chamada H, e uma chave secreta K. Assume-se que H é uma função de *hash* criptográfica onde os dados são modificados através da iteração de uma função de compressão básica sobre os blocos de dados.

Chama-se de B o tamanho em *bytes* destes blocos, e de L o tamanho em *bytes* das saídas. A chave de autenticação K pode ser de qualquer tamanho até B, o tamanho de bloco da função *hash*.

Aplicações que usam chaves maiores que B *bytes* primeiro modificarão a chave usando H e então usarão a *string* de L *bytes* resultante como a chave atual do HMAC. Em qualquer caso o tamanho mínimo recomendado para K é L *bytes*.

Definiram-se duas *strings* fixas e diferentes, *ipad* e *opad,* como segue:

ipad = o *byte* 0x36 repetido B vezes

opad = o *byte* 0x5C repetido B vezes

Para calcular o HMAC sobre o dado “exemplo” executamos

H (K XOR *opad,* H (K XOR *ipad,* exemplo))

Algoritmo MD5

O MD5 é um algoritmo de *hash* de 128 *bits* unidirecional desenvolvido pela *RSA Data Security, Inc.*, usado por *softwares* com protocolo ponto-a-ponto. “HASH” é uma função matemática que transforma arquivos de qualquer extensão em um código de tamanho fixo.

O MD5 é um algoritmo para cálculo de *message digest*. Ele é descrito na RFC 1321 [30], como um algoritmo para cálculo de *message digest,* que tem como entrada uma mensagem de tamanho arbitrário e produz como saída uma “impressão digital” da entrada com 128 *bits*.

O algoritmo MD5 foi projetado para ser um pouco mais rápido em máquinas de 32 *bits*. Ele é uma extensão do algoritmo de *message digest* MD4.

Na RFC 1321 [30], uma "palavra" é uma quantidade de 32 *bits,* e um "*byte*" é uma quantidade de oito *bits*. Define-se *x_i* como "x sub i".

Define-se, também, o símbolo "+" como adição de palavras. Define-se *X xor Y* como o XOR *bit-a-bit* de X e Y, e XY como o AND *bit-a-bit* de X e Y.

Começamos supondo que temos uma mensagem de b *bits* como entrada, e que queremos encontrar seu *message digest*. Imaginamos os *bits* da mensagem escritos assim: *m_0 m_1 ... m_{b – 1}*.

Os cinco passos seguintes são executados para calcular o *message digest* da mensagem: adicionar *bits* de preenchimento, adicionar o tamanho, inicializar o *buffer* MD5, processar a mensagem em blocos de 16 palavras e produzir a saída. O algoritmo de *message digest* MD5 é simples de implementar, e fornece uma "impressão digital" ou *message digest* de uma mensagem de tamanho arbitrário.

Assume-se que é computacionalmente inviável produzir duas mensagens tendo o mesmo *message digest,* ou produzir qualquer mensagem tendo um dado *message digest* alvo pré-especificado. O algoritmo MD5 pretende ser utilizado em aplicações de assinatura digital, onde um arquivo grande precisa ser "comprimido" de uma forma segura antes de ser encriptado com uma chave privada – em um sistema de criptografia baseado em chaves pública / privada – ou chave secreta.

Algoritmo HMAC - MD5

O HMAC – MD5 é uma construção da função de autenticação de mensagens HMAC usando a função de *hash* MD5. Esta construção é usada pelo IPSEC e por outros protocolos para autenticar mensagens.

O método geral para construir uma função de autenticação de mensagens HMAC foi descrito anteriormente. O código fonte C das funções usadas para gerar resultados HMAC – MD5 é listado na tabela 7.

Tabela 7 ***Código fonte C das funções HMAC – MD5***

```
#ifndef MD5_DIGESTSIZE
#define MD5_DIGESTSIZE  16
#endif

#ifndef MD5_BLOCKSIZE
#define MD5_BLOCKSIZE   64
#endif

/* Função para imprimir o digest */
void
pr_md5(FILE* fp, char* s, int t)
{
        int     i ;

        fprintf(fp, "0x") ;
        for (i = 0 ; i < t ; i++)
                fprintf(fp, "%02x", s[i]) ;
        fprintf(fp, "0) ;
}

void truncate
(
 char*   d1,   /* dado a ser “truncado” */
 char*   d2,   /* dado “truncado” */
 int     len   /* tamanho em bytes a ser mantido */
)
{
        int     i ;
        for (i = 0 ; i < len ; i++) d2[i] = d1[i];
}



/* Função para calcular o digest */
void
hmac_md5
(
 char*    k,     /* chave secreta */
 int      lk,    /* tamanho da chave em bytes */
 char*    d,     /* dado */
 int      ld,    /* tamanho dos dados em bytes */
 char*    out,   /* buffer de saída, pelo menos "t" bytes */
 int      t
)
{
        MD5_CTX ictx, octx ;
        char    imd5 [MD5_DIGESTSIZE], omd5[MD5_DIGESTSIZE] ;
        char    key[MD5_DIGESTSIZE] ;
        char    buf[MD5_BLOCKSIZE] ;
        int     i ;

        if (lk > MD5_BLOCKSIZE) {

                MD5_CTX         tctx ;

                MD5Init(&tctx) ;
                MD5Update(&tctx, k, lk) ;
                MD5Final(key, &tctx) ;

                k = key ;
                lk = MD5_DIGESTSIZE ;
        }
```

```
        /**** Digest interno****/

        MD5Init(&ictx) ;

        /* Completa o conteúdo da chave para o digest  interno*/
        for (i = 0 ; i < lk ; ++i) buf[i] = k[i] ^ 0x36 ;
        for (i = lk ; i < MD5_BLOCKSIZE ; ++i) buf[i] = 0x36 ;

        MD5Update(&ictx, buf, MD5_BLOCKSIZE) ;
        MD5Update(&ictx, d, ld) ;

        MD5Final(imd5, &ictx) ;

        /**** Digest externo****/

        MD5Init(&octx) ;

        /* Completa o conteúdo da chave para o digest externo*/
        for (i = 0 ; i < lk ; ++i) buf[i] = k[i] ^ 0x5C ;
        for (i = lk ; i < MD5_BLOCKSIZE ; ++i) buf[i] = 0x5C ;

        MD5Update(&octx, buf, MD5_BLOCKSIZE) ;
        MD5Update(&octx, imd5, MD5_DIGESTSIZE) ;

        MD5Final(omd5, &octx) ;

        /* trunca e imprime os resultados */
        t = t > MD5_DIGESTSIZE ? MD5_DIGESTSIZE : t ;
        truncate(omd5, out, t) ;
        pr_md5(stdout, out, t) ;
}
```

O HMAC – MD5 é o protocolo padrão para assinar mensagens de atualização dinâmica do DNS, daí sua importância para nosso trabalho, uma vez que precisamos utilizá-lo para autenticar a alteração da entrada DNS do MN.