UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSTA E IMPLEMENTAÇÃO DE UMA MIB PARA O PROTOCOLO OLSR

VINÍCIUS MAIA PACHECO

ORIENTADOR: RICARDO STACIARINI PUTTINI

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGENE.DM – 317 A/07

BRASÍLIA / DF: NOVEMBRO / 2007

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSTA E IMPLEMENTAÇÃO DE UMA MIB PARA O PROTOCOLO OLSR

VINÍCIUS MAIA PACHECO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

RICARDO STACIARINI PUTTINI, Doutor, ENE/FT/UnB
(ORIENTADOR)

RAFAEL TIMÓTEO DE SOUSA JR., Doutor, ENE/FT/UnB
(EXAMINADOR INTERNO)

JACIR LUIZ BORDIM, Doutor, CIC/UnB
(EXAMINADOR EXTERNO)

ANDERSON CLAYTON ALVES NASCIMENTO, Doutor, ENE/FT/UnB
(SUPLENTE)

DATA: BRASÍLIA/DF, 26 DE NOVEMBRO DE 2007.

FICHA CATALOGRÁFICA

PACHECO, VINÍCIUS MAIA
Proposta e Implementação de uma MIB para o Protocolo OLSR [Distrito Federal] 2007, xxii, 141p., 297 mm
(ENE/FT/UnB, Mestre, Engenharia Elétrica, 2007).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. OLSR 2. MIB
3. SNMP

I. ENE/FT/UnB. II. Proposta e Implementação de uma MIB para o Protocolo OLSR

REFERÊNCIA BIBLIOGRÁFICA

PACHECO, V. M. (2007). Proposta e Implementação de uma MIB para o Protocolo OLSR. Dissertação de Mestrado, Publicação 317 A/07, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 141p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Vinícius Maia Pacheco
TÍTULO DA DISSERTAÇÃO: Prosposta e Implementação de uma MIB para o Protocolo OLSR.
GRAU/ANO: Mestre/2007.

Vinícius Maia Pacheco
SHIS Qi 26 conjunto 01 casa 08
CEP 71670-010 – Brasília – DF - Brasil

à minha família e à Maria Carla.

AGRADECIMENTOS

À minha família por ter suportado emocionalmente e financeiramente minha educação.
À Maria Carla por ter inspirado e motivado a decisão de aceitar este desafio.
Ao meu orientador Prof. Dr. Ricardo Staciarini Puttini, pelo constante apoio, incentivo, dedicação e amizade essenciais para o desenvolvimento deste trabalho e para o meu desenvolvimento como pesquisador.
Ao Prof. Dr. Rafael Timóteo de Sousa Júnior, do Curso de Engenharia de Redes de Comunicação - Departamento de Engenharia Elétrica.
Aos bolsistas do Laboratório de Engenharia de Redes de Comunicação – LabRedes da Universidade de Brasília, pelas conversas enriquecedoras, ajuda em diversos aspectos, colaboração e amizade.
A todos, os meus sinceros agradecimentos.

RESUMO

PROPOSTA E IMPLEMENTAÇÃO DE UMA MIB PARA O PROTOCOLO OLSR

Autor: Vinícius Maia Pacheco

Orientador: Ricardo Staciarini Puttini

Programa de Pós-graduação em Engenharia Elétrica

Brasília, Novembro de 2007

Dentre todos protocolos de roteamento destinados às MANET – *Mobile Ad Hoc Networks*, destaca-se o OLSR – *Optimized Link State Routing*. Ele é classificado como pró-ativo e de estado de enlaces. A característica de pró-atividade, em contraposição à reatividade, dita que tais protocolos disponham de todas as rotas necessárias antes de serem demandados. Já os protocolos de estado de enlaces, diferentemente dos de vetor-distância, compilam a informação recebida de toda a rede, não só dos vizinhos, e calculam individualmente em cada nó sua tabela de rotas.

O OLSR como todo protocolo de roteamento impõe, também, a necessidade de ser gerenciado. Para isso é importante tanto coletar as informações e estatísticas relativas ao seu funcionamento, quanto poder, em reposta à análise de tais dados, alterar e ajustar a operação do protocolo. Foi, portanto, a percepção de tais requisitos de administração que motivou o trabalho desta dissertação.

Assim, foram propostas uma MIB – *Management Information Base* – e uma respectiva implementação para que, juntas, pudessem oferecer uma estrutura de gerenciamento. A OLSR MIB é um documento que compila um modelo de informação traduzindo a essência do protocolo e a implementação, por sua vez, é o programa que materializa essas definições, conseqüentemente, permitindo o gerenciamento propriamente dito do OLSR. Esta dissertação apresentará estas duas entidades e detalhará como elas disponibilizam o ambiente de administração para o protocolo OLSR.

ABSTRACT

PROPOSAL AND IMPLEMENTATION OF A MIB FOR THE OLSR PROTOCOL

Author: Vinícius Maia Pacheco

Supervisor: Ricardo Staciarini Puttini

Programa de Pós-graduação em Engenharia Elétrica

Brasília, November of 2007

Amongst the MANET – Mobile Ad Hoc Networks – routing protocols, the OLSR – Optimized Link State Routing – stands out. It is classified as proactive and as link-state. The proactivity characteristic, in opposition to the reactivity, states that such protocols have at their disposal all routes prior to being demanded. On the other hand, link-state routing protocols, differently from the distance-vector ones, compile the information gathered from the entire network, not only from the neighbors, and calculate individually in each node its routing table.

The OLSR as all routing protocols impose, also, the necessity of being managed. To do that, it is important to not only gather the information and statistics related to its operation, but also to be able, in response to the analysis of such data, to alter and adjust the way the protocol works. It was, hence, the perception of such administration requisites that motivated the work in this dissertation.

Thus, a MIB – Management Information Base – and a respective implementation were proposed in order to, together, offer a management framework for the OLSR protocol. The OLSR MIB is a document that compiles an information model translating the essence of the protocol and its implementation is the program that materializes these definitions, consequently, allowing the management per say of the OLSR. This dissertation will present these two entities and will detail how they offer the administration environment for the OLSR protocol.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1. INTRODUÇÃO

Com a evolução das tecnologias de comunicação, novos segmentos da área de transmissão sem fio estão experimentando crescente uso e aplicação. Particularmente, as redes móveis *ad hoc* (MANET – *Mobile Ad Hoc Networks*) conquistaram seu nicho oferecendo conectividade para ambientes onde a infra-estrutura fixa é ausente. Dinamicidade e aleatoriedade são as características principais dessas redes, e para acomodar, então, esse tipo de operação, é necessário incorporar a capacidade de roteamento em seus elementos móveis.

Dentre os mais importantes protocolos de roteamento para redes móveis *ad hoc* se destaca o OLSR (*Optimized Link State Routing*). Simples e eficiente, ele controla a difusão de rotas em uma dada MANET. Assim, para incorporar este protocolo em um ambiente desejado, uma implementação do mesmo é necessária. Uma implementação é um programa que provê para o dispositivo a utilizando a habilidade de se comportar de acordo com o padrão escolhido. Existem, atualmente, um número de implementações do protocolo OLSR disponíveis para a comunidade, e uma das mais difundidas e base para a estrutura de gerenciamento para o protocolo OLSR proposta aqui é a *OLSR Daemon*, ou simplesmente *OLSRD* [2], disponível em www.olsr.org.

Uma vez que a implementação do OLSR esteja instalada e iniciada, os nós na rede *ad hoc* começam a propagar suas rotas, e depois que a convergência é atingida, todos os dispositivos podem se intercomunicar.

Um problema, entretanto, surge quando a necessidade de coletar informação ou mesmo de alterar os parâmetros de roteamento se impõe. Esta necessidade é regularmente associada com a importância de construção de estatísticas ou até de otimizar e proteger a implementação enquanto ela esteja operando.

Em relação à importância de coletar dados de roteamento, é interessante estabelecer que todos os protocolos de roteamento, enquanto funcionando, produzem informação a respeito de sua operação. Seja o estado de seus enlaces ou as características de seus vizinhos, por exemplo, essa informação é de grande valor para qualquer indivíduo ou aplicação com o objetivo de monitorar o ambiente de roteamento em questão. Acesso a este conhecimento é, todavia, nem sempre trivial.

Somando-se a isso, uma implementação de um protocolo de roteamento freqüentemente

faz uso de um arquivo de configuração onde guarda os parâmetros que ditarão o seu comportamento. Dessa forma, quando o protocolo é iniciado, o arquivo é lido e o programa irá operar respeitando tais valores. Contudo, é algumas vezes atraente ou mesmo necessário modificar essas diretivas sem ter que interromper a execução da implementação, alterar o arquivo de configuração e iniciar o programa novamente.

Aplicações como gerentes de rede (NMS – *Network Management Systems*) e detectores de intrusão (IDS – *Intrusion Detection Systems*), por exemplo, têm uma significativa parte de suas operações baseadas tanto no uso da habilidade de coletar informações quanto na eventual alteração de certos parâmetros..

É nesse contexto que a estrutura de gerenciamento para o protocolo OLSR proposta nesta dissertação está posicionada. Para prover um ambiente para a monitoração e controle do protocolo de roteamento, a estrutura é composta de duas principais contribuições: uma OLSR MIB e sua respectiva implementação.

A OLSR MIB é um documento do tipo base de informações (MIB – *Management Information Base*) modelado para o protocolo OLSR. Ele foi compilado especificando todos os parâmetros necessários para monitorar, controlar e proteger este protocolo.

Após a definição da OLSR MIB, uma implementação foi desenvolvida para permitir a manipulação propriamente dita dos parâmetros. Projetada como um subagente SNMP AgentX [24] acoplado ao *OLSRD*, essa implementação provê para as aplicações externas que se comunicam via protocolo SNMP a habilidade de consultar e controlar a operação do protocolo OLSR.

Este trabalho esta organizado da seguinte forma: no capítulo 2, o protocolo OLSR é descrito; no capítulo 3, o *OLSRD* é apresentado; no capítulo 4, os fundamentos do gerenciamento de redes são explorados; no capítulo 5, a estrutura propriamente dita para o gerenciamento do protocolo OLSR é detalhada; e, finalmente, no capítulo 6, a conclusão da dissertação é oferecida.

2. PROTOCOLO OLSR

Desenvolvido especificamente para as redes *ad hoc*, o protocolo OLSR (*Optimized Link State Routing*) – RFC 3626 [1] – é pró-ativo e baseado em estado de enlaces. Tais algoritmos mantêm localmente informações sobre a configuração da rede e distribuem regularmente esse conhecimento para os outros nós. Utilizando-se, destarte, desses dados, cada elemento calcula individualmente o melhor caminho para os destinos disponíveis. Cada roteador disporá, a qualquer momento, de rotas pré-estabelecidas e nenhum tempo de descoberta será despendido quando uma comunicação for imediatamente requisitada.

O conhecimento prévio de rotas vem, entretanto, com a difusão regular de informações sobre o estado da rede pelos nós. Esse comportamento provoca consumo de banda, um recurso que, na maioria das vezes, e especialmente no caso das redes móveis *ad hoc*, é considerado escasso. Para abordar esse problema de disseminação de tráfego de controle, o protocolo OLSR em sua RFC 3626 [1] propõe uma otimização conhecida como MPR (*Multipont Relay*), justificando a denominação do próprio protocolo como **Roteamento de Estado de Enlace Otimizado**.

A RFC 3626 [1] modulariza o protocolo em **funcionalidade núcleo** e **funções auxiliares**. A **funcionalidade núcleo** é justamente o conjunto operacional mínimo que qualquer entidade deve prover para implementar o protocolo OLSR. **Funções auxiliares** são definidas para introduzir, quando necessário, novas funcionalidades compatíveis com o núcleo operacional. Isso permite, então, que o protocolo acomode uma maior quantidade de ambientes e peculiaridades.

2.1. FUNCIONALIDADE NÚCLEO

A funcionalidade núcleo do OLSR especifica o comportamento de um nó equipado com interfaces participando de uma rede móvel *ad hoc* e utilizando o OLSR como protocolo de roteamento. Especificamente, a funcionalidade núcleo define os seguintes aspectos a serem atendidos:

- Endereçamento,
- Repositórios de Informação,
- Formato de Pacote,

- Mecanismo MPR,
- Mecanismo de Processamento e Encaminhamento de Pacotes,
- Múltiplas Interfaces,
- Descoberta de Vizinhança,
- Controle de Topologia,
- Calculo de Rotas.

2.1.1. Endereçamento

O OLSR utiliza um endereço IP como identificador único dos nós em uma rede. Como o OLSR foi desenvolvido para ser capaz de acomodar elementos com múltiplas interfaces de comunicação, cada nó deverá selecionar um de seus endereços IP para ser definido como seu **endereço principal**. Em nós com apenas uma interface, a escolha do **endereço principal** é simples e restringe-se à única possível. Já para roteadores com mais de uma interface OLSR, não é importante qual endereço seja eleito, porém, uma vez escolhido, o nó deverá sempre utilizá-lo como o **endereço principal**.

A RFC 3626 [1] dita que o OLSR pode ser utilizado tanto em ambientes com a versão 4 do protocolo IP (IPv4) [3], bem como naqueles em que se utiliza a versão 6 (IPv6) [4]. Em um contexto OLSR, as diferenças entre IPv4 e IPv6 são o tamanho dos endereços IP transmitidos nas mensagens de controle, o tamanho mínimo das mensagens e o endereço para ser utilizado como destino do tráfego de controle.

2.1.2. Repositórios de Informação

Seguindo o paradigma dos algoritmos de estado de enlace, o OLSR mantém localmente as informações sobre o comportamento do ambiente de roteamento. Esse conhecimento é obtido via tráfego de controle e está compilado nos **repositórios de informação**. A Tabela 2.1 mostra tais repositórios e oferece uma descrição dos mesmos.

Tabela 2.1 – Repositórios de Informação

REPOSITÓRIO DE INFORMAÇÃO	DESCRIÇÃO
BASE DE INFORMAÇÕES SOBRE ASSOCIAÇÕES DE MÚLTIPLAS INTERFACES	Este é o repositório de informações onde constam as associações de nós com mais de uma interface de rede participando do roteamento OLSR.
DETECÇÃO DE ENLACES: BASE DE INFORMAÇÕES DE ENLACES LOCAIS	Nesta base de informações são armazenados os enlaces referentes aos vizinhos deste nó.
• Conjunto de Enlaces	Esta entidade é utilizada para manter o estado de enlace com os vizinhos de um salto e é a única base de informações que não utiliza os endereços principais dos nós, uma vez que opera especificamente com enlaces de interface para interface. Vizinhos de um salto, neste contexto, são os roteadores com enlaces diretos para o nó em questão.
DETECÇÃO DE VIZINHOS: BASE DE INFORMAÇÕES DA VIZINHANÇA	Este repositório de informações guarda conhecimento sobre os vizinhos, vizinhos de dois saltos, MPRs e seletores MPR.

• Conjunto de Vizinhos	Aqui são controladas as informações referentes aos vizinhos de um salto, incluindo o endereço principal, simetria e valor de *willingness* obtidos. Simetria se refere ao estado do enlace em relação aos nós envolvidos. Se a informação pode transitar em ambos os sentidos, o enlace é considerado simétrico, e, se não, é denominado assimétrico. *Willingness* é um parâmetro que especifica a inclinação de um nó a encaminhar tráfego de rede para outros nós. É um número inteiro e varia de 0 (nunca) a 7 (sempre).
• Conjunto de Vizinhos de Dois Saltos	Aqui são mantidas as informações referentes aos vizinhos de dois saltos, ou seja, nós atingíveis através de vizinhos de um salto.
• Conjunto MPR	Um nó deve manter uma lista dos nós que ele selecionou como MPRs. Neste local reside tal informação. MPR (*Multipoint Relay*) é uma entidade utilizada na otimização da transmissão do tráfego de controle, e será propriamente descrita posteriormente.
• Conjunto de Seletores MPR	O Conjunto de Seletores MPR é uma lista contendo os roteadores que selecionaram o nó local como MPR.

BASE DE INFORMAÇÕES DE CONTROLE DE TOPOLOGIA	Este repositório contém informação sobre a topologia de rede recebida pelo ambiente de roteamento OLSR. Os dados neste repositório são utilizados para o cálculo de rotas.
BASE DE INFORMAÇÕES DE DUPLICATAS	Este repositório contém informação sobre as mensagens processadas e encaminhadas.

É importante explicitar que a maioria das informações residentes nos repositórios tem uma validade temporária. Dessa forma, dados que tiverem sua validade expirada são automaticamente eliminados das bases de informações. A determinação do tempo de expiração das entradas é feita com base nas mensagens de controle, onde o tempo de validade é transmitido junto com a própria respectiva informação.

2.1.3. Formato de Pacote

A RFC 3626 [1] define que toda a comunicação OLSR utilize um formato de pacote unificado. Isso permite uma maior facilidade em extender o protocolo sem quebrar a compatibilidade com versões anteriores. Mais ainda, essa adoção de um mesmo formato de pacote possibilita a transmissão de diferentes tipos de mensagens em uma mesma transmissão, reduzindo o desperdício de banda com informações de controle.

Pacotes OLSR são transmitidos via UDP [5] pela porta 698, designada pela IANA (*Internet Assigned Numbers Authority*) para ser utilizada em tais comunicações. A RFC também explicita que o tráfego OLSR deve ser feito via *broadcast*, embora um endereço específico não seja definido. Como endereços *broadcast* não existem em IPv6, assume-se que se utilizarão endereços *multicast* neste caso. O formato do pacote, considerando endereços IPv4, é exposto na Figura 2.1.

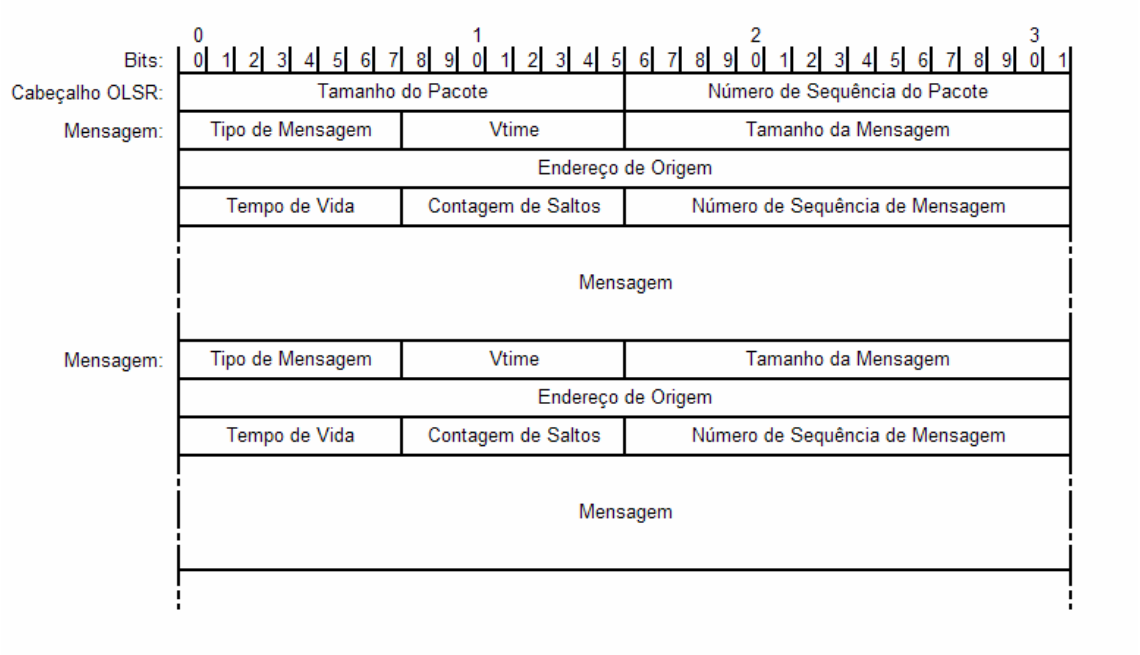

Figura 2.1 – Formato do Pacote OLSR

Todo o tráfego OLSR é feito com pacotes OLSR na forma da Figura 2.1. Assim, várias mensagens podem ser encaminhadas utilizando um único cabeçalho OLSR. Os campos do cabeçalho OLSR são:

- **Tamanho do Pacote**: o tamanho em *bytes* do pacote OLSR, incluindo o cabeçalho;
- **Número de Seqüência do Pacote**: toda vez que um pacote OLSR é transmitido por um nó, um número de seqüência é incrementado e registrado neste campo. Números de seqüência independentes são utilizados por cada interface do mesmo nó.

Cada uma das possíveis mensagens incluídas em um mesmo pacote OLSR deverá apresentar seu próprio cabeçalho. Os campos do cabeçalho de uma mensagem OLSR são:

- **Tipo de Mensagem**: este campo indica o tipo de mensagem sendo transmitida. Os tipos de mensagem reservados pelo OLSR estão no escopo de 0 até 127. Os tipos de 128 a 255 são considerados privados e podem ser utilizados para extensões customizadas do protocolo;
- **Vtime**: campo que indica por quanto tempo a informação contida na mensagem será considerada válida. O tempo de validade é representado pela sua mantissa (quatro maiores

bits do campo Vtime) e pelo seu expoente (quatro menores bits do campo Vtime) conforme a seguinte equação:

$Vtime = C * (1 + \frac{mantissa}{16}) * 2^{\exp oente}$, com C=0,0625 segundos, segundo proposto pela RFC;

- **Tamanho da Mensagem**: tamanho da mensagem em *bytes*, contados do começo do campo **Tipo de Mensagem** até o começo do próximo campo **Tipo de Mensagem** ou até o final do pacote se não houver mais mensagens no pacote;
- **Endereço de Origem**: campo contendo o endereço principal do nó que originou a mensagem;
- **Tempo de Vida**: o número máximo de saltos que esta mensagem deve realizar;
- **Contagem de Saltos**: a quantidade de saltos que a mensagem já sofreu;
- **Número de Seqüência de Mensagem**: toda vez que um nó gerar uma mensagem, um número de seqüência será atribuído a ela. Este número é incrementado de 1 a cada nova mensagem criada pelo roteador.

2.1.4. Mecanismo MPR

Em uma rede, *flooding* é o processo pelo qual um roteador encaminha uma informação recebida de um outro nó para todos os roteadores conectados a ele, com exceção daquele de onde veio a transmissão. Essa é uma conhecida técnica e vem sendo utilizada para difundir rapidamente informação de roteamento em redes amplas.

No caso do protocolo OLSR, devido às restrições de utilização de banda, as informações que permitirão o cálculo das rotas são disseminadas via otimização do processo de *flooding*, introduzindo a entidade chamada MPR (*Multipoint Relay*).

Nesse contexto, cada nó seleciona um conjunto de elementos de sua vizinhança de um salto simétrica para retransmitir suas mensagens. Esse conjunto de nós selecionados é chamado de conjunto MPR do nó. Vizinhos de um nó que não estão em seu conjunto MPR recebem e processam as mensagens de *broadcast*, mas não as retransmitem como os MPRs.

O conjunto MPR é escolhido de forma a satisfazer a seguinte regra: *o conjunto MPR de um nó é o subconjunto de vizinhos simétricos de um salto capaz de atingir com enlaces também*

simétricos de um salto todos os vizinhos de dois saltos do nó em questão. Obviamente, quanto menor o conjunto MPR, menos tráfego de controle é utilizado pelo protocolo de roteamento. A RFC 3626 [1] propõe uma heurística simples para a escolha do conjunto MPR; e encontrar o conjunto MPR ótimo provou ser um problema NP-completo [6].

A Figura 2.2 mostra a comparação de um *flooding* feito em uma rede sem fio sem o mecanismo MPR e outro feito no mesmo ambiente mas com a introdução das entidades MPR (em cor verde). Em ambos os casos, o nó central é que origina o processo. É possível, também, constatar que, ao contrário do que aconteceria em um *flooding* em rede cabeada, a retransmissão do *broadcast* feita pelos MPRs atinge de volta o nó central em ambos os casos; e no primeiro ambiente, tal fenômeno também ocorre com os demais nós. Essa peculiaridade do *flooding* ocorre, em ambos os casos, porque em ambientes sem fio a retransmissão é realizada pela mesma interface de rede que recebeu a mensagem. No *flooding* em rede cabeada, no entanto, as retransmissões nunca são feitas pela mesma interface de onde veio a mensagem.

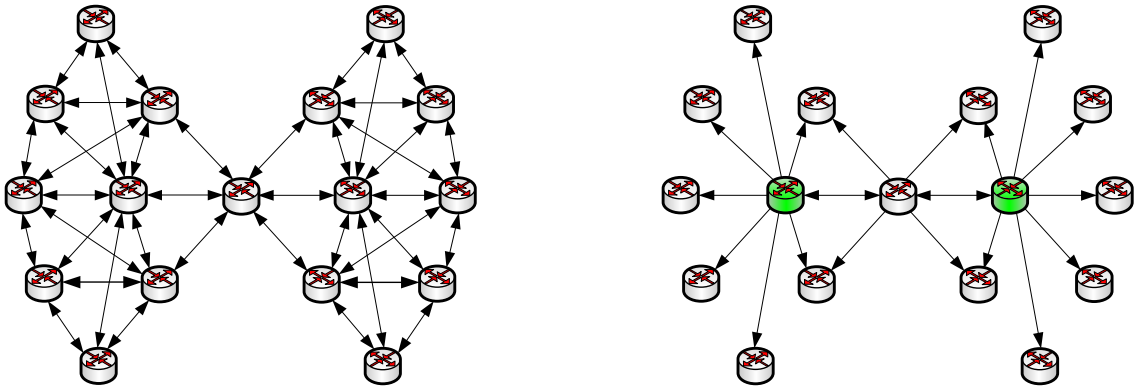

Figura 2.2 – Comparação do processo de *flooding* entre ambientes sem fio não utilizando e utilizando o mecanismo MPR.

2.1.5. Mecanismo de Processamento e Encaminhamento de Pacotes

Uma vez recebido um pacote OLSR, o nó examina o campo **Tipo de Mensagem** de cada mensagem contida no pacote para determinar o destino das mesmas. Para tipos de mensagem

conhecidos, estas serão processadas de acordo com as respectivas implementações locais de seus tipos. Já para mensagens de tipos desconhecidos, o protocolo OLSR determina que o mecanismo de encaminhamento padrão seja aplicado.

O mecanismo de encaminhamento padrão se utiliza do esquema MPR para operar e seu algoritmo de funcionamento pode ser delineado da seguinte maneira:

1. Se o enlace por onde a mensagem chegou for considerado assimétrico, a mensagem é descartada. Para determinar a simetria do enlace, a **base de informações de enlaces locais**, mais especificamente o **conjunto de enlaces**, é consultado.

2. Se o campo **Tempo de Vida** do cabeçalho da mensagem for zero, a mensagem é descartada.

3. Se a mensagem já tiver sido encaminhada, ela é descartada. Para determinar se a mensagem já foi encaminhada, a **base de informações de duplicatas** é consultada.

4. Se o último salto da mensagem tiver selecionado este nó como um MPR, a mensagem é encaminhada. Se não, a mensagem é descartada. Para determinar se o último nó elegeu este nó como MPR, a **base de informações da vizinhança**, mais especificamente o **conjunto de seletores MPR**, é consultado.

5. Se foi determinado que a mensagem deverá ser encaminhada, o campo **Tempo de Vida** é reduzido de 1 e o campo **Contagem de Saltos** é acrescido de 1. A mensagem é, então, transmitida via *broadcast*, utilizando o esquema MPR, em todas as interfaces.

Deve ser compreendido que processamento e encaminhamento de mensagens são duas atividades distintas. Processamento se refere à utilização do conteúdo da mensagem para tratamento do protocolo de roteamento. Já encaminhamento é a retransmissão de mensagens para os outros nós do ambiente.

Para mensagens de tipos conhecidos sempre haverá processamento da mensagem, e este processamento poderá ou não incluir o encaminhamento da mesma. No caso de mensagens de

tipos desconhecidos, como já foi especificado anteriormente, somente o algoritmo de encaminhamento será aplicado.

A RFC 3626 [1] define um conjunto de tipos de mensagem que devem obrigatoriamente ser implementadas, e, por conseguinte, processadas em todos os nós aderentes ao protocolo OLSR. Os tipos de mensagem obrigatórios para a funcionalidade núcleo do OLSR são:

- **Mensagens HELLO:** Realizam a tarefa de detecção de enlaces, detecção de vizinhos e sinalização MPR,
- **Mensagens TC:** Realizam a tarefa de declaração de topologia,
- **Mensagens MID:** Realizam a tarefa de declarar a presença de múltiplas interfaces em um nó.

Outro aspecto importante do mecanismo de encaminhamento é a medida de prevenção da sincronização das mensagens de controle. A sincronização pode ocorrer quando nós utilizando os mesmos valores de intervalos de emissão de mensagens transmitem nos mesmos exatos momentos. Isso pode causar colisões e, portanto, perda destas e das subseqüentes mensagens de controle.

Para evitar o sincronismo, o mecanismo de introdução do *jitter* de transmissão é utilizado. Neste paradigma, toda vez que uma mensagem for ser transmitida, um valor randômico variando de zero a *jitter* é subtraído do período de emissão da mensagem. Assim, os intervalos de transmissão serão todos diferentes, evitando, então, a sincronização do tráfego de controle.

2.1.6. Múltiplas Interfaces

Como já foi explicitado, nós em um ambiente de roteamento OLSR podem ter múltiplas interfaces. Para abarcar essa possibilidade, incorporando a informação de roteamento proveniente destas interfaces adicionais ao domínio OLSR, as mensagens MID (*Multiple Interface Declaration*) foram introduzidas.

Como o suporte às múltiplas interfaces também faz parte da funcionalidade núcleo do OLSR, o campo **Tipo de Mensagem** das mensagens MID é de conhecimento obrigatório em todos os

nós, tornando, então, mandatório o processamento delas em todos os roteadores.

O processamento das mensagens MID envolve a atualização da **base de informações sobre associações de múltiplas interfaces**. Quando, por conseguinte, um nó recebe uma mensagem MID, ele registra todos os endereços nela contidos como atrelados ao endereço principal do nó origem. O endereço principal é encontrado no campo **Endereço de Origem** do cabeçalho da própria mensagem MID. Também, quando uma rota para um nó tiver que ser adicionada, o OLSR irá adicionar rotas para todos os outros endereços declarados do nó utilizando o mesmo caminho.

Todos os nós rodando o protocolo OLSR em mais de uma interface devem gerar mensagens MID em intervalos regulares. Todos os nós OLSR, sem exceção, deverão processar tais mensagens MID como foi especificado e, também, deverão transmiti-las, após o processamento, para toda a rede OLSR utilizando o mecanismo de encaminhamento padrão. O formato da mensagem MID pode ser visualizado na Figura 2.3.

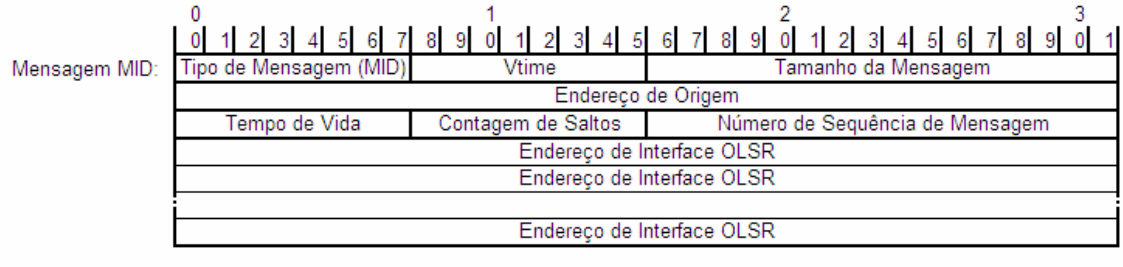

Figura 2.3 – Mensagem MID

2.1.7. Descoberta de Vizinhança

Por ser um protocolo pró-ativo e de estado de enlace, o conhecimento do ambiente de rede próximo ao nó é de fundamental importância para o funcionamento do protocolo OLSR.

O processo de descoberta de vizinhança concorre, nesse sentido, para a atualização da **base de informações da vizinhança**, incluindo o **conjunto de vizinhos**, o **conjunto de vizinhos de dois saltos**, o **conjunto MPR** e o **conjunto de seletores MPR**. E o mecanismo que rege a

manutenção de tal repositório é o processamento das mensagens HELLO.

As mensagens HELLO, como já estabelecido, compreendem um tipo de mensagem de implementação e conseqüente processamento obrigatórios em todos os nós OLSR. Seu formato pode ser visualizado na Figura 2.4 e seus campos são os seguintes:

- **Reservado:** deve ter todos os seus bits escritos como zeros para atender à especificação;
- **Htime:** especifica o período de emissão de mensagens HELLO utilizado pelo nó nesta interface particular. O valor está expresso pela sua mantissa (quatro maiores bits do campo Htime) e pelo seu expoente (quatro menores bits do campo Htime);
- **Willingness:** O campo especifica a inclinação do nó para encaminhar tráfego de rede em nome de outros nós. É um valor que varia entre 0 (nunca) até 7 (sempre);
- **Link Code:** este campo determina o estado do enlace entre este nó e os endereços declarados nos campos **Endereço da Interface Vizinha** subseqüentes. Valores de **Link Code** desconhecidos são silenciosamente descartados pelo nó;
- **Tamanho da Mensagem Link:** contém o tamanho da respectiva mensagem Link, contados em bytes e medidos do início do campo **Link Code** até o início do próximo campo **Link Code** (ou até o final da mensagem, se for o caso);
- **Endereço da Interface Vizinha:** endereço IP de uma interface vizinha deste nó.

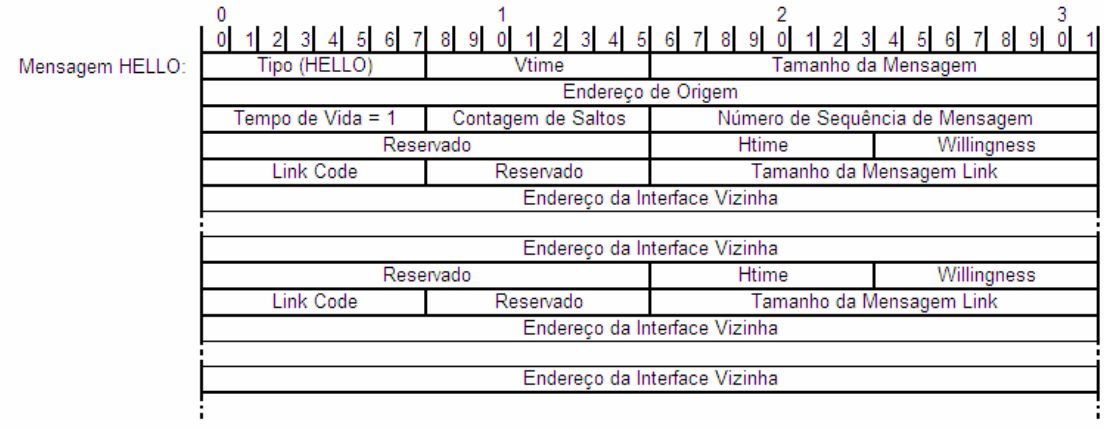

Figura 2.4 – Mensagem HELLO

Todas as mensagens HELLO têm seus campos de **Tempo de Vida** gerados com valor 1 e, portanto, tem escopo de difusão limitado à vizinhança de um salto de cada nó. Durante seu processamento, as mensagens HELLO incorrem na compilação do estado atual da vizinhança do nó em um repositório unificado de informações. Essa base de informações, apesar de compreender o termo "vizinhança" em geral, abrange os seguintes aspectos diversos do processamento OLSR:

- **Povoamento do Conjunto de Vizinhos,**
- **Povoamento do Conjunto de Vizinhos de Dois Saltos,**
- **Povoamento do Conjunto MPR,**
- **Povoamento do Conjunto de Seletores MPR.**

2.1.7.1. Povoamento do Conjunto de Vizinhos

O repositório **conjunto de vizinhos** mantém as informações referentes aos vizinhos de um salto do nó. Aqui, a simetria, o valor de *willingness* e os endereços principais dos nós imediatamente próximos são armazenados e consultados conforme necessário.

O processamento das mensagens HELLO possibilita a obtenção dos valores supracitados. Para a simetria em relação a um nó específico, os dois nós envolvidos observam o encadeamento de eventos demonstrado na Figura 2.5. Já para o endereço principal e o parâmetro de *willingness*, a leitura direta dos campos da mensagem HELLO é suficiente.

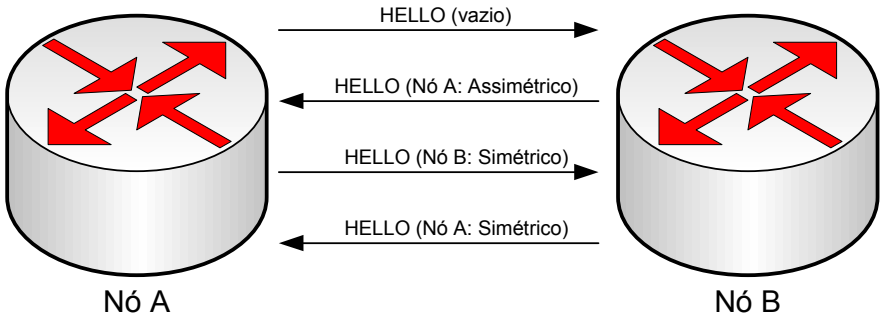

Figura 2.5 – Detecção de Simetria

O **conjunto de vizinhos** goza de uma relação direta com a **base de informações de enlaces locais**, mais especificamente com o **conjunto de enlaces**. A cada mensagem de HELLO recebida são criadas, conforme adequado, entradas no **conjunto de enlaces** para cada enlace detectado. É importante observar que entradas no **conjunto de enlaces** são armazenadas utilizando os endereços de interface e não os endereços principais dos nós. Esse comportamento dá origem à relação entre tais repositórios: para cada entrada específica no **conjunto de vizinhos** existirão uma ou mais entradas no **conjunto de enlaces**. Deste modo, para um vizinho de um salto ser considerado simétrico, deverá existir ao menos um enlace simétrico entre quaisquer interfaces dos respectivos nós registrado no **conjunto de enlaces**. Mais ainda, só existirá uma entrada no **conjunto de vizinhos** se houver pelo menos um enlace registrado para este vizinho de um salto no **conjunto de enlaces**.

2.1.7.2. Povoamento do Conjunto de Vizinhos de Dois Saltos

O **conjunto de vizinhos de dois saltos** de um nó contém os endereços principais dos vizinhos simétricos dos vizinhos de um salto simétricos.

Para construir essa base de informações o seguinte processamento das mensagens HELLO é aplicado: para cada endereço listado na mensagem como vizinho simétrico ou vizinho MPR (nós selecionados como MPRs são identificados nas mensagens HELLO) que não for do próprio nó recebendo a mensagem é adicionada uma entrada no repositório.

Esse processamento, entretanto, pode incluir nós no **conjunto de vizinhos de dois saltos** que também são atingíveis com apenas um salto. Porém, tal comportamento não prejudica a operação do protocolo OLSR.

2.1.7.3. Povoamento do Conjunto MPR

Conforme descrito anteriormente, nós MPR são utilizados para otimizar o processo de *flooding* em um ambiente OLSR, reduzindo o número de retransmissões necessárias.

Cada nó, portanto, constrói independentemente seu próprio **conjunto MPR** se utilizando de determinados nós constantes de sua vizinhança simétrica de um nó. O **conjunto MPR** é povoado de forma a compreender um número suficiente de vizinhos de um nó simétricos capazes de prover conectividade também simétrica para todos os vizinhos de dois saltos do nó. Embora não seja mandatório pela RFC que o **conjunto MPR** seja mínimo, quanto menor o for, menor será o desperdício de banda com tráfego de controle.

O **conjunto MPR** de um nó pode coincidir com sua vizinhança simétrica. Este será o caso da inicialização da rede, correspondendo, em tal instante, ao algoritmo de estado de enlace clássico, sem, então, otimização.

O processamento das mensagens HELLO é o suporte do algoritmo de computação do **conjunto MPR** utilizado pela RFC 3626 [1]. A heurística proposta para a seleção dos MPRs é computada por interface e a união dos conjuntos MPR das interfaces dá origem ao **conjunto MPR** do nó.

Para a descrição do algoritmo de obtenção do **conjunto MPR** a seguinte terminologia será utilizada:

- **V:** V é o conjunto de vizinhos do nó, que são vizinhos da interface I;
- **V2:** V2 é o conjunto de vizinhos de dois saltos atingíveis pela interface I, excluindo os nós só atingíveis por membros de V com *willingness* igual a 0, o próprio nó e os vizinhos simétricos;
- **N(x):** N(x) é o nível de um vizinho de um salto x. Esse nível é definido como o número de vizinhos simétricos de x, excluindo todos os membros de V e o próprio nó executando o algoritmo.

Nesse contexto, a heurística proposta pelo protocolo OLSR é a seguinte:

1. Cria-se, a princípio, um conjunto MPR composto pelos membros de V com *willingness* igual a 7 (sempre).

2. Calcula-se N(x), onde x é um membro de V, para todos os nós em V.

3. Adiciona-se ao conjunto MPR aqueles nós em V que são os únicos capazes de

prover conectividade para um nó em V2. Remove-se, então, os nós de V2 para os quais é provida conexão por um nó no conjunto MPR.

4. Enquanto existirem nós em V2 não cobertos por ao menos um nó no conjunto MPR:

    a. Para cada nó em V, calcular a alcançabilidade, ou seja, o número de nós em V2 que não são cobertos por ao menos um nó no conjunto MPR e que são alcançáveis por este vizinho;

    b. Selecionar como um MPR o nó com o maior valor de *willingness* dentre os nós em V com alcançabilidade diferente de zero. No caso de múltiplas escolhas, selecionar o nó que possa prover alcançabilidade para o maior número de nós em V2. No caso de múltiplos nós provendo a mesma alcançabilidade, escolher o nó cujo N(x) é o maior. Remover os nós de V2 agora cobertos por um nó do conjunto MPR.

5. Como uma otimização proposta, pode-se processar cada nó x do conjunto MPR em ordem crescente de *willingness*. Se todos os nós em V2 ainda se mantiverem cobertos por pelo menos um nó do conjunto MPR quando da exclusão de x e se x tem *willingness* diferente de sempre (7), então x poderá ser removido do conjunto MPR. Finalmente, o **conjunto MPR** do nó será a união dos conjuntos MPR computados acima para cada interface I.

2.1.7.4. Povoamento do Conjunto de Seletores MPR

Outra informação importante fornecida pelas mensagens HELLO é o conjunto de nós que escolheram o roteador em questão como seu MPR. A cada mensagem HELLO recebida, deve-se checar a lista de enlaces sendo divulgada para verificar a existência de algum endereço de interface próprio sendo publicado como vizinho MPR. Se assim o for, adiciona-se o endereço principal do nó que gerou a mensagem HELLO no **conjunto de seletores MPR** do nó. Adiciona-se também, ou renova-se, quando for o caso, o tempo de validade da entrada computado através do campo **Vtime** obtido do cabeçalho da mensagem HELLO (Figura 2.4). Tal nó só será, então, considerado como um seletor MPR enquanto a validade de sua entrada

não estiver expirada.

2.1.8. Controle de Topologia

O processo de descoberta da vizinhança exposto oferece para cada nó uma compreensão do estado de rede à sua volta. Esse conhecimento deve, então, ser processado e adequadamente disseminado pelo ambiente OLSR para que o cálculo de rotas possa ser realizado. A informação a ser, portanto, divulgada é definida como o estado de enlace do nó para com seus vizinhos. Isso é realizado através das mensagens de Controle de Topologia, ou, mensagens TC (*Topology Control*). O formato de uma mensagem TC pode ser visualizado na Figura 2.6 e seus campos são os seguintes:

- **ANSN (*Advertised Neighbor Sequence Number*):** Este é o número de seqüência associado ao conjunto de vizinhos sendo anunciado. Toda vez que o nó detecta uma mudança em tal conjunto, esse número é incrementado.
- **Endereço Principal do Vizinho Divulgado:** Este campo contém o endereço principal de um nó vizinho que está sendo divulgado.
- **Reservado:** Campo que deve ter todos seus bits marcados como zero para atender à especificação da RFC.

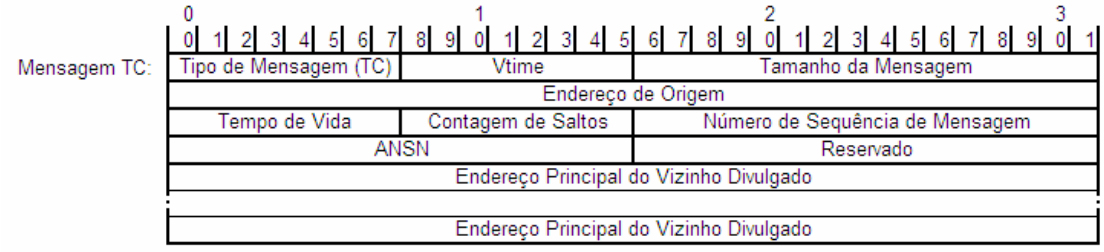

Figura 2.6 – Mensagem TC

Para possibilitar ao algoritmo OLSR calcular as rotas adequadamente, cada nó deverá divulgar, via mensagens TC, pelo menos os enlaces que possui com os nós vizinhos presentes

em seu **conjunto de seletores MPR**. Essa otimização de não enviar todos os enlaces do nó permite uma economia de banda com a diminuição do tamanho das mensagens TC. De fato, o nó poderá divulgar, para efeito de redundância, mais enlaces do que somente aqueles com os nós do **conjunto de seletores MPR**. Isso é, entretanto, opcional e será exposto quando as funções auxiliares do OLSR forem demonstradas. Outra otimização derivada da anterior é que se somente se divulga os enlaces que existem com os nós no **conjunto de seletores MPR**, então somente MPRs irão gerar mensagens TCs, reduzindo, conseqüentemente, o tráfego de controle.

2.1.8.1. Povoamento da Base de Informações de Controle de Topologia

As mensagens TC são disseminadas na rede de acordo com o paradigma do *flooding* via MPRs. O repositório de informações que mantém o controle de topologia do nó é a **base de informações de controle de topologia**. Assim sendo, o processamento das mensagens TC ocorre da seguinte maneira:

1. Ao receber uma mensagem TC, o nó verifica se existe uma entrada no repositório de controle de topologia com o endereço de origem do nó que enviou a mensagem. Se não há, uma é criada com o tempo de validade e ANSN presentes na mensagem, onde constaram os vizinhos divulgados pelo originador.

2. Se já houver uma entrada na **base de informações de controle de topologia**, mas o ANSN for anterior ao recebido, a entrada toda é atualizada com os novos valores.

3. Se já houver uma entrada com o mesmo ANSN recebido, então somente o tempo de validade da entrada é atualizado.

2.1.9. Cálculo de Rotas

Cada nó mantém uma tabela de roteamento que permite o devido encaminhamento da informação na rede. Em um ambiente OLSR, a tabela de roteamento é atualizada e

recalculada toda vez que uma mudança nos seguintes repositórios é detectada:

- **Conjunto de Enlaces,**
- **Conjunto de Vizinhos,**
- **Conjunto de Vizinhos de Dois Saltos,**
- **Base de Informações de Controle de Topologia,**
- **Base de Informações sobre Associação de Múltiplas Interfaces.**

O algoritmo proposto pela RFC 3626 [1] para o cálculo de rotas é um SPF (*Shortest Path First*) simples, ou seja, se baseia no menor caminho para os destinos. Sua heurística é a seguinte:

1. Adiciona-se à tabela de roteamento com métrica 1 todos os vizinhos de um salto registrados como simétricos.

2. Para cada vizinho de um salto adicionado, adiciona-se, também, todos os vizinhos de dois saltos registrados para tal vizinho que sejam à ele simétricos e que já não tenham sidos adicionados. Tais entradas terão 2 como métrica e próximo salto como o vizinho que a gerou.

3. Depois, para cada nó da tabela de roteamento com métrica M = 2, adiciona-se todos os nós da **base de informações de controle de topologia** registrados sob a responsabilidade de tal nó e que já não tenham sido adicionados. Tais entradas terão M + 1 como métrica e próximo salto como o próximo salto do nó que gerou a entrada.

4. Em seguida, incrementa-se M de 1 e repete-se a etapa 3 até que todos os nós com métrica igual ao novo M tenham sido processados.

5. Finalmente, para todas as entradas na tabela de roteamento, a **base de informações sobre associações de múltiplas interfaces** é consultada. Se para a entrada examinada existirem mais endereços de interface associados, uma nova entrada na tabela de roteamento é criada para cada associação, tendo como métrica e próximo salto os valores registrados para a entrada original.

2.1.10. Visão Geral da Funcionalidade Núcleo

Viu-se que a funcionalidade núcleo rege o funcionamento básico necessário para um nó operar em um ambiente OLSR. Como conseqüência, relações imperativas entre os mecanismos operacionais básicos e os repositórios de informações do OLSR são traçadas. A Figura 2.7 demonstra, por conseguinte, uma visão geral de tais repositórios de informações e suas relações com os mecanismos de processamento de mensagens, geração de mensagens e cálculo de rotas.

Neste contexto, mensagens HELLO recebidas disparam atualizações no **conjunto de enlaces**, no **conjunto de vizinhos de dois saltos** e no **conjunto de seletores MPR**. Alterações na **base de informações de enlaces locais**, mais especificadamente no **conjunto de enlaces**, provocam automaticamente mudanças no **conjunto de vizinhos**. Por sua vez, modificações no **conjunto de vizinhos** provocam o recálculo do **conjunto MPR**. Mais ainda, as atualizações no **conjunto de vizinhos de dois saltos** também induzem à recomputação do **conjunto MPR**.

Para as mensagens TC e MID, estas promovem, ao seu recebimento, a atualização da **base de informações de controle de topologia** e da **base de informações sobre associações de múltiplas interfaces**, respectivamente.

Todas as mensagens recebidas são registradas na **base de informações de duplicatas** se forem consideradas inéditas. Quando, então, o mecanismo de encaminhamento é utilizado, a própria **base de informação de duplicatas** é consultada, juntamente com o **conjunto MPR**, para determinar se e como o tráfego de controle será repassado.

Para o processo de geração de mensagens, mais especificamente para as mensagens HELLO, o **conjunto de enlaces**, o **conjunto de vizinhos** e o **conjunto MPR** são consultados. E para o caso das mensagens TC, somente o **conjunto de seletores MPR** é examinado.

Finalmente, para o mecanismo de cálculo de rotas, o **conjunto de vizinhos**, o **conjunto de vizinhos de dois saltos**, a **base de informações de controle de topologia** e a **base de informações sobre associações de múltiplas interfaces** são todos consultados durante o processo de construção e atualização da tabela de roteamento.

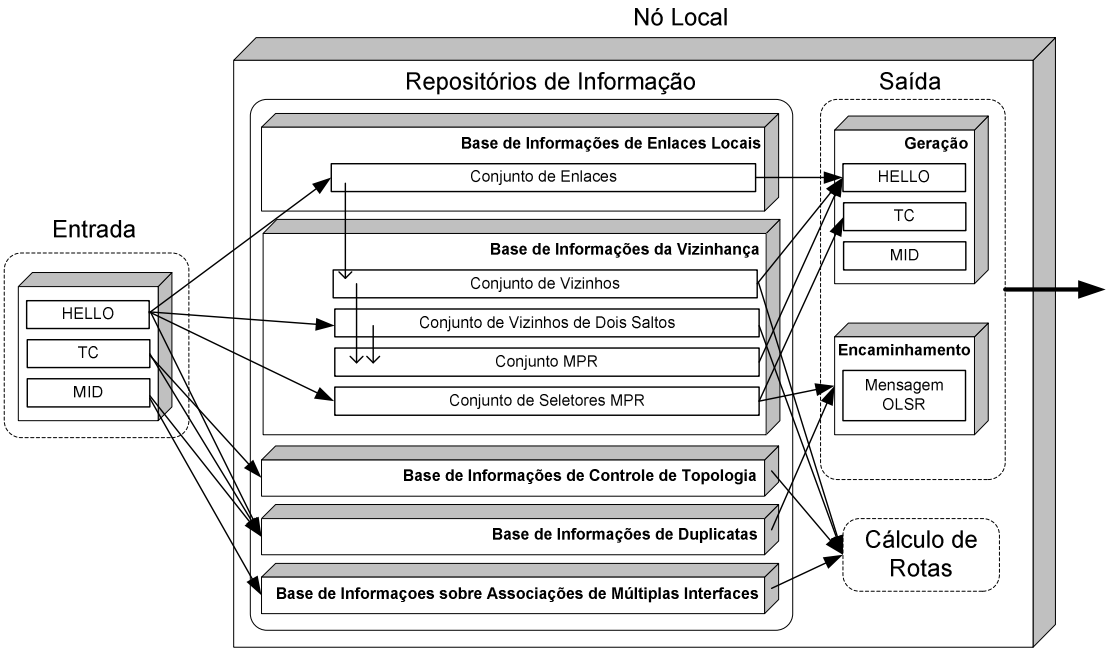

Figura 2.7 – Relação entre os repositórios de informações e a funcionalidade núcleo do OLSR.

2.2. FUNÇÕES AUXILIARES

A RFC 3626 [1], como especificado anteriormente, modulariza o protocolo OLSR em funcionalidade núcleo e funções auxiliares. As funções auxiliares são especificadas de forma a prover ao OLSR mecanismos de funcionamento extras aplicáveis para cenários específicos. Tais mecanismos, entretanto, podem ser implementados sem impactar nos nós com apenas a funcionalidade núcleo. Isso é possível devido ao mecanismo de encaminhamento padrão já demonstrado, onde mensagens de tipos desconhecidos não são descartadas e sim retransmitidas respeitando o paradigma MPR. As seguintes funções auxiliares estão definidas na RFC 3626 [1]:

- HNA,
- Histerese,
- Redundância TC,
- Redundância MPR,

- Notificações de Camada de Enlace.

2.2.1. HNA

Redes OLSR podem existir totalmente independentes de outras redes. No entanto, poderá haver o interesse de interligar outros domínios de roteamento com o ambiente OLSR. É neste contexto que as mensagens HNA (*Host and Network Association*) se inserem. Elas possibilitam a injeção da informação externa de roteamento para dentro da rede OLSR.

Para prover tal capacidade, os nós OLSR de borda (nós que tenham interfaces de rede em ambos os domínios a serem interligados) deverão anunciar essa conectividade para a rede OLSR via mensagens HNA. Nós exclusivamente OLSR que desejarem suportar o acesso para redes externas deverão ser capazes de processar as mensagens HNA; e nós que não o desejarem deverão respeitar a RFC em sua funcionalidade núcleo e aplicar o mecanismo de encaminhamento padrão nessas mensagens.

O caminho inverso da interligação, ou seja, a injeção da informação do roteamento OLSR para outras redes não é, porém, objeto desta especificação. Deve ficar claro, no entanto, que tal informação de roteamento OLSR pode ser extraída da **base de informações de controle de topologia** ou diretamente a tabela de roteamento OLSR e ser injetada nas interfaces não OLSR locais utilizando qualquer mecanismo que seja provido pelo outro domínio de roteamento.

2.2.1.1. Formato da Mensagem HNA

Uma mensagem HNA é basicamente uma lista de endereços de redes com suas respectivas máscaras de rede. Seu formato pode ser visualizado na Figura 2.8.

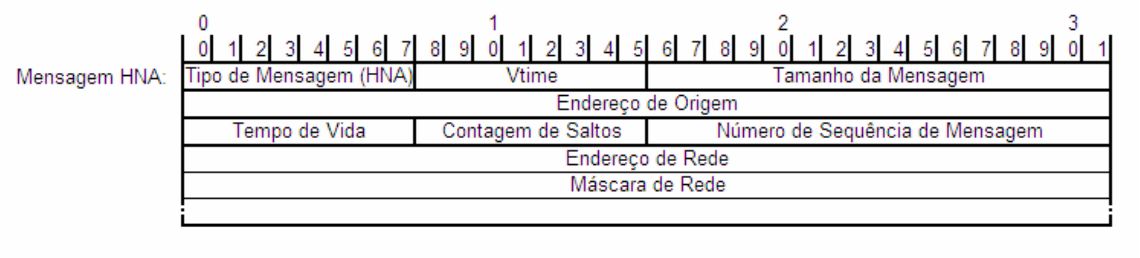

Figura 2.8 – Mensagem HNA

2.2.1.2. Processamento da Mensagem HNA

Toda informação HNA recebida é registrada em uma nova base chamada **base de informações HNA**. Este repositório de informações adicional funciona da mesma maneira que os já citados, ou seja, suas entradas são mantidas até que suas validades expirem. Cada entrada HNA registrada contém os seguintes campos:

- ***Gateway*:** endereço principal do nó anunciando conectividade com redes externas. Este valor é extraído do campo **Endereço de Origem** da mensagem HNA;
- **Rede:** a rede a ser atingida;
- **Máscara de Rede:** a máscara de rede delimitando o endereço da rede a ser atingida;
- **Vtime:** o tempo de validade da entrada, retirado do campo **Vtime** da mensagem HNA.

Quando uma mensagem HNA é recebida, a **base de informações HNA** é atualizada de forma a renovar a validade de uma entrada, se a mesma já tiver sido registrada; ou a criar uma nova, se para o campo **Gateway** recebido não existirem tais entradas.

2.2.1.3. Cálculo de Rotas com Suporte HNA

Se o nó em questão implementar a função auxiliar HNA, ele deverá adicionar ao algoritmo de cálculo de rotas do OLSR já descrito os seguintes passos:

1. Para cada entrada na **base de informações HNA,** a tabela de roteamento do nó

é percorrida em busca de uma rota com o destino igual à rede/máscara da entrada sendo processada.

a. Se não existir tal rota, uma é criada com destino igual à rede/máscara da entrada HNA, com a métrica igual à distância para o nó ***Gateway*** e com próximo salto igual ao próximo salto da rota para atingir o nó ***Gateway***.

b. Se existir tal rota, a métrica da rota é verificada.

    i. Se a métrica for maior que o número de saltos para atingir o ***Gateway*** sendo processado, a entrada da tabela de roteamento será modificada. Essa modificação dará origem a uma rota com destino igual à rede/máscara da entrada HNA, com a métrica igual à distância para o nó ***Gateway*** e com próximo salto igual ao próximo salto da rota para atingir o nó ***Gateway***.

    ii. Se a métrica for menor ou igual ao número de saltos para atingir o ***Gateway*** sendo processado, nada é modificado.

2.2.2. Histerese

É desejável que, uma vez estabelecidos, os enlaces sejam tão confiáveis quanto possível. Isso implica que a própria detecção de enlaces seja robusta e, conseqüentemente, resistente às perdas por rajadas ou à instabilidade temporária entre os nós. Assim, para aperfeiçoar a robustez do mecanismo de detecção de enlaces, a RFC 3626 [1] propõe a função auxiliar de histerese.

A estratégia sugerida pela histerese é baseada em duas funções: uma função de **estabilidade** e uma função de **instabilidade**. Além disso, são definidos mais três parâmetros: um fator de escalabilidade (HYST_SCALING) e dois limiares de qualidade de enlace (HYST_THRESHOLD_LOW e HYST_THRESHOLD_HIGH). Essas funções e parâmetros delineiam, então, o controle da entidade adicional chamada **qualidade de enlace**. Tal entidade tem como objetivo traduzir um valor para a condição atual do estado de cada enlace e é armazenada como em um campo adicional na **base de informações de enlaces locais**, mais

especificamente no **conjunto de enlaces**.

O valor corrente da **qualidade de enlace** vai disparar atualizações no estado dos enlaces quando os limiares definidos forem atingidos. Esse comportamento vai proteger o protocolo das mudanças instantâneas, conferindo, portanto, mais robustez ao ambiente.

No paradigma da histerese, o estado dos enlaces só é alterado segundo duas condições:

- Um enlace será marcado como simétrico apenas se ele estiver atualmente marcado como assimétrico e a **qualidade de enlace** estiver maior que o limiar superior de qualidade (HYST_THRESHOLD_HIGH).
- Um enlace será marcado como assimétrico apenas se ele estiver atualmente marcado como simétrico e a **qualidade de enlace** estiver menor que o limiar inferior de qualidade (HYST_THRESHOLD_LOW).

Assim, a entidade **qualidade de enlace** rege como se procederão as mudanças nas simetrias dos enlaces. O valor propriamente dito da **qualidade de enlace** é calculado pela aplicação das funções de **estabilidade** e **instabilidade**. A RFC do protocolo OLSR define tais funções da seguinte forma:

- **Função de Estabilidade: QUALIDADE DE ENLACE** = (1 – HYST_SCALING)***QUALIDADE DE ENLACE** + HYST_SCALING
- **Função de Instabilidade: QUALIDADE DE ENLACE** = (1 – HYST_SCALING)***QUALIDADE DE ENLACE**

A função de **estabilidade** deve ser aplicada em um enlace registrado toda vez que um pacote OLSR for recebido por este enlace. Já a função de **instabilidade** deverá ser acionada para um enlace registrado toda vez que um pacote OLSR for perdido no enlace. Perdas de pacotes OLSR são detectadas pela ordem dos números de seqüência dos pacotes, obtidos do campo **Número de Seqüência do Pacote** (Figura 2.1), e pela observação do tempo de emissão das mensagens HELLO, definido no campo **Htime** (Figura 2.4).

A RFC sugere os seguintes valores para o cálculo da histerese:

- **HYST_THRESHOLD_HIGH = 0.8**
- **HYST_THRESHOLD_LOW = 0.3**
- **HYST_SCALING = 0.5**

A função auxiliar histerese é uma técnica altamente ajustável e terá seu comportamento significativamente alterado dados outros valores para seus parâmetros.

2.2.3. Redundância TC

Como foi exposto no tópico Controle de Topologia, dentro da funcionalidade núcleo do OLSR, os nós irão divulgar via mensagens TC apenas aqueles vizinhos que o selecionaram como MPR. Esse comportamento, apesar de trazer a economia de banda, diminui a robustez do entendimento do ambiente de rede. Os nós poderão nunca conhecer os múltiplos caminhos que eventualmente existam para os destinos.

Logo, para permitir um conhecimento mais robusto da topologia de rede, os nós poderão anunciar, através das mensagens TC, mais do que apenas os integrantes do **conjunto de seletores MPR**. Para isso, a função auxiliar de redundância TC introduz o parâmetro TC_REDUNDANCY. Todos os dispositivos implementando a redundância TC construirão suas mensagens TC baseadas neste parâmetro, e os seguintes comportamentos serão adotados para os respectivos valores de TC_REDUNDANCY:

- **0:** O conjunto de nós anunciados nas mensagens TC é limitado ao **conjunto de seletores MPR**. Este é o comportamento padrão.
- **1:** O conjunto de nós anunciados nas mensagens TC será a união do **conjunto MPR** com o **conjunto de seletores MPR**.
- **2:** O conjunto de nós anunciados nas mensagens TC será o conjunto composto de todos os vizinhos simétricos do nó.

Se todos os nós de um ambiente OLSR implementarem a função auxiliar de redundância TC e todos estiverem com TC_REDUNDANCY marcado com 2, então todos os enlaces simétricos serão anunciados para todos os nós. Isso significará que todos os dispositivos, não somente os MPRs, irão gerar mensagens TC.

Finalmente, um nó nunca irá saber como o parâmetro TC_REDUNDANCY dos outros dispositivos estará marcado. Mais ainda, para manter o proposto pelas funções auxiliares, um nó não precisará ter conhecimento dos parâmetros TC_REDUNDANCY alheios e dispositivos com e sem a redundância TC poderão coexistir no mesmo ambiente.

2.2.4. Redundância MPR

Como foi exposto no tópico Povoamento do Conjunto MPR, dentro da funcionalidade núcleo do OLSR, a heurística proposta visa gerar um **conjunto MPR** o mais restrito possível para economizar banda com tráfego de controle. Todavia, a escolha de cobrir os vizinhos de dois saltos com apenas pelo menos um vizinho de um salto simétrico sacrifica mais uma vez a robustez do protocolo.

Neste contexto, a função auxiliar redundância MPR introduz o parâmetro MPR_COVERAGE. Com ele, o algoritmo para escolha do **conjunto MPR** é adaptado para objetivar selecionar um conjunto de nós onde cada vizinho de dois saltos seja coberto por pelo menos MPR_COVERAGE vizinhos de um salto simétricos. Obviamente, nem todos os nós poderão ser cobertos pela quantidade de vizinhos de um salto simétricos propostos pelo parâmetro MPR_COVERAGE. Assim, a redundância MPR se torna a tentativa de ter todos os vizinhos de dois saltos cobertos por pelo menos MPR_COVERAGE MPRs.

2.2.4.1. Povoamento do Conjunto MPR adaptado para a Redundância MPR

Ao utilizar a função auxiliar redundância MPR, os nós deverão adaptar seus algoritmos de seleção do conjunto MPR. Mais uma vez é importante frisar que a opção pela redundância MPR não influencia na interoperabilidade OLSR com os nós que não a empregam. É de simples observação, também, que a escolha de MPR_COVERAGE como 1 torna a operação

de selecionar o conjunto MPR equivalente à da funcionalidade núcleo.

Assim sendo, para a descrição do algoritmo de eleição do conjunto MPR adaptado para redundância MPR a seguinte terminologia será empregada:

- **V:** V é o conjunto de vizinhos do nó, que são vizinhos da interface I;
- **V2:** V2 é o conjunto de vizinhos de dois saltos atingíveis pela interface I, excluindo os nós só atingíveis por membros de V com *willingness* igual a 0, o próprio nó e os vizinhos simétricos;
- **N(x):** N(x) é o nível de um vizinho de um salto x. Esse nível é definido como o número de vizinhos simétricos de x, excluindo todos os membros de V e o próprio nó executando o algoritmo;
- **P:** P é um nó em V2 pobremente coberto, ou seja, é um nó atingível por uma quantidade menor do que MPR_COVERAGE nós de V.

Agora, a heurística proposta pela função auxiliar redundância MPR para povoar o **conjunto MPR** do nó se adaptará para a seguinte:

1. Cria-se, a princípio, um conjunto MPR composto pelos membros de V com *willingness* igual a 7 (sempre).

2. Calcula-se N(x), onde x é um membro de V, para todos os nós em V.

3. Adiciona-se ao conjunto MPR aqueles nós em V que são capazes de prover conectividade para os nós P em V2. Remove-se, então, os nós P de V2 para os quais é provida conexão pelo conjunto MPR.

4. Enquanto existirem nós em V2 não cobertos por ao menos MPR_COVERAGE nós no conjunto MPR:

    a. Para cada nó em V, calcular a alcançabilidade, ou seja, agora o número de nós em V2 que não são cobertos por ao menos MPR_COVERAGE nós no conjunto MPR e que são alcançáveis por este vizinho;

    b. Selecionar como um MPR o nó com o maior valor de *willingness*

dentre os nós em V com alcançabilidade diferente de zero. No caso de múltiplas escolhas, selecionar o nó que possa prover alcançabilidade para o maior número de nós em V2. No caso de múltiplos nós provendo a mesma alcançabilidade, escolher o nó cujo N(x) é o maior. Remover os nós de V2 agora cobertos por MPR_COVERAGE nós do conjunto MPR.

5. Como uma otimização proposta, pode-se processar cada nó x do conjunto MPR em ordem crescente de *willingness*. Se todos os nós em V2 ainda se mantiverem cobertos por pelo menos MPR_COVERAGE nós do conjunto MPR quando da exclusão de x e se x tem *willingness* diferente de sempre (7), então x poderá ser removido do conjunto MPR. Finalmente, o conjunto MPR do nó será a união dos conjuntos MPR computados acima para cada interface I.

2.2.5. Notificações de Camada de Enlace

O protocolo OLSR, como especificado na RFC 3626, funciona independentemente das camadas inferiores de rede. Assim, todo o processamento de detecção e manutenção de enlaces é de responsabilidade da própria operação do OLSR.

Contudo, no caso de tais camadas inferiores poderem fornecer recursos para melhorar a robustez dos mecanismos de detecção e manutenção de enlaces, é interessante que o protocolo OLSR possa aproveitar essa possibilidade.

É para isso que a função auxiliar de notificações de camada de enlace é proposta. Com ela, o protocolo OLSR introduz um novo campo nas entradas do repositório **conjunto de enlaces** chamado L_LOST_LINK. Este campo será utilizado quando uma notificação de uma camada de enlace for recebida indicando que um enlace foi perdido. Nesse momento o OLSR marcará o L_LOST_LINK com um temporizador evidenciando a falha do enlace e toda a operação do protocolo considerará este enlace como perdido.

Esta, como todas as outras funções auxiliares, pode operar independentemente, não influenciando na operação das demais funções auxiliares, se implementadas, ou mesmo na operação da funcionalidade núcleo dos outros nós.

3. OLSR DAEMON - *OLSRD*

No capítulo anterior foi exposta a RFC do protocolo OLSR. Padronizando o comportamento de entidades que ensejam adotá-lo em seus domínios de roteamento, tal documento é, porém, apenas a especificação de como se deve materializar o protocolo.

Para que os nós de rede efetivamente roteiem utilizando o OLSR, implementações se tornam mister. Implementações são, neste contexto, os programas que concretizam o que foi delineado no documento oficial.

No caso da RFC 3626 [1], disponíveis e em conformidade com o protocolo OLSR, existem algumas implementações. Reconhecidamente, as mais importantes são a *NRL olsrd* [7], a *QOLSR* [8] e a *Unik OLSR daemon* [2].

A *NRL olsrd* é um projeto do laboratório de pesquisa naval (*Naval Research Laboratory*) do exército dos EUA. Desenvolvida em C++, essa implementação está disponível para várias plataformas, mas tem como ponto frágil o não suporte de múltiplas interfaces.

Também desenvolvida em C++, a *QOLSR*, por sua vez, é voltada para a pesquisa de Qualidade de Serviço para MANET. Essa implementação é parte de um projeto do *Laboratoire de Recherche en Informatique* da França.

Já a *Unik OLSR daemon* é uma implementação do OLSR desenvolvida em C pelo *Unik Graduate Center* da universidade de Oslo. Hoje registrada em http://www.olsr.org, essa implementação se tornou bastante popular e conta com uma ativa comunidade suportando as evoluções do projeto. Agora conhecida como *OLSR Daemon*, ou apenas *OLSRD*, essa implementação estava inicialmente disponível apenas para GNU/Linux, mas atualmente a lista de plataformas suportadas cresceu para acomodar FreeBSD, Linux, OpenBSD, OS X, Win2K, WinXP e Win98 OSR2. Além disso, o *OLSRD* disponibiliza todas as funções auxiliares sugeridas na RFC 3626, além da mandatória funcionalidade núcleo. Mais ainda, essa implementação é a única a fornecer para a comunidade a possibilidade de adicionar extensões à implementação via bibliotecas dinamicamente carregáveis (*plugins*). Tal interface *plugin* confere à implementação mais modularidade e, conseqüentemente, mais extensibilidade. Outra importante característica importante do *OLSRD* veio com a grande difusão da implementação e da subseqüente experimentação da comunidade com o protocolo.

Esses fatores possibilitaram, então, a descoberta de problemas práticos com alguns mecanismos descritos na RFC, motivando a criação da extensão *Link Quality*. Essa proposta de alteração do funcionamento da RFC permite uma significativa melhora de performance do protocolo e será, posteriormente, explicada neste capítulo. Tudo isso, enfim, consolidou o *OLSRD* como a principal implementação do protocolo OLSR disponível para a comunidade.

É, por conseguinte, nesta conjuntura que é possível posicionar a referida implementação junto ao trabalho objetivado nesta dissertação. Oferecer uma estrutura de gerenciamento para o protocolo OLSR necessariamente inclui o interfaceamento com o funcionamento efetivo de um ambiente OLSR e, conseqüentemente, a comunicação entre tal estrutura de gerenciamento e uma implementação se torna imprescindível. Dessa forma, reconheceu-se no *OLSRD* o potencial de oferecer a base na qual o modelo proposto de administração do protocolo poderia se instalar. Neste capítulo, oferece-se, portanto, uma exposição desta escolhida implementação, evidenciando sua estrutura de funcionamento e seus componentes.

3.1. VISÃO GERAL

O *OLSRD* foi desenvolvido na linguagem de programação C e hoje continua recebendo atualizações e extensões. No caso específico da RFC 3626 [1] é possível existirem implementações que não cumpram o documento em sua íntegra, uma vez que são definidas, como já foi exposto no capítulo anterior, uma funcionalidade núcleo, essa sim obrigatória, e funções auxiliares opcionais. O *OLSRD*, entretanto, disponibiliza para funcionamento tudo que é disposto na RFC, inclusive as partes optativas.

Essa abrangência tornou a implementação extensa e alguns princípios tiveram que ser seguidos para manter a qualidade do projeto [2]:

- **Modularidade:** Todo código que puder ser visto como um mecanismo genérico por outras entidades deve ser implementado o mais modular possível. Isso, em muitos casos, significa que entidades utilizando tais funções devem se registrar dinamicamente com a função. Um exemplo disso é a funcionalidade de agendamento. Nela, entidades que desejam gerar mensagens em intervalos regulares devem registrar uma função de geração de mensagens no agendador. Esse tipo de princípio se torna basilar quando se introduz o

conceito da interface *plugin*, onde funcionalidades externas poderão ser adicionadas sem alterar o código principal do OLSR.

- **Estrutura de Dados Consistente:** Em um protocolo regido por tabelas, a maior parte da computação será processada obviamente em tabelas. Isso significa que a maior parte do código será relacionada com listas de diferentes tipos. Contudo, deve-se almejar, nos casos onde seja razoável, que todas essas partilhem da mesma estrutura.
- **Transparência IP:** O *OLSRD* deverá, para manter a compatibilidade com a RFC, operar tanto com endereços IPv4 [3] quanto com IPv6 [4]. Deve-se procurar, todavia, que esse duplo suporte seja feito de maneira o mais transparente possível, significando que o código deva operar para ambos os tipos de endereços. Nos casos onde funções separadas sejam feitas para cada tipo de endereço, uma função de encapsulamento (*wrapper*) deverá ser implementada.
- **Código Intuitivo:** O código deverá ser o de mais simples leitura externa possível. Isso é importante uma vez que se trata de um projeto extensível e aberto.
- **Código Independente de Plataforma:** Código que seja dependente de plataforma deverá ser separado do resto em uma maneira modular. Uma interface definida deverá reger o acesso a tal parte de código. Dessa forma, implementar esse tipo de funções será o mais simples possível.

Descrever-se-á, agora, como o *OLSRD* está organizado. Sua estrutura de funcionamento, seus componentes e principais características serão explanados na seguinte ordem:

- Estrutura de Funcionamento,
- Extensão *Link Quality*,
- Interface *Plugin*.

3.2. ESTRUTURA DE FUNCIONAMENTO

Atualmente o *OLSRD* está licenciado sob os termos da licença de código aberto BSD [9], e tem seu código disponível em www.olsr.org. Para instalá-lo é necessário buscar a distribuição via arquivo de forma **olsrd-x.x.x.tar.bz2** (com as letras x representando as eventuais versões)

e descompactá-lo. Será visualizada a árvore do pacote contendo os seguintes diretórios:

- **.:** arquivos diversos, incluindo as instruções de instalação e licença do programa;
- ***files*:** onde se encontram os arquivos de configuração que regem o funcionamento do *OLSRD*;
- ***GUI:*** interface gráfica opcional da implementação;
- ***lib:*** código de vários *plugins*;
- ***make:*** arquivos que preparam a instalação para as várias plataformas;
- ***src:*** arquivos do código fonte do *OLSRD.*

Em seguida, deve-se observar as instruções referentes a cada plataforma e compilar o programa de acordo. Ao término, o arquivo de configuração com as variáveis e diretivas gerais do protocolo precisa ser ajustado com os parâmetros de funcionamento desejados; e, depois, pode-se iniciar a operação do protocolo OLSR no nó.

A Tabela 3.1 expõe os parâmetros presentes no arquivo de configuração do *OLSRD*. Seguindo as recomendações da RFC 3626 [1], são sugeridos valores padrões iniciais para as variáveis e, também, é especificado a que mecanismo se relaciona cada diretiva e de que tipo ela é. Uma breve descrição dos parâmetros é oferecida também.

Tabela 3.1 – Parâmetros do Arquivo de Configuração do *OLSRD* – em Conformidade com a RFC 3626

PARÂMETRO	**VALOR SUGERIDO**	**MECANISMO**	**TIPO**	**DESCRIÇÃO**
Parâmetros Globais				

DebugLevel	1	Funcionamento Global do *OLSRD*	-	Especifica a quantidade de informação de diagnóstico a ser fornecida pelo processo do *OLSRD*.
IpVersion	4	Funcionamento Global do *OLSRD*	Funcionalidade Núcleo	Versão do protocolo IP a ser utilizada.
ClearScreen	yes	Funcionamento Global do *OLSRD*	-	Utilizado para desobstruir a tela com as informações de diagnóstico geradas.
Hna4	-	HNA	Função Auxiliar	Rotas HNA em IPv4 a serem divulgadas pelo nó.
Hna6	-	HNA	Função Auxiliar	Rotas HNA em IPv6 a serem divulgadas pelo nó.
AllowNoInt	yes	Funcionamento Global do *OLSRD*	-	Sobre a permissão de o protocolo OLSR funcionar mesmo sem interfaces de rede.

TosValue	16	Funcionamento Global do *OLSRD*	-	Valor ToS para ser utilizado pelo tráfego de controle do OLSR.
Willingness	3	Parâmetro Global do Protocolo OLSR	Funcionalidade Núcleo	Valor que indica a inclinação do nó a transmitir informação por outros nós.
UseHysteresis	yes	Histerese	Função Auxiliar	Indica se o mecanismo de histerese será utilizado.
HystScaling	0.50	Histerese	Função Auxiliar	Parâmetro do mecanismo de histerese.
HystThrHigh	0.80	Histerese	Função Auxiliar	Parâmetro do mecanismo de histerese.
HystThrLow	0.30	Histerese	Função Auxiliar	Parâmetro do mecanismo de histerese.

Pollrate	0.05	Funcionamento Global do *OLSRD*	-	Freqüência com que a entidade **Agendador** funcionará.
TcRedundancy	0	Redundância TC	Função Auxiliar	Parâmetro do mecanismo de Redundância TC.
MprCoverage	1	Redundância MPR	Função Auxiliar	Parâmetro do mecanismo de Redundância MPR.
LoadPlugin	-	Funcionamento Global do *OLSRD*	-	Diretivas do *plugin* a ser carregado pelo *OLSRD*.
Parâmetros por Interface				
Ip4Broadcast	-	Funcionamento da Interface OLSR.	-	Endereço de *broadcast* IPv4 a ser utilizado pela interface.
Ip6AddrType	site-local	Funcionamento da Interface OLSR.	-	Escopo IPv6 a ser utilizado pela interface no caso de funcionamento em IPv6. Pode ser site-local ou global.

Ip6MulticastSite	ff05::11	Funcionamento da Interface OLSR.	-	Endereço *multicast* IPv6 a ser utilizado pela interface no caso de funcionamento site-local em IPv6.
Ip6MulticastGlobal	ff0e::1	Funcionamento da Interface OLSR.	-	Endereço *multicast* IPv6 a ser utilizado pela interface no caso de funcionamento global em IPv6.
HelloInterval	2.0	Mensagens HELLO	Funcionalidade Núcleo	Intervalo da Emissão da Mensagem.
HelloValityTime	6.0	Mensagens HELLO	Funcionalidade Núcleo	Validade da mensagem HELLO a ser divulgada.
TcInterval	5.0	Mensagens TC	Funcionalidade Núcleo	Intervalo de Emissão de Mensagens TC.
TcValityTime	15.0	Mensagens TC	Funcionalidade Núcleo	Validade da mensagem TC a ser divulgada.

MidInterval	5.0	Mensagens MID	Funcionalidade Núcleo	Intervalo de Emissão de Mensagens MID.
MidValityTime	15.0	Mensagens MID	Funcionalidade Núcleo	Validade da mensagem MID a ser divulgada.
HnaInterval	5.0	Mensagens HNA	Função Auxiliar	Intervalo de Emissão de Mensagens HNA.
HnaValityTime	15.0	Mensagens HNA	Função Auxiliar	Validade da mensagem HNA a ser divulgada.

Finalmente, então, observa-se que o *OLSRD* foi implementado seguindo a estrutura demonstrada na Figura 3.1. São nela definidas as entidades ***Parser* de *Sockets***, ***Parser* de Pacotes**, **Repositórios de Informação** e **Agendador**.

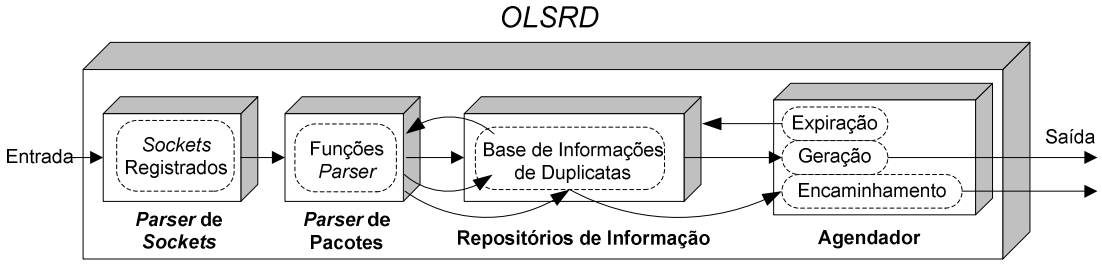

Figura 3.1 – Estrutura do *OLSRD*

3.2.1. *Parser* de *Sockets*

Como explicitado da Figura 3.1, todo tráfego OLSR é primeiramente tratado pelo ***Parser* de *Sockets***. Essa entidade é responsável por escutar por informação em um dado conjunto de *sockets* [10]. Estes podem ser registrados a qualquer momento e, para efetuar o registro de um, basta fazer uma chamada para função **add_olsr_socket**. Para a remoção a chamada deve ser feita para **remove_olsr_socket**. Em ambos os casos, deve-se passar, também, como parâmetro de registro, a função *parser* a ser atrelada ao *socket*. As funções **add_olsr_socket** e **remove_olsr_socket** estão prototipadas no arquivo fonte **socket_parser.h**, presente no já referenciado diretório **src**.

O esquema de funcionamento do ***Parser* de *Sockets*** pode ser visto na Figura 3.2.

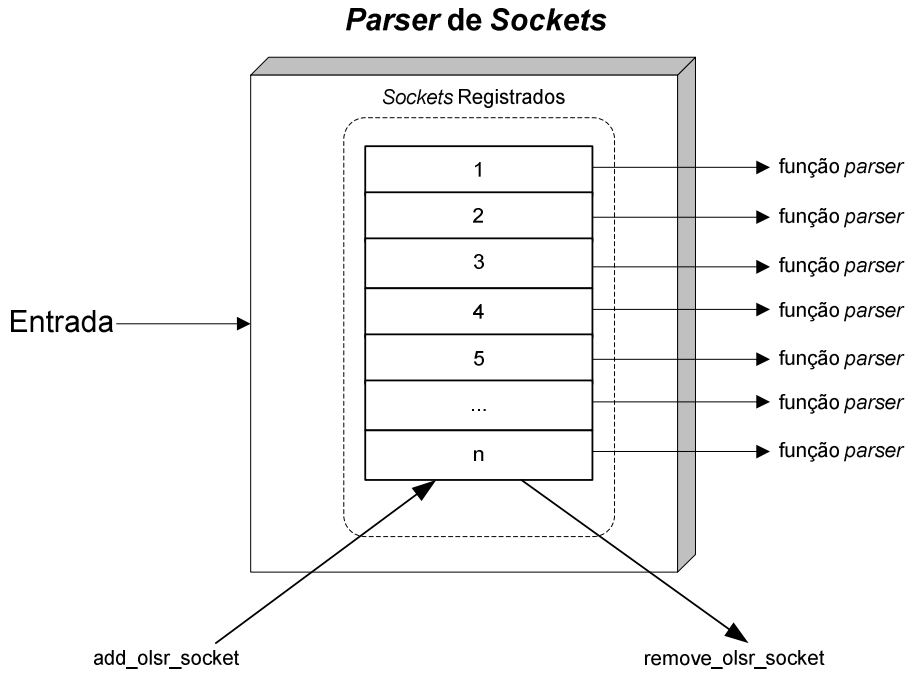

Figura 3.2 – *Parser* de *Sockets*

3.2.2. *Parser* de Pacotes

O *OLSRD* registra junto ao ***Parser* de *Sockets***, ao ser iniciado, todos os *sockets* relativos ao tráfego de controle. Esses *sockets* são registrados juntamente com a função *parser* a

ser chamada quando a informação estiver disponível.

O ***Parser* de Pacotes** recebe, então, do ***Parser* de *Sockets*** um pacote endereçado para sua respectiva função *parser*. Essa função *parser* transforma o pacote em questão em uma ou mais mensagens OLSR. Tais mensagens deverão ter uma função *parser* também registrada para serem processadas. Conclui-se disso que, no mínimo, as mensagens HELLO, TC e MID deverão ter suas funções *parser* registradas, pois são as mensagens de implementação e conseqüente processamento obrigatórios, segundo a funcionalidade núcleo da RFC 3626 [1]. Caso após o processamento da mensagem for decidido que ela também deva ser encaminhada, a função *parser* da mesma a enviará para o mecanismo de encaminhamento padrão. Mensagens que não tiverem função *parser* registrada em seu nome não são descartadas e sim enviadas também para o mecanismo de encaminhamento padrão, respeitando o que foi declarado no protocolo OLSR.

O esquema de funcionamento do ***Parser* de Pacotes** pode ser visto na Figura 3.3.

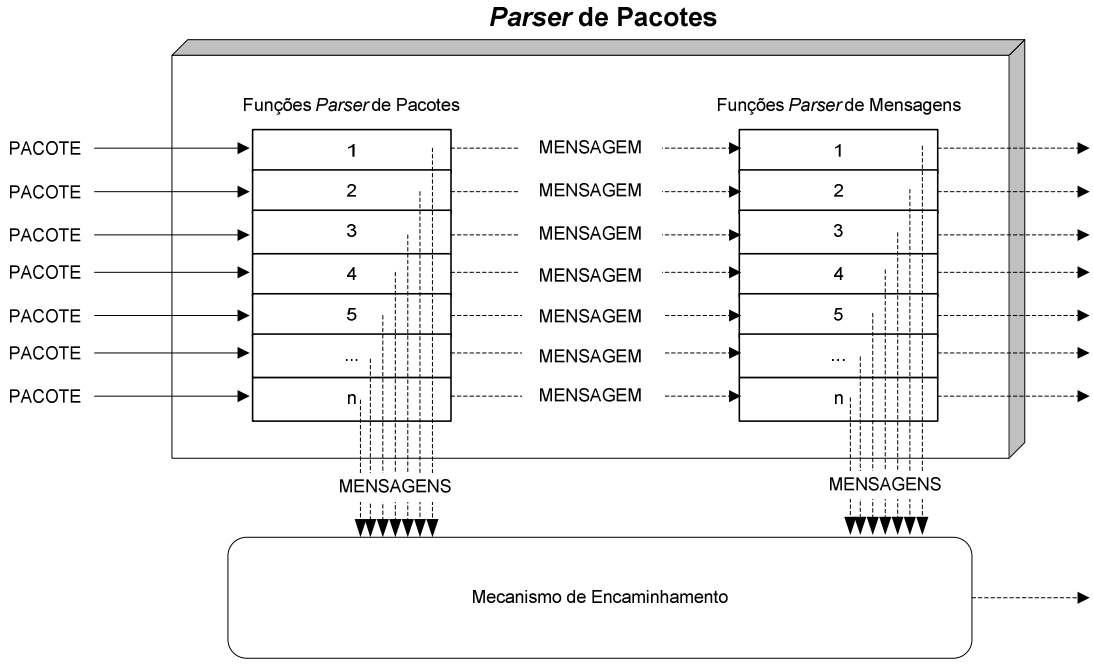

Figura 3.3 – *Parser* de Pacotes

3.2.3. Repositórios de Informação

Protocolos de roteamento regidos por tabelas têm como característica a dinamicidade do estado da informação. Isso provoca o recálculo constante da situação corrente do ambiente de rede. No caso do OLSR, o estado de enlace é compilado em tabelas que compõem os já descritos **repositórios de informação**. Como tais tabelas serão percorridas e atualizadas constantemente, justifica-se um projeto eficiente das estruturas de dados que as conterão.

Basicamente, o *OLSRD* optou por implementar os **repositórios de informação** do OLSR via listas encadeadas indexadas por *hashes*.

3.2.3.1. Listas Encadeadas

As listas encadeadas são uma das soluções mais simples para a estruturação de conjuntos de dados de tamanho dinâmico. Em uma lista encadeada, as entradas são armazenadas de modo que cada elemento aponta para o próximo. O último elemento poderá apontar de volta para o primeiro ou simplesmente para algum outro valor predeterminado (tipicamente *NULL* em um ambiente C).

Para a otimização do processo de eliminação de elementos, pode-se introduzir o conceito das listas encadeadas duplas. Nelas, em adição às referencias aos próximos elementos, cada entrada também aponta para o elemento anterior. Dessa forma, não é preciso o conhecimento de outras entradas da lista quando se objetiva a eliminação de uma.

Mais ainda, uma lista encadeada dupla poderá ser ordenada de maneira a minimizar o tempo de procura. Com os itens dispostos respeitando a ordem de algum valor das próprias entradas, a determinação da existência ou não de um elemento se torna mais célere, além de tornar trivial a procura por elementos menores ou maiores do que uma específica entrada, bastando, para isso, a travessia da lista para trás ou para frente, respectivamente.

3.2.3.2. *HASH*

O *hashing* de algum valor fonte é a transformação da informação original em um valor

usualmente menor e sempre de tamanho fixo. Esse valor ou chave representando o valor fonte é chamado de *hash*. Uma função *hash* é, neste contexto, responsável por, a partir da informação original, gerar o seu *hash*.

É desejável que funções *hash* originem sempre *hashes* diferentes para entradas diferentes. Como, entretanto, uma função *hash* gera tipicamente chaves menores do que a origem, existe a probabilidade de valores diferentes darem origem à *hashes* iguais, provocando as chamadas colisões. O objetivo a ser almejado é, portanto, obter funções *hash* que tenham a probabilidade o mais baixa possível de gerar tais colisões. Nada obstante, não importando a probabilidade de ocorrência de colisões, uma mesma informação original deverá sempre gerar o mesmo *hash* ao ser submetida à uma função *hash* específica.

No contexto de estrutura de dados, os *hashes* são caracteristicamente utilizados como chaves para indexar itens. O tamanho tipicamente menor dos *hashes* em relação à informação original dinamiza o processo de pesquisa e manutenção nas bases de dados. Assim sendo, para identificar unicamente n elementos em uma estrutura de dados, basta selecionar uma função *hash* que produza, preferencialmente com nenhuma colisão, chaves com tamanho fixo e com valor variando entre 1 e n para os valores originais escolhidos das entradas.

3.2.3.3. Tabelas no *OLSRD*

Tradicionalmente, o protocolo OLSR não é voltado para o roteamento de redes muito grandes. Assim, as tabelas dos **repositórios de informação** apresentarão um número relativamente pequeno de entradas. Além disso, por uma característica de operação do próprio protocolo, as tabelas no *OLSRD* registram usualmente apenas uma entrada por cada nó. Tudo isso corrobora para uma implementação das bases de dados em tabelas com listas encadeadas duplas utilizando indexação por *hashes*.

O *OLSRD* emprega uma função *hash* nos endereços principais dos nós para indexar os mesmos em um *array* estaticamente alocado. Cada elemento do *array* é a entidade raiz de uma lista encadeada dupla. Essa estrutura pode ser visualizada na Figura 3.4.

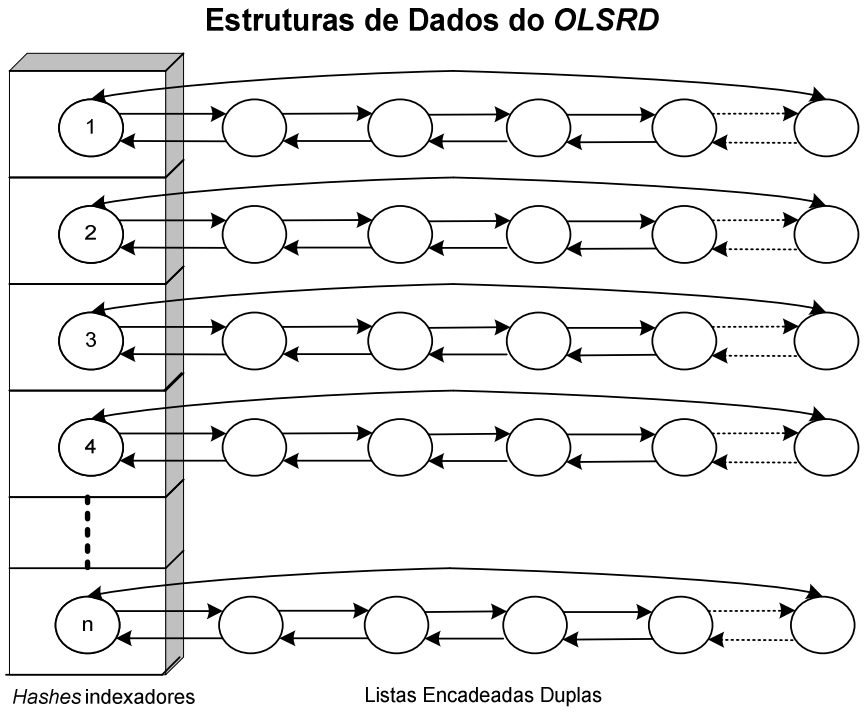

Figura 3.4 – Estruturas de Dados no *OLSRD*

A função *hash* utilizada é bastante simples. Como em endereços IP os bits menores compõem conhecidamente a parte mais única, é natural os empregar em uma função *hash*. No *OLSRD*, a função *hash* empregada é meramente a utilização dos últimos 5 menores bits do endereço IP. Isso, obviamente, poderá causar colisões se endereços similares em subredes diferentes forem utilizados. Isto, no entanto, não configura um problema, uma vez que o *hash* é apenas uma forma de tornar mais célere a busca. Todos os elementos podem, em uma situação crítica, compartilhar o mesmo *hash*, mas depois da busca do *hash*, ainda há a checagem do endereço IP principal do nó para a distinção definitiva dos elementos.

Dessa forma, para procurar elementos nas bases de dados, deverá ser calculado o *hash* do endereço IP buscado. Em seguida, deve-se acessar o elemento raiz da lista encadeada dupla referente ao *hash* e começar a percorrer tal lista buscando pelo endereço IP desejado. Na maioria dos casos, existirão apenas dois elementos em cada lista encadeada dupla: um fixo e vazio (utilizado apenas como índice) e um nó válido com o mesmo *hash* mas com o endereço IP buscado. Esse elemento fixo e vazio é apenas utilizado como raiz referenciadora, tornando

a inserção e exclusão de novos nós simples e rápida.

Como observação final a respeito dos **repositórios de informação** do *OLSRD*, vale ressaltar que todas as entradas constantes nas bases de dados são registradas com tempos de validade. Esses valores ditarão por quando tempo as entradas serão consideradas legítimas, e a entidade **Agendador** fará o papel de verificador da conformidade de tais valores.

3.2.4. Agendador

O **Agendador** do *OLSRD* é a entidade que realiza as tarefas registradas em intervalos regulares. O intervalo com que o **Agendador** irá funcionar é definido pelo parâmetro **Pollrate** do arquivo de configuração (ver Tabela 3.1). Esse intervalo deve ser configurado para possibilitar a execução de todas as tarefas regulares em um tempo de processamento menor que o próprio intervalo. Em se utilizando os valores sugeridos pela RFC para os tempos de emissão e validade das mensagens, o intervalo padrão de 0.05 segundo é considerado seguro.

De forma similar às entidades ***Parser* de *Sockets*** e ***Parser* de Pacotes**, o **Agendador** também foi projetado de maneira modular. Pode-se, segundo esse paradigma, registrar e remover funções no **Agendador** em tempo de execução. Conformemente, entidades que queiram ter alguma tarefa sendo realizada em intervalos regulares poderão registrar uma função correspondente, utilizando, para isso, ou a função **olsr_register_scheduler_event** ou a **olsr_register_timeout_function**. A primeira registrará uma função a ser chamada em intervalos definidos nos próprios parâmetros de registro, e a segunda registrará uma função a ser executada em todos os ciclos do **Agendador**. Ambas, entretanto, estão prototipadas no arquivo fonte **sheduler.h**, presente no diretório ***src***.

A cada ciclo do **Agendador**, todas as funções registradas são checadas em seus temporizadores para determinar se deverão ou não ser executadas. Para aquelas com os temporizadores expirados, suas chamadas são feitas e seus temporizadores reiniciados. Em seguida, todas as funções registradas para serem executadas a cada ciclo são chamadas e, se o caso for, as mudanças de topologia e vizinhança são processadas. Por fim, todo o tempo gasto para processar o ciclo é calculado e o **Agendador** espera por (**Pollrate** - tempo gasto) para iniciar o próximo ciclo. A Figura 3.5 resume o funcionamento da entidade.

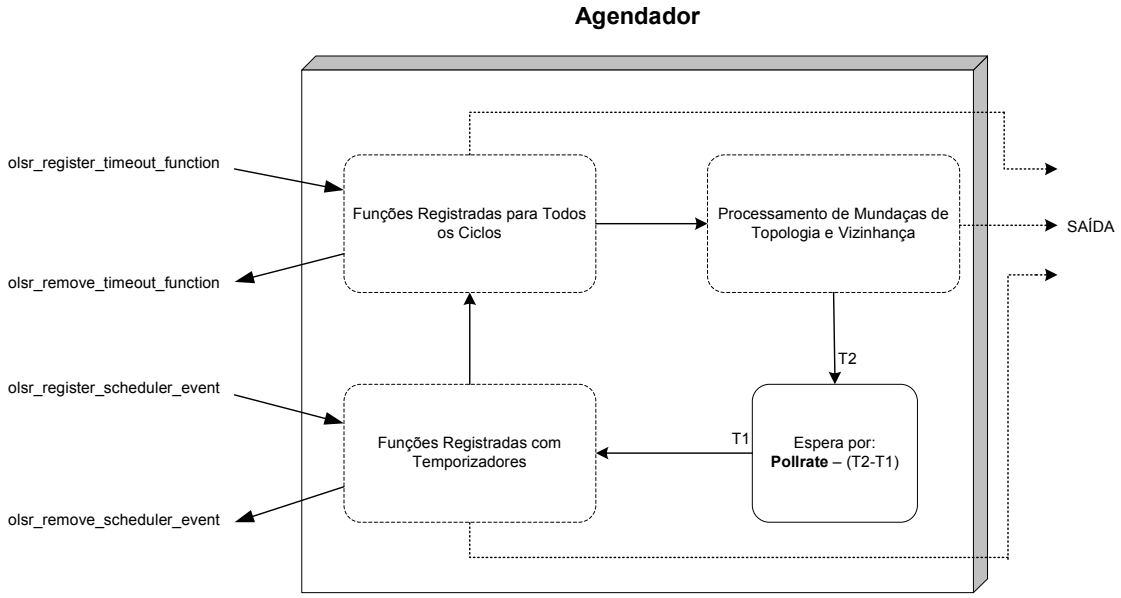

Figura 3.5 – Agendador

3.3. EXTENSÃO *LINK QUALITY*

Com a estrutura descrita até o momento, o *OLSRD* é capaz de operar segundo as especificações da RFC 3626 [1]. Todavia, com a crescente popularidade e conseqüente disseminação desta implementação, experimentos e testes de desempenho passaram a ser realizados pela comunidade com o intuito de avaliar o protocolo.

Especialmente no *workshop* proposto na conferência *Wizards of OS III* [11], verificou-se que, ao utilizar o protocolo OLSR em uma nuvem com mais de 25 nós, problemas importantes emergem. São eles:

- tabelas de roteamento demoram para serem construídas e são facilmente e rapidamente perdidas,
- rotas podem se tornar extremamente instáveis,
- rotas em *loop* podem surgir,
- performance de transmissão é notavelmente baixa.

Estudos, então, foram processados na operação do *OLSRD* para investigar as causas de tais dificuldades. As seguintes conclusões foram alcançadas:

- o mecanismo de histerese freqüentemente tira os MPRs da tabela de roteamento. Isso interfere no mecanismo de encaminhamento via MPRs, causando o próprio recálculo da estrutura MPR e das tabelas de roteamento;
- o mecanismo de seleção MPR pode eleger nós distantes como MPRs para reduzir e otimizar o próprio número de MPRs. Isso ocasionalmente gera MPRs com enlaces fracos, possibilitando a constante retirada e inclusão dos mesmos e a conseqüente operação de recálculo de MPRs e tabelas de roteamento;
- o mecanismo de seleção MPR, ao se tornar instável, interfere gravemente no mecanismo de controle de topologia. O entendimento do ambiente OLSR pode, por conseguinte, perder a sincronia, possibilitando o aparecimento de rotas em *loop*;
- essa instabilidade no cálculo das rotas e a possibilidade de existirem *loops* impacta seriamente no desempenho de transmissão do OLSR. Mais ainda, o fato de que este é um protocolo onde a decisão de rotas é feita com base na menor métrica, a escolha de rotas de pouca qualidade mas com métrica baixa em detrimento de rotas boas mas com métrica alta é uma constante e piora a já crítica performance de transmissão.

Basicamente, portanto, o que estava causando os problemas na implementação do protocolo OLSR eram o mecanismo MPR e a técnica de histerese. Tais características, todavia, são justamente as otimizações propostas pela RFC 3626 [1] que supostamente tornariam o protocolo recomendável para MANET e justificariam a própria denominação do OLSR.

Para evitar o descarte das otimizações do protocolo, e com o intuito de elevar a performance do *OLSRD*, a comunidade criou, conseguintemente, uma extensão para o protocolo chamada *Link Quality*.

3.3.1. Teoria

A extensão *Link Quality* descrita em [12] objetiva a alteração do critério de seleção dos enlaces. Numa implementação puramente conforme com a RFC, a escolha é sempre feita minimizando o número de saltos entre os nós. Essa determinação usualmente elege um enlace com métrica baixa mas com qualidade também baixa. É desejável, no entanto, poder optar pelo enlace de maior qualidade, mesmo que sua métrica seja elevada.

Para alterar o paradigma da escolha dos enlaces é necessário, então, fornecer ao *OLSRD* a capacidade de avaliar a qualidade de um enlace. Isso é feito pela medição das perdas de pacotes OLSR vindos dos vizinhos. Sabe-se que vizinhos de um salto geram mensagens HELLO em intervalos regulares declarados (campo **Htime** – Figura 2.4). Assim, basta observar se durante o tempo esperado a mensagem HELLO é recebida.

Se, por exemplo, 2 de 10 mensagens HELLO são detectadas perdidas no caminho do vizinho para o nó em questão, tem-se uma percentagem de pacotes perdidos de 20% nesse sentido. Obviamente, tem-se, também, uma taxa de sucesso de 80%. Essa probabilidade de sucesso será, a partir de agora, chamada de qualidade do enlace, ou simplesmente LQ (*Link Quality*).

No contexto da extensão *Link Quality*, é importante quantificar a qualidade do enlace na direção oposta também. Define-se, conseqüentemente, como NLQ (*Neighbor Link Quality*) a qualidade do enlace observada pelo vizinho de um salto considerando o sentido do nó para o próprio vizinho de um salto.

Agora se pode determinar a probabilidade de que uma transmissão de ida e volta ocorra, ou seja, a taxa de sucesso de ao enviar uma mensagem para o vizinho, este envie de volta uma confirmação de recebimento que atinja o nó original. Esse será o percentual de que os dois pacotes atravessem ambos sentidos, ou simplesmente LQ x NLQ. Esta é, também, a probabilidade de que retransmissões não ocorram, pois se houve sucesso na ida e volta, não haverá necessidade de reenviar a transmissão.

Já é possível, neste momento, especificar o número de transmissões que serão tipicamente necessárias para que um pacote saia do nó e seja recebido pelo vizinho, ou para que este vizinho transmita algo que chegue ao nó original. Tal entidade é definida logicamente pelo

inverso da probabilidade de que não ocorram retransmissões, a saber, $\frac{1}{LQ \times NLQ}$.

O valor $\frac{1}{LQ \times NLQ}$ é, conseqüentemente, denominado ETX (*Expected Transmission Count*) e traduz a quantidade de transmissões esperadas para o caminho sendo analisado. O ETX vale para ambos os sentidos do enlace, já que a probabilidade envolvida no seu cálculo é a da transmissão de ida e volta, não importando a ordem dos fatores.

O ETX pode ser calculado para apenas enlaces ou ser extendido para rotas também. No caso das rotas, assume-se que o ETX traduzirá a quantidade de transmissões esperadas para que se atinja o ultimo nó da rota. Infere-se disso que o ETX da rota será basicamente a soma dos ETX de cada enlace envolvido no caminho. Isso se deve ao fato de que para alcançar o nó final, deverá se transmitir o número de vezes necessárias para se chegar ao primeiro nó intermediário e continuar o processo até o destino, adicionando as transmissões esperadas para cada etapa galgada.

Uma discussão mais profunda sobre a teoria envolvida no uso da métrica ETX pode ser encontrada em [13] e [14].

3.3.2. Funcionamento

Com as entidades LQ, NLQ e ETX definidas, para que a extensão *Link Quality* possa operar, é fundamental, também, disponibilizar ao *OLSRD* ferramentas para calcular tais valores e implementar a utilização dos mesmos no funcionamento do protocolo.

Para a entidade LQ, como já foi descrito, o cálculo é feito pela determinação da quantidade de mensagens HELLO perdidas em um escopo tomado. Para decidir se uma mensagem HELLO foi perdida, o campo **Htime** (Figura 2.4) é utilizado como marcador do tempo limite para que a mensagem chegue.

Para o NLQ, por ser um valor calculado localmente no vizinho de um salto, um mecanismo de difusão da entidade para as outras pontas do enlace deve ser utilizado. Para isso, a extensão *Link Quality* introduz um novo tipo de mensagem HELLO: a chamada LQ HELLO. Essa mensagem oferece um campo adicional onde o valor de LQ percebido é incluído para cada

enlace divulgado. Esse LQ divulgado é, pela perspectiva do nó que recebe a mensagem, o NLQ buscado.

A mensagem LQ HELLO permite, agora, o cálculo do NLQ e conseqüentemente do ETX para os vizinhos de um salto do nó. Resta, entretanto, impetrar recursos para a obtenção dos ETX das rotas.

O cálculo do ETX de uma rota resume-se em somar os ETX de todos os enlaces incluídos no caminho. Porém, com a estrutura exposta até aqui, dispõe-se somente do primeiro ETX da rota.

No contexto da funcionalidade núcleo do OLSR, mais especificamente no mecanismo de Controle de Topologia, as mensagens TC são utilizadas para divulgar em todo o ambiente OLSR o estado de enlace de cada nó. Tal informação é imprescindível para o mecanismo de cálculo de rotas. A extensão *Link Quality*, aproveitando o objetivo das mensagens TC e seu escopo de difusão, adicionou a informação de LQ nas mensagens TC, introduzindo, assim, o novo tipo de mensagem TC: a chamada LQ TC. Agora, dispõe-se de toda a informação necessária para introduzir o novo paradigma de eleição de enlaces, primando, a partir deste momento, pela qualidade total e não pela minimização da métrica.

Infelizmente, por alterar as mensagens HELLO e TC para LQ HELLO e LQ TC, a extensão *Link Quality* quebra a conformidade com a RFC 3626 [1]. Dessa forma, para aproveitar as vantagens oferecidas pelo paradigma da qualidade dos enlaces, o ambiente OLSR deve ter todos os seus nós funcionando somente no novo modo de operação.

Basicamente, a extensão *Link Quality* oferece dois modos de funcionamento controlados pelo parâmetro de configuração **LinkQualityLevel**. Esse é um parâmetro adicional presente no arquivo de configuração do *OLSRD* e soma-se àqueles apresentados na Tabela 3.1. Suas possibilidades de marcação e as conseqüentes características são:

- **0:** Marcando o parâmetro como 0, a extensão *Link Quality* é desabilitada. O *OLSRD* irá funcionar de acordo com a RFC, utilizará as mensagens HELLO e TC oficiais e calculará as rotas minimizando a contagem de saltos.
- **1:** Ao configurar como 1 o **LinkQualityLevel**, o *OLSRD* ativará a extensão *Link Quality* para a seleção de MPRs. Aqui, o mecanismo de eleição os MPR utilizará as informações de qualidade de enlace para escolher os nós que atuarão como MPRs. Um vizinho

de um salto será um MPR somente se o ETX da rota do nó original até o vizinho de dois saltos, tendo esse vizinho de salto como nó intermediário, for o menor encontrado. As mensagens LQ HELLO e LQ TC substituirão respectivamente as mensagens HELLO e TC.

- **2:** Finalmente, ao utilizar o valor 2 como diretiva do **LinkQualityLevel**, o *OLSRD* não somente irá utilizar as informações de qualidade de enlace para eleger os MPR, mas irá também calcular sua tabela de rotas considerando os ETX envolvidos. Nesse paradigma, uma rota é somente adicionada à tabela se ela for aquela que apresentar o menor valor de ETX acumulado até o destino considerado. As mensagens HELLO e TC também são substituídas pelas suas equivalentes LQ HELLO e LQ TC.

Somando-se ao parâmetro **LinkQualityLevel**, existem outros também introduzidos no arquivo de configuração para customizar a operação da extensão *Link Quality*. Todavia, eles só são considerados pelo *OLSRD* se o valor de **LinkQualityLevel** implicar na operação da extensão. A Tabela 3.2 apresenta os parâmetros exclusivos da extensão *Link Quality*, sugere seus valores iniciais e oferece uma breve descrição de suas funções.

Tabela 3.2 – Parâmetros do Arquivo de Configuração do *OLSRD* - Extensão *Link Quality.*

PARÂMETRO	**VALOR SUGERIDO**	**DESCRIÇAO**
LinkQualityLevel	2	Parâmetro que define a utilização ou não da extensão *Link Quality*. Define, também, o seu modo de operação. Seus valores possíveis são: 0: Extensão desabilitada – funcionamento RFC 1: Extensão habilitada – MPRs são eleitos segundo o quesito ETX. 2: Extensão habilitada – MPRs e rotas são selecionas segundo o quesito ETX.
LinkQualityWinSize	10	Escopo a ser considerado no cálculo da entidade LQ. Gera a seguinte fórmula: LQ = (pacotes OLSR recebidos com sucesso) / LinkQualityWinSize

LinkQualityFishEye	1	As mensagens TC, neste caso LQ TC, incorrem, ao serem recebidas pelos nós, em uma reavaliação das tabelas de rotas do **conjunto MPR**. Para, então, evitar que mudanças pequenas locais provoquem processamento desnecessário em nós afastados, o algoritmo *Fish Eye* foi introduzido. A idéia é, ao invés de utilizar o TTL padrão de 255 para todas as mensagens LQ TC geradas, utilizar TTL variados, aumentando a quantidade de mensagens com TTL pequenos e diminuindo a de TTL maiores. Dessa forma, somente algumas mensagens LQ TC serão transmitidas para toda a rede. O esquema implementado pelo *OLSRD* ao ativar o *Fish Eye* é gerar mensagens LQ TC com os campos TTL da seguinte forma rotativa: 255 3 2 1 2 1 1 3 2 1 2 1 1 Dessa forma, a cada 13 mensagens geradas, 6 têm TTL 1, 4 têm TTL 2, 2 têm TTL 3 e apenas 1 tem TTL 255. Esse algoritmo, além de preservar o nó de mudanças que provavelmente não o influenciarão, também reduz o tráfego de controle na rede. 0: Desabilita o algoritmo *Fish Eye* 1: Habilita o algoritmo *Fish Eye*

LinkQualityDijkstraLimit	3 5.0	À semelhança do algoritmo *Fish Eye*, o parâmetro **LinkQualityDijkstraLimit** também objetiva a redução da importância de mensagens LQ TC oriundas de nós distantes para a economia de processamento local. Composto de dois valores, o **LinkQualityDijkstraLimit** dita ao *OLSRD* em quais momentos é adequado recalcular a tabela de rotas. O primeiro valor é um inteiro e permite ao *OLSRD* recalcular a tabela somente se uma mensagem LQ TC chegar oriunda de um nó distante não mais que este inteiro de saltos. O segundo valor é o intervalo de tempo em que o recálculo será realizado de qualquer forma, não importando a chegada ou não de LQ TC adequados.

Em conclusão, ao se configurar o parâmetro **LinkQualityLevel** com valores maiores do que 0, a extensão *Link Quality* é habilitada. Deve-se atentar para o fato de que, apesar de oferecer uma significativa melhora na performance do protocolo OLSR, a extensão não é compatível com a RFC. Portanto, para que o ambiente OLSR considerado mantenha o funcionamento esperado, todos os nós nele presentes deverão estar operando segundo o mesmo paradigma, seja ele compatível com a RFC ou com a extensão *Link Quality*.

3.4. INTERFACE *PLUGIN*

Introduzir a habilidade de se adicionar funcionalidades ou mesmo de se alterar a operação do

protocolo OLSR foi a motivação principal no projeto da interface *plugin* para o *OLSRD*. No caso específico da estrutura proposta nesta dissertação, tal interface foi uma das ferramentas fundamentais utilizadas para se incorporar a capacidade de gerência ao protocolo OLSR.

3.4.1. Teoria

Já foi exposto anteriormente que a modularidade é um dos mais importantes pilares do *OLSRD*. Neste contexto, os *plugins* se inserem propriamente, se utilizando e conferindo mais compartibilização para a implementação.

Plugins são programas que se acoplam à uma aplicação pré-existente e a ela fornecem novas funções sem interferirem no código original. A interface *plugin* do *OLSRD* é, então, o que especifica e molda a maneira como esses *plugins* irão ser desenvolvidos e utilizados [15].

Definiu-se que no *OLSRD* os *plugins* seriam bibliotecas dinamicamente carregadas, ou DLLs (*Dynamic Linked Libraries*). DLLs são tipicamente coleções de funções compartilhadas com um ou vários processos e que podem ser por eles serem acessadas, interceptando o seu fluxo de execução e adicionando a ele novas operações. Esse paradigma pode ser visto na Figura 3.6.

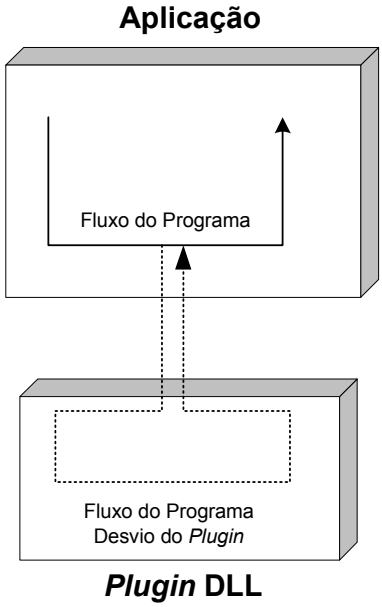

Figura 3.6 – Paradigma de Funcionamento dos *Plugins* como DLLs.

A concepção de DLLs existe para todos os sistemas operacionais comuns. No ambiente Linux, elas são conhecidas como os arquivos de extensão **.so**, e no Microsoft Windows recebem a extensão **.DLL**.

Assim sendo, o conceito de *plugins* como DLLs foi escolhido no *OLSRD* pelas seguintes razões:

- Com os *plugins* não há necessidade de nenhuma mudança no código do *OLSRD* para se adicionar pacotes customizados ou funcionalidades extras;
- A comunidade é livre para implementar *plugins* e licenciá-los sob os termos de escolha. Como o *OLSRD* em si está sob os termos da licença de código aberto BSD [9], qualquer alteração em seu código teria que seguir o que foi especificado na licença;
- Os *plugins* podem ser escritos em qualquer linguagem de programação que possa ser compilada como uma biblioteca dinamicamente carregada (DLL);
- Não há a necessidade de se aplicar grandes correções nas funcionalidades extendidas do OLSR quando do lançamento de novas versões do *OLSRD*. A interface *plugin* sempre será compatível com versões anteriores.

3.4.2. Funcionamento

Durante o desenvolvimento de um *plugin*, deve-se compreender o escopo de abrangência da interface para avaliar o que poderá ser implementado com as ferramentas disponíveis.

É neste ponto que o projeto modular do *OLSRD* prova sua utilidade. Com a independência das unidades estruturais do *OLSRD*, a interface *plugin* pode permitir o acesso às estruturas e, conseqüentemente, ajustar a operação do *OLSRD* para acomodar as novas funcionalidades sob o comando dos *plugins*. A Figura 3.7 expõe, então, o padrão de intercomunicação dos mecanismos operacionais do *OLSRD* com um *plugin* qualquer.

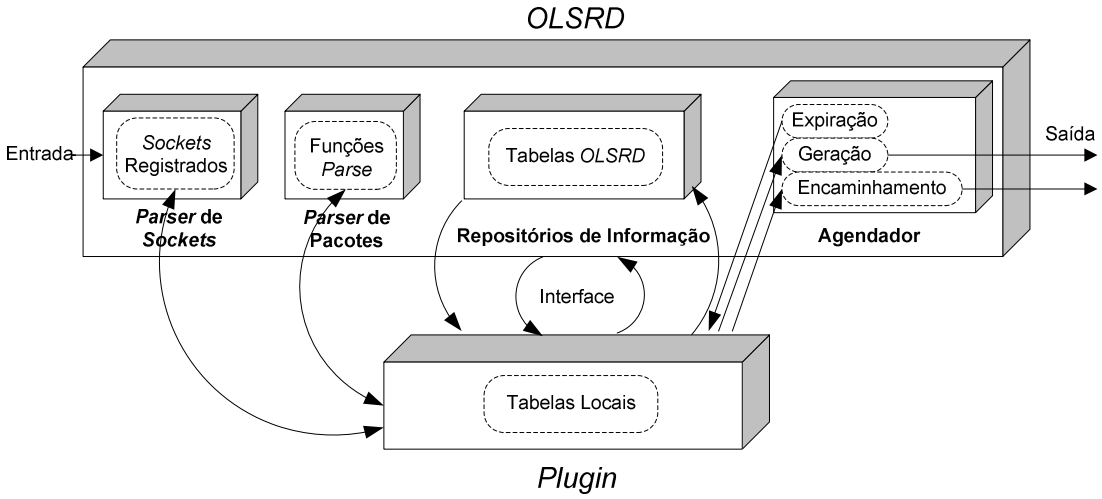

Figura 3.7 – Padrão de Intercomunicação entre *OLSRD* e *Plugin*

Assim, de acordo com a especificação da interface, um *plugin* poderá realizar as seguintes tarefas em cada entidade:

- ***Parser* de *Sockets*:** No ***Parser* de *Sockets***, um *plugin* poderá registrar e remover *sockets*.
- ***Parser* de Pacotes:** Aqui, poderão ser registradas e removidas funções para tratar pacotes ou tipos de mensagens. Isso é útil para introduzir ao protocolo novas mensagens que trarão funcionalidades diferentes.
- **Repositórios de Informação:** Todas as tabelas do *OLSRD* são acessíveis pelos *plugins*. Dessa maneira, os valores nelas presentes poderão ser alterados ou removidos sob a demanda externa.
- **Agendador:** Nesta entidade poderão ser registradas funções para serem executadas com temporizadores ou em todos os ciclos. Isso é vantajoso, por exemplo, ao se implementar novos tipos de mensagens, pois é neste local onde se pode gerar eventos, ou no caso em questão, transmissões em intervalos regulares. Também é possível, no **Agendador**, acessar a função de transmissão do *OLSRD*, permitindo o envio de informação sob o comando do *plugin*.

Resumindo, à interface *plugin* é permitido o acesso a virtualmente todas as funções do *OLSRD*. Um *plugin* genérico deverá, destarte, selecionar aquelas que servirão ao seu

propósito pré-definido e as utilizar em seu código.

Finalmente, para iniciar a execução de um *plugin* específico, este deverá ser compilado e instalado no sistema operacional como uma biblioteca dinamicamente carregada. Em seguida, antes da inicialização do *OLSRD*, a diretiva **LoadPlugin** do arquivo de configuração (Tabela 3.1) deverá apontar para o nome do *plugin* desejado, discriminando os parâmetros específicos a serem enviados e, conseqüentemente, utilizados pela DLL.

4. GERENCIAMENTO DE REDES

O objetivo proposto nesta dissertação é o da criação e implementação de uma estrutura de gerenciamento para o protocolo OLSR. Tal estrutura foi conscientemente concebida sobre as bases de um modelo de gerenciamento de redes já consolidado e utilizado na comunidade: a infra-estrutura SNMP.

Neste capítulo será delineada brevemente a infra-estrutura SNMP. E, dessa forma, os alicerces teóricos da estrutura de gerenciamento para o protocolo OLSR estarão cobertos previamente à sua apresentação.

4.1. INFRA-ESTRUTURA SNMP

Originalmente, muitas redes possuíam protocolos de gerenciamento próprios, usualmente na camada de enlace. Se, portanto, um dispositivo contido em tais redes apresentasse problemas, um administrador poderia, de um dispositivo vizinho semelhante, enviar controles especiais para tentar solucionar o problema. Como tais ferramentas de gerenciamento faziam parte de protocolos de camadas mais baixas, como a de enlace, a gerência poderia ser realizada mesmo com falha em protocolos superiores.

Entretanto, diferentemente de redes homogêneas, a internet não possui um único protocolo de enlace nem um único tipo de dispositivo. Logo, elaborar um sistema de gerenciamento padrão deverá levar em conta essa heterogeneidade.

Conforme esse paradigma, idealizou-se a infra-estrutura SNMP. Localizada na camada de aplicação, ela permite o gerenciamento dos dispositivos diversos sem a preocupação com os protocolos das camadas inferiores. Assim sendo, é possível gerenciar dispositivos em redes distintas, pois o roteamento presente na camada de rede será responsável por transmitir os comandos entre os domínios de *broadcast*. Mais ainda, como tais comandos de gerenciamento são padronizados, equipamentos diferentes poderão ser controlados da mesma maneira, facilitando a elaboração de *softwares* de gerenciamento.

A arquitetura da infra-estrutura SNMP define um modelo cliente-servidor. Aqui, dispositivos que permitem gerenciamento serão servidores e possuirão *softwares* definidos como **agentes**.

Os dispositivos que, por sua vez, forem realizar a tarefa de gerenciamento propriamente dita terão os *softwares* clientes, definidos agora como **gerentes**. Portanto, uma atitude gerencial irá partir de um **gerente**, que enviará um comando padrão para o **agente** e este realizará a requisição, enviando de volta o resultado.

A infra-estrutura SNMP divide a atividade de gerenciamento de redes em duas partes específicas: a primeira se preocupa com a informação sendo gerenciada e a segunda diz respeito ao modo como essa informação é trocada entre os dispositivos [16].

Para a primeira parte é definido um padrão chamado MIB (*Management Information Base*), constituído das especificações necessárias para definir a informação pela qual um dispositivo deverá ser responsável.

Já para a segunda parte, protocolos de comunicação são especificados para determinar como a manipulação das informações definidas na MIB será realizada. Neste contexto, consolidam-se os protocolos SNMP e AgentX.

Expor-se-á, agora, os componentes de ambas as partes seguindo a ordem:

- MIB,
- SNMP,
- AgentX.

4.1.1. MIB

Um dispositivo sendo gerenciado deverá manter informação a ser acessada pelo **gerente**. Deve-se, destarte, definir o que especificamente poderá ser consultado e/ou alterado. Conhecido como MIB (*Management Information Base*), esse padrão explicita os objetos a serem implementados pelo dispositivo. Define-se, também, que operações serão permitidas para cada objeto bem como o significado dos mesmos.

Originalmente, foi definida uma RFC para uma única e extensa MIB. Nela constavam todos os objetos possíveis de serem implementados por algum dispositivo a ser gerenciado, ou seja, por um *software* **agente**. Era a chamada MIB-I, descrita na RFC 1156 [17]. Contudo, com a evolução da atividade de gerência de redes, a comunidade decidiu que uma abordagem

diferente era necessária. Surgiu a MIB-II, descrita na RFC 1213 [18]. Agora, as variáveis ou objetos seriam definidos em RFCs separadas para cada grupo lógico, dependendo do tipo de dispositivo envolvido. Isso permitiu, também, que diferentes fabricantes pudessem definir suas RFCs de MIBs proprietárias para seus produtos específicos. Cada nova MIB, todavia, foi e deverá ser escrita utilizando o mesmo padrão, garantido compatibilidade e uniformidade na integração dos *softwares*.

4.1.1.1. SMI

Enquanto a MIB é responsável por delinear os objetos a serem implementados e seus significados, existe um padrão separado que especifica as regras a serem utilizadas para definir e especificar tais variáveis. Essa norma é conhecida como SMI (*Structure of Management Information*) e foi declarada na RFC 1155 [19] em sua primeira versão, seguida pela RFC 2578 [20] para sua segunda versão. Com o intuito de manter os protocolos de gerenciamento simples, a SMI introduz restrições aos tipos de variáveis permitidas na MIB, especifica regras para seus nomes, e, também, cria métodos para a definição de tipos de objetos. Dessa forma, como exemplo, se em uma MIB for necessário especificar um objeto que seja um endereço de rede IP, este, pelo que foi instituído pela SMI, deverá ser definido como uma variável do tipo *IpAddress*, que por sua vez é especificada como uma *string* de 8 octetos e tem sua definição e referência escritas segundo a norma ASN.1.

4.1.1.2. ASN.1

Como já exposto acima, a SMI é o padrão que define como uma MIB deve ser elaborada. Um dos aspectos mais importantes da SMI é a institucionalização da utilização da norma ISO ASN.1 (*Abstract Syntax Notation 1*) para definição e referência de quaisquer variáveis MIB.

ASN.1 é uma linguagem formal que tem duas características principais: uma notação usada em documentos que humanos podem ler e uma representação compacta codificada da mesma informação utilizada em protocolos de comunicação. Em ambos os casos, a linguagem ASN.1 remove quaisquer ambigüidades possíveis de tanto a representação como do significado. Por exemplo, ao invés de dizer que um objeto é um inteiro, a definição via ASN.1 deve

estabelecer a forma e escopo exatos da variável. Essa precisão é especialmente importante quando implementações incluem ambientes heterogêneos que não compartilham das mesmas representações para itens de informação.

Além de manter documentos padrão livres de ambigüidades, a norma ASN.1 também auxilia na simplificação da implementação de protocolos de gerência de rede. Ela define precisamente como codificar nomes e valores em uma mensagem. E, uma vez que uma MIB foi escrita segundo a ASN.1, a forma legível para humanos pode ser traduzida direta e mecanicamente para a forma codificada utilizada em mensagens [16].

4.1.1.3. Estrutura e Representação dos Nomes de Objetos em MIBs

A norma ASN.1 especifica como representar tanto itens de informação quanto nomes. Porém, compreender os nomes utilizados para variáveis MIB requer uma contextualização das bases do **escopo de nomes**. Nomes utilizados para objetos em MIBs são tirados do identificador de objetos do **escopo de nomes** administrado pelo ISO e ITU. A idéia principal de tal identificador é prover um espaço de nomes em qual todos os possíveis objetos possam ser designados.

O identificador de objetos do **escopo de nomes** é absoluto, ou seja, os nomes são estruturados de forma a torná-los globalmente únicos. Seguindo a tendência de todos os escopos de nomes absolutos e extensos, este adota uma estrutura hierárquica. A autoridade de cada parte é subdividida em cada nível, permitindo aos grupos separados obter autoridade para designar os nomes de objetos abaixo de seu nível.

A raiz da hierarquia não é especificada, embora seus três descendentes diretos sejam gerenciados por: ISO, ITU e em conjunto pelo ISO e ITU. Para cada e todo descendente derivado é designado tanto um pequeno texto quanto um inteiro para identificação global. Os nomes ou pequenos textos têm como objetivo a compreensão fácil para os humanos, enquanto os números inteiros são otimizados para as representações por dispositivos computacionais.

O nome de uma variável ou objeto na hierarquia é a seqüência de números dos nós pelo caminho da raiz até o mesmo. A seqüência é escrita com pontos separando os componentes

individuais, sejam eles os números ou os nomes.

Para o padrão MIB foi designada uma sub-árvore da hierarquia. Assim, todas as variáveis ou objetos derivados correspondentes compartilharão o mesmo prefixo atribuído. Esse é **1.3.6.1.2.1**, ou, em notação por nomes**, iso.org.dod.internet.mgmt.mib**.

Como exemplo explicativo, toma-se a variável de MIB chamada *ipInReceives*. Ela está contida sob o nó *ip* abaixo do prefixo comum das variáveis MIB. Sua representação numérica se torna **1.3.6.1.2.1.4.3**, ou em notação por nomes, **iso.org.dod.internet.mgmt.mib.ip.ipInReceives**. A Figura 4.1 explicita tal estrutura, demonstrando o caminho que a define, indicando, também, em cinza o prefixo comum de todas as MIBs.

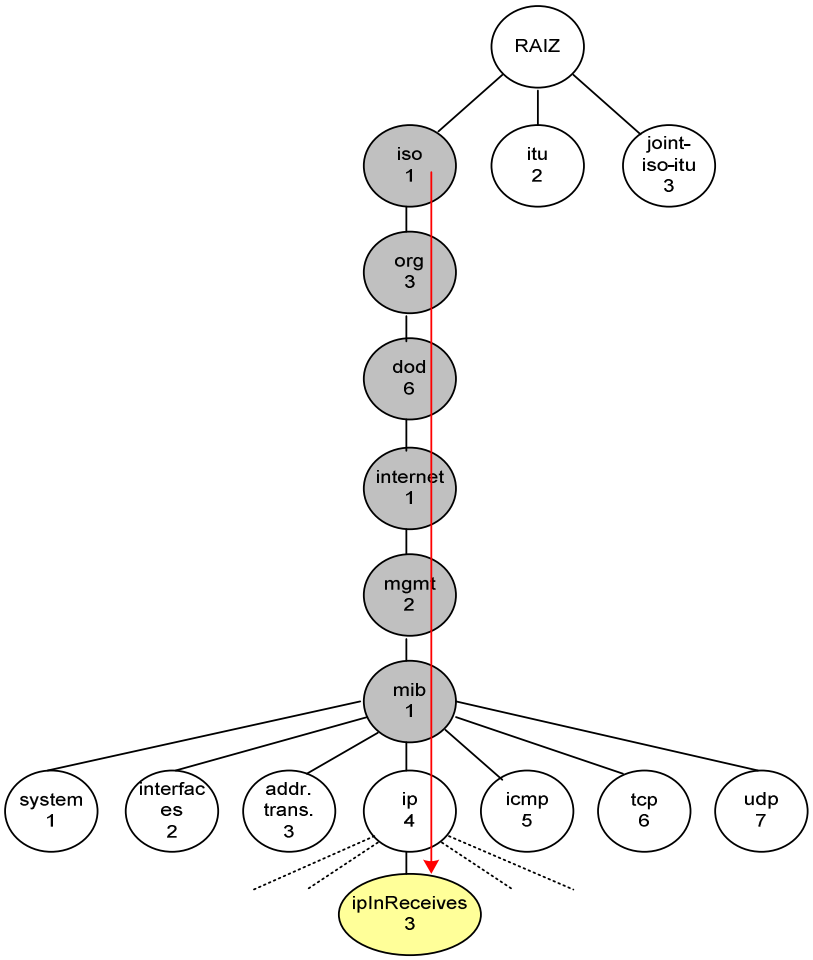

Figura 4.1 – Estrutura de Nomes da MIB

4.1.2. SNMP

Em contraposição com o padrão MIB, que define a essência dos objetos implementados pelos *softwares* **agentes**, os protocolos de comunicação tratam da manipulação remota de tais variáveis.

É neste contexto que se insere o SNMP (*Simple Network Management Protocol*). Este protocolo utiliza uma abordagem simplificada para permitir o gerenciamento das redes. Ao invés de implementar inúmeros comandos de controle, o SNMP basicamente dispõe de comandos de leitura e escrita. Ações complexas, como reiniciar um dispositivo, são realizadas por variáveis específicas. De tal modo, ao contrário de existir um comando particular para a reinicialização, um dispositivo pode ser desligado e ligado pela marcação de uma variável que determina o tempo para reinicio do sistema com o valor 0 segundos.

As principais vantagens do esquema leitura-marcação são a estabilidade, a simplicidade e a flexibilidade. O SNMP é notavelmente estável, pois suas definições se mantêm fixas, embora novos objetos sejam adicionados na MIB e novas operações gerenciais sejam derivadas dos efeitos da escrita em tais variáveis. A simplicidade vem com a facilidade de implementação, compreensão e análise, evitando a complexidade da existência de casos especiais para cada comando. Finalmente, o SNMP é flexível porque ele acomoda um número crescente e extenso de atitudes gerenciais com uma estrutura coesa de comandos simples.

A primeira abordagem ao protocolo SNMP, a chamada SNMPv1, surgiu em 1990 e está descrita na RFC 1157 [21]. Seguindo, uma nova versão foi apresentada com a SNMPv2, RFC 1901 [22], na qual foram adicionados novos tipos de mensagens e estruturas de dados. Um problema importante, entretanto, persistia em ambas as versões: a segurança. Tanto a SNMPv1 quanto a SNMPv2, apesar de implementarem uma autenticação básica via comunidade de acesso e escrita, eram suscetíveis à ataques de captura de pacotes, pois todas as informações trocadas eram feitas em texto claro. Surgiu, então, a mais recente versão, a SNMPv3, descrita nas RFCs 3411 a 3418. O SNMPv3 é baseado nos tipos de mensagens e estruturas de dados do SNMPv2. Todavia, seu esquema de segurança é totalmente diverso. No SNMPv3 não existem as comunidades de acesso e escrita, mas sim uma estrutura completa de autenticação, privacidade e controle de acesso. Atualmente, considera-se de segura utilização somente essa versão 3 do protocolo, enquanto as versões 1 e 2 estão marcadas como obsoletas. Contudo, muitos *softwares* **gerentes** e **agentes** implementam geralmente mais de

uma versão. Uma RFC trata de coexistência das versões do SNMP e está disponível em [23].

O protocolo SNMP, em todas as suas versões, opera na camada de aplicação. As operações permitidas em todas as versões podem ser visualizadas na Tabela 4.1. Apesar de mais de duas operações estarem definidas, deve-se compreender que na essência o protocolo SNMP baseia-se no paradigma simples da leitura e marcação.

Tabela 4.1 – Operações SNMP

OPERAÇÃO	**SIGNIFICADO**
SNMPv1, SNMPv2 e SNMPv3	
get-request	Usada para buscar uma informação.
get-next-request	Usada iterativamente para buscar seqüências de informação.
get-response	Usada pelo **agente** para responder às operações de leitura e marcação.
set-request	Usada para alterar o valor de algum objeto ou variável.
trap	Usada para informar eventos ou alertas assíncronos ocorridos no **agente**.
SNMPv2 e SNMPv3	
get-bulk-request	Uma operação idealizada para buscar grandes volumes de informação.
inform	Uma operação trap com aviso de recebimento

Em resumo, as operações *get-request* e *set-request* fornecem o esquema básico de busca e marcação dos objetos; *get-response* disponibiliza as respostas. Já a operação *trap* foca na notificação de eventos assíncronos, indicando para os **gerentes** quando acontecimentos não esperados ocorrem. O SNMP tipicamente utiliza o protocolo de transporte UDP para funcionar. As requisições são direcionadas para porta 161 dos *softwares* **agentes** e os *traps* são enviados para a porta 162 dos **gerentes**.

A Figura 4.2 resume o funcionamento da infra-estrutura SNMP incluindo as entidades discutidas até o momento. Como primeira etapa, o dispositivo cliente, em seu *software* **gerente**, faz uma consulta ao documento MIB, que por sua vez está escrito segundo a norma SMI. Essa primeira consulta local tem o objetivo de selecionar e compreender o objeto a ser manipulado remotamente. Em seguida, na segunda etapa, uma operação de leitura ou escrita é requisitada, pela rede, via protocolo SNMP, para o dispositivo servidor. Na terceira etapa, este dispositivo servidor, mais especificamente o *software* **agente**, realiza o que foi requisitado pela operação. Isso é feito pelo acesso de leitura ou escrita no repositório de objetos da MIB em questão. O acesso e controle de tal repositório é realizado pela implementação local das definições do documento MIB. Dessa forma, cada dispositivo específico pode ter sua implementação distinta da mesma MIB, bastando que as definições e objetivos definidos sejam respeitados. Finalmente, na quarta e última etapa, o resultado é encaminhado de volta para o dispositivo cliente, através de uma resposta SNMP.

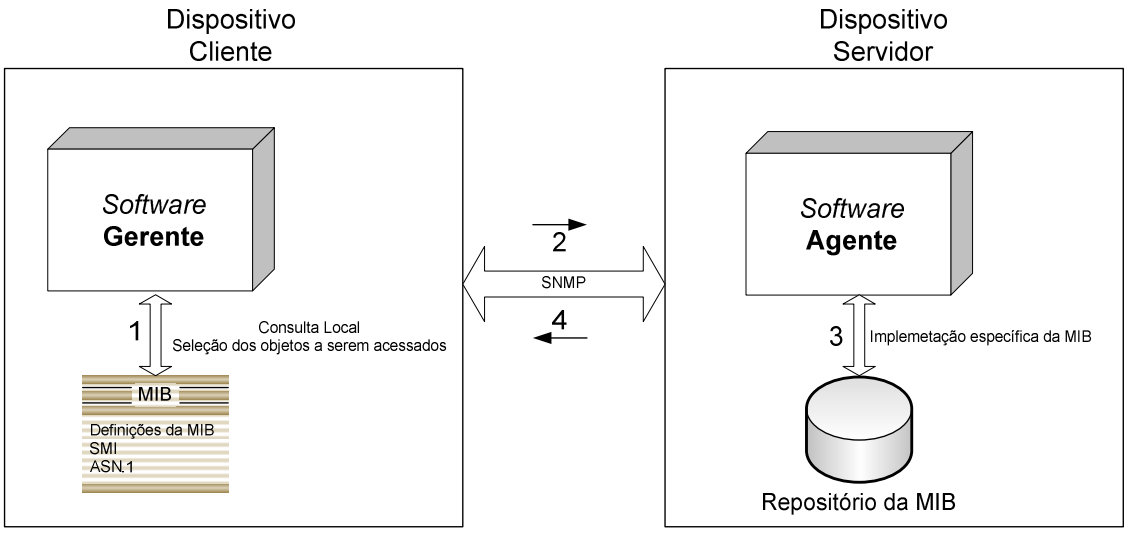

Figura 4.2 – Infra-estrutura SNMP

4.1.3. AgentX

A infra-estrutura composta do conjunto MIB e SNMP (Figura 4.2) oferece uma base competente para o gerenciamento dos objetos definidos na MIB padrão. No entanto, tais variáveis não são suficientes para o controle de qualquer dispositivo criado. Assim, um novo padrão foi proposto para acomodar a criação e manipulação de novos módulos MIB extendidos.

O padrão AgentX (*Agent Extensibility*), RFC 2741 [24], define, portanto, uma nova estrutura para agentes SNMP extensíveis. Ele caracteriza entidades chamadas **agentes máster** e **subagentes**, além de um protocolo de comunicação que rege a integração entre esses componentes.

No paradigma proposto pelo padrão AgentX, a entidade **agente máster** é responsável por todas as operações SNMP e pelo gerenciamento das MIBs padrão. Quando, porém, uma operação SNMP é requisitada para um objeto implementado por uma MIB não nativa, neste caso uma MIB extendida, esse **agente máster** irá encaminhar a requisição para o *software* **agente** responsável pela variável, fazendo uso para isso do protocolo AgentX. Este outro **agente**, agora definido como **subagente**, não necessitará implementar o protocolo SNMP, e todas as operações de requisição e resposta serão por ele realizadas via protocolo AgentX, de acordo com a RFC 2741 [24]. A Figura 4.3 mostra como o padrão AgentX complementa a infra-estrutura SNMP mostrada na Figura 4.2.

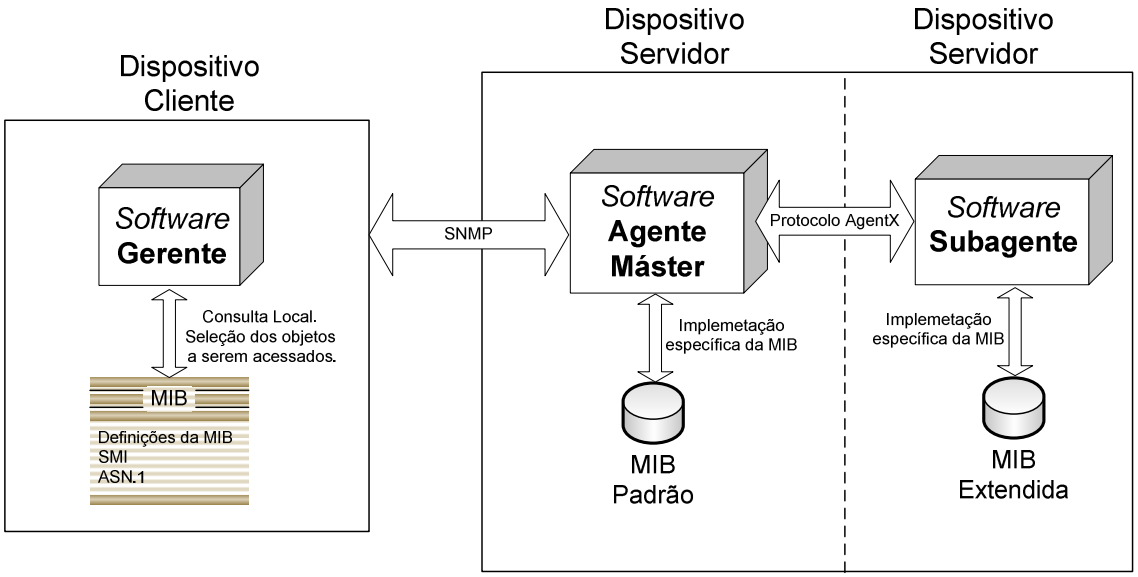

Figura 4.3 – Infra-estrutura SNMP incluindo o padrão AgentX.

Resumindo, quando o **subagente** é iniciado, ele se conecta ao **agente máster** e se registra. Essa conexão pode ser feita via a porta TCP 705 ou via *Unix-domain Sockets* no *endpoint* /var/agentx/master. A conexão via TCP é recomendada para quando o **subagente** se encontra em dispositivo separado do **agente máster** e a conexão via *Unix-domain Sockets* oferece mais performance para quando as entidades se encontram no mesmo dispositivo. Após a conexão e registro, quando o **agente máster** receber uma requisição SNMP relacionada com qualquer objeto da MIB extendida, esta operação será encaminhada pelo protocolo AgentX para o **subagente** específico responsável por tal variável. Esse **subagente** será responsável por acessar a MIB extendida e encaminhar de volta a resposta para o **agente máster** para que ela seja entregue para o requerente, completando, conseqüentemente, o ciclo proposto pela infra-estrutura completa de gerenciamento SNMP.

5. ESTRUTURA DE GERENCIAMENTO PARA O PROTOCOLO OLSR

Após a especificação de um padrão e, conseqüentemente, seguindo sua eventual implementação, uma necessidade básica se impõe: o gerenciamento. Gerenciar um protocolo, nesse contexto, envolve coletar informação e ainda alterar seu funcionamento em resposta à algum evento ou apenas para aprimorar sua performance.

É, então, com o objetivo de fornecer um ambiente para avaliação e controle que a estrutura de gerenciamento para o protocolo OLSR é proposta aqui. Composta basicamente de dois componentes, essa estrutura permite à qualquer aplicação ou administrador, locais ou remotos, a habilidade de acessar e manipular o domínio de roteamento OLSR.

O primeiro componente é a OLSR MIB. Esse documento, escrito segundo os padrões adequados à infra-estrutura SNMP já expostos no capítulo anterior, define e especifica o que é essencialmente importante em um ambiente OLSR. Assim, são nele compilados os objetos, suas descrições e operações derivadas que juntos criam um modelo conceitual de gerenciamento do protocolo OLSR.

Como segundo componente da estrutura é apresentada uma implementação da OLSR MIB. Essa implementação é um programa que materializa as definições apresentadas no documento da OLSR MIB. É por ela oferecido um meio de efetivamente acessar e manipular os objetos modelados, concretizando o conceito de administração.

Ambos os componentes da estrutura se apóiam nos pilares teóricos delineados nos capítulos anteriores. A exposição do OLSR, no capítulo dois, fundamenta como esse protocolo opera, possibilitando a compreensão da essência do que se objetiva gerenciar. Já no capítulo seguinte, onde se introduz o *OLSR Daemon*, ou *OLSRD*, apresenta-se a principal implementação do protocolo OLSR. Ela é peça fundamental na estrutura de gerenciamento uma vez que é sobre sua interface extensível que se acopla a implementação da OLSR MIB, além do fato de que seus parâmetros de funcionamento inspiraram a confecção da OLSR MIB. Finalmente, o capítulo 3 aborda o gerenciamento de redes, explanando os conceitos e padrões que foram utilizados por toda a estrutura de gerenciamento para o protocolo OLSR por esta dissertação oferecida.

5.1. ESTRUTURA

Faz-se necessário, agora, situar os componentes da estrutura de gerenciamento no contexto de um ambiente de administração. A Figura 5.1 expõe tal conjunto.

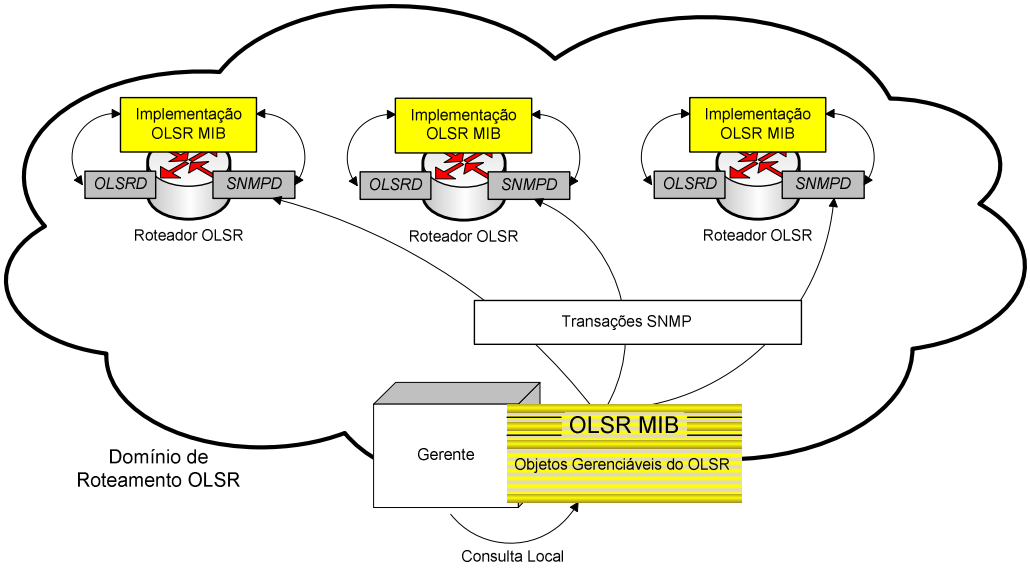

Figura 5.1 – Estrutura de Gerenciamento para o Protocolo OLSR.

Basicamente, o que ocorre segue o paradigma explicado no capítulo anterior da infra-estrutura SNMP. Um dispositivo **gerente**, que pode ser local ou remoto em relação ao roteador a ser gerenciado, seleciona do documento OLSR MIB o que deseja acessar ou gerenciar. Em seguida, operações SNMP são enviadas para o serviço SNMP, ou *SNMPD*, do roteador desejado e este, internamente, encaminha a requisição para a implementação da OLSR MIB. Esta, em seqüência, acessa o *OLSRD* e realiza o que foi demandando, enviando o resultado de volta.

Os detalhes da OLSR MIB e de sua implementação, que juntos compõem a estrutura de gerenciamento para o protocolo OLSR, serão apresentados adiante. Também, serão realizados experimentos com o objetivo de validar o ambiente de administração oferecido. Ao fim, serão

expostos os resultados. Dessa forma, a seguinte ordem será obedecida neste capítulo:

- OLSR MIB,
- Implementação da OLSR MIB,
- Experimentos,
- Resultados.

5.2. OLSR MIB

O primeiro componente da estrutura de gerenciamento para o protocolo OLSR é a OLSR MIB. Tendo sido escrito seguindo as normas ditadas pela SMI (*Structure of Management Information*), esse padrão compila os objetos que traduzem a essência do OLSR.

A definição de quais variáveis seriam incorporadas à OLSR MIB foi uma escolha importante. Era necessário que tais objetos gerenciados oferecessem uma visão completa da operação do protocolo, bem como permitissem a possibilidade de também alterar e controlar o funcionamento do mesmo.

Para a parte de avaliação da operação do protocolo, verifica-se que a própria RFC 3626 [1] e, conseqüentemente, o segundo capítulo desta dissertação expõem os repositórios de informação que compilam o estado momentâneo em que se encontra o dispositivo OLSR. Assim, essas informações foram modeladas em um formato adequado e incorporadas à OLSR MIB.

Já para a possibilidade de alteração e controle do protocolo, o *OLSRD* se provou de relevância fundamental. Como tal implementação já estava consolidada, com seu funcionamento validado pela comunidade, os parâmetros estáticos de seu arquivo de configuração (Tabela 3.1) serviram de inspiração para a modelagem dos objetos com suporte à escrita. Logo, tais variáveis foram formatadas e incorporadas à OLSR MIB. Vislumbrou-se, com isso, que a marcação de tais objetos permitiria a manipulação da operação do OLSR em tempo de execução, não apenas com a prévia escrita estática em arquivo de configuração do serviço de roteamento.

Outro aspecto importante da OLSR MIB foi a escolha de também acomodar objetos que

fossem relacionados com a extensão *Link Quality*. Apesar desta extensão não ser compatível com a RFC 3626 [1], sua utilização soluciona problemas de otimização do protocolo OLSR identificados na conferência *Wizards of OS III* [11], como já foi discutido no terceiro capítulo. Por conseguinte, decidiu-se incorporar à OLSR MIB os parâmetros de avaliação e controle à extensão *Link Quality* referentes, com a ressalva de que tais variáveis seriam somente válidas quando o protocolo OLSR estivesse operando nesse modo. Os objetos que traduzem o funcionamento do protocolo no modo *Link Quality* foram modelados a partir de informações de estado de operação do OLSR. Já os objetos que permitem a manipulação da execução do OLSR em extensão *Link Quality* foram inspirados nos parâmetros correspondentes do arquivo de configuração do *OLSRD* (Tabela 3.2). Para manter a conformidade da OLSR MIB tanto com a RFC do OLSR quanto com a extensão *Link Quality*, os objetos não em compatibilidade com a RFC foram assim assinalados em suas descrições. Os objetos que não possuem tais observações são totalmente compatíveis com a RFC 3626, e para a utilização da OLSR MIB com uma implementação estritamente condizente com a RFC basta não se considerar os objetos não conformes.

Finalmente, o documento padrão da OLSR MIB foi totalmente proposto por esta dissertação e se encontra disponível para consulta no APÊNDICE A. Nele estão presentes todos os objetos mencionados compilados em um formato adequado à norma SMI (*Structure of Management Information*).

5.2.1. Trabalhos Correlatos

É importante contextualizar que, à época em que a OLSR MIB desta dissertação foi proposta, já existia uma outra MIB também aplicável ao OLSR e apresentada em [31]. Contudo, tal MIB focava-se em uma arquitetura de gerenciamento baseada em dispositivos externos de coleta. Ou seja, as informações nela constantes deveriam ser obtidas por elementos observando o tráfego de rede em modo promíscuo. Isso, então, rendeu à MIB somente um aspecto de avaliação, e não também de controle do protocolo. Dessa forma, a OLSR MIB aqui proposta, por ser direcionada para a gerência direta do protocolo, não somente possui objetos com a possibilidade de leitura, mas compreende também objetos gerenciáveis, podendo, portanto, moldar o funcionamento do OLSR em tempo de execução. Mais ainda, não se pode desconsiderar a já discutida inclusão dos objetos referentes à extensão *Link Quality*, o que

completa esta OLSR MIB com um aspecto de maior abrangência para os ambientes de roteamento OLSR hoje sendo utilizados.

5.2.2. Objetos da OLSR MIB

Abaixo, a Figura 5.2 apresenta a árvore da OLSR MIB. Os objetos e tabelas que compõe o primeiro componente da estrutura de gerenciamento para o protocolo OLSR estão nela hierarquizados. Apresentar-se-á, agora, uma breve descrição de cada variável, expondo a funcionalidade de cada uma no contexto da operação do OLSR.

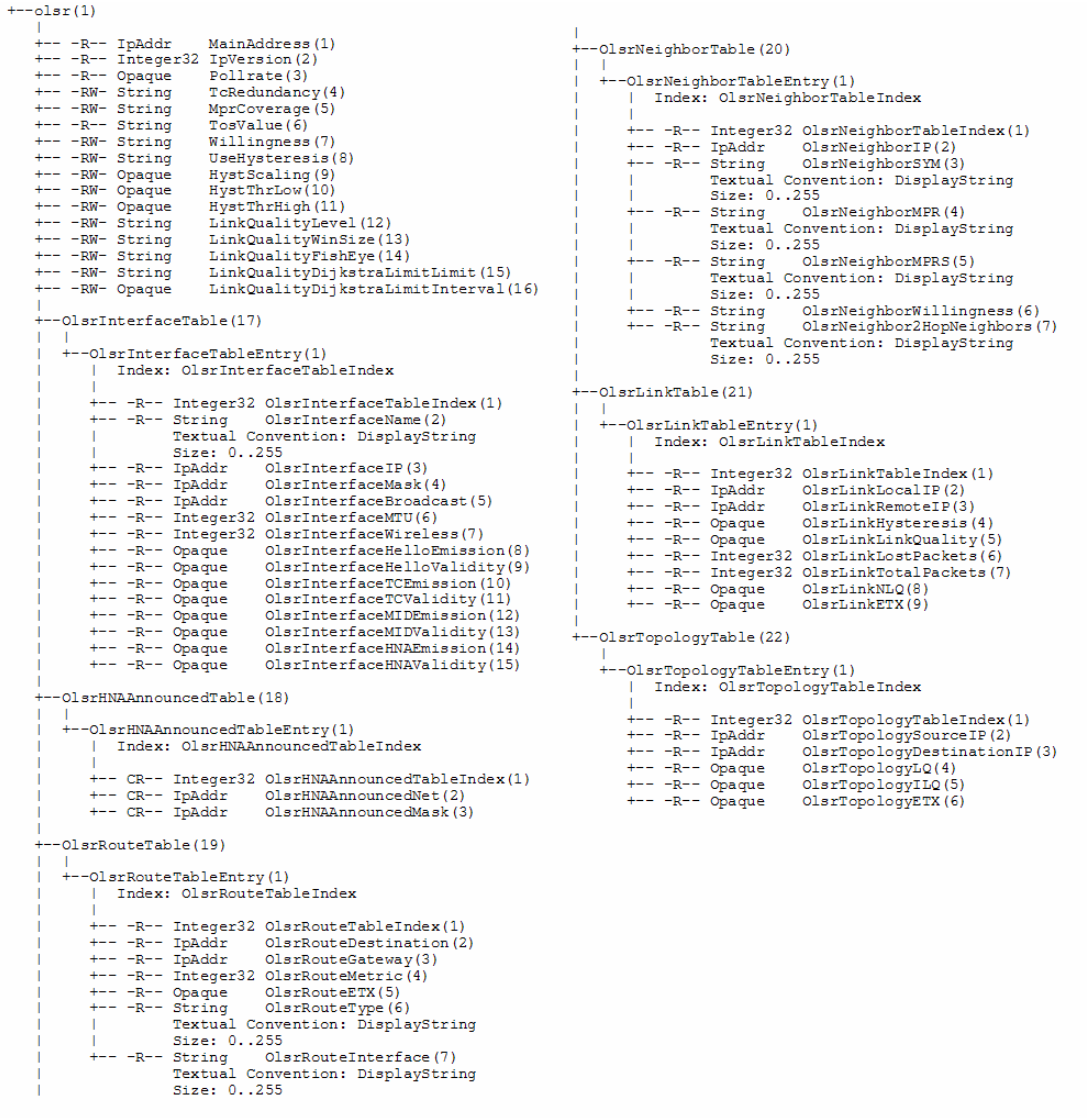

Figura 5.2 – Árvore da OLSR MIB.

5.2.2.1. *MainAddress*

É o endereço principal do nó. Será utilizado no tráfego de controle como o endereço de origem das mensagens.

```
TIPO:     Endereço IP.
Acesso:   Leitura.
```

5.2.2.2. *IpVersion*

Refere-se à versão do protocolo IP sendo utilizada pelo nó.

```
Tipo:     Inteiro.
Acesso:   Leitura.
```

5.2.2.3. *Pollrate*

Este objeto remete-se à entidade **Agendador** do *OLSRD*, demonstrada no terceiro capítulo. No contexto da OLSR MIB, ***Pollrate*** determina a freqüência, em segundos, do ciclo do **Agendador**.

```
Tipo:     Float.
Acesso:   Leitura.
```

5.2.2.4. *TcRedundancy*

É o valor que controla a redundância TC do nó. A redundância TC é uma função auxiliar do protocolo OLSR. Seu significado está declarado no segundo capítulo.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.5. *MprCoverage*

Objeto que traduz o valor da função auxiliar de redundância MPR, também descrita no segundo capítulo.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.6. *TosValue*

Especifica o valor que será utilizado no campo *Type of Service* do cabeçalho IP do tráfego de controle do OLSR.

```
Tipo:     Hexadecimal.
Acesso:   Leitura.
```

5.2.2.7. *Willingness*

Valor de *willingness* do nó, ou seja, a inclinação do mesmo para encaminhar tráfego em nome de outros nós. Varia de 00 (nunca) à 07 (sempre).

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.8. *UseHysteresis*

Este objeto dita se a função auxiliar de histerese será implementada pelo nó. Ao ser marcado como 00 00 00 00 implicará na não utilização da técnica, enquanto uma marcação de 01 00 00 00 a habilitará.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.9. *HystScaling*

Determina o fator de escalabilidade que será aplicado à função de histerese, caso esta seja implementada. Seu significado e contexto estão descritos no item sobre a função auxiliar de histerese no capítulo dois. Deve ser um ponto flutuante com valor menor que 1.0.

```
Tipo:     Float.
Acesso:   Leitura e Escrita.
```

5.2.2.10. *HystThrLow*

Objeto que especifica o limiar inferior para atribuir a assimetria para um enlace. Deve ser um ponto flutuante menor que o limiar superior e opera como determinado pela função auxiliar de histerese.

```
Tipo:     Float.
Acesso:   Leitura e Escrita.
```

5.2.2.11. *HystThrHigh*

Objeto que especifica o limiar superior para o aceite de um enlace. Deve ser um ponto flutuante maior que o limiar inferior e opera como determinado pela função auxiliar de histerese.

```
Tipo:     Float.
Acesso:   Leitura e Escrita.
```

5.2.2.12. *LinkQualityLevel*

Este objeto define se a extensão *Link Quality* será utilizada, bem como, se ela o for, qual o esquema de funcionamento por ela adotado. Dessa forma, uma marcação de 00 despreza a extensão e o *OLSRD* opera em modo RFC 3626. Já a escrita da variável com o valor 01 inicia a operação em modo *Link Quality*, utilizando os cálculos alterados apenas para a seleção de MPRs. A escolha do valor 02 implica que a extensão será utilizada para o cálculo das rotas

também. Uma marcação de 01 ou 02 quebra a compatibilidade com a RFC do OLSR e, conseqüentemente, só deverá ser utilizada se todos os nós do ambiente se comprometerem em fazer o mesmo.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.13. *LinkQualityWinSize*

Variável que determina o tamanho da janela de pacotes a ser considerada no cálculo da qualidade de enlace na operação da extensão *Link Quality*. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.14. *LinkQualityFishEye*

Dita se o algoritmo *Fish Eye* será ou não habilitado. A escolha pela implementação do algoritmo evita que pequenas mudanças locais no ambiente de roteamento influenciem em nós afastados, causando processamento desnecessário. A descrição do funcionamento em modo *Fish Eye* pode ser visualizada na Tabela 3.2. A escrita do objeto com 01 habilita o algoritmo, enquanto 00 o descarta. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.15. *LinkQualityDijkstraLimitLimit*

À semelhança do que ocorre com o *Fish Eye*, a utilização do parâmetro **LinkQualityDijkstraLimit** da Tabela 3.2 objetiva diminuir o gasto com processamento

desnecessário. Entretanto, neste caso específico o que se deseja economizar é processamento local com mudanças distantes. Portanto, o parâmetro **LinkQualityDijkstraLimit** foi desmembrado em dois objetos da OLSR MIB: ***LinkQualityDijkstraLimitLimit*** e ***LinkQualityDijkstraLimitInterval***. O objeto sendo aqui descrito é o ***LinkQualityDijkstraLimitLimit***. Ele permite ao *OLSRD* recalcular a tabela de rotas somente se uma mensagem LQ TC chegar oriunda de um nó distante não mais que este objeto de saltos. Uma compreensão melhor do algoritmo pode ser encontrada no parâmetro correspondente da Tabela 3.2. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.16. *LinkQualityDijkstraLimitInterval*

Este objeto opera em conjunto com o ***LinkQualityDijkstraLimitLimit***. Aqui ele define o intervalo de tempo, em segundos, em que a tabela de rotas será recalculada de qualquer forma (de acordo com o contexto apresentado na Tabela 3.2). Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.

```
Tipo:     Hexadecimal.
Acesso:   Leitura e Escrita.
```

5.2.2.17. *OlsrInterfaceTable*

Esta é a tabela da OLSR MIB que contém os objetos referentes à cada interface local participando do domínio de roteamento OLSR. Seus objetos, tipos e níveis de acesso são:

OlsrInterfaceTableIndex: Inteiro que indexa a tabela.
```
     Tipo:     Inteiro.
     Acesso:   Leitura.
```

OlsrInterfaceName: O nome da interface dado pelo sistema operacional.

Tipo: String.
Acesso: Leitura.

OlsrInterfaceIP: O endereço IP específico da interface.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrInterfaceMask: A máscara de rede do endereço IP da interface.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrInterfaceBroadcast: O endereço de *broadcast* da interface.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrInterfaceMTU: O tamanho do MTU sendo utilizado pela interface.
Tipo: Inteiro.
Acesso: Leitura.

OlsrInterfaceWireless: Sinal que indica se a interface é sem fio. Mostra 1 se ela o é, e 0 se não.
Tipo: Inteiro.
Acesso: Leitura.

OlsrInterfaceHelloEmission: Intervalo, em segundos, em que uma mensagem HELLO é emitida pela interface.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceHelloValidity: Intervalo, em segundos, pelo qual a mensagem HELLO deverá ser considerada válida.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceTCEmission: Intervalo, em segundos, em que uma mensagem TC é emitida pela interface.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceTCValidity: Intervalo, em segundos, pelo qual a mensagem TC deverá ser considerada válida.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceMIDEmission: Intervalo, em segundos, em que uma mensagem MID é emitida pela interface.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceMIDValidity: Intervalo, em segundos, pelo qual a mensagem MID deverá ser considerada válida.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceHNAEmission: Intervalo, em segundos, em que uma mensagem HNA é emitida pela interface.
Tipo: Float.
Acesso: Leitura.

OlsrInterfaceHNAValidity: Intervalo, em segundos, pelo qual a mensagem HNA deverá ser considerada válida.
Tipo: Float.
Acesso: Leitura.

5.2.2.18. *OlsrHNAAnnouncedTable*

De acordo com a função auxiliar HNA, demonstrada no segundo capítulo, esta tabela implementa tal funcionalidade. Dessa forma, pode-se visualizar as redes sendo anunciadas pelo nó, bem como também se adicionar, via operação de criação de linhas, uma nova entrada na tabela, ou mesmo modificar uma entrada já existente. Seus objetos, tipos e níveis de acesso são:

OlsrHNAAnnouncedTableIndex: Inteiro que indexa a tabela.
Tipo: Inteiro.
Acesso: Leitura e Escrita.

OlsrHNAAnnouncedNet: A rede sendo anunciada pelo nó.
Tipo: Endereço IP.
Acesso: Leitura e Escrita.

OlsrHNAAnnouncedMask: A máscara de rede da rede sendo anunciada pelo nó.
Tipo: Endereço IP.
Acesso: Leitura e Escrita.

5.2.2.19. *OlsrRouteTable*

A ***OlsrRouteTable*** é a tabela da OLSR MIB que contém os objetos referentes às rotas aprendidas pelo nó via o protocolo OLSR. Seus objetos, tipos e níveis de acesso são:

OlsrRouteTableIndex: Inteiro que indexa a tabela.

Tipo: Inteiro.
Acesso: Leitura.

OlsrRouteDestination: Endereço IP do destino da rota.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrRouteGateway: Endereço IP a ser utilizado para atingir o destino.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrRouteMetric: Quantidade de saltos necessária para se atingir o destino.
Tipo: Inteiro.
Acesso: Leitura.

OlsrRouteETX: O ETX de uma rota. Conforme explicado na extensão *Link Quality*, é o número de transmissões esperadas para o pacote atingir o destino. É definido como a soma de todos os ETXs dos enlaces envolvidos. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

OlsrRouteType: Tipo da rota aprendida. Se for para um nó específico, constará como *HOST*. Se for para uma rede anunciada como HNA, *HNA* será seu valor.
Tipo: String.
Acesso: Leitura.

OlsrRouteInterface: Nome da interface OLSR local a ser utilizada para atingir o destino.
Tipo: String.
Acesso: Leitura.

5.2.2.20. *OlsrNeighborTable*

Os objetos que traduzem os vizinhos de um nó estão compilados na tabela ***OlsrNeighborTable***. Os respectivos objetos, tipos e níveis de acesso são:

OlsrNeighborTableIndex: Inteiro que indexa a tabela.
Tipo: Inteiro.
Acesso: Leitura.

OlsrNeighborIP: Endereço IP do vizinho.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrNeighborSYM: Se apresentado como *YES* indica que existe ao menos um enlace simétrico entre o vizinho em questão e o nó local. Um valor *NO* significa que todos os enlaces entre o vizinho e este nó são assimétricos.
Tipo: String.
Acesso: Leitura.

OlsrNeighborMPR: Objeto que expõe se o vizinho foi escolhido como MPR por este nó.
Tipo: String.
Acesso: Leitura.

OlsrNeighborMPRS: Este objeto indica se o vizinho selecionou este nó como um MPR.
Tipo: String.
Acesso: Leitura.

OlsrNeighborWillingness: Traduz o valor de *willingness* do vizinho.
Tipo: Hexadecimal.
Acesso: Leitura.

OlsrNeighbor2HopNeighbors: Objeto que oferece uma lista dos vizinhos de dois saltos atingíveis por este nó. Os endereços IP de tais nós serão apresentados separados pelo caractere "-".
Tipo: String.
Acesso: Leitura.

5.2.2.21. *OlsrLinkTable*

Esta tabela da OLSR MIB compila todos os enlaces conhecidos pelo nó. Seus objetos, tipos e níveis de acesso são:

OlsrLinkTableIndex: Inteiro que indexa a tabela.
Tipo: Inteiro.
Acesso: Leitura.

OlsrLinkLocalIP: O endereço IP local do enlace.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrLinkRemoteIP: O endereço IP remoto do enlace.

Tipo: Endereço IP.
Acesso: Leitura.

OlsrLinkHysteresis: Quando a função auxiliar de histerese estiver habilitada, seu valor momentâneo aplicado ao enlace é visualizado neste objeto.
Tipo: Float.
Acesso: Leitura.

OlsrLinkLinkQuality: Quando a extensão *Link Quality* está habilitada, o valor da qualidade do enlace LQ é visualizado neste objeto. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

OlsrLinkLostPackets: Permite a visualização de quantos pacotes enviados pelo vizinho por este enlace não atingiram o nó. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Inteiro.
Acesso: Leitura.

OlsrLinkTotalPackets: Este objeto refere-se ao valor escolhido no objeto ***LinkQualityWinSize*** desta mesma OLSR MIB. Traduz o número de pacotes a ser considerado no cálculo de LQ do enlace. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Inteiro.
Acesso: Leitura.

OlsrLinkNLQ: Varia entre 0 e 1 e apresenta a qualidade de enlace LQ segundo a extensão *Link Quality* percebida pelo nó vizinho. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

OlsrLinkETX: Apresenta o valor de ETX do enlace, de acordo com a operação *Link Quality*. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

5.2.2.22. *OlsrTopologyTable*

O objeto ***OlsrTopologyTable*** é a tabela da OLSR MIB que expõe as entradas da **base de informações de controle de topologia**, conforme apresentado no segundo capítulo. Tal repositório contém a informação sobre a topologia de rede recebida pelo ambiente de roteamento OLSR. Os dados neste repositório são utilizados para o cálculo de rotas. Seus objetos, tipos e níveis de acesso são:

OlsrTopologyTableIndex: Inteiro que indexa a tabela.
Tipo: Inteiro.
Acesso: Leitura.

OlsrTopologySourceIP: Endereço IP do nó que reportou o enlace.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrTopologyDestinationIP: Endereço IP do nó para o qual o nó origem reportou o enlace.
Tipo: Endereço IP.
Acesso: Leitura.

OlsrTopologyLQ: Na operação da extensão *Link Quality*, o valor da qualidade do enlace LQ percebido pelo nó de origem é apresentado neste objeto. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

OlsrTopologyILQ: Também na operação da extensão *Link Quality*, este é o valor da qualidade de enlace LQ percebido pelo nó de destino em relação ao nó de origem. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

OlsrTopologyETX: Este objeto informa o valor de ETX para este enlace. Este é um objeto não compatível com a RFC 3626, por conseguinte só deverá ser utilizado quando todos os nós do ambiente estiverem operando segundo a extensão *Link Quality*.
Tipo: Float.
Acesso: Leitura.

5.3. IMPLEMENTAÇÃO DA OLSR MIB

Depois da especificação da OLSR MIB como um documento compilando os objetos relevantes em um nó OLSR, a necessidade de avaliação e manipulação das definições recém-criadas motivou a criação de uma implementação. Seguindo esse caminho, um *software* implementando a OLSR MIB foi idealizado e confeccionado, concretizando, então, o segundo componente da estrutura de gerenciamento para o protocolo OLSR. Finalmente, o ambiente de administração objetivado estaria completo.

5.3.1. Visão Geral

Na Figura 5.1 do início deste capítulo demonstra-se brevemente o contexto estrutural da implementação da OLSR MIB no ambiente OLSR. Agora, na Figura 5.3, expande-se e detalha-se a situação de comunicação do programa.

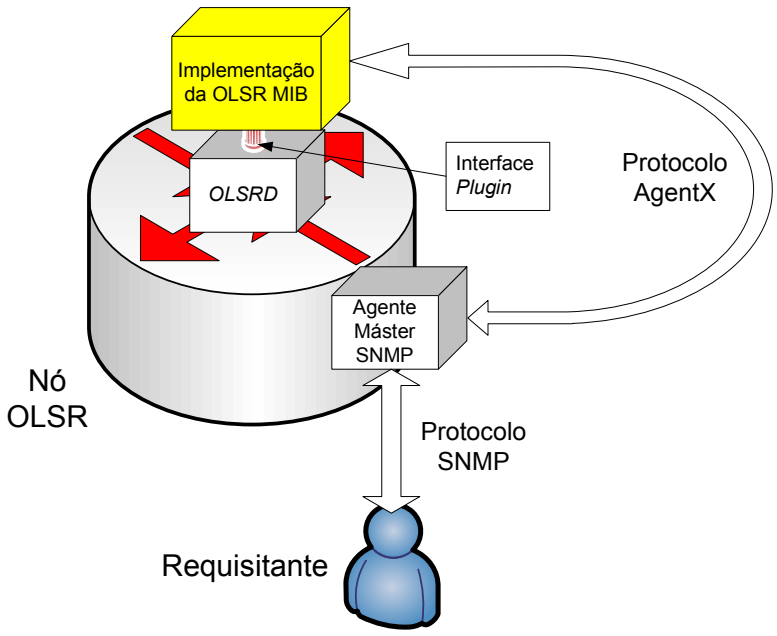

Figura 5.3 – Implementação da OLSR MIB

Nesse contexto, verifica-se que existem duas importantes interfaces do programa com o exterior: uma com o **agente máster** SNMP e outra com a implementação do protocolo OLSR. Retomando a infra-estrutura SNMP exposta no quarto capítulo, a implementação da OLSR MIB em sua comunicação com o **agente máster** SNMP se torna um **subagente**, enquanto em sua comunicação com o *OLSRD*, ela utiliza essa implementação do protocolo OLSR como seu repositório MIB. O interfaceamento com o **agente máster** SNMP se dá via protocolo AgentX; já com o *OLSRD*, a interface *plugin* descrita no terceiro capítulo é utilizada.

A solução adotada para incorporar tanto o suporte ao protocolo AgentX quanto a funcionalidade de interface *plugin* foi implementar o programa como um *plugin OLSRD* com bibliotecas AgentX implantadas. Para que o *software* fosse confeccionado como um *plugin OLSRD*, as especificações e possibilidades determinadas pela interface *plugin* disponibilizada pelo *OLSRD* e explicadas no terceiro capítulo tiveram que ser seguidas. Assim, o programa tornou-se uma biblioteca dinamicamente carregada, ou DLL (*Dynamic Linked Library*). Para a implantação das bibliotecas AgentX, utilizou-se as oferecidas gratuitamente pelo projeto NET-SNMP [25]. É importante destacar que, apesar das bibliotecas de suporte ao AgentX serem oferecidas pelo NET-SNMP, em nenhum momento fica obrigatório a utilização do **agente máster** também oferecido pelo NET-SNMP. Por serem totalmente aderentes à RFC 2741 [24] do AgentX, essas bibliotecas permitem a interação com qualquer **agente máster** genérico, ampliando o escopo de utilização desta estrutura de gerenciamento para o protocolo OLSR.

5.3.2. Estrutura de Arquivos

Após o processo de concepção e codificação da implementação da OLSR MIB, chegou-se à primeira versão do programa. A linguagem de programação escolhida foi a C, sendo a opção esperada uma vez que o próprio *OLSRD* foi com ela escrito e a interface *plugin* do mesmo dita a sua utilização. Um pacote de instalação foi, então, criado e registrado como projeto aberto em [26].

O conteúdo de tal pacote e, conseqüentemente, da estrutura de arquivos do *software* é:

No diretório raiz:

- **Makefile**: arquivo contendo as diretivas de compilação da implementação da OLSR MIB,
- **OLSR-MIB.txt**: última versão da OLSR MIB,
- **README**: instruções de instalação do *software*,
- **version-script.txt**: arquivo contendo as funções de interfaceamento com o *OLSRD*,
- **src**: diretório contendo os arquivos fonte do programa.

No diretório **src**:

- **olsrd_snmpd_agentx.h**: arquivo cabeçalho contendo as definições do arquivo fonte **olsrd_snmpd_agentx.c**,
- **olsrd_snmpd_agentx.c**: arquivo fonte do programa responsável pelo interfaceamento da implementação com o *OLSRD* e com o **agente máster** SNMP,
- **olsrd_mib.h**: arquivo cabeçalho com as definições do arquivo fonte **olsrd_mib.c**,
- **olsrd_mib.c**: arquivo fonte responsável por mapear os objetos conceituais da OLSR MIB em variáveis e tabelas reais do *OLSRD*.

Como foi delineado nos arquivos fontes da implementação da OLSR MIB, o programa tem duas funcionalidades importantes principais: o interfaceamento da implementação com o *OLSRD* e com o **agente máster** e o mapeamento dos objetos da OLSR MIB. Este dois aspectos irão imbuir o *software* com o paradigma de funcionamento explicitado nas Figuras 5.3 e 5.1. Agora, detalhar-se-á a operação das duas funcionalidades, e, em seguida, o fluxo de execução do programa será demonstrado.

5.3.3. Interfaceamento com o *OLSRD* e com o Agente Máster

O interfaceamento com a implementação *OLSRD* e com o **agente máster** SNMP têm como objetivos, respectivamente, tornar real um repositório MIB OLSR e permitir a comunicação

em protocolo AgentX deste programa com um **agente máster**. Esse contexto de operação visa respeitar a infra-estrutura SNMP explicitada na Figura 4.3.

Conforme estabelecido anteriormente, o interfaceamento com o *OLSRD* é feito pela interface *plugin*. No caso específico da implementação da OLSR MIB, duas tarefas operacionais dentre aquelas possíveis pelo paradigma de comunicação entre *OLSRD* e *plugin* (Figura 3.7) serão necessárias para atingir o objetivo do ambiente de administração:

- **acesso aos repositórios de informação do OLSR**: para que as variáveis e tabelas reais do *OLSRD* sejam mapeadas para os objetos da OLSR MIB, acesso direto aos endereços de memória que guardam tais informações dentro do *OLSRD* é necessário,
- **possibilidade de registro de funções junto à entidade Agendador do *OLSRD***: registrar funções temporizadas no **Agendador**, conforme demonstrado no terceiro capítulo, permite à implementação da OLSR MIB a capacidade de coordenação dos eventos e transações a serem implementadas pelo programa.

Para o caso da comunicação da implementação da OLSR MIB com um **agente máster** genérico, as bibliotecas do NET-SNMP [25] foram utilizadas. Assim, o registro junto ao **agente máster** SNMP e as transações AgentX são permitidas por funções disponíveis por tais bibliotecas.

O código que implementa ambos os interfaceamentos acima descritos está contido dentro do arquivo fonte **olsrd_snmpd_agentx.c**, mencionado anteriormente e presente no APÊNDICE B desta dissertação.

Expor-se-á, neste momento, as especificidades de cada interfaceamento.

5.3.3.1. Interfaceamento com o *OLSRD*

O interfaceamento com o *OLSRD* se baseia nas possibilidades da interface *plugin*. Todavia, neste *software*, somente as funcionalidades de acesso aos **repositórios de informações** e o registro de funções junto ao **Agendador** serão necessárias.

Para o primeiro caso, ou seja, para o acesso aos **repositórios de informação**, ao ser iniciado, o *OLSRD* verifica em seu arquivo de configuração se algum *plugin* deverá ser chamado. Para isso o parâmetro **LoadPlugin** do arquivo de configuração (Tabela 3.1) deverá apontar para o nome do *plugin* desejado, discriminando os parâmetros específicos a serem enviados e, conseqüentemente, utilizados pela DLL. Em seguida, as funções **olsrd_plugin_register_param()** e **olsrd_plugin_init()**, presentes no arquivo fonte **olsrd_snmpd_agentx.c** (APÊNDICE B) são chamadas e à elas fornecidas, respectivamente, acesso aos parâmetros especificados para a chamada do plugin e aos **repositórios de informações** do *OLSRD*.

Já para o segundo caso, o do registro de funções com o **Agendador**, as funções **olsr_register_scheduler_event()** e **olsr_register_timeout_function()**, disponibilizadas pela interface *plugin*, são utilizadas. A primeira possibilita o registro de uma função a ser chamada pelo **Agendador** em ciclos determinados por um valor de intervalo passado na própria chamada de registro. A segunda registrará uma função a ser chamada em todos os ciclos do **Agendador**. Para este caso, duas funções são registradas:

- **olsrd_snmpd()**: esta função registrada pela **olsr_register_scheduler_event()** tem como objetivo implementar o *daemon* de atualização das variáveis e tabelas dinâmicas. Isso é necessário, pois algumas variáveis e tabelas do *OLSRD* sofrem mudanças constantemente e precisam ter seus valores atualizados junto aos objetos da OLSR MIB. O intervalo de chamada da função e, consequentemente, de atualização dos objetos da OLSR MIB, é determinado pelo parâmetro arbitrário passado pela diretiva **LoadPlugin** do carregamento deste *software*.
- **agentx()**: esta função registrada pela **olsr_register_timeout_function()** tem como objetivo tratar das transações AgentX com o **agente máster** SNMP registrado. Ela é chamada em todos os ciclos do **Agendador**.

Ambas as funções descritas acima estão implementadas dentro do arquivo fonte **olsrd_snmpd_agentx.c** e podem ser analisadas no APÊNDICE B.

5.3.3.2. Interfaceamento com o Agente Máster

O interfaceamento com o **agente máster** SNMP tem dois objetivos: o registro da implementação OLSR MIB como um **subagente** e o tratamento das conseqüentes transações futuras SNMP via protocolo AgentX.

Para ambos os casos, bibliotecas NET-SNMP [25] são utilizadas. Porém, como já citado anteriormente, a utilização de tais bibliotecas não força a utilização do **agente máster** do próprio NET-SNMP, pois os padrões da RFC 2741 [24] são respeitados, possibilitando a interação com qualquer **agente máster** disponível.

Para o registro deste programa como um **subagente** as seguintes diretivas são utilizadas:

```
/* AgentX initialization*/
snmp_enable_stderrlog();
netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID,
NETSNMP_DS_AGENT_ROLE, 1);
init_agent("olsrd_snmpd_agentx");
init_olsrd_mib();
olsr_printf(3, "*** OLSRD_SNMPD: \n" );
olsr_printf(3,  "***  OLSRD_SNMPD:  olsrd_snmpd_agentx  AgentX
conecting:\n");
init_snmp("olsrd_snmpd_agentx");
/* End of AgentX initialization */
```

Elas são chamadas no início da execução do *plugin* e se encontram dentro da função **olsrd_plugin_init()** do arquivo fonte **olsrd_snmpd_agentx.c** (APÊNDICE B).

Para o tratamento das transações SNMP a serem recebidas vindas do **agente máster** SNMP em qual este **subagente** se registrou, a função oferecida pelo NET-SNMP [25] **agent_check_and_process()** é utilizada. Ela é chamada dentro da função **agentx()** registrada junto ao **Agendador** e é, conseguintemente, acionada em todos os ciclos da entidade.

5.3.4. Mapeamento dos Objetos da OLSR MIB

Segundo a infra-estrutura SNMP incluindo **subagentes** AgentX, esta implementação da MIB extendida OLSR será responsável por responder e manipular os objetos à ela referentes. No

entanto, esses elementos são conceituais e modelados em um documento. É necessário que o **subagente** mapeie para o mundo externo os objetos da OLSR MIB em seus valores reais. No caso deste programa, a informação concreta reside no processo do *OLSRD*. Logo, é cogente que, via interface *plugin*, estas variáveis e tabelas sejam acessadas e mapeadas para os objetos especificados na OLSR MIB.

O mapeamento das informações do *OLSRD* para a OLSR MIB foi implementado dentro do arquivo fonte **olsrd_mib.c**, disponível no APÊNDICE C. Uma vez feito o mapeamento, o **subagente**, quando receber uma requisição SNMP pelo protocolo AgentX, saberá em que local do *OLSRD* poderá buscar o valor procurado.

Existirão dois casos de mapeamento: um para variáveis escalares e outro para tabelas. Para o caso das variáveis escalares, a função de registro **netsnmp_register_watched_scalar()** é utilizada. Para as tabelas, faz-se uso da **netsnmp_register_table_data_set()**. Ambas são importadas das bibliotecas NET-SNMP [25].

Na Tabela 5.1 está presente o mapeamento completo. Nela, se pode observar quais variáveis e tabelas do *OLSRD* são responsáveis por cada objeto da OLSR MIB e como são seus formatos.

Tabela 5.1 – Mapeamento dos Objetos da OLSR MIB

OBJETO OLSR MIB	**VARIÁVEL *OLSRD***	**FORMATO *OLSRD***
MainAddress	olsr_cnf->main_addr	u_int32_t
IpVersion	olsr_cnf->ip_version	Int
Pollrate	olsr_cnf->pollrate	Float
TcRedundancy	olsr_cnf->tc_redundancy	u_int8_t
MprCoverage	olsr_cnf->mpr_coverage	u_int8_t
TosValue	olsr_cnf->tos	u_int16_t
Willingness	olsr_cnf->willingness	u_int8_t
UseHysteresis	olsr_cnf->use_hysteresis	olsr_bool
HystScaling	olsr_cnf->hysteresis_param.scaling	Float
HystThrLow	olsr_cnf->hysteresis_param.thr_low	Float
HystThrHigh	olsr_cnf->hysteresis_param.thr_high	Float
LinkQualityLevel	olsr_cnf->lq_level	u_int8_t
LinkQualityWinSize	olsr_cnf->lq_wsize	u_int32_t
LinkQualityFishEye	olsr_cnf->lq_fish	u_int8_t
LinkQualityDijkstraLimitLimit	olsr_cnf->lq_dlimit	u_int8_t
LinkQualityDijkstraLimitInterval	olsr_cnf->lq_dinter	Float

Tabela OlsrInterfaceTable		
OlsrInterfaceName	olsr_cnf->interfaces->name	Char
OlsrInterfaceIP	olsr_cnf->interfaces->interf->int_addr	sockaddr
OlsrInterfaceMask	olsr_cnf->interfaces->interf->int_netmask	sockaddr
OlsrInterfaceBroadcast	olsr_cnf->interfaces->interf->int_broadaddr	sockaddr
OlsrInterfaceMTU	olsr_cnf->interfaces->interf->int_mtu	Int
OlsrInterfaceWireless	olsr_cnf->interfaces->interf->is_wireless	Int
OlsrInterfaceHelloEmission	olsr_cnf->interfaces->cnf->hello_params.emission_interval	Float
OlsrInterfaceHelloValidity	olsr_cnf->interfaces->cnf->hello_params.validity_time	Float
OlsrInterfaceTCEmission	olsr_cnf->interfaces->cnf->tc_params.emission_interval	Float
OlsrInterfaceTCValidity	olsr_cnf->interfaces->cnf->tc_params.validity_time	Float
OlsrInterfaceMIDEmission	olsr_cnf->interfaces->cnf->mid_params.emission_interval	Float
OlsrInterfaceMIDValidity	olsr_cnf->interfaces->cnf->mid_params.validity_time	Float
OlsrInterfaceHNAEmission	olsr_cnf->interfaces->cnf->hna_params.emission_interval	Float
OlsrInterfaceHNAValidity	olsr_cnf->interfaces->cnf->hna_params.validity_time	Float
Tabela OlsrHNAAnnouncedTable		
HNAAnnouncedNet	olsr_cnf->hna4_entries->net	u_int32_t
HNAAnnouncedMask	olsr_cnf->hna4_entries->netmask	u_int32_t
Tabela OlsrRouteTable		
OlsrRouteDestination	rt_entry->rt_dst	u_int32_t
OlsrRouteGateway	rt_entry->rt_router	u_int32_t
OlsrRouteMetric	rt_entry->rt_metric	u_int16_t
OlsrRouteETX	rt_entry->rt_etx	Float
OlsrRouteType	“HNA” ou “HOST”	Char
OlsrRouteInterface	rt_entry->rt_if->int_name	Char
Tabela OlsrNeighborTable		
OlsrNeighborIP	neighbor_entry->neighbor_main_addr	u_int32_t
OlsrNeighborSYM	neighbor_entry->status	u_int8_t
OlsrNeighborMPR	neighbor_entry->is_mpr	olsr_bool
OlsrNeighborMPRS	olsr_lookup_mprs_set(neighbor_entry->neighbor_main_addr)	mpr_selector ou NULL

OlsrNeighborWillingness	neighbor_entry->willingness	u_int8_t
OlsrNeighbor2HopNeighbors	Todas as entradas em: neighbor_entry->neighbor_2_list_entry->neighbor_2->neighbor_2_addr	Tabela de: u_int32_t
Tabela OlsrLinkTable		
OlsrLinkLocalIP	link->local_iface_addr	u_int32_t
OlsrLinkRemoteIP	link->neighbor_iface_addr	u_int32_t
OlsrLinkHysteresis	link->L_link_quality	Float
OlsrLinkLinkQuality	link->loss_link_quality	Float
OlsrLinkLostPackets	link->lost_packets	Int
OlsrLinkTotalPackets	link->total_packets	Int
OlsrLinkNLQ	link->neigh_link_quality	Float
OlsrLinkETX	1.0 / (link->loss_link_quality * link->neigh_link_quality)	Float
Tabela OlsrTopologyTable		
OlsrTopologySourceIP	entry->T_last_addr	u_int32_t
OlsrTopologyDestinationIP	dst_entry->T_dest_addr	u_int32_t
OlsrTopologyLQ	dst_entry->link_quality	Float
OlsrTopologyILQ	dst_entry->inverse_link_quality	Float
OlsrTopologyETX	1.0 / (dst_entry->link_quality * dst_entry->inverse_link_quality)	Float

5.3.5. Fluxo de Execução

O fluxo de execução expõe como a implementação da OLSR MIB se comporta temporalmente.

Ao ser iniciado em um nó, o *OLSRD* verifica seu arquivo de configuração e começa a operar segundo os parâmetros lidos (Tabelas 3.1 e 3.2). Uma destas diretivas é a **LoadPlugin** (Tabela 3.1). Ela, como explicado na interface *plugin*, especifica quais *plugins* deverão ser carregados e, também, permite o envio de variáveis para os mesmos. No caso da implementação da OLSR MIB, o parâmetro enviado é o intervalo a ser empregado para cada ciclo da função **olsrd_snmpd()** a ser registrada junto ao **Agendador**. Esse envio é realizado pela função **olsrd_plugin_register_param()**.

Em seguida, a função **olsrd_plugin_init()** marca o início da execução da implementação da OLSR MIB. Segue-se o registro do programa como **subagente** do **agente máster** SNMP e o acesso à entidade **Agendador** para implementação das funções **olsrd_snmpd()** e **agentx()**.

Finalmente, a **olsrd_snmpd()** ficará responsável pela atualização das variáveis e tabelas reais junto aos objetos da OLSR MIB e a **agentx()** se encarregará das eventuais transações futuras com o **agente máster** SNMP.

A Figura 5.4 resume esse fluxo de execução do *software*.

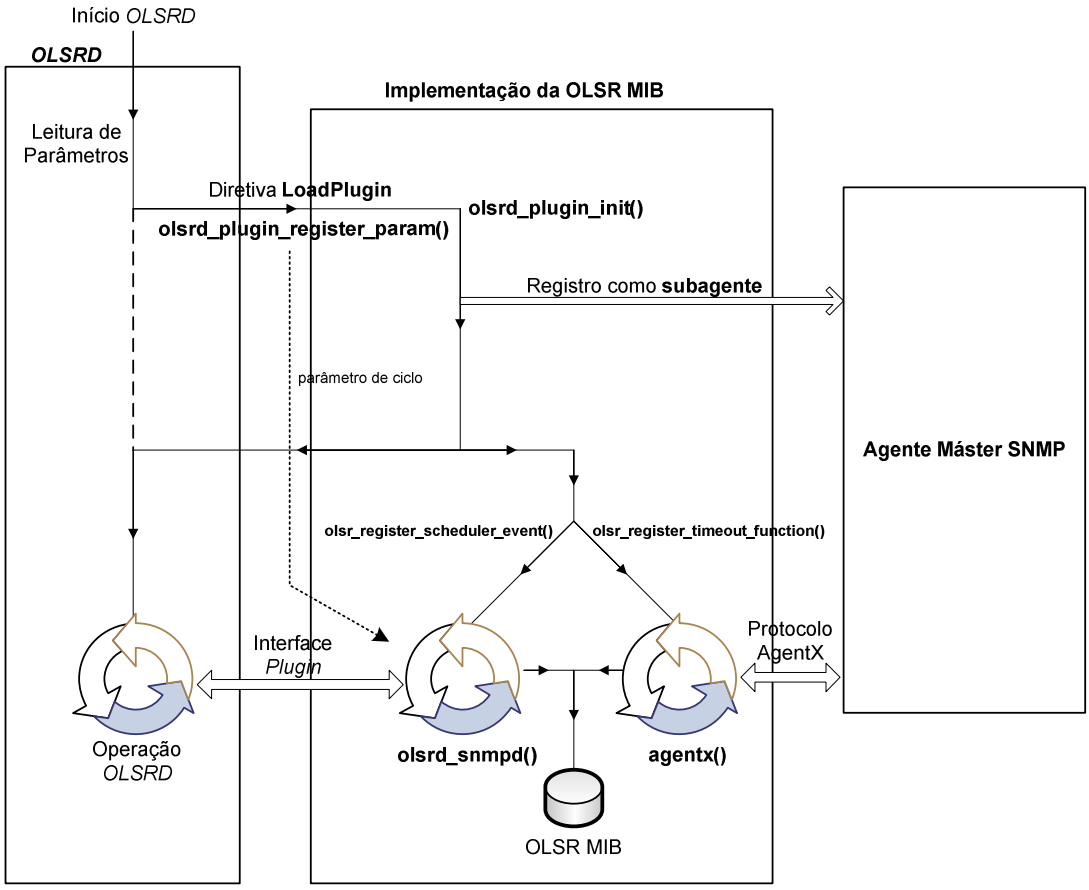

Figura 5.4 – Fluxo de Execução da Implementação da OLSR MIB

5.4. EXPERIMENTOS

Com a OLSR MIB elaborada e a implementação terminada, experimentos foram realizados com o objetivo de validar a estrutura de gerenciamento para o protocolo OLSR. Apesar do fato de interpretações ou mesmo aplicações complexas poderem ser derivadas deste ambiente de administração, os experimentos realizados visaram somente legitimar as funcionalidades

da estrutura.

Para os experimentos, o cenário presente na Figura 5.5 foi preparado e o *OLSRD* foi instalado e configurado em todos os roteadores. Além disso, para provar a possibilidade de coexistência de nós com e sem a OLSR MIB e sua implementação, apenas o **Roteador 1** teve a estrutura de gerenciamento para o protocolo OLSR instalada. O NET-SNMP [25] também foi instalado e configurado no **Roteador 1** para desempenhar a função de **agente máster** SNMP.

O sistema operacional utilizado nos roteadores foi o Linux Fedora [27]. Todos os *OLSRD*, versão 0.5.3, foram configurados para operar segundo o modo RFC 3626 [1] e, depois que a convergência foi atingida, os experimentos foram iniciados.

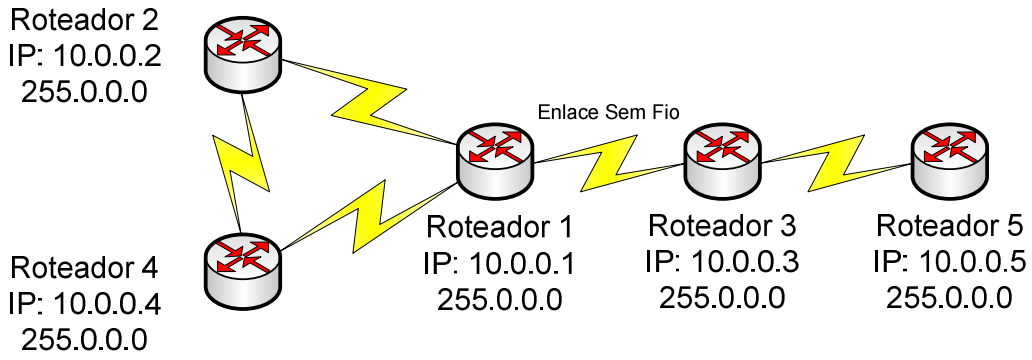

Figura 5.5 – Cenário dos Experimentos

5.4.1. Leitura dos Objetos da OLSR MIB

Como primeiro experimento, uma leitura completa da OLSR MIB foi realizada. A visualização de todos os objetos gerenciáveis disponíveis proporcionou um entendimento abrangente de como o ambiente OLSR pode ser traduzido pelo nó em questão.

Para realizar a leitura, era necessário efetuar a operação SNMP *get-next-request* (Tabela 4.1) em toda a OLSR MIB. A aplicação SNMPWALK disponibilizada pelo NET-SNMP [25] fez justamente essa tarefa. Abaixo, na Figura 5.6, é apresentado o resultado das operações.

```
[root@Roteador 1]# snmpwalk -v 2c -c public 10.0.0.1 olsr
OLSR-MIB::MainAddress.0 = IpAddress: 10.0.0.1
OLSR-MIB::IpVersion.0 = INTEGER: 4
OLSR-MIB::Pollrate.0 = Opaque: Float: 0.05
OLSR-MIB::TcRedundancy.0 = Hex-STRING: 00
OLSR-MIB::MprCoverage.0 = Hex-STRING: 01
OLSR-MIB::TosValue.0 = Hex-STRING: 10 00
OLSR-MIB::Willingness.0 = Hex-STRING: 03
OLSR-MIB::UseHysteresis.0 = Hex-STRING: 01 00 00 00
OLSR-MIB::HystScaling.0 = Opaque: Float: 0.50
OLSR-MIB::HystThrLow.0 = Opaque: Float: 0.30
OLSR-MIB::HystThrHigh.0 = Opaque: Float: 0.80
OLSR-MIB::LinkQualityLevel.0 = Hex-STRING: 00
OLSR-MIB::LinkQualityWinSize.0 = Hex-STRING: 0A 00 00 00
OLSR-MIB::LinkQualityFishEye.0 = Hex-STRING: 00
OLSR-MIB::LinkQualityDijkstraLimitLimit.0 = Hex-STRING: FF
OLSR-MIB::LinkQualityDijkstraLimitInterval.0 = Opaque: Float: 0
OLSR-MIB::OlsrInterfaceTableIndex.1 = INTEGER: 1
OLSR-MIB::OlsrInterfaceName.1 = STRING: eth1
OLSR-MIB::OlsrInterfaceIP.1 = IpAddress: 10.0.0.1
OLSR-MIB::OlsrInterfaceMask.1 = IpAddress: 255.0.0.0
OLSR-MIB::OlsrInterfaceBroadcast.1 = IpAddress: 10.255.255.255
OLSR-MIB::OlsrInterfaceMTU.1 = INTEGER: 1472
OLSR-MIB::OlsrInterfaceWireless.1 = INTEGER: 1
OLSR-MIB::OlsrInterfaceHelloEmission.1 = Opaque: Float: 2.00
OLSR-MIB::OlsrInterfaceHelloValidity.1 = Opaque: Float: 6.00
OLSR-MIB::OlsrInterfaceTCEmission.1 = Opaque: Float: 5.00
OLSR-MIB::OlsrInterfaceTCValidity.1 = Opaque: Float: 15.00
OLSR-MIB::OlsrInterfaceMIDEmission.1 = Opaque: Float: 5.00
OLSR-MIB::OlsrInterfaceMIDValidity.1 = Opaque: Float: 15.00
OLSR-MIB::OlsrInterfaceHNAEmission.1 = Opaque: Float: 5.00
OLSR-MIB::OlsrInterfaceHNAValidity.1 = Opaque: Float: 15.00
OLSR-MIB::OlsrRouteTableIndex.1 = INTEGER: 1
OLSR-MIB::OlsrRouteTableIndex.2 = INTEGER: 2
OLSR-MIB::OlsrRouteTableIndex.3 = INTEGER: 3
OLSR-MIB::OlsrRouteTableIndex.4 = INTEGER: 4
OLSR-MIB::OlsrRouteDestination.1 = IpAddress: 10.0.0.2
OLSR-MIB::OlsrRouteDestination.2 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrRouteDestination.3 = IpAddress: 10.0.0.4
OLSR-MIB::OlsrRouteDestination.4 = IpAddress: 10.0.0.5
OLSR-MIB::OlsrRouteGateway.1 = IpAddress: 10.0.0.2
OLSR-MIB::OlsrRouteGateway.2 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrRouteGateway.3 = IpAddress: 10.0.0.4
OLSR-MIB::OlsrRouteGateway.4 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrRouteMetric.1 = INTEGER: 1
OLSR-MIB::OlsrRouteMetric.2 = INTEGER: 1
OLSR-MIB::OlsrRouteMetric.3 = INTEGER: 1
OLSR-MIB::OlsrRouteMetric.4 = INTEGER: 2
OLSR-MIB::OlsrRouteETX.1 = Opaque: Float: 1.00
OLSR-MIB::OlsrRouteETX.2 = Opaque: Float: 1.00
OLSR-MIB::OlsrRouteETX.3 = Opaque: Float: 1.00
OLSR-MIB::OlsrRouteETX.4 = Opaque: Float: 2.00
OLSR-MIB::OlsrRouteType.1 = STRING: HOST
OLSR-MIB::OlsrRouteType.2 = STRING: HOST
OLSR-MIB::OlsrRouteType.3 = STRING: HOST
OLSR-MIB::OlsrRouteType.4 = STRING: HOST
OLSR-MIB::OlsrRouteInterface.1 = STRING: eth1
OLSR-MIB::OlsrRouteInterface.2 = STRING: eth1
OLSR-MIB::OlsrRouteInterface.3 = STRING: eth1
OLSR-MIB::OlsrRouteInterface.4 = STRING: eth1
OLSR-MIB::OlsrNeighborTableIndex.1 = INTEGER: 1
OLSR-MIB::OlsrNeighborTableIndex.2 = INTEGER: 2
OLSR-MIB::OlsrNeighborTableIndex.3 = INTEGER: 3
OLSR-MIB::OlsrNeighborIP.1 = IpAddress: 10.0.0.2
OLSR-MIB::OlsrNeighborIP.2 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrNeighborIP.3 = IpAddress: 10.0.0.4
OLSR-MIB::OlsrNeighborSYM.1 = STRING: YES
OLSR-MIB::OlsrNeighborSYM.2 = STRING: YES
OLSR-MIB::OlsrNeighborSYM.3 = STRING: YES
OLSR-MIB::OlsrNeighborMPR.1 = STRING: NO
OLSR-MIB::OlsrNeighborMPR.2 = STRING: YES
OLSR-MIB::OlsrNeighborMPR.3 = STRING: NO
OLSR-MIB::OlsrNeighborMPRS.1 = STRING: YES
OLSR-MIB::OlsrNeighborMPRS.2 = STRING: YES
OLSR-MIB::OlsrNeighborMPRS.3 = STRING: YES
OLSR-MIB::OlsrNeighborWillingness.1 = Hex-STRING: 03
OLSR-MIB::OlsrNeighborWillingness.2 = Hex-STRING: 03
OLSR-MIB::OlsrNeighborWillingness.3 = Hex-STRING: 03
OLSR-MIB::OlsrNeighbor2HopNeighbors.1 = STRING: -10.0.0.4-
OLSR-MIB::OlsrNeighbor2HopNeighbors.2 = STRING: -10.0.0.5-
OLSR-MIB::OlsrNeighbor2HopNeighbors.3 = STRING: -10.0.0.2-
OLSR-MIB::OlsrLinkTableIndex.1 = INTEGER: 1
OLSR-MIB::OlsrLinkTableIndex.2 = INTEGER: 2
OLSR-MIB::OlsrLinkTableIndex.3 = INTEGER: 3
OLSR-MIB::OlsrLinkLocalIP.1 = IpAddress: 10.0.0.1
OLSR-MIB::OlsrLinkLocalIP.2 = IpAddress: 10.0.0.1
OLSR-MIB::OlsrLinkLocalIP.3 = IpAddress: 10.0.0.1
OLSR-MIB::OlsrLinkRemoteIP.1 = IpAddress: 10.0.0.4
OLSR-MIB::OlsrLinkRemoteIP.2 = IpAddress: 10.0.0.2
OLSR-MIB::OlsrLinkRemoteIP.3 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrLinkHysteresis.1 = Opaque: Float: 0.687012
OLSR-MIB::OlsrLinkHysteresis.2 = Opaque: Float: 0.999995
OLSR-MIB::OlsrLinkHysteresis.3 = Opaque: Float: 1.00
OLSR-MIB::OlsrLinkLinkQuality.1 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkLinkQuality.2 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkLinkQuality.3 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkLostPackets.1 = INTEGER: 0
OLSR-MIB::OlsrLinkLostPackets.2 = INTEGER: 0
OLSR-MIB::OlsrLinkLostPackets.3 = INTEGER: 0
OLSR-MIB::OlsrLinkTotalPackets.1 = INTEGER: 0
OLSR-MIB::OlsrLinkTotalPackets.2 = INTEGER: 0
OLSR-MIB::OlsrLinkTotalPackets.3 = INTEGER: 0
OLSR-MIB::OlsrLinkNLQ.1 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkNLQ.2 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkNLQ.3 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkETX.1 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkETX.2 = Opaque: Float: 0.00
OLSR-MIB::OlsrLinkETX.3 = Opaque: Float: 0.00
OLSR-MIB::OlsrTopologyTableIndex.1 = INTEGER: 1
OLSR-MIB::OlsrTopologyTableIndex.2 = INTEGER: 2
OLSR-MIB::OlsrTopologySourceIP.1 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrTopologySourceIP.2 = IpAddress: 10.0.0.3
OLSR-MIB::OlsrTopologyDestinationIP.1 = IpAddress: 10.0.0.5
OLSR-MIB::OlsrTopologyDestinationIP.2 = IpAddress: 10.0.0.1
OLSR-MIB::OlsrTopologyLQ.1 = Opaque: Float: 0.00
OLSR-MIB::OlsrTopologyLQ.2 = Opaque: Float: 0.00
OLSR-MIB::OlsrTopologyILQ.1 = Opaque: Float: 0.00
OLSR-MIB::OlsrTopologyILQ.2 = Opaque: Float: 0.00
OLSR-MIB::OlsrTopologyETX.1 = Opaque: Float: 0.00
OLSR-MIB::OlsrTopologyETX.2 = Opaque: Float: 0.00
```

Figura 5.6 – Leitura dos Objetos da OLSR MIB

A visualização das variáveis globais, de ***MainAddress*** até ***LinkQualityDijkstraLimitInterval***, e da primeira tabela, a ***OlsrInterfaceTable***, é essencial porque permite a compreensão da configuração do protocolo OLSR no nó. Neste caso específico, algumas observações importantes são que o **Roteador 1** está configurado com *Willingness* 3, irá utilizar o esquema de histerese, está operando segundo o modo RFC e a interface OLSR presente está configurada com os intervalos padrões de emissão e validade de mensagens.

Seguindo, a ***OlsrRouteTable*** contém as rotas aprendidas através do protocolo OLSR. Aqui, rotas para todos os roteadores estão disponíveis, e apenas uma, a para o **Roteador 5**, tem métrica 2, como a coluna ***OlsrRouteMetric*** demonstra.

Informação vital está presente na tabela ***OlsrNeighborTable***, onde todos os vizinhos de um salto do nó estão caracterizados. É possível visualizar que o **Roteador 1** tem três vizinhos de um salto: o **Roteador 2**, o **Roteador 3** e o **Roteador 4**. Todos esses, de acordo com a coluna ***OlsrNeighborMPRS***, elegeram o **Roteador 1** como seu MPR. Isso era previsível, pois o **Roteador 1** é capaz de prover conectividade para os vizinhos de dois saltos de cada um desses nós. Porém, como pode ser visto na coluna ***OlsrNeighborMPR***, o **Roteador 1** também elegeu o **Roteador 3** como seu MPR. Isso justifica-se pelo fato de que o **Roteador 5**, vizinho de dois saltos do **Roteador 1**, só pode ser alcançado utilizando o **Roteador 3** como MPR.

A ***OlsrLinkTable*** expõe os enlaces do nó. Nesta situação, a única informação válida presente, além dos endereços IPs envolvidos no enlace, é os valores de histerese percebidos pelo **Roteador 1** para os enlaces com seus vizinhos de um salto (coluna ***OlsrLinkHysteresis***).

Finalmente, a tabela ***OlsrTopologyTable*** apresenta as entradas da **base de informações de controle de topologia**. Como visto no capítulo dois, essas informações são recebidas pelas mensagens TC e utilizadas para o cálculo de rotas. Em um ambiente em que não é utilizada a função auxiliar de redundância TC, ou em que o objeto ***TcRedundancy*** esteja marcado com o valor 0 (como é o caso em todos os nós deste cenário), apenas MPRs geram mensagens TC. Assim, como o **Roteador 1** é um MPR, mas não acrescenta suas próprias entradas na **base de informações de controle de topologia**, as únicas linhas da tabela presentes são aquelas originadas pelo único outro MPR no cenário: o **Roteador 3**.

Como observação importante, vale destacar que alguns objetos da OLSR MIB, embora presentes na leitura da Figura 5.6, são válidos somente quando o *OLSRD* estiver operando

segundo a extensão *Link Quality*. Para identificar tais parâmetros é necessário examinar suas descrições no documento da OLSR MIB (APÊNDICE A).

5.4.2. Escrita dos Objetos da OLSR MIB

Como próxima etapa dos experimentos, a modificação em tempo de execução de alguns objetos da OLSR MIB foi experimentada. Selecionou-se, então, para a exemplificação da funcionalidade de escrita, os seguintes objetos que permitem alteração: ***Willingness***, ***OlsrHNAAnnouncedTable*** e ***TcRedundancy***. O cenário para os testes foi o mesmo utilizado para a leitura. Porém, para possibilitar a melhor visualização da manipulação do objeto ***TcRedundancy***, a implementação da OLSR MIB também foi instalada no **Roteador 4**.

5.4.2.1. *Willingness*

Willingness é o valor que indica a inclinação do nó para encaminhar tráfego em nome dos outros dispositivos do domínio de roteamento OLSR. *Willingness* pode variar de 0 (nunca) até 7 (sempre), com 3 como opção padrão. Pela leitura apresentada na Figura 5.6, o **Roteador 1** apresenta o valor neutro de 3. Esse objeto será, portanto, manipulado para 0 e os resultados serão analisados. A Figura 5.7 mostra como a modificação é processada.

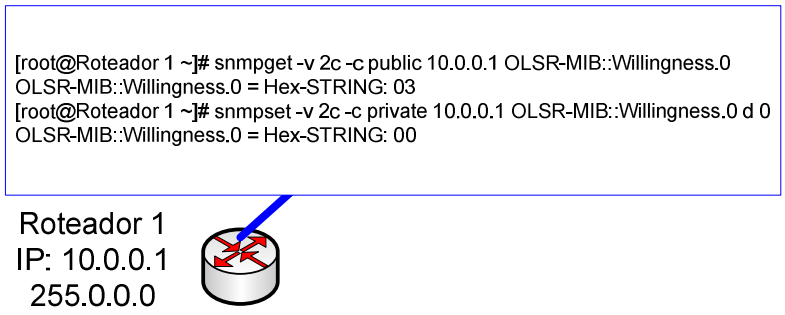

Figura 5.7 – Alteração do Objeto *Willingness*

Antes da alteração executada na Figura 5.7, a tabela de roteamento do **Roteador 4** foi

consultada e, após o procedimento de marcação, outra amostra foi obtida. Essas visualizações da tabela de roteamento demonstraram o efeito da modificação do objeto ***Willingness*** no **Roteador 1** e são expostas na Figura 5.8.

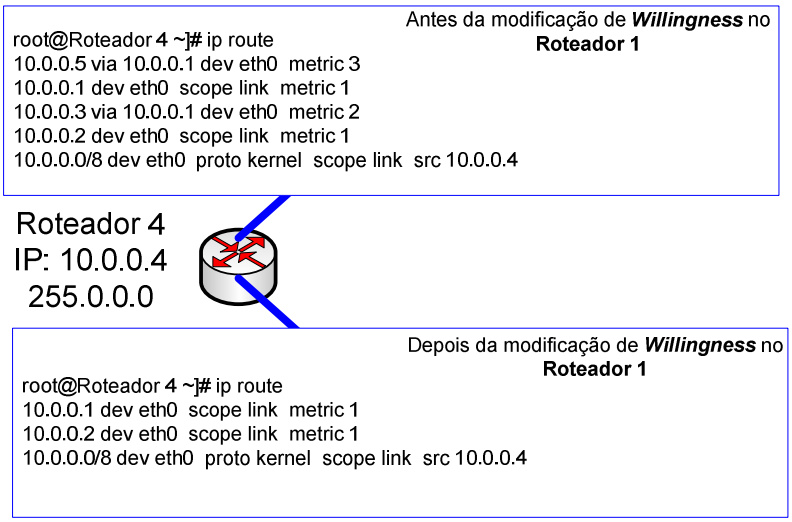

Figura 5.8 – Impacto da Modificação do Objeto *Willingness*

A manipulação do ***Willingness*** de um nó tem um grande impacto no ambiente de roteamento. Antes da alteração, o **Roteador 4** tinha rotas para alcançar os **Roteadores 5 e 3**, mas tais caminhos desaparecem quando o **Roteador 1** desiste de encaminhar tráfego em nome de outros nós. Os **Roteadores 2, 3 e 5** também foram afetados por essa mudança e, apesar de suas tabelas de roteamento não estarem expostas, todos os caminhos que apresentavam o **Roteador 1** como um nó intermediário desapareceram deles.

5.4.2.2. *OlsrHNAAnnouncedTable*

O segundo experimento de escrita foi a criação do anúncio de uma rede externa, utilizando a função auxiliar HNA descrita no segundo capítulo. Neste teste, uma nova rota padrão para a internet irá começar a ser anunciada pelo **Roteador 1**. Para isso, apenas uma nova linha na tabela ***OlsrHNAAnnouncedTable*** precisa ser criada. O comando mostrado na Figura 5.9

realiza tal tarefa.

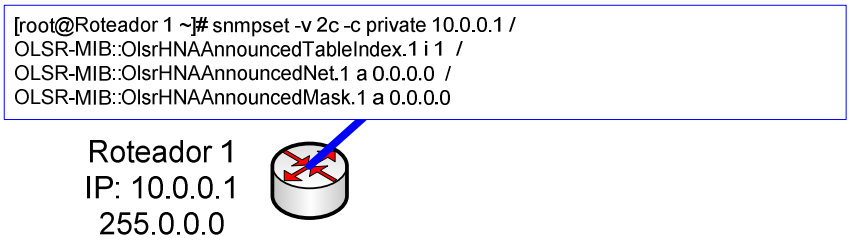

Figura 5.9 – A Criação de uma Nova HNA

Novamente, para analisar o impacto da criação de uma nova HNA no **Roteador 1**, duas amostras da tabela de roteamento do **Roteador 4** são obtidas. A operação de criação da linha fez com que o **Roteador 4** apresentasse uma nova rota padrão em seu kernel, como pode ser visto na Figura 5.10. Mais ainda, todos os outros roteadores também tiverem essa entrada adicionada em suas tabelas de roteamento.

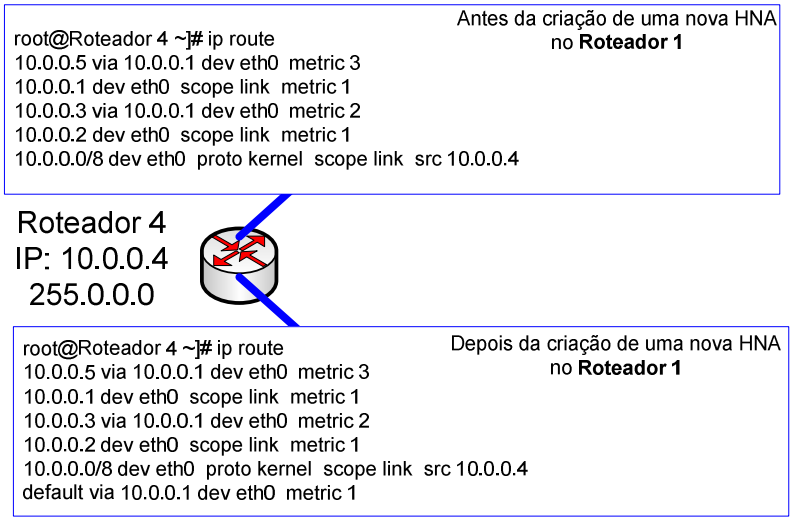

Figura 5.10 – Impacto de Criação de uma Nova HNA

5.4.2.3. *TcRedundancy*

Como último experimento, optou-se por testar a função auxiliar de redundância TC implementada pelo objeto ***TcRedundancy*** da OLSR MIB.

Na Figura 5.6 é apresentada a situação inicial do **Roteador 1**. Nela, a variável ***TcRedundancy*** está com o valor 0. Isso significa, segundo a descrição do objeto ***TcRedundancy*** na OLSR MIB (APÊNDICE A) e do parâmetro TC_REDUNDANCY da função auxiliar de redundância TC no capítulo dois, que apenas os nós MPRs irão gerar mensagens TC. Isso foi comprovado também no experimento de leitura onde se verificou que apenas existiam entradas de topologia na tabela ***OlsrTopologyTable*** do **Roteador 1** oriundas do outro único MPR do cenário, o **Roteador 3**.

Agora, alterar-se-á o valor de ***TcRedundancy*** do **Roteador 4** de 0 para 2 e será observada a inclusão de novas entradas de topologia no **Roteador 1**. A marcação com 2 significa que o nó em questão irá gerar mensagens TC anunciando todos os seus vizinhos. A Figura 5.11 mostra a alteração do valor de ***TcRedundancy*** no **Roteador 4**, e a Figura 5.12 mostra o impacto de tal alteração na tabela ***OlsrTopologyTable*** do **Roteador 1**.

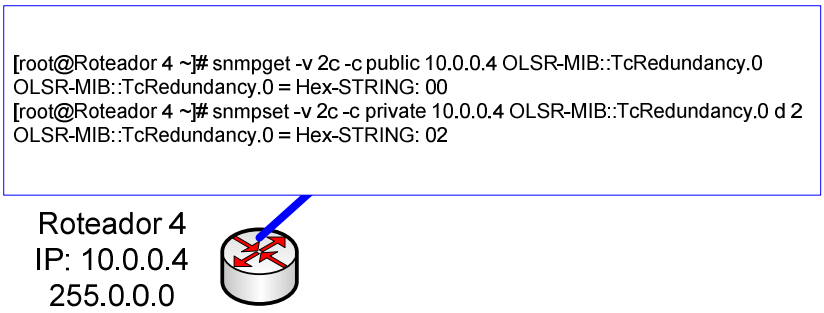

Figura 5.11 – Alteração do Objeto *TcRedundancy*

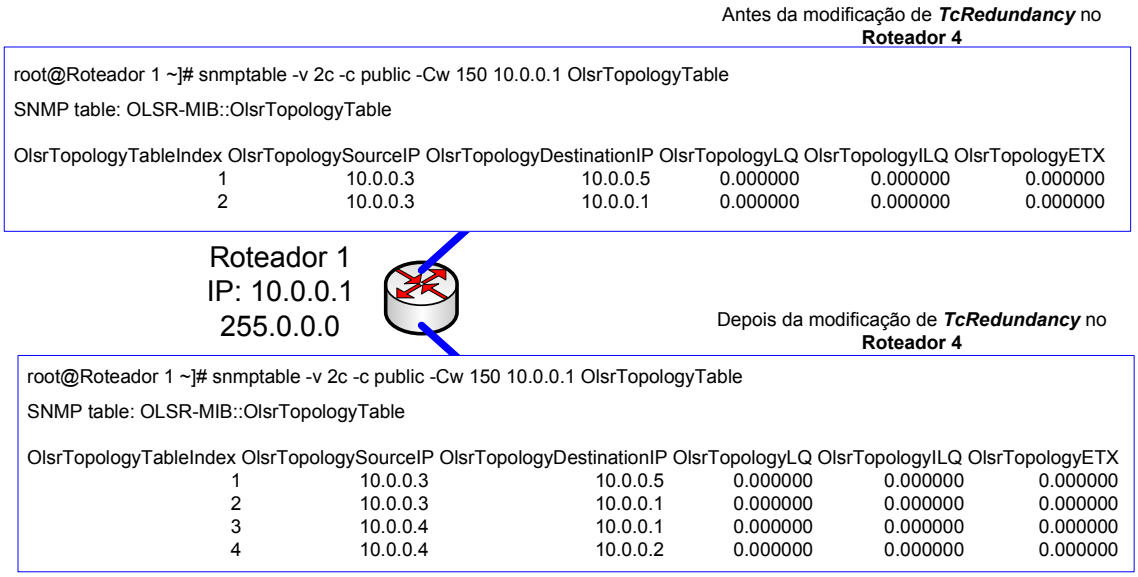

Figura 5.12 – Impacto da Alteração do Objeto *TcRedundancy*

5.5. RESULTADOS

Os experimentos realizados visaram validar tanto a coerência da OLSR MIB quanto a operacionalidade da sua implementação.

Para o documento da OLSR MIB (APÊNDICE A), observou-se, pela sua distribuição junto com o *software* que a implementa [26], que a oferecida abragência dos aspectos do protocolo OLSR tem sido elogiada pela comunidade que a utiliza. Um dos fatores importantes para tal destaque é a incorporação não somente dos objetos referentes estritamente à RFC 3626 [1], mas também dos parâmetros em conformidade com a extensão *Link Quality*. Além disso, a inclusão de objetos com permissão de escrita rendeu a esta MIB uma característica basilar não somente de avaliação, mas também de gerência completa. Seguindo, portanto, a evolução do documento, uma versão da OLSR MIB em RFC está em processo de elaboração para uma futura aprovação como padrão. Tal procedimento está sendo realizado seguindo a orientação do grupo de trabalho para MANET do IETF (*Internet Engineering Task Force*) [30], mais especificamente do presidente do grupo, Ian Chakeres.

A implementação da OLSR MIB, como exposto anteriormente, está disponível para a comunidade em [26]. Atualmente como um dos projetos OLSR do sourceforge.net com mais atividade e relevância como se pode verificar na Figura 5.13, o programa tem sido

utilizado e testado extensivamente. Os experimentos nesta dissertação processados objetivaram a demonstração do seu funcionamento. O teste de leitura dos objetos da OLSR MIB mostrou que a implementação pode eficazmente traduzir a situação do ambiente de roteamento pela visualização dos valores dos objetos gerenciados. Já o experimento de escrita demonstrou como a manipulação dos objetos da OLSR MIB pode ser feita com sucesso. Embora alguns resultados importantes derivados da alteração dos parâmetros do OLSR tenham sido observados, o objetivo do teste de modificação era apenas legitimar a funcionalidade de escrita. Destarte, a utilização dos demais objetos manipuláveis, bem como o emprego da combinação de leitura e escrita de vários parâmetros em conjunto, introduz uma ampla área para a evolução do gerenciamento do protocolo OLSR.

É importante, também, mencionar alguns aspectos relativos à perfomance da implementação da OLSR MIB. Como foi exposto durante a apresentação do programa, este é uma DLL e faz uso da interface *plugin* para povoar o repositório MIB. Todavia, a Tabela 5.1 demonstra que existe um mapeamento direto entre os objetos conceituais da OLSR MIB e as variáveis reais do *OLSRD*. Isso possibilita que dentro da implementação da OLSR MIB não sejam alocadas variáveis adicionais com a mesma informação presente na implementação do protocolo OLSR. Basta, então, o simples apontamento para os endereços de memória dos apropriados objetos dentro do repositório de informações do *OLSRD*. Em tais termos, a utilização de espaço de armazenamento pela DLL se torna consideralvelmente baixa, conferindo ao *software* um baixo requisito de memória; o que, por sua vez, é de relevância fundamental em um ambiente de recursos escassos.

Enfim, como resultado final mais recente, um artigo expondo tanto a OLSR MIB quanto a implementação foi aceito, apresentado e publicado na conferência ICCSA 2007 (*International Conference on Computational Science and Its Applications*) [28].

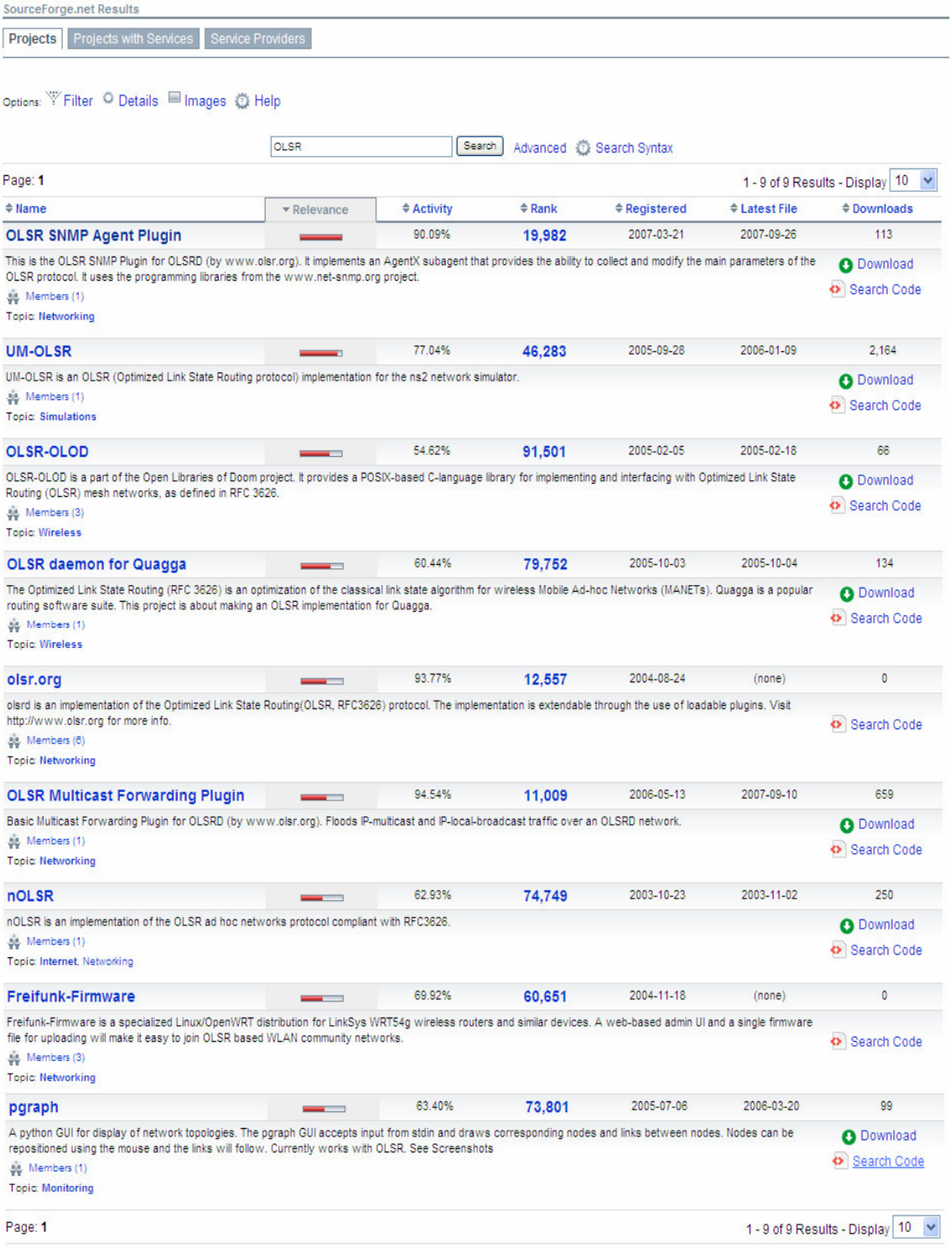

Figura 5.13 – Atividade e Relevância do Projeto [26] em Sourceforge.net

6. CONCLUSÕES

A introdução da possibilidade de monitoração e controle do protocolo OLSR foi a principal motivação deste trabalho. Como decorrência de sucesso, compilou-se a estrutura de gerenciamento para o protocolo OLSR. Os dois componentes principais de tal ambiente, a saber, a OLSR MIB e a implementação concernente, objetivam e alcançam, portanto, o suporte para o atual e futuro desenvolvimento da tecnologia MANET.

Tanto a OLSR MIB quanto a sua implementação foram detalhadas na extensão desta dissertação e tiveram seus pré-requisitos teóricos cumpridos anteriormente à sua apresentação. Foi possível, dessa maneira, a compreensão completa dos passos lógicos que levaram enfim à versão atual da estrutura de gerenciamento para o protocolo OLSR.

Mais ainda, como os resultados apresentados demonstraram, este ambiente de administração tem sido utilizado e, também, validado pela comunidade acadêmica. A grande atividade percebida no projeto disponível em [25] e a aprovação do artigo correspondente em [28] atestam, por conseguinte, este êxito.

Todavia, a inserção da contribuição aqui proposta no espaço intelectual da engenharia de redes não constitui um evento pontual, findo após sua disponibilização. É sim um processo contínuo, com atualizações e desdobramentos esperados para o futuro. Dessa forma, a MIB do protocolo e a implementação da OLSR MIB deverão ser adaptadas para as eventuais mudanças tanto do protocolo OLSR (a segunda versão, OLSRv2, já se encontra em vias de aprovação [29]) quanto da implementação do mesmo, a *OLSRD*. Um documento em formato RFC para a OLSR MIB, também, já está em processo de elaboração. E, finalmente, utilizando a estrutura de gerenciamento para o protocolo OLSR como ferramenta padrão de interfaceamento com protocolo OLSR, sistemas IDS (*Intrusion Detection Systems*) e NMS (*Network Management Systems*) poderão ser mais facilmente desenvolvidos.

Conclui-se, destarte, aqui esta dissertação e espera-se que a estrutura de gerenciamento para o protocolo OLSR proposta contribua para a solidificação e promoção do emprego da tecnologia MANET em ambientes acadêmicos e comerciais onde o protocolo OLSR esteja presente.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] CLAUSEN, T.; JACQUET, P. *Optimized Link State Routing Protocol (OLSR)*, RFC 3626, Outubro de 2003.

[2] TØNNESEN, A. *Implementing and extending the Optimized Link State Routing Protocol* – UniK University Graduate Center – University of Oslo, Oslo, Agosto de 2004.

[3] *Internet Protocol – Darpa Internet Program – Protocol Specification*, RFC 791, Setembro de 1981.

[4] DEERING, S.; HINDEN, R. *Internet Protocol, Version 6 – Specification*, RFC 2460, Dezembro de 1998.

[5] POSTEL, J. *User Datagram Protocol*, RFC 768, Agosto de 1980.

[6] VIENNOT, L. *Complexity results on election of multipoint relays in wireless networks* – Technical report, INRIA, 1998.

[7] *NRL OLSR implementation*, Acessado em 31/06/2007 em http://cs.itd.nrl.navy.mil/work/olsr/index.php.

[8] *QoS OLSR implementation*, Acessado em 31/06/2007 em http://qolsr.lri.fr/.

[9] *The BSD License,* Acessado em 10/07/2007 em http://www.opensource.org/licenses/bsd-license.php.

[10] WINETT, J. *The Definition of a Socket*, RFC 147, Maio de 1971.

[11] WAGNER, S. *Free (Wireless) Networks, Workshop* – Wizards of OS III, Berlin, Junho de 2004.

[12] TØNNESEN, A. *OLSRD Link Quality Extensions*. Acessado em 31/07/2007 em http://www.olsr.org/docs/README-Link-Quality.html.

[13] *The ETX Protocol*, Acessado em 01/08/2007 em http://www.cuwin.net/manual/techdocs/etx.

[14] DECOUTO, D.; AGUAYO, D.; BICKET, J.; MORRIS, R. *A high-throughput path metric for multi-hop wireless networks* – Proc. Of Mobicom, 2003.

[15] TØNNESEN, A; HAFSLUND, A.; KURE, Ø. *The Unik OLSR plugin interface* – Proc. Of OLSR Interop and Workshop, Berlim, Junho de 2004.

[16] COMER, D. E. *Internetworking with TCP/IP – Principles, Protocols, and Architectures* – 4ª edição. Prentice Hall, Upper Saddle River, New Jersey.

[17] MCCLOGHRIE, K.; ROSE, M. *Management Information Base for Network Management of TCP/IP-based internets*, RFC 1156, Maio de 1990.

[18] MCCLOGHRIE, K.; ROSE, M. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC 1213, Março de 1991.

[19] ROSE, M.; MCCLOGHRIE, K. *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC 1155, Maio de 1990.

[20] MCCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. *Structure of Management Information Version 2 (SMIv2)*, RFC 2578, Abril de 1999.

[21] CASE, J.; FEDOR, M.; SHOFFSTALL, M.; DAVIN, J. *A Simple Network Management Protocol (SNMP)*, RFC 1157, Maio de 1990.

[22] CASE, J.; MCCLOGHRIE, K.; ROSE, M.; WALDBUSSER, S. *Introduction to Community-based SNMPv2*, RFC 1901, Janeiro de 1996.

[23] FRYE, R.; LEVI, D.; ROUTHIER, S.; WIJNEN, D. *Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework*, RFC 3584, Agosto de 2003.

[24] DANIELE, M.; WIJNEN, B.; ELLISON, M.; FRANCISCO, D. *Agent Extensibility (AgentX) Protocol Version 1*, RFC 2741, Janeiro de 2000.

[25] *Net-SNMP – SNMP implementation*, Acessado em 27/09/2007 em http://www.net-snmp.org.

[26] PACHECO, V.; PUTTINI, R. *OLSR SNMP Agent Plugin*, Projeto SourceForge.net, Acessado em 30/09/2007 em http://sourceforge.net/projects/olsrd-snmpd.

[27] *Fedora Project*, Acessado em 04/10/2007 em http://fedoraproject.org.

[28] PACHECO, V.; PUTTINI, R. *An Administration Structure for the OLSR Protocol* – Proc. Of ICCSA, 2007.

[29] CLAUSEN, T.; DEARLOVE, C.; JACQUET, P. *The Optimized Link State Routing Protocol version 2*, Internet-Draft, Julho de 2007

[30] CHAKERES, I.; MACKER, J. *MANET IETF WORKING GROUP*, Grupo de Trabalho, IETF, Acessado em 19/11/2007 em http://tools.ietf.org/wg/manet.

[31] BADONNEL, R.; STATE, R.; FESTOR, O. *Management of mobile ad hoc networks: information model and probe-based architecture*, Int. J. Network Mgmt., 2005.

APÊNDICES

A – OLSR MIB

```
OLSR-MIB DEFINITIONS ::= BEGIN

IMPORTS
  MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
  IpAddress, Integer32, Gauge32, enterprises
  Counter32, experimental,
  TimeTicks, snmpModules                    FROM SNMPv2-SMI

  RowStatus, DisplayString,
  TestAndIncr, TimeStamp,
  TEXTUAL-CONVENTION, TruthValue,
  DateAndTime, AutonomousType               FROM SNMPv2-TC

  MODULE-COMPLIANCE, OBJECT-GROUP,
  NOTIFICATION-GROUP                        FROM SNMPv2-CONF

  SnmpAdminString                           FROM SNMP-FRAMEWORK-MIB

  InterfaceIndexOrZero                      FROM IF-MIB

  ip                                        FROM IP-MIB

  IANAipRouteProtocol                       FROM IANA-RTPROTO-MIB

  InetAddress, InetAddressType,
  InetAddressPrefixLength,
  InetAutonomousSystemNumber                FROM INET-ADDRESS-MIB;


OLSR-MIB MODULE-IDENTITY
  LAST-UPDATED "02070511457Z"
  ORGANIZATION "UnB"
  CONTACT-INFO "vinicius.pacheco@gmail.com, puttini@unb.br"
  DESCRIPTION "The MIB module for the OLSR protocol - Vinicius Maia Pacheco, Ricardo
Puttini." ::= { enterprises 9363 }

br OBJECT IDENTIFIER ::= { enterprises 9363 }

unb OBJECT IDENTIFIER ::= { br 1 }

manet OBJECT IDENTIFIER ::= { unb 1 }

--the olsr group

olsr OBJECT IDENTIFIER ::= { manet 1 }


       MainAddress OBJECT-TYPE
       SYNTAX IpAddress
       MAX-ACCESS read-only
       STATUS current
       DESCRIPTION "The main address of a node, which will be used in OLSR control
traffic as the originator address of all messages emitted by this node. It is the
address of one of the OLSR interfaces of the node. A single OLSR interface node MUST use
the address of its only OLSR interface as the main address. A multiple OLSR interface
node MUST choose one of its OLSR interface addresses as its main address (equivalent of
router ID or node identifier). It is of no importance which address is chosen, however a
node SHOULD always use the same address as its main address."
       ::= { olsr 1 }

       IpVersion OBJECT-TYPE
       SYNTAX Integer32
       MAX-ACCESS read-only
       STATUS current
       DESCRIPTION "Olsrd supports both IP version 4 and 6. This option shows what IP
version olsrd is using. Defaults to 4."
       ::= { olsr 2 }
```

```
        Pollrate OBJECT-TYPE
        SYNTAX Opaque
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "This option presents the interval, in seconds, that the olsrd event
scheduler is set to poll. A setting of 0.2 will set olsrd to poll for events every 0.2
seconds. Defaults to 0.05. Should be always smaller than 1 and bigger than 0."
        ::= { olsr 3 }

        TcRedundancy OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "This value controls the TC redundancy used by the local node in TC
message generation. To enable a more robust understanding of the topology, nodes can be
set to announce more than just their MPR selector set in TC messages. If set to 0 the
advertised link set of the node is limited to the MPR selectors. If set to 1 the
advertised link set of the node is the union of its MPR set and its MPR selector set.
Finally, if set to 2 the advertised link set of the node is the full symmetric neighbor
set of the node. Defaults to 0."
        ::= { olsr 4 }

        MprCoverage OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "This value decides how many MPRs a node should attempt to select for
every two hop neighbor. Defaults to 1 , and any other setting will severely reduce the
optimization introduced by the MPR scheme."
        ::= { olsr 5 }

        TosValue OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "This value controls the type of service value to set in the IP
header of OLSR control traffic. Defaults to 16."
        ::= { olsr 6 }

        Willingness OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "Nodes participating in an OLSR routed network will announce their
willingness to act as relays for OLSR control traffic for their neighbors. This option
specifies a fixed willingness value to be announced by the local node.  3 is a neutral
option here, while 0 specifies that this node will never act as a relay, and 7 specifies
that this node will always act as such a relay."
        ::= { olsr 7 }

        UseHysteresis OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "The link between a node and some of its neighbor interfaces might be
bad, i.e., from time to time let HELLOs pass through only to fade out immediately after.
In this case, the neighbor information base would contain a bad link for at least
validity time. The hysteresis strategy SHOULD be adopted to counter this situation. If
set to 01 00 00 00 hysteresis will be used, and if set to 00 00 00 00 it will be
disabled."
        ::= { olsr 8 }

        HystScaling OBJECT-TYPE
        SYNTAX Opaque
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "Sets the scaling value used by the hysteresis algorithm. This must
be a positive floating point value smaller than 1.0."
        ::= { olsr 9 }

        HystThrLow OBJECT-TYPE
        SYNTAX Opaque
```

```
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "This option sets the lower threshold for setting a link to
asymmetric using hysteresis. The value must be lower than the one set as the upper
threshold. Defaults to 0.3"
        ::= { olsr 10 }

        HystThrHigh OBJECT-TYPE
        SYNTAX Opaque
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "This option sets the upper threshold for accepting a link in
hysteresis calculation. The value must be higher than the one set as the lower
threshold. Defaults to 0.8."
        ::= { olsr 11 }

        LinkQualityLevel OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "This setting decides the Link Quality scheme to use. If set to 0
link quality is not regarded and olsrd runs in RFC3626 mode. If set to 1 link quality is
used when calculating MPRs. If set to 2 routes will also be calculated based on
distributed link quality information. Note that a setting of 1 or 2 breaks RFC3626
compability! This option should therefore only be set to 1 or 2 if such a setting is
used by all other nodes in the network."
        ::= { olsr 12 }

        LinkQualityWinSize OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "Link quality window size. Defaults to 10. When determining the
packet loss of the packets received from a neighbor, olsrd only looks at the n most
recent packets. By default n is set to 10, so olsrd looks at the ten most recent packets
received (or not received) from a neighbor and then determines the packet loss. Let's
assume that of the 10 packets we have received 7, then we have missed 3, which
corresponds to a packet loss of 3/10 = 0.3 = 30%. The corresponding Link Quality is 7/10
= 0.7 = 70%. Note that this option is not RFC compliant. Only valid if LinkQualityLevel
is set to 1 or 2."
        ::= { olsr 13 }

        LinkQualityFishEye OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "Enables(1) or disables(0) the use of the experimental Fish Eye
algorithm. Link Quality Fish Eye is a new (experimental) algorithm introduced in olsrd
0.4.10. To increase stability in a mesh, TC messages should be sent quite frequently.
However, the network would then suffer from the resulting overhead. The idea is to
frequently send TC messages to adjacent nodes, i.e. nodes that are likely to be involved
in routing loops, without flooding the whole mesh with each sent TC message. OLSR
packets carry a Time To Live (TTL) that specifies the maximal number of hops that the
packets is allowed to travel in the mesh. The Link Quality Fish Eye mechanism generates
TC messages not only with the default TTL of 255, but with different TTLs, namely 1, 2,
3, and 255, restricting the distribution of TC messages to nodes 1, 2, 3, and 255 hops
away. A TC message with a TTL of 1 will just travel to all one-hop neighbors, a message
with a TTL of 2 will in addition reach all two-hop neighbors, etc. TC messages with
small TTLs are sent more frequently than TC messages with higher TTLs, such that
immediate neighbors are more up to date with respect to our links than the rest of the
mesh. We hope that this reduces the likelihood of routing loops. Note that this option
is not RFC compliant. Only valid if LinkQualityLevel is set to 1 or 2."
        ::= { olsr 14 }

        LinkQualityDijkstraLimitLimit OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "The parameter tells the algorithm to recalculate the Dijkstra-table
when a TC-Message arrives from a node not more than n hops away. If set to 0 it will not
recalculate it upon incoming TCs. Note that this option is not RFC compliant. Only valid
if LinkQualityLevel is set to 1 or 2."
```

```
        ::= { olsr 15 }

        LinkQualityDijkstraLimitInterval OBJECT-TYPE
        SYNTAX Opaque
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION "The parameter sets the time interval when the Dijkstra table will be
recalculated anyway. Note that this option is not RFC compliant. Only valid if
LinkQualityLevel is set to 1 or 2."
        ::= { olsr 16 }

        OlsrInterfaceTable OBJECT-TYPE
        SYNTAX SEQUENCE OF OlsrInterfaceTableEntry
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION  "A table containing the OLSR protocol values applied to each
interface in the node."
        ::= { olsr 17 }

                OlsrInterfaceTableEntry OBJECT-TYPE
                SYNTAX  OlsrInterfaceTableEntry
                MAX-ACCESS read-only
                STATUS current
                DESCRIPTION "A conceptual row of the OlsrInterfaceTableEntry."
                INDEX { OlsrInterfaceTableIndex }
                ::= { OlsrInterfaceTable 1 }

                OlsrInterfaceTableEntry ::= SEQUENCE {
                OlsrInterfaceTableIndex              Integer32,
                OlsrInterfaceName                    DisplayString,
                OlsrInterfaceIP                      IpAddress,
                OlsrInterfaceMask                    IpAddress,
                OlsrInterfaceBroadcast               IpAddress,
                OlsrInterfaceMTU                     Integer32,
                OlsrInterfaceWireless                Integer32,
                OlsrInterfaceHelloEmission           Opaque,
                OlsrInterfaceHelloValidity           Opaque,
                OlsrInterfaceTCEmission              Opaque,
                OlsrInterfaceTCValidity              Opaque,
                OlsrInterfaceMIDEmission             Opaque,
                OlsrInterfaceMIDValidity             Opaque,
                OlsrInterfaceHNAEmission             Opaque,
                OlsrInterfaceHNAValidity             Opaque
                }

                        OlsrInterfaceTableIndex OBJECT-TYPE
                        SYNTAX Integer32
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The integer index of the OlsrInterfaceTable."
                        ::= { OlsrInterfaceTableEntry 1 }

                        OlsrInterfaceName OBJECT-TYPE
                        SYNTAX DisplayString
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The name of the interface."
                        ::= { OlsrInterfaceTableEntry 2 }

                        OlsrInterfaceIP OBJECT-TYPE
                        SYNTAX IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The IP address of the interface."
                        ::= { OlsrInterfaceTableEntry 3 }

                        OlsrInterfaceMask OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The network mask of the interface."
                        ::= { OlsrInterfaceTableEntry 4 }
```

```
                    OlsrInterfaceBroadcast OBJECT-TYPE
                    SYNTAX  IpAddress
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The broadcast address of the interface"
                    ::= { OlsrInterfaceTableEntry 5 }

                    OlsrInterfaceMTU OBJECT-TYPE
                    SYNTAX Integer32
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The MTU being used by the interface"
                    ::= { OlsrInterfaceTableEntry 6 }

                    OlsrInterfaceWireless OBJECT-TYPE
                    SYNTAX Integer32
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "1 if the interface is wireless and 0 if it is not."
                    ::= { OlsrInterfaceTableEntry 7 }

                    OlsrInterfaceHelloEmission OBJECT-TYPE
                    SYNTAX Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The interval in seconds that a HELLO message will be
emitted."
                    ::= { OlsrInterfaceTableEntry 8 }

                    OlsrInterfaceHelloValidity OBJECT-TYPE
                    SYNTAX Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The interval in seconds that a HELLO message will be
considered valid."
                    ::= { OlsrInterfaceTableEntry 9 }

                    OlsrInterfaceTCEmission OBJECT-TYPE
                    SYNTAX Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The interval in seconds that a TC message will be
emitted."
                    ::= { OlsrInterfaceTableEntry 10 }

                    OlsrInterfaceTCValidity OBJECT-TYPE
                    SYNTAX Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The interval in seconds that a TC message will be
considered valid."
                    ::= { OlsrInterfaceTableEntry 11 }

                    OlsrInterfaceMIDEmission OBJECT-TYPE
                    SYNTAX Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The interval in seconds that a MID message will be
emitted."
                    ::= { OlsrInterfaceTableEntry 12 }

                    OlsrInterfaceMIDValidity OBJECT-TYPE
                    SYNTAX Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The interval in seconds that a MID message will be
considered valid."
                    ::= { OlsrInterfaceTableEntry 13 }

                    OlsrInterfaceHNAEmission OBJECT-TYPE
                    SYNTAX Opaque
```

```
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The interval in seconds that a HNA message will be
emitted."
                        ::= { OlsrInterfaceTableEntry 14 }

                        OlsrInterfaceHNAValidity OBJECT-TYPE
                        SYNTAX Opaque
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The interval in seconds that a HNA message will be
considered valid."
                        ::= { OlsrInterfaceTableEntry 15 }

        OlsrHNAAnnouncedTable OBJECT-TYPE
        SYNTAX SEQUENCE OF OlsrHNAAnnouncedTableEntry
        MAX-ACCESS read-create
        STATUS current
        DESCRIPTION "A table containing the HNAs announced by the node."
        ::= { olsr 18 }

                OlsrHNAAnnouncedTableEntry OBJECT-TYPE
                SYNTAX  OlsrHNAAnnouncedTableEntry
                MAX-ACCESS read-create
                STATUS current
                DESCRIPTION "A conceptual row of the OlsrHNAAnnouncedTableEntry."
                INDEX { OlsrHNAAnnouncedTableIndex }
                ::= { OlsrHNAAnnouncedTable 1 }

                OlsrHNAAnnouncedTableEntry ::= SEQUENCE {
                OlsrHNAAnnouncedTableIndex          Integer32,
                OlsrHNAAnnouncedNet                 IpAddress,
                OlsrHNAAnnouncedMask                IpAddress
                }

                        OlsrHNAAnnouncedTableIndex OBJECT-TYPE
                        SYNTAX  Integer32
                        MAX-ACCESS read-create
                        STATUS current
                        DESCRIPTION "The integer index of the OlsrHNAAnnouncedTable."
                        ::= { OlsrHNAAnnouncedTableEntry 1 }

                        OlsrHNAAnnouncedNet OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-create
                        STATUS current
                        DESCRIPTION "The network address being announced by the node."
                        ::= { OlsrHNAAnnouncedTableEntry 2 }

                        OlsrHNAAnnouncedMask OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-create
                        STATUS current
                        DESCRIPTION "The network mask of the network being announced by
the node."
                        ::= { OlsrHNAAnnouncedTableEntry 3 }

        OlsrRouteTable OBJECT-TYPE
        SYNTAX SEQUENCE OF OlsrRouteTableEntry
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The table containing the routes learned from the OLSR protocol by
the node."
        ::= { olsr 19 }

                OlsrRouteTableEntry OBJECT-TYPE
                SYNTAX  OlsrRouteTableEntry
                MAX-ACCESS read-only
                STATUS current
                DESCRIPTION "A conceptual row of the OlsrRouteTableEntry."
                INDEX { OlsrRouteTableIndex }
                ::= { OlsrRouteTable 1 }
```

```
OlsrRouteTableEntry ::= SEQUENCE {
OlsrRouteTableIndex          Integer32,
OlsrRouteDestination         IpAddress,
OlsrRouteGateway             IpAddress,
OlsrRouteMetric              Integer32,
OlsrRouteETX                 Opaque,
OlsrRouteType                DisplayString,
OlsrRouteInterface           DisplayString
}

        OlsrRouteTableIndex OBJECT-TYPE
        SYNTAX   Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The integer index of the OlsrRouteTable."
        ::= { OlsrRouteTableEntry 1 }

        OlsrRouteDestination OBJECT-TYPE
        SYNTAX  IpAddress
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The IP address of the destination."
        ::= { OlsrRouteTableEntry 2 }

        OlsrRouteGateway OBJECT-TYPE
        SYNTAX  IpAddress
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The IP address of the gateway to be used in order to
achieve the destination."
        ::= { OlsrRouteTableEntry 3 }

        OlsrRouteMetric OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The number of hops that are between the node and the
destination."
        ::= { OlsrRouteTableEntry 4 }

        OlsrRouteETX OBJECT-TYPE
        SYNTAX Opaque
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The Expected Transmission Count or ETX of the route
is defined as how often is our packet retransmitted on its way from the node to the
destination. It is the sum of all links ETXs from the node to the destination. Note that
this option is not RFC compliant. Only valid if LinkQualityLevel is set to 1 or 2."
        ::= { OlsrRouteTableEntry 5 }

        OlsrRouteType OBJECT-TYPE
        SYNTAX DisplayString
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The type of the route learned. If it is a route to a
node it is presented as HOST, and if it is a route to a network learned via HNA messages
it is presented as HNA."
        ::= { OlsrRouteTableEntry 6 }

        OlsrRouteInterface OBJECT-TYPE
        SYNTAX DisplayString
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "The name of the interface to be used in order to
achieve the destination."
        ::= { OlsrRouteTableEntry 7 }

    OlsrNeighborTable OBJECT-TYPE
    SYNTAX SEQUENCE OF OlsrNeighborTableEntry
    MAX-ACCESS read-only
    STATUS current
```

```
        DESCRIPTION "A table containing the neighbors known by the node."
        ::= { olsr 20 }

                OlsrNeighborTableEntry OBJECT-TYPE
                SYNTAX  OlsrNeighborTableEntry
                MAX-ACCESS read-only
                STATUS current
                DESCRIPTION "A conceptual row of the OlsrNeighborTableEntry."
                INDEX { OlsrNeighborTableIndex }
                ::= { OlsrNeighborTable 1 }

                OlsrNeighborTableEntry ::= SEQUENCE {
                OlsrNeighborTableIndex              Integer32,
                OlsrNeighborIP                      IpAddress,
                OlsrNeighborSYM                     DisplayString,
                OlsrNeighborMPR                     DisplayString,
                OlsrNeighborMPRS                    DisplayString,
                OlsrNeighborWillingness             OCTET STRING,
                OlsrNeighbor2HopNeighbors           DisplayString
                }

                        OlsrNeighborTableIndex OBJECT-TYPE
                        SYNTAX  Integer32
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The integer index of the OlsrNeighborTable."
                        ::= { OlsrNeighborTableEntry 1 }

                        OlsrNeighborIP OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The IP address of the neighbor."
                        ::= { OlsrNeighborTableEntry 2 }

                        OlsrNeighborSYM OBJECT-TYPE
                        SYNTAX  DisplayString
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "If presented as YES indicates that the neighbor has a
symmetric link to the node, and if presented as NO, the neighbor has the link to the
node considered asymmetric."
                        ::= { OlsrNeighborTableEntry 3 }

                        OlsrNeighborMPR OBJECT-TYPE
                        SYNTAX DisplayString
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "If presented as YES indicates that the neighbor has
been selected as MPR for the node."
                        ::= { OlsrNeighborTableEntry 4 }

                        OlsrNeighborMPRS OBJECT-TYPE
                        SYNTAX DisplayString
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "If presented as YES indicates that the neighbor has
selected the node as his MPR."
                        ::= { OlsrNeighborTableEntry 5 }

                        OlsrNeighborWillingness OBJECT-TYPE
                        SYNTAX OCTET STRING
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "Presents the willingness of the neighbor."
                        ::= { OlsrNeighborTableEntry 6 }

                        OlsrNeighbor2HopNeighbors OBJECT-TYPE
                        SYNTAX DisplayString
                        MAX-ACCESS read-only
                        STATUS current
```

```
                    DESCRIPTION "Shows the IP addresses of the 2 hop neighbors that
can be reached through this neighbor by the node. The addresses will be shown as a
string of IPs separated by a -."
                    ::= { OlsrNeighborTableEntry 7 }

        OlsrLinkTable OBJECT-TYPE
        SYNTAX SEQUENCE OF OlsrLinkTableEntry
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "A table containing the links known to the node."
        ::= { olsr 21 }

                OlsrLinkTableEntry OBJECT-TYPE
                SYNTAX  OlsrLinkTableEntry
                MAX-ACCESS read-only
                STATUS current
                DESCRIPTION "A conceptual row of the OlsrLinkTableEntry."
                INDEX { OlsrLinkTableIndex }
                ::= { OlsrLinkTable 1 }

                OlsrLinkTableEntry ::= SEQUENCE {
                OlsrLinkTableIndex           Integer32,
                OlsrLinkLocalIP              IpAddress,
                OlsrLinkRemoteIP             IpAddress,
                OlsrLinkHysteresis           Opaque,
                OlsrLinkLinkQuality          Opaque,
                OlsrLinkLostPackets          Integer32,
                OlsrLinkTotalPackets         Integer32,
                OlsrLinkNLQ                  Opaque,
                OlsrLinkETX                  Opaque
                }

                        OlsrLinkTableIndex OBJECT-TYPE
                        SYNTAX  Integer32
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The integer index of the OlsrLinkTable."
                        ::= { OlsrLinkTableEntry 1 }

                        OlsrLinkLocalIP OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The local IP address of the link."
                        ::= { OlsrLinkTableEntry 2 }

                        OlsrLinkRemoteIP OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The remote IP address of the link."
                        ::= { OlsrLinkTableEntry 3 }

                        OlsrLinkHysteresis OBJECT-TYPE
                        SYNTAX  Opaque
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The current value of the hysteresis applied to the
link."
                        ::= { OlsrLinkTableEntry 4 }

                        OlsrLinkLinkQuality OBJECT-TYPE
                        SYNTAX  Opaque
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "Varies from 0 to 1. Represents the probability that a
packet that our neighbor sends actually makes it to node. Note that this option is not
RFC compliant. Only valid if LinkQualityLevel is set to 1 or 2."
                        ::= { OlsrLinkTableEntry 5 }

                        OlsrLinkLostPackets OBJECT-TYPE
                        SYNTAX  Integer32
```

```
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The number of packets that are sent from the neighbor
that are not received by the node. Note that this option is not RFC compliant. Only
valid if LinkQualityLevel is set to 1 or 2."
                    ::= { OlsrLinkTableEntry 6 }

                    OlsrLinkTotalPackets OBJECT-TYPE
                    SYNTAX  Integer32
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "This option is set in LinkQualityWinSize and informs
the total number of recent packets considered in the calculation of the Link Quality LQ
for the link. Note that this option is not RFC compliant. Only valid if LinkQualityLevel
is set to 1 or 2."
                    ::= { OlsrLinkTableEntry 7 }

                    OlsrLinkNLQ OBJECT-TYPE
                    SYNTAX  Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "Varies from 0 to 1. Represents the probability that a
packet that the node sends actually makes it to the neighbor. Note that this option is
not RFC compliant. Only valid if LinkQualityLevel is set to 1 or 2."
                    ::= { OlsrLinkTableEntry 8 }

                    OlsrLinkETX OBJECT-TYPE
                    SYNTAX  Opaque
                    MAX-ACCESS read-only
                    STATUS current
                    DESCRIPTION "The value 1 / (NLQ x LQ) is called the Expected
Transmission Count or ETX . Note that this option is not RFC compliant. Only valid if
LinkQualityLevel is set to 1 or 2."
                    ::= { OlsrLinkTableEntry 9 }

        OlsrTopologyTable OBJECT-TYPE
        SYNTAX SEQUENCE OF OlsrTopologyTableEntry
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION "A table containing the topology entries of the node."
        ::= { olsr 22 }

                OlsrTopologyTableEntry OBJECT-TYPE
                SYNTAX  OlsrTopologyTableEntry
                MAX-ACCESS read-only
                STATUS current
                DESCRIPTION "A conceptual row of the OlsrTopologyTableEntry."
                INDEX { OlsrTopologyTableIndex }
                ::= { OlsrTopologyTable 1 }

                OlsrTopologyTableEntry ::= SEQUENCE {
                OlsrTopologyTableIndex       Integer32,
                OlsrTopologySourceIP         IpAddress,
                OlsrTopologyDestinationIP    IpAddress,
                OlsrTopologyLQ               Opaque,
                OlsrTopologyILQ              Opaque,
                OlsrTopologyETX              Opaque
                }

                        OlsrTopologyTableIndex OBJECT-TYPE
                        SYNTAX  Integer32
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The integer index of the OlsrTopologyTable."
                        ::= { OlsrTopologyTableEntry 1 }

                        OlsrTopologySourceIP OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The node that reports a link."
                        ::= { OlsrTopologyTableEntry 2 }
```

```
                        OlsrTopologyDestinationIP OBJECT-TYPE
                        SYNTAX  IpAddress
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The node to which the source node reports the link."
                        ::= { OlsrTopologyTableEntry 3 }

                        OlsrTopologyLQ OBJECT-TYPE
                        SYNTAX  Opaque
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The quality of the link as determined by the source
node. For the source node this is the Link Quality. For the destination node this is the
Neighbor Link Quality. Note that this option is not RFC compliant. Only valid if
LinkQualityLevel is set to 1 or 2."
                        ::= { OlsrTopologyTableEntry 4 }

                        OlsrTopologyILQ OBJECT-TYPE
                        SYNTAX  Opaque
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The quality of the link as determined by the
destination node. For the source node this is the Neighbor Link Quality. For the
destination node this is the Link Quality. It is not named NLQ, as NLQ is only used for
the link quality reported by neighbors. But functionality is equivalent to the NLQ from
the link and neighbor tables. Represents the probability that a packet that our neighbor
sends actually makes it to node. Note that this option is not RFC compliant. Only valid
if LinkQualityLevel is set to 1 or 2."
                        ::= { OlsrTopologyTableEntry 5 }

                        OlsrTopologyETX OBJECT-TYPE
                        SYNTAX  Opaque
                        MAX-ACCESS read-only
                        STATUS current
                        DESCRIPTION "The ETX value for this link, calculated by ETX = 1 /
(ILQ x LQ). Note that this option is not RFC compliant. Only valid if LinkQualityLevel
is set to 1 or 2."
                        ::= { OlsrTopologyTableEntry 6 }

END
```

B – OLSRD_SNMPD_AGENTX.C

```
/*
 * OLSR SNMP Agent Plugin.
 * Copyright (c) 2006, 2007, Vinicius Maia Pacheco <vinicius.pacheco@gmail.com>
 *                           Ricardo Puttini <puttini@unb.br>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 * * Neither the name of the OLSR SNMP Agent Plugin nor the names of its contributors
 *   may be used to endorse or promote products derived from this software
 *   without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */


/*
 * File:       olsrd_snmpd_agentx.c
 * Description:       Plugin part of the OLSR SNMP Agent Plugin
 * Created:    04 October 2006
 */

#include <olsrd_snmpd_agentx.h>



/*****************************************************************************
 *                  Functions that the plugin MUST provide                   *
 *****************************************************************************/

/**
 * Plugin interface version
 * Used by main olsrd to check plugin interface version
 */
int
olsrd_plugin_interface_version(void)
{
       return OLSRD_PLUGIN_INTERFACE_VERSION;
}


/**
 * Register parameters from config file
 * Called for all plugin parameters
 */
int
olsrd_plugin_register_param(char *key, char *value)
{
       if(!strcmp(key, "poll")) {
              plugin_poll = atoi(value);
              olsr_printf(3, "\n*** OLSRD_SNMPD: plugin polling: %.2f :", plugin_poll);
```

```
                return 1;
        }
        return 0;
}


/**
 * Initialize plugin
 * Called after all parameters are passed
 */
int
olsrd_plugin_init(void)
{
        /* Plugin init variables */
        struct olsr_if *ifs;

        /* End of Plugin init variables */


        printf("*** OLSRD_SNMPD: plugin_init\n");


        /* AgentX initialization*/
        snmp_enable_stderrlog();
        netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID, NETSNMP_DS_AGENT_ROLE, 1);
        init_agent("olsrd_snmpd_agentx");
        init_olsrd_mib();
        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
        olsr_printf(3, "*** OLSRD_SNMPD: olsrd_snmpd_agentx AgentX conecting:\n");
        init_snmp("olsrd_snmpd_agentx");
        /* End of AgentX initialization */


        /* Query */
        olsr_printf(3, "\n\n*** OLSRD_SNMPD: Obtained static variables:\n");
        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
        olsr_printf(3,       "***      OLSRD_SNMPD:       MainAddress:       %s       \n",
olsr_ip_to_string(&olsr_cnf->main_addr));
        olsr_printf(3,  "***  OLSRD_SNMPD:  IpVersion:  %d  \n",  olsr_cnf->ip_version  ==
AF_INET ? 4 : 6);
        olsr_printf(3, "*** OLSRD_SNMPD: Pollrate: %0.2f \n", olsr_cnf->pollrate);
        olsr_printf(3, "*** OLSRD_SNMPD: TcRedundancy: %d \n", olsr_cnf->tc_redundancy);
        olsr_printf(3, "*** OLSRD_SNMPD: MprCoverage: %d \n", olsr_cnf->mpr_coverage);
        olsr_printf(3, "*** OLSRD_SNMPD: TosValue: 0x%04x \n", olsr_cnf->tos);
        olsr_printf(3, "*** OLSRD_SNMPD: Willingness: %d %s \n", olsr_cnf->willingness,
olsr_cnf->willingness_auto ? "(auto)" : "");
        olsr_printf(3, "*** OLSRD_SNMPD: UseHysteresis: %s \n", olsr_cnf->use_hysteresis
? "Enabled" : "Disabled");
        olsr_printf(3,    "***    OLSRD_SNMPD:    HystScaling:    %0.2f    \n",    olsr_cnf-
>hysteresis_param.scaling);
        olsr_printf(3,    "***    OLSRD_SNMPD:    HystThrLow:    %0.2f    \n",    olsr_cnf-
>hysteresis_param.thr_low);
        olsr_printf(3,    "***    OLSRD_SNMPD:    HystThrHigh:    %0.2f    \n",    olsr_cnf-
>hysteresis_param.thr_high);
        olsr_printf(3, "*** OLSRD_SNMPD: LinkQualityLevel: %d \n", olsr_cnf->lq_level);
        olsr_printf(3, "*** OLSRD_SNMPD: LinkQualityWinSize: %d \n", olsr_cnf->lq_wsize);
        olsr_printf(3, "*** OLSRD_SNMPD: LinkQualityFishEye: %d \n", olsr_cnf->lq_fish);
        olsr_printf(3, "*** OLSRD_SNMPD: LinkQualityDijkstraLimitLimit: %d \n", olsr_cnf-
>lq_dlimit);
        olsr_printf(3,  "***  OLSRD_SNMPD:  LinkQualityDijkstraLimitInterval:  %0.2f  \n",
olsr_cnf->lq_dinter);
        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
        olsr_printf(3, "*** OLSRD_SNMPD: Interfaces: \n" );
        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
        for(ifs = olsr_cnf->interfaces; ifs; ifs = ifs->next)
        {
                struct interface *rifs = ifs->interf;
                olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceName: %s \n", ifs->name);
                if(!rifs)
                {
                        olsr_printf(3, "*** OLSRD_SNMPD: Status: DOWN\n");
                        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
```

```
                        continue;
                }
                if(olsr_cnf->ip_version == AF_INET)
                {
                        olsr_printf(3,   "***   OLSRD_SNMPD:   OlsrInterfaceIP:   %s   \n",
sockaddr_to_string(&rifs->int_addr));
                        olsr_printf(3,   "***   OLSRD_SNMPD:   OlsrInterfaceMask:   %s   \n",
sockaddr_to_string(&rifs->int_netmask));
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceBroadcast: %s \n",
sockaddr_to_string(&rifs->int_broadaddr));
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceMTU: %d \n", rifs-
>int_mtu);
                        olsr_printf(3,  "***  OLSRD_SNMPD:  OlsrInterfaceWireless:  %s  \n",
rifs->is_wireless ? "Yes" : "No");
                        olsr_printf(3,  "***  OLSRD_SNMPD:  OlsrInterfaceHelloEmission:  %f
\n", ifs->cnf->hello_params.emission_interval);
                        olsr_printf(3,  "***  OLSRD_SNMPD:  OlsrInterfaceHelloValidity:  %f
\n", ifs->cnf->hello_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceTCEmission: %f \n",
ifs->cnf->tc_params.emission_interval);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceTCValidity: %f \n",
ifs->cnf->tc_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceMIDEmission: %f \n",
ifs->cnf->mid_params.emission_interval);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceMIDValidity: %f \n",
ifs->cnf->mid_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceHNAEmission: %f \n",
ifs->cnf->hna_params.emission_interval);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceHNAValidity: %f \n",
ifs->cnf->hna_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: STATUS: UP \n");
                        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
                }
                else
                {
                        /* Interface is IPv6 */
                        olsr_printf(3,   "***   OLSRD_SNMPD:   OlsrInterfaceIP:   %s   \n",
olsr_ip_to_string((union olsr_ip_addr *)&rifs->int6_addr.sin6_addr));
                        olsr_printf(3,   "***   OLSRD_SNMPD:   OlsrInterfaceMCAST:   %s\n",
olsr_ip_to_string((union olsr_ip_addr *)&rifs->int6_multaddr.sin6_addr));
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceMTU: %d \n", rifs-
>int_mtu);
                        olsr_printf(3,  "***  OLSRD_SNMPD:  OlsrInterfaceWireless:  %s  \n",
rifs->is_wireless ? "Yes" : "No");
                        olsr_printf(3,  "***  OLSRD_SNMPD:  OlsrInterfaceHelloEmission:  %f
\n", ifs->cnf->hello_params.emission_interval);
                        olsr_printf(3,  "***  OLSRD_SNMPD:  OlsrInterfaceHelloValidity:  %f
\n", ifs->cnf->hello_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceTCEmission: %f \n",
ifs->cnf->tc_params.emission_interval);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceTCValidity: %f \n",
ifs->cnf->tc_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceMIDEmission: %f \n",
ifs->cnf->mid_params.emission_interval);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceMIDValidity: %f \n",
ifs->cnf->mid_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceHNAEmission: %f \n",
ifs->cnf->hna_params.emission_interval);
                        olsr_printf(3, "*** OLSRD_SNMPD: OlsrInterfaceHNAValidity: %f \n",
ifs->cnf->hna_params.validity_time);
                        olsr_printf(3, "*** OLSRD_SNMPD: STATUS: UP \n");
                        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
                }
        }

        if((olsr_cnf->ip_version == AF_INET) && (olsr_cnf->hna4_entries))
        {
                struct hna4_entry *hna4;
                olsr_printf(3,  "*** OLSRD_SNMPD: Announced HNAs \n");
                for(hna4 = olsr_cnf->hna4_entries; hna4; hna4 = hna4->next)
                {
```

```
                olsr_printf(3,  "***  OLSRD_SNMPD:  HNAAnnouncedNet:%s  HNAAnnouncedMask:%s
\n",
                            olsr_ip_to_string((union olsr_ip_addr *)&hna4->net),
                            olsr_ip_to_string((union olsr_ip_addr *)&hna4->netmask));
                }
        }
        else if((olsr_cnf->ip_version == AF_INET6) && (olsr_cnf->hna6_entries))
        {
                struct hna6_entry *hna6;
                olsr_printf(3, "*** OLSRD_SNMPD: Announced HNAs: \n");
                for(hna6 = olsr_cnf->hna6_entries; hna6; hna6 = hna6->next)
                {
                        olsr_printf(3,      "***      OLSRD_SNMPD:      HNAAnnouncedNet:%s
HNAAnnouncedPrefix:%d \n",
                            olsr_ip_to_string((union olsr_ip_addr *)&hna6->net),
                            hna6->prefix_len);
                }
        }
        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
        olsr_printf(3, "*** OLSRD_SNMPD: \n" );
        olsr_printf(3, "*** OLSRD_SNMPD: End of obtained static variables\n\n");
        /* End of query */



        /* Registering function that will poll dynamic variables #DAEMON# */
        olsr_register_scheduler_event(&olsrd_snmpd, NULL, plugin_poll, 0, NULL);
        /* Registering AgentX Handler*/
        olsr_register_timeout_function(&agentx);

        return 1;
}


/*****************************************************************************
 *       Optional private constructor and destructor functions               *
 *****************************************************************************/

/* attention: make static to avoid name clashes */

static void __attribute__ ((constructor))
my_init(void);

static void __attribute__ ((destructor))
my_fini(void);


/**
 * Optional Private Constructor
 */
static void
my_init(void)
{
        printf("*** OLSRD_SNMPD: constructor\n");
}


/**
 * Optional Private Destructor
 */
static void
my_fini(void)
{
        printf("*** OLSRD_SNMPD: destructor\n");
        snmp_shutdown("olsrd_snmpd_agentx");        // Shutting down the agent
}


/**
 * Daemon
 */
void
```

```
olsrd_snmpd(void *foo __attribute__((unused)))
{
	/* DAEMON variables */
	int index;
	float etx;
	struct rt_entry *routes;
	struct neighbor_entry *neigh;
	struct neighbor_2_list_entry *list_2;
	struct link_entry *link = NULL;
	struct tc_entry *entry;
	struct topo_dst *dst_entry;

	if (olsr_cnf->debug_level > 2) {
		clear_console();
	}
	olsr_printf(3, "\n*** OLSRD_SNMPD: DAEMON running\n");
	olsr_printf(3, "\n");

	/* OLSR routes */
	olsr_printf(3, "*** OLSRD_SNMPD: OLSR routes:\n");
	for(index = 0;index < HASHSIZE;index++)
	{
		for(routes          =          routingtable[index].next;routes          !=
&routingtable[index];routes = routes->next)
		{
			olsr_printf(3,     "OlsrRouteDestination:%s     OlsrRouteGateway:%s
OlsrRouteMetric:%d OlsrRouteETX:%.2f Type:HOST OlsrRouteInterface:%s\n",
				olsr_ip_to_string(&routes->rt_dst),
				olsr_ip_to_string(&routes->rt_router),
				routes->rt_metric,
				routes->rt_etx,
				routes->rt_if->int_name);
		}
	}

	for(index = 0;index < HASHSIZE;index++)
	{
		for(routes = hna_routes[index].next;routes != &hna_routes[index];routes =
routes->next)
		{
			olsr_printf(3,     "OlsrRouteDestination:%s     OlsrRouteGateway:%s
OlsrRouteMetric:%d OlsrRouteETX:- Type:HNA OlsrRouteInterface:%s\n",
				olsr_ip_to_string(&routes->rt_dst),
				olsr_ip_to_string(&routes->rt_router),
				routes->rt_metric,
				routes->rt_if->int_name);
		}
	}
	olsr_printf(3, "\n");

	/* Neighborhood */
	olsr_printf(3, "*** OLSRD_SNMPD: Neighborhood:\n");
	for(index=0;index<HASHSIZE;index++)
	{
		for(neigh          =          neighbortable[index].next;neigh          !=
&neighbortable[index];neigh = neigh->next)
		{
			olsr_printf(3,        "OlsrNeighborIP:%s        OlsrNeighborSYM:%s
OlsrNeighborMPR:%s OlsrNeighborMPRS:%s OlsrNeighborWillingness:%d\n",
				olsr_ip_to_string(&neigh->neighbor_main_addr),
				(neigh->status == SYM) ? "YES" : "NO",
				neigh->is_mpr ? "YES" : "NO",
				olsr_lookup_mprs_set(&neigh->neighbor_main_addr) ? "YES" :
"NO",
				neigh->willingness);
			/* 2 Hop Neighborhood */
			for(list_2   =   neigh->neighbor_2_list.next;list_2   !=   &neigh-
>neighbor_2_list;list_2 = list_2->next)
			{
				olsr_printf(3, "OlsrNeighbor2HopNeighbors:%s - via *%s*\n",
					olsr_ip_to_string(&list_2->neighbor_2-
>neighbor_2_addr),
```

```
                                        olsr_ip_to_string(&neigh->neighbor_main_addr));
                        }
                        olsr_printf(3, "\n");
                }
        }
        olsr_printf(3, "\n");

        /* Links */
        olsr_printf(3, "*** OLSRD_SNMPD: Links:\n");
        link = link_set;
        while(link)
        {
                olsr_printf(3,          "OlsrLinkLocalIP:%s          OlsrLinkRemoteIP:%s
OlsrLinkHysteresis:%0.2f        OlsrLinkLinkQuality:%0.2f        OlsrLinkLostPackets:%d
OlsrLinkTotalPackets:%d OlsrLinkNLQ:%0.2f OlsrLinkETX:%0.2f\n",
                        olsr_ip_to_string(&link->local_iface_addr),
                        olsr_ip_to_string(&link->neighbor_iface_addr),
                        link->L_link_quality,
                        link->loss_link_quality,
                        link->lost_packets,
                        link->total_packets,
                        link->neigh_link_quality,
                        (link->loss_link_quality  *  link->neigh_link_quality)  ?  1.0  /
(link->loss_link_quality * link->neigh_link_quality) : 0.0);

                link = link->next;
        }
        olsr_printf(3, "\n");


        /* Topology */
        olsr_printf(3, "*** OLSRD_SNMPD: Topology:\n");
        for(index=0;index<HASHSIZE;index++)
        {
                for(entry = tc_table[index].next;entry != &tc_table[index];entry = entry-
>next)
                {
                        for(dst_entry  =  entry->destinations.next;dst_entry  !=  &entry-
>destinations;dst_entry = dst_entry->next)
                        {
                        if     (dst_entry->link_quality     <     0.01     ||     dst_entry-
>inverse_link_quality < 0.01)
                                etx = 0.0;
                        else
                                etx  =  1.0  /  (dst_entry->link_quality  *  dst_entry-
>inverse_link_quality);

                                olsr_printf(3,                      "OlsrTopologySourceIP:%s
OlsrTopologyDestinationIP:%s         OlsrTopologyLQ:%0.2f          OlsrTopologyILQ:%0.2f
OlsrTopologyETX:%0.2f\n",
                                        olsr_ip_to_string(&entry->T_last_addr),
                                        olsr_ip_to_string(&dst_entry->T_dest_addr),
                                        dst_entry->link_quality,
                                        dst_entry->inverse_link_quality,
                                        etx);
                        }
                }
        }
        olsr_printf(3, "\n");

        olsrd_snmpd_dynamic_tables();        // Refreshing the SNMP Tables
        return;
}
/**
 * AgentX
 */
void
agentx(void)
{
        agent_check_and_process(0);
        return;
}
```

C – OLSRD_MIB.C

```
/*
 * OLSR SNMP Agent Plugin.
 * Copyright (c) 2006, 2007, Vinicius Maia Pacheco <vinicius.pacheco@gmail.com>
 *                           Ricardo Puttini <puttini@unb.br>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 * * Neither the name of the OLSR SNMP Agent Plugin nor the names of its contributors
 *   may be used to endorse or promote products derived from this software
 *   without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */


/*
 * File:        olsrd_mib.c
 * Description: OLSR-MIB implementation part of the OLSR SNMP Agent Plugin
 * Created:     04 October 2006
 */


#include "olsrd_mib.h"

void
init_olsrd_mib(void)
{
       // Global Variables
       static oid     MainAddress_oid[]                          = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 1  };
       static oid     IpVersion_oid[]                            = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 2  };
       static oid     Pollrate_oid[]                             = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 3  };
       static oid     TcRedundancy_oid[]                         = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 4  };
       static oid     MprCoverage_oid[]                          = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 5  };
       static oid     TosValue_oid[]                             = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 6  };
       static oid     Willingness_oid[]                          = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 7  };
       static oid     UseHysteresis_oid[]                        = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 8  };
       static oid     HystScaling_oid[]                          = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 9  };
       static oid     HystThrLow_oid[]                           = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 10 };
       static oid     HystThrHigh_oid[]                          = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 11 };
```

```
       static oid     LinkQualityLevel_oid[]                      = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 12 };
       static oid     LinkQualityWinSize_oid[]                    = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 13 };
       static oid     LinkQualityFishEye_oid[]                    = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 14 };
       static oid     LinkQualityDijkstraLimitLimit_oid[]         = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 15 };
       static oid     LinkQualityDijkstraLimitInterval_oid[]      = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 16 };

       // Tables
       static oid     OlsrInterfaceTable_oid[]                    = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 17 };
       static oid     OlsrHNAAnnouncedTable_oid[]                 = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 18 };
       static oid     OlsrRouteTable_oid[]                        = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 19 };
       static oid     OlsrNeighborTable_oid[]                     = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 20 };
       static oid     OlsrLinkTable_oid[]                         = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 21 };
       static oid     OlsrTopologyTable_oid[]                     = { 1, 3, 6, 1, 4, 1,
9363, 1, 1, 1, 22 };

       // Auxiliary Variables
       netsnmp_table_data_set*table_set;
       netsnmp_table_row     *row;
       struct olsr_if        *ifs;
       struct in_addr        in;
       static int            ip_version;
       int                   aux;

       ip_version = olsr_cnf->ip_version == AF_INET ? 4 : 6;

       // Global variables
     netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
             "MainAddress",
             NULL, // access handler funtion
             MainAddress_oid,
             OID_LENGTH(MainAddress_oid),
             HANDLER_CAN_RONLY),
                                netsnmp_create_watcher_info(
             &olsr_cnf->main_addr, sizeof(olsr_cnf->main_addr),
             ASN_IPADDRESS, WATCHER_FIXED_SIZE)
     );

     netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
              "IpVersion",
              NULL, // access handler funtion
              IpVersion_oid,
              OID_LENGTH(IpVersion_oid),
              HANDLER_CAN_RONLY),
                                netsnmp_create_watcher_info(
              &ip_version, sizeof(ip_version),
              ASN_INTEGER, WATCHER_FIXED_SIZE)
     );

     netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
              "Pollrate",
              NULL, // access handler funtion
              Pollrate_oid,
              OID_LENGTH(Pollrate_oid),
              HANDLER_CAN_RONLY),
                                netsnmp_create_watcher_info(
              &olsr_cnf->pollrate, sizeof(olsr_cnf->pollrate),
              ASN_OPAQUE_FLOAT, WATCHER_FIXED_SIZE)
     );

     netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
              "TcRedundancy",
              NULL, // access handler funtion
```

```
                TcRedundancy_oid,
                OID_LENGTH(TcRedundancy_oid),
                HANDLER_CAN_RWRITE),
                                        netsnmp_create_watcher_info(
                &olsr_cnf->tc_redundancy, sizeof(olsr_cnf->tc_redundancy),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
        );

        netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "MprCoverage",
                NULL, // access handler funtion
                MprCoverage_oid,
                OID_LENGTH(MprCoverage_oid),
                HANDLER_CAN_RWRITE),
                                        netsnmp_create_watcher_info(
                &olsr_cnf->mpr_coverage, sizeof(olsr_cnf->mpr_coverage),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
        );

        netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "TosValue",
                NULL, // access handler funtion
                TosValue_oid,
                OID_LENGTH(TosValue_oid),
                HANDLER_CAN_RONLY),
                                        netsnmp_create_watcher_info(
                &olsr_cnf->tos, sizeof(olsr_cnf->tos),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
        );

        netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "Willingness",
                NULL, // access handler funtion
                Willingness_oid,
                OID_LENGTH(Willingness_oid),
                HANDLER_CAN_RWRITE),
                                        netsnmp_create_watcher_info(
                &olsr_cnf->willingness, sizeof(olsr_cnf->willingness),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
        );

        netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "UseHysteresis",
                NULL, // access handler funtion
                UseHysteresis_oid,
                OID_LENGTH(UseHysteresis_oid),
                HANDLER_CAN_RWRITE),
                                        netsnmp_create_watcher_info(
                &olsr_cnf->use_hysteresis, sizeof(olsr_cnf->use_hysteresis),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
        );

        netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "HystScaling",
                NULL, // access handler funtion
                HystScaling_oid,
                OID_LENGTH(HystScaling_oid),
                HANDLER_CAN_RWRITE),
                                        netsnmp_create_watcher_info(
                &olsr_cnf->hysteresis_param.scaling,                    sizeof(olsr_cnf-
>hysteresis_param.scaling),
                ASN_OPAQUE_FLOAT, WATCHER_FIXED_SIZE)
        );

        netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "HystThrLow",
                NULL, // access handler funtion
                HystThrLow_oid,
                OID_LENGTH(HystThrLow_oid),
                HANDLER_CAN_RWRITE),
                                        netsnmp_create_watcher_info(
```

```
                &olsr_cnf->hysteresis_param.thr_low,                    sizeof(olsr_cnf-
>hysteresis_param.thr_low),
                ASN_OPAQUE_FLOAT, WATCHER_FIXED_SIZE)
      );

      netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "HystThrHigh",
                NULL, // access handler funtion
                HystThrHigh_oid,
                OID_LENGTH(HystThrHigh_oid),
                HANDLER_CAN_RWRITE),
                                    netsnmp_create_watcher_info(
                &olsr_cnf->hysteresis_param.thr_high,                   sizeof(olsr_cnf-
>hysteresis_param.thr_high),
                ASN_OPAQUE_FLOAT, WATCHER_FIXED_SIZE)
      );

      netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "LinkQualityLevel",
                NULL, // access handler funtion
                LinkQualityLevel_oid,
                OID_LENGTH(LinkQualityLevel_oid),
                HANDLER_CAN_RWRITE),
                                    netsnmp_create_watcher_info(
                &olsr_cnf->lq_level, sizeof(olsr_cnf->lq_level),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
      );

      netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "LinkQualityWinSize",
                NULL, // access handler funtion
                LinkQualityWinSize_oid,
                OID_LENGTH(LinkQualityWinSize_oid),
                HANDLER_CAN_RWRITE),
                                    netsnmp_create_watcher_info(
                &olsr_cnf->lq_wsize, sizeof(olsr_cnf->lq_wsize),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
      );

      netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "LinkQualityFishEye",
                NULL, // access handler funtion
                LinkQualityFishEye_oid,
                OID_LENGTH(LinkQualityFishEye_oid),
                HANDLER_CAN_RWRITE),
                                    netsnmp_create_watcher_info(
                &olsr_cnf->lq_fish, sizeof(olsr_cnf->lq_fish),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
      );

      netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "LinkQualityDijkstraLimitLimit",
                NULL, // access handler funtion
                LinkQualityDijkstraLimitLimit_oid,
                OID_LENGTH(LinkQualityDijkstraLimitLimit_oid),
                HANDLER_CAN_RWRITE),
                                    netsnmp_create_watcher_info(
                &olsr_cnf->lq_dlimit, sizeof(olsr_cnf->lq_dlimit),
                ASN_OCTET_STR, WATCHER_FIXED_SIZE)
      );

      netsnmp_register_watched_scalar(  netsnmp_create_handler_registration(
                "LinkQualityDijkstraLimitInterval",
                NULL, // access handler funtion
                LinkQualityDijkstraLimitInterval_oid,
                OID_LENGTH(LinkQualityDijkstraLimitInterval_oid),
                HANDLER_CAN_RWRITE),
                                    netsnmp_create_watcher_info(
                &olsr_cnf->lq_dinter, sizeof(olsr_cnf->lq_dinter),
                ASN_OPAQUE_FLOAT, WATCHER_FIXED_SIZE)
      );
// End of global variables
```

```
// Tables Variables
        // OlsrInterfaceTable
        table_set = netsnmp_create_table_data_set("OlsrInterfaceTable");
        table_set->allow_creation = 0;
        netsnmp_table_dataset_add_index(table_set, ASN_INTEGER);
        netsnmp_table_set_multi_add_default_row(table_set,
                /*
                 * column 2 = IPADDRESS,
                 * writable = 0,
                 * default value = NULL,
                 * default value len = 0
                 */
                1,ASN_INTEGER,0,NULL,0,              // OlsrInterfaceTableIndex
                2,ASN_OCTET_STR,0,NULL,0,            // OlsrInterfaceName
                3,ASN_IPADDRESS,0,NULL,0,            // OlsrInterfaceIP
                4,ASN_IPADDRESS,0,NULL,0,            // OlsrInterfaceMask
                5,ASN_IPADDRESS,0,NULL,0,            // OlsrInterfaceBroadcast
                6,ASN_INTEGER,0,NULL,0,              // OlsrInterfaceMTU
                7,ASN_INTEGER,0,NULL,0,              // OlsrInterfaceWireless
                8,ASN_OPAQUE_FLOAT,0,NULL,0,         // OlsrInterfaceHelloEmission
                9,ASN_OPAQUE_FLOAT,0,NULL,0,         // OlsrInterfaceHelloValidity
                10,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrInterfaceTCEmission
                11,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrInterfaceTCValidity
                12,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrInterfaceMIDEmission
                13,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrInterfaceMIDValidity
                14,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrInterfaceHNAEmission
                15,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrInterfaceHNAValidity

                0 /* done */ );

        netsnmp_register_table_data_set( netsnmp_create_handler_registration(
                        "OlsrInterfaceTable",
                        NULL,
                        OlsrInterfaceTable_oid,
                        OID_LENGTH(OlsrInterfaceTable_oid),
                        HANDLER_CAN_RONLY),
                                        table_set,
                                        NULL
        );

        // Now, create the rows
        aux = 1;
        for(ifs = olsr_cnf->interfaces; ifs; ifs = ifs->next) {
                struct interface *rifs = ifs->interf;
                if(!rifs) {
                        continue;                                               //    Interface
is DOWN
                }
                if(olsr_cnf->ip_version == AF_INET) {                            //
Interface is UP and is IPv4
                        row = netsnmp_create_table_data_row();
                        netsnmp_table_row_add_index(row, ASN_INTEGER, &aux, sizeof(aux));
        // OlsrInterfaceTableIndex

                        netsnmp_set_row_column(row,   1,   ASN_INTEGER,   (char   *)   &aux,
sizeof(aux));
                        netsnmp_mark_row_column_writable(row, 1, 0);
                        netsnmp_set_row_column(row,    2,    ASN_OCTET_STR,    ifs->name,
strlen(ifs->name));
                        netsnmp_mark_row_column_writable(row, 2, 0);              // make
writable or not via SETs
                        inet_aton( sockaddr_to_string(&rifs->int_addr), &in);
                        netsnmp_set_row_column(row,   3,   ASN_IPADDRESS,   (char   *)   &in,
sizeof(in));
                        netsnmp_mark_row_column_writable(row, 3, 0);        //         make
writable or not via SETs
                        inet_aton( sockaddr_to_string(&rifs->int_netmask), &in);
                        netsnmp_set_row_column(row,   4,   ASN_IPADDRESS,   (char   *)   &in,
sizeof(in));
```

```
                        netsnmp_mark_row_column_writable(row, 4, 0);        //         make
writable or not via SETs
                        inet_aton( sockaddr_to_string(&rifs->int_broadaddr), &in);
                        netsnmp_set_row_column(row,   5,   ASN_IPADDRESS,   (char   *)   &in,
sizeof(in));
                        netsnmp_mark_row_column_writable(row, 5, 0);        //         make
writable or not via SETs
                        netsnmp_set_row_column(row,   6,   ASN_INTEGER,   (char   *)   &rifs-
>int_mtu, sizeof(rifs->int_mtu));
                        netsnmp_mark_row_column_writable(row, 6, 0);        //         make
writable or not via SETs
                        netsnmp_set_row_column(row,   7,   ASN_INTEGER,   (char   *)   &rifs-
>is_wireless, sizeof(rifs->is_wireless));
                        netsnmp_mark_row_column_writable(row, 7, 0);        //         make
writable or not via SETs
                        netsnmp_set_row_column(row,  8,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->hello_params.emission_interval, sizeof(ifs->cnf->hello_params.emission_interval));
                        netsnmp_mark_row_column_writable(row, 8, 0);        //         make
writable or not via SETs
                        netsnmp_set_row_column(row,  9,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->hello_params.validity_time, sizeof(ifs->cnf->hello_params.validity_time));
                        netsnmp_mark_row_column_writable(row, 9, 0);        //         make
writable or not via SETs
                        netsnmp_set_row_column(row,  10,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->tc_params.emission_interval, sizeof(ifs->cnf->tc_params.emission_interval));
                        netsnmp_mark_row_column_writable(row, 10, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  11,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->tc_params.validity_time, sizeof(ifs->cnf->tc_params.validity_time));
                        netsnmp_mark_row_column_writable(row, 11, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  12,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->mid_params.emission_interval, sizeof(ifs->cnf->mid_params.emission_interval));
                        netsnmp_mark_row_column_writable(row, 12, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  13,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->mid_params.validity_time, sizeof(ifs->cnf->mid_params.validity_time));
                        netsnmp_mark_row_column_writable(row, 13, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  14,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->hna_params.emission_interval, sizeof(ifs->cnf->hna_params.emission_interval));
                        netsnmp_mark_row_column_writable(row, 14, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  15,  ASN_OPAQUE_FLOAT,  (char  *)  &ifs-
>cnf->hna_params.validity_time, sizeof(ifs->cnf->hna_params.validity_time));
                        netsnmp_mark_row_column_writable(row, 15, 0);              // make
writable or not via SETs

                        netsnmp_table_dataset_add_row(table_set, row);             //
Adding the row
                        aux++;
                }
        }
        // End of OlsrInterfaceTable

        // OlsrHNAAnnouncedTable
        table_set = netsnmp_create_table_data_set("OlsrHNAAnnouncedTable");
        table_set->allow_creation = 1;
        netsnmp_table_dataset_add_index(table_set, ASN_INTEGER);
        netsnmp_table_set_multi_add_default_row(table_set,
                /*
                 * column 2 = IPADDRESS,
                 * writable = 0,
                 * default value = NULL,
                 * default value len = 0
                 */
                1,ASN_INTEGER,1,NULL,0,       // OlsrHNAAnnouncedTableIndex
                2,ASN_IPADDRESS,1,NULL,0,     // OlsrHNAAnnouncedNet
                3,ASN_IPADDRESS,1,NULL,0,     // OlsrHNAAnnouncedMask

                0 /* done */ );
```

```
	netsnmp_register_table_data_set( netsnmp_create_handler_registration(
			"OlsrHNAAnnouncedTable",
			NULL,
			OlsrHNAAnnouncedTable_oid,
			OID_LENGTH(OlsrHNAAnnouncedTable_oid),
			HANDLER_CAN_RWRITE),
				table_set,
				NULL
	);
	// Now, the rows
	aux = 1;
	if((olsr_cnf->ip_version == AF_INET) && (olsr_cnf->hna4_entries)) {
		struct hna4_entry *hna4;
		for(hna4 = olsr_cnf->hna4_entries; hna4; hna4 = hna4->next) {
			row = netsnmp_create_table_data_row();
			netsnmp_table_row_add_index(row, ASN_INTEGER, &aux, sizeof(aux));
// OlsrHNAAnnouncedTableIndex

			netsnmp_set_row_column(row, 1, ASN_INTEGER, (char *) &aux,
sizeof(aux));
			netsnmp_mark_row_column_writable(row, 1, 1);
			netsnmp_set_row_column(row, 2, ASN_IPADDRESS, (char *) &hna4->net,
sizeof(hna4->net));
			netsnmp_mark_row_column_writable(row, 2, 1);              // make
writable or not via SETs
			netsnmp_set_row_column(row, 3, ASN_IPADDRESS, (char *) &hna4-
>netmask, sizeof(hna4->netmask));
			netsnmp_mark_row_column_writable(row, 3, 1);              // make
writable or not via SETs

			netsnmp_table_dataset_add_row(table_set, row);           // Adding
the row
			aux++;
		}
	}
	table_set_OlsrHNAAnnouncedTable = table_set;// Exporting OlsrHNAAnnounced Table
	// End of OlsrHNAAnnouncedTable

	// OlsrRouteTable
	table_set = netsnmp_create_table_data_set("OlsrRouteTable");
	table_set->allow_creation = 0;
	netsnmp_table_dataset_add_index(table_set, ASN_INTEGER);
	netsnmp_table_set_multi_add_default_row(table_set,
		/*
		 * column 2 = IPADDRESS,
		 * writable = 0,
		 * default value = NULL,
		 * default value len = 0
		 */
		1,ASN_INTEGER,0,NULL,0,             // OlsrRouteTableIndex
		2,ASN_IPADDRESS,0,NULL,0,           // OlsrRouteDestination
		3,ASN_IPADDRESS,0,NULL,0,           // OlsrRouteGateway
		4,ASN_INTEGER,0,NULL,0,             // OlsrRouteMetric
		5,ASN_OPAQUE_FLOAT,0,NULL,0,        // OlsrRouteETX
		6,ASN_OCTET_STR,0,NULL,0,           // OlsrRouteType
		7,ASN_OCTET_STR,0,NULL,0,           // OlsrRouteInterface

		0 /* done */ );

	netsnmp_register_table_data_set( netsnmp_create_handler_registration(
			"OlsrRouteTable",
			NULL,
			OlsrRouteTable_oid,
			OID_LENGTH(OlsrRouteTable_oid),
			HANDLER_CAN_RONLY),
				table_set,
				NULL
	);
	table_set_OlsrRouteTable = table_set;       // Exporting OlsrRoute table
	// End of OlsrRouteTable

	// OlsrNeighborTable
```

```
table_set = netsnmp_create_table_data_set("OlsrNeighborTable");
 table_set->allow_creation = 0;
 netsnmp_table_dataset_add_index(table_set, ASN_INTEGER);
 netsnmp_table_set_multi_add_default_row(table_set,
         /*
          * column 2 = IPADDRESS,
          * writable = 0,
          * default value = NULL,
          * default value len = 0
          */
         1,ASN_INTEGER,0,NULL,0,         // OlsrNeighborTableIndex
         2,ASN_IPADDRESS,0,NULL,0,       // OlsrNeighborIP
         3,ASN_OCTET_STR,0,NULL,0,       // OlsrNeighborSYM
         4,ASN_OCTET_STR,0,NULL,0,       // OlsrNeighborMPR
         5,ASN_OCTET_STR,0,NULL,0,       // OlsrNeighborMPRS
         6,ASN_OCTET_STR,0,NULL,0,       // OlsrNeighborWillingness
         7,ASN_OCTET_STR,0,NULL,0,       // OlsrNeighbor2HopNeighbors

         0 /* done */ );

 netsnmp_register_table_data_set( netsnmp_create_handler_registration(
                 "OlsrNeighborTable",
                 NULL,
                 OlsrNeighborTable_oid,
                 OID_LENGTH(OlsrNeighborTable_oid),
                 HANDLER_CAN_RONLY),
                         table_set,
                         NULL
 );
 table_set_OlsrNeighborTable = table_set;   // Exporting OlsrNeighbor table
// End of OlsrNeighborTable

// OlsrLinkTable
table_set = netsnmp_create_table_data_set("OlsrLinkTable");
table_set->allow_creation = 0;
netsnmp_table_dataset_add_index(table_set, ASN_INTEGER);
netsnmp_table_set_multi_add_default_row(table_set,
         /*
          * column 2 = IPADDRESS,
          * writable = 0,
          * default value = NULL,
          * default value len = 0
          */
        1,ASN_INTEGER,0,NULL,0,         // OlsrLinkTableIndex
        2,ASN_IPADDRESS,0,NULL,0,       // OlsrLinkLocalIP
        3,ASN_IPADDRESS,0,NULL,0,       // OlsrLinkRemoteIP
        4,ASN_OPAQUE_FLOAT,0,NULL,0,    // OlsrLinkHysteresis
        5,ASN_OPAQUE_FLOAT,0,NULL,0,    // OlsrLinkLinkQuality
        6,ASN_INTEGER,0,NULL,0,         // OlsrLinkLostPackets
        7,ASN_INTEGER,0,NULL,0,         // OlsrLinkTotalPackets
        8,ASN_OPAQUE_FLOAT,0,NULL,0,    // OlsrLinkNLQ
        9,ASN_OPAQUE_FLOAT,0,NULL,0,    // OlsrLinkETX

        0 /* done */ );

netsnmp_register_table_data_set( netsnmp_create_handler_registration(
                "OlsrLinkTable",
                NULL,
                OlsrLinkTable_oid,
                OID_LENGTH(OlsrLinkTable_oid),
                HANDLER_CAN_RONLY),
                        table_set,
                        NULL
);
table_set_OlsrLinkTable = table_set;   // Exporting OlsrLink table
// End of OlsrLinkTable


// OlsrTopologyTable
table_set = netsnmp_create_table_data_set("OlsrTopologyTable");
table_set->allow_creation = 0;
netsnmp_table_dataset_add_index(table_set, ASN_INTEGER);
```

```
        netsnmp_table_set_multi_add_default_row(table_set,
                /*
                 * column 2 = IPADDRESS,
                 * writable = 0,
                 * default value = NULL,
                 * default value len = 0
                 */
                1,ASN_INTEGER,0,NULL,0,              // OlsrTopologyTableIndex
                2,ASN_IPADDRESS,0,NULL,0,            // OlsrTopologySourceIP
                3,ASN_IPADDRESS,0,NULL,0,            // OlsrTopologyDestinationIP
                4,ASN_OPAQUE_FLOAT,0,NULL,0,         // OlsrTopologyLQ
                5,ASN_OPAQUE_FLOAT,0,NULL,0,         // OlsrTopologyNLQ
                6,ASN_OPAQUE_FLOAT,0,NULL,0,         // OlsrTopologyETX

                0 /* done */ );

        netsnmp_register_table_data_set( netsnmp_create_handler_registration(
                        "OlsrTopologyTable",
                        NULL,
                        OlsrTopologyTable_oid,
                        OID_LENGTH(OlsrTopologyTable_oid),
                        HANDLER_CAN_RONLY),
                                table_set,
                                NULL
                );
        table_set_OlsrTopologyTable = table_set;   // Exporting OlsrLink table

        // End of OlsrTopologyTable

// End of Tables Variables
        return;
}

void
olsrd_snmpd_dynamic_tables(void)
{

        // Function Variables
         int index;
        int aux;
        float link_LQ_temp;
        float link_NLQ_temp;
        float link_ETX_temp;
        float topology_LQ_temp;
        float topology_ILQ_temp;
        float etx;
        char hopneigh_temp[200];
        char str_hna_net_temp[200];
        char str_hna_mask_temp[200];
        struct in_addr hna_net_temp;
        struct in_addr hna_mask_temp;
         struct rt_entry *routes;
         struct neighbor_entry *neigh;
         struct neighbor_2_list_entry *list_2;
        struct link_entry *link = NULL;
        struct tc_entry *entry;
        struct topo_dst *dst_entry;
        netsnmp_table_row       *row;
        netsnmp_table_data_set_storage *data;

        // HNAAnnouncedTable
        // Removing HNAs
        if((olsr_cnf->ip_version == AF_INET) && (olsr_cnf->hna4_entries)) {
                struct hna4_entry *hna4;
                for(hna4 = olsr_cnf->hna4_entries; hna4; hna4 = hna4->next) {
                        remove_local_hna4_entry((union  olsr_ip_addr  *)&hna4->net,  (union
olsr_ip_addr *)&hna4->netmask);
                }
        }
        // Now, Adding the correct HNAs coming from the snmpd
```

```
	for	(row	=	netsnmp_table_data_set_get_first_row
(table_set_OlsrHNAAnnouncedTable);	row;	row	=	netsnmp_table_data_set_get_next_row
(table_set_OlsrHNAAnnouncedTable, row)) {
		data = netsnmp_table_data_set_find_column((netsnmp_table_data_set_storage
*)row->data , 2);
		sprintf(str_hna_net_temp,	"%d.%d.%d.%d",	data->data.string[0],	data-
>data.string[1], data->data.string[2], data->data.string[3]);
		inet_aton(str_hna_net_temp, &hna_net_temp);
		data = netsnmp_table_data_set_find_column((netsnmp_table_data_set_storage
*)row->data , 3);
		sprintf(str_hna_mask_temp,	"%d.%d.%d.%d",	data->data.string[0],	data-
>data.string[1], data->data.string[2], data->data.string[3]);
		inet_aton(str_hna_mask_temp, &hna_mask_temp);
		add_local_hna4_entry((union	olsr_ip_addr	*)&hna_net_temp.s_addr,	(union
olsr_ip_addr *)&hna_mask_temp.s_addr);
	}


	// End of HNAAnnouncedTable

	// OlsrRouteTable
	// First, emptying the rows
	while( netsnmp_table_data_set_get_first_row (table_set_OlsrRouteTable) ) {
		netsnmp_table_dataset_remove_and_delete_row	(table_set_OlsrRouteTable,
netsnmp_table_data_set_get_first_row (table_set_OlsrRouteTable));
	}

	// Now, adding new rows
	aux =1;
	for(index = 0;index < HASHSIZE;index++) {
		for(routes	=	routingtable[index].next;routes	!=
&routingtable[index];routes = routes->next) {
			row = netsnmp_create_table_data_row();
			netsnmp_table_row_add_index(row,	ASN_INTEGER,	&aux,	sizeof(aux));
// OlsrRouteTableIndex

			netsnmp_set_row_column(row,	1,	ASN_INTEGER,	(char	*)	&aux,
sizeof(aux));
			netsnmp_mark_row_column_writable(row, 1, 0);
			netsnmp_set_row_column(row,	2,	ASN_IPADDRESS,	(char	*)	&routes-
>rt_dst, sizeof(routes->rt_dst));
			netsnmp_mark_row_column_writable(row, 2, 0);		// make
writable or not via SETs
			netsnmp_set_row_column(row,	3,	ASN_IPADDRESS,	(char	*)	&routes-
>rt_router, sizeof(routes->rt_router));
			netsnmp_mark_row_column_writable(row, 3, 0);		// make
writable or not via SETs
			netsnmp_set_row_column(row,	4,	ASN_INTEGER,	(char	*)	&routes-
>rt_metric, sizeof(routes->rt_metric));
			netsnmp_mark_row_column_writable(row, 4, 0);
			netsnmp_set_row_column(row, 5, ASN_OPAQUE_FLOAT, (char *) &routes-
>rt_etx, sizeof(routes->rt_etx));
			netsnmp_mark_row_column_writable(row, 5, 0);
			netsnmp_set_row_column(row,	6,	ASN_OCTET_STR,	"HOST",
strlen("HOST"));
			netsnmp_mark_row_column_writable(row, 6, 0);
			netsnmp_set_row_column(row,	7,	ASN_OCTET_STR,	routes->rt_if-
>int_name, strlen(routes->rt_if->int_name));
			netsnmp_mark_row_column_writable(row, 7, 0);

			netsnmp_table_dataset_add_row(table_set_OlsrRouteTable,	row);
// Adding the row


			aux++;
		}
	}
	for(index = 0;index < HASHSIZE;index++) {
		for(routes = hna_routes[index].next;routes != &hna_routes[index];routes =
routes->next) {
			row = netsnmp_create_table_data_row();
```

```
                        netsnmp_table_row_add_index(row, ASN_INTEGER, &aux, sizeof(aux));
// OlsrRouteTableIndex

                        netsnmp_set_row_column(row,  1,  ASN_INTEGER,  (char  *)  &aux,
sizeof(aux));
                        netsnmp_mark_row_column_writable(row, 1, 0);
                        netsnmp_set_row_column(row, 2, ASN_IPADDRESS, (char *) &routes-
>rt_dst, sizeof(routes->rt_dst));
                        netsnmp_mark_row_column_writable(row, 2, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row, 3, ASN_IPADDRESS, (char *) &routes-
>rt_router, sizeof(routes->rt_router));
                        netsnmp_mark_row_column_writable(row, 3, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  4,  ASN_INTEGER,  (char  *)  &routes-
>rt_metric, sizeof(routes->rt_metric));
                        netsnmp_mark_row_column_writable(row, 4, 0);
                        netsnmp_set_row_column(row, 5, ASN_OPAQUE_FLOAT, (char *) &routes-
>rt_etx, sizeof(routes->rt_etx));
                        netsnmp_mark_row_column_writable(row, 5, 0);
                        netsnmp_set_row_column(row,      6,      ASN_OCTET_STR,      "HNA",
strlen("HNA"));
                        netsnmp_mark_row_column_writable(row, 6, 0);
                        netsnmp_set_row_column(row,   7,   ASN_OCTET_STR,   routes->rt_if-
>int_name, strlen(routes->rt_if->int_name));
                        netsnmp_mark_row_column_writable(row, 7, 0);

                        netsnmp_table_dataset_add_row(table_set_OlsrRouteTable,      row);
// Adding the row

                        aux++;
                }
        }
        // End of OlsrRouteTable

        // OlsrNeighborTable

        // First, emptying the rows
         while( netsnmp_table_data_set_get_first_row (table_set_OlsrNeighborTable) ) {
                netsnmp_table_dataset_remove_and_delete_row  (table_set_OlsrNeighborTable,
netsnmp_table_data_set_get_first_row (table_set_OlsrNeighborTable));
        }

        // Now, adding new rows
        aux =1;
        for(index=0;index<HASHSIZE;index++) {
                for(neigh          =          neighbortable[index].next;neigh          !=
&neighbortable[index];neigh = neigh->next) {
                        row = netsnmp_create_table_data_row();
                        netsnmp_table_row_add_index(row, ASN_INTEGER, &aux, sizeof(aux));
// OlsrNeighborTableIndex

                        netsnmp_set_row_column(row,  1,  ASN_INTEGER,  (char  *)  &aux,
sizeof(aux));
                        netsnmp_mark_row_column_writable(row, 1, 0);
                        netsnmp_set_row_column(row, 2, ASN_IPADDRESS, (char *) &neigh-
>neighbor_main_addr, sizeof(neigh->neighbor_main_addr));
                        netsnmp_mark_row_column_writable(row, 2, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row, 3, ASN_OCTET_STR, ((neigh->status ==
SYM) ? "YES" : "NO"), strlen(((neigh->status == SYM) ? "YES" : "NO")));
                        netsnmp_mark_row_column_writable(row, 3, 0);              // make
writable or not via SETs
                        netsnmp_set_row_column(row,  4,  ASN_OCTET_STR,  (neigh->is_mpr  ?
"YES" : "NO"), strlen(neigh->is_mpr ? "YES" : "NO"));
                        netsnmp_mark_row_column_writable(row, 4, 0);
                        netsnmp_set_row_column(row,            5,            ASN_OCTET_STR,
(olsr_lookup_mprs_set(&neigh->neighbor_main_addr)      ?      "YES"      :      "NO"),
strlen(olsr_lookup_mprs_set(&neigh->neighbor_main_addr) ? "YES" : "NO"));
                        netsnmp_mark_row_column_writable(row, 5, 0);
                        netsnmp_set_row_column(row, 6, ASN_OCTET_STR, (char *) &neigh-
>willingness, sizeof(neigh->willingness));
```

```
                        netsnmp_mark_row_column_writable(row, 6, 0);


                        /* 2 Hop Neighbors */
                        for(list_2 = neigh->neighbor_2_list.next, strcpy(hopneigh_temp, "-
") ;list_2 != &neigh->neighbor_2_list;list_2 = list_2->next) {
                                strcat(hopneigh_temp      ,      olsr_ip_to_string(&list_2-
>neighbor_2->neighbor_2_addr) );
                                strcat(hopneigh_temp , "-");
                        }
                        netsnmp_set_row_column(row,   7,   ASN_OCTET_STR,   hopneigh_temp,
strlen(hopneigh_temp));
                        netsnmp_mark_row_column_writable(row, 7, 0);

                        netsnmp_table_dataset_add_row(table_set_OlsrNeighborTable,   row);
// Adding the row
                        aux++;
                }
        }
        // End of OlsrNeighborTable

        // OlsrLinkTable

        // First, emptying the rows
        while( netsnmp_table_data_set_get_first_row (table_set_OlsrLinkTable) ) {
                netsnmp_table_dataset_remove_and_delete_row      (table_set_OlsrLinkTable,
netsnmp_table_data_set_get_first_row (table_set_OlsrLinkTable));
        }

         // Now, adding new rows
         aux =1;
        link = link_set;
        while(link) {
                row = netsnmp_create_table_data_row();
                netsnmp_table_row_add_index(row,    ASN_INTEGER,    &aux,    sizeof(aux));
// OlsrLinkTableIndex

                netsnmp_set_row_column(row, 1, ASN_INTEGER, (char *) &aux, sizeof(aux));
                netsnmp_mark_row_column_writable(row, 1, 0);
                netsnmp_set_row_column(row,    2,    ASN_IPADDRESS,    (char    *)    &link-
>local_iface_addr, sizeof(link->local_iface_addr));
                netsnmp_mark_row_column_writable(row, 2, 0);             // make writable
or not via SETs
                netsnmp_set_row_column(row,    3,    ASN_IPADDRESS,    (char    *)    &link-
>neighbor_iface_addr, sizeof(link->neighbor_iface_addr));
                netsnmp_mark_row_column_writable(row, 3, 0);             // make writable
or not via SETs
                netsnmp_set_row_column(row,   4,   ASN_OPAQUE_FLOAT,   (char   *)   &link-
>L_link_quality, sizeof(link->L_link_quality));
                netsnmp_mark_row_column_writable(row, 4, 0);
                link_LQ_temp = link->loss_link_quality;
                netsnmp_set_row_column(row, 5, ASN_OPAQUE_FLOAT, (char *) &link_LQ_temp,
sizeof(link_LQ_temp));
                netsnmp_mark_row_column_writable(row, 5, 0);
                netsnmp_set_row_column(row, 6, ASN_INTEGER, (char *) &link->lost_packets,
sizeof(link->lost_packets));
                netsnmp_mark_row_column_writable(row, 6, 0);
                netsnmp_set_row_column(row,    7,    ASN_INTEGER,    (char    *)    &link-
>total_packets, sizeof(link->total_packets));
                netsnmp_mark_row_column_writable(row, 7, 0);
                link_NLQ_temp = link->neigh_link_quality;
                netsnmp_set_row_column(row, 8, ASN_OPAQUE_FLOAT, (char *) &link_NLQ_temp,
sizeof(link_NLQ_temp));
                netsnmp_mark_row_column_writable(row, 8, 0);
                link_ETX_temp  =  (link->loss_link_quality  *  link->neigh_link_quality)  ?
1.0 / (link->loss_link_quality * link->neigh_link_quality) : 0.0;
                netsnmp_set_row_column(row, 9, ASN_OPAQUE_FLOAT, (char *) &link_ETX_temp,
sizeof(link_ETX_temp));
                netsnmp_mark_row_column_writable(row, 9, 0);

                netsnmp_table_dataset_add_row(table_set_OlsrLinkTable, row);            //
Adding the row
```

```
                link = link->next;
                aux++;
        }
        // End of OlsrLinkTable

        // OlsrTopologyTable

        // First, emptying the rows
        while( netsnmp_table_data_set_get_first_row (table_set_OlsrTopologyTable) ) {
                netsnmp_table_dataset_remove_and_delete_row  (table_set_OlsrTopologyTable,
netsnmp_table_data_set_get_first_row (table_set_OlsrTopologyTable));
        }

        // Now, adding new rows
        aux =1;
        for(index=0;index<HASHSIZE;index++)
        {
                for(entry = tc_table[index].next;entry != &tc_table[index];entry = entry-
>next)
                {
                        for(dst_entry   =  entry->destinations.next;dst_entry  !=  &entry-
>destinations;dst_entry = dst_entry->next)
                        {
                                if   (dst_entry->link_quality   <   0.01   ||   dst_entry-
>inverse_link_quality < 0.01)
                                        etx = 0.0;
                                else
                                        etx = 1.0 / (dst_entry->link_quality * dst_entry-
>inverse_link_quality);

                                topology_LQ_temp = dst_entry->link_quality;
                                topology_ILQ_temp = dst_entry->inverse_link_quality;

                                row = netsnmp_create_table_data_row();
                                netsnmp_table_row_add_index(row,     ASN_INTEGER,     &aux,
sizeof(aux));       // OlsrTopologyTableIndex

                                netsnmp_set_row_column(row, 1, ASN_INTEGER, (char *) &aux,
sizeof(aux));
                                netsnmp_mark_row_column_writable(row, 1, 0);
                                netsnmp_set_row_column(row,  2,  ASN_IPADDRESS,  (char  *)
&entry->T_last_addr, sizeof(entry->T_last_addr));
                                netsnmp_mark_row_column_writable(row, 2, 0);             //
make writable or not via SETs
                                netsnmp_set_row_column(row,  3,  ASN_IPADDRESS,  (char  *)
&dst_entry->T_dest_addr, sizeof(dst_entry->T_dest_addr));
                                netsnmp_mark_row_column_writable(row, 3, 0);             //
make writable or not via SETs
                                netsnmp_set_row_column(row,  4,  ASN_OPAQUE_FLOAT,  (char  *)
&topology_LQ_temp, sizeof(topology_LQ_temp));
                                netsnmp_mark_row_column_writable(row, 4, 0);
                                netsnmp_set_row_column(row,  5,  ASN_OPAQUE_FLOAT,  (char  *)
&topology_ILQ_temp, sizeof(topology_ILQ_temp));
                                netsnmp_mark_row_column_writable(row, 5, 0);
                                netsnmp_set_row_column(row,  6,  ASN_OPAQUE_FLOAT,  (char  *)
&etx, sizeof(etx));
                                netsnmp_mark_row_column_writable(row, 6, 0);
                                netsnmp_table_dataset_add_row(table_set_OlsrTopologyTable,
row);          // Adding the row
                                aux++;
                        }
                }
        }

        // End of OlsrTopologyTable

        return;
}
```