Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

Proposta de Implementação de uma Arquitetura para a Internet de Nova Geração

Autor: Walter Wong
Orientador: Maurício Ferreira Magalhães
Co-orientador: Fábio Luciano Verdi

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Campinas, SP
2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Wong, Walter
W864p Proposta de implementação de uma arquitetura para a Internet de nova geração. / Walter Wong. –Campinas, SP: [s.n.], 2007.

Orientadores: Maurício Ferreira Magalhães, Fábio Luciano Verdi
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Arquitetura de redes de computador. 2. Redes de computação - Medidas de segurança. 3. Internet sem fio. 4. Sistemas de comunicação sem fio. 5. Criptografia de dados (Computação). 6. Redes de computação - Protocolos. I. Magalhães, Maurício Ferreira. II. Verdi, Fábio Luciano. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título

Título em Inglês:	An implementation proposal of a next generation internet architecture
Palavras-chave em Inglês:	Next-generation internet architectures, Multihoming, Security, Heterogeneous Networks
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Carlos Alberto Kamienski, Eleri Cardozo e Marco Aurélio Amaral Henriques
Data da defesa:	11/07/2007
Programa de Pós-Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Walter Wong

Data da Defesa: 11 de julho de 2007

Título da Tese: "Proposta de Implementação de uma Arquitetura para a Internet de Nova Geração"

Prof. Dr. Maurício Ferreira Magalhães (Presidente): _____

Prof. Dr. Carlos Alberto Kamienski: _____

Prof. Dr. Eleri Cardozo: _____

Prof. Dr. Marco Aurélio Amaral Henriques: _____

Resumo

A concepção original da arquitetura da Internet foi baseada em uma rede fixa e confiável. Hoje em dia, a Internet se tornou dinâmica e vulnerável aos ataques de segurança. Também não era prevista a necessidade de integração de tecnologias heterogêneas nem de ambientes sem fio. A arquitetura atual apresenta uma série de barreiras técnicas para prover estes serviços, sendo uma das maiores a sobrecarga semântica do *Internet Protocol* (IP). O endereço IP atua como localizador na camada de rede e como identificador na camada de transporte, impossibilitando novas funcionalidades como a mobilidade e abrindo brechas de segurança. Este trabalho apresenta uma proposta de implementação de uma arquitetura para Internet de nova geração para o provisionamento de novos serviços de forma natural e integrada para a Internet atual. A proposta de arquitetura de implementação oferece suporte à mobilidade, ao *multihoming*, à segurança, à integração de redes heterogêneas e às aplicações legadas através da introdução de uma nova camada de identificação na arquitetura atual. Esta nova camada tem por objetivo separar a identidade da localização e se tornar uma opção de comunicação para as redes heterogêneas. Mecanismos adicionais foram propostos para prover o suporte às funcionalidades da arquitetura, tais como a resolução de nomes em identificadores, o roteamento baseado no identificador, a gerência de localização e um plano de controle para a troca de mensagens de sinalização fim-a-fim entre os componentes da arquitetura. Para a validação da arquitetura proposta, um protótipo foi implementado e vários testes de desempenho foram realizados para avaliação do *overhead* da implementação, do modelo de segurança, da robustez e do suporte à mobilidade e às aplicações legadas.

Palavras-chave: Arquiteturas para a Internet de nova geração, mobilidade, *multihoming*, segurança, redes heterogêneas.

Abstract

The original concept of the Internet architecture was based on static and reliable networks. Nowadays, the Internet became more dynamic and vulnerable to security attacks. The integration of heterogeneous technologies and wireless environment were not predicted. The current architecture presents some technical barriers to provide these services. One of these problems is the semantic overload of the Internet Protocol (IP). The IP address acts as locator in the network layer and identifier in the transport layer, preventing new features such as mobility and allowing security flaws. This work presents an implementation proposal of a next generation Internet architecture to provide new services naturally integrated to the Internet. The implementation proposal supports mobility, multihoming, security, heterogeneous networks integration and legacy applications by the introduction of a new identification layer in the current architecture. This new layer will separate the identity from the location and become an option for communication between heterogeneous networks. Additional mechanisms were proposed to support the new functionalities of the architecture, e.g., resolution of names

to identifiers, identifier-based routing, location management and a control plane to exchange end-to-end signalling control messages between the components of the architecture. In order to evaluate the proposed architecture, a prototype was implemented and some tests were performed considering implementation overhead, security model, robustness and support for mobility and legacy applications.

Keywords: *Next-Generation Internet Architectures, mobility, multihoming, security, heterogeneous networks*.

Agradecimentos

Ao meu orientador e co-orientador, Prof. Dr. Maurício Ferreira Magalhães e Dr. Fábio Luciano Verdi, sou grato pela orientação.

Aos colegas de projeto, Luciano de Paula, Rodolfo Villaça, Rafael Pasquini, e Daniel Barboza, sou grato pelas sugestões.

To my fiancée Amy Yuan Shao for the inspiration of my work.

Aos demais colegas de pós-graduação, pelas críticas e sugestões.

À minha família pelo apoio durante esta jornada.

À Ericsson do Brasil, pelo apoio financeiro.

À minha família e a que está por vir e aos meus amigos

Sumário

Lista de Figuras **xi**

Lista de Tabelas **xiii**

Glossário **xv**

Trabalhos Publicados Pelo Autor **xix**

1 Introdução **1**
1.1 Motivações 1

2 Trabalhos e Tecnologias Relacionadas **5**
2.1 *Mobile Internet Protocol* 5
2.2 *Host Identity Protocol* 8
2.3 *Node Identity Internetworking Architecture* 13
2.4 Outras Propostas de Arquitetura 18
2.5 Tecnologias de Segurança 20
2.5.1 *Public Key Infrastructure* 20
2.5.2 Protocolo de troca de chaves *Diffie-Hellman* 22
2.5.3 *Keyed-Hash Message Authentication Code* 22
2.6 Resumo 23

3 Proposta de Implementação de uma Arquitetura para a Internet de Nova Geração **25**
3.1 Proposta de Implementação 25
3.1.1 Identificador global estático 25
3.1.2 Roteamento baseado na identidade 27
3.1.3 Mecanismo de resolução de nomes 29
3.1.4 Gerência da localização 30
3.2 Funcionalidades da arquitetura 31
3.2.1 Mobilidade 31
3.2.2 *Multihoming* 33
3.2.3 Transparência 34
3.2.4 Segurança 34
3.2.5 Heterogeneidade 36
3.2.6 Plano de Controle 36

3.3 Módulos da arquitetura 45
3.3.1 Módulos Internos 46
3.3.2 Módulos Externos 52
3.3.3 Análise do modelo de segurança 54
3.4 Resumo 55

4 Implementação e Resultados **57**
4.1 Módulos Internos 57
4.1.1 Interface Virtual 59
4.1.2 Comunicação entre os módulos 60
4.2 Módulos Externos 73
4.3 Resultados 74
4.4 Resumo 78

5 Conclusões **79**
5.1 Considerações Finais 79
5.2 Trabalhos Futuros 80
5.2.1 Roteamento 80
5.2.2 *Multihoming* e Qualidade de Serviço 80
5.2.3 Redes auto-configuráveis 80
5.2.4 Composição de domínios 81
5.2.5 Provisionamento de serviços 81
5.2.6 Segurança 81
5.2.7 Heterogeneidade 81

Referências bibliográficas **82**

A Interface virtual e recepção de eventos do Kernel **87**

B Diagramas de sequência da arquitetura **93**

Lista de Figuras

2.1 *Mobile IP versão 6.* 8
2.2 Diagrama de camadas do TCP/IP. 10
2.3 *Base Exchange* do HIP. 11
2.4 Arquitetura Node ID. 15
2.5 Cabeçalho Node ID. 15
2.6 Infraestrutura de chaves públicas. 21
2.7 Protocolo de troca de chaves *Diffie-Hellman*. 22

3.1 Geração dos identificadores de um nó. 26
3.2 Ligação da aplicação legada com a camada de identificação. 27
3.3 Simplificação do modelo da Internet. 28
3.4 Cabeçalho para a camada de identificação (cabeçalho ID). 29
3.5 Mobilidade utilizando a interface virtual. 32
3.6 Cenários de mobilidade considerados na arquitetura. 32
3.7 Exemplo de *Multihoming*. 33
3.8 Exemplo de heterogeneidade. 36
3.9 Cabeçalho das mensagens de controle. 37
3.10 Mensagem de controle REGISTRY. 38
3.11 Mensagem de controle REDIRECT. 38
3.12 Atuação da mensagem de controle REDIRECT_CORE. 39
3.13 Mensagem de sincronização SYNC. 40
3.14 Cabeçalho das mensagens de segurança. 41
3.15 Estabelecimento de uma associação segura. 42
3.16 Mensagem de segurança *Hello*. 42
3.17 Mensagem de segurança *Howau*. 43
3.18 Mensagem de segurança *Fine*. 44
3.19 Mensagem de segurança *Bye*. 44
3.20 Cabeçalho das mensagens RVS. 45
3.21 Mensagens de controle para comunicação com o RVS. 45
3.22 Módulos da arquitetura. 46
3.23 Procedimento de resolução do nome. 47
3.24 Aplicação do modelo de segurança nos pacotes. 49
3.25 Campos do cabeçalho ID utilizados no cálculo do HMAC. 49
3.26 Procedimento de autenticação e estabelecimento da chave simétrica. 52

4.1 Módulos carregados na memória conforme a funcionalidade selecionada. 58
4.2 Fluxograma de roteamento das *threads* do módulo *Routing*. 68
4.3 Cenários de mobilidade avaliados no protótipo. 75

B.1 Diagrama de sequência da resolução do nome. 93
B.2 Diagrama de sequência da resolução do identificador. 94
B.3 Diagrama de sequência do envio de dados. 95
B.4 Diagrama de sequência da recepção de dados. 96
B.5 Diagrama de sequência da mensagem de REDIRECT. 97

Lista de Tabelas

4.1 *Overhead* do protótipo na arquitetura atual. 75
4.2 Tempo de transferência de um arquivo utilizando o SCP. 76
4.3 Perda de pacotes UDP. 77
4.4 *Overhead* do protótipo utilizando o modelo de segurança. 77

Glossário

i^3 - *Internet Indirection Infrastructure*

AH - *Authentication Header*

API - *Application Programming Interface*

AS - *Autonomous System*

BA - *Binding Update Acknowledgement*

BU - *Binding Update*

CA - *Certificate Authority*

CN - *Correspondent Node*

CoA - *Care-of-address*

CSR - *Certificate Signing Request*

DH - *Diffie-Hellman*

DHCP - *Dynamic Host Configuration Protocol*

DHT - *Distributed Hash Table*

DNS - *Domain Name Service*

DNSSec - *Domain Name Service Security Extensions*

DoS - *Denial of Service*

ESP - *Encapsulating Security Payload*

FA - *Foreign Agent*

FQDN - *Fully Qualified Domain Name*

HA - *Home Agent*

HI - *Host Identity*

HIP - *Host Identity Protocol*

HIT - *Host Identity Tag*

HMAC - *Hashed Key Message Authentication Code*

IP - *Internet Protocol*

IPSec - *Internet Protocol Security*

IPv4 - *Internet Protocol version 4*

IPv6 - *Internet Protocol version 6*

LD - *Locator Domain*

LRU - *Least Recently Used*

MAC - *Message Authentication Code*

MIP - *Mobile Internet Protocol*

MIPv6 - *Mobile IP version 6*

MN - *Mobile Node*

NAT - *Network Address Translation*

NID - *Node Identifier*

NIDGW - *Gateway Identifier*

Node ID - *Node Identity Internetworking Architecture*

NR - *Node ID Router*

PC - *Peer Cache*

PH - *Packet Handler*

PHI - *Packet Handler Input*

PHO - *Packet Handler Output*

PKI - *Public Key Infrastructure*

PPP - *Point-to-Point Protocol*

RA - *Registration Authority*

RPC - *Remote Procedure Call*

RVS - *Rendezvous Server*

SA - *Security Association*

SCP - *Secure Copy*

SPI - *Security Parameter Index*

TCP - *Transmission Control Protocol*

TLI - *Transport Layer Identifier*

TTL - *Time to Live*

UDP - *User Datagram Protocol*

VPN - *Virtual Private Network*

XML - *Extensible Markup Language*

Trabalhos Publicados Pelo Autor

1. W. Wong, R. Pasquini, R. Villaça, L. de Paula, F. Verdi, M. Magalhães. "A Framework for Mobility and Flat Addressing in Heterogeneous Domains". *25° Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007), Belém, Pará, Brasil, 28 de Maio à 1 de Junho, 2007*.

Capítulo 1

Introdução

Este capítulo discute os principais problemas da Internet atualmente e as limitações da rede no provisionamento de novas funcionalidades. São apresentadas as motivações para o desenvolvimento de uma nova arquitetura e a sua implementação visando solucionar estes problemas e atender aos novos requisitos da Internet. Por fim, são apresentadas as contribuições deste trabalho e a organização do texto.

1.1 Motivações

A arquitetura original da Internet, concebida no início dos anos 60 e atualizada na década de 80, não acompanhou o avanço tecnológico, o aumento da banda disponível e o crescimento exponencial do número de usuários. A concepção original do TCP/IP previa a sua utilização em redes confiáveis e fixas voltadas para a simples transferência de arquivos. Atualmente, com a grande expansão da rede e a introdução de novas tecnologias, a arquitetura inicial já não se adequa mais à realidade. Redes heterogêneas, ambientes sem fio e redes auto-configuráveis são alguns dos novos cenários presentes atualmente. Com o acesso globalizado à rede, a Internet também se tornou mais insegura, necessitando de novos esquemas de autenticação e sigilo dos dados.

A arquitetura da Internet atual apresenta deficiências como a escassez de endereços IPv4 (*Internet Protocol* versão 4), as vulnerabilidades de segurança e a falta de suporte à mobilidade. Várias soluções pontuais foram propostas na literatura, como o *Network Address Translation* (NAT) [1], o *Internet Protocol Security* (IPSec) [2] e o *Mobile Internet Protocol* (MIP) [3]. Estas soluções atacam problemas específicos e nem sempre se integram de forma cooperativa, a exemplo da utilização do NAT com o IPSec, onde alguns modos do IPSec não funcionam em conjunto com o NAT.

Nos projetos de rede, a comunicação é organizada como uma pilha de protocolos com o objetivo de separar funcionalidades, tornando-as independentes umas das outras a fim de facilitar o desenvolvimento de novos serviços e protocolos de forma independente. Esta independência entre camadas nem sempre é respeitada. Um dos exemplos clássicos é o compartilhamento do *Internet Protocol* (IP) entre as camadas de rede e de transporte. Na camada de rede, ele atua como localizador (endereço IP) e na camada de transporte atua como identificador (*Transport Layer Identifier* - TLI). Como resultado, a conexão da camada de transporte somente perdura enquanto o endereço IP não mudar. Outra consequência desta sobrecarga semântica é a impossibilidade de integração de redes heterogêneas,

uma vez que não há um identificador comum entre as tecnologias adotadas em cada uma das redes.

A questão semântica relativa ao endereço IP mencionada no parágrafo anterior tem se tornado um denominador comum em praticamente todas as novas propostas voltadas para o desenvolvimento de uma nova arquitetura para a Internet. A estratégia básica utilizada nestas propostas caracteriza-se pela separação dos conceitos de identificação (quem) e localização (onde). O primeiro deve ter como característica básica o aspecto de persistência, ou seja, atuar como um identificador do nó (ou da aplicação em algumas propostas) independente, por exemplo, de questões topológicas ou de encaminhamento. A forma básica de oferecer persistência à identificação de um nó consiste em eliminar do identificador qualquer tipo de referência semântica. Isto implica na adoção de identificadores planos (*flat*) normalmente associados a um espaço de 128 bits (ou 32 bits para compatibilidade com o endereço IPv4).

O segundo conceito resultante da separação semântica diz respeito à localização e inclui os aspectos relativos à topologia (onde) e encaminhamento (como chegar). Alterações topológicas devidas à mobilidade, ou à conexão a mais de um provedor (*multihoming* [4]), implicam na mudança do localizador (neste texto usado como sinônimo de endereço) do nó mas não devem alterar a sua identificação.

Um outro aspecto importante associado à separação mencionada no parágrafo anterior é o fato de que, na maioria das propostas, o identificador é o resultado da aplicação de uma função de *hash* a uma chave pública vinculada ao nó [5]. Este fato abre perspectivas bastante interessantes para as novas arquiteturas do ponto de vista da segurança, tais como a possibilidade de verificação da identidade a partir do identificador da entidade.

O objetivo deste trabalho é propor uma arquitetura de implementação baseada em um novo modelo para interconexão de redes com o objetivo de oferecer novas soluções aos problemas mencionados anteriormente e contribuir com novas funcionalidades à Internet atual [6]. A arquitetura de implementação proposta nesta dissertação utiliza como referência o roteamento baseado no identificador do modelo *Node Identity Internetworking Architecture* [7]. Este modelo, definido no contexto do projeto *Ambient Networks* [8], foi escolhido como referência para este trabalho por possibilitar uma introdução gradual dos novos conceitos à arquitetura atual da Internet. A proposta utiliza também muitos dos conceitos relacionados aos identificadores criptográficos propostos pelo *Host Identity Protocol* [5].

A proposta de arquitetura de implementação oferece suporte à mobilidade, *multihoming*, integração de redes heterogêneas, segurança, aplicações legadas e oferece soluções para alguns dos problemas presentes na Internet como a escassez de identificadores globais. A arquitetura de implementação desenvolvida neste trabalho define a inserção de uma camada de identificação localizada entre a camada de transporte e a camada de rede. Esta camada soluciona o problema da sobrecarga semântica do endereço IP e oferece um ponto de ligação estável (persistência) para as aplicações possibilitando a troca dinâmica do localizador sem a quebra de conexão. Como resultado, a mobilidade e o *multihoming* se tornam nativos à arquitetura. A integração de redes heterogêneas é possibilitada pela utilização de *gateways* tradutores localizados nas bordas dos domínios heterogêneos. Os identificadores da camada de identificação possuem propriedades criptográficas, o que lhes concede um esquema de segurança nativo para a autenticação das identidades durante a comunicação. O modelo de segurança é baseado na *Public Key Infrastructure* (PKI) [9] e utiliza os certificados digitais para a autenticação das entidades. Por fim, a proposta de implementação de uma arquitetura de nova geração suporta as aplicações legadas, possibilitando a migração gradual entre as arquiteturas de forma transparente para

os usuários.

Além deste capítulo de introdução, o texto encontra-se organizado da seguinte forma: o Capítulo 2 apresenta os principais trabalhos e tecnologias relacionados com a dissertação; o Capítulo 3 discute a arquitetura proposta considerando os requisitos baseados nas arquiteturas descritas no Capítulo 2; o Capítulo 4 detalha a implementação da arquitetura e a avaliação do protótipo. Por último, o Capítulo 5 apresenta as conclusões finais e os possíveis trabalhos futuros.

Capítulo 2

Trabalhos e Tecnologias Relacionadas

Este capítulo discute os principais trabalhos relacionados à arquitetura de implementação proposta nesta dissertação e as várias tecnologias utilizadas. O capítulo encontra-se dividido em duas partes. A primeira apresenta as arquiteturas que mais influenciaram na definição da arquitetura, destacando os seus requisitos e as principais funcionalidades oferecidas. A segunda parte descreve as tecnologias empregadas no modelo de segurança da arquitetura de implementação.

A primeira proposta a ser apresentada é o *Mobile Internet Protocol* (MIP) [3] que contribui com os mecanismos de suporte à mobilidade em redes IP. A segunda proposta, denominada *Host Identity Protocol* (HIP) [5], cria um novo espaço de nomes chamado *Host Identity* de modo a desacoplar a identificação da localização, beneficiando as funções associadas à mobilidade e à segurança. O terceiro modelo discutido, *Node Identity Internetworking Architecture* (Node ID) [7], além de propor a separação dos aspectos de identificação e localização como no caso do HIP, abrange também a questão da heterogeneidade dos domínios.

2.1 *Mobile Internet Protocol*

O *Mobile Internet Protocol* (MIP) [3] é uma extensão ao IP para prover suporte à mobilidade através da auto-reconfiguração dos endereços, e da manutenção da conectividade da camada de transporte de forma transparente para as aplicações. Ao invés de modificar as aplicações para suportar a mobilidade, o MIP oferece um endereço IP estático, ou seja, independente da sua localização na rede. Isto permite às aplicações o estabelecimento de comunicação sem interrupção, mesmo durante os eventos de mobilidade. Para os usuários, o MIP mantém a continuidade de sessão das aplicações de rede conforme o usuário atravessa várias redes através da disponibilização de um endereço IP consistente para essas aplicações.

Componentes do MIP

Para auxiliar o suporte à mobilidade, o MIP define os seguintes componentes:

- *Mobile Node* (MN) - O MN consiste de um dispositivo móvel que se move entre diferentes redes e troca o seu endereço IP, mantendo a sua conectividade durante a mobilidade através do seu *Home Address*;

- *Home Network* - Rede onde o MN está associado originalmente;
- *Home Address* - É um endereço dado a um MN que é utilizado quando este se encontra na sua rede *Home* e através do qual é sempre alcançável, independente da sua localização na rede. Devido ao fato do MN estar associado ao seu *Home Address*, ele está dessa forma logicamente conectado à sua rede *Home*;
- *Home Agent* (HA) - O HA é um roteador na rede *Home* responsável pela manutenção dos registros dos MNs que estão fora da rede *Home* e dos seus atuais endereços IPs. Além disso, o HA é responsável pela interceptação de pacotes e o seu correto encaminhamento para o MN quando este se encontra fora do seu *Home*;
- *Foreign Network* - Rede que não é o *Home* de um MN;
- *Care-of Address* (CoA) - É um endereço topologicamente correto utilizado pelo MN quando este chega a uma rede estrangeira;
- *Correspondent Node* (CN) - O CN é um dispositivo, fixo ou móvel, que se comunica com o MN e não necessariamente precisa ser habilitado com o MIP;
- *Foreign Agent* (FA) - O FA é um roteador localizado nas redes estrangeiras responsável pelo armazenamento das informações referentes aos endereços CoA dos nós visitantes.

Suporte à mobilidade

O MIP oferece suporte à mobilidade através da disponibilização de dois endereços IPs: o *Home Address* e o CoA. O primeiro é utilizado como identificador do nó, sendo ele estático e oferecendo um ponto de ligação estável para as aplicações durante eventos de mobilidade. Dentro da sua rede *Home*, o MN utiliza o *Home Address* para o envio e a recepção de pacotes IP. O segundo endereço é utilizado pelo MN quando este visita uma rede estrangeira. O MN obtém o CoA a partir de mecanismos de descoberta de endereços como o DHCP ou de auto-configuração de endereços a partir de mecanismos de divulgação de prefixos, este último caso somente disponível em redes IPv6. O CoA possui o endereço IP topologicamente correto da rede visitada, possibilitando a alcançabilidade do MN na rede. Além disso, o CoA soluciona o problema da imutabilidade do endereço *Home Address*, que é utilizado como identificador das conexões. Dessa forma, as aplicações utilizam o *Home Address* como identificador da conexão e, quando os MNs se movem para uma rede estrangeira, os pacotes endereçados ao *Home Address* são capturados e tunelados pelo HA até o CoA atual do MN, posibilitando a mobilidade e ao mesmo tempo a manutenção da conexão das aplicações.

Um MN que se encontra na sua rede *Home* recebe diretamente os pacotes endereçados ao seu *Home Address*. Quando se move para uma rede estrangeira, o MN pode enviar e receber pacotes de duas maneiras: através do seu HA ou diretamente. O primeiro cenário, conhecido como túnel bidirecional, ocorre quando o CN não está habilitado com o *Mobile IP* em redes IPv6 (*Mobile IPv6* - MIPv6) ou o processo de registro com o CN (descrito a seguir) ainda não foi concluído. Dessa forma, o HA intercepta os pacotes e tunela-os com destino ao CoA do MN. No segundo cenário, também conhecido como otimização de rota, o CN está habilitado com o MIPv6 e o processo de registro já foi concluído. Dessa forma, os pacotes são entregues diretamente ao MN sem a necessidade do

encaminhamento pelo HA. Este cenário de otimização é viabilizado pelo registro do CoA do nó móvel no CN.

O processo de registro do MN com o CN consiste de dois procedimentos: o *Return Routability* e a troca de mensagens de *Binding Update* (BU) e *Binding Update Acknowledgement* (BA).

O procedimento de *Return Routability* permite a verificação da alcançabilidade do MN através do seu *Home Address* e do seu novo CoA. Este procedimento é necessário para a proteção contra ataques de segurança do tipo *Connection Hijacking* e negação de serviço (*Denial of Service*). Durante o procedimento de *Return Routability*, o MN envia dois diferentes pacotes de testes para o CN. Um deles é enviado através do seu HA e outro é enviado diretamento para o CN. Este recebe os dois pacotes e responde ao MN, cada um deles contendo um *token* criptográfico.

Após a conclusão do procedimento de *Return Routability*, o MN envia uma mensagem de BU para o CN contendo informações criptográficas utilizando os *tokens* enviados previamente pelo CN durante o *Return Routability*, possibilitando ao CN validar a informação. Caso a informação criptográfica seja verdadeira, o CN adiciona uma entrada no seu *Binding Cache* para o MN e envia uma mensagem BA concluindo o processo de registro correspondente.

Mobilidade no MIP

Para ilustrar o suporte do MIPv6, consideremos a Figura 2.1. Inicialmente, o MN se encontra na sua rede *Home* e inicia a comunicação com o CN utilizando o seu *Home Address* como identificador da comunicação, ilustrado na Figura 2.1 (a). No passo 1 da Figura 2.1 (b), o MN se move para uma rede estrangeira, onde ele obtém o seu novo CoA através de mecanismos de auto-configuração como, por exemplo, um serviço de divulgação de prefixos de roteadores, ou através de um servidor DHCPv6 presente na rede. No passo (2), o MN informa o seu novo CoA na rede estrangeira através do envio de uma mensagem de BU para o seu HA, que armazena na sua *Binding Table* e responde uma mensagem de BA informando o sucesso da atualização. No passo (3), o HA intercepta todos os pacotes endereçados para o *Home Address* do MN e utiliza o tunelamento IPv6-IPv6 para a entrega dos pacotes ao CoA do MN registrado e o MN inicia o processo de registro com o CN. Finalmente no passo (4), após concluído o processo de registro com o CN, o MN utiliza a mensagem de BU para atualização do seu CoA no CN para habilitar a funcionalidade de otimização da rota. Esta otimização evita a triangulação dos pacotes no HA, diminuindo a latência e o congestionamento, além de aumentar a escalabilidade do sistema. Esta última funcionalidade descrita somente é implementada na versão MIPv6, pois as redes IPv4 oferecem barreiras técnicas para tal funcionalidade, como por exemplo o NAT, que pode impedir a conectividade direta entre os nós após o evento de mobilidade.

Discussão do suporte à mobilidade do MIP

Para implementar a mobilidade, pode-se atuar de duas formas como descrito em [10]: mudança explícita do endereço IP ou a preservação do endereço e utilização de algum mecanismo de encaminhamento para a entrega de pacotes. A primeira abordagem redefine a função do IP para atuar como um simples localizador, e as camadas acima dele não poderão utilizá-lo como identificador, removendo esta interdependência entre camadas. A segunda abordagem, adotada pelo MIP, oculta a mudança do endereço IP. Dessa forma, um endereço IP, ou o seu *Home Address*, torna-se o seu identificador, enquanto a camada de rede opera sobre o seu endereço IP atual (CoA). Esta aborda-

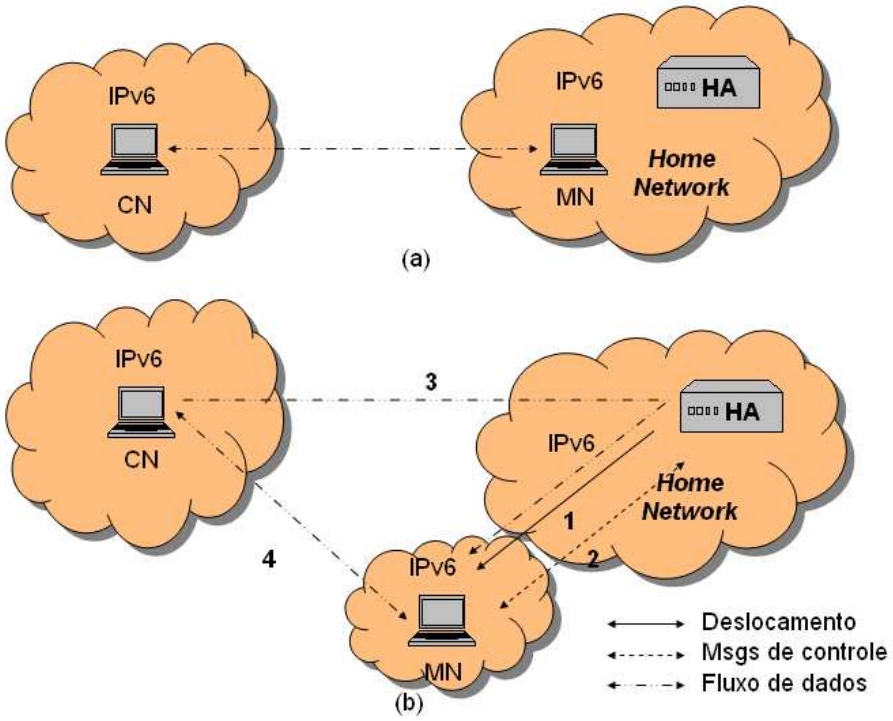

Fig. 2.1: *Mobile IP versão 6.*

gem, no entanto, ainda mantém a interdependência entre camadas, somente retirando a informação que a camada de rede provê para a camada de transporte. Como mencionado por [11], utilizar localizadores para entidades móveis continua sendo um problema fundamental, pois o propósito do IP é representar uma localização hierárquica na topologia de rede. O roteamento efetivo é baseado na hierarquia da rede e nos seus endereços. Uma deficiência que o MIP apresenta é a utilização do IP para representar uma localização que não corresponde à realidade quando o MN encontra-se fora da sua rede *Home*. Do ponto de vista da camada de transporte, isto cria a ilusão de que a localização do CN pode ser deduzida do seu *Home Address*. Um exemplo disso é que algumas funcionalidades da camada de transporte que monitoram o tráfego até o CN, como por exemplo, mecanismos de controle de congestionamento, são enganados ao considerar que o *Home Address* do destino realmente possui alguma informação topológica relativa à posição atual do CN.

O ponto forte do MIP é o seu suporte às aplicações legadas, que podem utilizá-lo de forma transparente. Isso se deve ao fato do MIP introduzir uma infraestrutura de rede adicional na Internet para o suporte à mobilidade, como o HA e o FA. Porém, a proposta apresenta algumas fraquezas relacionadas ao desempenho (triangulação), compatibilidade (IPv6) e segurança. Outra limitação do MIP é que ele não prevê a existência de múltiplos CoAs simultaneamente para suportar o *multihoming*.

2.2 *Host Identity Protocol*

O *Host Identity Protocol* (HIP) [5] é uma proposta de protocolo para arquiteturas de Internet de nova geração e propõe a criação de um novo espaço de nomes para a identificação das entidades na

Internet. Esta nova camada, chamada de *Host Identity*, está situada entre a camada de rede e a camada de transporte, e tem por objetivo identificar unicamente um entidade na Internet através da utilização de um identificador plano. Ao mesmo tempo, a introdução deste novo espaço de nomes soluciona o problema da sobrecarga semântica, possibilitando a introdução natural de novos serviços.

Um nome no espaço de nomes do *Host Identity*, um *Host Identifier* (HI), representa um nome global único, não existindo estatisticamente dois nós com o mesmo identificador [1]. O HI é a chave pública de um par de chaves pública/privada de uma entidade, adicionando propriedades criptográficas na identificação das entidades. Ao empregar a chave pública como identificador, uma entidade pode auto-afirmar a sua própria identidade através da emissão da sua assinatura digital, ou utilizar um mecanismo de terceiros para autenticá-la, como por exemplo o DNSSec [13].

O HI adiciona duas novas funcionalidades ao IP. A primeira corresponde ao desacoplamento da camada de rede da camada de transporte, possibilitando a evolução independente da camada de rede e abrindo caminho para o provisionamento de novos serviços fim-a-fim sobre vários domínios da Internet. A segunda funcionalidade refere-se à introdução de um mecanismo nativo de autenticação limitada das entidades na Internet. Devido à sua natureza criptográfica, o HI pode ser utilizado para a verificação das assinaturas digitais, garantindo a autenticidade da entidade emissora da assinatura. A autenticação é limitada pois ela somente garante a autenticidade do emissor da assinatura digital, por exemplo Bob, assumindo que somente Bob possui a chave privada. Porém ela não garante a autenticidade da identidade por trás do emissor, no caso se a identidade de Bob é realmente verdadeira, uma vez que não há uma entidade externa garantindo tal informação, como uma autoridade certificadora [9].

Uma entidade normalmente possui mais de um HI simultaneamente para ser utilizado na comunicação. Alguns dos HIs são publicados em serviços de diretório para acesso global e outros permanecem não-divulgados. Os nós utilizam o HI publicado para a localização e o contato inicial. Após o procedimento de autenticação, a entidade envia o HI não-divulgado para a entidade par, preservando a privacidade da comunicação entre as partes. Neste caso a privacidade é preservada pois o HI não-divulgado é gerado a partir de chaves públicas não-divulgadas pelo nó.

A introdução da nova camada de identificação soluciona o problema da sobrecarga semântica do IP, possibilitando às aplicações a utilização do identificador para identificação das conexões enquanto o localizador é utilizado para localizar topologicamente um nó na rede. Como resultado direto da resolução do problema da interdependência entre as camadas, as conexões são mantidas enquanto há a troca dinâmica do localizador. A Figura 2.2 (a) mostra a sobrecarga semântica do IP na arquitetura atual, que atua como identificador na camada de transporte e localizador na camada de rede. Com a inserção da nova camada de identificação, ilustrado na Figura 2.2 (b), a camada de transporte utiliza o HI para identificar a conexão, enquanto que o IP atua somente como localizador na camada de rede.

Uma vez que os HIs são derivados a partir das chaves públicas das entidades, eles podem possuir tamanhos variados conforme os parâmetros passados na geração do par de chaves público/privada (1024, 2048, etc). Dessa forma, com a finalidade de fixar o tamanho do HI para facilitar a codificação nos protocolos, criou-se o *Host Identity Tag* (HIT). O HIT é um identificador de 128 bits que representa o HI e é criado a partir do cômputo do *hash* criptográfico do HI. O HIT passa a ser utilizado

[1] Por estatisticamente únicos consideramos a baixa chance de colisão no espaço do HI. Baseado no paradoxo do aniversário [12], podemos esperar a chance de uma colisão, ou seja, dois identificadores com o mesmo HI, aproximadamente após $1.2 * sqrt(2^n)$ HI gerados. Para um espaço de HI de 128 bits, é esperado que a chance de colisão ocorra após a geração de aproximadamente 2^50 (1 quatrilhão) HIs

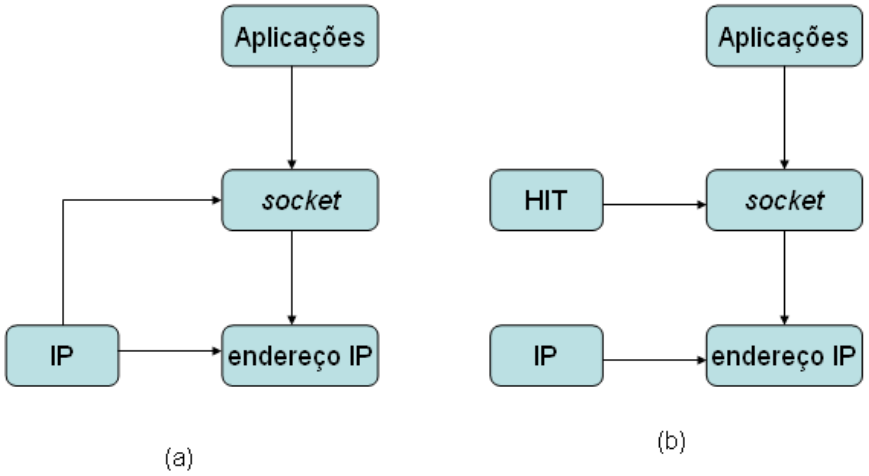

Fig. 2.2: Diagrama de camadas do TCP/IP.

como identificador dos pacotes, herdando as propriedades criptográficas do HI, como a possibilidade de autenticação limitada do identificador mencionada anteriormente.

A seguir serão apresentados os mecanismos de suporte à mobilidade e ao *multihoming* com a introdução do novo espaço de nomes criptográfico.

Mobilidade e *Multihoming*

O suporte à mobilidade ocorre de forma natural com a utilização do HIP, uma vez que a introdução da camada de identificação possibilita a troca de localizadores sem a quebra de conectividade. O HIP desacopla a camada de transporte da camada de rede e utiliza os HIs na identificação das conexões na camada de transporte. Consequentemente, o HIP provê suporte à mobilidade a um baixo custo de infraestrutura se comparado ao MIP discutido na seção anterior [5]. Além disso, o HIP provê o suporte natural ao *multihoming* ao possibilitar a ligação de um identificador a vários localizadores simultâneamente. Da mesma forma, um sistema é considerado *multihomed* se ele possuir mais que um IP roteável globalmente e simultaneamente. O HIP realiza a ligação de um identificador a vários localizadores simultaneamente, possibilitando a troca dinâmica entre eles. Se um dos localizadores tornar-se instável, as associações de transporte podem ser movidas para um outro localizador sem maiores complicações.

Após um evento de mobilidade, como por exemplo a mobilidade de um nó para uma rede estrangeira, o nó móvel envia uma mensagem de *Readdress* para o seu nó par. Esta mensagem tem por objetivo informar o seu novo localizador, possibilitando a restauração da comunicação entre os nós pares. A fim de evitar ataques de negação de serviço lançados contra as entidades, utiliza-se um mecanismo de autenticação da mensagem e verificação da alcançabilidade do novo localizador da entidade. A autenticação é necessária para evitar ataques de envenenamento dos mapeamentos, levando o nó a redirecionar o tráfego a uma possível vítima. A verificação da alcançabilidade, semelhante à funcionalidade de *Return Routability* do MIP, tem por objetivo verificar a validade do novo localizador, uma vez que um atacante pode capturar uma mensagem autêntica de *Readdress* e utilizá-la em

eventos futuros após a entidade ter se movido para uma outra localidade (ataque de repetição).

O estabelecimento da comunicação entre duas entidades inicia-se com o envio de uma mensagem destinado a um elemento da infraestrutura do HIP chamado de *Rendezvous Server* (RVS). O RVS armazena os mapeamentos de identificadores em localizadores e é responsável pelo encaminhamento da mensagem inicial para o estabelecimento da comunicação entre os nós pares. Além disso, o RVS provê o suporte à mobilidade simultânea dos nós, também conhecido como *double jump*.

Base Exchange

O procedimento de *Base Exchange* do HIP [14] realiza a autenticação das entidades e estabelece a chave simétrica a ser utilizada na criptografia dos dados através da troca de quatro mensagens de segurança representados por I1, R1, I2 e R2, ilustradas na Figura 2.3.

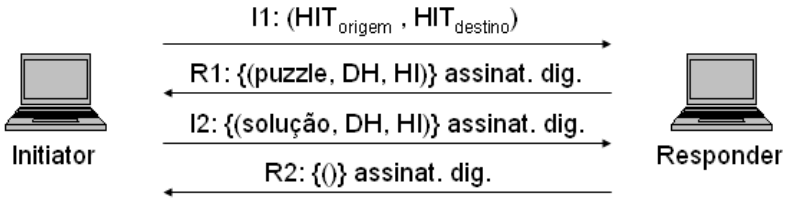

Fig. 2.3: *Base Exchange* do HIP.

O *Base Exchange* inicia com o envio da mensagem I1 pelo emissor, que aqui definimos de *Initiator* ou I. O seu correspondente é definido como *Responder*, ou R. O I inicia a troca de parâmetros com o envio da mensagem I1, que contém o HIT de origem e o HIT de destino. O R, ao receber a mensagem I1, responde com a mensagem R1, contendo os parâmetros *Diffie-Hellman* e o *puzzle*. O protocolo de troca de chaves criptográficas *Diffie-Hellman* será explicado na segunda parte deste capítulo.

O *puzzle* da mensagem R1 consiste de um desafio que requer a busca por um número aleatório que satisfaça a condição enviada por R. O objetivo do *puzzle* é evitar ataques de negação de serviços lançados contra as entidades, retardando o envio de novas mensagens através do lançamentos de desafios. O *puzzle* consiste no envio de um número aleatório i juntamente com o nível de dificuldade k. O nó *Initiator* (I), ao receber os parâmetros i e k, deverá computar o *hash* criptográfico da concatenação dos parâmetros $i + HIT_{origem} + HIT_{destino} + j$ e encontrar um j tal que o *hash* sobre os parâmetros gere k bits iguais a zero. O parâmetro k pode ser ajustado conforme o grau de confiança com a entidade, ou parâmetros da rede como congestionamento, entre outros. Criado o *puzzle*, o emissor I envia a mensagem R1 contendo o seu HI, o *puzzle* recém-criado e assina digitalmente a mensagem. O envio do HI, juntamente com a mensagem, possibilita a verificação da assinatura digital do emissor sem recorrer a entidades externas em busca da chave pública associada.

Ao receber a mensagem R1, o I verifica a assinatura digital e busca a solução do *puzzle*. Ao encontrar a solução j, I computa a chave simétrica a partir dos parâmetros *Diffie-Hellman* recebidos e então envia a mensagem I2 contendo a solução do *puzzle*, os parâmetros *Diffie-Hellman* e o seu HI, assinando digitalmente a mensagem. R, ao receber a mensagem, computa a chave simétrica a ser utilizado pelo modelo de segurança do HIP a partir dos parâmetros *Diffie-Hellman* e responde com a mensagem R2 que é assinada digitalmente, concluindo o procedimento de *Base Exchange*.

A mensagem R2 contém o HMAC do pacote HIP, e tem por objetivo evitar ataques de negação de serviço.

Segurança

O *Base Exchange*, como descrito anteriormente, é um mecanismo de troca de parâmetros de segurança sobre um canal inseguro de comunicação. O modelo de segurança do HIP utiliza como base o *Internet Protocol Security* (IPSec) [2], herdando as suas principais características e funcionalidades. O IPSec consiste de um conjunto de protocolos que visa garantir a autenticação, o sigilo e a integridade dos dados na camada de rede. A opção pelo IPSec como solução de segurança se deve ao fato da possibilidade de utilização no HIP de algumas estruturas de segurança do IPSec de forma integrada ao protocolo. Na realidade, o *Base Exchange* proposto é uma versão mais leve do protocolo de troca de chaves do IPSec que estabelece uma chave simétrica entre duas entidades através de um canal inseguro utilizando o protocolo *Diffie-Hellman*.

As associações de segurança do IPSec (*Security Asssociation* - SA), que definem as políticas a serem aplicadas a determinadas entidades, são atreladas aos HIs das entidades no lugar dos seus localizadores. Na prática, o HIP realiza uma troca de chaves utilizando o HIT para inicializar o par de SAs e habilitar o *Encapsulating Security Payload* (ESP) no modo fim-a-fim. Os canais de comunicação com os nós pares permanecerão abertos uma vez que os HITs são estáticos, suportando naturalmente a mobilidade e o *multihoming* sem a necessidade de reestabelecimento dos parâmetros de segurança. Cenários de perda de conectividade momentânea, como por exemplo durante um evento de mobilidade, não requerem o reestabelecimento das políticas de segurança entre os nós pares. Caso o período de desconexão seja longo, as SAs podem expirar, necessitando a renegociação dos parâmetros de segurança do IPSec baseados nas políticas de segurança negociados pelo HIP, sendo esta uma característica do IPSec que requer a renovação periódica dos parâmetros de segurança a fim de evitar ataques de como, por exemplo, a quebra da chave simétrica por força bruta.

A arquitetura de segurança do HIP provê suporte a dois tipos de protocolos criptográficos herdados do IPSec para garantir a segurança: o *Authentication Header* (AH) e o ESP. O AH provê a autenticação das entidades e garante a integridade dos dados através da utilização da assinatura digital no pacote. O ESP, além de prover a autenticação e a integridade, garante o sigilo dos dados, cifrando a carga do pacote IP.

HIP e NAT

A adoção do IPSec como solução de segurança para a Internet é retardada devido a vários fatores, entre eles a introdução dos *middleboxes* e especificamente o NAT. O NAT cria sub-redes privadas a partir de um ou mais endereços IPs da rede, realizando a translação do cabeçalho IP da rede externa para a interna e vice-versa. Um dos modos do IPSec, o modo transporte, utilizado em conjunto com o AH, garante a autenticidade e a integridade do pacote através do cômputo do *hash* criptográfico das informações do cabeçalho IP, garantindo a integridade até o destino. Como o NAT realiza a translação dos IPs internos e externos e o IPSec não possibilita a modificação do cabeçalho IP, então o NAT não é compatível com este modo de operação especificamente.

Como o HIP utiliza o IPSec como base do seu modelo de segurança, é natural que o HIP seja conflitante com o NAT. Para solucionar este problema, o protocolo propõe a utilização de NATs

sensíveis ao IPSec, isto é, ele deve ser capazes de realizar a translação entre os HITs (que o NAT entende como endereços IPs) nos seus correspondentes SPIs (*Security Parameter Index*) e depois mapear os HITs em IPs. Os SPIs são índices utilizados pelo IPSec para identificar as políticas de segurança associadas a determinados pacotes IP que chegam ao destino. Vários HITs podem ser mapeados para um único endereço IP em um NAT, simplificando as conexões em uma interface NAT com poucos endereços. Neste caso, o NAT não deverá alterar os datagramas dentro do envelope IPSec, sendo necessário a translação de endereços de aplicativos nos sistemas finais.

Uma vantagem herdada da utilização de HITs em ambientes com NATs é a preservação da conectividade fim-a-fim. Em ambientes onde a identificação é baseada no endereço IP, a identificação é ocultada pelo NAT. Com o HIP, a camada de transporte se associa aos HIs resultando na possibilidade da conectividade entre *hosts* passar por vários domínios sem perder a identificação. Neste caso, os endereços IPs são utilizados apenas com o propósito de roteamento.

Análise do HIP

O HIP introduz uma camada de identificação localizada entre a camada de rede e a camada de transporte com a finalidade de resolver a interdependência entre as camadas. Do ponto de vista da camada de rede, o HIP retira a sobrecarga do gerenciamento da mobilidade de forma transparente para as camadas superiores. Na camada de transporte, o HIP cria um identificador confiável, o HIT, que realmente representa o destino onde pode ser criada uma conexão fim-a-fim. Além disso, a criação de identificadores também facilita as questões de segurança, pois as identidades podem ser autenticadas através de assinaturas digitais das entidades.

Comparado com a solução proposta pelo MIP, que utiliza dois endereços IP e oculta a mobilidade para as aplicações no HIP, ainda persiste o problema da obtenção de informações da rede a partir dos identificadores. Isso se deve ao fato das aplicações legadas não diferenciarem o conceito de identificador e localizador. Neste caso, as aplicações utilizam o endereço IP para identificar a conexão e ao mesmo tempo para a obtenção de informações topológicas da rede. Como consequência, a aplicação legada que recupera informações de congestionamento da rede ainda apresenta o mesmo problema do MIP. Porém, o HIP elimina a confusão conceitual entre um endereço IP que representa a sua identidade e a sua localização simultâneamente.

Uma outra limitação do HIP é a necessidade de múltiplos HITs na comunicação com uma mesma entidade no caso da necessidade de múltiplos canais de comunicação. Esta limitação é uma herança do IPSec, que requer um par de endereços IPs associados a um conjunto de políticas de segurança. Caso sejam necessárias outras políticas ligando as duas entidades, então será necessário um outro par de HITs paralelos.

2.3 *Node Identity Internetworking Architecture*

A *Node Identity Internetworking Architecture* (NodeID) [7] propõe uma arquitetura para a Internet de nova geração com a introdução de uma camada de identificação na pilha de protocolos de rede. A camada de identificação possui propriedades criptográficas e é utilizada na identificação dos nós. Além disso, ela é utilizada para prover novos serviços à Internet, como o roteamento baseado na identidade, a segurança e o suporte à mobilidade e ao *multihoming*.

Premissas

A arquitetura NodeID apresenta algumas premissas, como a existência de domínios independentes, a conexão dinâmica entre domínios e a não-distinção entre consumidores e encaminhadores de tráfego.

A primeira premissa estabelece que cada domínio, também chamado de *Locator Domain* (LD), possui um sistema interno de endereçamento e roteamento. Os nós internos de um LD podem se comunicar livremente recorrendo somente aos serviços internos do LD. O requisito fundamental é a independência de uma entidade externa ao LD para a comunicação interna, garantindo a existência de domínios autônomos e possibilitando a sua mobilidade.

A segunda premissa é a conexão dinâmica entre os domínios, possibilitando cenários de mobilidade e *multihoming*. Nós pertencentes a um LD podem migrar para um outro LD e a conectividade é mantida através de mecanismos de rastreio e ligação da identificação do nó com o seu correspondente localizador da tecnologia do LD visitado.

A terceira premissa é a igualdade de funcionalidade entre os nós e os roteadores no encaminhamento dos pacotes. Os nós podem encaminhar temporariamente, ou permanentemente,o tráfego de pacotes, dependendo do cenário dinâmico da topologia. Exemplos incluem dispositivos móveis que atuam como roteadores em redes pessoais (PANs).

Assumindo as premissas apresentadas anteriormente, a arquitetura NodeID simplifica o problema para o provisionamento de conectividade fim-a-fim entre nós de uma federação de LDs com topologias dinâmicas.

Modelo de Roteamento

A arquitetura NodeID apresenta um modelo de simplificação da Internet para auxiliar no roteamento baseado no identificador, como ilustra a Figura 2.4. A arquitetura define a existência de um *Core* IPv4 no topo da hierarquia onde estão localizados os nós estáticos representando a Internet atual. Todas as entidades da arquitetura possuem um identificador criptográfico associado chamado de *Node Identifier* (NID), que é gerado a partir do *hash* criptográfico da chave pública do nó.

Para auxiliar o roteamento no *Core*, a arquitetura prevê a utilização de uma *Distributed Hash Table* (DHT) que provê o serviço de resolução de identificadores dos *gateways* ligados ao *Core* (NID-GW) em endereços IPv4, possibilitando o roteamento baseado nos identificadores dos *gateways* de *Core*. Além disso, a arquitetura também prevê a utilização do serviço de resolução de nomes *Domain Name Service* (DNS) para a resolução de *Fully Qualified Domain Names* (FQDNs) em identificadores (NIDs) e localizadores de um nó da arquitetura.

A conexão entre os diferentes domínios é feita através dos roteadores NID (*NID Routers* - NR), que são responsáveis pelo encaminhamento de pacotes e pela tradução de diferentes tecnologias de rede entre os domínios.

O processo de configuração de um nó ao chegar a um domínio inicia-se com a obtenção dos parâmetros do domínio através de mecanismos específicos da tecnologia de rede, como por exemplo, a obtenção dos localizadores através de um servidor DHCP (*Dynamic Host Configuration Protocol*). O segundo passo é o registro permanente do nó no DNS, enviando as seguintes informações ao servidor DNS do domínio, que será o seu domínio *Home*:

- *Fully Qualified Domain Name* (FQDN);

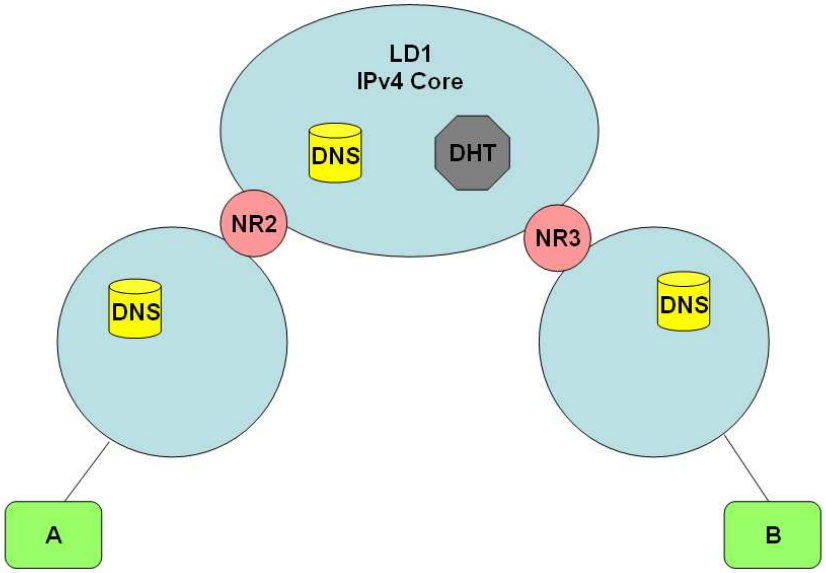

Fig. 2.4: Arquitetura Node ID.

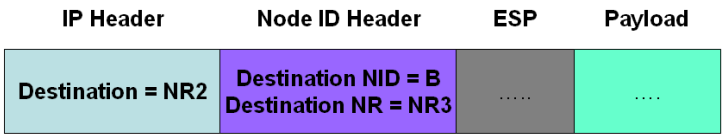

Fig. 2.5: Cabeçalho Node ID.

- NID;
- NID-GW;
- Localizador.

Os registros contendo as informações acima são propagados através dos NRs até o *Core*, informando os NRs intermediários sobre o identificador e o *Gateway* antecessor sob o qual o nó está registrado, possibilitando a alcançabilidade do nó na rede. Todos os NRs ligados ao *Core* chamados de *Gateways* de *Core* são estáticos e possuem os seus identificadores e localizadores registrados na DHT, que está localizada no *Core* para auxiliar no roteamento baseado na identidade. Pacotes destinados à outra árvore demandam a resolução do NID-GW do destino no localizador IPv4 do NR respectivo ligado ao *Core*.

Para ilustrar o roteamento baseado na identifidade, consideremos a Figura 2.4 na qual o nó A deseja se comunicar com o nó B. O estabelecimento da comunicação se inicia com a resolução do nome do nó alvo no servidor DNS. O nós A e B possuem os seus nomes, NID e NID-GW registrados nos roteadores NID dos seus domínios, NR2 e NR3, respectivamente. O nó A resolve o FQDN do nó B utilizando o serviço DNS, obtendo como resposta o par <NID, NID-GW> referente ao nó B.

O nó envia o primeiro pacote para o NR do domínio, no caso NR2, que a princípio desconhece o localizador do NR de *Core* do destino. O NR2 realiza a resolução do NID-GW do nó B na DHT e recupera o endereço IPv4 do NR3 ligado ao *Core*. NR2 então encaminha os pacotes para o NR3, que possui o registro do nó B devido ao procedimento de registro estático, encaminhando o pacote ao seu destino.

Roteamento Híbrido

A arquitetura Node ID propõe a utilização de um esquema de roteamento híbrido, utilizando a tecnologia local de roteamento, quando ambos os nós estão no mesmo domínio, ou quando há domínios com a mesma tecnologia ligando ambos os nós. Como cada domínio é independente, o roteamento interno não depende de entidades externas, como definido na premissa. Quando a comunicação entre os nós passa por um domínio com rede heterogênea, utiliza-se o roteamento baseado no identificador. Os NRs criam uma rede sobreposta na infraestrutura de rede existente e o roteamento é realizado sobre a identidade. A vantagem de se utilizar um roteamento híbrido é a herança dos mecanismos de roteamento pré-existentes e a interligação dos nós que utilizam tecnologias de rede diferentes. Baseado neste mecanismo, os NRs suportam a mobilidade assim como a comunicação entre domínios heterogêneos utilizando um roteamento baseado nos identificadores.

O esquema de roteamento híbrido na arquitetura Node ID divide-se em três regiões distintas: as pontas (árvores ligadas ao *Core*), o *Core* e o roteamento entre *Cores*. Na primeira região, todos os domínios possuem rota *default* para o *Core*. Os domínios na ponta se registram em um NR e este propaga os seus registros até chegar ao NR ligado ao *Core*. Este padrão elimina a necessidade de um esquema de roteamento global para suportar topologias complexas nas pontas, assumindo que nessa região há um número limitado de nós.

Na segunda região, dentro do *Core*, o roteamento é mais complexo do que nas pontas pois não existem rotas *default*. Neste caso, utiliza-se uma DHT para a resolução dos NRs ligados ao *Core* em endereços IPv4 para o roteamento. A utilização do NID-GW no roteamento estabelece uma forma de roteamento com rota na origem. Se o roteamento não utilizasse o NID-GW, a DHT precisaria armazenar todos os NIDs dos nós, não sendo uma opção escalável globalmente.

A terceira região, entre os múltiplos *Cores*, o roteamento utiliza a DHT para a encontrar o endereço IP do NR de acesso ao NR de destino. Dessa forma, todos os NR do núcleo devem manter rotas a outros NRs que levam a outros *Cores*.

Resolução de Identificador

A resolução do identificador na arquitetura Node ID inicia com a resolução do FQDN via o DNS. Se a requisição ao DNS retornar um localizador, então o mecanismo de comunicação é específico do domínio, indicando que o nó alvo não suporta as funcionalidades da arquitetura Node ID. Dessa forma, é necessário que o nó alvo esteja no mesmo domínio que o nó emissor, ou que existam domínios homogêneos até o destino. Se a requisição ao DNS retornar um NID, então a comunicação entre domínios heterogêneos é possível devido ao roteamento baseado na identidade. Caso o DNS retorne mais de um NID, então ele estabelece uma rota na origem fraca. As consultas dentro de um domínio resultam em um mapeamento entre o FQDN, o seu NID e localizador daquele domínio. Isso se estende para toda a árvore da topologia, permitindo a alcançabilidade de todos os nós da árvore.

Para os outros nós, utiliza-se uma rota *default* em direção ao *Core* até chegar ao NR ligado ao *Core*, que realiza a consulta à DHT para encontrar o localizador do NR de destino ligado ao *Core*.

Mobilidade e *Multihoming*

O suporte à mobilidade e ao *multihoming* se deve à atuação dos NRs, que ocultam os localizadores dos nós internos ao LD, divulgando somente o localizador do NR acessível externamente. Como resultado, os NRs ocultam a mudança de localizador, possibilitando a mobilidade de forma natural. Além disso, os NRs permitem o registro de NIDs associados a vários localizadores simultâneamente, possibilitando o serviço de *multihoming*.

Segurança

A segurança na arquitetura Node ID atua basicamente sobre a camada de identidade. Cada nó da arquitetura pode possuir vários NIDs, sendo um deles registrado no DNS para que ele seja alcançável. Os outros são utilizados somente na comunicação com o propósito da manutenção da privacidade. Dentro do domínio, o localizador é ocultado pelo NR. A proteção da comunicação fim-a-fim é baseada nos NIDs. Todo o tráfego trocado entre os nós utiliza um esquema de criptografia e, caso haja mudança de localizador devido a um evento de mobilidade, o nó móvel atualiza o seu localizador junto ao seu NR com uma simples autenticação, para evitar ataques de negação de serviço (*Denial of Service* - DoS). Caso não existisse esta autenticação, um nó atacante poderia enviar várias atualizações de localizador referente a um nó alvo para o NR, e todo o tráfego de outros nós seriam direcionados a esse nó. Caso um ataque de DoS seja lançado contra um nó, este pode solicitar o bloqueio do tráfego, ou até mesmo o desregistro do NID no NR do domínio e registrar uma outra identidade reserva para a comunicação.

Extensões da arquitetura Node ID

A arquitetura Node ID propõe vários mecanismos que podem ser implementados conforme a necessidade dos requisitos dos domínios. Um exemplo é a possibilidade de criação de uma rede sobreposta entre os NRs de um domínio, possibilitando o balanceamento de carga e consequente aumento da resistência a ataques de DoS e a tolerância a falhas. Nós que possuem vários NIDs podem solicitar à rede sobreposta de NRs a desativação ou o bloqueio do tráfego a determinados NIDs a fim de evitar ataques. Os NRs possibilitam outros meios de comunicação, como o *multicast* e o *anycast*, através do encaminhamento de pacotes para múltiplos destinos. Neste caso, é possível a utilização de certificados de delegação de identidade ou a utilização de um identificador de grupo. O certificado de delegação é uma mensagem que possui a assinatura digital do nó responsável autorizando determinado nó a integrar o grupo ou a receber ou enviar mensagens em seu nome.

Análise da arquitetura Node ID

A arquitetura NodeID contribui com um modelo de roteamento baseado na identidade através da simplificação do modelo da Internet. O NodeID considera a existência de um *Core* representando a Internet atual e domínios que se ligam dinâmicamente a este *Core*, como por exemplo as PANs. A utilização da camada de identidade cria uma rede sobreposta sobre a camada IP existente e possibilita a

introdução de novos serviços, como mobilidade e *multihoming*. A mobilidade é herdada naturalmente uma vez que as conexões são baseadas no identificador e estes são estáticos mesmo durante eventos de mobilidade. O *multihoming* também é herdado naturalmente uma vez que uma identidade pode estar associada a mais de um localizador simultaneamente viabilizando o balanceamento de carga.

No entanto, a arquitetura Node ID apresenta algumas deficiências, como a utilização do HIP como solução de segurança e o DNS para o armazenamento de informações relativas à localização dos nós. A Node ID propõe a utilização do HIP como solução de segurança uma vez que ele possui mecanismos bem definidos de segurança baseados no IPSec. No entanto, a arquitetura NodeID assume a existência de redes heterogêneas o que inclui redes não-IP. Dessa forma, a utilização do HIP não é compatível com o modelo de redes heterogêneas proposta pela arquitetura NodeID. Outra deficiência da arquitetura é a utilização do DNS para o armazenamento de informações dinâmicas relativas a localização dos nós. Como descrito em [15], o DNS não é adequado para o armazenamento de informações dinâmicas como os localizadores, devido à característica transiente da informação, sendo necessário uma outra infraestrutura que auxilie na resolução de identificadores em localizadores.

2.4 Outras Propostas de Arquitetura

Outras propostas analisadas e que servem como fundamento teórico para este trabalho são:

- *Towards Autonomous Network Domains* (TurfNet) [16];
- *Plutarch* [17];
- *Internet Indirection Infrastructure* (i^3) [18];
- *FARA: Reorganizing the Addressing Architecture* [19];
- *Host Identity Indirection Infrastructure* (Hi^3) [20];
- *Routing on Flat Labels (ROFL)* [21];
- *Overlay Convergence Architecture for Legacy Applications* (OCALA) [22].

A arquitetura TurfNet propõe a introdução de domínios autônomos, onde cada domínio possui mecanismos internos próprios de endereçamento, protocolos de roteamento, mecanismos de resolução de nomes e plano de controle. A comunicação entre as entidades se dá através da utilização de nomes e identificadores únicos e globalmente alcançáveis. Cada domínio, também chamado de *Turf*, possui *gateways* que são responsáveis pelas funcionalidades de busca por identificadores, tradução de tecnologias entre os domínios e a composição dos mesmos. No TurfNet, os *Turfs* podem de mover e se compor dinâmicamente, abrindo novas possibilidades, como por exemplo, o compartilhamento de recursos entre os mesmos.

O *Plutarch* propõe uma nova arquitetura de interconexão de redes que herda as arquiteturas já existentes, como a Internet, e assume a introdução de novos domínios heterogêneos. Este novos domínios heterogêneos podem se comunicar através do conhecimento do contexto dos domínios e das funções intersticiais que as ligam. O contexto define o conjunto básico de funcionalidades a serem

trocadas para habilitar a comunicação, tais como o endereçamento, o formato dos pacotes, protocolos de transporte e o serviço de nomes. As funções intersticiais são responsáveis pela translação dos parâmetros relativos ao endereçamento, roteamento, transporte e nomes entre os domínios do *Plutarch*.

O i^3 introduz uma infraestrutura de rede sobreposta para o provisionamento de novos serviços, tais como a mobilidade, o *multicast*, o *anycast* e a composição de serviços. Estes serviços são provisionados através da introdução de identificadores planos utilizados para a identificação das entidades na Internet. As entidades enviam os dados para os servidores i^3 da infraestrutura destinados a um identificador, e os servidores são responsáveis pelo correto encaminhamento dos pacotes ao seu destino final via roteamento IP. Ao desacoplar o ato do envio da recepção, o i^3 abre possibilidades de novos serviços de forma natural, como por exemplo a mobilidade. Neste caso, um nó, após um evento de mobilidade, atualiza o seu endereço IP no i^3, possibilitando a entrega dos pacotes e sem a quebra de conectividade.

O FARA (*Forwarding directive, Association, and Rendezvous Architecture*) propõe uma nova organização dos conceitos da arquitetura de redes baseado no desacoplamento dos nomes dos endereços de rede das entidades. O modelo FARA define um conjunto abstrato de componentes, como entidade, associação e substrato de comunicação, e a relação entre eles, deixando indefinidos vários detalhes técnicos ou decisões de projetos dos mecanimos que implementam o modelo. Como resultado, o FARA define uma classe de arquiteturas da qual uma variedade de outras arquiteturas específicas podem ser instanciadas através da adição e definição dos mecanismos da nova arquitetura.

O Hi^3 é a arquitetura resultante da integração das propostas HIP e i^3 e provisiona o suporte à mobilidade fim-a-fim do HIP em conjunto com ao suporte à mobilidade indireta do i^3. O Hi^3 propõe a utilização do i^3 para a correta entrega das mensagens de sinalização do HIP e o estabelecimento dos parâmetros de segurança, e utiliza o roteamento IP para a entrega fim-a-fim dos dados protegidos com o IPSec. Como resultado, há a combinação da mobilidade fim-a-fim provido pelo HIP em conjunto com a mobilidade indireta pelo i^3, resultando em maior eficiência com a eliminação das triangulações na entrega dos pacotes e maior robustez com a utilização da infra-estrutura de indireção i^3.

O ROFL propõe a retirada da semântica de localização da camada de rede, recorrendo somente ao roteamento baseado no identificador. Para isso, cada AS (*Autonomous System*) possui uma DHT utilizando o algoritmo de roteamento *Chord* [23] onde estão registrados todos os identificadores dos nós sob aquele AS. Estes anéis *Chord* se compõe formando uma DHT hierárquica como descrito em [24] que possibilita o roteamento baseado no identificador entre os AS. O roteamento proposto utiliza o algoritmo guloso, ou seja, o identificador é roteado em direção ao identificador mais próximo do destino, nunca o antecessor.

O OCALA é uma arquitetura de convergência de redes sobrepostas que permite às aplicações legadas utilizar as funcionalidades das novas arquiteturas e de serviços de redes sobrepostas sem nenhuma modificação do código fonte, recompilação ou reconfiguração. OCALA possibilita aos usuários acessar simultaneamente múltiplas redes sobrepostas para diferentes propósitos, agrupando as diferentes funcionalidades, assim como a comunicação das entidades que residem ou utilizam diferentes tipos de redes sobrepostas.

2.5 Tecnologias de Segurança

Esta seção apresenta as tecnologias relacionadas ao modelo de segurança para a arquitetura proposta neste trabalho. O modelo utiliza o *Public Key Infrastructure* [9] para a autenticação das entidades através dos certificados digitais, e o protocolo de troca de chaves criptográficas *Diffie-Hellman* para a troca dos parâmetros de segurança.

2.5.1 *Public Key Infrastructure*

O *Public Key Infrastructure* (PKI) é um *framework* que consiste de um conjunto de serviços de segurança que provê suporte para a criptografia de chaves públicas no gerenciamento de chaves e certificados. O PKI provê mecanismos para o estabelecimento de comunicação com sigilo, integridade dos dados e autenticação sem uma troca prévia de chaves ou contato anterior entre as entidades em ambientes tipicamente inseguros como a Internet. O PKI possui três modelos de segurança: hierárquico, distribuído e direto.

O modelo hierárquico é composto por uma série de *Certificate Authorities* (CA) que se organizam de forma hierárquica: no topo da hierarquia há uma CA global que assina digitalmente os certificados de outras CA regionais. O modelo hierárquico pode possuir somente uma hierarquia de CA, tornando-se plano. Há uma relação de custo-benefício entre a escolha do modelo hierárquico de camadas e o plano. No modelo de camadas, a vantagem é a regionalização das CAs, isolando o risco de comprometimento da segurança global no caso de um ataque a uma CA. Porém, a manutenção é mais complexa uma vez que há dependências entre as CAs na hierarquia. No modelo plano com uma única CA, a administração é mais simples, porém, o comprometimento da CA inviabiliza todo o sistema.

O modelo distribuído, também conhecido como *Web of Trust*, utiliza graus de confiança para a tomada de decisão na aceitação dos certificados no lugar de CAs externas. A aceitação ou recusa de um certificado baseia-se no critério de confiança dado às entidades endossantes do certificado. Certificados emitidos por entidades desconhecidas podem ser endossadas por outras entidades para garantir a associação de uma chave pública com uma identidade. Um exemplo de regra do *Web of Trust* para a aceitação de um certificado digital é o seu endosso por uma entidade confiável ou por duas entidades conhecidas.

O modelo direto utiliza o compartilhamento prévio de uma chave secreta entre os pares para o estabelecimento de comunicação segura entre as entidades. Este modelo também não utiliza a CA para a associação de uma chave pública com uma identidade.

Componentes da PKI

A PKI possui três componentes principais: a *Certificate Authority* (CA), a *Registration Authority* (RA) e o repositório de certificados .

A Autoridade Certificadora (*Certificate Authority* - CA) é uma entidade confiável que certifica a identidade de uma entidade através da verificação de informações que comprovem a autenticidade da identidade. A CA associa uma chave pública a uma identidade por meio de um certificado que é assinado digitalmente pela CA.

A Autoridade Registradora (*Registration Authorization* - RA) é opcional na PKI e é utilizada para realizar algumas etapas que são normalmente realizadas pela CA. A CA delega autoridade para a

RA verificar a identidade de uma entidade para a geração de um certificado digital. Para sistemas pequenos, a CA desempenha o papel da RA.

O repositório de certificados é um repositório público gerenciado por uma CA onde são publicados os certificados. Entidades que desejam verificar a autenticidade de uma identidade podem acessar o repositório público para averiguar se um certificado foi revogado antes de seu vencimento. O repositório é gerenciado por uma CA para garantir a segurança das modificações e evitar corrupções na base de certificados. A Figura 2.6 ilustra o procedimento de utilização de uma PKI. No passo (1), o nó A envia a solicitação de requisição de certificado (*Certificate Signing Request* - CSR) para a CA, que verifica a autenticidade das informações e lhe devolve um certificado. No passo (2), a CA armazena o certificado digital no seu repositório público. O nó B, que deseja contactar o nó A, busca pelo seu certificado digital no diretório público (3) e extrai a chave pública para enviar uma mensagem secreta para A (4). Como somente o nó A possui a chave privada que é par da chave pública contida no certificado só ele pode abrir a mensagem secreta e ler o seu conteúdo.

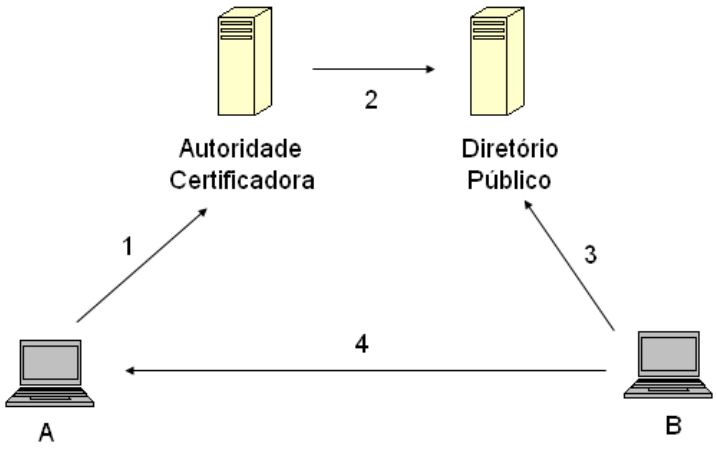

Fig. 2.6: Infraestrutura de chaves públicas.

Os serviços de segurança oferecidos pelo PKI são:

- sigilo: garantia de privacidade dos dados provido por mecanismos de cifragem;
- integridade: dados e transações são protegidos contra corrupções ou modificações;
- autenticação: utilização de certificados digitais para garantir a origem das informações;
- não-repúdio: assinatura digital garante o não-repúdio das mensagens, pois somente a entidade dona da chave privada pode emitir uma assinatura digital.

A criptografia de chaves públicas utiliza a assinatura digital para a autenticação das entidades. A assinatura digital consiste na cifragem de uma mensagem em texto claro com a chave privada da entidade. As entidades que recebem estas mensagens podem verificar a autenticidade das mesmas sem a troca prévia de informações ou chaves secretas por meio da decifragem da mensagem com a chave pública da entidade, pois somente o portador da chave privada poderia ter cifrado a mensagem

original. Como algoritmos de chave pública costumam ser lentos, não é costume a aplicação dos mesmos sobre textos inteiros, mas sim sobre o resultado de funções *hash* aplicada sobre os textos. Asso, a assinatura digital de um certificado consiste em aplicar uma função de *hash* criptográfico sobre as informações do certificado digital, cifrá-lo com a sua chave privada e adicionar ao certificado digital. Para verificar a autenticidade do certificado digital, a entidade aplica um *hash* critptográfico sobre as informações contidas no certificado, decifra a assinatura digital do certificado e compara o *hash* da mensagem original com a enviada junto com o certificado. Se ambos são idênticos, então a mensagem é autêntica e íntegra.

O certificado digital possui informações que atestam a ligação entre uma chave pública e uma entidade. As informações necessárias para a construção de um certificado digital são o nome da entidade, informações de identificação como cidade, estado, tempo de validade e a chave pública.

2.5.2 Protocolo de troca de chaves *Diffie-Hellman*

A troca da chave simétrica a ser utilizada na integridade e confidencialidade das mensagens é realizada através do protocolo de troca de chaves criptográficas *Diffie-Hellman* (DH). O protocolo DH permite a troca de chaves entre duas entidades através de um canal inseguro de comunicação. O procedimento de troca de chaves inicia com a escolha de um número primo p e um gerador g, como mostra a Figura 2.7. O nó A escolhe um número aleatório a e envia em texto claro os parâmetros p, g, e $g^a mod\ p$ para B. B recebe os parâmetros e escolhe um número aleatório b e retorna para A o valor de $g^b mod\ p$. A computa $(g^b)^a mod\ p$ enquanto B computa $(g^a)^b mod\ p$, resultando em uma mesma chave simétrica trocada entre as partes igual a $g^{ab} mod\ p$. A segurança neste protocolo de troca de chaves está na dificuldade e se calcular a mesmo conhecendo-se p e g em $g^a mod\ p$, problema conhecido como cálculo do logaritmo discreto. Como o atacante não conhece a e b, ele não tem como descobrir $g^{ab} mod\ p$ facilmente, especialmente se o valor de p for grande.

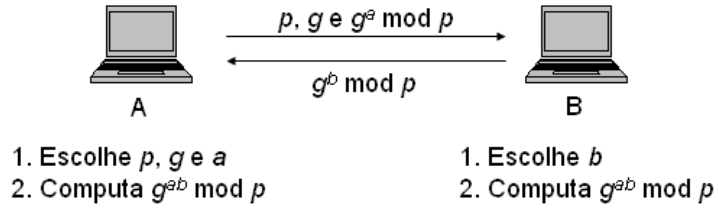

Fig. 2.7: Protocolo de troca de chaves *Diffie-Hellman*.

2.5.3 *Keyed-Hash Message Authentication Code*

O *Keyed-Hash Message Authentication Code* (HMAC) é um tipo de código de autenticação de mensagens (*Message Authentication Code* (MAC)) que provê a autenticação e a integridade das mensagens. O HMAC é computado a partir do *hash* criptográfico da mensagem a ser transmitida em conjunto com uma chave secreta trocada previamente entre as partes. Ao chegar ao seu destino, a entidade receptora computa o HMAC da mensagem através do *hash* criptográfico da mensagem mais a

chave secreta compartilhada entre as partes, certificando a integridade e a autenticidade da mensagem recebida, assumindo que a entidade par é a única a conhecer a chave secreta.

2.6 Resumo

O Capítulo 2 apresentou os trabalhos e as tecnologias relacionadas a esta dissertação. A primeira proposta apresentada foi o *Mobile IP* e a sua importância se deve ao fato de ser a primeira proposta a separar o conceito de identificador do localizador. Esta separação é realizada através da utilização de dois endereços IPs, um atuando como identificador topológico na rede e outro como identificador de conexões. A segunda proposta apresentada foi o *Host Identity Protocol* que introduz o conceito de identificador criptográfico derivado a partir do par de chaves pública/privada da entidade. Este protocolo provê mecanismos de segurança baseados no IPSec para prover a autenticação, a integridade e o sigilo dos dados trocados entre as entidades. As principais contribuições herdadas são o conceito de identificador criptográfico e o modelo de segurança do protocolo. A terceira e última proposta apresentada foi o *Node Identity Internetworking Architecture* que se coloca como um modelo para a Internet de nova geração. O modelo utiliza uma estratégia de roteamento estático como forma de viabilizar a conexão de domínios dinâmicos na borda da Internet, esta última representada pelo *Core* da arquitetura.

A segunda parte do Capítulo 2 apresentou as tecnologias de segurança relacionadas ao trabalho desenvolvido. As tecnologias envolvidas no modelo de segurança proposto como o *Public Key Infrastructure* (PKI) e o protocolo criptográfico de troca de chaves *Diffie-Hellman* (DH) foram apresentadas. O PKI oferece um modelo de autenticação na Internet através da associação de chaves públicas a entidades por meio dos certificados digitais emitidos por uma autoridade certificadora. O protocolo criptográfico DH permite o estabelecimento de uma chave simétrica entre duas entidades através da troca de parâmetros em texto claro por um canal inseguro como a Internet. Ao final do procedimento de troca de chaves, o protocolo estabelece uma chave simétrica sem a troca prévia de qualquer informação sigilosa.

Capítulo 3

Proposta de Implementação de uma Arquitetura para a Internet de Nova Geração

Este capítulo apresenta uma proposta de implementação de uma arquitetura para a Internet de nova geração, destacando os requisitos necessários à proposta de implementação e os mecanismos para o provisionamento das novas funcionalidades da arquitetura.

3.1 Proposta de Implementação

A proposta de implementação de uma arquitetura para a Internet oferece suporte integrado à mobilidade, ao *multihoming*, à segurança e à heterogeneidade. Para isso, a implementação deverá atender aos seguintes requisitos:

- Identificador global estático;
- Roteamento baseado no identificador;
- Mecanismos de resolução de nomes;
- Gerência de localização.

3.1.1 Identificador global estático

A arquitetura propõe a introdução de uma nova camada na pilha de protocolos de rede, a camada de identificação. Esta nova camada está localizada entre a camada de rede e a camada de transporte e tem por objetivo identificar o nó. Atualmente não existe um identificador estável para a identificação de uma entidade, recorrendo-se ao endereço IP para a identificação de conexões entre nós. A camada de identidade soluciona o problema da sobrecarga semântica exercida sobre o IP, que atualmente é utilizado como localizador na camada de rede e como identificador de conexão na camada de transporte. O termo localizador é definido neste contexto como o identificador topológico de um nó, representando o local onde o nó se encontra. No caso do IP, o localizador é o próprio endereço IP.

A camada de identificação introduz um identificador estático e global, que permite a identificação de um único nó na Internet através da introdução de um espaço de identificação de 128 bits. O

identificador possui propriedades criptográficas pois ele é derivado do par de chaves público/privada gerado pelo nó, como proposto pelo *Host Identity Protocol* (HIP) descrito no Capítulo 2. A chave pública é o identificador do nó enquanto a sua chave privada é a sua identidade. Para padronizar o tamanho do identificador a ser utilizado na identificação dos pacotes, utiliza-se o *hash* criptográfico *Message Digest 5* (MD5) da chave pública, gerando um identificador de 128 bits denominado de ID, como mostra a Figura 3.1. Para a utilização do ID gerado a partir da sua chave pública, partimos do pressuposto que a chance de colisão é estatisticamente improvável, isto é, a possibilidade de colisão dos *hashes* gerados é muito baixa. [1]

A função de *hash* criptográfico utilizada na geração do ID liga a chave pública com o ID gerado, permitindo ao nó receptor verificar a autenticidade da identidade do emissor do pacote através do cômputo do *hash* da chave pública e comparando-a com o ID contido no pacote. No entanto, esta verificação não garante a autenticidade do emissor devido à vulnerabilidade do MD5 à ataques [25]. Hoje em dia já é possível forjar o mesmo *hash* criptográfico a partir de entradas diferentes. Este tipo de autenticação não cobre todos os casos, sendo complementado pelo modelo de segurança da arquitetura.

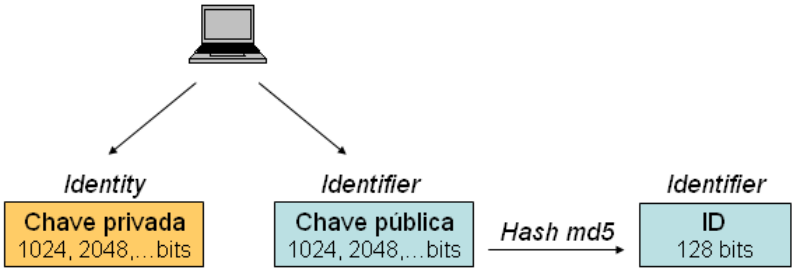

Fig. 3.1: Geração dos identificadores de um nó.

Uma entidade possui a possobilidade de manter mais de um identificador ativo simultaneamente. Parte destes identificadores são públicos, isto é, divulgados em serviços de diretórios como o DNS e parte destes identificadores permanecem não-divulgados. O identificador público é utilizado para a localização da entidade alvo por meio da consulta ao serviço DNS, enquanto o identificador não-divulgado é utilizado efetivamente na comunicação entre as entidades. Esta abordagem apresenta algumas vantagens, como a preservação da privacidade e a possibilidade de repelir ataques de negação de serviço. A privacidade é preservada devido à não-divulgação do identificador público, resultando na impossibilidade de descobrir a chave pública utilizada na geração do identificador não-divulgado. Dessa forma, um atacante que capture pacotes com determinados identificadores não consegue computacionalmente obter a partir do identificador de 128 bits a chave pública utilizada para gerá-lo. Ataques de negação de serviço são repelidos por meio do desregistro do identificador atacado e o registro de um novo identificador não-divulgado na comunicação com as entidades pares.

A introdução da nova camada de identificação permite às aplicações legadas utilizarem os pares *<ID origem, porta origem>,<ID destino, porta destino>* na identificação das conexões no lugar do localizador, possibilitando a associação dinâmica de novos localizadores sem a quebra de conectividade, como mostra a Figura 3.2.

[1] Para o espaço de 128-bits a ordem é de 1 em 1 quatrilhão.

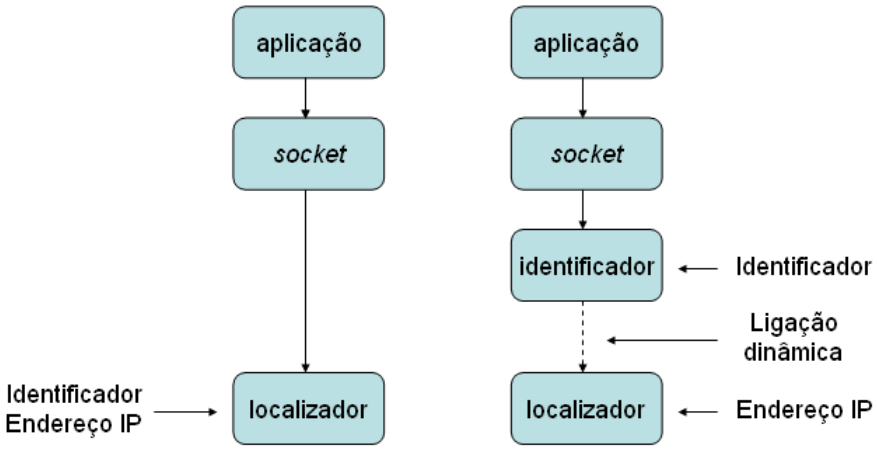

Fig. 3.2: Ligação da aplicação legada com a camada de identificação.

Outro benefício da introdução do identificador global estático é a possibilidade da associação de um identificador a vários localizadores, permitindo novas funcionalidades como o *multihoming*. Com esta nova funcionalidade, um nó poderá se conectar simultâneamente a mais de um provedor e realizar a troca simultânea do acesso entre eles sem a necessidade de reiniciar a conexão. Esta nova abordagem abre possibilidade para a troca dinâmica de provedores baseada em critérios como a qualidade de serviço ou tarifação de modo transparente para o usuário.

Outra possibilidade é a utilização do identificador para a integração de redes heterogêneas. O identificador é independente de tecnologia e disponibiliza um ponto de ligação estável para a camada de transporte, possibilitando a troca transparente de tecnologias de rede sem a quebra da conectividade. A translação de tecnologias de rede é realizada por *gateways* de bordas dos domínios heterogêneos.

3.1.2 Roteamento baseado na identidade

A introdução de um novo espaço de nomes localizado entre a camada de rede e a camada de transporte cria um modelo de rede sobreposta sobre o IP, possibilitando a introdução de novos serviços baseados na camada de identidade, como o roteamento sobre o identificador global e estático. O modelo proposto neste trabalho utiliza a simplificação da Internet do *Node Identity Internetworking Architecture* [7] (Node ID) descrita no Capítulo 2, ao assumir a existência de um *Core* no topo da hierarquia da Internet e vários domínios ligados a este *Core* formando árvores de domínios, como mostra a Figura 3.3.

O *Core* central é composto de domínios estáticos e representam a Internet atual. Ligados à borda do *Core* estão as árvores representando os domínios que se ligam dinamicamente ao núcleo. Estes domínios podem se mover e se compor com outros domínios ou com o *Core* dinamicamente. Todos os domínios conectados ao *Core* possuem um *gateway* (GW) ligado diretamente ao núcleo chamados de *Gateway* de *Core* (Core-GW). Os identificadores de todos os Core-GW estão registrados em um serviço de resolução de identificador em localizador global com a finalidade de possibilitar o roteamento através do *Core*. Uma das propostas encontradas na literatura para o provisionamento de um serviço de resolução global e escalável é por meio da utilização de uma *Distributed Hash Table* (DHT)

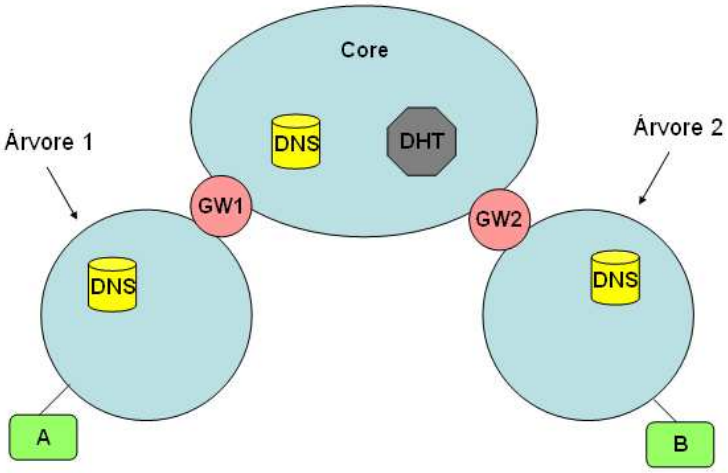

Fig. 3.3: Simplificação do modelo da Internet.

[26] localizado no topo da hierarquia da Internet. A DHT oferece a resolução dos identificadores dos Core-GW nos seus respectivos endereços IPv4 possibilitando o roteamento baseado no identificador passando pelo *Core*.

O roteamento nas árvores é baseado em uma rota estática em direção ao *Core*, semelhante ao modelo proposto pela arquitetura *Node ID*. Pacotes destinados a um nó na mesma árvore são encaminhados via rota estática até o GW do domínio na árvore onde o nó está registrado, de onde ele é encaminhado até o destino. Pacotes destinados a uma outra árvore seguem via rota estática até alcançar o GW-Core da árvore, onde é realizada a resolução do identificador do Core-GW (IDGW) no localizador do GW-Core de destino a partir de uma consulta à DHT, de onde é encaminhado até o destino através do roteamento IPv4.

Para realizar o roteamento baseado na identidade, utiliza-se o cabeçalho de identidade (cabeçalho ID) contendo os pares de identificadores de origem e destino e identificadores dos GW-Core de origem e destino, ilustrados na Figura 3.4.

A seguir, a descrição dos campos do cabeçalho de identidade.

- *Version* - Campo de 4 bits que identifica a versão do cabeçalho utilizado na arquitetura, possibilitando a descoberta do tipo de roteamento utilizado. A versão 0 do protocolo utiliza os IDGWs para o roteamento. A versão 1, a ser especificada como trabalho futuro, utiliza o identificador de domínio no lugar do identificador do GW-Core de modo a flexibilizar o roteamento estático adotado pela *Node ID Architecture*.

- *Type* - Campo de 4 bits que define o tipo de conteúdo na carga do pacote. Os tipos possíveis são: DATA, REGISTRY, REDIRECT, REDIRECT_CORE, SEC_CTRL, SYNC, RVS_UPDATE, RVS_ACK, RVS_GET, RVS_RESPONSE. As mensagens de controle são descritas nas próximas seções.

- *Reserved* - Campo de 24 bits reservado para uso futuro.

- *Data Length* - Campo de 16 bits contendo o tamanho em bytes da carga do pacote.

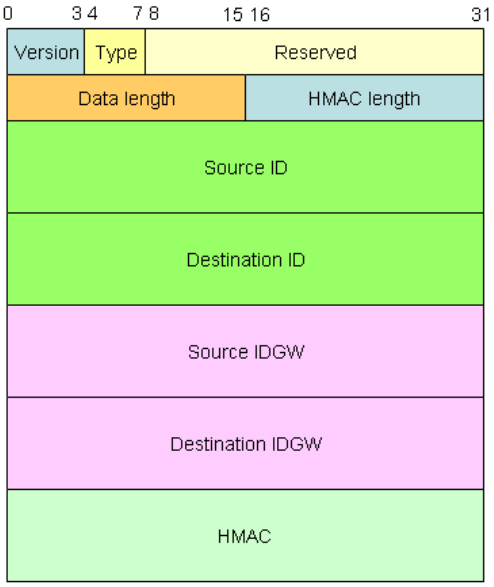

Fig. 3.4: Cabeçalho para a camada de identificação (cabeçalho ID).

- *HMAC Length* - Campo de 16 bits contendo o tamanho em bytes do *Keyed-Hash Message Authentication Code* (HMAC) do pacote. O campo HMAC é parte do modelo de segurança da arquitetura e a sua função é garantir a integridade do pacote.
- *Source ID* - Campo de 128 bits contendo o ID de origem.
- *Destination ID* - Campo de 128 bits contendo o ID de destino.
- *Source IDGW* - Campo de 128 bits contendo o identificador do GW-Core de origem.
- *Destination IDGW* - Campo de 128 bits contendo o identificador do GW-Core de destino.
- *HMAC* - Campo de tamanho definido pelo campo *Data Length* contendo o HMAC do pacote.

3.1.3 Mecanismo de resolução de nomes

A arquitetura propõe a utilização de *Fully Qualified Domain Names* (FQDNs) na identificação dos nós. A resolução do FQDN do nó no seu localizador é realizada em duas etapas: resolução do FQDN no par de identificadores <ID, IDGW> e resolução do identificador ID no localizador da tecnologia de rede do domínio. A decomposição do procedimento de resolução do FQDN no localizador em duas etapas visa separar as informações estáticas das dinâmicas para melhor gerenciamento, pois o DNS é adequado para armazenar informações que pouco se alteram ao longo do tempo.

A primeira etapa do mapeamento do FQDN no par de <ID, IDGW> pode ser realizada através de uma consulta aos servidores DNS. A característica estática do par de identificadores é apropriada para armazenamento no DNS, pois a eficiência deste advém do uso da hierarquia de resolução e do *caching*. Consultas realizadas ao DNS para mapeamentos do FQDN no par de identificadores podem

possuir validades longas, ao contrário do mapeamento do identificador no localizador, que pode ser dinâmico e possuir tempo de vida curto.

A segunda etapa de resolução é o mapeamento do identificador no localizador utilizado na entrega dos pacotes no domínio de destino. Como a informação armazenada é relativamente volátil, a arquitetura define um componente responsável pelo mapeamento do identificador no localizador e pelo suporte aos eventos de mobilidade simultânea. Este componente é chamado de *Rendezvous Server* (RVS) e está presente em todos os domínio excetuando o *Core*, sendo responsável pela recepção de mapeamentos de identificadores em localizadores e pelo suporte à mobilidade, a ser descrito na próxima seção. O RVS proposto por esta implementação difere do servidor proposto pelo HIP, que realiza a função de encaminhamento do pacote inicial semelhante ao *Home Agent* do MIP para o estabelecimento da sessão, com o objetivo de preservar a privacidade dos nós, como propõe Eggert *et al.* [27].

3.1.4 Gerência da localização

A gerência da localização engloba os mecanismos de localização dos nós estáticos e móveis utilizando o DNS e o RVS. Em [15], o autor mostra que o DNS não se adequa como ponto de atualização de informações dinâmicas como os mapeamentos de identificadores em localizadores. Isso se deve à característica estática do DNS, que retorna os dados requisitados junto com a sua validade. O sistema operacional armazena as entradas até a sua expiração traduzida em horas ou até dias. Para nós que se movem frequentemente, as entradas no DNS necessitariam de *Time-to-Live* (TTL) pequenos, o que resultaria no aumento da carga nos servidores raízes. Há uma relação de custo-benefício entre a escalabilidade e o tempo de restauração da conexão na escolha do TTL dos registros. Com a finalidade de prover um mecanismo eficiente para a gerência de localização, utilizamos o DNS para mapear a parte estática da resolução do FQDN enquanto utilizamos o *Rendezvous Server* (RVS) para mapear a parte dinâmica.

O RVS é um servidor localizado em todos os domínios onde os nós atualizam ou realizam resoluções de identificadores em localizadores. Os nós localizados dentro do domínio do RVS inserem o par <ID, localizador> mais o tempo de validade da entrada no servidor. Esta validade é variável e depende do grau de mobilidade do nó; caso o nó se mova frequentemente, o nó pode reduzir o tempo de validade da entrada com a finalidade de indicar ao nó par para consultar com maior frequência o seu localizador, diminuindo o tempo de restauração da comunicação após a mobilidade do nó. Por outro lado, a redução da validade do mapeamento requer a atualização freqüente por parte do nó no RVS, o que pode resultar em uma sobrecarga de atualizações no RVS. Há uma relação de custo-benefício entre desempenho e o tempo de restauração da conexão na escolha deste parâmetro.

Cada domínio poderá ter um ou mais RVS com a finalidade de prover maior escalabilidade e tolerância a falhas, formando uma *Distributed Hash Table* (DHT) dentro de um domínio. Esta DHT forma um anel de servidores onde os dados podem ser replicados entre eles provendo tolerância a falhas. A consulta a um identificador é encaminhada para o servidor mais próximo obedecendo um algoritmo de roteamento *flat*, como por exemplo *Chord* [23], *Pastry* [28] ou *Tapestry* [29], entre outros. Esta proposta se assemelha ao *Internet Indirection Infrastrucure* (i^3), onde os nós atualizam os seus localizadores nos servidores i^3 para suporte à mobilidade.

Outra funcionalidade do RVS é a sua utilização para suportar a mobilidade simultânea dos nós sem a quebra de conectividade, também conhecida como *double jump*. Dois nós que se movem si-

multaneamente durante uma comunicação restaurarão a comunicação após atualizarem os seus novos localizadores no RVS e realizarem a consulta dos localizadores pares no RVS.

3.2 Funcionalidades da arquitetura

A fim de atender aos requisitos mencionados na seção anterior, esta seção apresenta as funcionalidades da arquitetura de implementação para a Internet de nova geração, destacando os mecanismos utilizados. As funcionalidades oferecidas pela implementação são:

- Mobilidade;
- *Multihoming*;
- Transparência;
- Segurança;
- Heterogeneidade.

Para oferecer suporte às funcionalidades mencionadas, a arquitetura propõe um plano de controle para a troca de mensagens de sinalização entre os módulos, a ser descrito nas próximas seções.

3.2.1 Mobilidade

A arquitetura provê suporte natural à mobilidade ao introduzir a camada de identificação que oferece um ponto estável de ligação para as aplicações. Estas passam a utilizar o identificador no lugar do localizador na identificação das conexões resolvendo a dependência entre as camadas de rede e a camada de transporte e possibilitando a troca dinâmica do localizador. A nova camada utiliza uma interface virtual cujo endereço é o identificador do nó. Dessa forma, a interface virtual possui o identificador no lugar do seu endereço IP. As aplicações legadas utilizam o endereço da interface virtual de destino, que na realizade é o identificador de destino, no lugar do localizador, possibilitando a mobilidade de forma natural. A Figura 3.5 mostra a ligação de uma aplicação legada ao identificador.

A arquitetura separa os eventos de mobilidade em três tipos: intra-domínio, inter-domínio na mesma árvore e inter-domínio inter-árvores. Na mobilidade intra-domínio, o nó troca somente o seu localizador, mantendo os mesmos parâmetros do domínio, tais como os IDs dos servidores DNS, RVS e GW. Neste tipo de mobilidade, o nó móvel, ao detectar o evento de mudança de localizador, recupera a lista de nós pares com os quais possui comunicações ativas e envia uma mensagem de controle informando o seu novo localizador. Nas mobilidades inter-domínio na mesma árvore e inter-árvores, o nó móvel troca o seu localizador e os parâmetros do domínio, recebendo os novos parâmetros do servidor DHCP do domínio visitado. Na mobilidade inter-domínio, o GW do domínio onde o nó está localizado detecta o evento de mobilidade ao notar que o mapeamento do nó não é mais atualizado como resultado da sua migração ou desligamento da rede. Nestes tipos de mobilidade inter-domínio, não há o envio da mensagem de controle explícita informando o novo localizador devido a possibilidade da existência de redes heterogêneas.

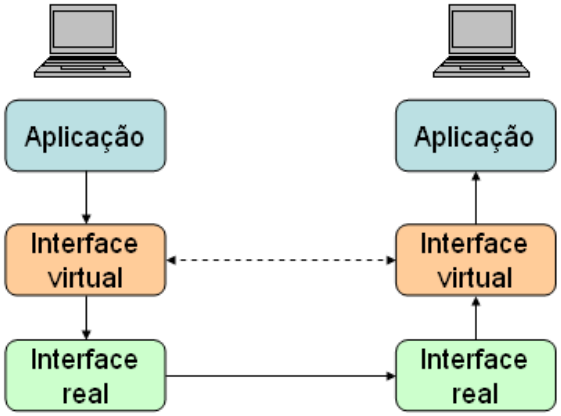

Fig. 3.5: Mobilidade utilizando a interface virtual.

A Figura 3.6 ilustra os tipos de mobilidade consideradas na arquitetura. A seta I mostra a mobilidade intra-domínio; a seta II mostra a mobilidade inter-domínio na mesma árvore e a seta III mostra a mobilidade inter-domínio entre árvores.

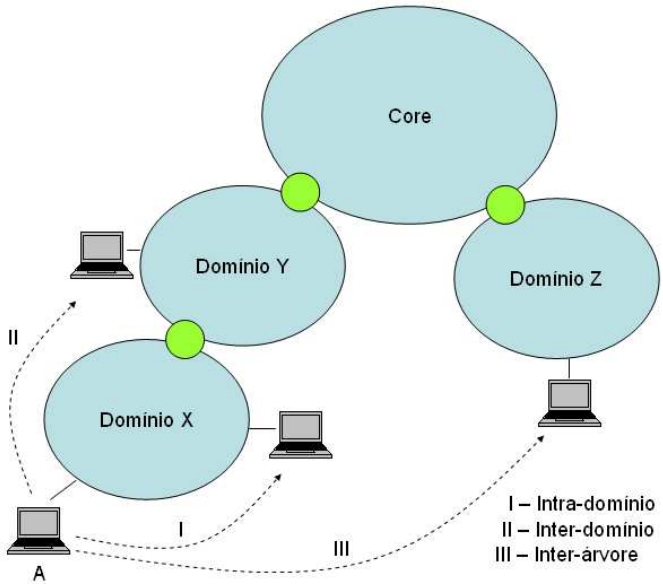

Fig. 3.6: Cenários de mobilidade considerados na arquitetura.

A utilização da mensagem de controle para informar o novo localizador do nó móvel possibilita a maior rapidez na restauração da comunicação sem a necessidade de aguardar a resolução do identificador no novo localizador do nó após a expiração do mapeamento (*timeout*).

O mecanismo de consulta periódica ao RVS para a atualização do mapeamento <ID, localizador> possibilita o cenário de mobilidade simultânea dos nós. Logo após a mobilidade simultânea intra-domínio dos nós, os nós atualizam os seus novos localizadores no RVS e enviam a mensagem de controle contendo o seu novo localizador para o localizador antigo dos nós pares. As mensagens

serão perdidas, uma vez que os nós receberam novos localizadores. Neste cenário, os nós recorrem à expiração da validade dos mapeamentos resultando em uma nova consulta ao RVS e na obtenção dos localizadores atualizados dos nós pares para a restauração da comunicação. Na mobilidade inter-domínio na mesma árvore ou inter-árvores, não há o envio da mensagem de controle, recorrendo somente à expiração dos mapeamentos e consequente consulta ao RVS para a obtenção do novo localizador do nó par.

Para auxiliar o mecanismo de mobilidade e a rápida restauração da comunicação entre os nós, é proposto um esquema de descoberta dos parâmetros dos domínios. Ao chegar a um novo domínio, o nó envia uma mensagem solicitando os parâmetros de configuração, como por exemplo o localizador do RVS, possibilitando a rápida reconfiguração do nó e o reestabelecimento das comunicações com os nós pares. Os parâmetros recebidos serão descritos nas próximas seções.

3.2.2 *Multihoming*

A arquitetura oferece suporte natural ao *multihoming* como resultado da introdução da camada de identificação. O *multihoming* é um procedimento utilizado para aumentar a confiabilidade do acesso à Internet através da conexão com múltiplos provedores de acesso. O identificador de uma entidade pode ter mais de um localizador associado simultaneamente e possibilita a troca dinâmica entre os provedores de acesso sem a quebra da conectividade ou necessidade de reinicialização da comunicação. A Figura 3.7 ilustra um exemplo de *multihoming*. Inicialmente, o nó A comunica-se com o nó B através do domínio de acesso *Y*. Após um certo instante de tempo, o gerenciador de *multihoming* toma a decisão de utilizar o domínio de acesso Z para se comunicar com B baseado em algum critério como a qualidade de serviço, tarifação ou simplesmente porque o enlace com o domínio Y está fora do ar.

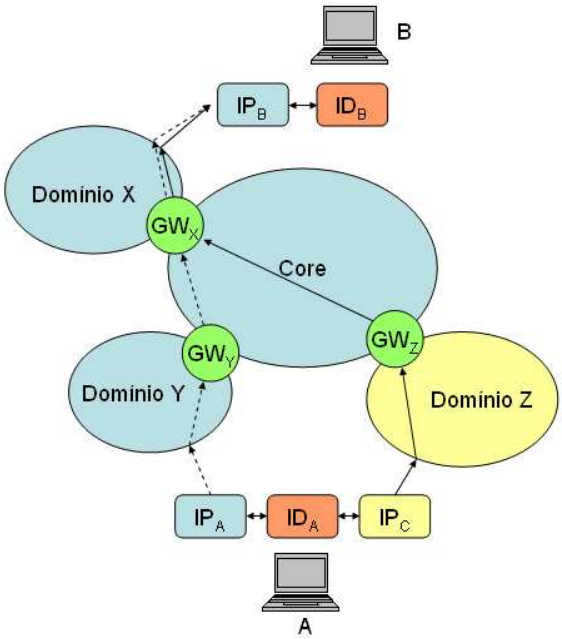

Fig. 3.7: Exemplo de *Multihoming*.

O gerenciador de *multihoming* provê um mecanismo de tomada de decisões de forma dinâmica com a utilização de tabelas de roteamento extras do sistema operacional. Ao invés de utilizar a tabela de roteamento principal (*Main Routing Table*) do sistema operacional, a arquitetura propõe a utilização de tabelas secundárias compostas de uma rota *default* principal e outras secundárias. Ao estabelecer a conexão com um nó par, o gerenciador de *multihoming* da arquitetura inicia um procedimento para o cálculo do *Round Trip Time* (RTT) das interfaces ligadas a outros provedores com a finalidade de obter o menor RTT até o destino. Após obter o valor, é inserida na tabela de roteamento da arquitetura a rota com o menor RTT até o destino, provendo o acesso ao destino via outra rota sem a quebra da conectividade e de forma transparente para o usuário. A tabela de roteamento utilizada pela arquitetura poderá suportar outros parâmetros para o provisionamento de qualidade de serviço, como banda disponível e perda de pacotes.

3.2.3 Transparência

A arquitetura proposta é totalmente transparente às aplicações legadas, possibilitando a utilização das funcionalidades da arquitetura sem a necessidade de modificação da aplicação. O suporte transparente às aplicações legadas é alcançado através da utilização de interceptadores de pacotes legados que os inserem na arquitetura para a adição da camada de identificação.

O primeiro mecanismo para o suporte às aplicações legadas é a interceptação das resoluções de nome utilizando o DNS. Resoluções de nomes em localizadores realizadas pelas aplicações legadas são interceptadas na máquina local através de um *proxy* e convertidas para resoluções de nomes em identidades para serem encaminhadas ao servidor DNS. Na implementação proposta, consideramos que o DNS é modificado para armazenar pares <ID, IDGW> no lugar dos localizadores. Após receber a resposta da requisição, o *proxy* armazena o par <ID, IDGW> em uma base interna de dados e monta uma resposta de resolução de nome em localizador e a encaminha para a aplicação legada contendo um ID de 32 ou 128 bits, dependendo do tipo de registro requisitado. Para a aplicação, trata-se de um localizador como outro qualquer e ela passa a transmitir pacotes destinados a este identificador.

O segundo mecanismo para suporte às aplicações legadas é a interceptação dos pacotes legados destinados aos identificadores. Estes pacotes são interceptados por meio da utilização da ferramenta *Iptables* [30] e inseridos na arquitetura para a adição das funcionalidades da arquitetura, como a camada de identificação e o roteamento baseado no identificador.

3.2.4 Segurança

O modelo de segurança proposto utiliza a camada de identificação para o estabelecimento da segurança nas comunicações fim-a-fim. A vantagem desta abordagem é que ela é nativa da arquitetura e suporta todas as funcionalidades propostas como mobilidade, *multihoming*, transparência e heterogeneidade. O *Internet Protocol Security* (IPSec) [2] atua na camada de rede e utiliza o endereço IP como identificador das associações de segurança, herdando todos os problemas da utilização do localizador como identificador, como a falta de suporte à mobilidade. Outro problema do IPSec é a utilização obrigatória da tecnologia de rede IP, impossibilitando cenários de integração de redes heterogêneas.

Outra solução de segurança amplamente utilizada é o *Security Socket Layer* (SSL) [31] que atua na camada de aplicação. A vantagem do modelo de segurança que atua na camada de aplicação é a independência de tecnologia da camada de rede, possibilitando a mobilidade. A desvantagem desta

abordagem é a necessidade da negociação dos parâmetros de segurança para cada sessão aberta com a entidade de destino. Esta abordagem é custosa em um cenário onde se estabelece a comunicação entre dois nós utilizando vários aplicativos. O SSL cria uma associação de segurança para cada aplicativo em comunicação com o nó, gerando um *overhead* desnecessário.

O modelo de segurança adotado pela arquitetura é baseado no comparativo entre o IPSec e o SSL [32] e provê uma solução intermediária entre o IPSec e o SSL, isto é, utiliza o modelo de túnel fim-a-fim com uma entidade como no modelo IPSec e é independente de tecnologia da camada de rede como o SSL. Uma entidade somente necessita negociar os parâmetros de segurança uma única vez para estabelecer qualquer número de sessões entre os aplicativos dos nós e com suporte à mobilidade, o *multihoming* e à heterogeneidade sem interferir nas aplicações legadas. O principal objetivo do modelo proposto é a validação de um modelo de segurança sobre a camada de identificação, provendo segurança básica para as aplicações legadas.

O modelo de segurança da arquitetura se baseia na Infraestrutura de Chaves Públicas (*Public Key Infrastructure* - PKI) descrita no Capítulo 2. A arquitetura propõe uma *Certificate Authority* (CA) global localizada no *Core* e várias CAs regionais distribuídas nas árvores. Cada árvore possui uma ou mais CAs ativas conforme o número de nós ou domínios ligados à ela. O registro inicial de um nó em um domínio inicia-se com o envio de informações para a CA solicitando um certificado digital. Após a verificação da autenticidade das informações providas, a CA emite um certificado com a sua assinatura digital garantindo a autenticidade das informações contidas no certificado e a sua ligação com a chave pública contida no certificado. De posse do certificado digital, o nó poderá inserir o mapeamento da sua FQDN no par de identificadores no servidor DNS.

O modelo de segurança proposto oferece a autenticação das identidades através dos certificados digitais, a integridade dos dados através dos códigos de autenticação de mensagens (HMAC) e sigilo por meio de algoritmos de chaves simétricas.

A autenticidade da identidade é garantida através da verificação da assinatura digital contida no certificado. Cada nó possui um conjunto inicial de chaves públicas de autoridades certificadoras conhecidas que permitem a verificação da autenticidade de uma determinada assinatura digital.

A integridade das mensagens é garantida através do uso do HMAC. Este gera um *hash* criptográfico a partir da mensagem a ser enviada juntamente com uma chave simétrica trocada previamente entre as partes. Ao chegar no destino, a entidade computa o HMAC do pacote e a compara com o HMAC enviado para certificar que a mensagem enviada é a mesma mensagem recebida. Uma das vantagens da utilização do HMAC é a possibilidade de verificação tanto da integridade da mensagem trocada quanto da autenticidade da entidade, assumindo que a entidade par é a única a conhecer a chave simétrica.

A chave simétrica utilizada na comunicação entre as entidades é trocada utilizando o protocolo de troca de chaves criptográficas *Diffie-Hellman* (DH) descrito no Capítulo 2. Esta chave é utilizada pelo HMAC para prover a integridade das mensagens e pela criptografia de chave simétrica para prover o sigilo dos dados.

A arquitetura garante a sigilo dos dados através do uso de algoritmos de chave simétrica como 3DES, AES, Blowfish, Camellia, IDEA e SEED, à escolha do usuário. Uma otimização proposta para o modelo de segurança consiste em um protocolo de negociação dos parâmetros de segurança baseado nos recursos computacionais disponíveis nos nós, como por exemplo, utilização de algoritmos de criptografia mais eficientes para dispositivos com pouca capacidade de processamento como os *Personal Digital Assistants* (PDAs).

3.2.5 Heterogeneidade

A arquitetura proposta provê suporte à heterogeneidade da tecnologia de rede através da inserção de uma camada de identificação homogênea e independente de tecnologia. As aplicações passam a se ligar ao identificador no lugar do localizador, possibilitando a troca dinâmica dos protocolos de rede nos GWs tradutores localizados nas bordas dos domínios heterogêneos. Os GWs de borda devem conhecer ambas as tecnologias de rede, realizar a tradução das tecnologias e encaminhar os pacotes para o destino. A Figura 3.8 ilustra um exemplo de heterogeneidade onde o nó A se comunica com o nó B, ambos em domínios IP, e são conectados por um domínio utilizando a tecnologia IPX [33]. Os GWs de borda conhecem ambas as tecnologias de rede e realizam a tradução IP-IPX e IPX-IP sem a quebra de conectividade para as aplicações fim-a-fim.

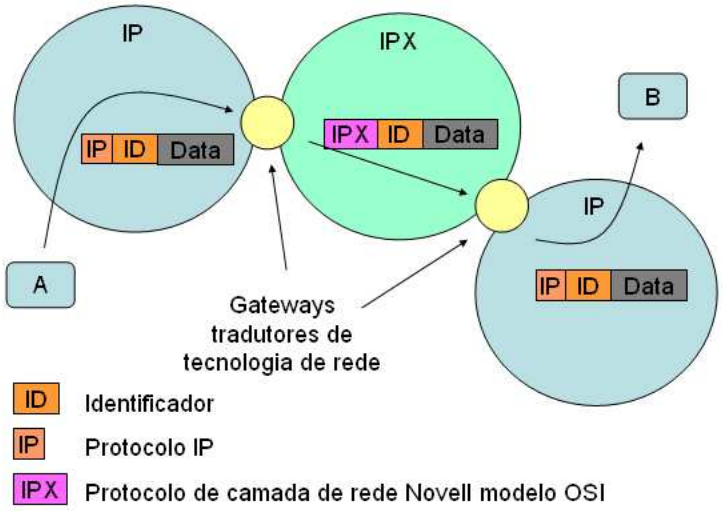

Fig. 3.8: Exemplo de heterogeneidade.

Outro fator que possibilita a heterogeneidade da tecnologia de rede é o modelo de segurança ser baseado na identidade ao invés do localizador. Propostas como o HIP, o MIP e outras utilizam soluções de segurança atrelados à tecnologia de rede IP, impossibilitando a heterogeneidade de rede onde cada nó utiliza uma tecnologia de rede diferente.

3.2.6 Plano de Controle

O plano de controle da proposta de arquitetura tem por objetivo prover um mecanismo de troca de mensagens de sinalização entre os componentes da arquitetura. Baseado no plano de controle do *Ambient Networks*, o plano de controle da proposta oferece serviços de registro, notificação, sincronização, resolução de identificadores e troca de parâmetros de segurança. As seguintes mensagens são definidas no plano de controle:

- REGISTRY;
- REDIRECT;

- REDIRECT_CORE;
- SYNC;
- SEC_CTRL;
- RVS_UPDATE;
- RVS_ACK;
- RVS_GET;
- RVS_RESPONSE.

O plano de controle opera sobre a camada de identificação da arquitetura e possui um cabeçalho especial para as mensagens de controle, ilustradas na Figura 3.9.

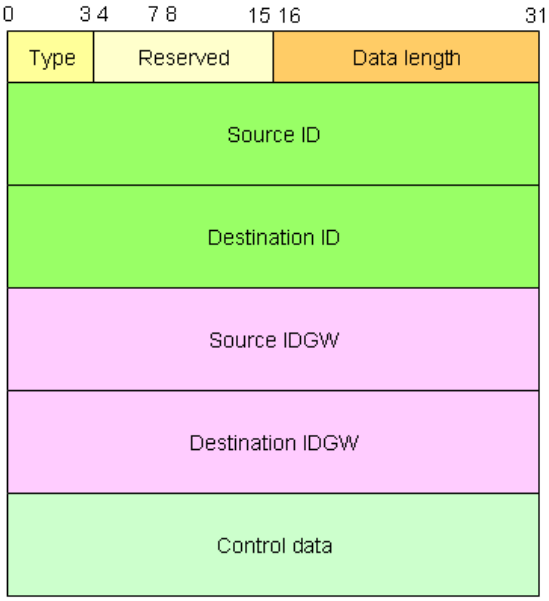

Fig. 3.9: Cabeçalho das mensagens de controle.

A mensagem de controle REGISTRY é utilizada para o registro de um nó em um domínio e é enviada do RVS para o GW do domínio. Ao chegar em um novo domínio, o nó envia uma mensagem de atualização do seu localizador para o RVS do domínio contendo o seu ID e o seu localizador atual. O RVS, ao receber a mensagem, envia uma mensagem de controle do tipo REGISTRY para o GW do domínio e este propaga através dos GWs intermediários até chegar ao GW-Core. Os GWs intermediários adicionam o mapeamento do identificador contido na mensagem e o localizador do GW antecessor na tabela de roteamento. A Figura 3.10 mostra o cenário do procedimento de registro de um nó ao chegar a um domínio.

A mensagem de controle REDIRECT é utilizada para a notificação do evento de mobilidade fim-a-fim, informando ao nó par o identificador e o novo localizador associado àquele identificador.

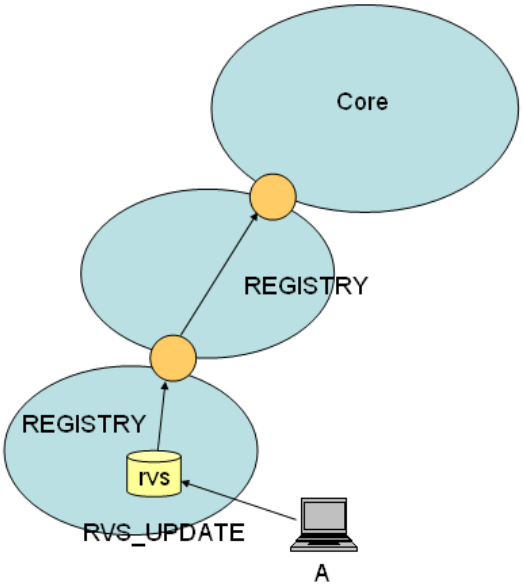

Fig. 3.10: Mensagem de controle REGISTRY.

O REDIRECT tem por objetivo otimizar o tempo de restauração da comunicação com o nó par, eliminando a necessidade do nó par de aguardar a expiração do mapeamento do localizador antigo e realizar uma nova resolução do identificador no novo localizador. A arquitetura detecta o evento de mobilidade e recupera a lista de comunicações ativas, enviando o seu novo localizador a todos os nós da lista que se encontram no domínio. A Figura 3.11 ilustra o cenário de mobilidade intra-domínio com o envio da mensagem de REDIRECT.

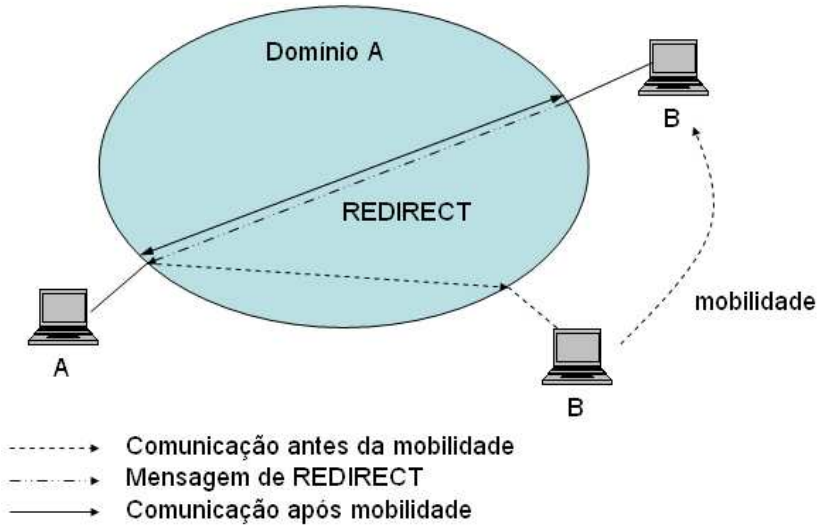

Fig. 3.11: Mensagem de controle REDIRECT.

A mensagem de controle REDIRECT_CORE é utilizada para a notificação dos GWs-Core responsáveis por nós que estão em uma outra árvore. A Figura 3.12 ilustra o cenário onde o nó B está inicialmente no domínio W e se move para o domínio Z, onde ele se registra no RVS local através da mensagem RVS_UPDATE. O RVS, ao receber a mensagem, cria a mensagem REGISTRY que é propagada até o Core-GW avisando a chegada de um novo nó na árvore. O GW-Core verifica o IDGW de origem da mensagem e conclui que o nó não pertence à árvore, pois ele utiliza o IDWG de um outro GW-Core. O GW-Core envia uma mensagem de REDIRECT_CORE contendo o identificador do nó B e o IDGW do GW-Core III para o GW-Core onde o nó A está registrado (GW-Core *Home* de A). O nó A, ao resolver o FQDN de B obterá como resposta <ID_B, $IDGW_{II}$> e transmitirá mensagens para o GW-Core II. Este consultará a tabela de registros e verificará que o nó B se moveu e está atualmente sob o GW-Core III, encaminhando os pacotes para o GW-Core atual do nó e possibilitando a localização do nó fora da sua árvore.

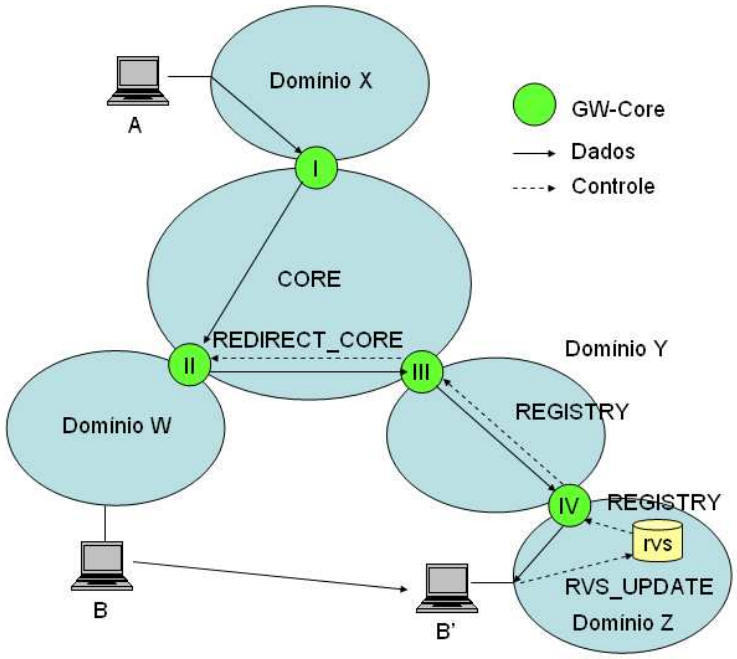

Fig. 3.12: Atuação da mensagem de controle REDIRECT_CORE.

A mensagem de controle SYNC é uma proposta de otimização para a arquitetura a ser utilizada na sincronização dos IDGWs atuais dos nós, resultando em duas otimizações para a arquitetura. A primeira otimização se refere às consultas no RVS. Durante a transmissão de um tráfego UDP onde não há mensagens de controle retornando para o emissor, o mapeamento da identidade no localizador do destino expira por tempo, sendo necessário ao nó emissor consultar o RVS, independentemente se o nó par se moveu ou não. A mensagem SYNC evita este tipo de cenário pois ela é trocada periodicamente entre os nós, indicando que ambos estão ativos e atualizando os mapeamentos dos IDs dos nós pares, se estiverem no mesmo domínio, ou dos IDGWs do domínio se estiverem em domínios diferentes.

A segunda otimização resultante da mensagem SYNC é atualização dos IDGWs após evento de mobilidade inter-árvores dos nós, evitando a triangulação entre os Core-GW na entrega de pacotes,

como mostra a Figura 3.13. Em (a) é ilustrado um cenário sem a mensagem SYNC. O nó A envia um fluxo para o nó B que se move para o domínio Z. O tráfego que chega em Core-GW Y é redirecionado para o Core-GW Z, onde o nó B está localizado. Como o fluxo é unidirecional, não há uma mensagem de atualização do Core-GW onde B está localizado atualmente, resultando na triangulação no *Core*. Em (b) é mostrado o cenário com a mensagem SYNC. Devido à troca periódica de mensagens SYNC entre os nós, o IDGW do Core-GW onde o nó B está localizado é atualizado no nó A evitando a triangulação e o consumo de recursos do Core-GW Y.

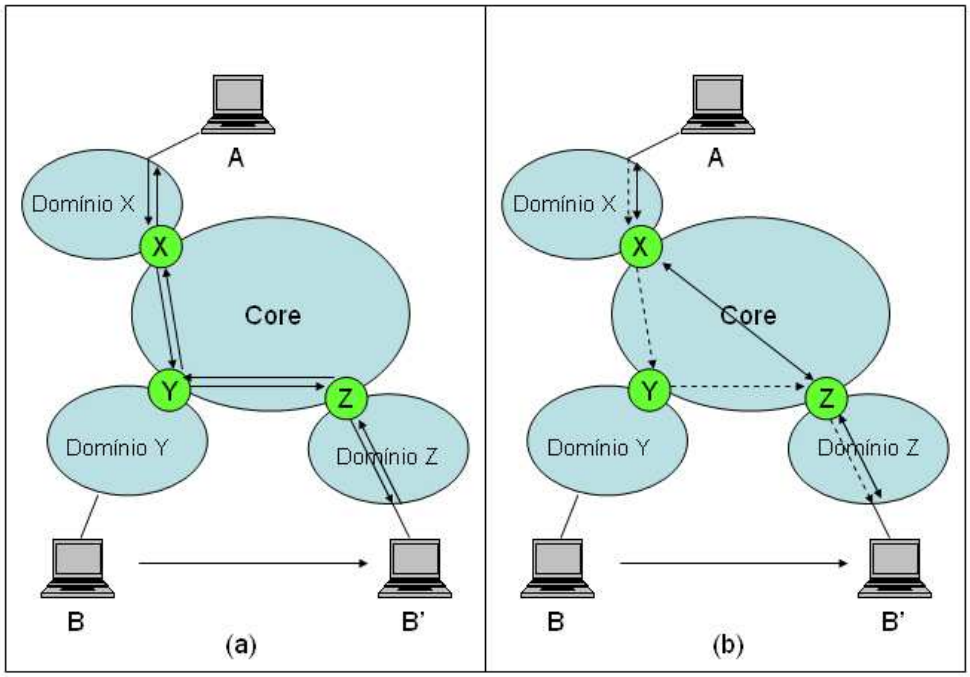

Fig. 3.13: Mensagem de sincronização SYNC.

A mensagem de controle SEC_CTRL é utilizado para o estabelecimento das associações de segurança entre os nós da arquitetura antes do início da comucanicação. As mensagens de segurança possuem um cabeçalho especial ilustrado na Figura 3.14 e descrito a seguir:

- *Version* - Campo de 4 bits que contém a versão do protocolo de negociação. Útil para futuras extensões do protocolo de negociação.

- *Type* - Campo de 4 bits indicando o tipo de mensagem de segurança que ela contém. As mensagens utilizadas na versão 1 são: HELLO, HOWAU, FINE e BYE.

- *Signature Type* - Campo de 4 bits que contém o tamanho em bytes do tipo do algoritmo utilizado para a geração da assinatura digital. Para esta versão do protótipo foi definido dois tipos: o tipo 0 ue corresponde a utilização do algoritmo RSA-MD5 e o tipo 1 que corresponde a utilização do algoritmo RSA-SHA1.

- *Transaction ID* - Campo de 16 bits que contém o identificador da negociação de segurança.

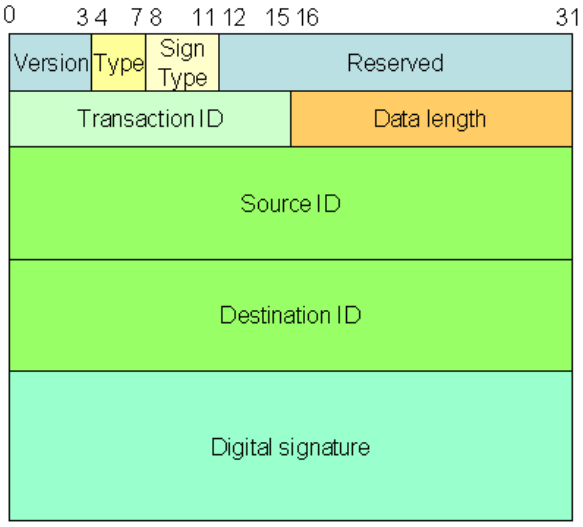

Fig. 3.14: Cabeçalho das mensagens de segurança.

- *Data Length* - Campo de 16 bits que contém o tamanho da carga em bytes da mensagem de segurança.
- *Source ID* - Campo de 128 bits contendo o ID de origem.
- *Destination ID* - Campo de 128 bits contendo o ID de destino.
- *Digital Signature* - Campo de tamanho definido pelo conteúdo do campo *Signature Length* que contém a assinatura digital da mensagem de segurança.

O cabeçalho contém o campo *Transaction ID* para identificar a negociação e repelir ataques de repetição. Este ataque consiste no armazenamento de pacotes autênticos capturados por um atacante e a sua reinserção futura na rede. Apesar do atacante não conhecer o conteúdo do pacote, ele pode fazer uso da autenticidade do pacote para proveito próprio. O exemplo clássico é a captura do fluxo de pacotes de uma transferência bancária e a reutilização destes pacotes para a multiplicação do valor transferido. No caso da arquitetura proposta, durante a negociação da associação de segurança, um identificador de transação é gerado aleatoriamente e armazenado para a utilização em uma única vez. Pacotes autênticos, porém com identificadores de transação já utilizados, são descartados pelo sistema como forma de proteção contra ataques de repetição.

O campo de assinatura digital recebe a assinatura digital do remetente, garantindo a autenticidade e a integridade da mensagem enviada. Este campo é importante para garantir a integridade das informações trocadas em texto plano como os parâmetros *Diffie-Hellman*, evitando a troca dos parâmetros por um atacante e impedindo o estabelecimento da associação segura. Quando ocorre alguma alteração na mensagem de segurança, a assinatura digital não confere e o nó descarta. Outra alternativa é a utilização dos identificadores anônimos para a comunicação. Como eles não são publicados, não há como descobrir a identidade associada àquele identificador possibilitando a comunicação segura entre os nós, uma vez que o atacante não consegue descobrir a identidade por trás daquele identificador.

O estabelecimento de uma associação segura é ilustrado na Figura 3.15. O procedimento inicia com a resolução do FQDN do nó no servidor DNS, obtendo o par de identificadores <ID, IDGW> do destino. Ao receber o primeiro pacote de dados destinado ao nó par, o módulo *Security* inicia a negociação dos parâmetros de segurança representados pelas mensagens HELLO, HOWAU e BYE.

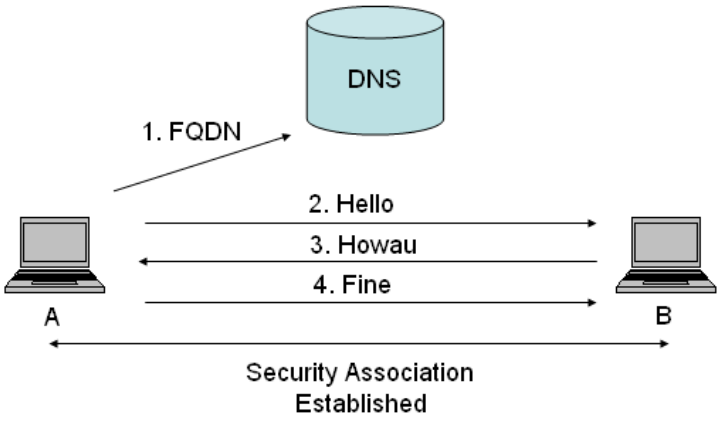

Fig. 3.15: Estabelecimento de uma associação segura.

A Figura 3.16 mostra a mensagem de segurança HELLO cujos campos estão descritos a seguir:

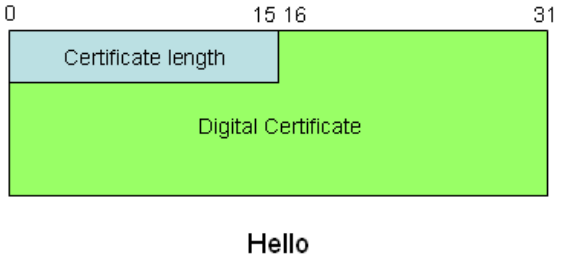

Fig. 3.16: Mensagem de segurança *Hello*.

- *Certificate Length* - Campo de 16 bits que contém o tamanho em bytes do certificado digital.
- *Digital Certificate* - Campo de tamanho definido pelo conteúdo do campo *Certificate Length* que contém o certicado digital do emissor.

A mensagem HELLO é utilizada para iniciar a negociação da associação de segurança. O emissor envia a mensagem contendo o seu certificado digital para o receptor, que verifica a assinatura digital contida no certificado. Após verificada a autenticidade do certificado, o receptor retorna a mensagem de HOWAU, ilustrada na Figura 3.17.

A seguir a descrição dos campos da mensagem HOWAU:

- *Signature Type* - Campo de 4 bits que contém o tamanho em bytes da assinatura digital. Os tipos definidos são os mesmos do campo homônimo da mensagem de segurança.

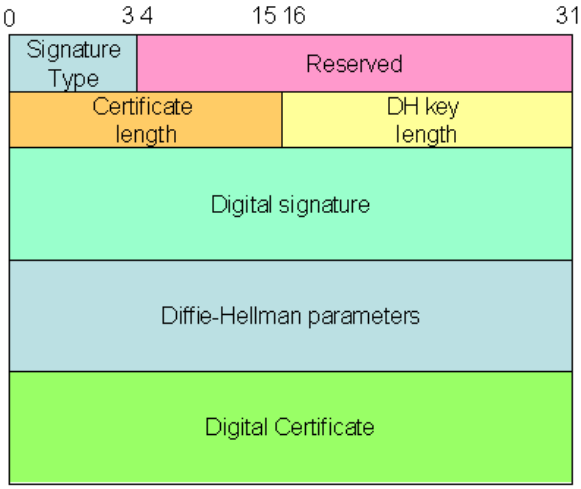

Fig. 3.17: Mensagem de segurança *Howau*.

- *Certificate Length* - Campo de 16 bits que contém o tamanho em bytes do certificado digital.
- *DH Key length* - Campo de 16 bits que contém o tamanho em bytes da estrutura utilizada para armazenar os parâmetros *Diffie-Hellman*.
- *Digital Signature* - Campo de tamanho definido pelo conteúdo do campo *Signature Length* que contém a assinatura digital da mensagem de segurança.
- *Diffie-Hellman parameters* - Campo de tamanho definido pelo conteúdo do campo *DH Key Length* que contém os parâmetros *Diffie-Hellman*.
- *Digital Certificate* - Campo de tamanho definido pelo conteúdo do campo *Certificate Length* que contém o certicado digital do emissor.

A mensagem HOWAU é utilizada como resposta da mensagem HELLO. Após a verificação da autenticidade da identidade do nó par com a verificação da assinatura digital do certificado, o nó envia a mensagem HOWAU contendo o seu certificado digital e os primeiros parâmetros *Diffie-Hellman* a serem utilizados no estabelecimento da chave simétrica. Antes do envio da mensagem ao nó par, a mensagem é assinada digitalmente.

Ao receber a mensagem HOWAU, o nó verifica a autenticidade da assinatura digital do certificado. Após a verificação, o nó gera os parâmetros DH locais e envia a mensagem FINE para o nó par contendo os parâmetros DH juntamente com a assinatura digital da mensagem. Após o envio, o nó gera a chave simétrica a partir dos parâmetros DH recebidos e armazena no módulo *Security Database*. A mensagem FINE é ilustrada na Figura 3.18.

- *Signature Type* - Campo de 4 bits que contém o tamanho em bytes da assinatura digital. Os tipos definidos são os mesmos do campo homônimo da mensagem de segurança.
- *DH Key Length* - Campo de 16 bits que contém o tamanho em bytes da estrutura utilizada para armazenar os parâmetros *Diffie-Hellman*.

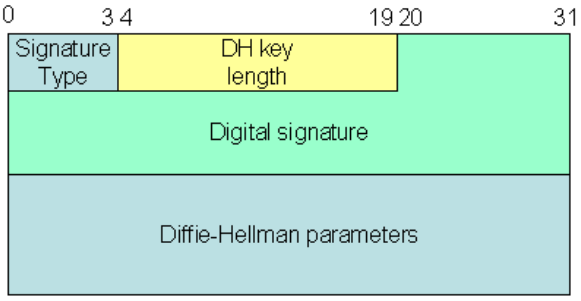

Fig. 3.18: Mensagem de segurança *Fine*.

- *Digital Signature* - Campo de tamanho definido pelo conteúdo do campo *Signature Length* que contém a assinatura digital da mensagem de segurança.
- *Diffie-Hellman parameters* - Campo de tamanho definido pelo conteúdo do campo *DH Key Length* que contém os parâmetros *Diffie-Hellman*.

Caso ocorra uma falha durante a negociação dos parâmetros de segurança, o nó que detectou o erro envia a mensagem BYE indicando o erro encontrado. A Figura 3.19 ilustra a mensagem BYE.

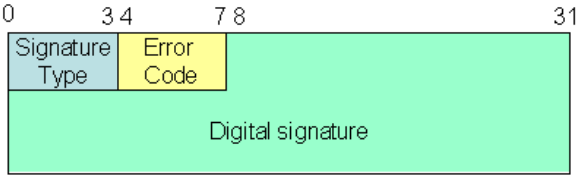

Fig. 3.19: Mensagem de segurança *Bye*.

- *Signature Type* - Campo de 4 bits que contém o tamanho em bytes da assinatura digital. Os tipos definidos são os mesmos do campo homônimo da mensagem de segurança.
- *Error Code* - Campo de 4 bits que contém o código do erro encontrado.
- *Digital Signature* - Campo de tamanho definido pelo conteúdo do campo *Signature Length* que contém a assinatura digital da mensagem de segurança.

O plano de controle é responsável pela resolução do identificador no localizador de um nó. Para isso, ele utiliza o cabeçalho ilustrado na Figura 3.20.

Quatro mensagens de controle são definidas para a comunicação com o RVS, que são RVS_UPDATE, RVS_ACK, RVS_GET e RVS_RESPONSE e estão ilustradas na Figura 3.21. A mensagem RVS_UPDATE é utilizada para a atualização do localizador do nó no RVS e é enviada no momento de ativação do nó ou na chegada do nó a um domínio após o evento de mobilidade. A mensagem de controle possui um campo com o mapeamento <ID, localizador> para ser inserido no RVS. Em resposta, o nó recebe

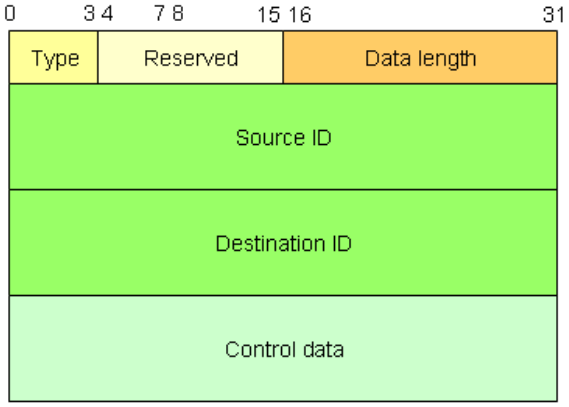

Fig. 3.20: Cabeçalho das mensagens RVS.

a mensagem RVS_ACK indicando o sucesso da inserção. Caso o nó não receba o RVS_ACK dentro de um intervalo de tempo definido na configuração, por exemplo, 1 segundo, ele torna a enviar uma mensagem de RVS_UPDATE para inserir o seu mapeamento na base de dados evitando a inalcançabilidade do nó entre uma atualização e outra. O RVS_ACK é importante para indicar que a mensagem de RVS_UPDATE foi recebida.

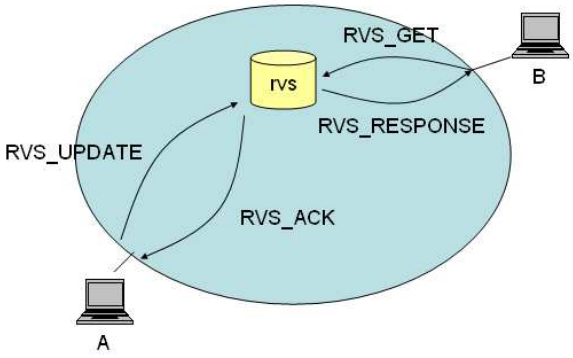

Fig. 3.21: Mensagens de controle para comunicação com o RVS.

As mensagens RVS_GET e RVS_RESPONSE são utilizadas na resolução de identificadores em localizadores. O plano de controle envia uma mensagem de RVS_GET para o RVS com a finalidade de resolver um dado ID no localizador utilizado no domínio. O RVS em contrapartida responde com uma mensagem RVS_RESPONSE contendo o localizador do nó desejado ou o campo vazio indicando que o nó não está registrado no domínio naquele momento.

3.3 Módulos da arquitetura

Esta seção apresenta o mapeamento das funcionalidades discutidas anteriormente nos módulos da arquitetura proposta e ilustrados na Figura 3.22. Os módulos são divididos em internos e externos.

Os módulos internos são responsáveis pelo provisionamento dos serviços propostos pela arquitetura como o suporte à mobilidade, a segurança e o plano de controle. Os módulos externos são entidades que provêem serviços à arquitetura como o DNS, o DHCP, o RVS e a DHT.

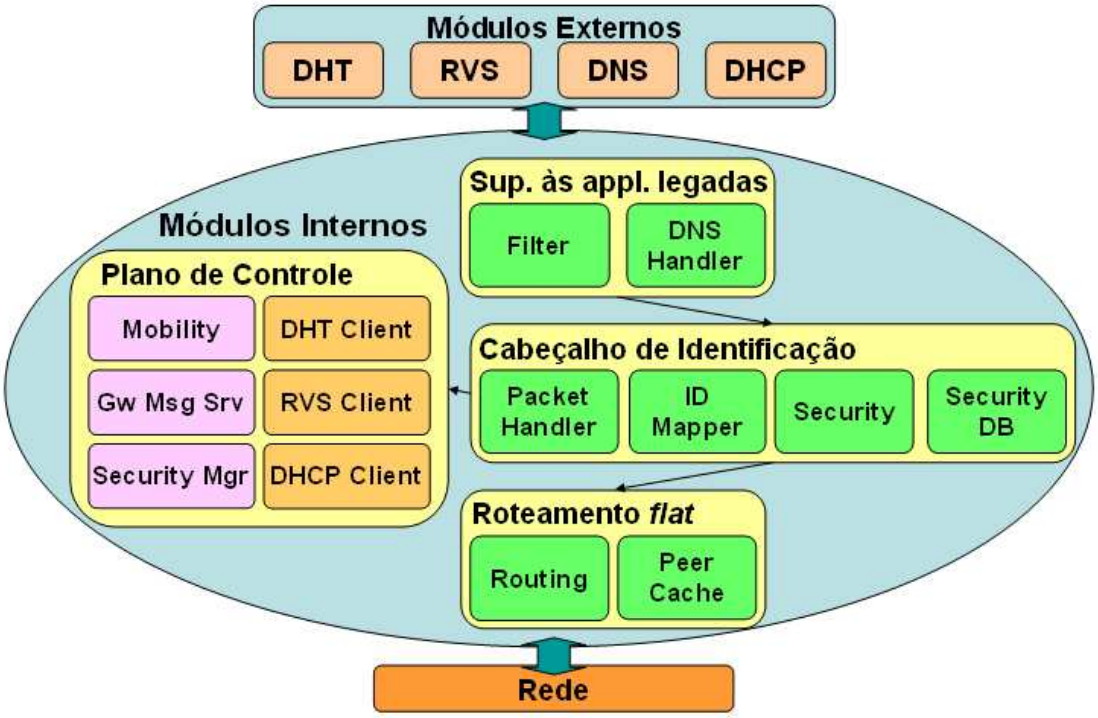

Fig. 3.22: Módulos da arquitetura.

3.3.1 Módulos Internos

Os módulos internos são divididos em plano de dado e plano de controle. Os módulos do plano de dado são responsáveis pelo suporte às aplicações legadas e à introdução da camada de identidade, da camada de segurança e da camada de roteamento. Os diagramas de seqüência envolvendo os módulos do plano de dados no envio e recepção dos pacotes se encontram no Apêndice B. O plano de controle é responsável pelas mensagens de sinalização trocadas entre os nós, tais como as mensagens de sinalização indicando eventos de mobilidade ou registros de nós.

Filter

O módulo *Filter* é responsável pela captura de pacotes de aplicações legadas que desejam utilizar as funcionalidades providas pela arquitetura. O módulo somente captura pacotes enviados a um identificador, permitindo a passagem de pacotes que utilizam localizadores. Os identificadores possuem uma máscara que lhes permite a diferenciação dos endereços IPv4 e IPv6. Esta diferenciação possibilita ao usuário optar pela utilização ou não dos serviços da arquitetura. Os identificadores de 32 bits possuem uma máscara de 8 bits, na forma de 1.x.x.x/8, enquanto os identificadores de 128

bits possuem uma máscara de 16 bits na forma de 1000::/16. Pacotes endereçados a qualquer destino que possuem endereços que se enquadram nas máscaras dos identificadores são capturados e tratados pela arquitetura, oferecendo as novas funcionalidades sem a modificação das aplicações legadas ou do sistema operacional. Após a captura dos pacotes legados, estes são encaminhados para o módulo *Packet Handler*.

DNS Handler

O módulo *DNS Handler* é responsável pela interceptação das mensagens de resolução de FQDNs no DNS para suporte às aplicações legadas. Este módulo captura as requisições enviadas ao DNS, que podem ser do tipo de registro A ou AAAA representando, respectivamente, os endereços IPv4 e IPv6, e as transformam em requisições de resolução no par de identificadores <ID, IDGW> como mostra a Figura 3.23. O tipo de registro solicitado é modificado para TXT com a finalidade de obter com uma única resolução ambos os identificadores (2). O servidor DNS retorna a consulta para o DNS Handler (3) e este armazena o par de identificadores em uma base de dados interna (4), o *ID Mapper*, para ser utilizado, posteriormente, na criação do pacote ID. Após inserção do par de identificadores na base de dados, o *DNS Handler* retorna o ID de 128 bits se a aplicação é IPv6, ou o *hash* de 32 bits do ID, se a aplicação é IPv4 (5). O diagrama de sequência detalhado contendo os módulos envolvidos está ilustrado na Figura B.1 do Apêndice B.

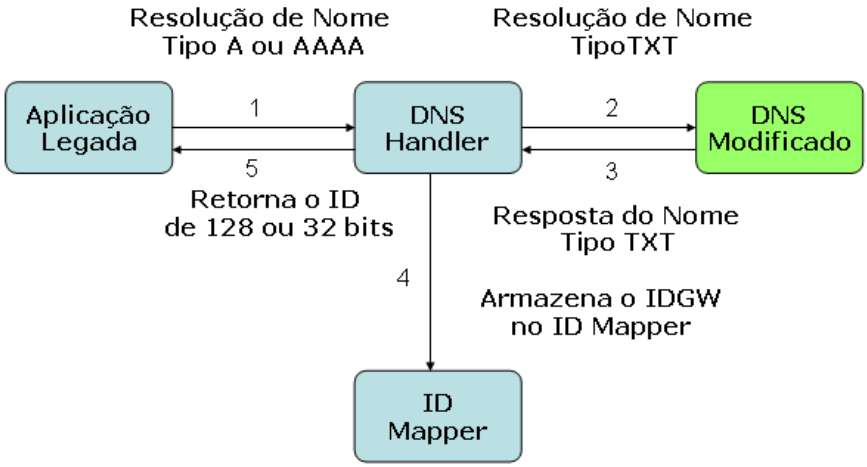

Fig. 3.23: Procedimento de resolução do nome.

Packet Handler

O módulo *Packet Handler* (PH) é responsável pela inserção e remoção da camada de identificação incluída nas mensagens pertencentes aos planos de dado e de controle. No fluxo de saída de pacotes, o PH consulta o módulo *ID Mapper* para obter o IDGW correspondente ao ID do pacote e encaminha para o módulo *Security*. Na recepção do pacote no destino, o PH retira o cabeçalho ID e, caso seja um pacote de dados, o módulo reinsere o pacote na pilha da rede para ser entregue à aplicação, caso contrário ele é enviado para o módulo responsável pela mensagem de controle.

ID Mapper

O módulo *ID Mapper* armazena o par de identificadores <ID, IDGW> obtidos pelo *DNS Handler* (passo 4 da Figura 3.23). Dado um ID, o *DNS Handler* retorna o IDGW associado para ser utilizado na criação do pacote ID. O armazenamento do IDGW é necessário pois as aplicações legadas não possuem conhecimento a respeito do GW-Core onde estão localizadas, necessitando que a arquitetura supra esta informação.

O *ID Mapper* possui duas políticas de reutilização das entradas no módulo. A primeira política baseia-se no algoritmo *Least Recently Used* (LRU) [34] e a segunda cria uma lista de entradas a serem removidas após um período de inatividade. A primeira é necessária para evitar a inserção de todas as entradas consultadas pelo nó, impedindo o crescimento excessivo da tabela. Cada entrada possui um *timestamp* indicando a sua última utilização e são organizados em uma fila. Caso a tabela esteja quase totalmente preenchida e há entradas na fila do LRU, então as primeiras entradas da fila são sobreescritas. A segunda política é necessária para evitar conflitos nos mapeamentos <ID, IDGW>. Isso pode ocorrer durante a migração do registro permanente do FQDN de um nó em um servidor DNS para outro, modificando o mapeamento <ID, IDGW>. Caso ainda exista uma entrada anterior no *ID Mapper*, este será utilizado e não refletirá a localização correta do GW-Core do servidor DNS.

Security Database

O módulo *Security Database* é uma base de dados interna utilizada para armazenar as informações do *Security Manager*. As informações armazenadas contêm como chave primária o ID do nó e são mapeadas em uma estrutura de dados que contém a chave pública do nó par, a chave simétrica de sessão trocada utilizando o protocolo DH e o tempo de validade da chave de sessão. A chave pública é utilizada para a verificação da assinatura digital durante a negociação das políticas de segurança e normalmente ela não muda. A chave simétrica é utilizada na criptografia simétrica e tem uma validade associada, sendo necessária a renovação periódica a fim de evitar ataques baseados na quebra das chaves por força bruta.

Security

O módulo *Security* é responsável pela aplicação das políticas de segurança estabelecidas pelo *Security Manager*. O módulo inspeciona o cabeçalho ID dos pacotes originados do *Packet Handler* para obter a chave simétrica do *Security Database* e aplicar o modelo de segurança proposto, como ilustra a Figura 3.24.

Na Figura 3.24 (a), o módulo *Security* recebe o pacote legado do módulo *Packet Handler* e cifra-o utilizando a chave simétrica estabelecida a partir do *Diffie-Hellman* para prover sigilo. No passo (b) o módulo computa o HMAC do pacote incluindo campos do cabeçalho ID ilustrados na Figura 3.25 (rachurados) e armazena no cabeçalho ID do pacote. No passo (c) o pacote é encapsulado em uma pacote IP+UDP e este é enviado até o destino (d), onde é desencapsulado e entregue para o módulo *Security* do nó par. O módulo verifica a integridade do pacote por meio do cálculo do HMAC e encaminha (e). Uma vez verificada a integridade, o pacote é decifrado e entregue ao módulo *Packet Handler* (f).

No caso da ausência de uma associação de segurança com o nó destino, o módulo *Security* envia uma mensagem para o módulo *Security Manager* solicitando o estabelecimento de uma associação

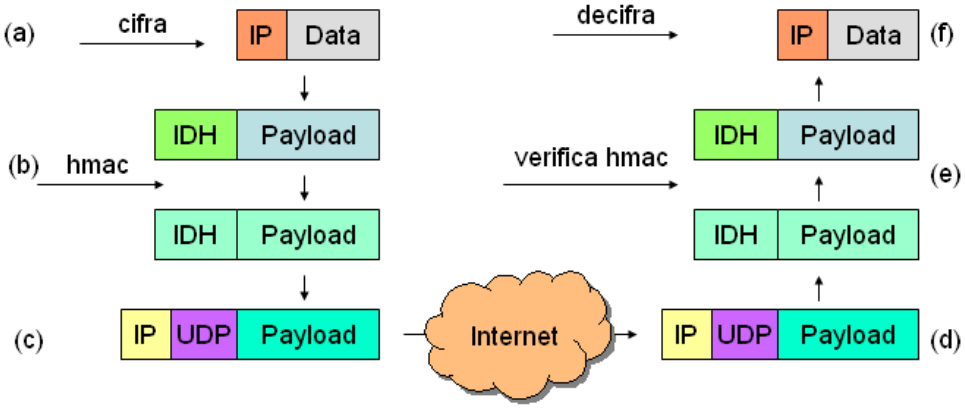

Fig. 3.24: Aplicação do modelo de segurança nos pacotes.

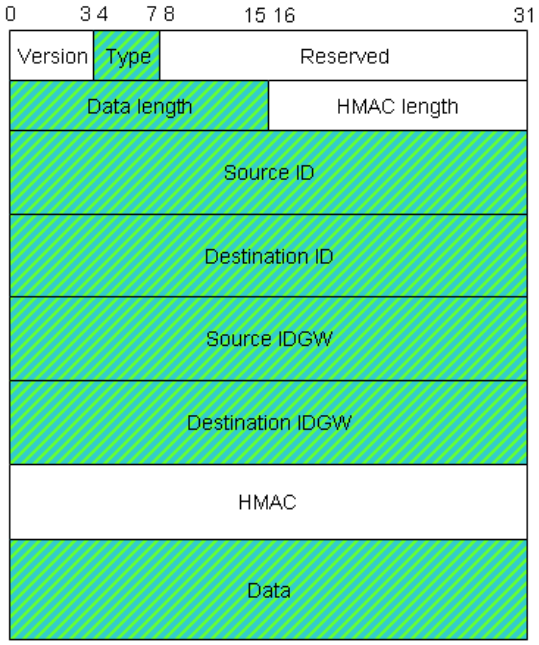

Fig. 3.25: Campos do cabeçalho ID utilizados no cálculo do HMAC.

de segurança com determinada entidade. Enquanto o *Security Manager* não finaliza uma associação segura com o destino, o módulo *Security* localiza a aplicação legada que originou os pacotes legados e a suspende até finalizar a associação para evitar a geração de pacotes. Caso a associação de segurança não seja estabelecida, a arquitetura retorna uma mensagem de ICMP *unreachable* para a aplicação, finalizando a transmissão de dados.

A arquitetura também possibilita o uso do modo oportunístico de segurança, ou seja, caso ambos os nós suportem a segurança, então ela é utilizada. Caso contrário, nenhum dos nós irá utilizar as funcionalidades de segurança. Além disso, a arquitetura permite somente a utilização da funcionalidade de autenticação das entidades, possibilitando a comunicação sem a criptografia. Este modo de

operação é interessante para dispositivos com pouca capacidade de processamento ou de aplicações que não necessitam de comunicação cifrada.

Peer Cache

O módulo *Peer Cache* (PC) é responsável pelo mapeamento de IDs em localizadores. O módulo é consultado pelo módulo *Routing* para a obtenção do localizador do próximo nó onde o pacote deve ser entregue para fins de roteamento, onde o próximo nó pode ser o destino, caso estejam localizados no mesmo domínio, ou o *gateways* de borda da arquitetura.

O módulo PC possui dois mecanismos adicionais utilizados na otimização do módulo. O primeiro é a resolução ativa do ID no localizador. Este procedimento inicia com a consulta a um ID que não consta na base interna. O PC adiciona o ID a uma tabela de espera de resoluções e envia uma mensagem para o plano de controle para realizar a resolução do ID. A tabela de espera impede que sucessivas requisições provoquem uma inundação de resoluções para um mesmo ID, retornando o localizador do GW de borda do domínio para as próximas consultas pelo mesmo identificador partindo do mesmo nó até o retorno do procedimento de resolução. Resolvido o localizador do nó de destino pelo plano de controle, o ID é retirado da tabela de espera e o localizador é inserido na tabela do PC. Consultas posteriores retornam o localizador do nó ao invés do localizador do GW do domínio.

O segundo mecanismo é a resolução proativa dos identificadores. Cada entrada no PC possui um tempo de validade. Após a sua expiração, a entrada é removida e é necessária uma nova resolução do ID para obter o localizador associado. O tempo de validade de uma entrada é dado pelo nó, onde um tempo de validade maior evita consultas sucessivas ao RVS e um tempo de validade menor possibilita a localização mais rápida do nó após a mobilidade. Para evitar a comutação na entrega dos pacotes entre o GW e o nó devido à expiração da entrada e consequente envio ao GW para o roteamento até o destino, o PC possui um mecanismo proativo de resolução prévia do ID no localizador. Para entradas que estão para expirar, o PC envia uma mensagem para o plano de controle solicitando uma nova resolução para aquele ID, evitando a expiração da entrada e a comutação na entrega dos pacotes.

Routing

O módulo *Routing* é responsável pelo roteamento do pacote ID até o seu destino. Ao receber pacotes do módulo *Security*, o módulo inspeciona o cabeçalho ID e consulta o módulo PC para obter o localizador associado àquele ID. Após receber o localizador de destino, o pacote é enviado à rede. Caso o nó seja um *Gateway* de domínio, ele realiza a consulta na sua tabela de roteamento após a consulta ao PC. Caso seja um *Gateway* de *Core*, além de realizar as consultas anteriores, ele consulta a tabela de registros e depois a tabela de mapeamentos da DHT. A tabela de registros contém todos os nós registrados abaixo do GW-Core e que estão sob outro GW-Core naquele instante. A tabela de mapeamentos da DHT contém mapeamentos IDGW em localizadores obtidos a partir de consultas à DHT.

Para pacotes ID recebidos da rede, o módulo inspeciona o cabeçalho ID e encaminha para o módulo *Security* se for o destino ou realiza o roteamento. No caso do nó com as funcionalidades básicas, ele somente encaminha para o GW do domínio. Caso seja um GW de domínio, ele consulta a tabela de roteamento para entregar o pacote. Caso não existam entradas, então o pacote segue via rota estática até encontrar um GW de domínio que conheça o nó ou até alcançar o GW-Core da árvore,

onde é realizado o roteamento baseado no IDGW de destino contido no pacote.

Mobility

O módulo *Mobility* faz parte do plano de controle da arquitetura e é responsável pela detecção dos eventos de mobilidade do nó. Inicialmente, ele se registra no *kernel* do sistema operacional para receber os eventos de alteração do localizador nas interface. Ao receber um evento, o módulo recupera a lista de nós pares com os quais possui comunicações ativas e espera por um sinal enviado pelo módulo *DHCP client* indicando o estabelecimento da conectividade com o novo domínio. Ao receber o sinal, ele envia uma mensagem de REDIRECT para todos os nós pares dentro do mesmo domínio em que ele se encontra e atualiza as entradas no *Peer Cache* relativas ao novo localizador no RVS. O diagrama de sequência dos passos executados na mobilidade intra-domínio está ilustrado na Figura B.5 do Apêndice B.

RVS Client

O módulo *RVS Client* é a interface de acesso do plano de controle da arquitetura com o RVS localizado no domínio. O módulo recebe mensagens do *Peer Cache* solicitando a resolução de um ID em um localizador. Para isso ele cria uma mensagem de controle destinada ao RVS do domínio para a resolução do ID. A resposta recebida é armazenada no módulo PC que é consultado pelo módulo *Routing*. O diagrama de sequência dos passos executados na resolução do identificador no localizador está ilustrado na Figura B.2 do Apêndice B.

DHT Client

O módulo *DHT Client* é a interface de acesso às funcionalidades da DHT localizada no *Core* sendo somente utilizado pelos GWs-Core. O módulo recebe mensagens de controle do módulo *Routing* para a resolução dos IDGWs de destino em localizadores dos GW-Cores. A resposta da DHT é armazenada na tabela de mapeamentos da DHT que atua como um *cache* das consultas à DHT.

DHCP Client

O módulo *DHCP Client* é responsável pela configuração inicial da arquitetura e pela reconfiguração após a mobilidade. Ele é composto de dois sub-módulos: um cliente DHCP e o próprio configurador em si. O cliente DHCP recebe os parâmetros da arquitetura enviados pelo servidor DHCP do domínio, e o segundo sub-módulo realiza o *parser* dos parâmetros e configura o nó. Ao ocorrer a mobilidade, o cliente DHCP envia uma mensagem para receber os novos parâmetros do domínio e atualiza os parâmetros dos módulos *DNS Handler* e *Peer Cache*.

Gateway Message Service

O módulo *Gateway Message Service* é responsável pela propagação das mensagens de REGISTRY até o *Core*. Além de realizar a propagação das mensagens, o módulo verifica o conteúdo da mensagem de REGISTRY recebida e adiciona o identificador contido na mensagem na sua tabela de roteamento, garantindo a sua alcançabilidade durante o roteamento.

Security Manager

O módulo *Security Manager* é responsável pela gerência das políticas de segurança utilizadas na comunicação entre os nós. O módulo recebe informações da camada de segurança para o estabelecimento de uma associação segura com um nó par ou com componentes da arquitetura e inicia a negociação através do envio da mensagem HELLO com destino ao nó par. O nó par recebe a mensagem HELLO e verifica a autenticidade do certificado. Verificada a autenticidade da identidade do nó emissor, o nó receptor gera os parâmetros DH a serem utilizados na comunicação e envia a mensagem HOWAU para o nó contendo o seu certificado, os parâmetros DH e a assinatura digital a fim de garantir a integridade. O nó par salva os parâmetros utilizados como semente para a geração dos parâmetros DH em uma base de dados interna. A origem recebe a mensagem de HOWAU, verifica a assinatura digital, gera os parâmetros DH para computar a chave simétrica a ser utilizada com o nó par e a armazena no módulo *Security Database*. O próximo passo é o envio da mensagem FINE, contendo os parâmetros DH utilizados, ao nó par juntamente com a assinatura digital. Caso ocorra uma falha na negociação, uma mensagem de segurança BYE é enviada ao nó comunicando a falha e indicando o tipo de erro. A Figura 3.26 resume o procedimento de autenticação e do estabelecimento da chave simétrica entre as entidades.

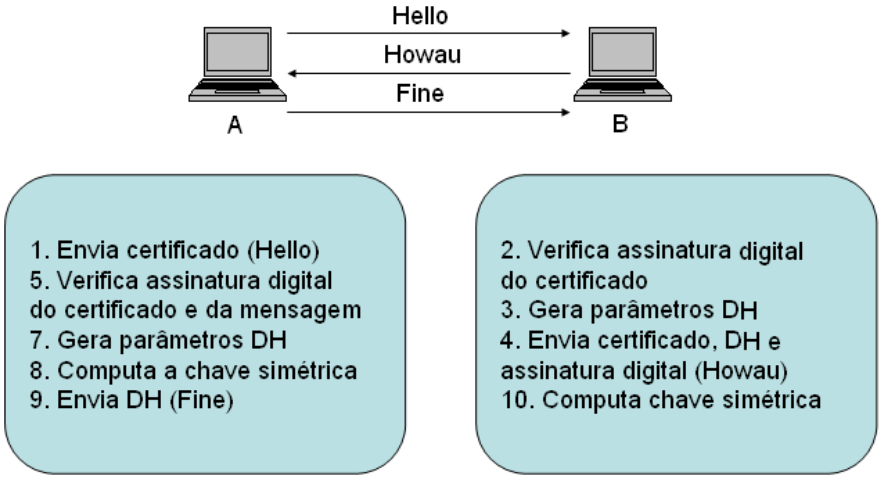

Fig. 3.26: Procedimento de autenticação e estabelecimento da chave simétrica.

3.3.2 Módulos Externos

Os módulos externos são componentes da arquitetura proposta que oferecem serviços externos aos nós da arquitetura. Dentre os serviços oferecidos estão a resolução de nomes em identificadores, a resolução de identificadores em localizadores e serviço de divulgação dos parâmetros do domínio.

Servidor DNS

O servidor DNS foi modificado para oferecer o serviço de resolução de nomes em identificadores. O servidor recebe requisições do módulo *DNS Handler* para a resolução de FQDNs em pares de

identificadores <ID, IDGW>. Para armazenar informações tais como os identificadores, o servidor DNS foi modificado para armazenar registros do tipo TXT no lugar de registros de tipo A ou AAAA. O registro do tipo TXT permite armazenar até 255 bytes de informação, ao contrário dos registros tipo A e AAAA que armazenam respectivamente 4 e 16 bytes de informação.

O servidor DNS é executado sobre um nó da implementação que possui todas as funcionalidades propostas. Dessa forma, o servidor recebe pacotes ID para a resolução de nomes via roteamento baseado no identificador, possibilitando cenários onde o servidor DNS esteja localizado em uma rede heterogênea.

Rendezvous Server

O *Rendezvous Server* (RVS) oferece o serviço de resolução de identificadores em localizadores e atua como um ponto de atualização de localizadores para eventos de mobilidade simultânea dentro dos domínios. O RVS permite o armazenamento de qualquer tipo de localizador, possibilitando o seu uso em ambientes heterogêneos. O servidor recebe requisições de resolução de identificador em localizador e atualizações de localizador diretamente do plano de controle da arquitetura. Todas as atualizações de localizador possuem tempo de validade com o objetivo de evitar o retorno de um localizador errado após a mobilidade de um nó para um outro domínio. Para a mobilidade simultânea dos nós dentro de um domínio, o RVS serve como ponto de atualização em comum entre os nós. Durante a mobilidade simultânea, ambas as mensagens de controle REDIRECT enviadas pelos nós irão se perder, forçando uma nova resolução no RVS após a entrada no *Peer Cache* expirar. Após a mobilidade e a obtenção dos novos localizadores, o nó atualiza suas informações no RVS o que o torna alcançável novamente e permitindo restaurar a comunicação com o nó par.

As operações de inserção e atualização das informações no RVS utilizam os mecanismos de segurança propostos na arquitetura. Para a inserção ou atualização, o nó deverá estabelecer uma associação de segurança com o RVS a fim de realizar a modificação. O modelo de segurança proposto impede ataques do tipo envenenamento de entradas no RVS, o que poderia resultar em outros ataques de segurança, como a personificação e os ataques de negação de serviço distribuído. Nós alvos realizam consulta no RVS e obtêm o localizador do atacante no lugar do nó destino. No ataque de negação de serviço distribuído, o atacante insere o localizador do alvo em todas as entradas do RVS. Resoluções de identidades retornarão o localizador do nó vítima do ataque, levando a impossibilidade do nó de responder a outras requisições. Somente as consultas ao RVS não necessitam de autenticação, pois o RVS é de acesso público.

Distributed Hash Table

A resolução de identificadores de GW-Core em localizadores é realizada por meio de consultas a uma *Distributed Hash Table* (DHT) localizada no *Core* da Internet. A DHT diferencia-se do RVS pois ela é uma base de dados distribuída em vários nós, garantindo maior escalabilidade e robustez. Os GW-Core registram os seus identificadores na DHT com a finalidade de habilitar o roteamento baseado no identificador no *Core*.

Servidor DHCP

O servidor DHCP oferece o serviço de divulgação de informações do domínio para automatizar o processo de (re)configuração de um nó móvel. O servidor atua como um servidor DHCP usual porém retorna parâmetros extras para o plano de controle do nó. Ao receber a solicitação dos parâmetros da rede, o servidor DHCP retorna os seguintes parâmetros:

- IDGW do Core-GW;
- ID do GW do domínio;
- Localizador do GW;
- ID do RVS;
- Localizador do RVS;
- ID do DNS;
- Localizador do nó.

3.3.3 Análise do modelo de segurança

O modelo de segurança proposto pela arquitetura oferece proteção contra vários ataques de segurança, listados a seguir:

- Personificação - Este ataque de segurança consiste na personificação de uma identidade alvo para a obtenção de benefícios, como informações sigilosas ou servir como porta de entrada para outros ataques. A arquitetura proposta repele este tipo de ataque ao utilizar os certificados digitais e de procedimentos de autenticação na atualização dos mapeamentos de identificadores em localizadores, evitando que o atacante se passe por uma outra identidade através da inserção de mapeamentos errados no RVS.
- *Connection Hijacking* - Este ataque consiste na tomada de controle de uma comunicação através da inserção de tráfego fraudulento no fluxo de dados. Este ataque é repelido através da utilização do estabelecimento de uma conexão segura utilizando a criptografia de chave simétrica e de códigos de autenticação de mensagens.
- Envenenamento de cache do RVS - Este ataque consiste na inserção de mapeamentos no RVS com o intuito de servir como base para ataques mais sofisticados, tais como personificação e ataques de negação de serviços. Para evitar este tipo de ataque, a arquitetura propõe o estabelecimento de associações de segurança do nó com o RVS e o GW do domínio, a fim de autenticar todas as inserções de mapeamentos no RVS.
- *Man-in-the-Middle* - Este ataque de segurança consiste na inserção do atacante no meio de uma comunicação entre duas entidades, obtendo as informações confidenciais. O modelo de segurança proposto evita este tipo de ataque ao utilizar os certificados digitais na autenticação das entidades, evitando que um atacante assuma uma identidade falsa.

- Ataque de negação de serviço distribuído - O ataque consiste no envio de tráfego a partir de várias máquinas, causando a sobrecarga da vítima e consequente incapacidade temporária de responder requisições legítimas. Este tipo de ataque é prevenido através da autenticação das mensagens de controle utilizadas pela arquitetura na atualização dos localizadores. A atualização errônea ou maliciosa pode levar ao redirecionamento do fluxo para um destino ou ao alvo de um ataque de segurança.

O modelo de segurança cobre os casos de ataques de segurança listados acima. No entanto, o modelo ainda não cobre os ataques de repetição, que consiste em um atacante armazenar pacotes legítimos da rede e depois reinserí-los na rede para benefício próprio. A proteção contra este tipo de ataque será realizada como trabalho futuro.

3.4 Resumo

O Capítulo 3 apresentou a proposta de implementação de uma arquitetura para a Internet de nova geração. Primeiramente, foram apresentados os requisitos baseados nos trabalhos relacionados citados anteriormente, como a introdução de uma camada de identificação, o roteamento baseado no identificador, novo mecanismo de resolução de nomes para suporte à camada de identificação e a gerência de localização para suporte à mobilidade. Uma vez atendidos a estes requisitos, a proposta de arquitetura oferece suporte nativo à mobilidade, ao *multihoming*, à segurança, à heterogeneidade e às aplicações legadas.

As funcionalidades propostas pela implementação são mapeadas em módulos para a separação de funcionalidades e o planejamento da implementação propriamente dita. São apresentados os módulos internos, responsáveis pelo provisionamento das funcionalidades dos nós da arquitetura, e os módulos externos, responsáveis pelo provisionamento de serviços para a implementação. Os módulos internos são divididos em plano de dados, responsável pelo suporte às aplicações legadas, inserção da camada de identificação, aplicação das políticas de segurança e pelo roteamento baseado na identidade; e plano de controle, responsável pela sinalização fim-a-fim para suporte aos serviços de mobilidade, segurança, resolução de identificadores, registro e sincronização.

Capítulo 4

Implementação e Resultados

Este capítulo apresenta a implementação da arquitetura proposta no capítulo anterior destacando os detalhes de cada módulo. A arquitetura foi implementada no espaço do usuário utilizando a linguagem C para maior compatibilidade dos mecanismos de acesso ao *kernel* do *Linux*. Os módulos foram implementados na forma de bibliotecas de ligação dinâmica com o objetivo de modularizar os componentes e permitir o seu carregamento na memória de forma dinâmica conforme o tipo de funcionalidade selecionada. São especificadas as tecnologias utilizadas no desenvolvimento do protótipo e exibidos alguns cenários de teste para validação da implementação. Os resultados obtidos serão apresentados e discutidos.

4.1 Módulos Internos

Os módulos internos da arquitetura foram implementados como bibliotecas de ligação dinâmica para modularizar as funcionalidades da arquitetura. Nós que possuem funcionalidades específicas, como o roteamento de pacotes, podem carregar dinâmicamente o módulo contendo as funções de roteamento. Esta decisão de projeto permite que somente as funcionalidades a serem utilizadas pelos nós sejam carregadas, reduzindo o tamanho total do protótipo na memória.

Os módulos internos são carregados por um programa principal chamado de *ID Daemon* (IDD). Antes de iniciar o IDD é necessário criar um arquivo de configuração utilizando *Extensible Markup Language* (XML) que contém as configurações iniciais do nó. Os campos obrigatórios são:

- *Type* - Define o tipo do nó. Conforme é escolhido o papel do nó, as bibliotecas relativas são carregadas na memória. As opções possíveis são NODE, RVS, GATEWAY, CORE e são definidas a seguir;
- *ID* - Identificador criptográfico do nó;
- *IDGWHome* - Identificador do *Gateway home* do nó;
- *PrimaryIface* - define a interface primária do nó cujo localizador será inserido no RVS.

Estes campos são características exclusivas de cada nó, sendo necessário o seu preenchimento manual. Os outros parâmetros são opcionais, isto é, pode-se utilizar o módulo de descoberta de parâmetros do domínio *DHCP Client* para a obtenção dos parâmetros do domínio listados abaixo:

- *DomainIDGW* - Identificador do GW do domínio;
- *LocatorGW* - Localizador do GW do domínio;
- *IDGW* - Identificador do *Gateway* atual do nó;
- *IDRVS* - Identificador do RVS;
- *LocatorRVS* - Localizador do RVS;
- *IDDHT* - Identificador da DHT no *Core*;
- *LocatorDHT* - Localizador da DHT no *Core*;
- *PortDHT* - Porta de serviço da DHT;
- *IDDNS* - Identificador do DNS do domínio;

A proposta de implementação identificou 4 tipos de nós: NODE, representa o nó simples da arquitetura que suporta todas as funcionalidades propostas; RVS que representa o nó que possui as funcionalidades básicas do nó mais os módulos extras de controle para o suporte à resolução de identificadores em localizadores; GATEWAY, representa o nó simples com suporte ao roteamento baseado no identificador e o CORE, que representa o nó tipo GATEWAY que está ligado ao núcleo. A Figura 4.1 mostra os módulos carregados conforme o tipo de nó escolhido.

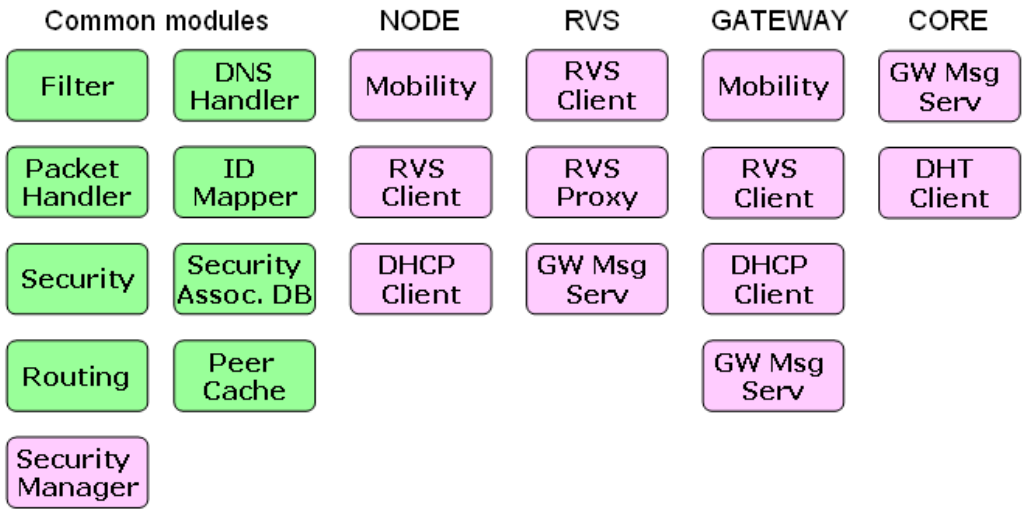

Fig. 4.1: Módulos carregados na memória conforme a funcionalidade selecionada.

O tipo NODE carrega os módulos extras para suportar a mobilidade, a resolução de identificadores e o acesso ao servidor DHCP através do *DHCP Client*. O tipo RVS somente carrega os módulos extras *RVS Client* para a resolução de identificadores no RVS e o RVS *proxy* para a tradução das mensagens de controle na comunicação com o RVS. Este tipo de nó não carrega os módulos de suporte à mobilidade nem o cliente DHCP pois assume-se que eles são estáticos no domínio. O tipo

GATEWAY carrega todos os módulos do tipo NODE mais o módulo *Gateway Message Service* para a propagação de mensagens de REGISTRY até os Core-GWs. Finalmente, o nó do tipo CORE carrega os módulos *Gateway Message Service* para a recepção das mensagens de REGISTRY e *DHT Client* para a comunicação com a DHT. Como ambos os *Gateways* são estáticos na arquitetura, não são carregados os módulos de suporte à mobilidade e do cliente DHCP.

A seguir descrevemos a interface virtual e a comunicação entre os módulos internos da arquitetura antes da descrição dos módulos.

4.1.1 Interface Virtual

A interface virtual é criada a partir do módulo TUN/TAP [35] do *Kernel* do Linux. O módulo oferece duas opções de interface virtual: a TUN, que emula um ponto de ligação na camada de rede e a TAP que emula um ponto de ligação na camada de enlace. A diferença entre elas é que o TUN é somente utilizado para conexões ponto-a-ponto, isto é, ela somente possui um único destino para todos os seus pacotes. O TUN é amplamente utilizado em protocolos ponto-a-ponto (*Point-to-Point Protocol* - PPP) para a conexão de um computador ao provedor de Internet. A TAP cria uma interface virtual na camada de enlace e recebe quadros *Ethernet*. Ela é utilizada pelo Linux na criação de *bridges* para a integração de duas sub-redes em uma única rede.

A arquitetura utiliza a interface virtual do tipo TAP pois ela pode receber e enviar de/para qualquer origem/destino. Ela é acessada a partir de uma chamada *open* do módulo TUN/TAP retornando o descritor de arquivo para a interface, que é utilizada para a inicialização do módulo através de uma chamada de sistema. Após a sua inicialização, são invocadas uma série de outras chamadas de sistema para a configuração da interface virtual. O código para a configuração se encontra no Apêndice A1.

A primeira chamada de sistema, TUNSETIFF, inicializa a interface virtual e configura o nome da interface definido no cabeçalho de configuração. Por padrão é chamado de ID0.

A segunda chamada de sistema, SIOCSIFADDR, configura o endereço da interface virtual. É configurado o seu endereço IP com o identificador de 32 e 128 bits do nó com a finalidade de emularmos uma camada de identificação utilizando a interface virtual. As aplicações legadas se ligam à interface virtual e recebem o endereço IP da interface virtual, que na verdade possui o identificador do nó. As aplicações são levadas a acreditar que o endereço retornado da interface virtual é um endereço IP, que na verdade é um identificador global de uma interface virtual. Como a interface virtual não muda de endereço IP, ela não é desligada do sistema e provê um ponto estático para as conexões durante eventos de mobilidade, *multihoming*, heterogeneidade e para a associações de segurança entre os nós.

A terceira chamada de sistema, SIOCSIFMTU, configura o MTU da interface virtual. Este parâmetro é importante para evitar a fragmentação dos pacotes durante a transmissão utilizando o IPv4. Na versão IPv6 não há fragmentação do pacote. Como os pacotes de ID possuem cabeçalho com tamanho de 88 bytes e o cabeçalho de tunelamento IP+UDP possui 28 bytes, então a interface virtual deverá ter MTU máxima de 1384 bytes [1] para evitar a fragmentação do pacote durante a transmissão. A fragmentação do pacote pode ocorrer em dois níveis: na camada de identificação ou na camada de rede. Na camada de identificação não há tratamento para fragmentação do pacote e, se um pacote com tamanho maior que 1412 bytes [2] chegar na interface virtual, ele será fragmentado pelo sistema

[1] (1500 bytes do quadro Ethernet - 88 bytes do cabeçalho ID - 28 bytes do cabeçalho IP+UDP)

[2] 1500 bytes - 88 do cabeçalho ID

operacional, porém, não será remontado no destino pois a implementação não trata a fragmentação do pacote ID. Dessa forma, o parâmetro MTU deverá ser menor que 1412 para o funcionamento do protótipo. Na camada de rede ocorrerá fragmentação se o pacote tiver tamanho superior a 1384 bytes. Neste caso não há problemas na recepção do pacote, uma vez que o próprio IP gerencia o processo de fragmentação. Apesar de não ocorrer nenhum problema na recepção, este cenário gera uma sobrecarga na rede devido à fragmentação do pacote IP.

A quarta e a quinta chamadas de sistema, SIOCGIFFLAGS e SIOCSIFFLAGS, recuperam e setam as *flags* da interface. Para habilitar a interface virtual no sistema operacional indicando que ela está ativa, o arquitetura recupera as *flags* da interface e adiciona a *flag* IFF_UP. Vale notar que não se pode simplesmente adicionar a *flag* diretamente pois há risco de apagar as *flags* anteriormente "setadas"para a interface. Após a passagem do novo *flag* para a interface virtual, o sistema operacional adiciona na sua tabela de roteamento a interface virtual pronta para enviar e receber pacotes da rede.

4.1.2 Comunicação entre os módulos

A comunicação entre os módulos é realizada através do *socketpair*, que é um mecanismo de comunicação interprocessos. A chamada *socketpair* da biblioteca *socket* recebe como parâmetro o domínio do *socket*, o protocolo a ser utilizado na comunicação e um vetor contendo os dois descritores de arquivos a serem conectados. A opção pela utilização do *socketpair* no lugar do *pipe* [3] é que o primeiro transmite os dados recebidos na forma de pacotes até o destino. A sobrecarga devido ao cabeçalho é desprezível pois o pacote não é enviado à rede, ficando restrito à própria máquina. A vantagem é a possibilidade de leitura do pacote na forma como ele foi enviado na origem. O *pipe* realiza função semelhante ao *socketpair*, porém sem encapsular em um pacote definido pelo protocolo. Dessa forma, o receptor tem problemas para receber o pacote como foi enviado, pois o *pipe* simplesmente concatena todas as informações recebidas na forma de um fluxo de bytes, não sendo adequado para a transmissão de pacotes entre os módulos do protótipo.

Filter

O módulo *Filter* é responsável pela captura de pacotes legados para serem tratados pela arquitetura. Ele utiliza a ferramenta *Iptables* [30] para a captura dos pacotes da rede e a biblioteca *libipq* para a transmissão dos pacotes do *kernel* para o espaço do usuário, onde é executado o protótipo.

Utilizamos o *Iptables* como a interface de entrada dos pacotes legados. Para diferenciar pacotes legados que desejam utilizar as funcionalidades da arquitetura, utilizamos as máscaras 1.0.0.0/8 e 1000::/16 para pacotes IPv4 e IPv6 respectivamente. Estas máscaras foram escolhidas porque os seus endereços são reservados de acordo com a IANA [36]. Dessa forma não há conflito entre um aplicativo não desejar utilizar as funcionalidades da arquitetura e enviar para estes endereços.

As regras utilizadas para a captura dos pacotes utilizando o *Iptables*:

```
(I) iptables -A OUTPUT -o id0 -s 1.0.0.0/8 -d ! localhost -j QUEUE
(II) ip6tables -A OUTPUT -o id0 -s 1000::/16 -d ! ::1/64 -j QUEUE
```

[3] O *pipe* é o mecanismo básico de comunicação interprocessos do Linux.

A regra (I) indica para o *Iptables* capturar todos os pacotes saindo pelo OUTPUT a partir da interface ID0 com máscara de origem 1.0.0.0/8 e cujo destino não seja o *localhost* e entregar para a fila de comunicação com o espaço do usuário. A regra (II) é semelhante à regra anterior porém é utilizada para a interceptação de pacotes legados IPv6.

A captura de pacotes legados é realizada no ponto OUTPUT, quando o pacote legado sai da aplicação e é encaminhado para o roteamento no sistema operacional. É escolhido este ponto de captura ao invés do POSTROUTING [4] [30] pois neste último ponto o sistema operacional realiza uma busca na tabela de roteamento pela interface de saída do pacote. Como há uma entrada na tabela de roteamento para a sub-rede 1.0.0.0/8 criada pela interface virtual, o sistema operacional realiza uma resolução ARP para obter o endereço MAC do próximo *hop*. Como não há necessariamente uma interface virtual no mesmo enlace, não há resposta e o sistema operacional bloqueia a saída dos pacotes de dados enquanto não é resolvido a resolução ARP, entrando em estado de *deadlock*. No ponto OUTPUT, o pacote acaba de sair da aplicação e não chega ao roteamento e, como resultado, não ocorre o problema mencionado. Uma vez capturados no OUTPUT, o *Iptables* envia os pacotes para a fila do *ip_queue*, que são lidos pelo módulo *Filter* e enviados ao módulo *Packet Handler*.

DNS Handler

O módulo *DNS Handler* é responsável pelo mecanismo de resolução de nomes em identificadores na implementação. As aplicações legadas passam a realizar a resolução de nomes localmente e o módulo se torna responsável pelo encaminhamento da requisição de resolução até o destino correto.

Para que as aplicações resolvam localmente os nomes, o arquivo *hosts* do *Linux* é modificado para indicar a própria máquina como caminho de resolução destinada à porta 53, correspondente ao serviço de DNS. O módulo *DNS Handler* atua na porta do DNS e aguarda as requisições de resolução de nome das aplicações legadas. Ao receber uma requisição, o módulo modifica o tipo de registro solicitado e envia para o servidor DNS da arquitetura. Os tipos de registros mais comuns solicitados ao DNS são o tipo A e o AAAA, referentes aos endereços IPv4 e IPv6, respectivamente. O módulo modifica o tipo de registro da solicitação original para TXT que suporta até 255 caracteres e pode armazenar o par <ID, IDGW>.

O módulo *DNS Handler*, ao receber a resposta do servidor DNS da arquitetura, analisa a resposta e armazena o mapeamento FQDN no par <ID, IDGW> na tabela *hash fqdn_table* e no módulo *ID Mapper*. A tabela *fqdn_table* atua como um *cache* do módulo *DNS Handler* armazenando as requisições mais recentes realizadas ao DNS.

ID Mapper

O módulo *ID Mapper* é responsável pelo armazenamento das informações obtidas do DNS. O módulo consiste de duas tabelas *hash* que possuem como chaves primária o ID de 32 ou 128 bits do nó de destino que mapeia em um IDGW. A escolha da tabela *hash* como estrutura para armazenamento dos mapeamentos se deve ao desempenho no tempo de busca por um conteúdo na tabela que, na

[4] O *Iptables* oferece 5 pontos de captura de pacotes: PREROUTING, INPUT, FORWARD, OUTPUT e POSTROUTING. O ponto de captura POSTROUTING se localiza após a tomada de rotamento pelo sistema operacional e antes de sair para a rede propriamente dita.

maioria dos casos, é O(1). O mapeamento ID em IDGW é utilizado pelo módulo *Packet Handler* para a criação do pacote ID, pois as aplicações legadas não possuem tal informação.

O módulo possui duas tabelas *hashs*: a primeira, chamada de *IDTable*, mapeia um ID de 128 bits em um IDGW também de 128 bits e a segunda, chamada de *ID32Table*, mapeia um ID de 32 bits no par <ID, IDGW>. Esta segunda tabela é utilizada para suporte às aplicações IPv4, pois estas somente possuem o ID de 32 bits no cabeçalho, sendo necessário recuperar o par <ID, IDGW> para preencher o cabeçalho ID. O identificador de 32 bits (ID32) é criado a partir da concatenação do byte 1 com os últimos 24 bits do ID de 128 bits, resultando no formato de 32 bits 1.X.X.X que é retornado então para as aplicações legadas.

As funções disponibilizadas pelo módulo são:

- *init_ID_mapper* - Inicializa o módulo *ID Mapper* criando as tabelas;
- *destroy_ID_mapper* - Finaliza o módulo *ID Mapper* liberando a memória alocada para as tabelas e seu conteúdo;
- *newIDEntry(ID, IDGW)* - Insere uma entrada na tabela *IDTable* utilizando como chave o ID do nó;
- *newIDEntry32(ID32, ID, IDGW)* - Insere uma entrada na tabela *IDTable32* utilizando como chave primária o ID32 do nó. Esta função é utilizada para suporte às aplicações IPv4;
- *getIDEntry(ID)* - Dado um ID, a função retorna o IDGW correspondente;
- *getIDEntry32(ID32)* - Dado um ID32, a função retorna uma estrutura contendo o par <ID, IDGW>.

A substituição das entradas utilizando a política *Least Recently Used* (LRU) ainda não foi finalizada e será implementada futuramente.

Packet Handler

O módulo *Packet Handler* é responsável pela inserção e remoção do cabeçalho ID e é implementado na forma de duas *threads*: a *Packet Handler Input* (PHI) e a *Packet Handler Output* (PHO).

A *thread* PHI recebe pacotes do módulo *Routing* e inspeciona o cabeçalho ID do pacote. Caso seja um pacote de controle, ela encaminha para o módulo responsável por aquela mensagem de controle. Caso seja um pacote de dado, a *thread* PHI reinsere o pacote na pilha de rede através da criação de *sockets* do tipo *RAW* com configurações específicas.

A primeira configuração necessária na criação do *socket* tipo *RAW* é a passagem de uma *flag* indicando o tipo de cabeçalho que sucede o cabeçalho IP. As *flags* utilizadas na arquitetura são IPPROTO_TCP, IPPROTO_UDP e IPPROTO_ICMP para indicar cabeçalhos TCP, UDP e ICMP, respectivamente. Sem a indicação correta do tipo do cabeçalho subsequente, o campo *Next Header* do cabeçalho IP não é preenchido corretamente e o pacote não será entregue à aplicação.

A segunda configuração necessária é a indicação para o *socket* do tipo *RAW* que o cabeçalho após o IP já está incluído no pacote. Isto é necessário pois a primeira configuração indicou para o *socket* que há um próximo cabeçalho. Dessa forma, o sistema operacional cria automaticamente o cabeçalho

indicado pela *flag* na criação do *socket*, sobrepondo ao cabeçalho original já contido no pacote. Para indicar que já existe um cabeçalho prévio, utiliza-se a chamada *setsockopt* na criação do *socket*.

Abaixo as linhas de código que indicam tal procedimento:

```
socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
setsockopt (raw_tcp, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
```

A *thread* PHO recebe os pacotes legados através do módulo *Filter* e as mensagens de controle internas do protótipo. Ao receber pacotes legados, a *thread* PHI consulta o módulo *ID Mapper* para a construção do cabeçalho ID. Para isso, a *thread* PHO analisa o cabeçalho ID para obter o ID de 32 ou 128 bits de destino que será utilizado como chave primária para a obtenção do IDGW correspondente. Após a criação do cabeçalho, o pacote é enviado para o módulo *Security*. As mensagens de controle são recebidas via *socketpair* dos módulos *Mobility*, *RVS Client*, *Gateway Message Service* e *Security*. Vale ressaltar que os módulos carregados dependem do tipo do nó especificado no arquivo de configuração. O módulo *Gateway Message Service*, por exemplo, somente é carregado nos nós do tipo RVS, GATEWAY e CORE.

Security

O módulo *Security* é responsável pela aplicação das políticas de segurança nos pacotes ID da implementação. O módulo aplica duas funções de segurança, a criptografia de chave simétrica para garantir a confidencialidade dos dados e códigos de autenticação de mensagem para garantir a autenticação e a integridade dos dados enviados. O algoritmo a ser utilizado com o nó par é obtido do módulo *Security Database* que armazena a chave simétrica trocada durante a fase do estabelecimento de associação de segurança.

O módulo é implementado utilizando a biblioteca *OpenSSL* [37] e é composto de duas *threads*. A *thread_security_output* recebe pacotes do módulo *Packet Handler* e busca no *Security Database* a chave simétrica a ser utilizada para a criptografia de chave simétrica e para o código de autenticação da mensagem. Ao retornar com a chave simétrica, a *thread* aplica, primeiramente, a criptografia seguida do cômputo do HMAC dos campos descritos no capítulo anterior. O HMAC é armazenado no cabeçalho ID e o pacote é encaminhado para o módulo *Routing*. Caso o retorno da requisição da chave no *Security Database* retorne nulo, a *thread* envia uma mensagem de controle para a *thread_security_control* para recuperar o *process ID* (PID) da aplicação que está gerando os pacotes e, em seguida, define seu estado como bloqueado até o estabelecimento da associação de segurança com o nó par. Esta última etapa será implementada futuramente.

A *thread_security_input* recebe pacotes do módulo *Routing* e busca pela chave simétrica associada ao ID do pacote. O módulo computa o HMAC do pacote e verifica se é o mesmo contido no cabeçalho ID, garantindo a integridade do dado enviado. Após a verificação, o pacote é decifrado e enviado para o módulo *Packet Handler*.

Security Database

O módulo *Security Database* armazena os parâmetros de segurança estabelecidos durante a negociação das associações de segurança, como a chave pública, a chave simétrica e a validade da chave

simétrica do nó par. O módulo é implementado utilizando uma estrutura de dados do tipo tabela *hash* e disponibiliza as seguintes funções para os outros módulos internos:

- *init_sdb* - Inicializa o módulo *Security Database* alocando memória para a estrutura *hash*;
- *destroy_sdb* - Finaliza a estrutura *hash* liberando o seu conteúdo da memória;
- *add_sdb_entry(ID, chaves)* - Adiciona uma entrada contendo o mapeamento de um ID em uma estrutura de dados contendo a chave pública, a chave simétrica e a validade da chave simétrica em segundos;
- *get_sdb_entry(ID)* - Recupera um mapeamento de um ID em uma estrutura contendo a chave pública e a chave simétrica.
- *del_sdb_entry(ID)* - Remove um mapeamento do *Security Database*.

O módulo *Security Database* realiza o gerenciamento da validade das chaves simétricas. Ao receber a solicitação de um mapeamento através da função *get_sdb_entry(ID)*, o módulo verifica a validade da chave simétrica no campo *validade* da estrutura da tabela *hash*. Se a entrada estiver expirada, o módulo a remove e envia uma mensagem de controle para o módulo *Security Manager* para o reestabelecimento de uma nova chave simétrica com o destino. A chave pública do destino não precisa ser reestabelecida, uma vez que ela é associada a uma identidade e não é modificada.

Routing

O módulo *Routing* é responsável pelo roteamento baseado na identidade provida pela arquitetura. O módulo provê duas funcionalidades distintas: a recepção de pacotes da rede para o encaminhamento e o roteamento na saída de pacotes para a rede. Para auxiliar a tomada de decisões de roteamento, são utilizadas três tabelas auxiliares: *Routing Table*, *Registry Table* e *DHT Table*. Além disso, utiliza-se um estrutura de dados especial chamada de tabela de espera com o objetivo de evitar a inundação de requisições.

A tabela de espera é uma estrutura de dados implementada com o objetivo de evitar a inundação de requisições de resolução de um mesmo identificador. A tabela consiste de uma tabela *hash* e de um temporizador para o armazenamento das requisições em andamento, evitando que o módulo *Routing* solicite ao módulo de controle *RVS Client* a resolução de um identificador no seu respectivo localizador para cada pacote que chega. Ao chegar a primeira requisição, verifica-se se o identificador está na tabela. Se não estiver, ele é inserido na tabela de espera e é enviada uma mensagem de controle para o módulo responsável pela resolução do identificador. Requisições sucessivas são ignoradas dentro de um intervalo de tempo estabelecido pelo temporizador. Ao chegar a resposta, o identificador é retirado da tabela de espera. Utilizou-se uma tabela *hash* com a finalidade de otimizar as buscas e um temporizador para evitar o estado de *deadlock* no caso da perda do pacote de resposta. Se a resposta não chegar, a entrada é removida depois de um intervalo de tempo possibilitando a resolução posterior do identificador. As funções disponibilizadas pela tabela de espera são:

- *init_queue_table(t)* - Inicializa a tabela de espera alocando memória para a tabela *hash*. Cada entrada possui validade de *t* milisegundos;

- *destroy_queue_table* - Finaliza a tabela de espera desalocando a memória alocada para a estrutura;
- *queue(ID)* - Insere o ID na tabela de espera;
- *unqueue(ID)* - Remove o ID da tabela de espera;
- *isQueued(ID)* - Verifica se o dado ID está na tabela de espera.

A estrutura de dados implementada auxilia a resolução de identificadores no módulo *Routing* e é utilizada no auxílio das resoluções de identificadores nos módulos *Peer Cache*, *RVS Client* e *DHT Client* com a finalidade de evitar solicitações sucessivas a um mesmo identificador que poderiam comprometer a escalabilidade do sistema.

A *Routing Table* é a tabela de roteamento interna do módulo *Routing*. A tabela é carregada para os tipos de nós GATEWAY e CORE e é populada através de mensagens de controle do tipo REGISTRY enviadas pelos nós. Todos os *Gateways* no caminho até o *Core* recebem essa mensagem de controle, adicionam a entrada na *Routing Table* e encaminham em direção ao Core-GW, permitindo que todos os *Gateways* no caminho conheçam o nó. As entradas na *Routing Table* possuem um tempo de vida definido pelo cabeçalho de configuração. Este parâmetro é importante para a mobilidade, removendo entradas antigas que não refletem a localização atual do nó. A *Routing Table* é implementada utilizando uma tabela *hash* para fins de desempenho e disponibiliza as seguintes funções:

- *init_rtable* - Inicializa a *Routing Table* alocando memória para a estrutura de dados;
- *destroy_rtable* - Finaliza a estrutura de dados liberando memória alocada dinamicamente para o sistema operacional;
- *newRTEntry(ID, locator)* - Adiciona uma mapeamento de um ID em um localizador na *Routing Table* com o *timestamp* atual;
- *staticRTEntry(ID, locator)* - Adiciona um mapeamento permanente de um ID em um localizador com o *timestamp* nulo na *Routing Table*. Os mapeamentos permanentes são utilizados para entradas que não expiram, como os mapeamentos do GW do domínio e do RVS, evitando que a rota estática e a consulta ao RVS sejam removidas da *Routing Table*;
- *getRTEntry(ID)* - Recupera o mapeamento da *Routing Table* para um dado ID.

A *Registry Table* é utilizada para rastrear os nós que estão registrados em um determinado GW-Core e se moveram temporariamente para outro GW-Core. A princípio, sem a mensagem de REDIRECT_CORE e da *Registry Table*, o nó não seria alcançável, pois a resolução do FQDN do nó de destino retornaria o IDGW de registro do nó e não o IDGW onde o nó se encontra atualmente. Para que o nó seja alcançável, os GWs-Core verificam todas as mensages de REGISTRY que chegam e verificam se o IDGW de registro é o mesmo do GW-Core *Home*. Caso não seja o mesmo, o GW-Core envia uma mensagem de REDIRECT_CORE para o GW-Core onde o nó está registrado. Ao receber a mensagem de controle, o GW-Core adiciona uma entrada na *Registry Table* contendo o localizador do GW-Core de onde o nó está localizado atualmente. As funções disponibilizadas pela *Registry Table* são:

- *init_registry_table* - Inicializa a *Registry Table* alocando memória para a estrutura de dados;
- *destroy_registry_table* - Finaliza a estrutura de dados liberando memória alocada dinamicamente para o sistema operacional;
- *addRegistry(ID, locator)* - Adiciona uma mapeamento de um ID em um localizador na *Registry Table* com o *timestamp* atual;
- *getRegistry(ID)* - Recupera o mapeamento da *Registry Table* para um dado ID.

A *DHT Table* atua como o *cache* das resoluções de IDGW em localizadores realizados na DHT localizada no *Core* com o auxílio de uma tabela de espera. Como as entradas na DHT são estáticas, as entradas na *DHT Table* não expiram, sendo necessário recorrer a políticas de substituição das entradas como a *Least Recently Used* (LRU) para a reutilização das entradas, como descrita no módulo *ID Mapper*. As funções disponibilizadas pela *DHT Table* são:

- *init_dhtable* - Inicializa a *DHT Table* alocando memória para a estrutura de dados;
- *destroy_dhtable* - Finaliza a estrutura de dados liberando memória alocada dinamicamente para o sistema operacional;
- *addDHTEntry(ID, locator)* - Adiciona uma mapeamento de um ID em um localizador na *DHT Table*;
- *getDHTEntry(ID)* - Recupera o mapeamento da *DHT Table* para um dado ID.

A recepção de pacotes da rede para encaminhamento é implementada com diferentes funcionalidades dependendo do tipo de nó definido no arquivo de configuração. São implementadas em duas *threads*:

- *thread_routing_input* - Esta *thread* é carregada somente pelos tipos de nós NODE e RVS e não realizam o roteamento. Ela recebe pacotes da rede na porta definida no cabeçalho do IDD (*default* 20000) e verifica o seu ID de destino. Caso seja o destino final, o módulo envia o pacote para o módulo *Security* para a verificação da integridade do pacote. Caso contrário, o pacote é descartado supondo que a *thread* recebeu o pacote devido a algum erro no serviço de entrega.
- *thread_routing_input_gw* - Esta *thread* é carregada pelos tipos de nós GATEWAY e CORE e realiza o roteamento sobre os pacotes que chegam da rede na porta definida no cabeçalho do IDD. O módulo inspeciona o ID de destino do pacote e, caso seja o destino final, o pacote é enviado para o módulo *Security*. Caso contrário, ele é enviado para a *thread* de roteamento do módulo *Routing* a ser descrito em seguida.

O roteamento dos pacotes na saída da rede é implementado de três formas diferentes para atender as diferentes funcionalidades dos nós, que neste caso são de três tipos: NODE (mesma funcionalidade que o tipo RVS), GATEWAY e CORE.

- *thread_routing_output* - Esta *thread* é carregada para nós do tipo NODE e RVS. A *thread* recebe pacotes do módulo *Security* e consulta o *Peer Cache* para a resolução do ID de destino do pacote no localizador do nó de destino mostrado no passo 1 da Figura 4.2 (a). Caso não seja encontrado o mapeamento para o ID de destino no *Peer Cache*, o pacote é enviado para o *Gateway* do domínio, como mostra o passo 2 da Figura 4.2 (a). Não é carregada nenhuma tabela de roteamento auxiliar uma vez que os tipos de nó NODE e RVS não possuem funções de roteamento.

- *thread_routing_output_gw* - Esta *thread* é carregada para nós do tipo GATEWAY. A *thread* recebe pacotes do módulo *Security* e da *thread_routing_input_gw* para a realização do roteamento. Para pacotes vindos do módulo *Security*, é considerado que o nó GATEWAY originou os pacotes para a comunicação com outro nó, desempenhando a funcionalidade de nó. Dessa forma, o roteamento consulta o *Peer Cache* para a resolução do ID de destino do pacote. Caso não seja encontrado o mapeamento, é realizada a consulta na *Routing Table* para a resolução do ID como mostra o passo 1 da Figura 4.2 (b). Caso a última resolução falhe, o pacote é encaminhado via rota estática até o GW sucessor como mostra o passo 2 da Figura 4.2 (b). Pacotes vindos da *thread_routing_input_gw* são roteados somente com a consulta à *Routing Table* e roteamento via rota estática até o próximo GW como mostra o passo 3 da Figura 4.2 (b). O *Peer Cache* não é consultado na tomada na decisão de roteamento pois o GW não mantém comunicação com o nó de destino, somente encaminha o fluxo de pacotes.

- *thread_routing_output_gw_core* - Esta *thread* é carregada para nós do tipo CORE e desempenha as mesmas funcionalidades da *thread_routing_output_gw*, utilizando duas tabelas auxiliares: a *Registry Table* e a *DHT Table*. Para pacotes destinados a uma outra árvore na borda do *Core*, o Core-GW da árvore consulta a *Registry Table* para verificar se o nó alvo está registrado no domínio porém se encontra momentaneamente em uma outra rede. Caso o retorno seja nulo, então é realizada uma resolução do ID de destino na DHT via o plano de controle, como mostra o passo 1 da Figura 4.2 (c). Após o retorno da resolução, é inserida uma entrada na *DHT Table* e o pacote é encaminhado até o destino. Caso não seja encontrado o mapeamento na DHT, o pacote é descartado pois é assumido que houve um erro na construção do pacote, uma vez que não existe um IDGW não mapeado na DHT. Pacotes vindos da *thread_routing_input_gw* são roteados primeiramente com uma consulta à *Routing Table*, seguido do *Registry Table* e finalmente à *DHT Table* como mostra o passo 3 da Figura 4.2 (c). Novamente, o *Peer Cache* não é consultado na tomada na decisão de roteamento pois o GW não mantém comunicação com o nó de destino, somente encaminhando o fluxo de pacotes.

Peer Cache

O módulo *Peer Cache* é responsável pela armazenamento dos mapeamentos de identificadores em localizadores das comunicações ativas do nó. Além do mapeamento, o módulo apresenta mecanismos extras para otimizar o funcionamento do módulo: a tabela de espera e a resolução proativa dos identificadores.

A requisição de resolução de um identificador inicia com a chegada do primeiro pacote ao módulo *Routing*, que invoca a função *getPCEntry* para a obtenção do localizador correspondente. Por ser a

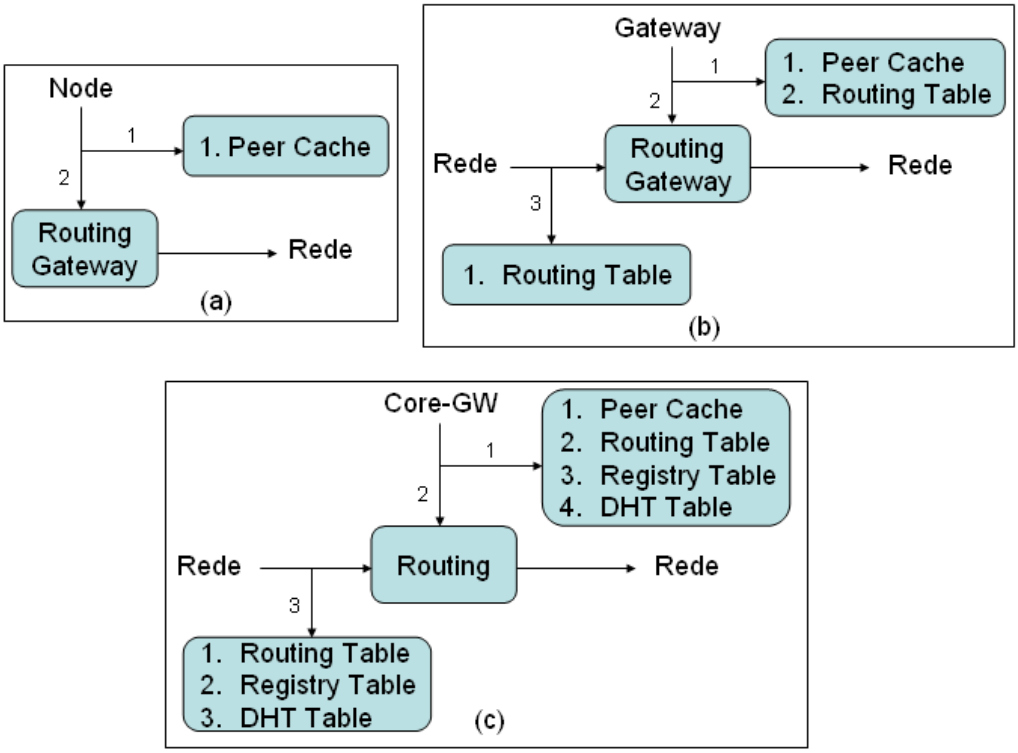

Fig. 4.2: Fluxograma de roteamento das *threads* do módulo *Routing*.

primeira requisição àquele identificador, o módulo não encontrará o mapeamento correspondente para o dado ID, retornando o localizador do GW do domínio e inserindo uma entrada na tabela de espera descrita anteriormente. Neste caso, a tabela de espera tem por objetivo evitar a inundação de requisições de resolução de um mesmo ID no RVS.

O gerenciamento dos mapeamentos válidos é realizado através da adição de validades para as entradas no *Peer Cache*. A validade das entradas é definida no arquivo de configuração. Quando uma consulta é realizada no *Peer Cache*, o módulo busca pela entrada e verifica a sua validade. Caso já tenha expirado, o módulo remove o mapeamento e a função retorna nulo.

A resolução proativa dos identificadores ocorre durante uma comunicação ativa. Para evitar o redirecionamento do fluxo do GW do domínio para o nó quando obtém-se o mapeamento e o retorno do fluxo para o GW quando ele expira, o módulo verifica as entradas que estão para expirar e envia uma mensagem de controle para a *thread_rvs_resolution* do módulo de controle *RVS Client* requisitando a resolução do ID da entrada a expirar. Dessa forma, evita-se o redirecionamento do fluxo entre o GW e o nó dentro do mesmo domínio.

O módulo também envia a mensagem de controle para a *thread_rvs_get* na resolução do ID do primeiro pacote. Ao chegar o primeiro pacote, o módulo retorna o localizador do GW do domínio e adiciona uma entrada no *Peer Cache* contendo o ID de destino e o localizador do GW e envia uma mensagem de controle solicitando a resolução do ID de destino. Ao receber o retorno da mensagem do RVS, o módulo *RVS Client* adiciona o mapeamento do ID no localizador atual do nó caso estejam no mesmo domínio, ou mantém o localizador do GW do domínio, caso o nó esteja em outro domínio.

Esta abordagem reduz o tempo de espera no envio de dados, uma vez que a aplicação não precisa aguardar o retorno do RVS. Caso o nó de destino esteja no mesmo domínio, o GW encaminhará diretamente para ele, caso contrário o GW encaminhará para o próximo GW via rota estática.

As funções disponibilizadas pelo módulo são:

- *init_peer_cache* - Inicializa o *Peer Cache* alocando memória para a estrutura de dados;
- *destroy_peer_cache* - Finaliza a estrutura de dados liberando memória alocada dinâmicamente para o sistema operacional;
- *newPCEntry(ID, locator)* - Adiciona um mapeamento de um ID em um localizador no *Peer Cache* com o *timestamp* atual;
- *staticPCEntry(ID, locator)* - Adiciona um mapeamento estático de um ID em um localizador no *Peer Cache* com o *timestamp* igual a zero. As entradas estáticas são utilizadas para os mapeamentos do GW e do RVS do domínio;
- *getPCEntry(ID)* - Recupera o mapeamento do *Peer Cache* para um dado ID;
- *getAllPCEntry* - Recupera todas as entradas ativas no *Peer Cache*. Esta função é utilizada em eventos de mobilidade para recuperar os IDs com os quais o nó mantém comunicação para o envio da mensagem de controle REDIRECT dentro de um domínio;
- delCMEntry(ID) - Remove a entrada com identificador ID do *Peer Cache*.

Mobility

O módulo *Mobility* é responsável pela detecção dos eventos de mobilidade do nó e o envio de mensagens de controle informando o novo localizador para os nós no mesmo domínio.

A detecção de mudança de localizador inicia com o registro no *kernel* para a recepção de mudanças de endereço na interface primária definida no arquivo de configuração. A comunicação com o *kernel* é realizada através de um *socket* do tipo *Netlink*. A opção de projeto em escolher o *Netlink* no lugar das chamadas de sistema do tipo *ioctl* é a característica assíncrona do *Netlink*, que permite a abertura de um *socket* para a espera da mensagem notificando a alteração do endereço da interface. Caso fosse utilizada chamadas de sistema *ioctl*, seria necessário a invocação periódica para verificar as alterações no endereço da interface, o que resultaria na redução do tempo de resposta do sistema devido ao intervalo de amostragem. O código que ilustra a criação de um *socket* do tipo *Netlink* se encontra no Apêndice A2.

Ao receber uma mensagem do *kernel*, o módulo verifica se o tipo de mobilidade é intra-domínio e se os nós pares se encontram no mesmo domínio para o envio da mensagem de controle REDIRECT informando o seu novo localizador. Além disso, o nó envia uma mensagem de RVS_UPDATE ao RVS para a atualização do seu novo localizador. Caso a mobilidade seja do tipo inter-domínio, o nó envia uma mensagem de RVS_UPDATE para o registro no novo domínio e remove as entradas antigas do *Peer Cache* para evitar o envio de pacotes com destino aos localizadores antigos dos nós pares.

Security Manager

O módulo de controle *Security Manager* é responsável pelo estabelecimento de uma associação de segurança provendo suporte para o módulo *Security* no estabelecimento da chave simétrica de sessão com o destino. O módulo é composto de uma *thread* principal, a *thread_security_manager*, que gerencia a troca de mensagens de segurança entre o nó e o seu destino.

A *thread* recebe mensagens de controle do módulo *Security* através de um *socketpair* e inicia o processo de estabelecimento da associação de segurança com o nó par através do envio da mensagem de controle HELLO. Chamamos de I o inicializador do procedimento de segurança e R o nó par. Ao receber a mensagem HELLO, R verifica a assinatura digital contida no certificado enviado na mensagem, extrai a chave pública de I e insere no módulo *Security Database*. Após a verificação de autenticidade da identidade de I, R cria a mensagem de HOWAU contendo os parâmetros *p, g,* e *pub_key* do protocolo *Diffie-Hellman* (DH) e o seu certificado digital e assina digitalmente a mensagem HOWAU com a sua chave privada antes de enviar a I. Os parâmetros DH gerados são armazenados em uma tabela *hash* para serem utilizados na geração da chave simétrica de sessão entre os nós.

I recebe a mensagem HOWAU e verifica a autenticidade do certificado digital. Verificada a autenticidade da identidade de R, I extrai do certificado a chave pública de R e armazena no *Security Database*. O módulo verifica a integridade dos parâmetros DH transmitidos na mensagem através da assinatura digital contida na mensagem e gera a chave simétrica a ser utilizada na comunicação entre os nós. Por fim, I cria a mensagem FINE para transmitir os parâmetros DH locais e assina digitalmente a mensagem.

R recebe a mensagem FINE, verifica a assinatura digital e computa a chave simétrica a ser utilizada na comunicação segura entre os nós finalizando o procedimento de estabelecimento da associação de segurança. O módulo *Security Manager* envia a mensagem BYE em qualquer etapa da troca de mensagens de segurança caso ocorra algum erro durante o estabelecimento da associação de segurança, como por exemplo, erro na assinatura digital, certificado digital não-reconhecido ou parâmetros DH não-íntegros.

RVS Client

O módulo RVS Client é responsável pela interface de acesso ao RVS. Para isso, o módulo implementa cinco *threads* diferentes que são carregados dependendo do tipo de nó. Para nós do tipo NODE e GATEWAY, as seguintes *threads* são carregadas:

- *thread_rvs_update - Thread* responsável pelo envio periódico de mensagens de RVS_UPDATE para atualização do localizador no RVS.

- *thread_rvs_get - Thread* responsável pela recepção de mensagens de controle do módulo *Peer Cache* solicitando a resolução de identificadores e a criação de uma requisição de resolução para o dado ID no RVS.

Para nós do tipo RVS são carregadas, além das *threads* anteriores, três *threads* auxiliares. A primeira *thread* é responsável pela criação das mensagens de controle REGISTRY e as duas últimas

implementam a interface de acesso ao RVS. Estas duas *threads* criam um formato especial de mensagem a ser enviada para o RVS pois assumimos que este possui diferentes *Application Programming Interfaces* (APIs).

- *thread_rvs2gw* - Esta *thread* é responsável pela recepção de mensagens de RVS_UPDATE enviada pelos nós do domínio e também é responsável pelo envio de mensagens de REGISTRY para o GW do domínio informando o registro do nó no RVS do domínio.

- *thread_rvs_proxy_send* - Esta *thread* atua como a interface de acesso ao RVS externo à arquitetura. Ao receber uma requisição de resolução no RVS, a *thread* converte a mensagem de controle ID em uma mensagem RVS e envia para o *localhost*. Neste caso estamos considerando que o RVS esteja localizado na mesma máquina onde o aplicativo IDD esteja está executando. A conversão da mensagem de controle ID na mensagen RVS permite que o RVS não esteja localizado na mesma máquina onde se encontra o aplicativo.

- *thread_rvs_proxy_recv* - Esta *thread* recebe as respostas do RVS via *socket* na porta de controle definida no cabeçalho do IDD (*default 20001*) e converte a mensagem RVS na mensagem de controle ID.

Os nós do tipo CORE não carregam o módulo *RVS Client* pois acessam a DHT para a resolução de identificadores em localizadores no *Core*.

DHT Client

O módulo *DHT Client* é a interface de acesso à DHT localizada no *Core* e somente é carregado pelos nós do tipo CORE. A DHT utilizada é a Bamboo DHT [38] que implementa o algoritmo Tapestry [29] e disponibiliza a seguinte interface de acesso utilizando chamadas de *Remote Procedure Call*.

- *put* - Dada uma chave, um valor e um tempo de vida, ele insere na DHT via RPC.

- *get* - Dada uma chave, a DHT retorna o valor associado.

As operações de acesso à DHT pelo módulo *DHT Client* são realizadas através das seguintes *threads*:

- *thread_dht_update* - Esta *thread* inicia uma conexão com a DHT utilizando RPC e atualiza em intervalos de tempo definidos no arquivo de configuração o seu localizador na DHT.

- *thread_dht_get* - Esta *thread* aguarda a recepção de mensagens de resolução de IDGW em localizadores enviados pelo módulo *Routing* para o roteamento no *Core*. Ao receber um IDGW, a *thread* abre uma conexão RPC com a DHT e envia uma requisição de resolução do IDGW em um localizador. Ao receber a resposta, este armazena na *DHT Table* do módulo *Routing*.

Gateway Message Service

O módulo *Gateway Message Service* é responsável pela propagação das mensagens de controle REGISTRY até o Core-GW do domínio. O módulo é composto pelas seguintes *threads*:

- *thread_gw_listener* - *Thread* carregada nos nós do tipo GATEWAY que recebe mensagens REGISTRY de outros GW e RVS. Ao receber a mensagem de REGISTRY, a *thread* adiciona uma entrada na *Routing Table* referente ao identificador e localizador contidos na mensagem REGISTRY e a envia para a *thread_gw_talker*.

- *thread_gw_core_listener* - *Thread* carregada nos nós do tipo CORE que recebe mensagens de REGISTRY e de REDIRECT_CORE de outros GW e RVS. Ao receber a mensagem REGISTRY, o Core-GW verifica o IDGW da mensagem. Caso seja o mesmo do Core-GW, então o nó está registrado no domínio e o Core-GW adiciona uma entrada na sua *Routing Table*. Caso não seja o IDGW da árvore, o Core-GW envia uma mensagem de REDIRECT_CORE destinada ao IDGW *Home* do nó notificando que determinado ID está localizado atualmente sob o seu IDGW. Ao receber uma mensagem de REDIRECT_CORE, a *thread* verifica se o IDGW está correto e adiciona uma entrada na *Registry Table* do módulo *Routing*.

- *thread_gw_talker* - *Thread* carregada somente nos nós do tipo GATEWAY e é responsável pela criação das mensagens de REGISTRY. Recebe mensagens da *thread_gw_listener* e envia para o GW do próximo domínio.

DHCP Client

O módulo *DHCP Client*, implementado através de dois sub-módulos, é responsável pela solicitação dos parâmetros do domínio durante a inicialização do nó, ou após um evento de mobilidade, e a sua configuração com os novos parâmetros recebidos.

O primeiro sub-módulo consiste de um cliente DHCP modificado para receber os parâmetros extras enviados pela arquitetura. Ao realizar uma requisição DHCP, o servidor DHCP retorna os parâmetros padrões adicionados dos parâmetros extras da arquitetura que são recebidos pelo cliente DHCP. O cliente DHCP não realiza o tratamento dos parâmetros extras, uma vez que isto é específico da implementação dos fabricantes. Para adicionar tal funcionalidade, o cliente DHCP foi modificado para interceptar os parâmetros extras da mensagem DHCPACK e enviar via um *socket* UNIX para o segundo sub-módulo.

O segundo sub-módulo é implementado através de uma *thread* que recebe os parâmetros extras e configura o nó. A *thread_dhcp* recebe os parâmetros extras, realiza o parser da mensagem para a extração dos parâmetros e reconfigura o nó enviando mensagens internas de controle para os módulos *DNS Handler*, *Packet Handler, Mobility, RVS Client, Routing, ID Mapper* e *Peer Cache* atualizando os identificadores de GW de domínio, RVS do domínio e DNS do domínio e os seus respectivos localizadores.

4.2 Módulos Externos

Servidor DNS

O servidor DNS provê suporte para a arquitetura na resolução de nomes em identificadores. O servidor armazena as informações na sua base de dados utilizando registros do tipo TXT, que suporta até 255 caracteres por entrada. Dessa forma é possível o armazenamento do par de identificadores <ID, IDGW>, ambos de 128 bits, no registro do tipo TXT naturalmente suportado pelo DNS.

A implementação utilizada foi o BIND versão 9 e ela foi alterada para atuar na porta 54 ao invés da 53, porta padrão do serviço DNS. Esta modificação foi introduzida para possibilitar o uso de um servidor DNS na arquitetura, uma vez que se ela atuasse na porta 53, entraria em conflito com o módulo *DNS Handler* que atua na mesma porta. Neste cenário, as aplicações legadas que não desejam utilizar as funcionalidades da arquitetura podem utilizar o serviço DNS direcionando as requisições para a porta 54 do servidor DNS.

Rendezvous Server

O RVS provê o serviço de resolução de identificadores em localizadores da tecnologia de rede de cada domínio. Todos os nós, ao chegarem em um domínio, se registram no RVS para que se tornem alcançáveis por outros nós. O RVS foi implementado utilizando a linguagem de programação C e consiste de uma tabela *hash* interna que mapeia identificadores em localizadores. Cada entrada dessa tabela possui uma validade e, expirando o seu prazo, tal entrada é descartada. Dessa forma, nós que porventura tenham migrado para outro domínio, ou que tenham sido desligados, deixam de ter entrada no RVS automaticamente. Além disso, a existência do RVS permite cenários de mobilidade onde dois nós se movem simultâneamente.

Servidor DHCP

O servidor DHCP provê o mecanismo de passagem dos parâmetros na arquitetura. Ele provê os parâmetros necessários para a configuração inicial e após a mobilidade de um nó. Para isso ele envia, juntamente com os parâmetros do DHCP, as informações do domínio como o GW, o RVS e o DNS, permitindo ao nó descobrir estes serviços.

O servidor DHCP utilizado é o *dhcp3-server* e o seu arquivo de configuração foi modificado para retornar os parâmetros extras. Utilizamos o parâmetro *Root Path* do DHCP para armazenar todos os parâmetros do domínio a serem retornados para o módulo *DHCP Client* no nó.

Distributed Hash Table

A DHT oferece o serviço de resolução de identificadores em localizadores no *Core*. A DHT possui como característica a sua base de dados distribuída e o roteamento dos dados baseado em um algoritmo de roteamento como o *Chord*, *Pastry*, *Tapestry* entre várias propostas. A vantagem da utilização de uma DHT no *Core*, no lugar de uma base de dados única e centralizada como o RVS, é a maior escalabilidade e robustez provisionada para a implementação, diminuindo o tempo de resposta às requisições de resolução de identificadores e evitando a presença de um único ponto de falha para o sistema de resolução de identificadores.

Outra característica favorável à utilização da DHT no *Core* é a sua indepedência à tecnologia de camada de rede específica, podendo ser adotada em ambientes heterogêneos. Cada nó da DHT pode se situar em um domíno heterogêneo, porém provisionando um sistema homogêneo de resolução de identidades em localizadores dos Core-GWs.

A DHT utilizada pela arquitetura é a Bamboo DHT [38] e ela disponibiliza uma interface de acesso às primitivas *get* e *put* utilizando o protocolo RPC, descritos na subseção *DHT Client*. Outras interfaces de acesso utilizando tecnologias de rede diferentes podem ser implementadas para realizar requisições na DHT.

A opção pelo Bamboo se deve ao fato dela estar instanciada no *PlanetLab* [39] com o nome de *openDHT* [26] podendo ser acessada para inserção e buscas, possibilitando a realização de testes de desempenho utilizando uma DHT global. Outro cenário possível é a utilização da DHT para a ligação de dois laboratórios localizados em regiões geográficas distintas para a comunicação em uma rede sobreposta.

4.3 Resultados

Os testes realizados nesta seção têm por objetivo validar a arquitetura proposta e avaliar o protótipo implementado, analisando os seguintes pontos:

- *overhead* do protótipo;
- suporte à mobilidade;
- modelo de segurança;
- robustez;
- suporte às aplicações legadas.

Para os testes envolvendo cenários de mobilidade no protótipo, há dois tempos envolvidos nos resultados: o tempo de associação com a antena e o tempo total da configuração via DHCP. Para o primeiro caso, utilizamos um *notebook* com uma placa integrada DellNIC com tempo de associação de 1835 ms, que é dependente do tipo de *hardware* e do *driver*. O segundo tempo associado nos eventos de mobilidade é o tempo de configuração via DHCP. Este tempo oscila devido a, por exemplo, perda de pacotes de configuração do DHCP e possivelmente das resoluções ARP. Para o cenário envolvendo a mobilidade simultânea dos nós, foi utilizada uma placa sem fio DellNIC com acesso via interface USB e tempo de associação de 1310 ms.

O primeiro teste realizado tem por objetivo verificar o *overhead* do protótipo com relação a arquitetura da Internet atual. Para isso, foi transferido um arquivo de 224 MB utilizando a ferramenta *Secure Copy* (SCP) sem a utilização do protótipo em redes cabeadas e sem fio. Em seguida foi realizado o mesmo experimento utilizando o protótipo. Para o experimento foram utilizadas duas máquinas Pentium IV 3.0 GHz, 1 GB de memória RAM utilizando Linux Debian. A Tabela 4.1 mostra a a média os valores obtidos após dez medições.

Comparando os resultados da Tabela 4.1, verificamos o *overhead* de 16,61% e 9,32% nas redes cabeadas e sem fio, respectivamente. Este processamento extra deve-se principalmente ao tratamento

Tab. 4.1: *Overhead* do protótipo na arquitetura atual.

Protótipo	Tipo	Taxa (MB/s)	Desvio	Tempo (s)	Desvio
Não	cabeado	11.2	0	20.10	0
	sem-fio	2.58	0.32	107.14	19.62
Sim	cabeado	9.54	0.22	23.44	0.33
	sem-fio	2.36	0.15	115.62	17.37

do pacote no espaço do usuário. A diferença entre os impactos de processamento nas redes cabeadas e sem-fio deve-se à taxa de transferência entre os nós. O protótipo possui uma taxa de recepção de pacotes legados limitada pelo tempo de execução dado pelo sistema operacional. Pacotes que não conseguem ser entregues ao protótipo são armazenados em *buffers* e, se estes estiverem cheios, os pacotes são descartados pelo sistema operacional. Dessa forma, o impacto do protótipo é maior em redes cabeadas do que redes sem-fio devido a maior taxa de transferência alcançada e consequente maior descarte dos pacotes legados pelo sistema operacional.

O segundo teste realizado visa avaliar o protótipo quanto o suporte à mobilidade. Para isso, avaliamos o protótipo em dois testes de desempenho nos cenários descritos na Figura 4.3. No primeiro teste, foi avaliado o tempo de transferência de um arquivo entre dois nós utilizando o SCP e, no segundo teste, foi avaliada a perda de pacotes utilizando um fluxo UDP.

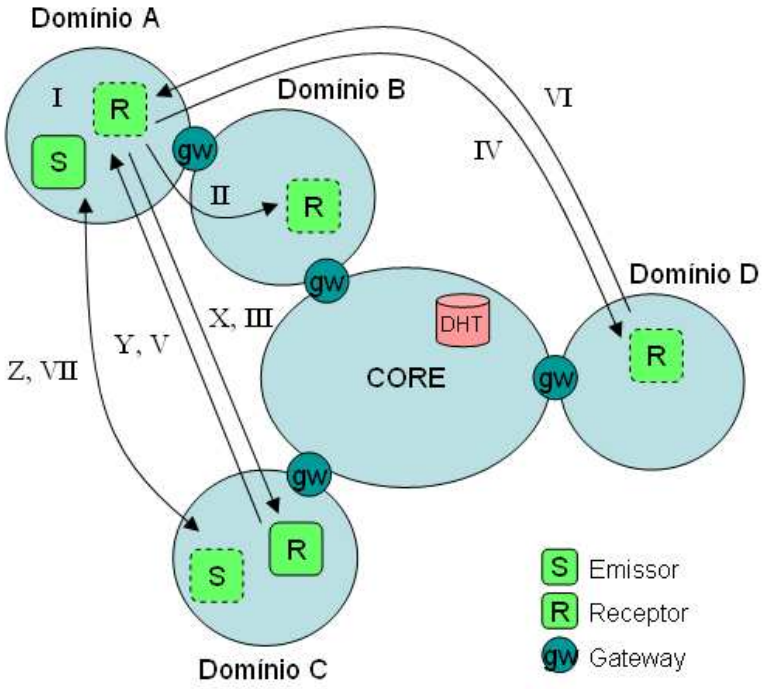

Fig. 4.3: Cenários de mobilidade avaliados no protótipo.

Para os testes, foram utilizadas 5 máquinas Pentium IV 3.0 GHz HT 1 Gb RAM para representar os quatro domínios (A, B, C, D) e um nó estático emissor (*Sender* - S) com uma interface aérea para o *double jump*, e um computador portátil Pentium Duo 1.8 Ghz 1 Gb RAM para ser o nó móvel receptor

(*Receiver* - R). Cada máquina estática utilizou uma máquina virtual Qemu [40] para emular o DNS e o RVS do domínio, não incluídos na figura.

Os eventos de mobilidade possuem dois tempos associados: o tempo de associação com a antena e o tempo de resposta do DHCP. Para os testes, foram desconsiderados os tempos de associação com a antena uma vez que estes tempos são dependentes do tipo do hardware e do *driver*. A interface aérea utilizada no experimento é um DellNIC e tem um tempo de associação de 1825 ms. O tempo de resposta do DHCP varia no protótipo devido a fatores externos, como a perda de pacotes em redes sem fio, como mostrado nos resultados.

A Tabela 4.2 mostra os resultados no tempo de transferência do arquivo. Para a obtenção dos valores, o experimento foi repetido dez vezes. O cenário X mostra a mobilidade inter-domínio com os dois nós inicialmente no mesmo domínio A, e R se move para o domínio C. O cenário Y mostra a mobilidade inter-domínio onde ambos os nós estão nos seus domínios *home* e R se move para o domínio A. O cenário Z mostra a mobilidade simultânea inter-domínio dos nós.

Tab. 4.2: Tempo de transferência de um arquivo utilizando o SCP.

Cenário	Taxa (MB/s)	Desvio	tempo (s)	Desvio	Associação (ms)	Desvio
X	2.18	0.04	101.46	1.69	2313.31	293.05
Y	2.18	0.04	102.42	1.94	2118.19	277.19
Z	1.56	0.15	144.72	13.9	1876.64	331.48

Os cenários X e Y mostram o atraso introduzido pela mobilidade e pela janela deslizante do TCP. Durante o evento de mobilidade, a perda de pacotes reduz a janela deslizante de transmissão do TCP, diminuindo a taxa de transferência entre os nós. O cenário Z mostra a mobilidade simultânea dos nós, que é o pior caso dentre os analisados pois o nó S somente enviará pacotes para o IDGW correto após a recepção do primeiro TCP ACK do nó R informando o IDGW atual. Este cenário também é influenciado pela janela deslizante do TCP.

A Tabela 4.3 mostra a perda de pacotes em uma transmissão de pacotes UDP entre dois nós em cenários de mobilidade ilustrados na Figura 4.3. O teste foi realizado utilizando o gerador de tráfego JTG [41], com tamanho de pacotes de 1300 Bytes e uma taxa constante de transmissão de 20 Mb/s durante 60 segundos. O experimento foi repetido 10 vezes e a média calculada. O cenário I mostra a mobilidade intra-domínio entre S e R dentro do domínio A. O cenário II mostra a mobilidade inter-domínio com R se movendo para o domínio B. O cenário III mostra ambos os nós no domínio A e R retorna para o seu domínio *home*. O cenário IV mostra ambos os nós no domínio A, e R move para um domínio estrangeiro D. O cenário V mostra S e R nos seus domínios *home* A e C, respectivamente, quando R move para o domínio A. O cenário VI mostra S no seu domínio *home* A, R em um domínio estrangeiro D e S move para o domínio A. O cenário VII mostra a mobilidade simultânea inter-domínio dos nós, onde S inicia no domínio A, R no domínio C e ambos trocam de domínio.

O tempo de associação da tabela refere-se ao tempo de associação da antena sem fio com o ponto de acesso e o tempo do DHCP refere-se ao tempo de resposta e configuração do protótipo ao chegar a um novo domínio. O tempo total considerado é a soma de ambos os tempos, e este representa o instante de tempo após o evento de mobilidade que o protótipo está pronto para receber pacotes.

O cenário I possui a menor perda de pacotes devido à mensagem de controle REDIRECT trocado entre os nós pares. A perda de pacote neste caso considera o tempo de associação com a antena (4% de

Tab. 4.3: Perda de pacotes UDP.

Cenário	Perda(%)	Desvio	Assoc. (ms)	Desvio	DHCP (ms)	Desvio	Total (ms)	Desvio
I	6.09	0.92	1841.73	19.59	726.96	299.25	2568.68	306.84
II	15.46	1.60	1855.14	16.83	579.32	363.94	2434.46	365.49
III	16.81	2.85	1857.73	16.99	486.37	167.28	2344.10	162.68
IV	18.46	1.72	1847.36	13.76	775.50	192.56	2622.86	190.99
V	7.23	1.48	1847.25	17.42	376.19	315.33	2223.44	315.71
VI	16.93	4.90	1825.03	11.49	598.93	276.14	2423.96	281.63
VII	25.78	4.56	1835.28	5.50	543.11	251.26	2378.40	251.76
	-	-	1309.34	6.57	563.40	330.36	1872.74	327.90

perda em 60 segundos com a placa do experimento) e o tempo que a mensagem de REDIRECT leva para chegar ao nó par. O cenário V mostra uma pequena perda de pacotes devido ao procedimento de registro do nó ao chegar no novo domínio. Como o nó se move para o mesmo domínio do emissor e se registra no *gateway* do domínio, os pacotes destinados a ele são entregues imediatamente ao nó.

Os cenários II, III, IV e VI são influenciados pelo tempo de validade das entradas no módulo *Peer Cache* nas transmissões unidirecionais e passando pelo *Core*, pois ele utiliza o mecanismo de *timeout* para buscar pelo novo localizador. O tempo de *caching* da entrada é variável, que no caso dos testes foi configurado para 5 segundos, afetando a restauração da comunicação de 0 a 5 segundos dependendo da validade da entrada quando o evento de mobilidade ocorreu. Como resultado, é esperado uma perda de pacotes e um desvio padrão maiores nos cenários de mobilidade que cruzam o *Core* do que nos cenários de mobilidade intra-domínio. O cenário VII possui dois tempos de associações: o primeiro se refere à placa sem fio do *notebook* e o segundo se refere à placa sem fio DellNIC com interface USB ligado ao nó receptor. Este cenário apresenta o pior desempenho pois ambos os nós precisam limpar as entradas no *Peer Cache* antes que a restauração da comunicação seja realizada.

O terceiro teste realizado avaliou o impacto do modelo de segurança utilizado na arquitetura. Primeiramente realizamos a comunicação entre duas máquinas sem a utilização do modelo de segurança da arquitetura. Em seguida utilizamos o modelo de segurança proposto, com a autenticação das entidades utilizando os certificados digitais, o HMAC das mensagens trocadas e o algoritmo de chave simétrica *Blowfish* para a criptografia do canal de comunicação. Para o experimento utilizamos duas máquinas Pentium III 1 GHz, 128 MB de memória RAM utilizando Linux Debian. A Tabela 4.4 apresenta a média dos resultados obtidos após 10 experimentos. Para este teste utilizamos o *TinyCA* [42] para a instanciação de uma autoridade certificadora e a emissão de certificados digitais. O *TinyCA* é um *front-end* da biblioteca *OpenSSL* que simplifica a gerência de certificados digitais.

Tab. 4.4: *Overhead* do protótipo utilizando o modelo de segurança.

Protótipo	Taxa (MB/s)	Desvio
Não	92.88	0.12
Sim	22.71	0.12

Os resultados da Tabela 4.4 mostram o *overhead* do modelo de segurança da arquitetura. Este resultado se deve ao procedimento de criptografia e cômputo do HMAC de todos os pacotes utilizados

na comunicação. Vale ressaltar que o desempenho do modelo de segurança tende a melhorar em máquinas dotadas de maior capacidade de processamento, uma vez que a computação das funções de criptografia é mais veloz em tais máquinas.

O quarto teste avaliou a robustez do protótipo. Para isso, realizamos um teste de esforço que consistiu em transmitir o tráfego entre dois nós para verificar o vazamento de memória, o que pode exaurir o sistema operacional, reiniciando-o devido à falta de memória. Utilizamos o software *valgrind* [43] para detectar vazamentos de memoria e o utilitário *lsof* do sistema operacional para detectar o número de descritores de arquivos abertos, corrigindo eventuais falhas. Os nós foram submetidos a um tráfego TCP na carga máxima utilizando a ferramenta *Iperf* [44] durante um período de 24 horas e verificamos o seu correto funcionamento sem qualquer vazamento de memória.

Finalmente, o quinto teste avaliou o suporte às aplicações legadas. Para isso, utilizamos o servidor de difusão de mídia *Icecast* [45] para a distribuição de músicas. Uma máquina cliente recebia a música e a tocava em tempo real. Durante a transmissão, a máquina cliente se movia para uma outra rede e verificamos a diminuição do *buffer* de armazenamento da música por um instante de tempo e depois o *buffer* retornava à posição original.

4.4 Resumo

O Capítulo 4 apresentou a implementação do protótipo detalhando os mecanismos e tecnologias empregadas. O protótipo foi avaliado nos testes de sobrecarga do protótipo, suporte à mobilidade, sobrecarga do modelo de segurança, robustez e suporte às aplicações legadas. O teste de sobrecarga do protótipo na arquitetura atual avaliou o *overhead* do protótipo ao ser utilizado na arquitetura atual. O teste de suporte à mobilidade avaliou o protótipo em diversos cenários de mobilidade, como a intra-domínio, inter-domínio, inter-domínio cruzando o *Core* e mobilidade simultânea dos nós. O terceiro teste avaliou o *overhead* do protótipo utilizando o modelo de segurança da arquitetura. O quarto teste avaliou a robustez do protótipo sob condições de esforço para verificar o vazamento de memória. Finalmente, o quinto teste avaliou o suporte de aplicações legadas com a utilização de um servidor de difusão de áudio.

Capítulo 5

Conclusões

Esta seção apresenta as considerações finais referentes ao trabalho desenvolvido nesta dissertação e discute as possíveis extensões relativas ao protótipo implementado.

5.1 Considerações Finais

Este trabalho propôs uma implementação de uma arquitetura para a Internet com a finalidade de solucionar problemas conhecidos e prover novos serviços para a Internet atual. A arquitetura oferece opções para vários problemas atuais na Internet, através da introdução de uma camada de identificação entre a camada de rede e a camada de transporte com identificadores de 128 bits, criando um espaço de nomes maior para atender aos requisitos atuais no número de identificadores disponíveis. A segurança é oferecida através de um modelo que atua em conjunto com a camada de identificação, provendo um modelo integrado de autenticação, integridade e sigilo dos dados.

Ao introduzir a nova camada de identificação, a arquitetura soluciona o problema clássico da sobrecarga semântica do endereço IP. O IP atua como identificador na camada de transporte e como localizador na camada de rede, criando uma dependência entre as camadas da pilha de protocolos de rede. Esta dependência viola o princípio da evolução das camadas de forma independente uma das outras e retarda a introdução de novos serviços na Internet atual. A introdução da camada de identificação provê suporte a serviços como a mobilidade e o *multihoming* são suportados de forma natural, e abre caminho para novos serviços como o suporte integrado da segurança baseado na identidade e na interconexão de domínios heterogêneos.

O modelo de segurança proposto atua sobre a camada de identificação e oferece mecanismos de autenticação, integridade e sigilo na comunicação entre os nós. Ao atuar sobre a camada de identificação, a segurança é naturalmente herdada para o cenário de mobilidade sem a necessidade de reestabelecimento dos parâmetros de segurança. Além disso, possibilita a heterogeneidade de forma natural, pois os nós dos domínios heterogêneos são identificados por uma camada homogênea de identificação. Os identificadores são ligados dinamicamente às tecnologias de rede de cada domínio e os *gateways* de borda tradutores de tecnologia realizam a ponte entre os domínios heterogêneos.

O desenvolvimento do protótipo validou as idéias propostas para a implementação de uma arquitetura para a Internet de nova geração. O protótipo foi avaliado em vários testes como a robustez e o suporte à mobilidade e às aplicações legadas. Ele se mostrou robusto quanto aos mecanismos de

suporte à mobilidade como a gerência de localização para rastreio dos nós. Além disso, foram implementados vários mecanismos auxiliares para a otimização do protótipo, como o plano de controle para redireção do fluxo no *Core* ou mensagens de redireção informando o novo localizador após o evento de mobilidade.

5.2 Trabalhos Futuros

5.2.1 Roteamento

Uma característica importante da arquitetura NodeID é o fato de utilizar roteamento estático na direção do *Core* para o tráfego entre nós situados em domínios distintos. A opção pelo roteamento estático implica no registro de cada nó em cada um dos *Gateways* situados acima na hierarquia. Isto implica, no caso do *Gateway de Core*, o registro de todos os nós situados naquela árvore. Uma forma de reduzir esta sobrecarga consiste na introdução do identificador de domínio. Neste caso, o encaminhamento dos pacotes ao destino não será mais baseado no identificador do *Gateway de Core* e no identificador do nó de destino, mas sim, no identificador do domínio onde se encontra o nó e no identificador deste último. Isto implica no registro dos identificadores de todos os domínios na DHT do *Core*. Associado a cada identificador de domínio na DHT teremos o localizador do *Gateway no Core* onde se encontra o domínio. Atualmente, estamos realizando estudos voltados para o roteamento *flat* com o objetivo de permitir uma maior flexibilidade na interconexão dos domínios situados em uma mesma árvore de domínios, ou entre domínios interconectados em árvores distintas.

5.2.2 *Multihoming* e Qualidade de Serviço

A arquitetura oferece suporte ao *multihoming* ao permitir a troca dinâmica entre vários provedores de acesso à Internet sem a quebra de conectividade. Uma extensão interessante para a arquitetura é a implementação de um modelo de *multihoming* onde a tomada de decisões entre os vários provedores seja baseado em algum critério baseado na qualidade de serviço, como largura de banda disponível e tempo de atraso. A arquitetura passa a disponibilizar para os usuários os melhores *links* de acesso aos provedores de forma transparente, ou até mesmo o usuário pode definir a qualidade de serviço mínima desejada.

5.2.3 Redes auto-configuráveis

A arquitetura suporta diferentes funcionalidades para cada tipo de nó selecionado, como o serviço de resolução de identificadores no domínio no RVS e a propagação de mensagens de registro entre os *Gateways*. Uma possível extensão para a arquitetura é a definição dos papéis de cada nó de forma dinâmica, como por exemplo em uma PAN. Neste cenário, os nós se unem e realizam uma eleição baseada em critérios como capacidade de processamento para definir um *Gateway* e um RVS da nova rede criada. Conforme outros dispositivos são adicionados à rede, os papéis de cada nó podem ser modificados dinâmicamente sem uma configuração manual dos nós e de forma transparente para o usuário final. Isto é possível graças à implementação dos módulos na forma de bibliotecas de ligação dinâmica.

5.2.4 Composição de domínios

A arquitura atualmente suporta todos os tipos de mobilidade de nós de forma transparente ao usuário. Um cenário futuro que pode ser explorado na arquitetura é a mobilidade de domínio para realizar a composição dos mesmos. Esta composição poderá ser realizada no plano de dados, caso a tecnologia de rede seja homogênea, ou a composição somente ocorrerá no plano de controle, com a negociação dos parâmetros a serem utilizados na comunicação entre os dois domínios. Atualmente, o protótipo está sendo estendido para suportar esta funcionalidade e já suporta a mobilidade de domínios.

5.2.5 Provisionamento de serviços

A arquitetura introduz um modelo de roteamento baseado na identidade, criando uma rede sobreposta que pode ser utilizada para o provisionamento de novos serviços, como o *multicast* e a composição de serviços. No caso do *multicast*, um identificador pode ser utilizado na identificação de um grupo que deseja receber uma divulgação de mídia. Na composição de serviços, um identificador pode ter como destino outro identificador antes de chegar ao destino final, com o agente intermediário provendo um serviço de translação de tecnologia de rede ou a tradução de um formato de mídia em outro.

5.2.6 Segurança

A arquitetura provê mecanismos de segurança para os planos de dado e controle na troca de informações entre os nós e componentes da arquitetura. Uma extensão natural do modelo de segurança é o suporte à negociação dos parâmetros de segurança desejados levando em conta a capacidade de processamento ou o nível de segurança requerido. PDAs e computadores portáteis precisarão utilizar algoritmos de criptografia e compactação que não acarretem em consumo excessivo dos recursos computacionais dos dispositivos.

5.2.7 Heterogeneidade

A arquitetura apresentou um modelo para o suporte natural da heterogeneidade dos domínios. Uma extensão natural da arquitetura é a implementação do modelo de suporte as redes heterogêneas com a integração, inicialmente, das redes IPv4 e IPv6. Uma possibilidade consiste na criação de um modelo de interface para o suporte à qualquer tecnologia de rede, bastando ter a biblioteca que implemente as funções deste protocolo. Atualmente já está implementada uma versão IPv6 do protótipo que, em conjunto com o protótipo IPv4, possibilitará testar a integração das redes heterogêneas IPv4-IPv6.

Referências Bibliográficas

[1] P. Srisuresh, M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. *RFC 2535*, August 1999.

[2] S. Frankel. *Demystifying the Ipsec Puzzle*. Artech House Computer Security Series. Artech House Publishers, April 2001.

[3] C. Perkins. *Mobile IP Design Principals and Practices*, volume 1st edition. Prentice Hall PTR, January 15, 1998.

[4] J. Abley, K. Lindqvist, E. Davies, B. Black and V. Gill. IPv4 Multihoming Practices and Limitations. *RFC 4116*, July 2005.

[5] R. Moskowitz, P. Nikander. Host Identity Protocol (HIP) Architecture. *RFC 4423*, May 2006.

[6] W. Wong, R. Pasquini, R. Villaca, L. de Paula, F. Verdi, M. Magalhães. A Framework for Mobility and Flat Addressing in Heterogeneous Domains. 25^o *Simposio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007), Belém, Pará, Brasil*, 28 de Maio a 1 de Junho 2007.

[7] B. Ahlgren, J. Arkko, L. Eggert, J. Rajahalme. A Node Identity Internetworking Architecture. *Proceedings of the 9th IEEE Global Internet Symposium, In conjunction with IEEE Infocom*, April 2006.

[8] Henrik Abramowicz. D1-A1 Ambient Networks Project Description and Dissemination Plan. January 2004.

[9] S. Lloyd, C. Adams. *Understanding the Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations*, volume 1st edition. Sams, November 12, 1999.

[10] J. Saltzer. On the Naming and Binding of Network Destinations. *RFC 1498*, August 1993.

[11] D.Clark, K. Sollins, J. Wroclawski, and T. Faber. Addressing Reality: an Architectural response to real-world demands on the evolving Internet. *In ACM SIGCOMM 2003 FDNA Workshop*, August 2003.

[12] William Stallings. *Cryptography and Network Security*. Prentice Hall, November 16, 2005.

[13] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose. DNS Security Introduction and Requirements. *RFC 4033*, March 2005.

[14] T. Henderson. End-Host Mobility and Multihoming with the Host Identity Protocol. *Draft-IETF, http://www.ietf.org/internet-drafts/draft-ietf-hip-mm-05.txt*, March 2007.

[15] J. Jung et all. DNS Performance and the Effectiveness of Caching. *IEEE trans. net*, volume 10:589–603, October 2002.

[16] S. Schmid, L. Eggert, M. Brunner and J. Quittek. Towards Autonomous Network Domains. *Proc. 8th IEEE Global Internet Symposium, Miami, FL, USA*, March 17-18 2005.

[17] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe and A. Warfield. Plutarch: An argument for Network Pluralism. *Proc ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA), Karlshule, Germany*, pages 258–266, August 2003.

[18] I. Stoica, D. Adkins, S. Zhuang, S. Shenker and S. Surana. Internet Indirection Infrastructure. *Proc ACM SIGCOMM, Pittsburgh, PA, USA*, pages 73–88, August 2002.

[19] D. Clark, R. Branden, A. Falk and V. Pingali. FARA: Reorganizing the Addressing Architecture. *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA, Karlshule, Germany)*, pages 313–321, August 2003.

[20] P. Nikander, J. Arkko, B. Ohlman. Host Identity Indirection Infrastructure (Hi3). *Proc. Second Swedish National Computer Networking Workshop 2004 (SNCNW2004), Karlstad University, Karlstad, Sweden*, Nov 23-24 2004.

[21] M. Cesar, K. Lakshminarayanan, T. Condie, I. Stoica, J. Kanna, S. Shenker. ROFL: Routing on Flat Labels. *In Proceedings of the ACM SIGCOMM, Pisa, Italy*, pages 363–374, September 1-15 2006.

[22] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, K. Wehrle. OCALA: An Architecture for Supporting Legacy Applications over Overlays. *Proc. 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '06) San Jose, CA*, 2006.

[23] I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM SIGCOMM, San Diego, CA*, pages 149–160, August 2001.

[24] P. Ganesan, K. Gummadi and H. Garcia-Molina. Canon in G major: designing DHTs with hierarchical structure. *ICDCS*, March 2004.

[25] Xiaoyun Wang and Hongbo Yu. *Lecture Notes in Computer Science*, volume 3494 of *Lecture Notes in Computer Science*, chapter How to Break MD5 and Other Hash Functions, pages 19–35. Springer Berlin / Heidelberg, May 11, 2005.

[26] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, H. Yu. OpenDHT: A Public DHT Service and Its Uses. *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, Philadelphia, Pennsylvania, USA*, pages 73–84, 2005.

[27] L. Eggert, J. Laganier, M. Liebsch, M. Stiemerling. HIP Resolution and Rendezvous Mechanisms. *Workshop on HIP and Related Architecture, Washington, DC, USA*, November 2004.

[28] A. Rowstron, P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-peer Systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, pages 329–350, November 2001.

[29] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, J. Kubiatowicz. Tapestry: a Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected areas in communications*, volume 22(number 1), January 2004.

[30] Iptables - The netfilter.org project. *http://www.netfilter.org/*.

[31] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley Professional, October 13, 2000.

[32] A. Alshamsi and T. Saito. A technical comparison of IPSec and SSL. *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference*, 2:395–398, March 2005.

[33] L. McLaughlin III. A Standard for the Transmission of 802.2 Packets over IPX Networks. *RFC 1132*, November 1989.

[34] Andrew Tanenbaum. *Modern Operating Systems*, chapter 6. Prentice Hall, February 28, 2001.

[35] M. Krasnyansky and M. Yevmenkin. Universal Tun/Tap Driver. *http://vtun.sourceforge.net/tun/*.

[36] IANA - Internet Assigned Number Authority. *http://www.iana.org*.

[37] OpenSSL: The Open Source toolkit for SSL/TLS. *http://www.openssl.org*.

[38] The Bamboo Distributed Hash Table. *http://bamboo-dht.org/*.

[39] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak and M. Bowman. PlanetLab: an Overlay Testbed for Broad-coverage Services. *ACM SIGCOMM Computer Communication Review*, Volume 33:3–12, July 2003.

[40] Qemu. *http://fabrice.bellard.free.fr/qemu/*.

[41] Jugi's Traffic Generator. *https://hoslab.cs.helsinki.fi/savane/projects/jtg/*.

[42] TinyCA. *http://tinyca.sm-zone.net/*.

[43] J. Seward N. Nethercote. Valgrind: A Program Supervision Framework. *Electronic Notes in Theoretical Computer Science*, 2003.

[44] Iperf - Internet Performance. *http://dast.nlanr.net/Projects/Iperf/*.

[45] Icecast - Media Streaming Server. *http://www.icecast.org/*.

Apêndice A

Interface virtual e recepção de eventos do Kernel

A1. Exemplo de interface virtual

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/if_tun.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "nid.h"
#include "tools.h"

int init_tap(char *dev, NID nid, NID32 nid32)
{
  int s = new_socket();
  int tapfd = 0;
  u_int flags = 0;
  u_int mask = 0;
  struct ifreq ifr;
  struct sockaddr_in *sin_pt  = (struct sockaddr_in *) &ifr.ifr_addr;
  char aux[128];

  if ((tapfd = open("/dev/net/tun", O_RDWR)) < 0){
```

```
  perror("can't open tap");
  exit(0);
}

bzero(&ifr, sizeof(struct ifreq));

ifr.ifr_flags = IFF_NO_PI | IFF_TAP;
memcpy(ifr.ifr_name, dev, strlen(dev));

if (ioctl(tapfd, TUNSETIFF, (void *)&ifr) < 0){
  perror("TUNSETIFF");
  exit(0);
}

bzero(&ifr, sizeof(struct ifreq));

sin_pt->sin_family = AF_INET;
memcpy(ifr.ifr_name, dev, strlen(dev));
memcpy(&sin_pt->sin_addr, &nid32, 4);

if (ioctl(s, SIOCSIFADDR, &ifr) < 0){
  perror("SIOCSIFADDR");
  exit(0);
}

printf("-------- Virtual Interface ----------\n");
inet_ntop(AF_INET, &sin_pt->sin_addr, aux, 128);
printf("Set device %s with address %s\n", dev, aux);

bzero(&ifr, sizeof(ifr));

memcpy(ifr.ifr_name, dev, strlen(dev));
ifr.ifr_mtu = DEVICE_MTU;

if (ioctl(s, SIOCSIFMTU, &ifr) < 0){
  perror("SIOCSIFMTU");
  exit(0);
}

printf("Set MTU to %d\n", DEVICE_MTU);

bzero(&ifr, sizeof(ifr));
memcpy(ifr.ifr_name, dev, strlen(dev));
```

```
  if (ioctl(s, SIOCGIFFLAGS, &ifr)){
    perror("SIOCGIFFLAGS");
    exit(0);
  }

  flags = mask = IFF_UP;

  if ((ifr.ifr_flags ^flags) & mask) {
    ifr.ifr_flags &= ~mask;
    ifr.ifr_flags |= mask & flags;

    if (ioctl(s, SIOCSIFFLAGS, &ifr) < 0){
      perror("SIOCIFFLAGS");
      exit(0);
    }
  }

  printf("Set device flag to UP\n");

  close(s);
  return tapfd;
}
```

A2. Exemplo de *socket netlink*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <asm/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <linux/netlink.h>
#include <linux/rtnetlink.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <net/if.h>

#include "nid.h"
#include "connection_mapper.h"
#include "configuration.h"
```

```
#define MAXBUFLEN 256

extern int sp_mo2ph[2], sp_ph2co[2];

static int open_netlink()
{
  struct sockaddr_nl addr;
  int nl_socket;

  bzero(&addr, sizeof(addr));

  if ((nl_socket = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE)) < 0){
    perror("socket Netlink");
    exit(-1);
  }

  addr.nl_family = AF_NETLINK;
  addr.nl_pid = getpid();
  addr.nl_groups = RTM_NEWADDR;

  if (bind(nl_socket, (struct sockaddr *)&addr, sizeof(addr)) < 0){
    perror("bind Netlink");
    return -1;
  }

  return nl_socket;
}


static int read_event(int sock)
{
  struct sockaddr_nl nladdr;
  struct msghdr msg;
  struct iovec iov[2];
  struct nlmsghdr nlh;
  char buffer[65536];
  int ret;

  iov[0].iov_base = (void *)&nlh;
  iov[0].iov_len = sizeof(nlh);
  iov[1].iov_base = (void *)buffer;
  iov[1].iov_len = sizeof(buffer);
```

```
  msg.msg_name = (void *)&(nladdr);
  msg.msg_namelen = sizeof(nladdr);
  msg.msg_iov = iov;
  msg.msg_iovlen = sizeof(iov)/sizeof(iov[0]);

  if ((ret = recvmsg(sock, &msg, 0)) < 0){
    perror("recvmsg Netlink");
    exit(-1);
  }

  return (nlh.nlmsg_type == RTM_NEWADDR);
}

void *thread_mobility(void)
{
  int nl_socket = 0;
  int count = 0;
  ll_head *ll;
  ll_node *lpt;
  NID_msg nmsg;
  struct timespec tv;

  nl_socket = open_netlink();

  while (1){
    if (read_event(nl_socket)){
      count++;

      if (count >= 2){
memset(&tv, 0, sizeof(tv));
tv.tv_nsec = 1000000;
nanosleep(&tv, NULL);

memset(&nmsg, 0, sizeof(nmsg));
nmsg.nid = nmsg.srcNID = getNID();

ll = getAllConnections();
lpt = ll->next;

if (get_mobility() == INTRADOMAIN){
  nmsg.type = REDIRECT;
  while (lpt){
    nmsg.destNID = lpt->nid;
    inet_pton(AF_INET, get_locator(getPrimaryIface()), &nmsg.ip);
```

```
    write(sp_mo2ph[1], &nmsg, sizeof(nmsg));
    lpt = lpt->next;
  }
}
updateCMconf();
count = 0;
rvs_update();
free(ll);
      }
    }
  }
}
```

Apêndice B

Diagramas de sequência da arquitetura

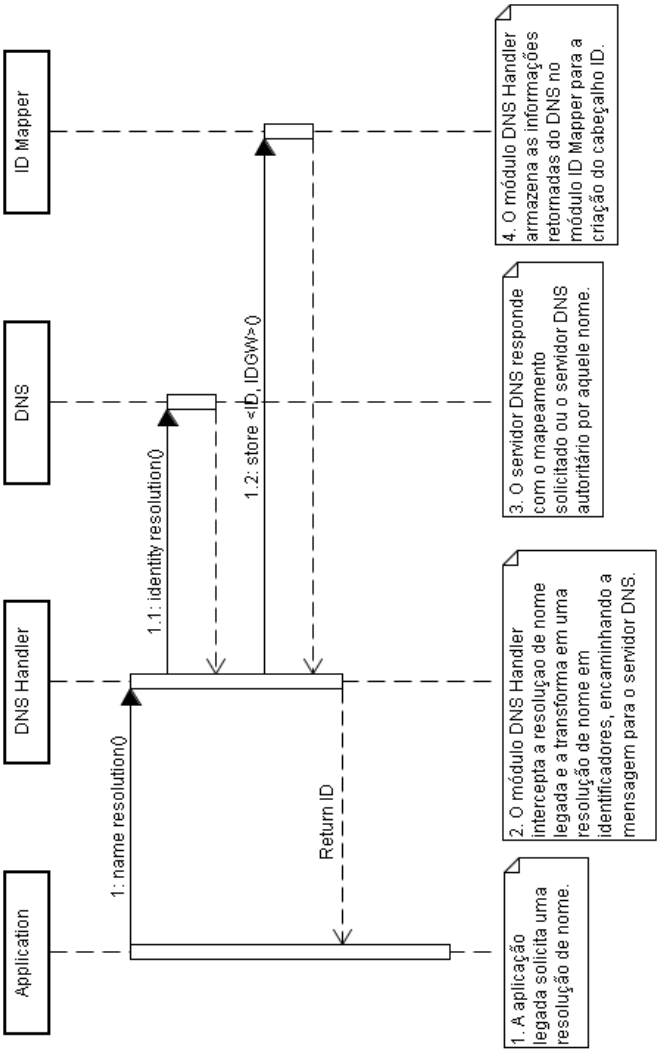

Fig. B.1: Diagrama de sequência da resolução do nome.

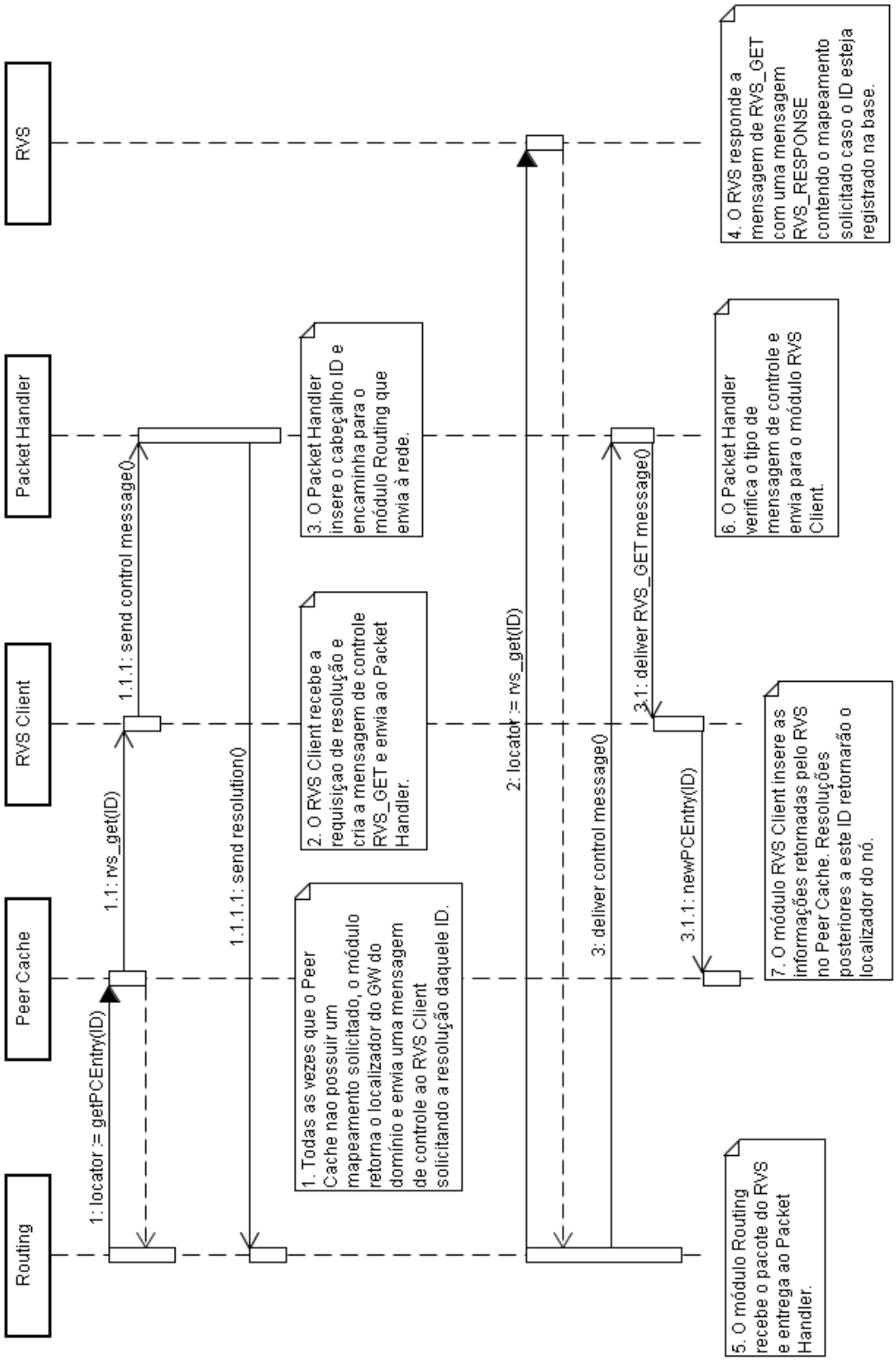

Fig. B.2: Diagrama de sequência da resolução do identificador.

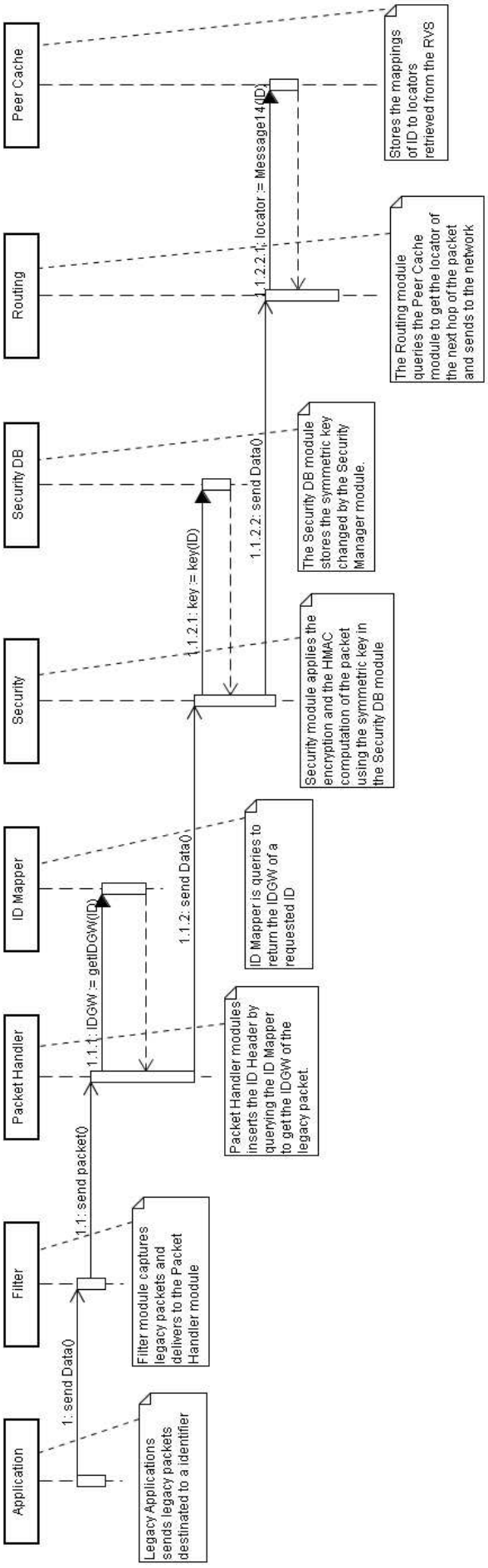

Fig. B.3: Diagrama de sequência do envio de dados.

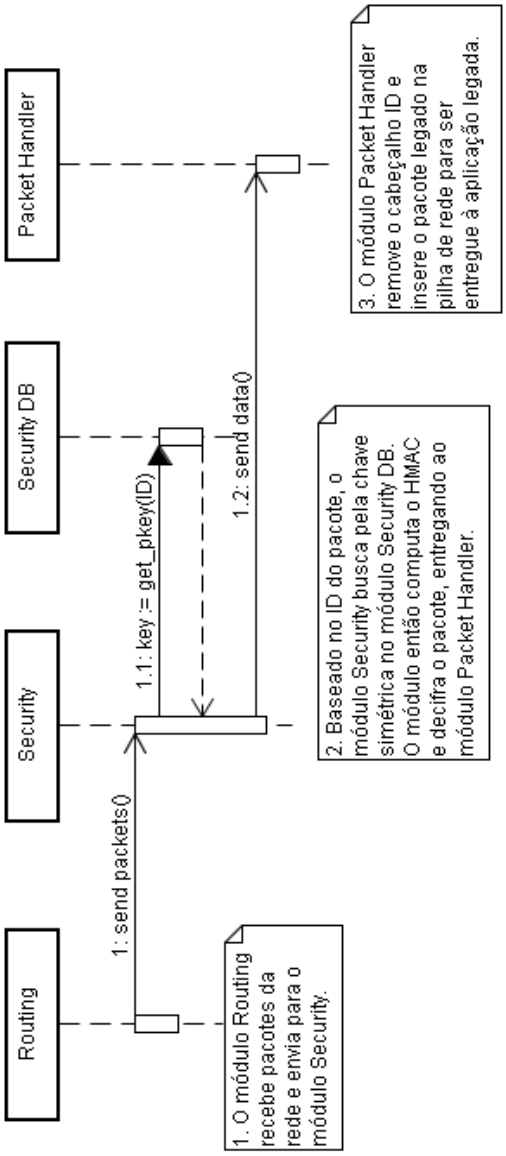

Fig. B.4: Diagrama de sequência da recepção de dados.

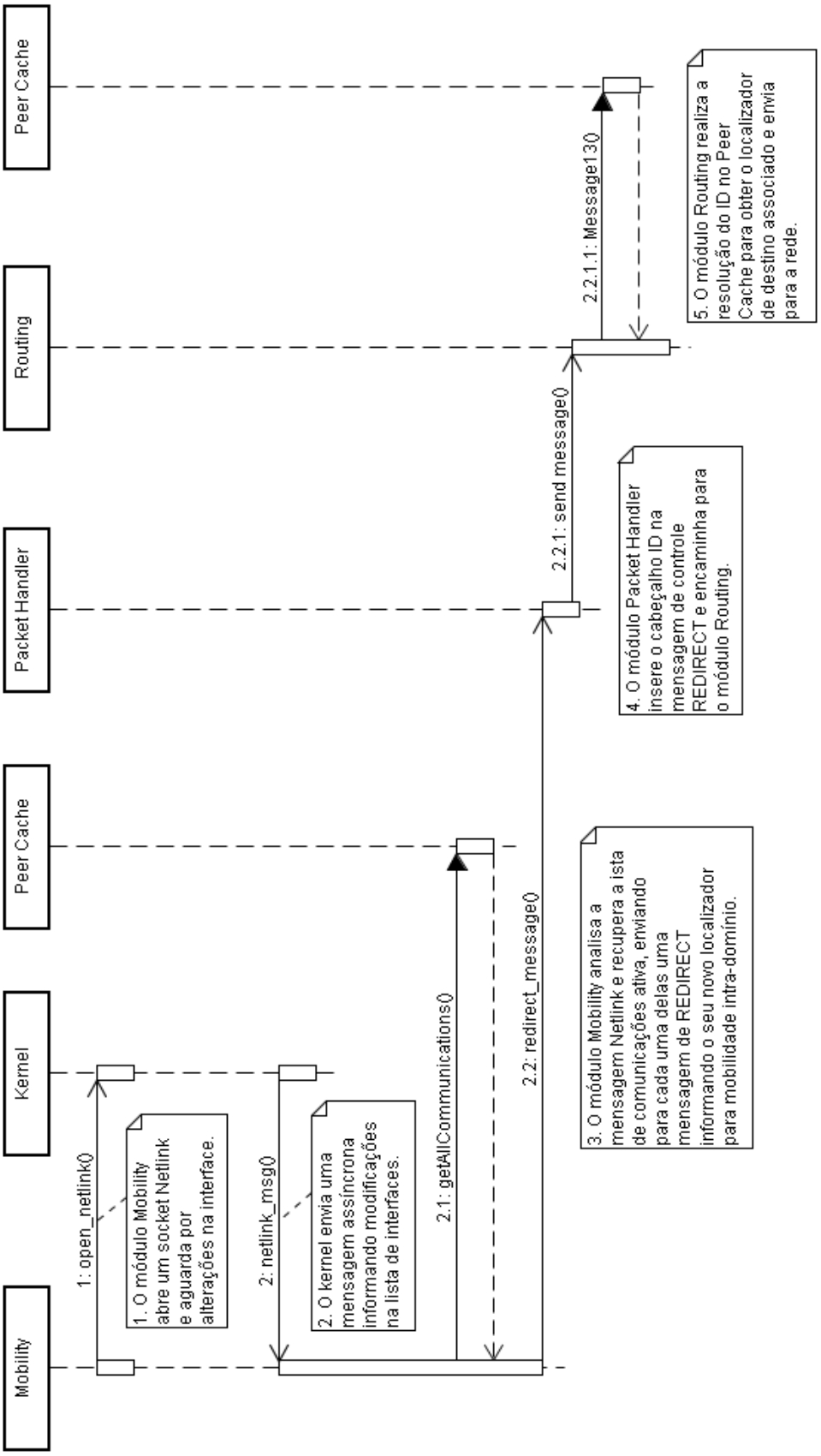

Fig. B.5: Diagrama de sequência da mensagem de REDIRECT.