UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DANILO EGÊA GONDOLFO

Implementação de uma Rede Endereçada por Interesses em Nível de Kernel

Maringá
2017

DANILO EGÊA GONDOLFO

Implementação de uma Rede Endereçada por Interesses em Nível de Kernel

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Nardênio Almeida Martins

Maringá
2017

Dados Internacionais de Catalogação na Publicação (CIP)
(Biblioteca Central - UEM, Maringá, PR, Brasil)

Gondolfo, Danilo Egêa
G637i Implementação de uma rede endereçada por interesses em nível de kernel / Danilo Egêa Gondolfo. -- Maringá, 2017.
80 f. : il. color., figs., tabs.

Orientador: Prof. Dr. Nardênio Almeida Martins.
Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2017.

1. Rede Ad Hoc Centrada em Interesses (Radnet). 2. Protocolo REPA. 3. FreeBSD. 4. Redes Ad Hoc. 5. Roteamento em redes Ad Hoc. I. Martins, Nardênio Almeida, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 23.ed.004.6
GVS-003714

FOLHA DE APROVAÇÃO

DANILO EGÊA GONDOLFO

Implementação de uma rede endereçada por interesses em nível de kernel

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Nardênio Almeida Martins
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Luciana Andréia Fondazzi Martimiano
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Rodrigo Campiolo
Universidade Tecnológica Federal do Paraná – DACOM/UTFPR-CM

Aprovada em: 26 de janeiro de 2017.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

DEDICATÓRIA

Aos meus pais...

AGRADECIMENTOS

Agradeço primeiramente à minha família pelo apoio, sem eles eu não teria chegado até aqui.

Agradeço à minha esposa Natália pela paciência (às vezes nem tanta) que teve nos momentos bons e ruins que passei durante o tempo em que me dediquei a este trabalho.

Agradeço ao professor Anderson Faustino pelo apoio e orientação no início do trabalho.

Agradeço imensamente aos professores Luciana A. F. Martimiano e Nardênio Almeida, pelo apoio que me foi dado nas fases finais do mestrado, sem os quais este trabalho, literalmente, não poderia ter sido concluído.

Agradeço ao meu chefe Samuel Antônio e aos meus colegas de trabalho pela paciência quando tive que sair mais cedo ou mesmo faltar ao trabalho para escrever ou ir a reuniões.

Agradeço a todos os professores do Mestrado em Ciência da Computação pelo conhecimento que me foi passado e em especial à Inês, funcionária da secretaria, por todos os momentos em que me ajudou de várias formas.

Por fim, agradeço a todos os meus amigos que aguentaram as minhas choradeiras nos momentos em que quis jogar tudo para o alto e me mudar para a praia.

Implementação de uma Rede Endereçada por Interesses em Nível de Kernel

RESUMO

Redes *Ad Hoc* podem ser empregadas em uma grande quantidade de situações, como por exemplo, em ambientes onde uma infraestrutura de rede é precária ou mesmo inexistente. Neste cenário, uma rede *Ad Hoc* pode ser criada entre dispositivos de comunicação pessoais, como *smartphones*, para que os usuários possam interagir e trocar informações. A Radnet (Rede *Ad Hoc* Centrada em Interesses) é uma rede *Ad hoc* no qual as mensagens são roteadas entre os dispositivos com base em características e interesses dos usuários. Este trabalho tem como principais objetivos propor e implementar a Radnet em nível de *kernel* do sistema operacional, sendo uma versão simplificada com foco na redução do consumo de recursos computacionais e com suporte à criptografia durante a troca de mensagens. Duas versões do protocolo REPA foram implementadas no *kernel* do sistema operacional FreeBSD e são utilizadas por meio da API de *sockets*. A redução do consumo de recursos se dá a partir de um cabeçalho simplificado no qual os interesses dos usuários são enviados em formato numérico ao invés de cadeia de caracteres. A implementação em nível de *kernel* também oferece um menor *overhead* durante o processamento de mensagens, o que pode significar uma redução no consumo de energia em dispositivos embarcados. Os experimentos mostraram que com o novo cabeçalho foi possível: i) reduzir consideravelmente a utilização de CPU, sendo obtidos ganhos de até 42% em relação à quantidade de instruções executadas por segundo; ii) aumentar a taxa de processamento de mensagens por minuto em aproximadamente 10%; e iii) reduzir o tempo de processamento de cada mensagem em até 22%.

Palavras-chave: Radnet. REPA. FreeBSD. Redes *Ad Hoc*. Roteamento.

ABSTRACT

Ad Hoc networks can be used in a lot of situations, for example in environments where there is a precarious network infrastructure or even an inexistent infrastructure. In this scenario, an Ad Hoc network can be created amongst personal communication devices, as smartphones, so that users can interact and share information. The Radnet (Interest-centric Ad Hoc Network) is an Ad hoc network in which the messages are routed between devices based on characteristics and interests of the users. This work has as main objectives the propose and the implementation of the Radnet in the operating system's kernel level, with a simplified version with focus on the computational resources reduction and encryption support during the messages exchange. Two versions of the REPA protocol have been implemented in the core of the FreeBSD operating system and are used through the sockets API. The resources comsumption reduction is based on a simplified header version in which the interests are sent in a numeric format instead of a string. The kernel level implementation also enable messages processing with less overhead, which could drive to less energy consumption in embedded systems. The experiments showed that with the new header is possible: i)to reduce the CPU consumption, obtaining gains up to 42% on the amount of instructions performed per second; ii) to increase the message processing rate per minute in about 10%; and iii) to reduce the per message processing time in about 22%.

Keywords: Radnet. REPA. FreeBSD. Ad Hoc Networks. Routing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AEAD: *Authenticated Encryption with Associated Data*
AES: *Advanced Encryption Standard*
AODV: *Ad hoc On-Demand Distance Vector*
API: *Application Program Interface*
ASCII: *American Standard Code for Information Interchange*
BSD: *Berkeley Software Distributions*
CBC: *Cipher Block Chaining*
CFB: *Cipher Feedback*
CPU: *Central Processing Unit*
ECB: *Electronic Codebook*
GCM: *Galois/Counter Mode*
GPS: *Global Position System*
I/O: *Input/Output*
IP: *Internet Protocol*
IV: *Initialization Vector*
KDC: *Key Distribution Center*
MANETS: *Mobile Ad Hoc Networks*
MAC: *Media Access Control*
mbuf: *Memory Buffer*
MOI: *Modelo Orientado a Interesses*
PA: Prefixo Ativo
PCB: *Protocol Control Block*
PMC: *Performance Monitoring Counters*
Radnet: *InteRest-Centric Mobile Ad Hoc Network*
Radnet-VE: *Interest-Centric Mobile Ad Hoc Network for Vehicular Environments*
REPA: Rede Endereçada Por Interesse *Ad Hoc*
REPI: Rede Endereçada Por Interesse
REPI-A: Rede Endereçada Por Interesse *Ad Hoc*
RSA: *Rivest-Shamir-Adleman*
RVEP: *RAdNet-VE Protocol*
SAMCRA: Sistema de Automação, Monitoração e Configuração de Redes *Ad Hoc*
SCTP: *Stream Control Transmission Protocol*
TCP: *Transmission Control Protocol*
TTL: *Time To Live*

UDP: *User Datagram Protocol*
VANET: *Vehicular Ad hoc Network*
XOR: *eXclusive OR*

SUMÁRIO

1 Introdução

Redes *Ad Hoc* podem ser utilizadas em uma grande quantidade de situações, por exemplo, em ambientes onde uma infraestrutura de rede é precária ou mesmo inexistente. Neste cenário, uma rede *Ad Hoc* pode ser criada entre dispositivos de comunicação pessoais, como *smartphones*, para que os usuários possam interagir e trocar informações. *InteRest-Centric Mobile Ad Hoc Network* (Radnet) é um modelo de rede utilizado sobre redes *Ad Hoc* no qual as mensagens são entregues e roteadas por meio de informações sobre as características e interesses do usuário (Dutra et al., 2010). As características podem ser, por exemplo, baseadas em perfis como: "cor do cabelo", "altura" e "sexo"; enquanto interesses são termos (cadeias de caracteres) como: "compras" e "viagem". Por meio das características, as mensagens são roteadas pela rede e por meio dos interesses elas são entregues aos usuários. O par características-interesses é chamado de Prefixo Ativo (PA) e o protocolo implementado neste trabalho, responsável pelo envio de mensagens na Radnet, é chamado neste trabalho de Rede Endereçada Por Interesses *Ad Hoc* (REPA).

Atualmente, é possível utilizar a Radnet de, pelo menos, duas maneiras: por meio de uma *Application Programming Interface* (API) e uma aplicação servidora fornecidas pelos desenvolvedores, em que toda a interação com a rede é feita por meio de um programa em execução no espaço de usuário do sistema, e por meio da API de *sockets* do sistema operacional FreeBSD, em que a implementação foi feita em nível de *kernel* (Gondolfo, 2014).

O principal problema da implementação em espaço de usuário é o fato de que todas as mensagens deverão ser movidas do *kernel* para a aplicação para serem processadas. Essa operação implica em consumo extra de recursos computacionais. Além disso, o código da implementação não está disponível para que outros pesquisadores possam colaborar

com o trabalho. A implementação em nível de *kernel* permite o uso da rede por meio da API de *sockets* do sistema operacional, além de todo o processamento das mensagens ser feito no *kernel*, não sendo necessário mover cada mensagem para uma aplicação para ser processada. O principal problema dessa implementação é não possuir compatibilidade com a versão original do protocolo, tornando impossível a interoperabilidade. Em ambas as implementações, os interesses são enviados pela rede em formato de texto plano, desse modo, para cada mensagem recebida o sistema precisa realizar o processamento de uma cadeia de caracteres que pode ser relativamente grande, acarretando em um longo tempo de computação.

Em algumas situações pode ser necessário o sigilo durante o repasse de mensagens com um interesse específico. Aplicações da Radnet nos quais o conteúdo da mensagem contém informações sensíveis não devem trafegar em formato de texto plano. Em certas ocasiões nem mesmo o interesse deve ser visível para todos os participantes da rede. Nesse contexto, a implementação proposta neste trabalho permite a criptografia transparente de tráfego em nível de *kernel*. Uma vez inicializado o suporte à criptografia, toda mensagem enviada pelo *socket* em questão é automaticamente codificada. Além disso, o suporte ao envio de interesses no formato numérico oferece o potencial de esconder o interesse das mensagens de determinados participantes da rede.

Neste contexto, os principais objetivos deste trabalho são implementar e avaliar a Radnet com uma versão simplificada do protocolo REPA em nível de *kernel* do sistema operacional e com suporte à criptografia de tráfego, visando a redução no consumo de recursos computacionais e segurança.

Para cumprir estes objetivos, foram definidos os seguintes objetivos específicos:

- Implementar uma versão do protocolo compatível com a versão disponibilizada pelos autores da Radnet para permitir a interoperabilidade entre as duas implementações;
- Implementar uma versão simplificada do protocolo REPA focada na redução do consumo de recursos computacionais;
- Implementar um mecanismo de criptografia transparente de tráfego de mensagens;
- Avaliar os ganhos de desempenho do protocolo com cabeçalho simplificado quando comparado com o cabeçalho original.

Dessa forma, as principais contribuições deste trabalho são a implementação de código aberto do protocolo compatível com a Radnet em nível de *kernel*, uma versão simplificada

do protocolo REPA focada em desempenho, um mecanismo de criptografia de mensagens e um conjunto de ferramentas para o monitoramento, configuração e utilização da rede.

Este trabalho se justifica pela inexistência das funcionalidades supracitadas. A implementação em nível de *kernel* oferece um menor *overhead* durante o roteamento de mensagens e facilita o desenvolvimento de aplicações sobre a Radnet, uma vez que o desenvolvedor poderá utilizar a API padrão de *sockets* em seus programas. A disponibilidade do código colabora para o avanço das pesquisas na área e a licença de *software* utilizada permite o uso da Radnet em sistemas comerciais sem a necessidade de disponibilização do código do produto.

Este trabalho está organizado da seguinte forma. O Capítulo 2 introduz o assunto sobre redes *Ad Hoc* móveis e o roteamento de mensagens nesse tipo de rede. É abordado sobre o modelo de redes orientadas a interesses e como ele se propõe a resolver o problema do roteamento, a organização do subsistema de rede do sistema operacional FreeBSD, as tecnologias referentes à criptografia empregadas no trabalho e, por fim, é apresentada uma revisão bibliográfica sobre o tema de Redes Endereçadas por Interesses. O Capítulo 3 apresenta as contribuições do trabalho e mostra como a Radnet foi implementada no FreeBSD. O Capítulo 4 mostra como a Radnet pode ser utilizada por meio da API de *sockets* do sistema operacional e monitorada. É apresentada a ferramenta *radnet_tool*, criada para capturar tráfego na Radnet, e como a rede pode ser configurada e ter suas estatísticas coletadas. O Capítulo 5 apresenta uma análise dos resultados. O Capítulo 6 apresenta as conclusões e as sugestões de trabalhos futuros.

2

Referencial teórico e trabalhos relacionados

Este Capítulo descreve as redes *Ad Hoc* móveis e o problema do roteamento nessas redes. São descritos, ainda, a Radnet, como os dispositivos e usuários são endereçados nela e como as mensagens são roteadas até eles. Também são descritos o algoritmo responsável por tomar as decisões de repasse de mensagens utilizado pelo protocolo REPA, a organização do subsistema de rede do sistema operacional FreeBSD. Uma descrição dos conceitos básicos de criptografia também é realizada e, por fim, alguns trabalhos relacionados são apresentados.

2.1 Redes *Ad hoc* Móveis

Redes *Ad hoc* móveis (*Mobile Ad Hoc Networks*, ou MANETS) são redes criadas sem uma infraestrutura previamente configurada. Nessas redes os dispositivos enviam mensagens diretamente entre si, sem a necessidade de um ponto de acesso central. As MANETS têm como característica a capacidade de auto configuração. Isso significa que mesmo que os nós mudem suas localizações geográficas, saiam da rede ou novos nós entrem na rede, ela é capaz de localizar e rotear mensagens até eles. Outra característica importante é a colaboração. Quando um nó não está acessível diretamente, por exemplo, por seu sinal estar fora de alcance, outros nós podem colaborar servindo como pontos intermediários por onde as mensagens irão passar até chegar ao destino (Conti e Giordano, 2007).

As MANETS possuem aplicações em áreas como redes hospitalares, segurança pública, sensoreamento, redes empresariais (AKYILDIZ et al., 2005), redes veiculares (Toor et al.,

2008) e compartilhamento de conteúdo (Krifa et al., 2009). A eficiência do roteamento de mensagens em MANETS é de extrema importância para que se tenha alto *throughput* e baixa perda de pacotes.

2.1.1 Roteamento em redes *Ad hoc* Móveis

O roteamento nas MANETS é realizado por meio de saltos entre os nós existentes na rede. Os objetivos básicos dos protocolos de roteamento se resumem em: maximizar o *throughput*, minimizar a perda de pacotes e controlar o *overhead* e o uso de energia (Boukerche et al., 2011).

Ainda segundo Boukerche et al. (2011), os protocolos de roteamento destas redes podem ser divididos nas seguintes classes:

- Iniciada pela origem (reativo ou sob-demanda). Esta classe de protocolos cria uma rota entre origem e destino apenas no momento do envio de mensagens. A Radnet pertence a essa classe de protocolos;
- Dirigida por tabela (proativo). Os protocolos mantêm uma tabela de roteamento atualizada constantemente. Isso é feito com base no envio de mensagens de controle para todos os nós da rede;
- Híbrida. Os protocolos dessa classe introduzem características dos protocolos de roteamento sob demanda e dirigidas por tabela de roteamento;
- Geográfica. Os protocolos dessa classe assumem que os nós da rede conhecem as localizações geográficas dos outros nós, utilizando, por exemplo, um GPS (*Global Positioning System).* Assim, o roteamento é feito com base nas localizações geográficas dos nós;
- *Multipath.* Os protocolos de roteamento de múltiplos caminhos mantêm mais de uma rota entre os nós de origem e destino. A vantagem principal é que com múltiplos caminhos o meio de transmissão pode ser utilizado mais eficientemente, por exemplo, evitando-se caminhos congestionados;
- Hierárquica. Os protocolos dessa classe criam hierarquias de nós nas quais nós em níveis mais altos podem fornecer serviços para os demais. Esta classe se propõe a tratar problemas de escalabilidade com o crescimento da rede nos quais outros modelos poderiam ter problemas com tamanhos de tabelas de rotas e *overhead* com pacotes de controle;

- *Multicast*. *Multicast* é o envio de mensagens para grupos de nós em uma rede. Os protocolos dessa classe criam grupos na rede de forma que as mensagens são roteadas por nós pertencentes ao grupo;

- *Multicast* Geográfico. São protocolos de roteamento *multicast* que levam em consideração a localização geográfica de um grupo de nós, por exemplo, por meio de GPS;

- Com foco em consumo de energia. Estes protocolos tomam as decisões de roteamento com base nos recursos de energia dos nós. Em geral, o objetivo é minimizar o consumo de energia da rede como um todo.

2.2 Radnet

Radnet é uma rede orientada a interesses e características do usuário proposta para ser utilizada sobre redes *Ad Hoc* sem fio móveis (Dutra, 2012). A Radnet possui como principais características: i) a colaboração entre os nós da rede, em que as mensagens são entregues por meio de saltos entre os nós até que o destinatário seja alcançado; ii) o roteamento baseado nos perfis dos usuários; e iii) a entrega de mensagens a usuários com os mesmos interesses na rede. As ideias empregadas na Radnet se aproximam das Redes Centradas em Conteúdo como apresentado por Jacobson et al. (2009) e Meisel et al. (2010).

Segundo Dutra (2012), a Radnet se enquadra no modelo Publicador/Subscritor, um modelo assíncrono no qual um nó Publicador envia mensagens destinadas a um determinado interesse e todos os Subscritores daquele interesse as recebem, embora, devido a falhas de comunicação, uma mensagem possa não ser entregue.

2.2.1 Endereçamento de dispositivos e usuários na Radnet

O endereçamento dos dispositivos da rede e dos usuários é realizado por meio do Prefixo Ativo (PA) (Dutra et al., 2012). O PA é dividido em dois campos, o Prefixo (também chamado de "Características") e o Interesse. O Prefixo é um campo de 32 bits logicamente subdividido em campos menores. Cada um desses subcampos representa uma característica do usuário. Na implementação utilizada neste trabalho, o Prefixo foi dividido em oito subcampos de 4 bits cada.

O Interesse é o campo que armazena e representa um dos interesses da aplicação. Na implementação da versão original do protocolo, o campo Interesse é uma cadeia de

caracteres que pode ter até 255 caracteres e se localiza imediatamente após o cabeçalho de tamanho fixo da mensagem. Na implementação simplificada também utilizada neste trabalho, o campo Interesse tem tamanho fixo de 4 bytes e é enviado em formato de *hash* como um inteiro.

2.2.2 Características do usuário

Os 32 bits referentes às características do usuário são divididos em oito campos de 4 bits. Durante a modelagem da rede, a característica que cada campo irá representar e seus valores devem ser previamente definidos. Um exemplo de característica é reservar o primeiro campo de 4 bits para "cor do cabelo", nos quais cada valor representado por esses 4 bits dizem qual a cor. Por exemplo, 0000 significa cabelo de cor preta, 0001 cabelo de cor vermelha, e assim por diante.

O Prefixo é utilizado da mesma maneira que um endereço do tipo *Internet Protocol* (IP) é utilizado para identificar computadores em uma rede. A diferença é que na implementação proposta neste trabalho, o prefixo está atrelado à cada aplicação e não ao equipamento e, ao contrário do IP, vários usuários podem ter o Prefixo exatamente iguais. O objetivo do uso do prefixo por aplicação se deve ao fato de que os sistemas operacionais podem ser utilizado por mais de um usuário. Dessa forma, cada usuário poderá configurar o seu próprio prefixo ao invés de utilizar um compartilhado por todo o sistema.

2.2.3 Interesses

As mensagens são entregues aos usuários com base nos seus interesses. Da mesma forma que as características são definidas durante a modelagem da rede, os interesses também devem ser. O campo de interesse é composto de uma cadeia de caracteres de tamanho variável limitada a 255 caracteres ou um campo de 4 bytes que armazena um valor inteiro. A implementação deste trabalho utiliza os dois formatos. Quando enviada, uma mensagem é destinada a apenas um interesse, assim, esse campo sempre conterá apenas uma cadeia. Para exemplificar a utilização de interesses considere a seguinte situação: o restaurante universitário publica constantemente o cardápio do dia por meio de uma Radnet associando o interesse "cardapioRU" às mensagens enviadas. As pessoas que queiram ver o cardápio do dia poderão executar uma aplicação cliente e registrar o mesmo interesse ("cardapioRU") à um *socket*. Assim, a aplicação começará a receber as mensagens enviadas pelo restaurante.

O Interesse é utilizado da mesma maneira que uma porta do tipo *User Datagram Protocol* (UDP) ou *Transmission Control Protocol* (TCP) é utilizada para identificar uma aplicação na rede.

2.3 Protocolo REPA

O protocolo REPA (sigla utilizada neste trabalho e derivada de REPI-A (Granja, 2010) - Rede Endereçada Por Interesses *Ad-hoc*) é o responsável por encaminhar as mensagens entre os dispositivos da rede. Ele decide se a mensagem deve ser encaminhada ou não com base no casamento dos campos de característica enviados com a mensagem e as características dos usuários do dispositivo que está recebendo aquela mensagem. Três políticas podem ser utilizadas no casamento dos campos: uma política restritiva, na qual todos os campos devem coincidir para o encaminhamento ser realizado; uma política relaxada, na qual pelo menos um campo deve coincidir; e uma política que encaminha a mensagem sem levar em consideração as características. Na implementação descrita neste trabalho, essas políticas podem ser selecionadas a qualquer momento por meio de *sysctl's* do sistema operacional.

São implementas duas versões do protocolo REPA. A primeira versão, chamada de REPA_UFRJ, é compatível com a implementação original do protocolo. A segunda versão, chamada de REPA_UEM, foi simplificada para permitir o envio de interesses em formato numérico ao invés de usar cadeias de caracteres. O objetivo dessa modificação foi a redução do custo do processamento das mensagens enviadas pela rede. Ambas as versões possuem as mesmas características no funcionamento do roteamento. As modificações foram introduzidas no cabeçalho de cada protocolo.

A Figura - 2.1 apresenta a estrutura das mensagens do protocolo REPA_UFRJ utilizada na implementação deste trabalho. Os campos são utilizados como se segue:

- Versão. Campo com tamanho de 4 bits que especifica a versão do protocolo;
- H. Campo com tamanho de 1 bit que especifica se o interesse da mensagem será ocultado. Este campo pode assumir os valores "0" e "1";
- C. Campo com tamanho de 1 bit que indica se o interesse e os dados estão criptografados. Este campo pode assumir os valores "0" e "1";
- O. Campo com tamanho de 2 bits reservado para uso futuro;
- Tempo de vida. Campo de 8 bits que indica a quantidade de saltos permitidos até que a mensagem seja descartada. O valor inicial desse campo é 64;

- Tamanho do cabeçalho. Campo com tamanho de 16 bits que indica o tamanho em bytes do cabeçalho da mensagem;
- Número de sequência. Campo com tamanho de 32 bits que indica o número de sequência da mensagem;
- Prefixo de destino. Campo com tamanho de 32 bits que contém o prefixo de destino da mensagem;
- Prefixo de origem. Campo com tamanho de 32 bits que contém o prefixo de origem da mensagem;
- *Timestamp*. Campo com tamanho de 64 bits contendo o *timestamp* em microsegundos do momento em que a mensagem foi enviada;
- Interesse. Campo com tamanho variável, limitado a 255 bytes, contendo o interesse da mensagem.

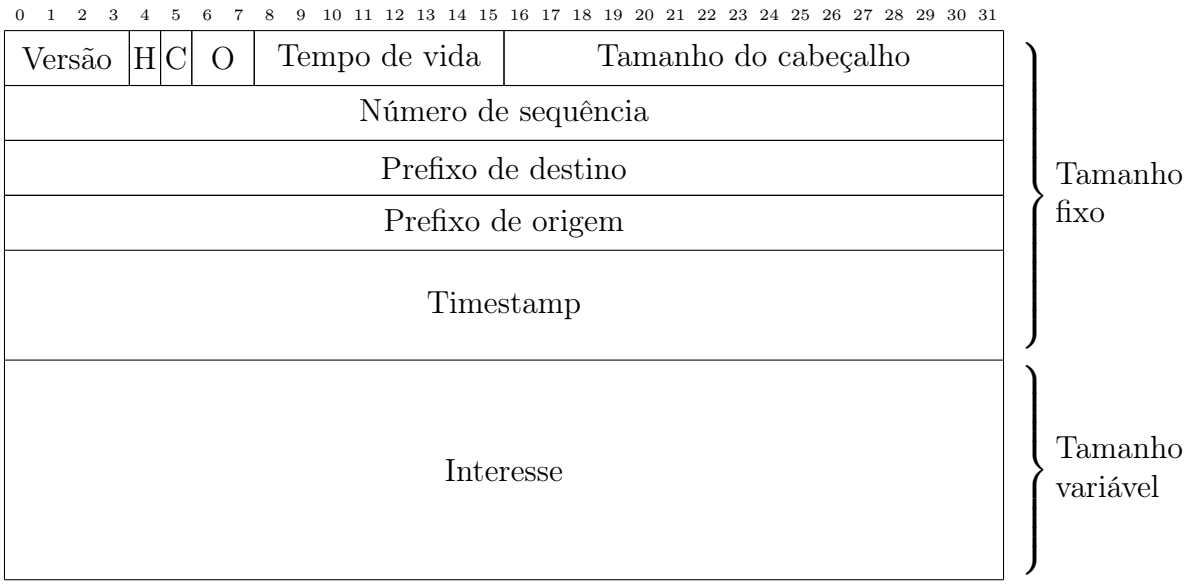


Figura 2.1: Cabeçalho utilizado pelo protocolo REPA_UFRJ

No cabeçalho utilizado pelo REPA_UEM alguns campos foram removidos por questões de simplificação. Nesta versão apenas um valor de Prefixo é utilizado. O Interesse foi transformado em um número inteiro de 4 bytes, armazenado e enviado em formato de *hash*. Essa modificação tem como objetivo acelerar o processo de comparação de interesses nos nós da rede. O *timestamp* foi modificado para ter precisão em nanosegundos. O

campo Tamanho do Cabeçalho foi removido devido ao cabeçalho do REPA_UEM possuir tamanho fixo e o campo Tamanho Total foi inserido para possibilitar a verificação do final da mensagem em enlaces que adicionam um complemento no final do quadro quando o mesmo não possui o tamanho mínimo de quadro requerido pelo enlace. A Figura - 2.2 apresenta o cabeçalho utilizado pela versão REPA_UEM do protocolo.

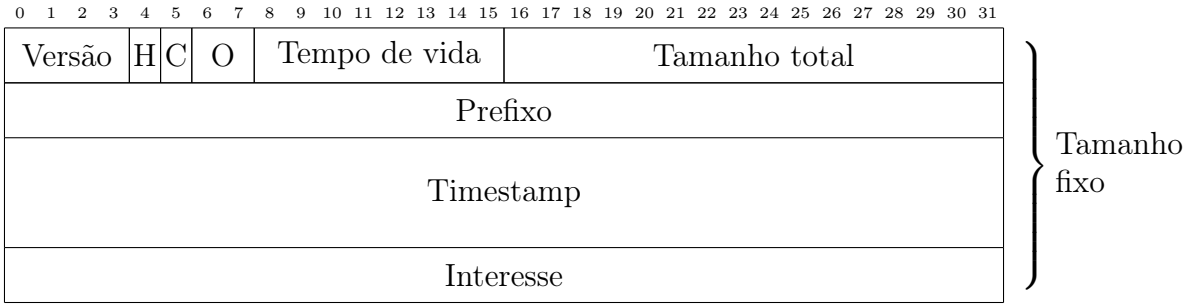


Figura 2.2: Cabeçalho utilizado pelo protocolo REPA_UEM

2.3.1 Roteamento na Radnet

O roteamento na Radnet é realizado por meio da colaboração entre os nós. As mensagens são enviadas por meio de saltos entre os nós intermediários até que alcance o destino.

Um nó decide se a mensagem será repassada para os outros com base no casamento dos campos do prefixo. O casamento pode ser parcial, no qual a mensagem é repassada se pelo menos um campo coincidir, ou completo, no qual a mensagem é repassada se todos os campos coincidirem. O Capítulo 4 discute como esse e outros parâmetros podem ser configurados.

Devido ao fato de o casamento dos prefixos depender de existência de características comuns entre o usuário da aplicação que enviou a mensagem e o usuário da aplicação que a recebeu, diz-se que o roteamento na Radnet é probabilístico. Caso o usuário de origem e destino não possuam características em comum, o nó que recebeu a mensagem não irá repassá-la de volta para a rede.

Quando um nó envia ou repassa uma mensagem para a rede, a mensagem é inserida em uma tabela *hash*, utilizando o campo *timestamp* como chave. Quando uma mensagem com o mesmo *timestamp* é recebida, há uma alta probabilidade de essa mensagem já ter passado pelo nó, então, ela é descartada. A quantidade de mensagens memorizadas é controlada por meio de uma fila circular com tamanho inicial de 100 elementos. O tamanho desta fila pode ser modificado como apresentado no Capítulo 4. Uma mensagem

também pode ser descartada quando o campo Tempo de vida da mensagem atinge o valor 0. Essas políticas evitam que uma mensagem trafegue indefinidamente pela rede.

O Algoritmo 1, baseado em Dutra (2012), apresenta a lógica utilizada para o repasse de mensagens para a rede.

```
Input: Mensagem
if Mensagem nunca passou pelo nó then
    Insere Mensagem na tabela;
end
if Ocorreu o casamento dos prefixos then
    Mensagem.TempoDeVida−−;
    if Mensagem.TempoDeVida != 0 then
        Envia Mensagem para todos os vizinhos;
    end
    else
        Descarta Mensagem;
    end
end
for Toda aplicação p no nó com o Interesse == Mensagem.Interesse do
    Envia Mensagem para a aplicação p;
end
```

Algorithm 1: Algoritmo responsável pelo repasse de mensagens utilizado no REPA_UEM e REPA_UFRJ

2.4 Organização da pilha de rede do sistema operacional FreeBSD

Esta Seção apresenta como a pilha de protocolos de rede é organizada no sistema operacional. São apresentados como os dispositivos de rede são representados dentro do *kernel* e as estruturas de dados básicas utilizadas para o armazenamento, envio e recebimento de pacotes de rede. Também são apresentadas a interface de *sockets* do sistema e como as chamadas de sistema estão relacionadas com as funções implementadas por essa camada.

2.4.1 Interfaces de rede

As interfaces dos dispositivos de rede são representadas no sistema como estruturas do tipo *ifnet*. Nessa estrutura, além de variáveis de controle e identificação dos dispositivos, também são encontrados ponteiros para funções que realizam tarefas como: enviar e receber pacotes e rotinas de inicialização dos dispositivos. Muitos desses parâmetros são configurados pelo *driver* dos dispositivos assim que eles são reconhecidos pelo sistema.

Após sua inicialização, um dispositivo de rede pode ter alguns parâmetros modificados pelos usuários por meio de comandos como o *ifconfig*. Isso é feito por meio da chamada de sistema *ioctl*, apresentada posteriormente neste capítulo, que é aplicada aos descritores de arquivo do tipo *socket*.

2.4.2 *Memory Buffers* - mbufs

Os *memory buffers*, ou *mbufs*, são as unidades de memória utilizadas para o armazenamento de pacotes recebidos e enviados pela rede ou mesmo no próprio sistema. A estrutura de um *mbuf* pode ser vista na Figura - 2.3. Um *mbuf* é composto de um cabeçalho (*m_hdr*), apresentado na Figura - 2.4, uma região para o armazenamento dos dados e outros campos que dependerão do seu tipo. A estrutura *mbuf* possui 256 bytes de tamanho na arquitetura Intel. Segundo Stevens e Wright (1995), são possíveis quatro tipos de *mbuf*, sendo que cada tipo dependerá das *flags* especificadas no campo *mh_flags* do cabeçalho *m_hdr*, a saber:

1. Se o campo *mh_flags* é igual a zero, o *mbuf* contém apenas dados. O local onde os dados foram alocados é apontado pelo campo *mh_data* e o tamanho do *buffer* é indicado em *mh_len*. Um *mbuf* que contém apenas dados possui 224 bytes para armazenamento (McKusick et al., 2014);

2. Se a *flag* M_PKTHDR estiver marcada no campo *mh_flags* significa que o *mbuf* possui um cabeçalho de pacote. Isso significa que o primeiro *mbuf* da cadeia descreve o pacote de dados com o cabeçalho do mesmo;

3. Se a *flag* M_EXT estiver marcada no campo *mh_flags* significa que o *mbuf* possui um *buffer* externo para o armazenamento dos dados. Na arquitetura Intel utilizada neste trabalho esse *buffer* possui um tamanho de 2048 bytes. Um *mbuf* que utiliza esse campo também é chamado de *cluster*;

4. Um *mbuf* pode conter ambas as *flags* M_PKTHDR e M_EXT, significando que ele possui um cabeçalho do pacote e uma área de armazenamento externa.

```
struct mbuf {
    struct m_hdr     m_hdr;
    union {
        struct {
            struct pkthdr    MH_pkthdr;   /* M_PKTHDR set */
            union {
                struct m_ext     MH_ext; /* M_EXT set */
                char             MH_databuf[MHLEN];
            } MH_dat;
        } MH;
        char     M_databuf[MLEN];      /* !M_PKTHDR, !M_EXT */
    } M_dat;
};
```

Figura 2.3: Estrutura do *mbuf*

```
struct m_hdr {
    struct mbuf *mh_next;     /* next buffer in chain */
    struct mbuf *mh_nextpkt; /* next chain in queue/record */
    caddr_t      mh_data;     /* location of data */
    int32_t      mh_len;      /* amount of data in this mbuf */
    uint32_t     mh_type:8,   /* type of data in this mbuf */
                 mh_flags:24;    /* flags; see below */
#if !defined(__LP64__)
    uint32_t     mh_pad;      /* pad for 64bit alignment */
#endif
};
```

Figura 2.4: Cabeçalho do *mbuf*

Um pacote pode ser armazenado em uma cadeia de *mbufs*, caso ele não caiba em apenas um. O encadeamento é realizado por meio do campo *mh_next* do cabeçalho.

2.4.3 Domínios e protocolos

O FreeBSD organiza os protocolos de rede em domínios. Um domínio de protocolos bem conhecido é o INET. Este domínio contém uma lista de todos os protocolos relacionados com o IPv4 no sistema, como: IP, TCP, UDP e o *Stream Control Transmission Protocol* (SCTP). Um domínio é representado por uma estrutura *domain* no *kernel* e possui um ponteiro para uma tabela conhecida como *Protocol Switch*, que contém uma lista de todos os protocolos pertencentes a um domínio.

Sempre que um usuário cria um *socket* fazendo referência a um domínio e um protocolo, o sistema faz uma busca na lista de domínios, e caso o encontre, localiza também a entrada da tabela *Protocol Switch* e nela busca pelo protocolo em questão. A Figura - 2.5 apresenta um exemplo da chamada de sistema *socket*. Nessa chamada são informados a família (ou domínio) de protocolos em questão, no caso PF_RADNET, e um protocolo pertencente à essa família, no caso RADNETPROTO_REPA_UEM.

```
int s = socket(PF_RADNET, SOCK_DGRAM, RADNETPROTO_REPA_UEM);
```

Figura 2.5: Exemplo de uso da chamada de sistema *socket*

2.4.4 *BSD Sockets*

A API de *sockets* é uma interface padronizada e genérica utilizada para o desenvolvimento de sistemas que necessitam enviar e receber informações de maneira distribuída. Esta interface foi criada no sistema operacional 4.2BSD com o objetivo de padronizar o modo como os desenvolvedores interagiam com protocolos de rede (McKusick et al., 2014) e atualmente é utilizada em muitos sistemas comerciais modernos.

Para o usuário, um *socket* é um descritor de arquivos comum. Em protocolos baseados em *stream* de dados, como o TCP, é possível utilizar funções como *read()* e *write()* em *sockets* do mesmo modo como são utilizadas em arquivos para enviar e receber dados, respectivamente. Em protocolos que não são orientados à conexão, como é o caso do protocolo REPA, as funções *sendto()* e *recvfrom()* devem ser utilizadas.

Essas funções são disponibilizadas por meio de chamadas de sistema. As chamadas de sistema referentes aos *sockets* invocam funções de uma interface com o protocolo. São essas funções que executam ações como enviar e receber dados por meio do protocolo.

2.4.5 Interface entre *sockets* e protocolos

A interface entre os *sockets* e os protocolos de rede é realizada por meio da estrutura *pr_usrreqs*. Essa estrutura relaciona as ações realizadas em espaço de usuário com funções implementadas no *kernel*. A Figura - 2.6 apresenta a estrutura utilizada pelo protocolo REPA. Cada elemento da estrutura é um ponteiro para uma função. As funções *repa_attach*, *repa_bind*, *repa_detach*, *repa_sosend* e *repa_soreceive* são invocadas quando as chamadas de sistema *socket*, *bind*, *close*, *sendto* e *recvfrom*, respectivamente, são invocadas pelos usuários. A função *repa_ioctl* é invocada quando a chamada de sistema *ioctl* é executada em um *socket* associado ao protocolo REPA_UEM.

```
struct pr_usrreqs repa_uem_usrreqs = {
        .pru_attach =           repa_uem_attach,
        .pru_bind =             repa_uem_bind,
        .pru_control =          repa_ioctl,
        .pru_detach =           repa_uem_detach,
        .pru_sosend =           repa_uem_sosend,
        .pru_soreceive =        repa_uem_soreceive,
};
```

Figura 2.6: Estrutura *pr_usrreqs* referente ao protocolo REPA_UEM

2.4.6 Interface entre protocolos e dispositivos de rede

As interfaces dos dispositivos de rede são implementadas no sistema operacional por meio da estrutura *ifnet*. Essa estrutura possui informações sobre o dispositivo, como nome, endereços de rede e ponteiros para funções implementadas no *driver* do dispositivo. Funções utilizadas para a inicialização do dispositivo, envio e recebimento de pacotes também são encontradas nessa estrutura.

Quando um pacote é recebido, um ponteiro para a estrutura referente à interface que recebeu o pacote é encontrado no cabeçalho *pkthdr* do *mbuf*. Assim, é possível saber por qual interface de rede o pacote chegou e, no caso do protocolo REPA, por qual interface de rede o pacote será reencaminhado.

2.4.7 *ioctls*

A chamada de sistema *ioctl* é uma interface comumente utilizada para o controle e comunicação com dispositivos. A *ioctl* é utiliza em contextos não relacionados com entrada e saída de dados, por exemplo, para ejetar um CD-ROM (Kong, 2012).

No contexto de redes, a chamada de sistema *ioctl* é utilizada em atividades como a atribuição de endereços às interfaces de rede, clonagem de interface virtuais e consulta ao estado dos dispositivos.

Este trabalho utiliza a chamada de sistema *ioctl* no processo de habilitar e desabilitar o suporte à Radnet em uma interface de rede.

2.5 Criptografia

Esta seção apresenta as tecnologias relacionadas à criptografia utilizadas neste trabalho.

O suporte à criptografia foi implementado em nível de *kernel* e é baseado na criptografia simétrica, em que a mesma chave é usada para codificar e decodificar a informação. Este trabalho tem foco no suporte ao padrão de criptografia simétrica *Advanced Encryption Standard* (AES) com suporte ao modo de cifra de bloco *(Cipher Block Chaining* (CBC). A API utilizada também possui suporte aos modos *Electronic Codebook* (ECB) e *Cipher Feedback* (CFB).

Como é apresentado no Capítulo 5, para fins de demonstração de como o suporte à criptografia pode ser utilizado na Radnet, um Centro de Distribuição de Chaves foi implementado com base na troca de chaves via criptografia assimétrica (ou de chave pública/privada).

2.5.1 Criptografia simétrica e assimétrica

Uma característica que os algoritmos de criptografia (reversíveis) possuem é o fato de dependerem de uma chave criptográfica. O algoritmo de codificação recebe como entrada um bloco de dados (também chamado de "mensagem clara") e a chave para produzir um bloco de dados criptografado (também chamado de "criptograma"). O algoritmo de decodificação, por sua vez, recebe o bloco de dados criptografado e uma chave para recuperar a mensagem clara. A chave utilizada para cada operação pode ser diferente dependendo do algoritmo utilizado.

Algoritmos simétricos são caracterizados por utilizarem a mesma chave para a operação de codificação e decodificação. Isso significa que todas as entidades envolvidas na troca de mensagens precisam ter conhecimento da chave. Por consequência, a distribuição da chave de maneira segura pode se tornar uma complicação.

Algoritmos assimétricos, também chamados de algoritmos de chave pública e privada, são caracterizados por utilizarem chaves diferentes para cada operação. Segundo Ferguson e Schneier (2003), nesse modo de criptografia é necessária a criação de um par de chaves, denominadas chave pública e chave privada, e tal método oferece uma solução para o problema da distribuição de chaves. A operação de codificação é realizada com a chave pública. A chave pública pode ser disponibilizada publicamente, como o nome sugere. Uma mensagem criptografada com a chave pública pode ser decodificada apenas com a chave privada, que deve ser mantida em segredo. A criptografia de chave pública foi apresentada pela primeira vez por Diffie e Hellman (1976).

Neste trabalho a criptografia assimétrica foi utilizada para a troca de chaves simétricas na Radnet. A biblioteca OpenSSL (OpenSSL, 2016) foi utilizada para esse propósito.

2.5.2 O padrão AES

O AES, definido em NIST (2001), é um padrão de criptografia simétrica que adota o algoritmo de cifras de bloco Rijndael (Daemen e Rijmen, 1999). O AES define que o algoritmo utilizado deve permitir o uso de chaves de 128, 192 e 256 bits e blocos de dados de 128 bits. Além do tamanho de bloco definido pelo AES, o Rijndael também permite o uso de blocos de 192 e 256 bits.

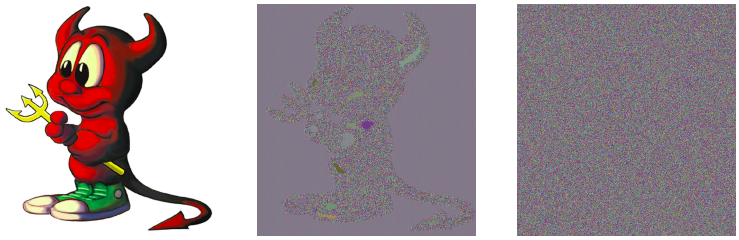
O AES foi adotado como padrão de criptografia simétrica pela indústria e atualmente é largamente utilizado em produtos corporativos.

2.5.3 Modos de cifra de bloco

O uso do algoritmo de criptografia por si só pode apresentar alguns problemas. A partir do princípio de que para cada bloco de entrada será produzido um bloco de mesmo tamanho como saída, caso blocos iguais sejam criptografados com a mesma chave um bloco criptografado igual será produzido (Ferguson e Schneier, 2003). Observando esse comportamento, um atacante poderá descobrir que informações repetidas estão sendo transmitidas. Conforme pode ser observado na Figura - 2.7, o uso do modo de cifra ECB produz um arquivo cifrado que permite a identificação de padrões. O uso do modo de cifra CBC evita esse vazamento de informações.

Para evitar esse comportamento no texto cifrado, pode-se usar modos de cifra de bloco em conjunto com o algoritmo de criptografia. Um modo de cifra aplica transformações nos blocos de dados de modo que a criptografia de blocos iguais gere como saída blocos diferentes. Esse efeito pode ser observado na Figura - 2.7. O modo de cifra ECB apenas codifica blocos individualmente e os armazena na mensagem de saída na mesma ordem em que foram codificados. O modo de cifra CBC executa uma operação XOR entre o bloco que será codificado e o bloco codificado anteriormente. O primeiro bloco é gerado a partir da operação XOR com um valor aleatório do mesmo tamanho do bloco, normalmente chamado de Vetor de Inicialização. O processo de criptografia pode ser observado na Figura - 2.8, onde B1, B2 e B3 são os blocos de texto plano que serão codificados e C1, C2 e C3 são os blocos de texto codificados. Após a utilização do Vetor de Inicialização no início do processo, os blocos cifrados são utilizados nas operações XOR com os próximos blocos de texto plano e só então o bloco resultante é codificado pelo algoritmo de criptografia.

Modos de cifra mais elaborados como o GCM (Galois/Counter Mode) (Dworkin, 2007) podem ser utilizados para garantir mais segurança no processo de criptografia. O modo de cifra GCM também provê a autenticação das mensagens cifradas. O modo de cifra utilizado neste trabalho é o CBC.



(a) (b) (c)

Figura 2.7: Uso dos modos de cifra (b) ECB e (c) CBC aplicados na imagem (a)[2]com AES de 256 bits

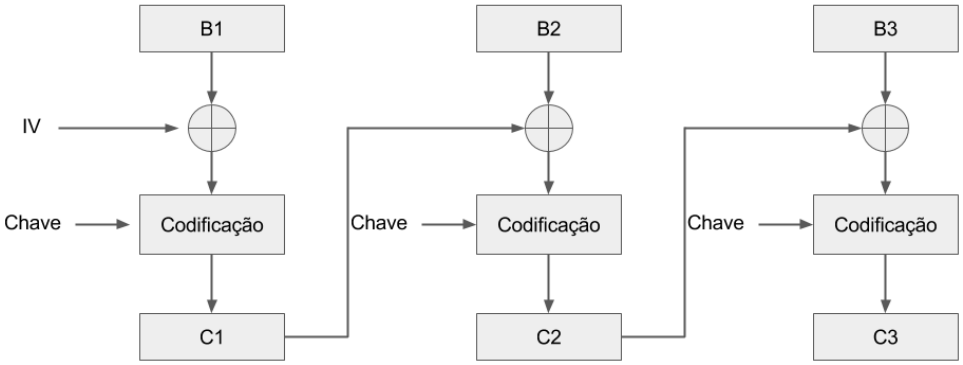


Figura 2.8: Processo de criptografia com o modo de cifra CBC

2.6 Trabalhos Relacionados

Esta Seção apresenta os trabalhos relacionados à proposta desta dissertação. São apresentados os trabalhos que introduziram o conceito de Prefixo Ativo e os protocolos utilizados sobre a Radnet.

2.6.1 Prefixo Ativo

Nos trabalhos desenvolvidos por Dutra et al. (2011) e Dutra et al. (2012) é introduzido o conceito de Prefixo Ativo como um conjunto de características dos nós da rede e interesses associados a uma aplicação. Os autores propõem que o Prefixo Ativo possua quatro características distintas: i) definição de endereçamento de nós da rede e interesses de aplicações de uma maneira distribuída; ii) os nós se comunicam pelo envio de mensagens utilizando seus Prefixos Ativos como cabeçalhos para, colaborativamente, apoiar o envio e endereçamento probabilístico; iii) os nós usam o casamento de Prefixos Ativos como

[2]BSD Daemon Copyright 1988 by Marshall Kirk McKusick. All Rights Reserved.

filtros para a comunicação entre camadas de rede; e iv) a rede de Prefixos Ativos pode ser adaptada para endereçar nós por meio do protocolo IP. Os autores sugerem que o endereçamento por IP pode ser feito inserindo-se os endereços de origem e destino nos campos de prefixo e encapsulando-se o pacote IP na mensagem a ser enviada.

A Figura - 2.9 apresenta como o cabeçalho das mensagens é organizado. O repasse das mensagens é realizado por meio do campo de Características do Prefixo Ativo. Esse campo é dividido em campos menores que representam as características do nó. A tomada de decisão para o repasse das mensagens é feita com base no casamento desses campos. O nó compara o valor de cada campo das suas características com os valores armazenados na mensagem recebida. Caso ocorra um casamento, o nó toma a decisão de repassar a mensagem para a rede. Dessa forma, a entrega colaborativa de mensagens é realizada.

No trabalho desenvolvido por Dutra (2012) é apresentado como o Prefixo Ativo pode ser aplicado em uma rede endereçada por interesses. A entrega da mensagem para a aplicação é feita com base no casamento do campo Interesse. Se houver alguma aplicação no nó cujo interesse registrado é o mesmo da mensagem recebida, o conteúdo da mensagem será enviado para a aplicação. Por meio da entrega colaborativa, a rede possibilita que cada nó encontre outros com os mesmos interesses.

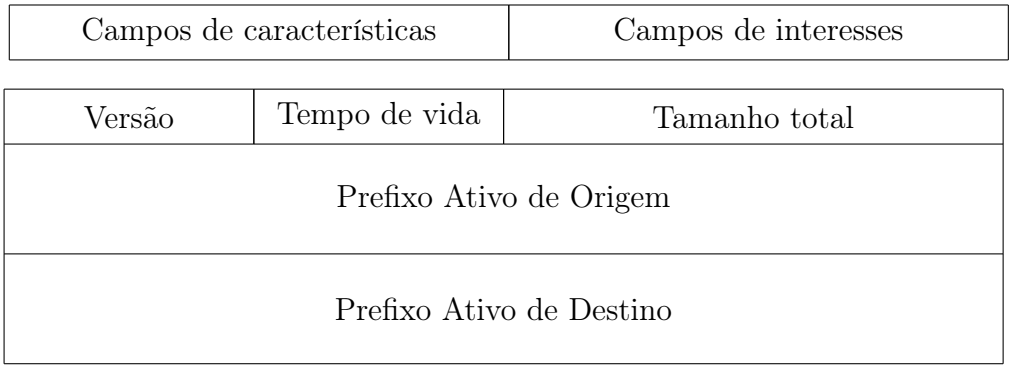


Figura 2.9: Prefixo Ativo e cabeçalho das mensagens. Imagem adaptada de Dutra et al. (2011)

Para comprovar a eficácia e a eficiência do Prefixo Ativo, os autores implementaram no simulador NS 3 (NS-3, 2014) um cenário com 150 nós móveis e realizaram uma comparação com os protocolos AODV (Ad hoc On-Demand Distance Vector) (Perkins et al., 2003) e Gossip (Haas et al., 2002). Nesse experimento foi possível constatar que o Prefixo Ativo permitiu uma taxa de envio de mensagem 16% maior com latência uma ordem de grandeza menor que os outros protocolos.

As principais contribuições do trabalho são: i) a introdução do Prefixo Ativo, permitindo a entrega probabilística de mensagem com baixa latência; ii) endereçamento de

nós por meio de características e interesses, permitindo inclusive o endereçamento de nós por IP; e iii) constatação da viabilidade do Prefixo Ativo por meio de experimentos e o desenvolvimento de uma aplicação de mensagem instantânea sobre o protocolo.

2.6.2 Protocolos da Rede Endereçada por Interesses

Em Dutra et al. (2010) é proposto o protocolo REPI (Rede Endereçada Por Interesses) para as redes orientadas por interesses. A sigla REPI também foi utilizada nos trabalhos para nomear o modelo de rede proposto. Os autores apresentam o problema do roteamento de mensagens em redes *Ad Hoc* sem fio e como o modelo de rede orientada a interesses pode ser utilizada nesse contexto. Também são apresentadas situações nas quais este modelo de rede pode ser empregado para facilitar a comunicação de indivíduos, como por exemplo, em casos de desastres naturais nos quais uma infraestrutura de rede pode ser precária ou mesmo inexistente. Como os autores sugerem, em um cenário como esse, as pessoas poderiam ser localizadas e localizar ajuda por meio de interesses como "médicos", "bombeiros" e "policiais".

O protocolo REPI emprega os conceitos de Prefixo Ativo apresentados em Dutra et al. (2011) e Dutra et al. (2012) e, segundo os autores, possui três características distintas: i) a rede é endereçada por termos; ii) a rede é de fato formada apenas no momento do envio de mensagens; e iii) ausência de endereçamento convencional fim-a-fim e independência de roteamento clássico, como em redes IP.

Com base nessas características, a rede opera da seguinte forma: as mensagens são enviadas por *broadcast*; o roteamento é realizado pelo casamento das características presentes nas mensagens e são entregues para as aplicações com base no casamento dos interesses; durante o envio das mensagens são formados grupos de interesses sob demanda. Os autores sugerem que os usuários podem utilizar seus dados biométricos nos campos de características da mensagem. Assim, em uma rede com uma grande quantidade de usuários, a probabilidade de entrega de mensagens se aproxima da distribuição normal. Em redes com poucos usuários tal característica não é válida e o sistema pode inserir valores automaticamente nesses campos para garantir a entrega.

O protocolo foi avaliado numa rede real composta por 20 nós e a taxa de entrega de mensagens foi comparada ao algoritmo "Gossip". O comportamento da rede foi monitorado com a ajuda do Sistema de Automação, Monitoramento e Configuração de Redes *Ad hoc* (SAMCRA) (Granja et al., 2010) para evitar que o tráfego gerado pela instrumentação do ambiente gerasse interferências nos experimentos. Os resultados apontaram a viabilidade

da rede orientada a interesses, sendo a taxa de entrega de mensagens equivalente ao "Gossip".

Em Granja (2010) são apresentadas as bases necessárias para a modelagem de uma rede orientada por interesses sobreposta a uma rede *Ad Hoc*. Foi especificada uma arquitetura denominada REPI-A, que é específica para rede *Ad Hoc*. As principais contribuições do trabalho são: i) o estabelecimento das bases necessárias para a implementação do Modelo Orientado a Interesses (MOI); ii) a implementação de uma REPI para redes *Ad Hoc* reconfigurável por meio de parâmetros; iii) desenvolvimento de métricas para a avaliação de uma REPI; e iv) desenvolvimento de um Sistema de Automação, Monitoramento e Configuração de Redes (SAMCRA) para auxiliar no processo de avaliação, parametrização e coleta das estatísticas de envio e recebimento de mensagens.

O meio físico utilizado para a troca de mensagens foi construído com aparelhos Tmote Sky (Tmote, 2016) usando o padrão 802.15.4/ZigBee (IEEE, 2003). O *middleware* necessário foi desenvolvido sobre o sistema operacional TinyOS (Hill et al., 2000). A aplicação mensageira foi desenvolvida para ser utilizada em computadores pessoais que usam os dispositivos Tmote para a comunicação. Tal aplicação também implementa o próprio protocolo REPI-A. No trabalho diferentes topologias são avaliadas com diferentes parâmetros com o uso de 20 aparelhos Tmote dispostos sobre mesas e com distâncias pré-determinadas. Os resultados apresentados mostram a viabilidade do uso da REPI-A, apresentando taxas de entrega de mensagens de mais de 90% em alguns cenários.

No trabalho desenvolvido por Moraes (2011) é apresentado o protocolo *Peer-to-Peer* REPI-Internet, uma aplicação da Rede Endereçada por Interesses no contexto da Internet. Nesta aplicação o modelo de rede endereçada por interesses foi sobreposto à Internet e um algoritmo de formação de vizinhança foi desenvolvido. Tal algoritmo pode selecionar aleatoriamente quais nós irão fazer parte da vizinhança ou utilizar os campos do Prefixo Ativo para isso. Um nó pode ter acesso aos Prefixos Ativos dos outros nós por meio de mensagens de controle enviadas para a rede. Uma vez em posse da lista de Prefixos Ativos, o nó pode estabelecer uma relação de vizinhança com os nós com o maior número de campos em comum.

O algoritmo de formação de vizinhança foi avaliado por meio de simulações no NS-3 (NS-3, 2014), variando-se a quantidade de nós na rede entre 50 e 100. A quantidade de mensagens de controle trocadas entre os nós utilizando-se a formação de vizinhança aleatória com base no Prefixo Ativo dos nós foi observada.

A avaliação do protocolo REPI-Internet foi realizada em um cenário no qual o número de nós variava entre 1024, 4096 e 10240. Nesse cenário o critério de formação de vizinhança

também foi variado entre o aleatório e o Prefixo Ativo. Os resultados da simulação apontaram uma taxa de entrega de 99% em todos os cenários avaliados.

2.6.3 Aplicações da Radnet

Em Moraes et al. (2012) é demonstrada a viabilidade da Rede Endereçada Por Interesses por meio do desenvolvimento de uma aplicação mensageira. Um programa servidor denominado **repd** foi desenvolvido para atuar como um intermediário entre as aplicações mensageiras. Para isso, uma API foi desenvolvida e disponibilizada aos usuários. Os testes foram realizados em *smartphones* executando o sistema operacional Android com as interfaces de rede configuradas em modo *Ad Hoc*. Vale salientar que os principais objetivos deste trabalho são realizar tal implementação no *kernel* do sistema operacional e permitir o uso da rede por meio da API de *sockets* padrão em sistemas UNIX.

Em Dutra et al. (2015) a Radnet é utilizada para transportar tarefas para a execução distribuída com o modelo de programação Dataflow. Para a implementação do trabalho, o serviço **repd** e a API apresentada por Moraes et al. (2012) foram utilizados. Nesse contexto, a Radnet foi utilizada para distribuir as tarefas e coletar os resultados. Novamente vale salientar que tal implementação pode ser realizada por meio da API de *sockets* a partir deste trabalho.

Em Goncalves et al. (2016) o modelo Radnet é empregado em redes veiculares, ou VANETs (*Vehicular Ad hoc Networks*). Nesse contexto, uma modificação do modelo Radnet foi proposto, denominado Radnet-VE (*Interest-Centric Mobile Ad Hoc Network for Vehicular Environments*), para possibilitar o roteamento e entrega de mensagens em ambientes nos quais veículos trocam informações sobre tráfego, condições da pista, etc.

Um novo protocolo denominado RVEP (*RAdNet-VE Protocol*) foi proposto para permitir o repasse de mensagens com base no prefixo de origem, interesse, posição relativa da origem e direção de propagação. O protocolo RVEP também é capaz de limitar o escopo da propagação de mensagens com base no número de nós registrados na rede e na identificação da estrada onde o nó se encontra. Para a realização dos experimentos, o *framework* de simulação de redes veiculares VEINS (Sommer et al., 2011) foi utilizado sobre redes *WiFi* 802.11n e 802.11p. Foram realizadas simulações em um segmento de pista de 10km com um fluxo de 1500 veículos/hora a uma velocidade de 80km/h.

Os experimentos mostraram que a aplicação da Radnet-VE permitiu uma comunicação com baixa latência, alta taxa de entrega de mensagens, escalabilidade e baixo *overhead* quando comparado com os modelos de rede existentes.

As principais contribuições do trabalho são: i) proposta do uso de redes centradas em conteúdo em VANETs; ii) extensão para Radnet para permitir seu uso em VANETs; e iii) demonstração da viabilidade do uso da Radnet-VE em VANETs.

No trabalho desenvolvido por Gondolfo (2014) foi realizada a implementação preliminar do modelo de rede Radnet em nível de *kernel* no sistema operacional FreeBSD. Foi adicionado suporte ao protocolo REPA na API de *sockets* do sistema para permitir que aplicações sejam desenvolvidas sobre a Radnet. Todo o monitoramento de estatísticas e também os mecanismos de configuração da rede foram disponibilizados por meio de *sysctls*. Uma ferramenta denominada *repa_dissector* foi desenvolvida para possibilitar a análise do tráfego da rede. O protocolo implementado neste trabalho não possui compatibilidade com a versão implementada por Moraes et al. (2012), o que impossibilita a interoperabilidade.

As principais contribuições do trabalho são permitir que a Radnet possa ser utilizada por meio de uma API padronizada e levar o processamento das mensagens para o *kernel* do sistema operacional, o que pode reduzir o consumo de recursos.

2.7 Considerações finais

Este Capítulo apresentou a fundamentação teórica que sustenta este trabalho. Redes *Ad hoc* possuem como uma de suas características o envio colaborativo de mensagens. O modo como as mensagens serão roteadas pelos nós da rede é um problema inerente a essa arquitetura. A Radnet é uma proposta de solução para esse problema, na qual as mensagens são encaminhadas com base no casamento de características dos nós ou usuários. Este trabalho se propõe a implementar a Radnet em nível de *kernel* do sistema operacional para possibilitar o seu uso por meio da API de *sockets* padrão, com suporte à criptografia transparente de mensagens. No próximo Capítulo são abordados os detalhes referentes a implementação da Radnet no *kernel* do sistema operacional FreeBSD.

3

Implementação da Radnet no *kernel*

Este Capítulo descreve as técnicas utilizadas para a implementação da Radnet no *kernel* do sistema operacional FreeBSD. Duas versões do protocolo foram implementadas, REPA_UEM e REPA_UFRJ. São apresentadas as estratégias utilizadas para a integração dos protocolos da Radnet com o *kernel* e como o suporte à criptografia foi implementado. O procedimento para acesso ao código pode ser obtido no Apêndice A.

3.1 Implementação

A organização da implementação em nível de *kernel* pode ser visto na Figura - 3.1. Um dos objetivos dessa estratégia é reduzir o *overhead* do processamento de pacotes.

Conforme os rótulos na Figura - 3.1, pode-se observar o seguinte comportamento do fluxo das mensagens: 1) um quadro chega na interface de rede conectada à um determinado meio físico; 2) o quadro poderá ser processado a partir do tratador de interrupções do sistema operacional; 3) o sistema irá alertar o subsistema de rede de que um quadro foi recebido, por meio do processamento do cabeçalho de camada de enlace o conteúdo do quadro será enviado para as rotinas específicas do protocolo encapsulado; 4) no caso de haver um programa de usuário com o mesmo interesse armazenado no cabeçalho da mensagem, ela será copiada para o espaço de usuário por meio de uma chamada de sistema; 5) as aplicações com o interesse registrado terão acesso ao conteúdo da mensagem; 6) caso ocorra um casamento dos prefixos, a mensagem é repassada para a rede.

A organização da implementação em espaço de usuário, como apresentado por Moraes et al. (2012), pode ser visto na Figura - 3.2. Como pode ser observado, existe um serviço em execução e uma API própria para o desenvolvimento de sistemas sobre a Radnet.

Conforme os rótulos na Figura - 3.2, pode-se observar o seguinte comportamento do fluxo das mensagens: 1) um quadro chega na interface de rede conectada à um determinado meio físico; 2) o quadro poderá ser processado a partir do tratador de interrupções do sistema operacional; 3) devido ao fato de o serviço que implementa a Radnet em espaço de usuário utilizar *sockets* brutos, o quadro é repassado sem modificações para o espaço de usuário; 4) e 5) por meio de uma chamada de sistema o quadro chega até a aplicação que implementa a Radnet; 6) se houverem aplicações com o mesmo interesse da mensagem recebida, elas terão acesso ao seu conteúdo; 7) o serviço realiza a tentativa de casamento do prefixo, e, se correr o casamento o quadro é enviado para o sistema operacional que 8) irá enviá-lo para rede sem modificações.

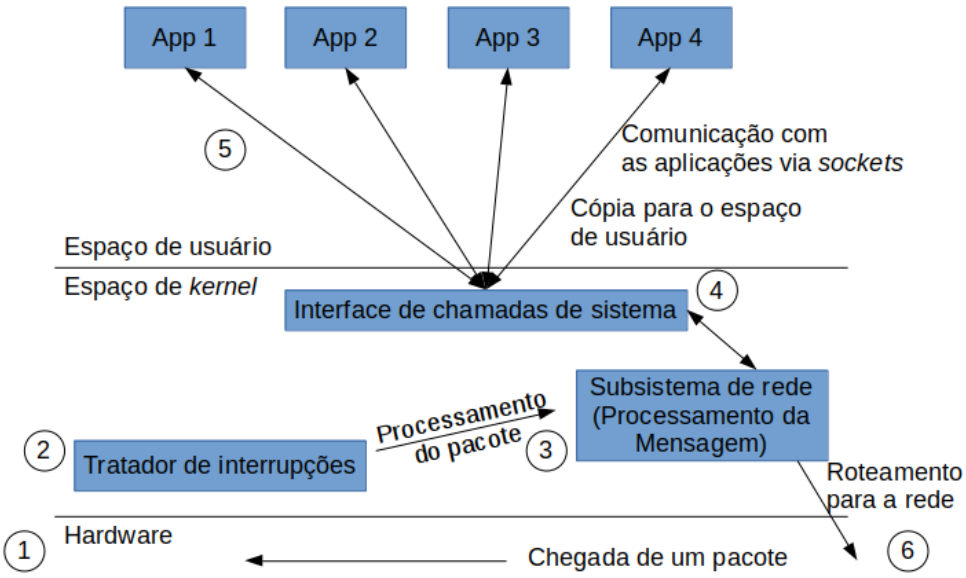


Figura 3.1: Implementação em espaço de *kernel*

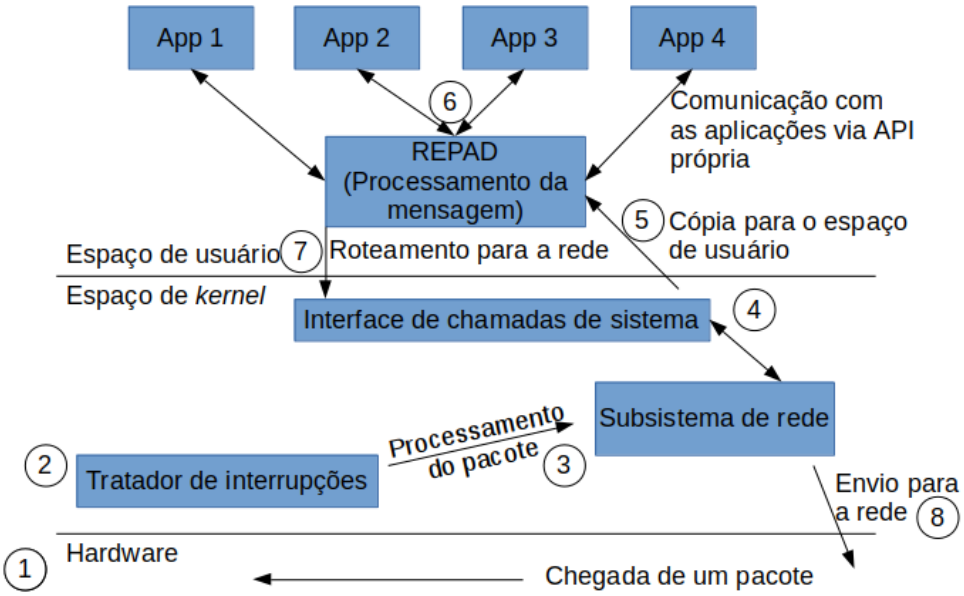


Figura 3.2: Implementação em espaço de usuário

A versão simplificada do protocolo REPA, aqui chamado de REPA_UEM, tem como principal objetivo reduzir o consumo de recursos computacionais durante a utilização da rede. Esse objetivo pode ser alcançado por meio da simplificação do cabeçalho das mensagens. Neste trabalho o campo de interesse da mensagem foi implementado em formato numérico, sendo armazenado como um *hash* da cadeia de caracteres (a função utilizada para isso foi a FNV (Fowler et al., 1991)) que representa o interesse. Além disso, outros campos do cabeçalho foram modificados, conforme é apresentado na Figura - 2.2 da Seção 2.3.

O protocolo REPA prevê o uso de mensagens criptografadas por meio da existência de um campo binário no cabeçalho que informa se a mensagem está cifrada ou não. Este trabalho implementa o suporte básico a criptografia diretamente no *kernel* do sistema operacional. Tal recurso, juntamente com o envio de interesses em formato de *hash*, garante o sigilo das informações que trafegam na rede.

Para o monitoramento e configuração da rede algumas ferramentas foram implementadas. As mensagens da rede podem ser monitoradas por meio da ferramenta *radnet_tool*, a qual permite a análise do cabeçalho das mensagens. A rede pode ser configurada e monitorada por meio das *sysctls* do sistema operacional. Com as *sysctls* são possíveis a alteração de parâmetros da rede em tempo de execução e a coleta de estatísticas. Alguns pontos do código podem ser monitorados por meio de sondas com a ferramenta de *profiling* dinâmico DTrade. Um novo provedor, denominado “radnet”, pode ser utilizado para a análise da execução da implementação.

Devido à política de distribuição do código fonte da implementação na qual este trabalho está baseado, a versão REPA_UFRJ foi implementada com base nas especificações encontradas nos trabalhos relacionados e na observação do tráfego de rede gerado pelos binários disponibilizados pelos autores do protocolo.

3.1.1 Inicialização

Durante a inicialização do sistema operacional, a rotina *radnet_init()* é executada. Ela é responsável pela alocação de memória de todas as estruturas de dados utilizadas durante o processamento de pacotes. Esta rotina é agendada para execução durante a inicialização do sistema por meio do *framework* SYSINIT (FreeBSD, 2016).

Outra tarefa importante da rotina *radnet_init()* é registrar a função *radnet_nh_input()* no serviço *netisr* do *kernel*. O *netisr* é utilizado pelo sistema para distribuir o processamento de pacotes de diferentes protocolos para suas respectivas funções de tratamento. A função *radnet_nh_input()* é a primeira a ter contato com os pacotes recebidos da Radnet.

A função *radnet_init()* é implementada no arquivo *sys/radnet/radnet.c*. Neste arquivo também são declaradas todas as *sysctls* utilizadas.

3.1.2 Habilitando uma interface de rede para permitir tráfego da Radnet

Na implementação deste trabalho, múltiplas interfaces de rede podem enviar e receber mensagens na rede. Essa decisão foi tomada visando-se permitir que pacotes possam ser roteados por diferentes meios físicos, por exemplo, recebidos via *WiFi* e reencaminhados via *Ethernet*. Essa funcionalidade também foi necessária para a implementação do ambiente de simulação.

Uma interface de rede é habilitada ou desabilitada para a Radnet por meio do comando *ifconfig*. Este comando foi modificado com chamadas à função *ioctl* aplicadas aos *sockets* do tipo Radnet. Esta ação invoca a função *repa_ioctl* implementada no *kernel*, que recebe o comando passado e um valor 0 ou 1, significando "interface desabilitada" e "interface habilitada", respectivamente. Uma nova variável, denominada *if_radnet_status*, foi inserida na estrutura *ifnet* para informar se a Radnet está habilitada ou não para uma determinada interface de rede.

Para habilitar e desabilitar a Radnet em uma interface de rede os seguintes comandos foram definidos, respectivamente: *ifconfig <interface> radnetenable* e *ifconfig <inter-*

face> radnetdisable. Para verificar se a Radnet está habilitada, basta executar o comando *ifconfig <interface>*.

3.1.3 Criação de *sockets*

No momento da criação de um *socket*, a rotina *repa_(uem,ufrj)_attach()*, implementada no arquivo *sys/radnet/(uem,ufrj)/repa_(uem,ufrj)_usrreq.c*, é executada e uma estrutura denominada PCB (*Protocol Control Block*) é criada. Para cada *socket* criado, um PCB é alocado e cada um deles irá armazenar um Prefixo Ativo.

3.1.4 Registro de interesses

O registro de interesses é feito por meio da chamada de sistema *bind()*. Um *socket* armazena apenas um interesse. Sempre que o desenvolvedor necessita registrar um novo interesse, um novo *socket* deve ser criado.

Sempre que um PCB é criado, ele é inserido em uma lista encadeada global que é utilizada para se ter acesso a todos os interesses de todos os processos que estão utilizando a Radnet no momento.

O PCB é utilizado principalmente durante o recebimento de mensagens. Sempre que uma mensagem é recebida, a lista de PCBs é percorrida em busca de todas as aplicações que registraram o interesse em questão. Caso alguma seja encontrada, ela é notificada para que possa acessar o conteúdo do pacote. Todas as rotinas de tratamento de PCBs estão implementadas no arquivo *sys/radnet/repa_pcb.c*.

3.1.5 Envio de mensagens

O envio de mensagens é feito por meio da chamada de sistema *sendto()*. Uma chamada à função *sendto()* invoca também a função *repa_(uem,ufrj)_sosend()*, implementada no arquivo *sys/radnet/(uem,ufrj)/repa_(uem,ufrj)_usrreq.c*. Esta função é responsável por alocar um *mbuf*, armazenar os dados vindos do espaço de usuário e configurar o cabeçalho do pacote no início do *mbuf*.

Por fim, um registro do pacote é inserido na tabela de controle de pacotes, para que não seja recebido da rede futuramente, e a rotina *repa_(uem,ufrj)_output()*, implementada no arquivo *sys/radnet/(uem,ufrj)/repa_(uem,ufrj)_output.c*, é invocada. Esta função é responsável por preparar o cabeçalho *ethernet* do pacote e invocar a rotina de envio implementada no *driver* do dispositivo de rede.

Também é possível enviar uma cópia do pacote para o próprio sistema local. Com isso, é possível permitir que as aplicações em execução no próprio nó recebam os pacotes enviados. Este recurso pode ser habilitado e desabilitado por meio de uma *sysctl* do sistema, como é descrito mais adiante.

3.1.6 Recebimento de mensagens

Após o recebimento de um pacote, a rotina *radnet_nh_input()* processa o início do cabeçalho da mensagem, determina qual a versão do protocolo e repassa-o para a rotina de tratamento do protocolo REPA, a *repa_(uem,ufrj)_input()*, implementada no arquivo *sys/radnet/(uem,ufrj)/repa_(uem,ufrj)_input.c*. É nesta função que a lógica do roteamento, descarte e envio das mensagens para as aplicações são implementadas.

Inicialmente, uma referência para o cabeçalho do pacote é acessada no *mbuf*. A partir do cabeçalho o interesse é copiado para um *buffer* e seu tamanho é calculado, no caso do protocolo REPA_UFRJ. Caso o protocolo seja o REPA_UEM, o interesse tem tamanho fixo de 4 *bytes*. A tabela que armazena o registro dos pacotes já processados é consultada, se o pacote em questão já foi recebido anteriormente, ele é descartado. Se o pacote nunca passou pelo nó, ele é registrado na tabela e a lista de PCBs é percorrida em busca de processos que registraram o interesse em questão. Caso algum seja encontrado, o processo é notificado para que possa ter acesso aos dados do pacote.

Por fim, caso o roteamento não tenha sido desabilitado pelo usuário, é realizada uma verificação da política de roteamento configurada no nó. Assim, o pacote será ou não repassado para a rede. O repasse de pacotes é feito pela rotina *repa_(uem,ufrj)_forward()*, implementada no arquivo *repa_(uem,ufrj)_output.c*

O recebimento de uma mensagem pela aplicação do usuário é feito por meio da chamada de sistema *recvfrom()*. Uma chamada a esta função invoca também a rotina *repa_(uem,ufrj)_soreceive()*, implementada no arquivo *sys/radnet/(uem,ufrj)/repa_(uem,ufrj)_usrreq.c*. Inicialmente, esta função adquire um *lock* do sistema, que faz a execução da aplicação ser suspendida. Esse *lock* será liberado no momento em que uma mensagem destinada a essa aplicação for recebida. Por fim, os dados são retirados da cadeia de *mbufs* e copiados para o espaço de usuário.

3.1.7 Monitoramento de eventos nos *sockets*

É possível utilizar a chamada de sistema *poll()* para a verificação de eventos de I/O (*Input/Output*) em um *socket* do protocolo REPA. A chamada de sistema *poll()* notifica a aplicação quando o *socket* possui dados que podem ser lidos ou quando nada foi recebido

em um período específico de tempo. Essa funcionalidade permite implementar rotinas que possam executar o tratamento de casos em que uma mensagem esperada nunca chegue à aplicação, por exemplo, devido a problemas na rede física. Nesses casos, a aplicação pode identificar a demora e reenviar uma mensagem de requisição de algum recurso.

3.1.8 Descarte de mensagens repetidas

O descarte é realizado quando uma mensagem que já havia sido repassada para a rede retorna ao nó devido ao envio em *broadcast*. A decisão de descartar a mensagem contribui com a redução do congestionamento na rede.

O controle das mensagens já enviadas é feito por meio de uma tabela *hash* e uma fila. A quantidade de mensagens memorizadas é controlada pelo tamanho da fila, que pode ser definido pelo usuário por meio de uma *sysctl*. Quando uma mensagem é enviada pelo nó ou repassada por ele, uma entrada na tabela é criada, usando-se o campo *timestamp* da mensagem como chave, e uma referência para essa entrada é inserida em uma fila. Quando a fila atinge um determinado tamanho, uma referência é removida (a mais antiga) e usada para remover a entrada da tabela. O controle de colisão na tabela é realizado por meio de *buckets* (listas encadeadas em que os elementos são armazenados caso uma colisão ocorra).

A tabela *hash* é utilizada para que a consulta por mensagens seja eficiente, dado que essa operação pode ser necessária milhares de vezes por segundo. A utilização do campo *timestamp* se mostrou suficiente para a identificação das mensagens, dado que sua precisão é em nanosegundos. A verificação da mensagem ocorre no momento de seu recebimento, antes de passar para as aplicações ou repassar para a rede. Uma consulta à tabela é realizada e, caso a mensagem esteja armazenada, o *mbuf* é liberado da memória e a função retorna imediatamente.

A função *hash* utilizada para a codificação do *timestamp* foi a Jenkins (Junior, 1997).

3.1.9 Estatísticas de tráfego

Informações sobre o tráfego são importantes durante investigações de problemas, por exemplo, de desempenho na rede. Esta implementação mantém contadores atualizados com as seguintes informações: quantidade de mensagens de entrada e saída, descartadas e repassadas para a rede, quantidade de *bytes* de entrada e saída e repassados para a rede.

Todas as estatísticas podem ser consultadas por meio de *sysctls*, como é apresentado no Capítulo 4. Para a implementação dos contadores foi utilizado o *framework counter*,

que possui como principal característica o fato de ser atualizado por cópias independentes em cada processador, evitando assim invalidações na memória *cache*.

3.1.10 Fechamento de *sockets*

Como os *sockets* são vistos pelas aplicações como descritores de arquivos, a chamada de sistema *close()* pode ser utilizada para fechar um *socket*.

A função *close()* invoca a rotina *repa_ (uem,ufrj)_ detach()*, implementada no arquivo *repa_ (uem,ufrj)_ usrreq.c*. Esta rotina é responsável por liberar a memória utilizada pelos PCBs.

3.1.11 Suporte à criptografia

O suporte à criptografia foi implementado em nível de *kernel* e é habilitado nos *sockets* individualmente por meio da chamada de sistema *setsockopt()*.

A implementação foi organizada de modo que qualquer algoritmo de criptografia simétrica possa ser utilizado. Neste trabalho apenas o algoritmo *Rijndael*, padrão AES, é utilizado. A implementação do algoritmo de criptografia utilizada faz parte do código já disponibilizado no *kernel* do sistema operacional FreeBSD e pode ser encontrado no diretório *sys/crypto/rijndael/*.

A criptografia das mensagens é ativada no *socket* quando a chamada de sistema *setsockopt* é aplicada recebendo como parâmetro a estrutura *repa_ crypto* contendo a configuração do algoritmo. Tal estrutura pode ser observada na Figura - 3.3. Os campos da estrutura são:

- *cipher_ algo*: especifica o algoritmo de criptografia simétrica que deve ser utilizado no *socket*. Em trabalhos futuros o suporte a diferentes algoritmos pode ser adicionado de maneira simples;
- *key_ size*: indica o tamanho em bits da chave que se deseja utilizar no algoritmo;
- *cipher_ mode*: especifica o modo de cifra deve ser utilizado em conjunto com o algoritmo de criptografia;
- *mac_ algo*: define o algoritmo de autenticação de mensagens deve ser utilizado. Neste trabalho o suporte a autenticação de mensagens não foi implementado;
- *key*: a chave que deverá ser utilizada.

```
struct repa_crypto {
    uint8_t      cipher_algo;
    uint16_t     key_size;
    uint8_t      cipher_mode;
    uint8_t      mac_algo;
    uint8_t      key[REPA_MAXKEYSIZE / 8];
};
```

Figura 3.3: Estrutura *repa_ crypto*

A criptografia da mensagem é realizada no momento em que o conteúdo é copiado do espaço de usuário para dentro do *kernel*. Nesse momento, a rotina de criptografia também recebe um Vetor de Inicialização (IV ou *Initialization Vector*) criado aleatoriamente. Esse IV é utilizado como entrada para o modo de cifra CBC e é enviado junto com a mensagem. Seu propósito é o de evitar que mensagens iguais deem origem ao mesmo texto cifrado. Vale salientar que o cabeçalho da mensagem não é criptografado, apenas o seu conteúdo.

Após a criptografia ser habilitada no *socket*, ela não pode ser desabilitada. O *socket* deve ser fechado, conforme apresentado anteriormente, e um novo deve ser criado.

3.2 Considerações finais

Neste Capítulo foram apresentados os detalhes inerentes a implementação da Radnet no sistema operacional FreeBSD. Uma das vantagens desta abordagem é a redução do *overhead* durante o processamento de mensagens, uma vez que elas não precisam mais serem copiadas para o espaço de usuário para que as decisões de roteamento sejam tomadas. O Capítulo também apresentou as rotinas implementadas para que o suporte à Radnet seja possível, o que inclui chamadas de sistema, mecanismos para configuração e coleta de estatísticas da rede e o suporte à criptografia. O próximo Capítulo apresenta como um usuário pode utilizar a API de *sockets* para interagir com a rede, os mecanismos empregados para alterar os parâmetros de configuração e monitoramento de tráfego.

4

Utilização e Monitoramento da Radnet

Este Capítulo apresenta como a Radnet pode ser utilizada por desenvolvedores por meio da API de *sockets* do sistema. Também são apresentados os mecanismos implementados para permitir o monitoramento da rede.

4.1 Utilização da Radnet

Esta Seção apresenta como os recursos adicionados à API de *sockets* podem ser utilizados para a criação de uma Radnet.

4.1.1 Criação de *sockets*

Os *sockets* são criados pelas aplicações por meio da chamada de sistema *socket()*, que recebe três parâmetros: o domínio de protocolos (AF_INET para IPv4, AF_INET6 para IPv6 e AF_RADNET para os protocolos da Radnet), o tipo do protocolo (a semântica da comunicação) e o protocolo dentro do domínio.

A Figura - 2.5 apresenta como a chamada de sistema *socket* pode ser utilizada. Optou-se por utilizar o tipo SOCK_DGRAM como segundo parâmetro devido à semântica ser a mesma do protocolo REPA: sem conexão e sem garantia de entrega.

Um *socket* pode ser fechado por meio da chamada de sistema *close()*.

4.1.2 Registro de interesses

O registro de interesses e características é feito por meio da chamada de sistema *bind()*. A Figura - 4.1 apresenta como o *bind()* pode ser utilizado. Os parâmetros utilizados são: o *socket* criado previamente, um ponteiro para a estrutura com as informações de endereçamento e o tamanho da estrutura.

A estrutura *sockaddr_radnet* é apresentada na Figura - 4.2. O campo *radnet_len* armazena o tamanho total da estrutura. Ele é utilizado pelas rotinas do *kernel* no momento da cópia dessas informações. O campo *radnet_family* armazena o número identificador da família de protocolos Radnet, definido como AF_RADNET. O campo *radnet_prefix* armazena o prefixo da aplicação. O campo *radnet_interest* é um ponteiro para uma cadeia de caracteres que representa um dos interesses da aplicação.

```
struct sockaddr_radnet radnet;

radnet.radnet_len = size;
radnet.radnet_family = AF_RADNET;
radnet.radnet_prefix = 0xAABBCCDD;
radnet.radnet_interest = "Interesse";

bind(s, (struct sockaddr *) &radnet, sizeof(radnet));
```

Figura 4.1: Utilização da chamada de sistema *bind()*

```
struct sockaddr_radnet {
    uint8_t          radnet_len;
    sa_family_t      radnet_family;
    uint32_t         radnet_prefix;
    char             *radnet_interest;
 };
```

Figura 4.2: Estrutura *sockaddr_radnet*

4.1.3 Envio de mensagens

O protocolo REPA da Radnet é caracterizado como sem conexão. Desse modo, as mensagens devem ser enviadas e recebidas por meio das chamadas de sistema *sendto()* e *recvfrom()* (ou *sendmsg()* e *recvmsg()*), respectivamente, ao contrário de protocolos orientados a *streams*, como o TCP, que podem utilizar as funções *write()* e *read()*.

A Figura - 4.3 apresenta como a chamada de sistema *sendto()* pode ser utilizada para enviar mensagens para a rede. Os parâmetros utilizados são: o *socket* previamente criado, o endereço do *buffer* contendo a mensagem, o tamanho da mensagem, as *flags* utilizadas (as *flags* são dependentes de protocolo e não são implementadas neste trabalho), um ponteiro para a estrutura contendo o interesse de destino da mensagem e o tamanho da estrutura.

Caso a execução seja bem sucedida, a chamada de sistema *sendto()* retornará a quantidade de *bytes* enviados na mensagem, caso contrário será retornado o valor -1.

```
sendto(s, "Oi!", 3, 0, (struct sockaddr *) &radnet, size);
```

Figura 4.3: Uso da chamada de sistema *sendto()*

4.1.4 Recebimento de mensagens

O recebimento de mensagens é feito por meio da chamada de sistema *recvfrom()*. A Figura - 4.4 apresenta como ela é utilizada. Os parâmetros utilizados são: o *socket* previamente criado, um ponteiro para um *buffer* onde a mensagem será armazenada, a quantidade máxima de *bytes* que poderão ser recebidos, as *flags* utilizadas (não implementadas neste trabalho), um ponteiro para uma estrutura do tipo *sockaddr_radnet* onde os dados do remetente serão armazenados (este parâmetro pode ser NULL caso não seja necessário) e um ponteiro para uma variável contendo o tamanho da estrutura.

```
rs = recvfrom(s, mensagem, 32, 0, (struct sockaddr *) &info,
    &size_info);
```

Figura 4.4: Uso da chamada de sistema *recvfrom()*

Na Figura - 4.5 é possível ver um código completo exemplificando a utilização do protocolo REPA_UEM com a API de *sockets* no envio e recebimento de uma mensagem.

4.1.5 Monitoramento de eventos

A chamada de sistema *poll()* pode ser utilizada nos *sockets* do protocolo REPA para o monitoramento de eventos de I/O. Em casos nos quais o sistema deve aguardar a resposta de alguma solicitação de recursos enviada para a rede, pode ocorrer de a resposta nunca chegar. Em uma situação como essas, o monitoramento de eventos pode ser utilizado

```
#include <stdio.h>
#include <sys/socket.h>
#include <radnet/radnet.h>

int main() {
    int s, rs;
    socklen_t size, size_info;
    char mensagem[32];
    struct sockaddr_radnet radnet, info;

    s = socket(AF_RADNET, SOCK_DGRAM, RADNETPROTO_REPA_UEM);

    size = sizeof(radnet);

    radnet.radnet_len = size;
    radnet.radnet_family = AF_RADNET;
    radnet.radnet_prefix = 0xAABBCCDD;
    radnet.radnet_interest = "Interesse";

    bind(s, (struct sockaddr *) &radnet, size);

    sendto(s, "Oi!", 3, 0, (struct sockaddr *) &radnet, size);

    size_info = sizeof(info);

    rs = recvfrom(s, mensagem, 32, 0, (struct sockaddr *) &info,
        &size_info);

    mensagem[rs] = '\0';
    printf("Prefixo: %u, Interesse: %s, Mensagem: %s",
        info.radnet_prefix, radnet.radnet_interest, mensagem);

    return 0;
}
```

Figura 4.5: Envio e recebimento de mensagens na Radnet com o protocolo REPA_UEM

para detectar a demora no recebimento e notificar a aplicação de que alguma ação deve ser tomada.

A chamada de sistema *poll()* recebe como argumentos um vetor contendo os descritores de arquivos que se deseja monitorar, a quantidade de descritores no vetor e um valor de tempo em milissegundos, utilizado para continuar a execução do código após o tempo

de espera atingir o valor passado como argumento. É possível aguardar por eventos nos descritores de arquivos por um período indeterminado de tempo, para isso basta utilizar o valor -1 como último parâmetro.

Um exemplo de utilização da chamada de sistema *poll()* pode ser encontrado na implementação do nosso Centro de Distribuição de Chaves, descrito no Capítulo 5. A Figura - 4.6 apresenta como a função pode ser utilizada.

```
struct pollfd pfd;
int key_received = 0;

pfd.fd = recv_sock;
pfd.events = POLLIN;

sendto(req_sock, "danilo", 6, 0,
    (struct sockaddr *) &req_raddr, sizeof(req_raddr));

while(!key_received) {
    int p = poll(&pfd, 1, 2000);
    if(p == 0) {
        printf("Chave nao recebida apos 2 segundos\n");
        printf("Solicitando novamente...\n");
        sendto(req_sock, "danilo", 6, 0,
            (struct sockaddr *) &req_raddr, sizeof(req_raddr));
    } else {
        msize = recvfrom(recv_sock, buff, 1200, 0, NULL, NULL);
        key_received = 1;
    }
}
```

Figura 4.6: Monitoramento de eventos

4.1.6 Utilização da chamada de sistema *ioctl()*

A chamada de sistema *ioctl* é utilizada neste trabalho pelo programa *ifconfig* para habilitar e desabilitar o suporte à Radnet em uma interface de rede. A função *ioctl()* recebe obrigatoriamente um descritor de arquivo e um comando. Após os dois parâmetros iniciais (o descritor de arquivo e o comando), uma lista de tamanho variável de parâmetros é aceita.

O descritor de arquivo deve ser um *socket* criado com o protocolo REPA, conforme mostra a Figura - 2.5. Os comandos possíveis são: *SIOCGRADNETSTATUS*, utilizado

para recuperar o estado da Radnet na interface de rede, e *SIOCSRADNETSTATUS*, utilizado para modificação do estado.

Um terceiro parâmetro utilizado é um ponteiro para uma estrutura do tipo *ifreq*. Foi inserido um novo campo nesta estrutura para armazenar o estado da Radnet na interface de rede. A Figura - 4.7 apresenta como a chamada de sistema *ioctl* pode ser utilizada. Este trecho de código foi utilizado no programa *ifconfig* para habilitar a Radnet em uma interface de rede.

```
struct ifreq ifreq = ifr;
int error;
s = socket(AF_RADNET, SOCK_DGRAM, RADNETPROTO_REPA_UEM);
ifreq.ifr_radnet = 1;

error = ioctl(s, SIOCSRADNETSTATUS, &ifreq);
```

Figura 4.7: Uso da chamada de sistema *ioctl()*

4.1.7 Ativação do suporte à criptografia em um *socket*

O suporte à criptografia pode ser habilitado no *socket* por meio da chamada de sistema *setsockopt*. A Figura - 4.8 apresenta como a rotina *setsockopt* pode ser utilizada.

Em trabalhos futuros uma maior variedade de algoritmos de criptografia, modos de cifra e algoritmos de autenticação de mensagens pode ser adicionada ao trabalho sem grandes dificuldades, pois a implementação proposta tenta oferecer uma interface genérica para chamada de funções criptográficas e não se restringe a um algoritmo específico.

```
struct repa_crypto rc;
rc.cipher_algo = REPA_ALGO_AES;
rc.key_size = REPA_KEYSIZE_256;
rc.cipher_mode = REPA_MODE_CBC;
rc.mac_algo = 0;
memcpy(&rc.key, key, 32);

setsockopt(sock, 0, SO_UEM_SETKEY, &rc, sizeof(rc));
```

Figura 4.8: Ativando a criptografia em um *socket*

4.2 Monitoramento e configuração da Radnet

Esta Seção apresenta a ferramenta *radnet_tool*, criada para o monitoramento e teste da rede, e os meios utilizados para a configuração e coleta de estatísticas da Radnet.

4.2.1 A ferramenta *radnet_tool*

Para o monitoramento do tráfego de mensagens na Radnet, foi desenvolvida a ferramenta *radnet_tool*. Ela é capaz de capturar mensagens em tempo real e apresentar o conteúdo tanto do cabeçalho quanto do *payload*.

Para a captura das mensagens foi utilizada a biblioteca *libpcap* (TCPDUMP/LIBPCAP, 2014). Com ela é possível capturar pacotes diretamente de interfaces de rede e também de arquivos, como por exemplo, tráfego capturado pelas ferramentas *tcpdump* (TCPDUMP/LIBPCAP, 2014) e *Wireshark* (Wireshark, 2014).

A *radnet_tool* é parametrizada para permitir uma utilização flexível, como por exemplo, a escolha de uma interface de rede específica para ser monitorada, determinar quantos pacotes deverão ser capturados e o nível de detalhamento no qual os pacotes serão apresentados. Os seguintes parâmetros são utilizados:

- -i <interface>. Realiza a captura do tráfego em uma interface de rede;
- -n. Quantidade de pacotes que se deseja capturar;
- -d. Nível de *debug*.

A Figura - 4.9 apresenta a saída da ferramenta para uma mensagem do protocolo REPA_UFRJ capturada. No exemplo, os parâmetros "-i" e "-d 2" foram utilizados.

4.2.2 *Profiling* e estatísticas de tráfego

A visualização das estatísticas de tráfego da Radnet foi implementada por meio de *sysctls*. O ramo *net.radnet* foi introduzido na árvore de *sysctls* do *kernel*.

O *framework counter* do *kernel* foi utilizado para a implementação dos contadores de mensagens recebidas, enviadas, repassadas e descartadas.

As estatísticas podem ser visualizadas com o comando *sysctl net.radnet.stats*. A Figura - 4.10 apresenta a saída do comando.

Os significados dos contadores de estatística são:

```
Type: REPA_UFRJ, Length: 60, From: 00:a0:98:27:8d:55, Version: 1,
Hide flag: 0, Encrypted: 0, Other Flags: 0, TTL: 64,
Hdr. Len: 30, Seq. Num: 2,
Dst. Prefix: 3737169374 (11011110110000001010110111011110),
Src. Prefix: 1234 (00000000000000000000001010011010),
Timestamp: 1426025114903463 (2015-03-10 19:05:14),
Interest: Goiaba
0000:.........'.U1E.@ FF FF FF FF FF FF 00 A0 98 27 8D 55 31 45 01 40
0010:................ 00 1E 00 00 00 02 DE C0 AD DE 00 00 02 9A 00 05
0020:..Xj#.GoiabaAAAA 10 F6 58 6A 23 A7 47 6F 69 61 62 61 41 41 41 41
0030:A.......... 41 00 00 00 00 00 00 00 00 00 00 00
```

Figura 4.9: Saída da ferramenta *radnet_tool*

```
net.radnet.stats.input_packets: 1048104
net.radnet.stats.output_packets: 467709
net.radnet.stats.forwarded_packets: 467709
net.radnet.stats.discarded_packets: 580372
net.radnet.stats.invalid_packets: 0
net.radnet.stats.input_bytes: 934908768
net.radnet.stats.output_bytes: 417196428
net.radnet.stats.forwarded_bytes: 417196428
```

Figura 4.10: Saída do comando *sysctl net.radnet.stats*

- net.radnet.stats.input_packets. Quantidade de mensagens recebidas pelas interfaces de rede;
- net.radnet.stats.output_packets. Quantidade de mensagens enviadas pelas interfaces de rede;
- net.radnet.stats.forwarded_packets. Quantidade de mensagens repassadas para a rede;
- net.radnet.stats.discarded_packets. Quantidade de mensagens descartadas;
- net.radnet.stats.invalid_packets. Quantidade de mensagens inválidas;
- net.radnet.stats.input_bytes. Quantidade de bytes recebidos pelas interfaces de rede;
- net.radnet.stats.output_bytes. Quantidade de bytes enviados pelas interfaces de rede;

- net.radnet.stats.forwarded_bytes. Quantidade de bytes repassados para a rede.

O *profiling* da implementação pode ser realizado por meio do novo provedor "radnet" adicionado ao DTrace. A Figura - 4.11 apresenta as sondas estáticas criadas no código. O significado de cada sonda é:

```
   ID   PROVIDER            MODULE   FUNCTION NAME
88144   radnet              radnet   radnet_input ufrj
88145   radnet              radnet   radnet_input uem
88146   radnet              radnet   radnet_input unknown
88147   radnet              radnet   radnet_loopback input
88148   radnet         repa_utils    packet_lookup hit
88149   radnet         repa_utils    packet_lookup miss
88150   radnet         repa_utils    prefix_match match
88151   radnet         repa_utils    prefix_match unmatch
88152   radnet     repa_uem_input    uem_input waking-up
88153   radnet     repa_uem_input    uem_input zero-ttl-reached
88154   radnet     repa_uem_input    uem_input forwarding
88155   radnet     repa_uem_input    uem_input already-forwarded
88156   radnet    repa_ufrj_input    ufrj_input waking-up
88157   radnet    repa_ufrj_input    ufrj_input zero-ttl-reached
88158   radnet    repa_ufrj_input    ufrj_input forwarding
88159   radnet    repa_ufrj_input    ufrj_input already-forwarded
```

Figura 4.11: Saída do comando *dtrace -ln radnet:::*

- radnet:radnet:radnet_input:ufrj. Disparada quando um pacote do protocolo REPA_UFRJ é processado;
- radnet:radnet:radnet:radnet_input:uem. Disparada quando um pacote do protocolo REPA_UEM é processado;
- radnet:radnet:radnet_input:unknown. Disparada quando um pacote desconhecido é processado;
- radnet:radnet:radnet_loopback:input. Disparada quando um pacote é recebido na interface de *loopback*;
- radnet:repa_utils:packet_lookup:hit. Disparada quando um pacote é encontrado na tabela que registra os pacotes já recebidos;
- radnet:repa_utils:packet_lookup:miss. Disparada quando um pacote não é encontrado na tabela que registra os pacotes já recebidos;

- radnet:repa_utils:prefix_match:match. Disparada quando ocorre um casamento de Prefixo;
- radnet:repa_utils:prefix_match:unmatch. Disparada quando não ocorre um casamento de Prefixo;
- radnet:repa_uem_input:uem_input:waking-up. Disparada quando um processo é acordado para receber um pacote do procotolo REPA_UEM destinado ao interesse registrado por ele;
- radnet:repa_uem_input:uem_input:zero-ttl-reached. Disparada quando o campo TTL (*Time To Live*) de um pacote do protocolo REPA_UEM chega a zero;
- radnet:repa_uem_input:uem_input:forwarding. Disparada quando ocorre um repasse de um pacote do protocolo REPA_UEM;
- radnet:repa_uem_input:uem_input:already-forwarded. Disparada quando um pacote do protocolo REPA_UEM recebido já foi repassado;
- radnet:repa_ufrj_input:ufrj_input:waking-up. Disparada quando um processo é acordado para receber um pacote do procotolo REPA_UFRJ destinado ao interesse registrado por ele;
- radnet:repa_ufrj_input:ufrj_input:zero-ttl-reached. Disparada quando o campo TTL de um pacote do protocolo REPA_UFRJ chega a zero;
- radnet:repa_ufrj_input:ufrj_input:forwarding. Disparada quando ocorre um repasse de um pacote do protocolo REPA_UFRJ;
- radnet:repa_ufrj_input:ufrj_input:already-forwarded. Disparada quando um pacote do protocolo REPA_UFRJ recebido já foi repassado.

Com o DTrace também é possível a observação de detalhes internos da execução do sistema operacional. A Figura - 4.12 apresenta um exemplo de como a pilha de chamadas de função pode ser observado com a ferramenta DTrace. A saída apresentada foi obtida com o comando "*dtrace -n 'fbt:kernel:repa_uem_interests_compare:entry {stack();}'* "

4.2.3 Configuração dos parâmetros da Radnet

A parametrização da Radnet é realizada por meio de *sysctls*. O ramo *net.radnet.control* foi criado com esse propósito, conforme pode ser observado na Figura - 4.13. A listagem a seguir apresenta o significado de cada variável:

```
0  26936 repa_uem_interests_compare:entry
          kernel`repa_uem_input+0x2c1
          kernel`radnet_input_internal+0x108
          kernel`radnet_nh_input+0x15
          kernel`netisr_dispatch_src+0xef
          kernel`netisr_dispatch+0x1f
          kernel`ether_demux+0x298
          kernel`ether_input_internal+0x706
          kernel`ether_nh_input+0x1d
          kernel`netisr_dispatch_src+0xef
          kernel`netisr_dispatch+0x1f
          kernel`ether_input+0x23
          kernel`re_rxeof+0x8b6
          kernel`re_int_task+0x19d
          kernel`taskqueue_run_locked+0x194
          kernel`taskqueue_run+0x15f
          kernel`taskqueue_fast_run+0x19
          kernel`intr_event_execute_handlers+0x2a1
          kernel`ithread_execute_handlers+0x4e
          kernel`ithread_loop+0xc2
          kernel`fork_exit+0x159
```

Figura 4.12: Pilha de chamadas das funções no momento da execução da rotina *repa_uem_interests_compare*

```
net.radnet.control.enable_loopback: 0
net.radnet.control.packets_queue_size: 100
net.radnet.control.packets_hash_size: 4096
net.radnet.control.matching_scheme: 0
net.radnet.control.disable_forwarding: 0
net.radnet.control.random_mac_address: 0
```

Figura 4.13: Saída do comando *sysctl net.radnet.control*

- net.radnet.control.random_mac_address. Para cada mensagem enviada um endereço MAC (*Media Access Control*) de origem aleatório é gerado. Essa funcionalidade foi inserida como um meio de aumentar o anonimato na rede. Com o valor 0 ela é desabilitada e com 1 habilitada;

- net.radnet.control.disable_forwarding. Permite controlar o repasse de mensagens para a rede. Com o valor 0 o repasse é habilitado e com 1 desabilitado;

- net.radnet.control.matching_scheme. Permite modificar o comportamento do casamento de prefixos. Com o valor 0, a mensagem é repassada para a rede se um ou mais campos do prefixo do remetente coincidir com o prefixo de uma das aplicações em execução no sistema. Com o valor 1, todos os campos devem coincidir. Com o valor 2, as mensagens são repassadas sem que haja casamento de campos do prefixo;

- net.radnet.control.packets_hash_size. Permite determinar o tamanho da tabela que armazena o histórico das mensagens reenviadas para a rede para evitar o repasse de uma mesma mensagem duas vezes. O valor padrão é 4096 entradas e pode ser modificado apenas durante o carregamento do sistema operacional;

- net.radnet.control.packets_queue_size. Permite determinar a quantidade de mensagens armazenadas no histórico. O valor padrão é 100 e pode ser modificado apenas durante o carregamento do sistema operacional;

- net.radnet.control.enable_loopback. Permite habilitar e desabilitar o envio de mensagens para o sistema local.

4.3 Considerações finais

Este Capítulo apresentou como a implementação proposta pode ser utilizada para criar e interagir com uma Radnet. Aplicações sobre a Radnet podem ser desenvolvidas com a API de *sockets* do sistema operacional. Também foram implementados mecanismos que permitem a coleta de estatísticas da rede, por meio de *sysctls*, análise da execução da rotinas implementadas, por meio de sondas da ferramenta Dtrace, e análise de tráfego, por meio da ferramenta radnet_tool. O próximo Capítulo apresenta a metodologia utilizada para a execução dos experimentos e os resultados obtidos no trabalho.

5

Avaliação dos Resultados

Para a realização da avaliação dos resultados foram realizados os seguintes experimentos: 1) avaliação da implementação por meio da simulação de uma rede *ad-hoc* com o uso de máquinas virtuais para a validação do funcionamento do roteamento de mensagens; 2) avaliação do protocolo REPA_UEM por meio do envio e recebimento massivo de mensagens para a análise do consumo de recursos computacionais; e 3) demonstração do suporte à criptografia por meio da implementação de um centro de distribuição de chaves para a troca de mensagens codificadas.

5.1 Metodologia dos Experimentos

Esta Seção apresenta as metodologias utilizadas para a execução dos experimentos. A Seção está organizada de acordo com a apresentação dos resultados.

5.1.1 Avaliação da Implementação

Para a validação da implementação em relação ao roteamento, uma rede com a topologia apresentada na Figura - 5.1 foi construída com máquinas virtuais e interligadas por meio de interfaces de rede *bridges*. A simulação foi realizada em um sistema operacional Linux (distribuição CentOS 7.2). Para a criação das máquinas virtuais foi utilizado o sistema de virtualização Qemu com suporte a aceleração via KVM (KVM, 2016). Cada máquina virtual possui 1 CPU, 128MB de memória e quantidade de interfaces de rede variável de acordo com o número de máquinas diretamente conectadas à ela no ambiente simulado.

Na Figura - 5.1, os nós da rede são representados por círculos com um número de identificação. Os círculos estão agrupados de acordo com o alcance entre os nós. Por exemplo, os nós 1, 2, 3, 4 e 5 conseguem enviar mensagens diretamente entre eles, sem o auxílio de saltos intermediários. Os nós localizados em grupos diferentes só conseguirão realizar a troca de mensagens por meio da comunicação (saltos) entre outros nós. Observa-se que alguns nós fazem parte de dois ou mais grupos, significando que conseguem enviar mensagens para nós de grupos diferentes. Na prática esses nós irão colaborar com o roteamento da rede, enviando mensagens para nós que não estão ao alcance direto uns dos outros.

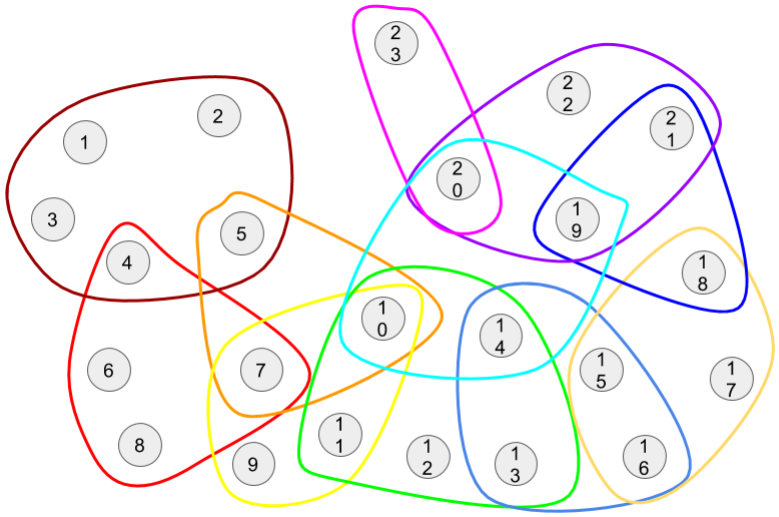


Figura 5.1: Topologia da rede utilizada nas simulações

5.1.2 Avaliação do protocolo REPA_UEM

Foram desenvolvidos um programa de envio e outro de recebimento de mensagens para a execução do experimento. No momento do carregamento do sistema operacional, o programa de recebimento é iniciado em segundo plano. O prefixo de cada *socket* é configurado dinamicamente quando o programa é executado e o número gerado é baseado na distribuição normal. Logo, o prefixo gerado será diferente a cada execução. O mesmo se aplica ao programa de envio de mensagens.

Foram utilizados dois computadores com processadores Intel Core i7-3770 de 3.4GHz, 8GB de memória principal com barramento de 1333MHz e interfaces de rede Gigabit Realtek 8168/8111. Os computadores foram interligados diretamente por meio de um cabo Ethernet Cat 6A.

O monitoramento do recebimento da mensagens foi feito por meio de interfaces seriais da máquina virtual mapeadas em arquivos de texto no sistema operacional anfitrião. Dessa forma, observando o conteúdo de cada arquivo é possível analisar quais nós estão recebendo as mensagens em uma determinada configuração de prefixo.

Para a realização dos experimentos de análise de consumo de recursos, dois programas, um de envio e outro de recebimento de mensagens, foram desenvolvidos. Tais programas permitem a configuração de parâmetros como tamanho da mensagem, tamanho do interesse, quantidade de pacotes a serem enviados e tempo em segundos em que mensagens são enviadas ininterruptamente.

Foram utilizadas quatro aplicações de cada protocolo registradas com o mesmo interesse no sistema receptor das mensagens. Isso significa que a rotina de processamento de mensagens do *kernel* irá percorrer uma lista contendo quatro elementos e executará o casamento entre os interesses de cada aplicação com o interesse da mensagem recebida. Essa verificação é executada para cada mensagem recebida. Este número foi escolhido para reduzir a concorrência pela CPU entre os programas usados no experimento e as rotinas do *kernel*, uma vez que a CPU utilizada possui oito *threads*.

No sistema transmissor, foi utilizada uma aplicação que envia mensagens de 1024 *bytes* variando o tamanho dos interesses entre os valores 4, 8, 16, 32, 64, 128 e 254 *bytes*. O valor 254 foi utilizado devido o tamanho máximo de um interesse ser definido como 255 e a cadeia de caracteres do interesse utilizar o caractere nulo para representar o fim da cadeia.

Para permitir uma maior taxa de entrega, o suporte ao repasse de mensagens foi desabilitado. Com o uso do repasse cada mensagem recebida seria devolvida ao remetente, causando um maior consumo dos recursos dos sistemas e da rede.

O uso de apenas dois nós durante os experimentos se justifica pela natureza dos testes realizados. O objetivo do experimento foi determinar o impacto no desempenho de envio e recebimento de pacotes causado pelas modificações no cabeçalho, sendo suficiente a utilização de dois nós de rede.

Para a coleta de informações sobre número de mensagens processadas e o tempo de execução da rotina de tratamento das mensagens recebidas foi utilizada a ferramenta de instrumentação dinâmica de código DTrace (Cantrill et al., 2004) que está disponível nativamente no sistema (McKusick et al., 2014). Com o DTrace é possível determinar o tempo de duração de uma chamada a uma determinada função. Ele também é capaz de calcular o tempo médio e o desvio padrão dos dados coletados.

A quantidade de instruções executadas durante o processamento das mensagens foi coletada com a ferramenta *pmcstat* (FreeBSD, 2015). O *pmcstat* funciona em conjunto com os PMCs (*Performance Monitoring Counters*) da CPU.

Durante a execução dos experimentos, os programas do sistema que podem executar tarefas agendadas, como o *cron*, e gravação de registros de *logs*, como o *syslog*, foram desabilitadas para reduzir a possibilidade de interferência nos resultados.

Devido ao fato de o cabeçalho utilizado na versão REPA_UEM possuir tamanho fixo, é esperado que os resultados dos experimentos para tamanhos diferentes de interesse sejam parecidos. A variação dos resultados ocorre com a versão REPA_UFRJ, no qual o tamanho do cabeçalho aumenta junto com o interesse.

Este experimento foi dividido em três classes: 1) quantidade de mensagens processadas pelo *kernel*; 2) quantidade de instruções de CPU executadas durante a transmissão de mensagens; e 3) tempo gasto no processamento de mensagens individuais.

5.1.3 Validação do Suporte à Criptografia

Para implementar o suporte à criptografia, foi desenvolvido um KDC (*Key Distribution Center*) simples que envia uma chave simétrica, utilizada para que os nós da rede possam codificar e decodificar o tráfego. Tal chave é criptografada por uma chave pública de cada nó armazenada em uma base de dados. Ao receber a chave simétrica, o nó irá decodificá-la e aplicá-la ao *socket* utilizando a função *setsockopt()* da API de *sockets*. A partir desse momento toda mensagem enviada pelo *socket* em questão é criptografada.

O cenário utilizado para o experimento com o KDC é o mesmo apresentado na Figura - 5.1 com a adição de mais um nó com a função de distribuir as chaves na rede. O cenário modificado pode ser observado na Figura - 5.2. O nó com o KDC foi introduzido no mesmo grupo dos nós 23 e 20, logo, para que um nó em outro grupo consiga ter acesso à chave criptográfica será necessário o roteamento colaborativo dos nós intermediários.

5.2 Simulação de uma Radnet

Para avaliar o funcionamento da implementação proposta, uma rede de 23 máquinas virtuais foi configurada. As máquinas foram agrupadas de forma que nós no mesmo grupo consigam trocar mensagens diretamente sem a necessidade do roteamento colaborativo.

O experimento executado tem como objetivo verificar se o roteamento probabilístico está ocorrendo de fato. Para isso, uma aplicação receptora de mensagens foi desenvolvida com a capacidade de registrar um conjunto de características aleatórias com base na

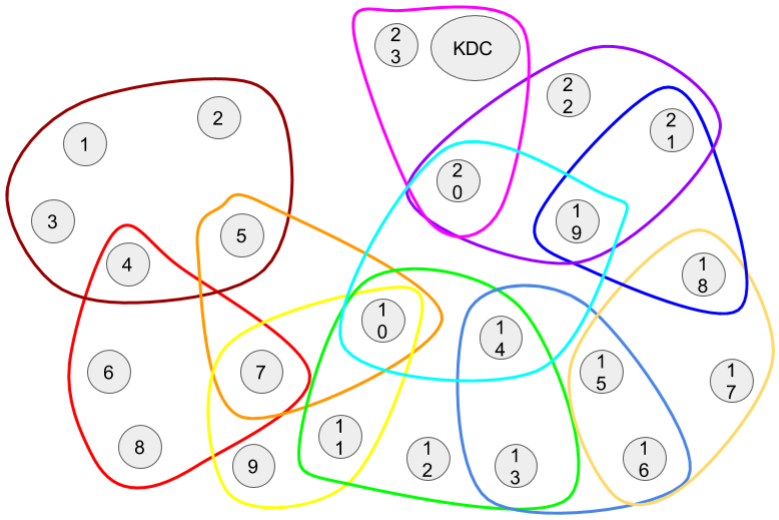


Figura 5.2: Topologia da rede utilizada nas simulações com o KDC

distribuição normal. No momento do carregamento do sistema operacional, cada nó inicia a aplicação em *background* que será responsável por registrar um interesse e as características do nó.

Um conjunto de nós foi selecionado para atuar como origem das mensagens. O critério utilizado para a seleção dos nós de origem teve como base a quantidade de nós alcançados diretamente a partir deles (sem a necessidade de saltos intermediários). Foram selecionados alguns nós com poucos elementos cujo o acesso pode ser feito diretamente e outros com muitos. No experimento foram enviadas mensagens de cada nós selecionado e foi observada a taxa de entrega. Devido ao ambiente simulado não gerar perdas de pacotes, o único motivo de um nó não ser alcançado é o fato de não ocorrer o casamento das características em nós adjacentes.

Em cada nó de origem uma aplicação, que também registra um conjunto de características aleatórias, foi desenvolvida para atuar como remetente das mensagens. Uma mensagem é enviada para a rede em intervalos de 1 segundo.

Para a análise do recebimento das mensagens, a aplicação de envio escreve a mensagem recebida em uma porta serial associada a um arquivo no sistema operacional que está executando as máquinas virtuais. Dessa forma, é possível saber se um nó recebeu uma mensagem observando as modificações dos arquivos associados às portas seriais de cada uma das máquinas.

A Tabela - 5.1 apresenta o resultado de cinco execuções do experimento. Em cada execução os processos de recebimento foram terminados e iniciados novamente de modo que gerassem um novo prefixo.

Os resultados obtidos se justificam com base na natureza probabilística do algoritmo de roteamento. Com um maior número de nós, a probabilidade do repasse de mensagens ocorrer também é maior. A implementação proposta também possibilita que a probabilidade também aumente quando mais aplicações são executadas no mesmo nó. No caso de existirem vários programas em execução no mesmo nó, vários prefixos diferentes são registrados. Assim, ao receber uma mensagem, o algoritmo de casamento de prefixos irá realizar a avaliação com todos os prefixos registrados. Caso algum tenha um campo que case com um dos campos da mensagem, ela será repassada para a rede.

	Execução 1		Execução 2		Execução 3		Execução 4		Execução 5	
Nó	% de alcance	Não alcançados	% de alcance	Não alcançados	% de alcance	Não alcançados	% de alcance	Não alcançados	% de alcance	Não alcançados
1	86	21, 22, 23	95	23	18	6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23	100		36	8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23
7	100		100		100		100		100	
10	100		86	6, 8, 23	100		100		72	1,2,3,4,6,8
14	100		59	1, 2, 3, 4, 5, 6, 7, 8, 9	72	1, 2, 3, 4, 6, 8	95	23	68	1, 2, 3, 4, 5, 6, 8
17	100		72	1, 2, 3, 4, 6, 8	100		81	1, 2, 3, 23	100	
20	100		100		86	1, 2, 23	86	15, 16, 17	100	
23	100		100		86	15, 16, 17	59	1, 2, 3, 4, 5, 6, 7, 8, 9	4	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22

Tabela 5.1: Alcance das mensagens enviadas a partir de nós selecionados

5.3 Avaliação do protocolo REPA_UEM

Esta seção apresenta os resultados obtidos nos experimentos de envio e recebimento massivo de mensagens. O objetivo do experimento é avaliar as diferenças no consumo de recursos computacionais das diferentes versões do protocolo.

5.3.1 Quantidade de mensagens processadas

Considera-se como mensagem processada uma mensagem que foi enviada para a aplicação em espaço de usuário. Devido ao tempo gasto no processamento das mensagens recebidas, o sistema não consegue entregar todas para as aplicações, muitas delas são simplesmente descartadas. O número de mensagens processadas foi coletado por meio da ferramenta *DTrace*. O código foi instrumentado com uma sonda estática que captura o momento exato em que uma mensagem é repassada para uma aplicação em espaço de usuário. A quantidade de repasses para as aplicações é contabilizada e recuperada por meio de um *script* em linguagem D (*DTrace scripts*).

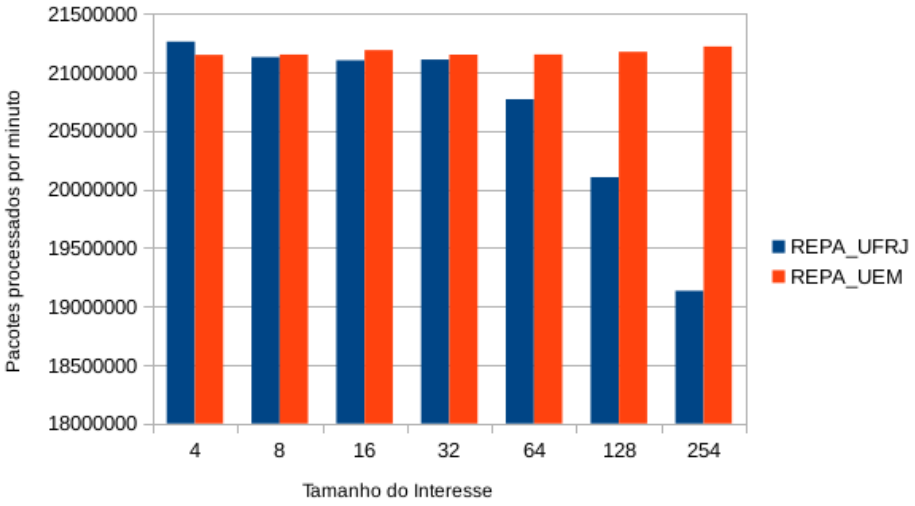


Figura 5.3: Quantidade de pacotes processados

É possível observar na Figura - 5.3 que a quantidade de mensagens do protocolo REPA_UFRJ processadas diminui conforme o tamanho do interesse aumenta. Essa quantidade se mantém aproximadamente constante para o protocolo REPA_UEM. Isso se deve ao tempo gasto durante o processamento do cabeçalho.

5.3.2 Quantidade de instruções de CPU

A quantidade de instruções de CPU executadas foi coletada durante o envio das mensagens. Para tal a ferramenta PMC foi utilizada. O valor coletado é a soma das instruções executadas por todos os processadores durante cada segundo da transmissão.

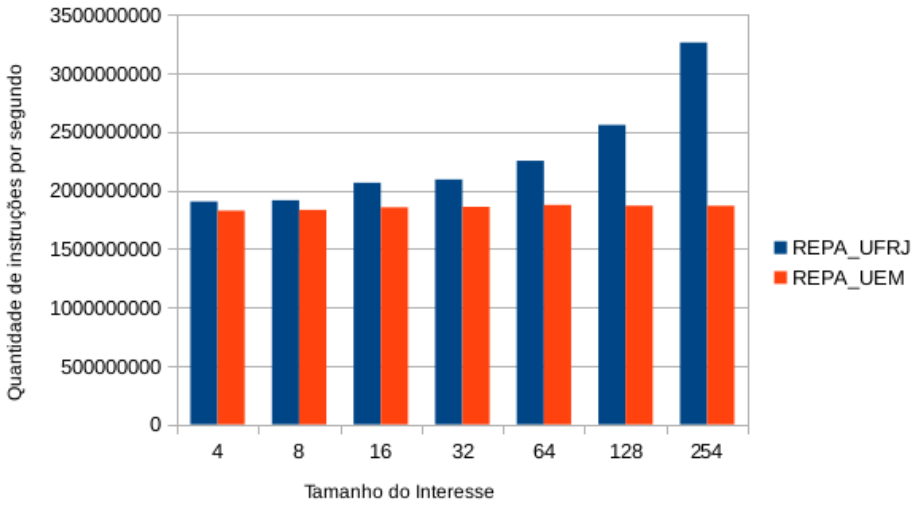


Figura 5.4: Quantidade de instruções de CPU executadas

A quantidade de instruções de CPU executadas por segundos durante a transmissão dos pacotes para cada tamanho de interesse pode ser observada na Figura - 5.4. Novamente pode-se observar que para o protocolo REPA_UEM os valores se mantêm próximos quando varia-se o tamanho do interesse e para o protocolo REPA_UFRJ a quantidade de instruções por segundo aumenta significativamente. No caso em que o interesse possui 254 caracteres, o ganho é de aproximadamente 42% quando o REPA_UEM é utilizado, uma vez que o protocolo REPA_UFRJ consome mais recursos de CPU durante o processamento de mensagens.

As estatísticas sobre a quantidade de instruções executadas foram obtidas com a ferramenta *pmcstat* em intervalos de 1 segundo. Com a execução da ferramenta *pmcstat* no modo de amostragem, foi observado que mais de 23% das instruções executadas pelo sistema são referentes a execução da rotina de comparação de interesses do protocolo REPA_UFRJ.

5.3.3 Tempo gasto no processamento de mensagens

O tempo médio gasto para o processamento de mensagens de cada protocolo foi coletado com a ferramenta *DTrace*. Para isso, as rotinas que processam e repassam cada mensagem

para espaço de usuário foram instrumentadas dinamicamente com o *DTrace* e o tempo gasto foi coletado por meio de um *script* em linguagem D.

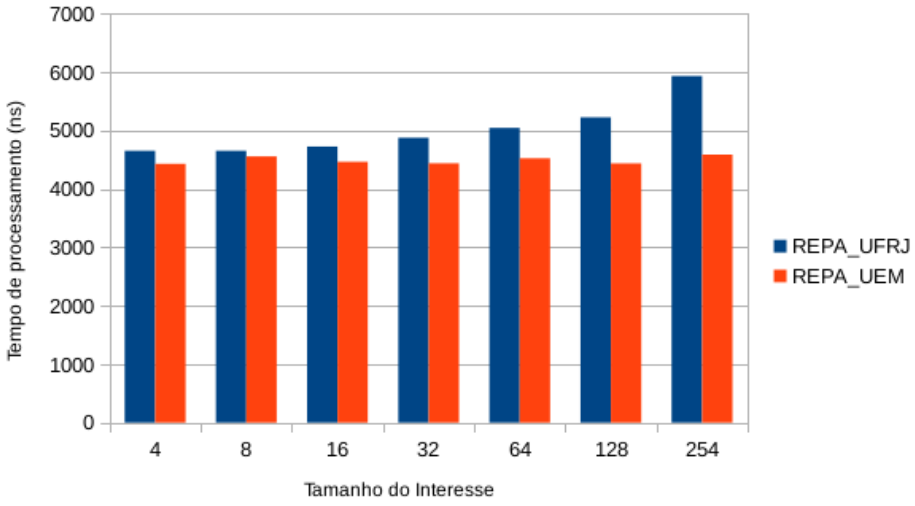


Figura 5.5: Tempo de execução da rotina de processamento das mensagens

Durante o recebimento das mensagens, duas rotinas do *kernel*, denominadas repa_uem_input e repa_ufrj_input, são invocadas para realizar o processamento. Os experimentos com a ferramenta *DTrace* mostraram que o tempo médio de execução é ligeiramente maior para a rotina repa_ufrj_input, como pode ser observado na Figura 5.5. Isso se deve a fatores como o maior tamanho dos pacotes, a necessidade da extração do interesse do cabeçalho e a comparação dos interesses das mensagens com os dos registrados pelos programas dos usuários.

Pode-se observar que os tempos de processamento das mensagens do protocolo REPA_UEM são próximos e para as mensagens do protocolo REPA_UFRJ o tempo aumenta conforme o tamanho do interesse. O tempo é apresentado em nanosegundos e no caso em que o interesse é utilizado com tamanho de 254 caracteres a diferença alcança aproximadamente 22%, significando que o protocolo REPA_UFRJ leva mais tempo para processar cada mensagem recebida.

5.4 Suporte à Criptografia

O suporte à criptografia foi implementado com o uso de APIs do próprio sistema operacional FreeBSD. A API utilizada implementa o algoritmo Rijndael (utilizado no padrão AES) com os modos de cifra ECB, CBC e CFB-1 e suporte a chaves de 128, 192 e 256 bits.

Para exemplificar como o suporte à criptografia pode ser utilizado, um centro de distribuição de chaves foi implementado sobre a Radnet.

5.4.1 Implementação de um Centro de Distribuição de Chaves sobre a Radnet

O KDC apresentado a seguir é um exemplo de como a rede e o suporte à criptografia podem ser utilizados. A implementação não tem o objetivo de ser utilizada na prática e serve apenas como prova de conceito, uma vez que não possui características de segurança essenciais como por exemplo meios de os cliente validarem a autenticidade do KDC.

A aplicação do KDC utiliza uma base de dados de chaves públicas de usuários da rede. Um usuário pode ser uma aplicação específica, um nó físico ou uma pessoa. Uma vez que a chave pública é armazenada juntamente com a identificação do usuário, o KDC pode enviar a chave simétrica da rede de maneira segura. O padrão de criptografia assimétrica utilizado na implementação é o RSA (Rivest et al., 1978).

Para solicitar uma chave, a aplicação deve enviar uma mensagem especial para a rede utilizando o interesse "*keyrequest*". No corpo da mensagem, o nó deve enviar apenas a identificação associada à sua chave pública, por exemplo, uma cadeia de caracteres contendo o nome do usuário. A resposta do KDC será enviada em uma mensagem com o interesse igual ao identificador do nó. Dessa forma, após o envio da solicitação, o nó deverá aguardar a resposta em outro *socket* no qual o interesse é o seu próprio identificador.

Em ambientes reais, a mensagem de solicitação de chave pode não chegar ao destino devido a perdas de pacotes. Nesses casos, a chamada de sistema *poll()* pode ser utilizada para monitorar o recebimento da mensagem no segundo *socket*. Tal chamada de sistema pode ser parametrizada com um valor de *timeout*. Dessa forma, caso a resposta não seja recebida em um determinado intervalo de tempo a solicitação pode ser reenviada.

Quando o KDC recebe uma mensagem com o interesse "*keyrequest*", a identificação do nó é retirada da mensagem e uma busca no banco é executada. Se a chave associada ao identificador for encontrada, o KDC irá criar um *socket* e usar o identificador como o valor do interesse. Em seguida, ele irá cifrar a chave simétrica previamente gerada e enviá-la para a rede por meio deste *socket*. Nesta mesma mensagem também é enviada a configuração do algoritmo de criptografia simétrica da rede, como o tamanho da chave e o modo de cifra utilizado.

A mensagem de resposta do KDC pode ser observada na Tabela - 5.2. O campo "Algoritmo" informa qual algoritmo de criptografia simétrica deve ser utilizado pelo nó, nesta implementação apenas o padrão AES é suportado. O campo "Tamanho da chave"

informa qual tamanho da chave, em *bytes*, deve ser utilizado. Os valores possíveis são 16 (128 bits) e 32 (256 bits). O campo "Modo de cifra" informa qual modo de cifra deve ser utilizado. Neste trabalho apenas os modos CBC e ECB são suportados devido a limitações na API utilizada. O campo "Algoritmo de autenticação" informa qual algoritmo de autenticação de mensagens deve ser utilizado. Neste trabalho, o suporte a autenticação de mensagens não foi implementado.

Algoritmo	Tamanho da chave	Modo de cifra	Algoritmo de autenticação
Identificação do usuário			
Chave criptografada			

Tabela 5.2: Mensagem de resposta do KDC

Devido ao envio probabilístico de mensagens, pode ocorrer um caso em que uma requisição de chave não alcance o KDC ou ainda uma resposta do KDC não alcance o nó que realizou a solicitação. Para aumentar as chances de a troca de chave ser bem sucedida, foi implementado um tratamento especial para mensagens de solicitação e resposta de chave. O campo "O" (*Others*) do cabeçalho da mensagem, como pode ser observado na Figura - 2.2, foi utilizado para identificar esse tipo de mensagem. Quando uma mensagem de solicitação ou resposta é recebida o sistema simplesmente a repassa para a rede sem realizar a verificação de casamento de prefixo.

Como mencionado anteriormente, o KDC apresentado neste trabalho é utilizado apenas como um exemplo de como o suporte à criptografia pode ser utilizado. A implementação do KDC proposta não garante a autenticidade do KDC, permitindo que um nó mal intencionado envie chaves para a rede e consiga decodificar o tráfego. Uma possível solução para esse problema é a implementação de verificação da assinatura digital do KDC. Dessa forma, além de codificar a chave que será enviada para os nós, o KDC também poderá assinar digitalmente a mensagem com sua chave privada. Assim, torna-se possível verificar a autenticidade das respostas com a chave pública do KDC.

5.5 Considerações finais

Este Capítulo apresentou os resultados obtidos no trabalho. Os experimentos mostraram que o cabeçalho simplificado proposto proporcionou uma redução considerável no consumo de recursos computacionais conforme o tamanho da cadeia de caracteres do interesse é aumentado. A redução do consumo de recursos pode estar diretamente relacionada com a redução do consumo de energia elétrica, o que é um fator importante principalmente

quando os elementos da rede são sistemas embarcados que dependem de baterias. O suporte à criptografia foi validado por meio do desenvolvimento de um Centro de Distribuição de Chaves (KDC). O KDC proposto se conecta à Radnet e aguarda solicitações de chaves por parte dos demais nós. No momento em que um nó se conecta à rede, ele solicita uma chave simétrica em *broadcast*. Quando o KDC recebe a solicitação, a chave simétrica é criptografada com a chave pública do usuário e enviada para a rede novamente. Quando o nó recebe a mensagem com a chave simétrica, ela é decodificada e utilizada na codificação do tráfego. O próximo Capítulo apresenta as conclusões do trabalho e as sugestões de trabalhos futuros.

6

Conclusões e Trabalhos Futuros

O uso de protocolos baseados em interesses pode ser interessante em diversos cenários, desde os mais usuais, como para o anúncio de promoções em feiras, até em situações de calamidade pública, nas quais pessoas podem ficar incomunicáveis devido à interrupção do fornecimento de energia elétrica. Nesses cenários, pessoas podem registrar e enviar mensagens para outras com os mesmos interesses e permitir que seus aparelhos móveis sejam utilizados para a entrega de mensagens de outras pessoas.

O roteamento baseado nas características do usuário pode reduzir o tráfego na rede, uma vez que é probabilístico. Um problema inerente desse modelo é o fato de que se houver poucos usuários na rede, a probabilidade de mensagens serem entregues tende a ser pequena.

Um ponto importante na utilização de sistemas embarcados móveis para a transmissão e roteamento de mensagens é o consumo de energia. Como apresentado, a utilização de campos simples nos cabeçalhos, como valores numéricos, e a implementação em nível de *kernel* podem colaborar para a redução da utilização de recursos computacionais, o que pode estar diretamente relacionado com a redução no consumo de energia, uma vez que as decisões de roteamento podem ser tomadas sem a necessidade da cópia das mensagens para uma aplicação em espaço de usuário. O estudo do consumo de energia relacionado ao uso dos protocolos está além do escopo deste trabalho.

Outro ponto importante em alguns cenários é a capacidade de maximizar o envio de mensagens. Em redes em que a quantidade de usuários é muito grande o tráfego pode prejudicar a entrega de mensagens. Com a redução dos recursos computacionais necessários para o processamento, mais mensagens podem ser entregues e roteadas.

Este trabalho teve como objetivos a implementação do modelo de rede endereçada por interesse em nível de *kernel* no sistema operacional FreeBSD, a proposta de um novo formato para o cabeçalho utilizado nas mensagens, em que o interesse é armazenado em formato numérico ao invés de ser em uma cadeias de caracteres, a implementação de um suporte à criptografia de mensagens e o desenvolvimento de ferramentas para o monitoramento e configuração da Radnet.

Foram executados experimentos baseados no envio de pacotes com interesses de tamanhos variados entre dois computadores ligados por uma rede *Gigabit Ethernet*. Os experimentos mostraram que a utilização dos recursos computacionais foi menor com o novo cabeçalho, o que proporcionou um aumento na quantidade de mensagens processadas por intervalo de tempo.

A implementação permitiu o desenvolvimento de aplicativos sobre e Radnet por meio da API de *sockets* do sistema operacional, podendo proporcionar a redução do *overhead* durante o processamento de mensagens.

Os experimentos mostraram que o cabeçalho proposto pode reduzir consideravelmente a utilização de CPU conforme o tamanho do campo de interesse aumenta, sendo obtidos ganhos de até 42% em relação à quantidade de instruções executadas por segundo, aumentar a taxa de processamento de mensagens por minuto em aproximadamente 10% e reduzir o tempo de processamento de cada mensagem em até 22%.

O funcionamento da implementação proposta foi avaliado por meio de um experimento em uma rede de máquinas virtuais. O cenário criado no experimento possibilitou verificar que o roteamento probabilístico de fato está ocorrendo e que a implementação está funcional.

O suporte à criptografia foi implementado por meio do desenvolvimento de um centro de distribuição de chaves. A implementação proposta permite ativar o suporte à criptografia de tráfego em um *socket* por meio da chamada de sistema *setsockopt*. A partir do momento em que a chamada de sistema é invocada com a configuração do algoritmo de criptografia, toda mensagem enviada pelo *socket* em questão será criptografada de modo transparente. O ambiente de distribuição de chaves apresenta uma ideia de como a criptografia pode ser utilizada de modo automático na rede.

A partir das contribuições deste trabalho, alguns pontos podem ser definidos para desenvolvimento em trabalhos futuros:

- Desenvolvimento de um mecanismo de priorização de tráfego. Em algumas situações pode ser necessário que os equipamentos conectados à Radnet priorizem o tráfego de mensagens com um determinado interesse, por exemplo, em situações de emergência;

- O suporte à criptografia proposto no trabalho é básico, não contemplando modos de cifra que suportem autenticação das mensagens (AEAD - *Authenticated Encryption with Associated Data*). A implementação de modos de cifra que suportem autenticação, como o GCM (*Galois/Counter Mode*) ou mecanismos baseados em *hash*, como o algoritmo SHA (*Secure Hash Algorithm*), podem contribuir com o aumento da segurança na troca de mensagens;

- O centro de distribuição de chaves proposto possui alguns pontos em que a segurança pode ser aperfeiçoada (como mencionado na Seção 5.4). Práticas mais seguras de troca de chaves podem ser implementadas, como por exemplo, a autenticação das mensagens enviadas pelo KDC;

- O envio de mensagens da Radnet não podem trafegar em redes IP. Um possível trabalho futuro seria a implementação de um aplicativo que possa encapsular mensagens em pacotes IP e roteá-los para redes Radnet remotas por meio de redes IP, como a própria Internet.

REFERÊNCIAS

AKYILDIZ, I. F.; WANG, X.; WANG, W. Wireless mesh networks: a survey. *Computer Networks*, v. 47, n. 4, p. 445 – 487, 2005.
Disponível em `http://www.sciencedirect.com/science/article/pii/S1389128604003457`

BOUKERCHE, A.; TURGUT, B.; AYDIN, N.; AHMAD, M. Z.; BÖLÖNI, L.; TURGUT, D. Routing protocols in ad hoc networks: A survey. *Computer Networks*, v. 55, n. 13, p. 3032 – 3080, 2011.
Disponível em `http://www.sciencedirect.com/science/article/pii/S1389128611001654`

CANTRILL, B. M.; SHAPIRO, M. W.; LEVENTHAL, A. H. Dynamic instrumentation of production systems. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, Berkeley, CA, USA: USENIX Association, 2004, p. 2–2 (*ATEC '04*,).
Disponível em `http://dl.acm.org/citation.cfm?id=1247415.1247417`

CONTI, M.; GIORDANO, S. Multihop ad hoc networking: The theory. *Communications Magazine, IEEE*, v. 45, n. 4, p. 78–86, 2007.

DAEMEN, J.; RIJMEN, V. Aes proposal: Rijndael. 1999.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. *IEEE Transactions on Information Theory*, v. 22, n. 6, p. 644–654, 1976.

DUTRA, D. L. C.; MORAES, H. F.; AMORIM, C. L. Radflow: An interest-centric task based dataflow runtime. In: *2015 International Symposium on Computer Architecture and High Performance Computing Workshops, SBAC-PAD Workshops, Florianópolis, Brazil, October 18-21, 2015*, 2015, p. 115–119.
Disponível em `http://dx.doi.org/10.1109/SBAC-PADW.2015.26`

Dutra, R. C. *Redes ad hoc centradas em interesses para ambientes móveis.* Doutorado, COPPE - UFRJ, 2012.

Dutra, R. C.; Moraes, H. F.; Amorim, C. L. *Active prefixes for mobile ad-hoc networks. technical report.* Relatório Técnico, Universidade Federal do Rio de Janeiro (UFRJ), 2011.

Dutra, R. C.; Moraes, H. F.; Amorim, C. L. Interest-centric mobile ad hoc networks. In: *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, 2012, p. 130–138.

Dutra, R. D. C.; Moraes, H. F. D.; Amorim, C. L. Repi: Rede de comunicação endereçada por interesses. In: *WP2P 2010: Anais do VI Workshop de Redes Dinâmicas e Sistemas P2P. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC2010)*, 2010.

Dworkin, M. J. *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac.* Relatório Técnico, Gaithersburg, MD, United States, 2007.

Ferguson, N.; Schneier, B. *Practical cryptography.* 1 ed. New York, NY, USA: John Wiley & Sons, Inc., 2003.

Fowler, G.; Vo, P.; and, L. C. N. Fnv hash function. [*On-line*], acessado em 01/03/2017.
Disponível em `http://www.isthe.com/chongo/tech/comp/fnv/index.html`

FreeBSD Performance measurement with performance monitoring hardware. [*On-line*], acessado em 03/05/2015.
Disponível em `https://www.freebsd.org/cgi/man.cgi?query=pmcstat&sektion=8`

FreeBSD Freebsd architecure handbook. the sysinit framework. [*On-line*], acessado em 06/07/2016.
Disponível em `https://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/sysinit.html`

Goncalves, F. B.; França, F. M. G.; Amorim, C. L. Radnet-ve: An interest-centric mobile ad hoc network for vehicular environments. *CoRR*, v. abs/1604.00589, 2016.
Disponível em `http://arxiv.org/abs/1604.00589`

GONDOLFO, D. E. *Implementação da radnet no sistema operacional freebsd.* Monografia, Universidade Estadual de maringá, 2014.

GRANJA, R. S. *Protocolos para redes de comunicação ad-hoc endereçadas por interesses.* Mestrado, COPPE - UFRJ, 2010.

GRANJA, R. S.; DUTRA, R. C.; MORAES, H. F.; AMORIM, C. L. Samcra: Um sistema para avaliação experimental de redes ad hoc. In: *WXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2010.

HAAS, Z. J.; HALPERN, J. Y.; LI, L. Gossip-based ad hoc routing. 2002, p. 1707–1716.

HILL, J.; SZEWCZYK, R.; WOO, A.; HOLLAR, S.; CULLER, D.; PISTER, K. System architecture directions for networked sensors. *SIGARCH Comput. Archit. News*, v. 28, n. 5, p. 93–104, 2000.
Disponível em `http://doi.acm.org/10.1145/378995.379006`

IEEE 802.15.4 standard, 2003. [*On-line*], acessado em 18/11/2016.
Disponível em `http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf`

JACOBSON, V.; SMETTERS, D. K.; THORNTON, J. D.; PLASS, M. F.; BRIGGS, N. H.; BRAYNARD, R. L. Networking named content. In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, New York, NY, USA: ACM, 2009, p. 1–12 (*CoNEXT '09*,).
Disponível em `http://doi.acm.org/10.1145/1658939.1658941`

JUNIOR, R. J. J. Jenkins hash function. [*On-line*], acessado em 01/03/2017.
Disponível em `http://www.burtleburtle.net/bob/hash/doobs.html`

KONG, J. *Freebsd device drivers: a guide for the intrepid.* No Starch Press, 2012.

KRIFA, A.; SBAI, M.; BARAKAT, C.; TURLETTI, T. Bithoc: A content sharing application for wireless ad hoc networks. In: *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, 2009, p. 1–3.

KVM Kernel-based virtual machine. [*On-line*], acessado em 18/11/2016.
Disponível em `http://www.linux-kvm.org/page/Main_Page`

MCKUSICK, M. K.; NEVILLE-NEIL, G. V.; WATSON, R. N. M. *The design and implementation of the freebsd operating system.* 2 ed. Addison Wesley, 2014.

MEISEL, M.; PAPPAS, V.; ZHANG, L. Ad hoc networking via named data. In: *Proceedings of the Fifth ACM International Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '10, New York, NY, USA: ACM, 2010, p. 3–8 (*MobiArch '10*,). Disponível em `http://doi.acm.org/10.1145/1859983.1859986`

MORAES, H.; BENITEZ, N.; DUTRA, R.; AMORIM, C. On developing interest-centric applications for ad hoc networks. In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, 2012, p. 1–3.

MORAES, H. F. *Desenvolvimento e avaliação de um protocolo peer-to-peer para aplicações da internet orientadas a interesse*. Mestrado, COPPE - UFRJ, 2011.

NIST Federal information processing standards publication (FIPS 197). Advanced Encryption Standard (AES). 2001.

NS-3 *Ns-3 simulator*. 2014.
Disponível em `http://www.nsnam.org/`

OPENSSL OpenSSL: Cryptography and ssl/tls toolkit. [*On-line*], acessado em 12/10/2015.
Disponível em `www.openssl.org`

PERKINS, C.; BELDING-ROYER, E.; DAS, S. Ad hoc on-demand distance vector (aodv) routing. 2003.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, v. 21, n. 2, p. 120–126, 1978.
Disponível em `http://doi.acm.org/10.1145/359340.359342`

SOMMER, C.; GERMAN, R.; DRESSLER, F. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing*, v. 10, n. 1, p. 3–15, 2011.

STEVENS, W. R.; WRIGHT, G. R. *Tcp/ip illustrated (vol. 2): The implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

TCPDUMP/LIBPCAP *Command-line packet analyzer*. 2014.
Disponível em `http://www.tcpdump.org/`

TMOTE Tmote sky. [*On-line*], acessado em 18/11/2016.
Disponível em `http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf`

TOOR, Y.; MUHLETHALER, P.; LAOUITI, A. Vehicle ad hoc networks: applications and related technical issues. *Communications Surveys Tutorials, IEEE*, v. 10, n. 3, p. 74–88, 2008.

WIRESHARK *Wireshark: network protocol analyzer*. 2014.
Disponível em `https://www.wireshark.org/`

Apêndice - Download do código e compilação do sistema

Todo o código deste trabalho está disponível para *download* por meio do endereço https://github.com/daniloegea/freebsd-radnet. O código pode ser baixado por meio do comando:

```
git clone https://github.com/daniloegea/freebsd-radnet.git
```

No arquivo de configuração do *kernel* deve ser inserida a seguinte opção:

```
options         RADNET
```

Um arquivo de configuração do *kernel* chamado RADNET já está disponível com esta opção incluída. Para compilar o *kernel* é necessário estar dentro do diretório onde o *download* do código foi feito e executar os seguintes comandos:

```
make KERNCONF=RADNET buildkernel
make KERNCONF=RADNET installkernel
```

Para compilar o restante do sistema (programas do espaço de usuário), os seguintes comando devem ser utilizados:

```
make includes
make buildworld
make installworld
```

A geração da imagem para a plataforma *Raspberry* pode ser gerada com o auxílio da ferramenta *crochet*, disponível em https://github.com/kientzle/crochet-freebsd. Para isso, faça o *download* da ferramenta com o comando:

```
git clone https://github.com/kientzle/crochet-freebsd.git
```

Dentro do diretório criado, crie uma cópia do arquivo *config.sh.sample* e modifique as configurações de acordo com as suas necessidades. Para criação da imagem, use o seguinte comando:

```
sh crochet.sh -c config_radnet.sh
```

Caso a instalação de alguma dependência seja necessária, basta seguir as instruções apresentadas pela ferramenta *crochet*. A gravação da imagem gerada no cartão de memória pode ser feita com o seguinte comando:

```
dd if=imagem_gerada.img of=/dev/da0 bs=1M
```

Todo o desenvolvimento foi realizado na versão 10.1 do FreeBSD. É recomendada a utilização desta mesma versão para a execução dos passos acima.