PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Gerenciamento de Roteamento BGP em Pontos de Troca de Tráfego

por

ANDREY VEDANA ANDREOLI
FABIO FALCI RODRIGUES

Relatório de Trabalho de Conclusão II

Prof. MSc. Ana Cristina Benso da Silva
Orientadora

Porto Alegre, dezembro de 2002

SUMÁRIO

Sumário ..... **ii**

Lista de Abreviaturas ..... **vi**

Lista de Figuras ..... **viii**

Lista de Tabelas ..... **xi**

Agradecimentos ..... **xii**

Resumo ..... **xiv**

Abstract ..... **xv**

1. Introdução **17**

1.1 Objetivos gerais e específicos **17**

1.2 Contribuições **18**

1.3 Organização do texto **18**

2. Roteamento **19**

2.1 Algoritmos de roteamento **21**

2.2 Protocolos IGP **22**

2.3 Protocolos EGP **23**

2.4 BGP (*Border Gateway Protocol*) **24**

2.4.1 Introdução 24

2.4.2 Modelo de divulgação e atualização das tabelas de rotas 25

2.4.3 Estados de uma conexão BGP 26

2.4.4 Funcionamento do algoritmo de decisão 28

2.4.5 Utilização de políticas 29

2.4.6 Mensagens do protocolo 31

2.4.6.1 Mensagem tipo OPEN 32

2.4.6.2 Mensagem tipo NOTIFICATION 33

2.4.6.3 Mensagem tipo KEEPALIVE 34

2.4.6.4 Mensagem tipo UPDATE 34

2.4.6.4.1 CIDR 38

2.4.7 Utilização de BGP em sistemas autônomos 39

2.4.8 O Futuro do BGP 40

3. Organização da Internet em PTT´s **41**

3.1.1 Conceitos Gerais 42

3.1.2 Arquiteturas de PTTs 42

3.1.3 Ferramentas Utilizadas 47

3.1.3.1 Software Zebra 47
3.1.3.2 Software Gated 48
3.1.3.3 Traceroute e Ping 49
3.1.4 PTTs no Brasil e a RNP 50
3.1.5 PTT RSIX (RS *Internet Exchange*) 51

4. Gerência de Redes 52

4.1 Structure of Management Information 53
4.1.1 Abstract Syntax Notation One 53
4.1.1.1 Tipos de Dados Abstratos 53
4.1.1.2 Módulos 53
4.1.1.3 Macros 54
4.1.2 Basic Encondig Rules 54

4.2 Management Information Base 57
4.2.1 MIB BGP 58

4.3 SNMPv1 59
4.3.1 Segurança 60
4.3.2 Protocolo 60
4.3.2.1 GetRequest 60
4.3.2.2 GetNextRequest 61
4.3.2.3 SetRequest 62
4.3.2.4 Trap 62
4.3.2.5 Message/Pdu 63

4.4 SNMPv2 64
4.4.1 Protocolo 64
4.4.1.1 GetBulkRequest 65
4.4.1.2 InformRequest 65
4.4.1.3 Report 65
4.4.1.4 Messages/PDU 65

4.5 SNMPv3 66
4.5.1 Segurança 67
4.5.2 Arquitetura da Entidade SNMPv3 68

4.6 RMON 69

4.7 Considerações 71

5. Sistema de Gerenciamento do Protocolo BGP em PTTs 72

5.1 Arquitetura do sistema 72

5.2 Informações para Gerenciamento 74
5.2.1 MIB II 75
5.2.2 MIB RMON II 76
5.2.3 MIB BGP 76

5.3 MIBs resultantes do trabalho 78
5.3.1 MIB BGPe 78

5.3.2 MIB BGPe Traps 82
5.3.3 MIB BGPe Conf 84
5.3.4 MIB BGPe RSUtil 87

5.4 Modelagem de Entidades SNMP 89

5.5 Implementação do sistema 90
5.5.1 Classe Transport 91
5.5.2 Classe Dispatcher 91
5.5.3 Classe Pendente 93
5.5.4 Classe CommandGenerator 94
5.5.5 Classe CommandResponder 94
5.5.6 Classes V1Methods/V2Methods/V3Methods 95
5.5.7 Classe MIB 96
5.5.7.1 Estrutura da MIB 96
5.5.7.2 Implementação de MIBs 97
5.5.8 Classe MIB Compiler 99
5.5.9 Classes Message/Pdu 100
5.5.10 Entidade Agente 101
5.5.11 Entidade Dual 101

5.6 Simulação do sistema 102
5.6.1 Arquiteturas de validação do sistema 102
5.6.2 Softwares utilizados no protótipo 104
5.6.3 Ativação do protótipo e sistema de gerência 105
5.6.4 Utilização da MIB BGPe Conf 105
5.6.5 Utilização da MIB BGPe 106
5.6.6 Utilização da MIB BGP Traps 106
5.6.7 Simulação de problemas em PTTs 107
5.6.7.1 Queda da sessão BGP de algum participante 107
5.6.7.2 Número de anúncios fora do intervalo esperado 109
5.6.7.3 Número de anúncios e histórico de anúncios de participantes 110
5.6.7.4 Quantificação da profundidade de anúncios 111
5.6.7.5 Anúncio de redes inválidas 112
5.6.7.6 Propagação de anúncios de outros sistemas autônomos 112
5.6.7.7 Saturação de *link* de participante do PTT 113

6. Conclusões 114

Referências Bibliográficas 115

Anexo A 117

Anexo B 124

Anexo C 126

Anexo D 131

Anexo E 133

Anexo F **139**

Anexo G **144**

Anexo H **147**

LISTA DE ABREVIATURAS

AS	*Autonomous System*
ASN.1	*Abstract Syntax Notation One*
BER	*Basic Encoding Rules*
BGP	*Border Gateway Protocol*
CCITT	*Community Colleges for Innovative Technology Transfer*
CMOT	*CMIP over TCP/IP*
CMIP	*Common Management Information Protocol*
DES	*Data Encryption Standard*
DOD	*U.S. Department of Defense*
DVMRP	*Distance Vector Multicast Routing Protocol*
EGP	*External Gateway Protocol*
EIGRP	*Enhanced Interior Gateway Routing Protocol*
HEMS	*High-Level Entity Management System*
HMAC	*Keyed-Hashing Message Authentication Codes*
HMP	*Host Monitoring Protocol*
IAB	*Internet Architecture Board*
ICMP	*Internet Control Message Protocol*
IETF	*Internet Engineering Task Force*
IGP	*Interior Gateway Protocol*
IGRP	*Interior Gateway Routing Protocol*
INOC	*Internet Network Operations Center*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
IX	*Internet Exchange*
MD5	*Message Digest*
MIB	*Management Information Base*
MSDP	*Multicast Source Discovery Protocol*
NAP	*Network Access Point*
NRLI	*Network Layer Reachability Information*
NSFNET	*National Science Foundation Network*
OID	*Object Identifier*
OSI	*Open Systems Interconnection*
OSPF	*Open Shortest Path First*
PDU	*Protocol Data Unit*
PIM	*Protocol Independent Multicast*
POP	Ponto de Presença na rede
PTT	Ponto de Troca de Tráfego
RIP	*Routing Information Base*
RMON	*Remote Monitoring*
SGMP	*Simple Gateway Monitoring Protocol*
SHA	*Secure Hash Algorithm*
SMI	*Structure of Management Information*
SMP	*Simple Management Protocol*
SMTP	*Simple Mail Transfer Protocol*

SNMP	*Simple Network Protocol*
SPF	*Shortest Path First*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
VLSM	*Variable Length Subnet Masks*

LISTA DE FIGURAS

Figura 2-1 - Exemplos de utilização de protocolos IGP e EGP em sistemas autônomos....20
Figura 2-2 - Decisão de melhor caminho segundo RIP....22
Figura 2-3 - Máquina de estados finito para sessões do protocolo BGP.....26
Figura 2-4 - Exemplo ilustrativo de situação de trânsito BGP com clientes *multihomed*. 30
Figura 2-5 – Formato *header* genérico do BGP 31
Figura 2-6 – Mensagem OPEN 32
Figura 2-7 – Mensagem NOTIFICATION....33
Figura 2-8 - Mensagem UPDATE....34
Figura 2-9 - Formato do campo *PathAttribute* 35
Figura 2-10 - Tabela BGP e atributos....38
Figura 2-11 - CIDR 39
Figura 2-12 - Exemplo de utilização de iBGP e eBGP 40
Figura 3-1 - Situação das Conexões entre Brasil, Uruguai e Argentina....41
Figura 3-2 - Exemplo de PTT baseado em acordos bilaterais.....43
Figura 3-3 - Exemplo de acordos bilaterais em PTT.....43
Figura 3-4 - Ilustração de PTT baseado em acordos multilaterais. 44
Figura 3-5 - Exemplo de PTT baseado em *Route Server*. 45
Figura 3-6 - PTT baseado em *Route Servers*, utilizando *Looking Glass*....46
Figura 3-7 – Configuração BGP no Zebra 48
Figura 3-8 – Configuração BGP no Gated 49
Figura 3-9 - *Traceroute* 50
Figura 3-10 - Conexões entre os backbones da RNP e os PTTs nacionais existentes. 50
Figura 4-1 Arquitetura de Gerência....52
Figura 4-2 – TLV (Type Length Value)....55
Figura 4-3- BER 56
Figura 4-4 - Codificação de OID....57
Figura 4-5 – Árvore da MIB....57
Figura 4-6 - MIB-II 58
Figura 4-7 - MIB BGP....59
Figura 4-8 – Operação *GetRequest*....61
Figura 4-9 – Operação *GetNextRequest* 61
Figura 4-10 - Operação *SetRequest* 62
Figura 4-11 - Operação *Trap* 63
Figura 4-12 - Message e PDU SNMPv1 63
Figura 4-13 - Message e PDU SNMPv2 66
Figura 4-14 - Evolução SNMP 66
Figura 4-15 - Entidade SNMPv3 68
Figura 4-16 - SNMPv3 Messages....69
Figura 4-17 - RMON 70
Figura 5-1 – Arquitetura do Sistema 73
Figura 5-2 – Exemplo de aplicação do conceito de VLAN....75

Figura 5-3 – Exemplo de consulta aos índices das interfaces do *Switch*.....76
Figura 5-4 – Exemplo de *Get* 76
Figura 5-5 - Topologia criada para teste da MIB BGP.....77
Figura 5-6 - Saída do *snmpwalk* 78
Figura 5-7 - MIB BGPe.....79
Figura 5-8 - MIB BGPeTrap 82
Figura 5-9 - Ilustração da MIB de configuração do sistema 85
Figura 5-10 - Arquivo da BgpeConf 87
Figura 5-11 - BGPeRSUtil 88
Figura 5-12 – Comunicação entre os módulos de um Agente SNMP.....89
Figura 5-13 – Comunicação entre os módulos de um Gerente SNMP.....90
Figura 5-14 – Classe Transport 91
Figura 5-15 – Classe Dispatcher.....92
Figura 5-16 – Filas da classe Pendente.....93
Figura 5-17 – Classe CommandResponder 94
Figura 5-18 – Métodos da classe VXMethods 95
Figura 5-19 – Métodos da classe V2Methods 95
Figura 5-20 - MIB Exemplo 96
Figura 5-21 – Classes empregadas na implementação de grupos (exemplo Bgpe).....97
Figura 5-22 –Métodos da Interface Variáveis 97
Figura 5-23 – Exemplos de classes tipo Nodo Folha 98
Figura 5-24 – MibListener.....98
Figura 5-25 – Exemplo de classe de MIB (Exemplo da Bgpe.Java).....100
Figura 5-26 – Estrutura das classes Message e Pdu 101
Figura 5-27 - Ilustração do protótipo de PTT no laboratório de redes.....102
Figura 5-28 - Arquitetura do protótipo final para validação 103
Figura 5-29 – Exemplo de Set no objeto BgpeConfMaxPath da MIB BgpeConf 105
Figura 5-30 – Exemplo de SnmpGet no objeto BgpeTotalPath da MIB Bgpe 106
Figura 5-31 – Trap de sessão BGP down do participante 2, recebida pelo software JopenEyes. 108
Figura 5-32 – Trap de sessão BGP up do participante 2, recebida pelo software JopenEyes. 108
Figura 5-33 – Corpo do e-mail que sinaliza a normalização da sessão BGP do participante 2. 108
Figura 5-34 - Exemplo de trap enviada quando o número de anúncios ultrapassa o esperado. 109
Figura 5-35 - Gráfico que ilustra o número total de anúncios de cada participante.....110
Figura 5-36 - Gráfico que ilustra o total de anúncios global no PTT.....111
Figura 5-37 - Classificação de anúncios por tamanho de um dos participantes do PTT 112
Figura A-1 - Configuração do participante 1 134
Figura A-2 - Configuração do *Route Server* 1.....135
Figura A-3 - Configuração do *Looking Glass* 135
Figura A-4 - *Status* das conexões BGP com os *Route Servers* do participante 1.....135
Figura A-5 - Sessões BGP de *Route Server* 1 136
Figura A-6 - Tabela BGP do *Route Server* 1.....136
Figura A-7 - Tabela BGP do participante 1 (AS 1).....137

Figura A-8 - Tabela BGP do *Looking Glass* 137
Figura A-9 - Tabela BGP do *Route Server* 2, sem filtros por AS_PATH.....138
Figura A-10 - Configuração do participante 1.....140
Figura A-11 – Configuração do RSD.....141
Figura A-12 – Visão das sessões.....141
Figura A-13 – Participante 1 142
Figura A-14 – Anúncios recebidos pelo AS1.....142
Figura A-15 – Rotas da tabela do participante 1 143
Figura A-16 - Diagrama de classes das mensagens SNMP.....147
Figura A-17 - Diagrama de classes do *MibCompiler*.....148
Figura A-18 - Diagrama de classes da entidade Agente.....149
Figura A-19 - Diagrama de classes da entidade Dual 150

LISTA DE TABELAS

Tabela 2-1 – Descrição dos campos do *header* 31
Tabela 2-2 – Descrição dos campos da mensagem OPEN 32
Tabela 2-3 – Descrição dos campos da mensagem NOTIFIFCATION 33
Tabela 2-4 – Erros da mensagem NOTIFICATION 33
Tabela 2-5 - Descrição do subcampo *Atribute flags* do campo *Path Attribute* 35
Tabela 2-6 - Atributos BGP 36
Tabela 2-7 - Relação de alguns atributos, classificação e RFC associada 36
Tabela 2-8 - Descrição dos atributos BGP 37
Tabela 3-1 – Portas Zebra 48
Tabela A-1 - Endereçamento de cada participante 133
Tabela A-2 - Definição de rede/bloco CIDR para cada participante 133
Tabela A-3 - Endereçamento dos PCs 139
Tabela A-4 - Definição de rede/bloco CIDR para cada participante 139
Tabela A-5 - Objetos BGPe 144
Tabela A-6 - Objetos BGPeConf 145
Tabela A-7 - Objetos BGPeRSUtil 145
Tabela A-8 - Objetos BGPeTraps 146

AGRADECIMENTOS

Agradeço a todos que contribuíram para o bom andamento deste Trabalho de Conclusão. Em especial para os meus Pais Ronaldo e Márcia que tornaram tudo isso possível. Para os meus irmãos e familiares que sempre me apoiaram. Para a nossa orientadora a professora Ana Benso que conseguiu nos guiar de forma brilhante para a conclusão do trabalho. Para o professor avaliador Avelino Zorzo que avaliou de forma séria e correta o trabalho. Para todos os professores da Faculdade de Informática que contribuíram para a minha formação e para todos os meus amigos e colegas do curso de Ciências da Computação. Em especial ao amigo Andrey com quem tive o privilégio de fazer este belo trabalho.

Um sincero abraço.

Fabio

Ao final de uma caminhada, nada melhor do que compartilhar com aqueles que nos acompanharam lealmente em todos as horas, inclusive nas horas difíceis.

Olho para meu ser e vejo uma Mãe e um Pai maravilhosos, abrindo mão de conviver com o único filho para ele poder estudar e sentir o vento da vida bater no rosto. Em todos os momentos, principalmente nos difíceis, vocês sempre estiveram ao meu lado. Em tudo o que lembro de maravilhoso em minha vida, vocês sempre estiveram comigo. E como era bom chegar em casa depois de uma semana de estudos e dar aquele abraço em vocês, matar a saudade, conversar, dividir alegrias e sentir tanto amor.

O orgulho de vocês foi aumentando cada vez mais, até o momento que me senti tão fraco, diante da situação de não conseguir fazer vocês ficarem mais tempo nesse Mundo. Senti-me frustrado por procurar vocês no meio de tanta gente e não encontrá-los. Pobre, por não poder abraçar as pessoas mais importantes de minha vida. Em seguida, de lá de cima, comecei a sentir a força que vocês sempre me deram de forma muito mais intensa e ao mesmo tempo em que as lágrimas caiam, aumentar a vontade de tornar os sonhos de vocês realidade. Sinto vocês no meu lado, me envolvendo com tanto amor que me fez voltar a sentir de leve o sabor das coisas boas.

Essa conquista, dedico inteiramente a vocês, pois vocês têm sido muito mais do que pais. Por maiores que sejam as vitórias e conquistas do futuro, ser filho de vocês sempre será minha maior conquista.

A você Alice, minha namorada, meu profundo agradecimento por toda a compreensão, apoio e amor que tens me dado.

A orientadora Prof. Ana Benso meu mais sincero agradecimento pela pessoa maravilhosa que tens sido, pelo apoio nas horas difíceis e pela grande amiga para conversar e me ajudar. Aos demais professores e funcionários, meu muito obrigado por cada lição e cada minuto de convivência.

Ao pessoal do CPD da UFRGS, em especial à Prof. Liane Tarouco e ao Leandro Bertholdo, minha gratidão por terem me dado a chance de trabalhar com vocês. Fabrício, Rodrigo, Leandro Rey, meu muito obrigado.

Ao colega Fábio Falci Rodrigues, um abraço e o meu sincero muito obrigado, pelas noites fazendo este trabalho, pelos finais de semana discutindo o trabalho e pela sua tão valiosa contribuição no TC e na nossa amizade.

A vocês colegas e grandes amigos, obrigado pela força, pelo companheirismo, pelos estudos, pela lealdade, pela convivência e pelos desafios que superamos juntos.

A Deus, obrigado pela força, pelas tantas coisas boas que tive em minha vida, pelos amigos, pela saúde e pelos pais eternos que tenho.

Sem todos vocês, essa conquista não teria o mínimo valor.

Forte abraço,

Andrey

RESUMO

Este trabalho apresenta um estudo de gerenciamento do protocolo de roteamento BGP (*Border Gateway Protocol*), pois seu correto funcionamento é vital para o sucesso do roteamento do tráfego entre sistemas de redes de computadores de diferentes domínios administrativos.

A interconexão de grandes redes e *backbones* da Internet são chamadas de NAP´s (*Network Access Points*) ou PTTs (Pontos de Troca de Tráfego). Assim sendo, essa proposta utiliza a arquitetura de um PTT para analisar o tráfego BGP, identificar os requisitos de gerenciamento do protocolo e elaborar um conjunto de informações úteis ao seu gerenciamento.

Os resultados do trabalho foram a especificação de uma MIB (*Management Information Base*) experimental para o BGP, baseada em MIBs já existentes e a implementação de agentes SNMP para execução das tarefas de gerenciamento necessárias.

Palavras-chave: Protocolos de roteamento, BGP, gerência de redes, PTTs, NAPs.

ABSTRACT

This work describes a management system of BGP (Border Gateway Protocol), because its corrects functioning is very important to traffic routing success between networks of different Autonomous Systems. The interconnection of large Internet networks and backbones is called NAPs (Network Access Point). Therefore, this work uses a PTT architecture to analyses BGP traffic, identify the requirements for protocol management, and collect of information for its management.

The results of this work are a specification of an experimental MIB (Management Information Base) based on existing MIBs, and the implementation of SNMP agent to execute the management tasks.

Keywords: Routing protocols, BGP, management of networks, PTTs, NAPs.

1. INTRODUÇÃO

Os elementos de interconexão de redes são essenciais para o funcionamento da Internet, em especial os roteadores. Eles estão ligados entre si de forma que uma mensagem possa chegar ao seu destino rapidamente e com eficácia. Logo, o processo de configuração do roteamento deve ser robusto e eficaz.

Os roteadores devem ser configurados para que reconheçam o caminho que os pacotes deverão seguir para atingir o seu destino. Essa configuração pode ser realizada através de protocolos de roteamento. Existem vários protocolos de roteamento, os quais são empregados com escopos diferenciados. O BGP (*Border Gateway Protocol*) é um dos protocolos utilizados para a configuração de roteadores que interligam redes de domínios administrativos diferenciados.

Em cada roteador que suporta BGP, tem-se uma tabela de rotas. Cada linha dessa tabela indica para quem este roteador deve enviar cada pacote para alcançar tal destino. Ele consegue esta informação trocando mensagens com os seus roteadores vizinhos e gravando na tabela de roteamento os anúncios que recebe. Isso é feito até se chegar num ponto de convergência, ou seja, quando os roteadores tiverem a mesma tabela de rotas. No começo, estas tabelas eram relativamente pequenas, mas com o crescimento da Internet as tabelas começaram a se tornar grandes e complexas. Outro ponto que interfere diretamente nas tabelas de rotas é a topologia das redes. Isso torna a construção da tabela de rotas uma tarefa difícil e contínua, pois podem existir várias rotas para chegar a um determinado destino e o BGP terá que decidir qual será a melhor rota.

Para ter controle sobre toda a rede, é necessário o uso efetivo do gerenciamento. Com a complexidade e a probabilidade de falhas crescendo cada vez mais, a atividade de gerência torna-se essencial para o funcionamento da rede. Se por exemplo, uma rota, calculada de forma errada, for propagada para os roteadores, muitos pacotes importantes podem se perder causando prejuízos a todos. Logo, a atividade de gerenciamento tem condições de perceber que algo de errado está acontecendo e enviar um aviso para o responsável, reportando a situação. Então esse responsável poderá, apoiando-se nas informações de gerência, identificar, corrigir e executar ações preventivas em relação às falhas que poderiam ocorrer. Tais informações, que são utilizadas para armazenar e representar informações relevantes sobre o gerenciamento são denominados MIBs (*Management Information Base*).

1.1 Objetivos gerais e específicos

Assim sendo, o objetivo principal deste trabalho foi desenvolver um sistema de gerenciamento do protocolo BGP, para utilização em locais de redes de computadores de domínios administrativos diferentes. O objetivo deste sistema foi identificar possíveis falhas e situações anômalas a respeito das operações do protocolo BGP, mantendo a equipe de suporte informada sobre esses eventos. O trabalho envolveu o estudo de arquiteturas e protocolos de gerenciamento de redes de computadores, do protocolo de roteamento BGP e de arquiteturas de PTTs (Pontos de Troca de Tráfego),

que representam umas das formas de interconexão das redes de diferentes domínios administrativos.

1.2 Contribuições

Baseado no contexto apresentado, o presente trabalho busca fornecer as seguintes contribuições:

- Implementação de um agente para captura de informações que não estão presentes em outras MIBs.
- Implementação de uma MIB para busca de informações complexas.
- Implementação de um agente Dual para implementação das MIBs.
- Implementação de um sistema de monitoração que faça o envio de TRAPs.
- Implementação de agente SNMP versão 2 na linguagem Java, que possa ser estendido para a versão 3, possibilitando seu uso em outros trabalhos da área.

Baseado nisso, buscou-se atender às necessidades para uma administração eficiente do protocolo BGP em PTTs.

1.3 Organização do texto

O texto encontra-se dividido em capítulos conforme descrito a seguir: o Capítulo 2 apresenta a evolução da Internet, o protocolo BGP; o Capítulo 3 apresenta os Pontos de Troca de Tráfego; o Capítulo 4 aborda o conceito de gerência de redes, a evolução do protocolo SNMP, bem como as suas MIBs; o Capítulo 5 apresenta a proposta detalhada deste trabalho e o Capítulo 6 apresenta a conclusão seguida pelas referências bibliográficas e anexos.

2. ROTEAMENTO

Segundo [COM98], a ARPANET foi a rede que iniciou o *backbone* da Internet. Nessa época cada participante administrava suas tabelas de roteamento e suas atualizações eram feitas manualmente.

Dessa forma, a topologia da Internet poderia ser descrita como um conjunto de roteadores básicos, conhecidos também por *core routers* ou também por roteadores de borda. Estes roteadores eram controlados pelo INOC (*Internet Network Operations Center*). Adicionalmente, existia também um conjunto de roteadores, conhecidos como secundários que eram administrados por grupos isolados e eram conectados aos *core routers*, fornecendo acesso às redes locais de cada instituição. Em seguida, surgiu a NFSNET que inicialmente estabeleceu um único ponto de conexão com a ARPANET, posteriormente aumentando o número dessas conexões.

Isso acabou aumentando a complexidade do roteamento global, sem contar que aos poucos o *backbone,* formado pelos *core routers*, passou a não ser mais administrado de forma centralizada. Assim, ficou evidente que a arquitetura de roteamento básica não seria suficiente para uma operação e administração satisfatória da rede como um todo. Dessa forma, notou-se que o esquema da rede constituída pelos roteadores básicos interligados não suportaria a conexão de todas as redes através dos roteadores secundários. Por esse motivo o esquema de divisão por hierarquias foi sendo substituído pelo modelo distribuído que é o modelo utilizado até os dias de hoje. Assim sendo, o roteamento não fica mais centralizado em alguns roteadores, como nos roteadores básicos da NFSNET, mas cada roteador é responsável pelas suas redes e pela comunicação com os demais roteadores da Internet.

Para acompanhar tais mudanças, foi criado o conceito de sistema autônomo (*Autonomous System*, ou simplesmente AS), que faz com que as redes e roteadores que estão sob uma mesma política sejam administradas pela própria entidade que os possui. Assim, o que ocorre internamente a um AS não será conhecido por outros sistemas autônomos, diminuindo a complexidade da Internet global. Para a comunicação com o mundo externo, ou seja, com os demais sistemas autônomos, deve ser utilizado pelo menos um dos roteadores do AS para trocar informações com os demais ASs e garantir a alcançabilidade entre suas redes. Para cada AS é atribuído um AS *number*, que é uma identificação única que o identifica para os demais sistemas autônomos. Dessa forma, a Internet global deixou de ser vista como um grande grupo de redes locais interligadas, mas agora como um conjunto de Sistemas Autônomos que trocam anúncios de rotas para suas redes entre si.

Diante de tais inovações e a distribuição desta nova arquitetura, surgiu a necessidade de protocolos de roteamento capazes de manter a consistência e as informações entre os sistemas autônomos da Internet, incluindo também suas redes internas. Assim surgiram diversos protocolos de roteamento com o intuito de atender a estas necessidades e funcionalidades. Por sua vez, esses protocolos baseiam-se em algoritmos de roteamento. Entre os algoritmos de roteamento mais populares, dois deles são apresentados na seção 2.1.

Os protocolos de roteamento dividem-se em dois grupos, explicados a seguir:

- EGP (*Exterior Gateway Protocol*): Grupo de protocolos utilizados para a comunicação inter AS, ou seja, usado para a comunicação entre roteadores que se encontram em diferentes sistemas autônomos. Os protocolos deste tipo garantem que todos os sistemas autônomos pela Internet mantenham informações consistentes para garantir o funcionamento do roteamento global. Exemplo de protocolo deste grupo seria o BGP.
- IGP (*Interior Gateway Protocol*): Grupo de protocolos utilizados na comunicação intra AS, ou seja, usados para comunicação entre roteadores em um mesmo sistema autônomo. Hoje este grupo é representado por vários protocolos, como RIP, OSPF, IGRP, entre outros.

Através da Figura 2-1 são apresentados exemplos de utilização dos dois grupos de protocolos. Entre o AS 1 e AS 2 é utilizado o grupo de protocolos EGP, utilizando o grupo IGP para propagação das rotas aos demais roteadores de cada sistema autônomo.

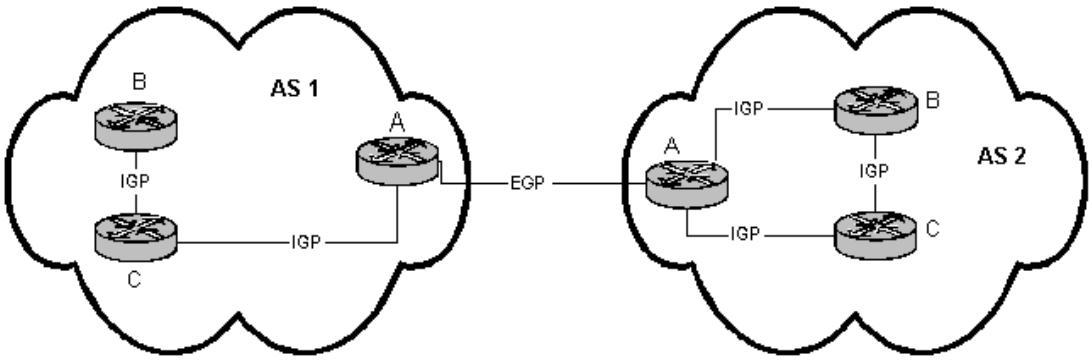

Figura 2-1 - Exemplos de utilização de protocolos IGP e EGP em sistemas autônomos

O processo de configuração do roteamento interno a um AS pode ser feito de duas formas. A primeira delas é estática, ou seja, é baseada na configuração manual de rotas já que em geral não existe mais que um caminho para chegar a determinada rede ou roteador do AS. A desvantagem desta forma de configuração é que, na ocorrência de alguma mudança na rede ou falha em alguma das conexões, o administrador deve fazer todas as alterações necessárias para restabelecimento da comunicação manualmente. Também se torna óbvio que a confiabilidade e o tempo de resposta a problemas podem não ser satisfatórios. Já a forma dinâmica, baseia-se na utilização de protocolos de roteamento que automatizam tal tarefa, ou seja, que fazem o anúncio de rotas e detecção do melhor caminho de forma automática, sem a intervenção do administrador da rede. Caso a decisão seja de utilizar um protocolo dinâmico em um

AS, deverá ser utilizado um protocolo tipo IGP para tal tarefa. Já para a configuração externa de um AS deverá obrigatoriamente ser utilizado um protocolo tipo EGP, visto que não existe outra forma de propagação das redes pertencentes a ele.

2.1 Algoritmos de roteamento

Os protocolos dinâmicos podem implementar diversos algoritmos de roteamento. Alguns destes algoritmos utilizados são apresentados abaixo:

a) Vetor distância: Também definido como algoritmo de *Bellman-Ford*, este algoritmo trabalha baseado na idéia que cada roteador propaga periodicamente uma tabela com todas as redes conhecidas e a distância para alcançá-las. Geralmente, a distância é calculada pelo número de *next hops* necessários para alcançar uma determinada rede. O termo *HOP* caracteriza-se pela passagem entre um roteador e outro. Esse termo poderia ser equivalente a palavra "salto". Sendo assim, cada roteador, ao receber os anúncios de todos os demais, calcula o caminho ótimo baseado no menor número de *HOPs* para chegar a determinada rede. Vale lembrar também, que cada roteador ao receber as informações de outras redes incrementa o número de *HOPs* e anuncia as rotas divulgadas para os demais roteadores.

b) Estado de enlace: Também definido como algoritmo *Link State*, este algoritmo trabalha baseado na idéia de que cada roteador possui informações sobre as redes que estão conectadas a ele e, periodicamente, testa para determinar se cada enlace está ativo. Com estas informações cada roteador divulga uma lista sobre o *status* de cada conexão, dizendo se estas estão ativas ou inativas. Baseado nessas informações, quando um roteador recebe um conjunto de mensagens sobre o estado dos enlaces das redes próximas a ele, é aplicado o algoritmo SPF de *Dijskstra*. Este algoritmo é aplicado baseado nas informações de cada roteador e é feito localmente a cada um destes, para o cálculo das melhores rotas para todos os destinos a partir de uma mesma origem. Em termos de expansão, este algoritmo tem vantagem sobre o Vetor Distância, pois o cálculo do melhor caminho é feito localmente e não depende do cálculo de roteadores intermediários. Outra vantagem é que devido a suas características, este algoritmo converge mais rapidamente devido a utilização de *flooding* para divulgação do estado de seus enlaces, ou seja, divulga de forma mais eficaz os melhores caminhos para suas redes a todos os roteadores conectados.

Baseados nesses algoritmos são implementados os protocolos dinâmicos, apresentando nas seções 2.2 e 2.3 os grupos de protocolos IGP e EGP, respectivamente.

2.2 Protocolos IGP

A partir dos algoritmos apresentados na seção 2.1, foram implementados diversos protocolos de roteamento IGP que são largamente utilizados nos dias de hoje, segundo [MAN97], [CIS97] e [COM98]. Os protocolos interiores (IGP) mais conhecidos são apresentados a seguir:

a) RIP: O *Routing Information Protocol* foi um dos primeiros protocolos IGP e foi muito popularizado pela implementação da ferramenta *routed*, muito usada em sistemas UNIX 4BSD. Basicamente, este protocolo trata da implementação direta do algoritmo de vetor distância. A métrica utilizada para o cálculo do melhor caminho baseia-se no número de roteadores que um pacote iria percorrer até chegar a seu destino, definido por *HOP*. Um dos problemas deste algoritmo é que pela aplicação da métrica sobre o número de *HOPs* o caminho ótimo nem sempre é contemplado. Um exemplo mais claro é mostrado pela Figura 2-2, onde de A para C existem dois caminhos. Um deles é pelo roteador B, tendo custo em 2 *HOPs* e o outro seria diretamente com custo 1. Mesmo que os *links* pelo roteador B sejam muito superiores em capacidade, ou ainda, mesmo que o *link* de 64 Kbps esteja totalmente utilizado, o melhor caminho sempre será pelo *link* de 64 Kbps já que a única métrica é o número de *HOPs*.

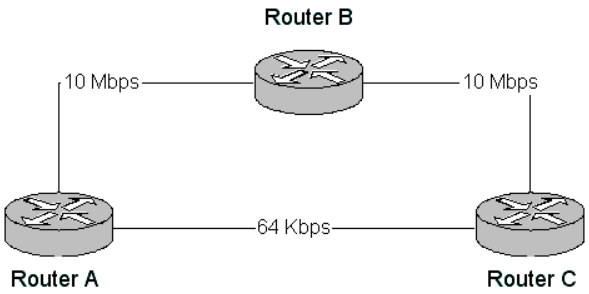

Figura 2-2 - Decisão de melhor caminho segundo RIP

O funcionamento do protocolo RIP baseia-se no anúncio de rotas feito a cada 30 segundos independente da alteração ou não de alguma rota. As rotas são removidas depois de 180 segundos se não houverem novos anúncios. Isso gera o problema de convergência, já que uma rota pode levar, no pior caso, até 180 segundos para ser removida de um roteador vizinho. Este problema fica ainda mais grave em redes com um grande número de roteadores. Existe também o problema que o RIP não carrega a informação de máscara das redes anunciadas, impossibilitando a divulgação de subredes. A versão 2 deste protocolo apresenta algumas novidades como a inclusão de divulgação da máscara de rede, permitindo a divulgação de subredes e também a possibilidade de autenticação.

b) OSPF: O *Open Shortest Path First* implementa o algoritmo do estado de enlace e, nos dias de hoje, é o mais popular entre os IGPs. Seu funcionamento é

baseado em *Link State Advertisements* (LSAs) que são a forma de divulgação sobre os estados dos enlaces de cada roteador. Este protocolo apresenta soluções para problemas de convergência, *loops* e outros problemas dos protocolos que implementam o algoritmo do vetor distância. Além disso, fornece funcionalidades importantes como balanceamento de carga, autenticação de mensagens, múltiplas métricas, separação de caminhos por tipo de serviço e ainda, fornece uma flexibilidade em sua operação. Um dos pontos negativos é que em redes de porte médio o OSPF começa a ser um protocolo pesado, exigindo que ao receber um LSA, cada roteador calcule novamente toda a sua matriz de rotas para verificar o caminho ótimo para cada rede.

c) IGRP: O *Interior Gateway Routing Protocol* baseia-se no algoritmo tipo vetor distância e é apresentado como uma correção das deficiências do protocolo RIP. Seu funcionamento é muito semelhante ao RIP, propondo melhoramentos quanto a escalabilidade, por possuir várias métricas e ainda permitir roteamento por múltiplos caminhos, ou seja, possibilitar um balanceamento de carga. Protocolo utilizado exclusivamente em roteadores Cisco.

d) EIGRP: O *Enhanced* IGRP usa o mesmo algoritmo vetor distância que o IGRP, mas implementa algumas funcionalidades como *Neighbor discovery*/*recovery*, as decisões de melhores rotas são feitas baseadas no *Diffusing Update Algorithm* (DUAL) e suporta VLSM (*variable length subnet masks*). Protocolo utilizado exclusivamente em roteadores Cisco.

e) IS-IS: O *Intermediate System to Intermediate System* é um protocolo baseado em estado de enlace usado em equipamentos *Digital* como as estações DECnet. Seu funcionamento é muito semelhante ao OSPF. Seu uso não é recomendado atualmente visto que poucos fabricantes ainda implementam este protocolo.

Dessa forma, mesmo existindo diversos protocolos tipo IGP, o mais utilizado deles é o protocolo OSPF, devido a suas funcionalidades que atendem às necessidades atuais. Em topologias mais simples e de menor escala, o segundo protocolo mais utilizado é o RIP.

2.3 Protocolos EGP

Uma vez que os protocolos interiores tenham sido apresentados, é abordado nessa seção o conjunto de protocolos EGP, que reúne os protocolos com funcionalidades específicas para comunicação entre sistemas autônomos. O número de protocolos deste tipo, se comparado aos IGPs, é reduzido visto que o protocolo BGP tem se tornado o padrão entre os EGPs.

Os protocolos exteriores (EGP) mais conhecidos na Internet são:

a) EGP: O *Exterior Gateway Protocol* foi o primeiro protocolo para comunicação utilizado principalmente entre ASs, de acordo com a RFC 904.

Foram encontrados diversas limitações técnicas e problemas em sua operação. Sendo assim, já que era necessária uma mudança em sua estrutura básica, este protocolo seguiu até a versão 3 e foi descontinuado. Alguns problemas e limitações que comprometeram o EGP foram problemas na detecção de *loops*, além de restrições de topologia.

b) BGP: O *Border Gateway Protocol* surgiu após a versão 3 do EGP, propondo soluções para os principais problemas encontrados no EGP, possibilitando uma flexibilidade melhor na utilização de políticas. O BGP é o protocolo que sustenta a Internet nos dias de hoje e é quem garante a operação global da rede. Essa tarefa se torna ainda mais especial quando lembramos da heterogeneidade que forma as redes atuais, sua complexidade e as necessidades no tratamento de novas aplicações e protocolos como IPv6 e *Multicast* por exemplo. Na próxima seção este protocolo será apresentado com maiores detalhes.

2.4 BGP (*Border Gateway Protocol*)

2.4.1 Introdução

O BGP é um protocolo de roteamento dinâmico utilizado para comunicação entre sistemas autônomos (ASs). Baseados nestas informações, os sistemas autônomos conseguem trocar informações e determinar o melhor caminho para as redes que formam a Internet. Tal papel é muito importante sabendo que a todo momento as redes podem sofrer alterações, podem ocorrer quedas de suas conexões, receber anúncios inválidos, aplicar políticas, manter a conectividade por outros caminhos, adaptando-se rapidamente e mantendo a consistência de seus anúncios de forma eficiente.

A divulgação das informações de roteamento BGP é feita entre roteadores que estabelecem uma relação de "vizinhança" (*peers*), sempre na forma de pares também chamada de *peering*. Tendo essa relação, são trocadas as informações contidas nas tabelas de roteamento BGP de cada um destes. Para estabelecer uma relação de vizinhança é necessário que dois roteadores tenham uma conexão direta entre eles, ou que algum protocolo IGP trate de garantir a alcançabilidade. Essa relação de vizinhança pode definir aos roteadores uma relação de *speakers* ou *peers*.

Tratando-se de um protocolo importante que requer confiabilidade em sua comunicação para garantir a alcançabilidade entre todas as redes da Internet, é necessário que seja utilizada uma forma confiável de troca de informações deste protocolo. Isso é obtido pela utilização do protocolo TCP entre dois roteadores que trocam informações do protocolo BGP. A porta utilizada para a comunicação é 179.

Para identificar univocamente cada sistema autônomo, cada AS possui um número que o identifica mediante os demais ASs da Internet. Este número varia entre 1 e 65535, sendo que a faixa entre 64512 e 65535 é destinada a uso privado.

Atualmente, segundo [CID02], o número de ASs ativos, em Novembro de 2002, que anunciam pelo menos uma rede é de 14002.

2.4.2 Modelo de divulgação e atualização das tabelas de rotas

O algoritmo que sustenta o BGP é definido como PATH VECTOR, assemelhando-se ao algoritmo de vetor distância, pois a partir de informações recebidas de outros sistemas autônomos é formado um vetor que armazena os ASs que formam um caminho para se chegar a determinada rede. Uma vez que os roteadores divulguem tal informação, é possível calcular o menor caminho para determinada rede. Nem sempre esse menor caminho é o escolhido, pois o BGP utiliza também diversos outros parâmetros para determinação do melhor caminho para determinada rede, que serão estudados a seguir.

Por tratar-se de tabelas de rotas de toda a Internet e a dinamicidade em que as alterações ocorrem, constantemente são trocadas mensagens de atualizações da tabela de roteamento. Para se ter uma idéia, a tabela de roteamento BGP completa da Internet no início do ano de 2002 possuía aproximadamente 107.000 rotas de acordo com [CID02]. Já o número extraído da mesma fonte, em Novembro de 2002 é de 116.000 rotas. A atualização de tabelas de rotas entre roteadores vizinhos não ocorre em intervalos de tempo pré-definidos, mas sim quando a tabela BGP sofre alguma mudança. Isso torna a divulgação mais leve, visto que ao nível do BGP o número total de rotas da Internet é muito grande e o anúncio de todas as rotas seria inviável. Esta forma de anúncio pode ser definida como incremental, ou seja, sendo enviadas apenas as atualizações. Este modo de atualizações incremental diminui consideravelmente o *overhead* e a banda utilizada para anúncios.

Para a comunicação entre roteadores BGP existem alguns tipos de mensagens onde cada um deles tem um papel importante na comunicação BGP.

- Mensagens tipo OPEN são utilizadas para o estabelecimento de uma conexão BGP;
- Mensagens tipo NOTIFICATION reportam erros e serve para representar possíveis problemas nas conexões BGP.
- Mensagens tipo UPDATE são utilizadas para os anúncios propriamente ditos, incluindo rotas que devem ser incluídas na tabela e também rotas que devem ser removidas da tabela BGP.
- Mensagens tipo KEEPALIVE são utilizadas para manter a conexão entre roteadores BGP caso não existam atualizações através de mensagens UPDATE.

Uma expressão utilizada para definir rotas que devem ser removidas da tabela BGP é *withdrawn*, que devido a dinamicidade da Internet ocorrem com muita freqüência.

Outra questão importante em roteadores BGP é a questão do chamado *Full Routing*. Este termo é usado em roteadores que recebem todos os anúncios de rotas da Internet. Esta característica é desejável em *core routers* que possuam múltiplos pontos de interconexão com outros *backbones*. Nesses casos, com a tabela de rotas

completa, será possível explorar e descobrir melhores rotas para uma determinada rede. Como efeito colateral, este recurso exige que os roteadores tenham bons recursos de CPU e memória.

Na maioria dos casos o recurso de *full routing* não é utilizado, pois os roteadores possuem geralmente apenas um ou dois pontos de interconexão com outros backbones, não permitindo nenhuma melhora significativa no roteamento, caso fosse usado *full routing.* A tabela de roteamento BGP possui um número que identifica sua versão, sendo incrementado cada vez que esta sofrer alguma modificação. Um exemplo de versão de tabela pode ser visto na Figura 2-10.

Na seção 2.4.6 estes tipos de mensagens serão detalhados e estudados mais profundamente.

2.4.3 Estados de uma conexão BGP

De acordo com [CIS00] a negociação de uma sessão BGP passa por diversos estados até o momento que é propriamente estabelecida e é iniciada a troca de anúncios de prefixos de cada vizinho BGP. Para demonstrar os estados possíveis na negociação, apresentamos a Figura 2-3 que ilustra a máquina de estados finitos:

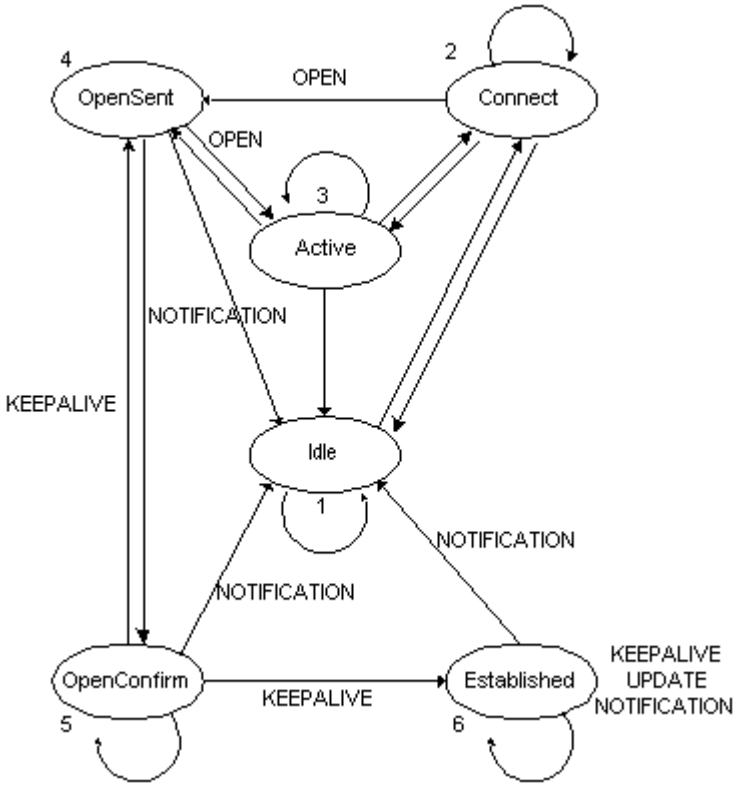

Figura 2-3 - Máquina de estados finito para sessões do protocolo BGP.

A seguir são apresentados a discutidos os 6 estados possíveis desta máquina de estados finitos:

- IDLE: Este estado identifica o primeiro estágio de uma conexão BGP, onde o protocolo está aguardando por uma conexão de um *peer* remoto. Esta conexão deve ter sido previamente configurada pelo administrador do sistema. O próximo estado é o de CONNECT e no caso da tentativa ser mal sucedida, volta ao estado IDLE.

- CONNECT: Neste estado o BGP aguarda pela conexão no nível de transporte, com destino na porta 179. Quando a conexão a este nível estiver estabelecida, ou seja, com o recebimento da mensagem de OPEN, passa-se ao estado de OPENSENT. Se a conexão do nível de transporte não for bem sucedida, o estado vai para ACTIVE. No caso do tempo de espera ter sido ultrapassado, o estado volta para CONNECT. Em qualquer outro evento, retorna-se para IDLE.

- ACTIVE: O BGP tenta estabelecer comunicação com um *peer* inicializando uma conexão no nível de transporte. Caso esta seja bem sucedida, passa-se ao estado OPENSENT. Se esta tentativa não for bem sucedida, pelo motivo de expiração do tempo, por exemplo, o estado passa para CONNECT. Em cada de interrupção pelo sistema ou pelo administrador, volta ao estado IDLE. Geralmente as transições entre o estado de CONNECT e ACTIVE refletem problemas com a camada de transporte TCP.

- OPENSENT: Neste estado o BGP aguarda pela mensagem de OPEN e faz uma checagem de seu conteúdo. Caso seja encontrado algum erro como número de AS incoerente ao esperado ou a própria versão do BGP, envia-se uma mensagem tipo NOTIFICATION e volta ao estado de IDLE. Caso não ocorram erros na checagem, inicia-se o envio de mensagens KEEPALIVE. Em seguida, acerta-se o tempo de *Hold Time*, sendo optado o menor tempo entre os dois *peers*. Depois deste acerto, compara-se o número AS local e o número AS enviado pelo *peer*, com o intuito de detectar se trata-se de uma conexão iBGP (números de AS iguais) ou eBGP (números de AS diferentes). Em caso de desconexão em nível de protocolo de transporte, o estado passa para ACTIVE. Para as demais situações de erro, como expiração do *Hold Time*, envia-se uma mensagem de NOTIFICATION com o código de erro correspondente e retorna-se ao estado de IDLE. No caso de intervenção do administrador ou o próprio sistema, também retorna-se o estado IDLE.

- OPENCONFIRM: Neste estado o BGP aguarda pela mensagem de KEEPALIVE e quando esta for recebida, o estado segue para ESTABLISHED e a negociação do *peer* é finalmente completa. Com o recebimento da mensagem de KEEPALIVE, é acertado o valor negociado de *Hold Time* entre os peers. Se o sistema receber uma mensagem tipo NOTIFICATION, retorna-se ao estado de IDLE. O sistema também envia periodicamente, segundo o tempo negociado, mensagens de KEEPALIVE. No caso da ocorrência de eventos como desconexão ou intervenção do operador, retorna-se ao estado de IDLE também. Por fim, na ocorrência de eventos diferentes aos citados, envia-se uma mensagem NOTIFICATION, retornando ao estado de IDLE.

- ESTABLISHED: Neste estado, o BGP inicia a troca de mensagens de UPDATE ou KEEPALIVE, de acordo com o *Hold Time* negociado. Caso seja recebida alguma mensagem tipo NOTIFICATION, retorna-se ao estado IDLE. No recebimento de cada mensagem tipo UPDATE, aplica-se uma checagem por atributos incorretos ou em falta, atributos duplicados e caso algum erro seja detectado, envia-se uma mensagem de NOTIFICATION, retornando ao estado IDLE. Por fim, se o *Hold Time* expirar ou for detectada desconexão ou intervenção do administrador, também retorna-se ao estado de IDLE.

A partir da máquina de estados apresentada anteriormente, é possível saber qual o *status* de uma sessão BGP entre dois roteadores, podendo também iniciar uma investigação sobre qual problema pode estar ocorrendo em alguma sessão. O objetivo esperado é que todas as sessões BGP de um roteador mantenham-se no estado ESTABLISHED, visto que somente neste estado ocorre a troca de anúncios com o roteador vizinho.

2.4.4 Funcionamento do algoritmo de decisão

O processo de decisão do BGP baseia-se nos valores dos atributos de cada anúncio. Para reforçar a importância do algoritmo de decisão, em sistemas autônomos *multihomed* - conexão com mais de um AS, tendo mais de um caminho de saída para a Internet - é normal a ocorrência de múltiplas rotas para a mesma rede e nestes casos o algoritmo de decisão do BGP é que toma a decisão da melhor rota a ser utilizada. Para esse cálculo, são apresentados os 9 critérios de decisão, apresentados por ordem de precedência:

- Se o *next hop* não for alcançável, a rota é ignorada;
- Será preferida a rota que tiver maior valor de *Weight*, que se trata de um parâmetro proprietário da Cisco, utilizado localmente em

um roteador. Caso o equipamento não seja Cisco, este passo do algoritmo não será efetuado;

- Caso o parâmetro anterior seja o mesmo, será preferida a rota que tiver o maior valor de *Local Preference* (LOCAL_PREF);
- Caso o valor de *Local Preference* seja o mesmo, será preferida a rota com menor AS_PATH.
- Caso o AS_PATH tenha o mesmo tamanho, será preferida a rota com menor tipo ORIGIN, ou seja, serão priorizados os anúncios tipo IGP (i), seguido pelos EGP (e) e INCOMPLETE (?).
- Caso o tipo ORIGIN seja o mesmo, será preferida a rota com o atributo MED mais baixo caso as rotas tenham sido aprendidas a partir do mesmo AS.
- Caso as rotas tenham o mesmo valor de MED, será preferida a rota por eBGP a iBGP.
- Se o valor de MED for o mesmo, será preferido o anúncio vindo do roteador conectado via IGP mais próximo deste.
- Se o caminho interno for o mesmo, o atributo BGP ROUTER_ID será o responsável pela decisão (*tiebreaker*). Neste caso, será preferido o caminho cujo roteador possuir o menor ROUTER_ID, que nas implementações Cisco é definido como IP da interface *loopback* se esta estiver configurada. No caso do roteador não possuir interface *loopback* configurada, será escolhido o IP mais alto do roteador. Vale lembrar que para cada fabricante o ROUTER_ID pode ser baseado em outras informações.

Dessa forma, os anúncios são incluídos na tabela BGP e baseado nestes critérios, é escolhido o melhor caminho. Este melhor caminho, por sua vez, será incluindo na *forwarding table,* que é utilizada de fato par o encaminhamento de pacotes pelo roteador.

2.4.5 Utilização de políticas

O protocolo também fornece diversos mecanismos para utilização de políticas de roteamento. Muitas das políticas aplicadas são relacionadas ao ato de troca de tráfego que tem relação direta com seus anúncios. O fato de um sistema autônomo ser trânsito define-se por este AS anunciar-se como caminho não somente para suas redes, mas para todas as demais redes que ele conhece. Outros ASs que não tenham o objetivo de fornecer trânsito, apenas anunciam suas próprias redes. Também podem existir casos que o AS anuncia suas rotas recebidas a apenas um conjunto restrito de ASs. Esta troca de tráfego é chamada de *peering* e é feita geralmente mediante acordos entre ASs, como é o caso dos NAPs ou PTTs ou em conexões particulares entre ASs. Para esclarecer melhor o assunto, apresentamos a Figura 2-4 que ilustra a situação de uma instituição X que compra acesso de dois provedores de acesso IP.

Ambos provedores, Embratel e BrasilTelecom, entre tantos outros que poderiam ser citados, possuem seu próprio AS. Com clientes que são *multihomed*, como é o caso a instituição X, que caracteriza-se por ter conectividade com mais um provedor

de *upstream*, o protocolo BGP é o mais aconselhável já que provê formas de aplicação de políticas de anúncios e formas eficientes de balanceamento de carga.

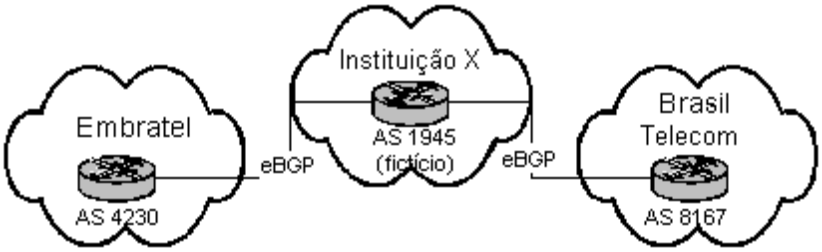

Figura 2-4 - Exemplo ilustrativo de situação de trânsito BGP com clientes *multihomed*.

Nesta situação a instituição X pretende utilizar tais conexões para acesso a redes da Internet e não para ser o elo de ligação entre a Embratel e BrasilTelecom (servir de trânsito). Para evitar que o AS 1945 se torne trânsito, a política que será aplicada será que todos os anúncios recebidos de um provedor nunca serão repassados ao outro provedor, ou seja, todos anúncios de redes da Embratel não serão divulgados a BrasilTelecom e vice-versa. Isso garantirá que ambas conexões sejam utilizadas para tráfego provenientes ou destinados apenas às redes do AS 1945 da instituição X. Essa postura de não servir como trânsito é um tanto óbvia já que o acesso a provedores como Embratel e BrasilTelecom é pago e por isso não faz sentido que alguém pague para servir de trânsito para outros *backbones*. Problemas como este já foram registrados nas operações com instituições conectadas ao POP-RS e a *backbones* comerciais.

O ato de servir de trânsito, se pelo lado dos clientes é evitado, pelos provedores de *upstream* como os citados são praticamente uma lei, pois estes devem propagar os anúncios de seus clientes para garantir que tais redes sejam alcançáveis por toda a Internet.

Em PTTs existem algumas restrições e pontos importantes sobre este tópico que serão explicados e aprofundados nos próximos capítulos.

Outro recurso importante é o *Route Dampening*, ou seja, uma espécie de "punição" que determinado AS pode levar caso seus anúncios sofram instabilidades constantes na tabela de roteamento (*FLAPs*). Isso faz com que determinado AS não seja "ouvido" pelos demais ASs por um tempo determinado, mantendo a estabilidade até que aquele anúncio estabilize. Isso evita que a instabilidade seja propagada por toda a Internet, consumindo banda e CPU de milhares de roteadores na inclusão/exclusão em suas tabelas de rotas BGP. Alguns *backbones* implementam tal funcionalidade, estabelecendo seus tempos de punição.

Outros recursos de políticas podem ser aplicados de acordo com a necessidade do administrador do AS, podendo filtrar tipos determinados de anúncios, baseado em algum parâmetro do protocolo BGP, aceitando ou filtrando tais anúncios. Esses procedimentos são muito utilizados e merecem cuidado ao manipula-los.

A utilização de políticas em um sistema autônomo é uma das tarefas mais importantes de um administrador, visto que sua configuração pode refletir em uma melhora no acesso a outras redes, até efeitos negativos, como problemas de alcançabilidade para outras redes, ou involuntariamente servir de trânsito para outros sistemas autônomos.

2.4.6 Mensagens do protocolo

O BGP possui basicamente 4 tipos de mensagens, segundo [CIS00]. Nestas mensagens existe um *header* que é comum a todos eles, apresentado na Figura 2-5 é explicado a seguir:

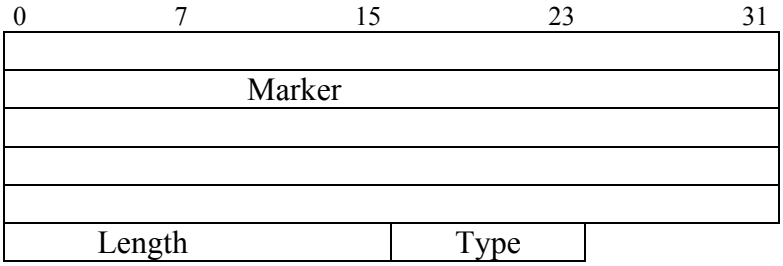

Figura 2-5 – Formato *header* genérico do BGP

A utilização de cada campo do *header* genérico do BGP é descrita pela Tabela 2-1.

Tabela 2-1 – Descrição dos campos do *header*

Marker	16 bytes	Em tipos de mensagem OPEN, todos os bits deste campo são preenchidos com 1´s. Se a mensagem não tiver nenhum tipo de autenticação, também deverá ser preenchida com 1´s. Caso seja utilizado algum tipo de autenticação e assinatura como MD5, este campo será utilizado para carregar informações de criptografia.
Length	2 bytes	Expressa o tamanho total da mensagem, incluindo o *header*. Este tamanho pode variar entre 19 bytes, que é o tamanho mínimo do próprio *header* e pode chegar até 4096 bytes.
Type	1 byte	Representa o tipo de mensagem, que pode ser OPEN, UPDATE, NOTIFICATION ou KEEPALIVE. Dependendo do tipo da mensagem os campos do corpo da mensagem variam. No caso do tipo KEEPALIVE, não existem campos adicionais além do próprio *header* da mensagem.

Uma vez que o *header* de mensagens BGP tenha sido apresentado, resta apresentar os campos que formam a área de dados de cada tipo de mensagem.

2.4.6.1 Mensagem tipo OPEN

Esta mensagem é utilizada para a negociação e estabelecimento de uma sessão BGP. Apenas com a aceitação da mensagem de OPEN que as demais mensagens podem ser trocadas. Abaixo é mostrada uma ilustração contendo os campos existentes em mensagens tipo OPEN, através da Figura 2-6.

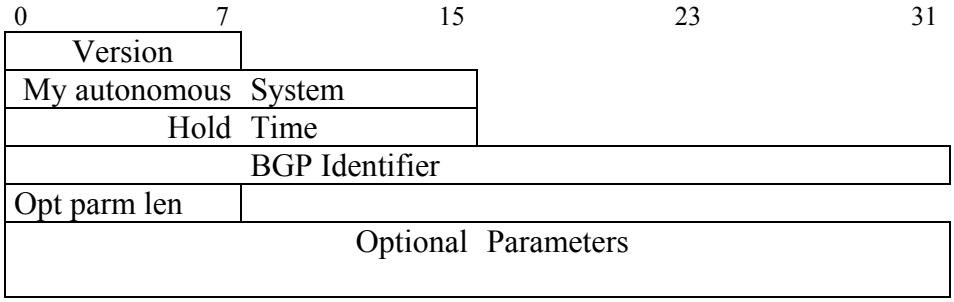

Figura 2-6 – Mensagem OPEN

A descrição destes campos é apresentada abaixo, através da Tabela 2.2:

Tabela 2-2 – Descrição dos campos da mensagem OPEN

Campo	Tamanho	Utilização
Version	1 byte	Versão da mensagem BGP. É negociada a maior versão existente nos *peers*. O *default* dessa mensagem é a versão 4.
My Autonomous System	2 bytes	Indica o número do AS que enviou a mensagem.
Hold Timer	2 bytes	Tempo máximo determinado para o envio das mensagens de KEEPALIVE ou UPDATE. Se dentro deste tempo nenhuma mensagem for recebida, a sessão BGP será considerada desativada.
BGP Identifier	4 bytes	Carrega a informação de BGP id, também conhecido como Router-id. Em geral, o router-id é escolhido como o IP mais alto existente no roteador, incluindo as interfaces *loopback*. Esse cálculo, dependendo do fabricante, pode ser diferente.
Optional Parameter Length	1 byte	Indica o tamanho do campo *Optional Parameters*. Caso não existam parâmetros adicionais, o conteúdo deste campo será 0.
Optional Parameters	Variável	Estes parâmetros são representados por tuplas formadas por <*Parameter Type*, *Parameter Length*, *Parameter Value*>. Estes dados possuem tamanho de um byte, com exceção do último campo que pode ter tamanho variável. Uma das utilizações deste campo seria nos parâmetros de autenticação nas mensagens tipo OPEN.

2.4.6.2 Mensagem tipo NOTIFICATION

Esta mensagem é utilizado na detecção de erros. Em geral este tipo de mensagem antecede o encerramento de uma sessão BGP. A ilustração dos campos pertencentes a este tipo de pacote é apresentada na Figura 2-7:

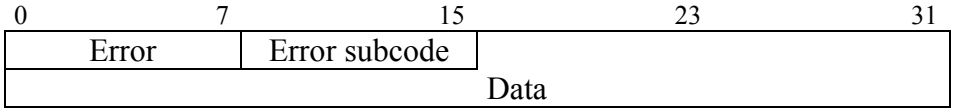

Figura 2-7 – Mensagem NOTIFICATION

A descrição dos campos deste tipo de mensagem é apresentada pela Tabela 2-3.

Tabela 2-3 – Descrição dos campos da mensagem NOTIFIFCATION

Campo	Tamanho	Utilização
Error Code	1 byte	Identifica o tipo de notificação.
Error Subcode	1 byte	Identifica de forma mais específica a natureza do erro.
Data	Variável	Apresenta informações relevantes sobre o erro detectado. Alguns exemplos seriam: *header* incorreto, número de AS inválido, entre outros.

Entre os grupos de erros e subdivisões (Error code e Error Subcode), a Tabela 2-4 lista os possíveis erros e subdivisões que podem ser reportados por este tipo de mensagem:

Tabela 2-4 – Erros da mensagem NOTIFICATION

Error Code	Error Subcode
1 – Message header error	1 – Connection Not Synchronized 2 – Bad Message Length 3 – Bad Message Type
2 – OPEN message error	1 – Unsupported Version Number 2 – Bad *Peer* AS 3 – Bad BGP Identifier 4 – Unsupported Version Number 5 – Authentication Failure 6 – Unacceptable Hold Timer 7 – Unsupported Capability
3 – UPDATE message error	1 – Malformed Attribute List 2 – Unrecognized Well-Know Attribute 3 – Missing Well-Know Attribute 4 – Attribute Flags Error 5 – Attribute Length Error 6 – Invalid Origin Attribute

	7 – AS Routing Loop 8 – Invalid NEXT_*HOP* Attribute 9 – Optional Attribute Error 10 – Invalid Network Field 11 – Malformed AS_PATH
4 – Hold Timer expired	Não aplicável
5 – Finite State Machine error (para erros detectados pela máquina de estados). Esta máquina de estados foi apresentada através da Figura 2-3.	Não aplicável
6 – Cease (trata erros considerados fatais e outros erros não listados).	Não aplicável

2.4.6.3 Mensagem tipo KEEPALIVE

Esta mensagem é utilizada para manter uma sessão BGP ativa. Para tanto, se dois roteadores que possuem uma sessão BGP não tiverem nenhuma mensagem tipo UPDATE para enviar ao outro, será enviada uma mensagem de KEEPALIVE para manter a conexão, antes que o *Hold Time* seja atingido e a conexão seja considerada inativa. Geralmente este tipo de mensagem é enviada ao atingir um terço do tempo de *Hold Time*. O tamanho desta mensagem é 19 bytes, sendo formado apenas pelo *header*, sem dados.

2.4.6.4 Mensagem tipo UPDATE

Esta mensagem pode ser considerada a mais importante, já que é responsável por intercambiar as atualizações de rotas. A mensagem de UPDATE é formada por campos que são divididos em três grupos por suas funcionalidades. Na Figura 2-8 é apresentado o formato da mensagem tipo UPDATE, bem como a apresentação de cada grupo e campos pertencentes a cada um destes.

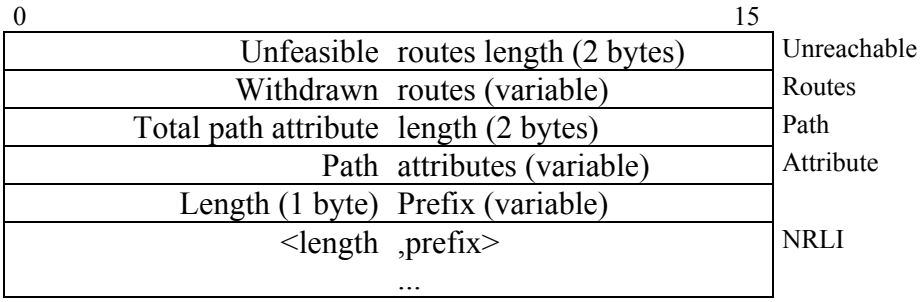

Figura 2-8 - Mensagem UPDATE

No primeiro grupo, definido como *Unreachable routes* são definidas as rotas que devem ser removidas da tabela de roteamento. Um termo comum para este tipo de evento é *withdrawn*. Da mesma forma, no grupo *Network Layer Reachability*

Information são definidas as rotas que devem ser incluídas na tabela de roteamento. A representação das rotas possui um mecanismo que suporta a funcionalidade conhecida como CIDR, apresentado na seção 2.4.6.4.1.

Em relação aos campos do grupo *Unreachable routes,* o campo *Unfeasible routes length*, com o tamanho de 2 bytes, representa o tamanho em bytes total do campo *Withdrawn routes*, que representa as redes que devem ser removidas da tabela de rotas. Internamente a este campo, existem os campos *Length* e *Prefix.* Nestes dois campos são representadas de fato as redes, através de instâncias de Length e Prefix. Um exemplo seria <16,143.54.0.0>, que representa a rede 143.54.0.0, dizendo que o endereço de rede é até o décimo sexto bit, ou seja, em notação CIDR representa a rede 143.54.0.0/16. Da mesma forma, através destas tuplas, são declaradas as redes a serem incluídas na tabela de rotas. Os campos utilizados para tal tarefa são os pertencentes ao grupo NLRI (*Network Layer Reachability Information*).

Através dos campos existentes no grupo *Path Attribute* é passado um conjunto de atributos necessários ao anúncio de uma rota, tais como: LOCAL_PREF, NEXT_HOP, ORIGIN, entre outros, que serão apresentados a seguir. Esses parâmetros são importantes para o processo do algoritmo de decisão do BGP para determinar os melhores caminhos.

Os dois campos existentes deste grupo, cuja definição é *Total path attribute length* e *Path attributes* representam respectivamente o tamanho em bytes e a declaração dos parâmetros. Esta declaração é feita através de uma estrutura definida como <tipo atributo, tamanho atributo, valor do atributo>. O campo *Path Attribute*, é por sua vez subdividido em diversas partes, explicadas na Figura 2-9.

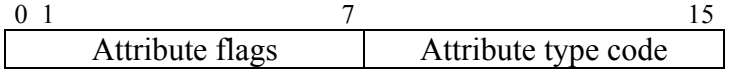

Figura 2-9 - Formato do campo *PathAttribute*

Os 8 bits de *Attribute flags* são divididos e utilizados conforme descrição na Tabela 2-5.

Tabela 2-5 - Descrição do subcampo *Atribute flags* do campo *Path Attribute*

Bit	**Utilização**
Primeiro bit – bit 0	Indica se o atributo é conhecido (0) ou opcional (1);
Segundo bit – bit1	Indica se o atributo é intransitivo (0) ou transitivo (1);
Terceiro bit – bit 2	Sendo o atributo opcional e transitivo, se este é completo (0) ou parcial (1);
Quarto bit – bit 3	Define se o atributo tem como tamanho 1 byte (0) ou 2 bytes (1)
Quinto ao sétimo bit – bit 4 a 7	Não utilizado, sendo sempre com conteúdo 0.

Em complemento, através da Tabela 2-6 são apresentados alguns termos que serão utilizados para categorizar os atributos do BGP que estão sendo apresentados.

Tabela 2-6 - Atributos BGP

Classificação	Significado
Conhecido obrigatório (Well-know mandatory)	Atributo que deve ser reconhecido em todas as implementações BGP de qualquer fabricante. Caso algum atributo deste tipo não esteja em uma mensagem UPDATE, será gerada uma mensagem tipo NOTIFICATION par reportar o erro.
Conhecido arbitrário (Well-known discretionary)	Atributo que deve ser reconhecido em todas as implementações de BGP, mas pode ou não estar presente em mensagens UPDATE. Um exemplo de atributo deste tipo é LOCAL_PREF.
Opcional e transitivo (Optional transitive)	Atributo que pode não ser reconhecido em todas as implementações. Sendo transitivo, significa que o atributo deve ser aceito e repassado aos demais *peers* BGP.
Opcional intransitivo (Optional nontrasitive)	Atributo também opcional. Sendo intransitivo, ele não é repassado a outros *peers* BGP.

Através da Tabela 2-7 são apresentados alguns tipos de atributos, juntamente com sua categoria e RFC que os descreve, segundo [CIS00]:

Tabela 2-7 - Relação de alguns atributos, classificação e RFC associada

Número do atributo	Nome do atributo	Categoria / Tipo	RFC / Internet Draft que o descreve
1	ORIGIN	Conhecido obrigatório, tipo 1	RFC 1771
2	AS_PATH	Conhecido obrigatório, tipo 2	RFC 1771
3	NEXT_HOP	Conhecido obrigatório, tipo 3	RFC 1771
4	MULTI_EXIT_DISC (MED)	Opcional intransitivo, tipo 4	RFC 1771
5	LOCAL_PREF	Conhecido arbitrário, tipo 5	RFC 1771
6	ATOMIC_AGGREGATE	Conhecido arbitrário, tipo 6	RFC 1771
7	AGGREGATOR	Opcional transitivo, tipo 7	RFC 1771
8	COMMUNITY	Opcional transitivo, tipo 8	RFC 1997
9	ORIGINATOR_ID	Opcional intransitivo, tipo 9	RFC 1966
10	Cluster List	Opcional intransitivo, tipo 10	RFC 1966

14	Multiprotocol Reachable NLRI	Opcional intransitive, tipo 14	RFC 2283
15	Multiprotocol Unreachable NLRI	Opcional intransitive, tipo 15	RFC 2283
16	Extended Communities		Draft-remachandra-bgp-ext-communities-00,txt "work in progress"
256	Reservada	Reservado para desenvolvimento	

Como foi descrito na Tabela 2-7, juntamente com os anúncios de rotas, são utilizados os chamados atributos que permitem determinar as melhores rotas do BGP. Na Tabela 2-8 são listados e explicados alguns dos atributos mais utilizados atualmente.

Tabela 2-8 - Descrição dos atributos BGP

Atributo	**Definição**
ORIGIN	Define a origem do anúncio, classificando-o em 3 grupos: 0 : IGP (i) - Significa que a origem do anúncio é interna ao referido AS. 1 : EGP (e) – Significa que a origem do anúncio é externa ao referido AS, aprendida via EGP. 2 : INCOMPLETE (?) – A origem do anúncio foi feita por algum mecanismo de redistribuição ou por meios desconhecidos. A ordem de preferência deste atributo é a própria ordem em que os grupos foram apresentados, visto que os anúncios do grupo 0 têm maior confiabilidade que os grupos 1 e 2.
AS_PATH	Atributo que representa a seqüência de ASs que uma rota segue para atingir determinado destino. Esses dados são incluídos na passagem ao anúncio em cada *peer*, que inclui seu número de AS juntamente ao anúncio da rota. Uma operação possível sobre este atributo é o chamado "prepend", que se caracteriza por "piorar" o AS_PATH para determinada rota, forçando com que outros caminhos possam ser escolhidos. Um exemplo de prepend seria transformar o AS_PATH 1916 4230 para 1916 1916 4230. O menor AS_PATH é escolhido.
NEXT_HOP	Refere-se ao IP do próximo roteador para atingir determinada rede. Geralmente o roteador que anuncia determinado prefixo repassa como *nexthop* o seu próprio IP, exceto em sessões iBGP ou em *route servers*, como será estudado adiante.
LOCAL_PREF	Usado para selecionar o caminho preferencial de saída a partir de um determinado AS. Seu escopo é interno ao AS, não sendo repassado a seus vizinhos (*peers*). Quanto mais alto seu valor, maior será a

	preferência.
COMMUNITY	Atributo que pode ser repassado a outros *peers*. Trata-se de uma espécie de "carimbo" que acompanha anúncios para que estes possam ser tratados de forma diferenciada.
WEIGHT	Atributo proprietário da CISCO, muito semelhante ao LOCA_PREF, embora não seja repassado para outros roteadores, mesmo dentro do mesmo AS. Quanto maior seu valor, maior será a preferência.

Através da tabela de rotas BGP representada pela Figura 2-10, são apresentados alguns dos atributos vistos anteriormente. Depois da versão e identificador do roteador, é apresentado o início da listagem de prefixos seus atributos. Cada linha a seguir representa um prefixo e seus respectivos atributos.

BGP table version is 1660291, local router ID is 200.10.20.30
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i12.0.48.0/20	198.32.252.254		100	0	11537 10578 1742 **i**
*>i12.6.208.0/20	198.32.252.254		100	0	11537 10578 1742 **i**
*>i12.6.252.0/24	198.32.252.254		100	0	11537 10578 14325 **?**
*>i12.16.126.192/26	198.32.252.254		100	0	11537 10578 14325 **?**
*>i12.144.59.0/24	198.32.252.254		100	0	11537 10466 13778 **i**

Figura 2-10 - Tabela BGP e atributos

A partir do início de cada linha, o símbolo * (asterisco) apresentado ao lado dos prefixos mostra que estes estão definidos como melhores caminhos para as redes em questão. Ao lado é apresentada a rede anunciada na forma de bloco CIDR, em seguida o *Next Hop* que define-se como próximo roteador que os pacotes para esta rede deverão ser enviados. Também todas as rotas possuem o atributo *Local Preference* com valor 100, o atributo *Weight* (proprietário da Cisco) com valor 0. Outro atributo é o AS_PATH que mostra a seqüência de sistemas autônomos até a chegada a rede destino e por fim, o atributo ORIGIN, que define a procedência do anúncio pelo AS que anunciou e é representado por i (IGP), e (EGP) ou "?" (indefinido ou incompleto).

2.4.6.4.1 CIDR

O recurso de *Classless Interdomain Routing* refere-se a evolução do conceito de divisão de faixas de IPs em classes conhecidas como A, B, C, D ou E, sendo que as duas últimas são utilizadas para outros fins. Isso surgiu frente ao problema de alocação e falta de flexibilidade na utilização de faixas de IPs pré-definidas em 3 classes principais. A novidade neste conceito é que a faixa de IPs pode ser dividida de acordo com as necessidades de cada rede, não devendo mais obedecer às classes definidas

como A, B ou C. Dessa forma, uma rede é anunciada com sua faixa, seguida pelo número de bits que são utilizados para a representação do endereço de rede. Os demais bits da direita a partir deste poderão ser utilizados para endereçamento de *hosts* ou de subredes. Um exemplo desta representação seria 143.54.0.0/16 que é equivalente ao classe B 143.54.0.0 cuja máscara é 255.255.0.0. Esta representação mostra que até o décimo sexto bit (da esquerda para direita) é feita a representação do endereço de rede e a partir do décimo sétimo bit são representados os endereços dos *hosts* desta rede. Outro exemplo desta notação, seria representar os endereços de classe C 192.32.136.0 (192.32.136.0/24) e 200.32.137.0 (200.32.137.0/24), como pode ser visto na Figura 2-11 em seus dois primeiros quadros.

11000000	00100000	10001000	00000000	192.32.136.0
11111111	11111111	11111111	00000000	255.255.255.0

11000000	00100000	10001001	00000000	192.32.137.0
11111111	11111111	11111111	00000000	255.255.255.0

11000000	00100000	10001000	00000000	192.32.136.0
11111111	11111111	1111111**0**	00000000	255.255.254.0

Figura 2-11 - CIDR

Para representar cada endereço de classe C a notação antiga seria suficiente. O CIDR acaba sento utilizado caso desejarmos representar redes que não teriam classe A, B ou C. Uma situação desse tipo seria a de incluir em um mesmo bloco os dois classe C declarados acima, resultando em um bloco que não seria mais considerado fora das classes A, B ou C. No terceiro quadro pode-se verificar a junção dos dois blocos e a máscara resultante dessa junção. Na máscara, como está destacado foi utilizado um bit a menos para representação da máscara, englobando com isso os dois blocos em apenas um endereço e uma máscara. A junção resultante é: 192.32.136.0/23.

2.4.7 Utilização de BGP em sistemas autônomos

O protocolo BGP é utilizado para propagar rotas entre sistemas autônomos. Esse é seu principal propósito, mas diante de grandes sistemas autônomos, os IGPs nem sempre conseguem sustentar o roteamento interno e seu crescimento. Nessas situações, o BGP também pode ser usado dentro de um mesmo AS. Supondo que determinado AS tenha apenas um de seus roteadores com sessões BGP com outros ASs, outros roteadores deste mesmo sistema autônomo podem receber os anúncios do roteador de borda via BGP e não exclusivamente por algum protocolo tipo IGP. Nesses casos existe o iBGP, que se trata de uma extensão do protocolo BGP para ser utilizado entre roteadores de um mesmo AS. Como justificativa para o surgimento do iBGP, pode-se dizer que os protocolos tipo IGP não são escalonáveis a ponto de fazerem a comunicação entre roteadores em grandes backbones de forma satisfatória. Por isso, utiliza-se iBGP por permitir uma escalabilidade mais poderosa e também pelas funcionalidades que o protocolo fornece muito superiores aos do tipo IGP. Um exemplo de utilização de iBGP e eBGP é mostrado na Figura 2-12.

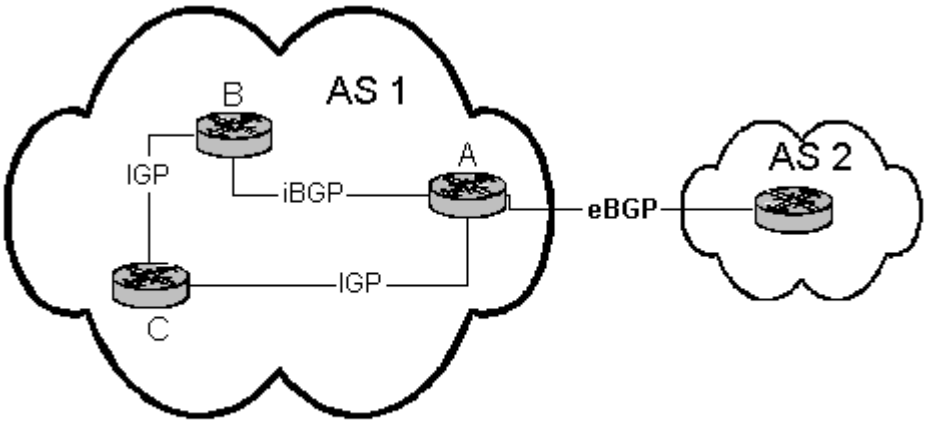

Figura 2-12 - Exemplo de utilização de iBGP e eBGP

De acordo com a figura, o roteador A é o *border router* e responsável pela comunicação com o mundo externo, que no caso é feita com o AS 2. O roteador A mantém com B e C uma sessão iBGP. Os roteadores B e C poderiam trocar informações via IGP com A, utilizando OSPF por exemplo, mas em grandes *backbones* isso não seria escalonável. Por isso, é utilizado iBGP para esta comunicação. Isso fica mais evidente em *backbones* em nível nacional, como é o caso da RNP que utiliza iBGP em seu AS em conjunto com OSPF.

2.4.8 O Futuro do BGP

A fim de acompanhar o crescimento da Internet e para atender às novas tecnologias como *Multicast* e o surgimento do Ipv6, foram propostas também funcionalidades adicionais ao protocolo BGP. Segundo [CIS00] essas funcionalidades formam o MBGP (*Multiprotocol BGP*), que surge com o intuito de permitir a troca de informações de roteamento por múltiplos protocolos em nível de rede. Tais protocolos são identificados por uma família de endereço (*Address Family*), como definido na RFC 1700. Algumas famílias de endereços que poderiam ser citadas seriam: IPv4, IPv6, IPX. Também o MBGP permite sub-famílias, a citar *unicast* e *multicast* por exemplo.

Outro recurso que provavelmente no futuro será mais utilizado será a assinatura das mensagens de sessões BGP, pois tem sido registrados alguns casos de ataques de hackers a roteadores, enviando pacotes falsificados, tentando incluir anúncios a roteadores com BGP. Neste caso, a utilização do método de assinatura TCP MD5 já está prevista, segundo a RFC 2385, utilizando para tanto o campo MARKER do *header* do pacote BGP.

3. ORGANIZAÇÃO DA INTERNET EM PTT´S

Desde os primórdios da Internet, quando apenas alguns centros de pesquisa dos EUA foram interconectados, a fim de compartilhar informações acadêmicas, não se imaginou que a grande rede mundial de computadores, a Internet, chegasse ao ponto que se encontra hoje. O crescimento apresentado assemelha uma curva exponencial. Com esse crescimento tão rápido, a Internet foi deixando de ser uma única rede homogênea para se tornar uma grande rede heterogênea, formada por diversos *backbones* pertencentes a empresas que inicialmente forneciam serviços de telefonia.

Assim, a fim de garantir a ligação entre todos os pontos, independente de qual *backbone* estivessem conectados, foi necessário criar pontos de conexão entre esses *backbones* para não isolá-los dos demais. Com os *backbones* ocupando regiões nacionais e até internacionais, a necessidade de conexão com outros *backbones* foi sendo mais evidente. A alcançabilidade a redes de outros *backbones* estava garantida, mas características como tempo de acesso foram se tornando importantíssimas, pois os serviços de videoconferência e aplicações de tempo real estão cada vez mais presentes na rede.

O problema encontrado nesta altura é que pela falta de cooperação entre *backbones* e em muitos casos, por interesse particular de algum dos *backbones*, os pontos de interconexão eram distantes dos pontos ótimos e o número destes era muito pequeno. Um exemplo deste problema é mostrado na Figura 3-1, e de certa forma existe até os dias de hoje nas redes acadêmicas.

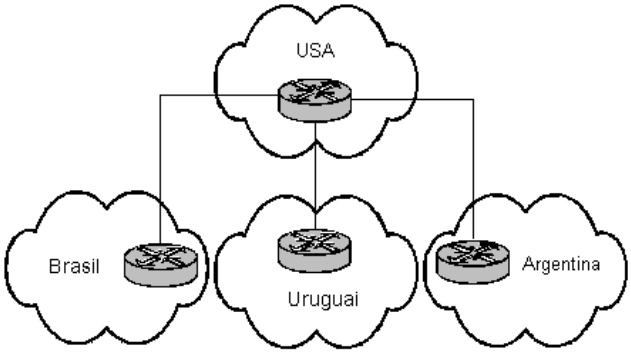

Figura 3-1 - Situação das Conexões entre Brasil, Uruguai e Argentina

Nesta figura, é descrita a situação de interconexão entre Brasil, Uruguai e Argentina. Esses três países da América Latina, mesmo sendo vizinhos, trocam tráfego entre seus backbones apenas nos Estados Unidos. Isso claramente exige maiores *links* até os Estados Unidos, gerando mais gastos e um tempo de acesso muito mais alto que o ideal.

Diante de situações como a apresentada na Figura 3-1, algo deveria ser feito para melhorar principalmente o tempo de resposta e diminuir os gastos. Assim, surgiu o termo *Network Access Point* (NAP) ou Ponto de Troca de Tráfego (PTT). Muito resumidamente, os NAPs ou PTTs são pontos onde diferentes *backbones*

estabelecem conexões locais com o intuito de trocar tráfego e conseguir fornecer um serviço de maior qualidade, mais econômico e com tempo de acesso mais baixo.

3.1.1 Conceitos Gerais

Fisicamente um PTT é formado por um conjunto de roteadores localizados em um único ponto neutro, formando uma rede local de alta velocidade, geralmente com tecnologias de nível 2 como *Fast Ethernet* ou *Gigabit Ethernet*. Cada roteador representa um sistema autônomo que deseja trocar tráfego com pelo menos um dos participantes. Para a conexão entre todos os roteadores, existe um comutador ou *switch* com alto poder de processamento.

Cada PTT possui suas próprias regras de funcionamento, e geralmente possui uma equipe de suporte que administra o PTT, com plantão 24x7 (24 horas, 7 dias por semana). Também é exigido que o participante seja um sistema autônomo, tendo um número de AS próprio e um bloco CIDR. Esses recursos são os mínimos para que este possa participar de um PTT utilizando o protocolo BGP. A cada participante é fornecido um espaço em rack para colocação do roteador, bem como o endereçamento da rede local do PTT e a forma de troca de rotas com os demais participantes. Esta comunicação com os demais participantes pode ser feita de diferentes formas, como veremos na seção Arquitetura de PTTs. Cada roteador e sua respectiva configuração fica de responsabilidade do próprio AS participante. Isso é feito dessa forma, pois assim cada participante pode configurar seu roteador de acordo com as políticas desejadas, sem que outros participantes ou a própria equipe do PTT possam alterar tais configurações. A conexão até o PTT fica a cargo do próprio participante.

Os PTTs podem ser públicos ou privados e sua diferença é que nos públicos qualquer sistema autônomo pode participar e nos privados a inclusão deve ser aprovada por quem sedia o espaço, além de acordos que podem ser exigidos.

3.1.2 Arquiteturas de PTTs

Existem basicamente dois tipos de Pontos de Troca de Tráfego. Cada um deles apresenta pontos positivos e negativos, privilegiando de forma mais ou menos satisfatória aspectos como gerência, centralização de anúncio de rotas, segurança e o nível de intervenção da equipe de administração de um PTT. Independente do tipo de PTT, o que é comum é que os participantes devem ser sistemas autônomos e devem ter um roteador que possa suportar o protocolo BGP. Os roteadores de cada participante são interligados por um *Switch* com portas de alta velocidade *Ethernet* ou *Gigabit Ethernet*, que forma uma rede local onde cada participante recebe um IP para endereçar sua interface com o PTT. Na maioria dos casos é utilizado endereçamento válido, geralmente um bloco classe C (em notação CIDR representado por /24).

O primeiro que será apresentado possui uma estrutura que é apresentada na Figura 3-2.

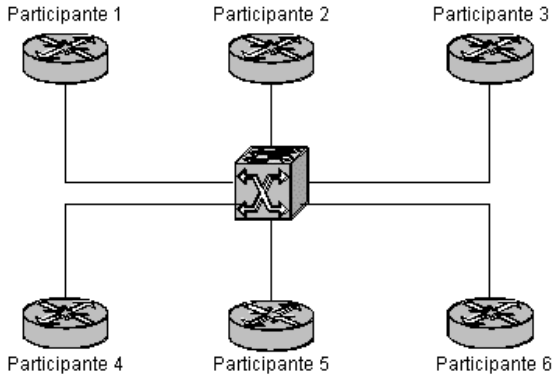

Figura 3-2 - Exemplo de PTT baseado em acordos bilaterais.

Neste tipo de PTT todos os participantes devem simplesmente estar na rede do PTT. Os acordos de troca de tráfego neste tipo de PTT são chamados bilaterais, ou seja, se determinado participante desejar trocar tráfego com um conjunto de ASs, este deverá estabelecer uma conexão BGP com cada um destes participantes diretamente. Neste caso, na entrada de um novo participante no PTT, se este desejar trocar tráfego com N participantes, N novas sessões BGP serão estabelecidas adicionalmente no PTT. Nesses casos não é necessária intervenção da equipe de suporte do PTT para estabelecer os acordos de *peering*, já que basta o contato entre os sistemas autônomos envolvidos no acordo. No protocolo BGP, podemos representar este tipo de PTT através da Figura 3-3, que representa alguns acordos bilaterais, tendo estabelecidas sessões BGP em linhas "hachuradas". Na Figura 3-3, existem acordos de troca de tráfego entre os ASs 1-5, 4-3 e 3-6;

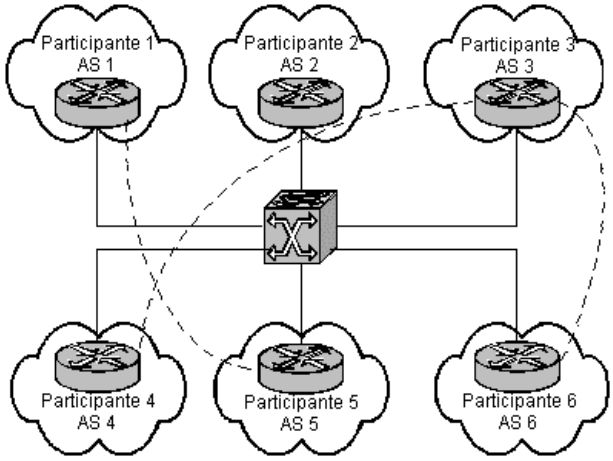

Figura 3-3 - Exemplo de acordos bilaterais em PTT.

Outra arquitetura de PTT muito comum é baseada em *Route Servers*, sendo ilustrada na Figura 3-4.

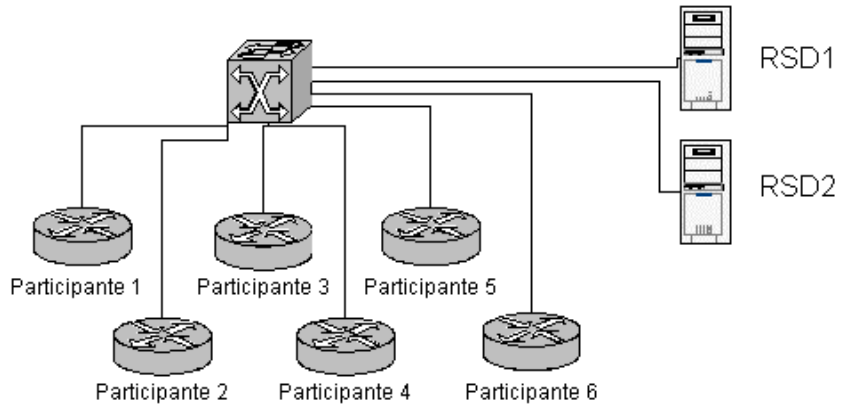

Figura 3-4 - Ilustração de PTT baseado em acordos multilaterais.

Neste caso, além dos roteadores de cada participante, são utilizadas estações de trabalho com *software* que implementam o protocolo BGP e suas operações. Sendo assim, os dois computadores trabalham como centralizadores das rotas anunciadas pelos participantes do PTT. Todos os participantes não mais estabelecem sessões entre eles, mas estabelecem sessões BGP com os dois *Route Servers*. Neste caso, são estabelecidos acordos de tráfego multilaterais (ATMs).

Na entrada de um novo participante no PTT, supondo que este tenha interesse em trocar tráfego com todos os demais participantes do PTT, serão necessárias apenas duas novas sessões BGP entre o novo participante e cada *Route Server*. A partir daí, sabendo que o *Route Server* trabalha como centralizador/distribuidor de rotas a todos participantes, o estabelecimento de duas sessões (uma para cada *Route Server*) já garante o recebimento dos anúncios dos demais participantes, evitando estabelecer sessões com cada um participante.

Dessa forma, a equipe de suporte do PTT deve ser acionada para estabelecer as novas sessões com os *Route Servers*, mas em compensação os demais participantes nada terão que fazer para iniciar a troca de tráfego com esse novo *peer*. Um ponto importante é que, sendo o *Route Server* centralizador dos anúncios, este torna-se um ponto único de falha, podendo comprometer todo o PTT, caso apresente alguma falha. Neste caso, adota-se a política de utilizar dois *Route Servers*, garantindo que na falha de um deles o outro possa manter o serviço sem interrupções. Também é aconselhável que sejam utilizados *softwares* diferentes em cada *Route Server*, evitando que uma possível falha provoque a interrupção de ambos.

Os *softwares* utilizados para os *Route Servers* são diversos. Entre eles, citamos o Gated [GAT01] e Zebra [ZEB01]. Estes dois *softwares* implementam, além do protocolo BGP, diversos protocolos de roteamento tais como RIP, OSPF, RIPv6, OSPFv6, entre outros. O *software* Gated é comercial, tendo disponível de forma livre apenas algumas versões mais antigas e o *software* Zebra é livre, podendo ser utilizado livremente sem o pagamento de licenças. Ambos são implementados para sistemas UNIX.

O funcionamento de um PTT baseado em *Route Server* (acordos de tráfego multilaterais) é um pouco mais complexo que o primeiro modelo apresentado (acordos de tráfego bilaterais), pois existem alguns detalhes importantes no seu funcionamento. No caso de PTTs baseados em acordos de tráfego bilaterais, a troca de anúncios é feita diretamente entre as partes envolvidas. Isso significa que a sessão BGP é estabelecida diretamente entre as interfaces que cada participante tem no PTT, no caso a interface da rede local de cada um. Dessa forma, os anúncios chegam ao destino com os mesmos valores que o AS que originou tais anúncios criou, sem nenhuma alteração no caminho. No caso de PTTs baseados em *Route Server*, cada participante estabelece sessões com os *Route Servers*, que por sua vez divulgam as rotas para os demais participantes. Vale lembrar aqui que os dois *Route Servers* estão logicamente em um AS diferente dos demais participantes. Este número geralmente é na faixa de número de AS privados, formando um "AS imaginário", que serve apenas para permitir que os *Route Servers* possam estabelecer sessões eBGP com os participantes. Nesse ponto surge um detalhe muito importante, onde a Figura 3-5 nos ajuda a demonstrar:

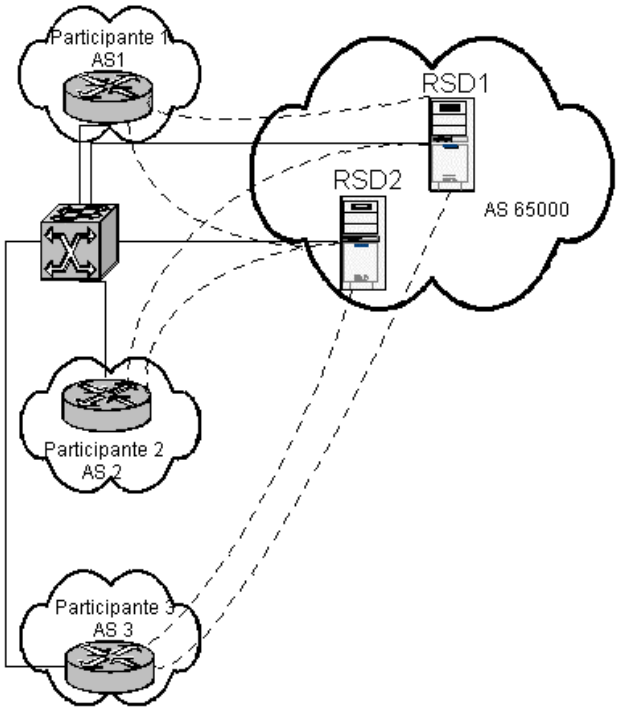

Figura 3-5 - Exemplo de PTT baseado em *Route Server*.

O detalhe importante a ser citado refere-se ao comportamento normal do protocolo BGP que é receber anúncios de rotas - neste caso de outros ASs - aplicar o algoritmo de escolha do melhor caminho e por fim divulgar a seus *peers*. Na divulgação, o atributo NEXT_HOP é alterado para a interface de quem está anunciando e no atributo AS_PATH é incluído o AS de quem está anunciando. Sendo assim, como todos os anúncios passam obrigatoriamente pelos *Route Servers* (AS 65000), todos os anúncios chegariam aos demais participantes do PTT com o atributo NEXT_HOP

definido para a interface do *Route Server* e o AS_PATH incluído do AS dos *Route Servers* (AS 65000). Dessa forma, se determinado AS enviar um pacote para uma rede anunciada por um outro participante, o pacote passará pelo *Route Server*, que foi quem anunciou a rota, incluindo a sua interface como NEXT_HOP e em seguida será repassado do roteador do participante destino. Fazendo uma análise sobre isso, pode-se concluir que esta abordagem funciona, mas poderia ser melhorada, fazendo com que os pacotes do tráfego do PTT propriamente ditos não passassem por um dos *Route Servers*.

Essa observação se torna uma necessidade, visto que a interface dos *Route Servers* será o "gargalo" do tráfego geral do PTT. Como em PTTs é normal alcançar altas taxas de tráfego (600 Mbps no PTT da ANSP por exemplo), torna-se quase impossível suportar esse tráfego por limitações de hardware dos *route servers*. Para contornar este problema, na configuração dos *Route Servers* é feita uma alteração da forma que nos anúncios recebidos não sejam alterados os atributos de NEXT HOP ou AS_PATH. Esse ajuste que parece simples faz com que o tráfego passe diretamente de um roteador para outro, passando apenas pelo *switch*. Assim, o tráfego que segue para os *Route Servers* será apenas do protocolo BGP. Pode-se também visualizar a figura anterior para demonstrar o fluxo obtido. Maiores detalhes sobre tal prática serão abordados nas próximas seções.

Um outro componente muito utilizado em PTTs deste tipo é o chamado *looking glass*, apresentado na Figura 3-6.

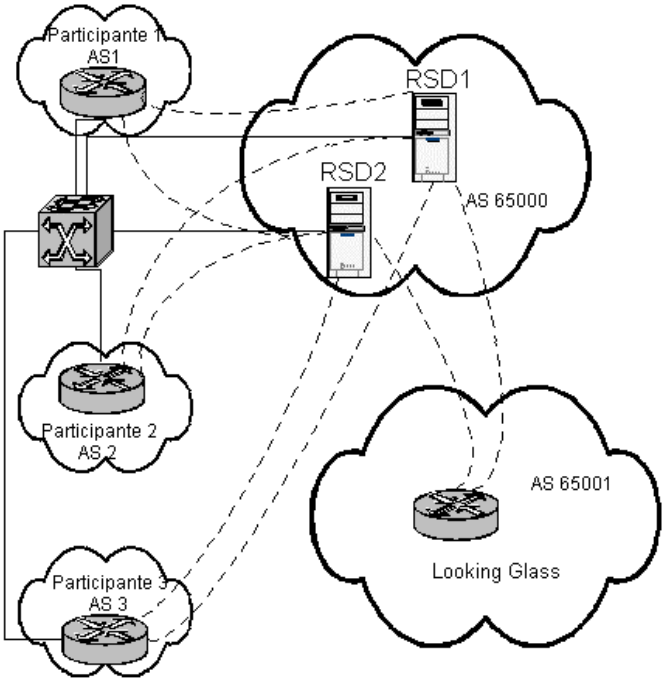

Figura 3-6 - PTT baseado em *Route Servers*, utilizando *Looking Glass*.

O componente *looking glass* trata-se de um roteador ou um PC que possui suporte a BGP, conectado aos *Route Servers*. Este componente utiliza um número de AS também privado, por exemplo 65001. O número de AS poderia ser qualquer um da faixa privada, desde que seja diferente do AS formado pelos *Route*

Servers. A utilidade de tal recurso é que este, uma vez conectado aos *Route Servers*, recebe todos os anúncios dos participantes do PTT e pode ser usado para consulta das rotas anunciadas no PTT e a preferência que cada uma destas tem sob as demais anunciadas. Isso acaba sendo útil aos participantes do PTT para verificar se seus anúncios estão sendo feitos corretamente e se o efeito está de acordo com o esperado, sem contar que a mesma informação pode ser consultada por pessoas de outros sistemas autônomos para verificar a visibilidade de determinado anúncio ou ainda, para verificar quais os anúncios existentes no PTT e usar tais informações como motivação para participação no PTT.

Por fim, mais um recurso utilizado em PTTs é a aplicação de um RA, conhecido como *Router Arbiter* [RAD98]. Esta ferramenta serve para certificar as rotas que cada sistema autônomo pode exportar e importar de um *peer*. Isso é feito mediante o registro de cada AS em uma base de dados, declarando quais prefixos pode aceitar e quais pode enviar. Dessa forma, os *Route Servers* podem ser configurados para que somente sejam repassados prefixos a outros *peers* em um PTT, caso as características dos prefixos estejam de acordo com a base de dados no RA. No exterior, essa funcionalidade é muito utilizada, tornando os PTTs baseados em *Route Servers* mais restritivos em seus anúncios. No Brasil, tal prática ainda não vem sendo utilizada.

3.1.3 Ferramentas Utilizadas

As ferramentas utilizadas em PTTs são basicamente *softwares* que implementam o protocolo BGP e tem por objetivo manter o funcionamento e divulgação de rotas de cada participante. Também são utilizadas ferramentas presentes em sistemas operacionais *Unix-like* e Windows, como o *traceroute*, utilizado para a verificação de anúncios a redes de participantes do PTT e seus clientes. Abaixo são apresentadas algumas características de cada uma destas ferramentas:

3.1.3.1 Software Zebra

O software Zebra é uma implementação *freeware* destinada a sistemas operacionais Unix como FreeBSD, Linux, NetBSD e OpenBSD. Suporta diversos protocolos de roteamento como RIP, OSPF, BGP e alguns destes para IP versão 6 também como RIP6 e OSPF6. No caso do BGP, o protocolo versão 6 também é suportado, mas isso é feito pelo mesmo *daemon* que suporta IP versão 4. Para cada protocolo destes apresentados, a ferramenta possui um *daemon* que pode ser ativado e configurado. Adicionalmente existe um arquivo de configuração independente para cada protocolo. Também existe um *daemon* chamado zebra que permite configuração global da ferramenta e as possíveis distribuições das informações entre um protocolo e outro. Os comandos utilizados por cada protocolo são semelhantes aos comandos de roteadores Cisco. Um recurso importante presente nesta ferramenta é o acesso via telnet a cada um dos *daemons* que estiverem ativos, semelhante a um terminal virtual. As portas associadas a cada *daemon* são exibida na Tabela 3-1.

Tabela 3-1 – Portas Zebra

Porta	Protocolo associado
2601	Zebra
2602	RIP
2603	RIP (IP versão 6)
2604	OSPF
2605	BGP (IP versão 4 e versão 6)
2606	OSPF (IP versão 6)

A partir destes terminais virtuais a configuração de cada protocolo pode ser alterada, podendo ser salva no arquivo de configuração correspondente. No caso da configuração do protocolo BGP, cujo *daemon* é o bgpd, um exemplo básico de possível configuração deste protocolo é mostrado na Figura 3-7.

```
hostname exemplo
password zebra
enable password bgpsecret
!
router bgp 65000
 bgp router-id 10.0.0.1
 network 192.168.10.0/24
 neighbor 10.0.0.2 remote-as 65001
!
log file bgpd.log
!
log stdout
```

Figura 3-7 – Configuração BGP no Zebra

A configuração exibida na Figura 3-7 trata de uma sessão BGP entre o roteador cujo router-id é 10.0.01 e o roteador 10.0.0.2. O AS que cada um pertence é respectivamente 65000 e 65001. A rede anunciada pelo AS 65000 é, neste caso, 192.168.10.0/24. Outras coisas que são configuradas neste *daemon* são as senhas de primeiro e segundo nível. As senhas de primeiro nível dão acesso a consultas de sessões BGP e dados das interfaces e as senhas de segundo nível dão acesso à configuração completa do sistema com permissões para alteração.

3.1.3.2 Software Gated

O Gated é outro software destinado a função de roteamento. Esta ferramenta é uma implementação comercial, tendo apenas versões antigas disponíveis para utilização livre. Na versão comercial são implementados todos os protocolos do software Zebra, além de protocolos *Multicast* como PIM, DVMRP e MSDP, além do

IS-IS. A versão que pôde ser testada foi mais antiga e mais restrita. Tal versão implementa apenas o protocolo BGP. A implementação é feita em um único *daemon* que, em sua inicialização, lê as configurações de um arquivo específico. Ao contrário do zebra, que pode ter suas configurações alteradas sem a interrupção do *daemon*, esta versão exige que o *daemon* seja reinicializado depois de cada modificação de configuração. A Figura 3-8 apresenta um exemplo de configuração de sessão BGP.

```
traceoptions nostamp normal route parse adv;
autonomoussystem 65000;
routerid 10.0.0.1;

bgp on {
traceoptions packets open update keepalive;
preference 100;

group type external peeras 65001 {
peer 10.0.0.2 holdtime 180 transparent;
};
};

view {
peer 10.0.0.2  preference 100;
import proto bgp as 65001 {
all;
};
};
```

Figura 3-8 – Configuração BGP no Gated

A configuração da figura anterior foi baseada na mesma situação abordada pela configuração da ferramenta Zebra.

3.1.3.3 Traceroute e Ping

Ferramentas como *traceroute* e *ping* são facilmente encontradas em sistemas operacionais *UNIX-like* e Windows. Ao contrário de ferramentas como Zebra e Gated, o *traceroute* e *ping* são utilizados para testes de alcançabilidade de rotas entre os participantes de um PTT. Com o *ping* é possível verificar os IPs de *hosts* ou roteadores ativos em determinada rede e o comando *traceroute* pode exibir o caminho percorrido para chegar em determinada rede. Um exemplo de uso de *traceroute* é mostrado na Figura 3-9. Nesta figura é mostrado o caminho entre um *host* do POP-RS e o *site* da Brasil Telecom (www.crt.net.br) que atualmente encontra-se conectada no RSIX.

```
[andrey@plutao]$ traceroute www.crt.net.br
traceroute to www.crt.net.br (200.181.14.22), 30 hops max, 38 byte packets
 1  bb3.pop-rs.rnp.br (200.132.0.17)  0.440 ms  0.387 ms  0.394 ms
 2  brasiltelecom.rsix.tche.br (200.132.1.7)  2.785 ms  1.996 ms  1.582 ms
 3  BrT-f2-0-0-paemt300.telebrasilia.net.br (200.180.143.49)  1.744 ms  3.316 ms  1.475 ms
```

```
 4  BrT-g3-1-paemtcore.telebrasilia.net.br (200.180.143.33)  2.235 ms  1.816 ms  3.801 ms
 5  BrT-a0-3-bsacecore.telebrasilia.net.br (200.180.143.6)  38.586 ms  61.348 ms  78.270 ms
 6  200.199.193.129 (200.199.193.129)  81.886 ms  38.346 ms  39.814 ms
 7  200.199.193.234 (200.199.193.234)  53.771 ms  62.679 ms  54.577 ms
 8  bd1.brturbo.com (200.199.201.3)  52.862 ms  48.311 ms  44.136 ms
 9  172.61.61.5 (172.61.61.5)  59.081 ms  63.648 ms  63.603 ms
10  172.61.61.5 (172.61.61.5)  69.546 ms  55.827 ms 60.436 ms
```

Figura 3-9 - *Traceroute*

3.1.4 PTTs no Brasil e a RNP

Em nível nacional, a primeira iniciativa de PTT foi feita pela ANSP [ANS01] em 1998 e atualmente conta com 32 participantes. Em 2000, foi criado o OPTix-LA (OptiGlobe *Internet eXchange - Latin American*) que é o PTT da empresa Optiglobe [OPT01], que trata-se de um PTT pertencente a uma empresa privada. Ambos PTTs tem sede geográfica em São Paulo, visto que, nesta região existe uma grande concentração de provedores de *backbones* e recursos para o estabelecimento de tais acordos de troca de tráfego. No mesmo ano, foi criado um PTT com sede em Porto Alegre. Este PTT é o RSIX [RSI01] que é um PTT com sede no Centro de Processamento de Dados da UFRGS, sob a administração da equipe do POP-RS / RNP.

O surgimento de PTTs no Brasil tem se tornado uma tendência forte por sua importância e seus benefícios. Um documento, que reforça essa tendência e fornece informações sobre a operação e administração de PTTs, foi elaborado em Janeiro/2000 pelo Grupo de Trabalho em Engenharia de Redes do Comitê Gestor Internet/BR [REC00]. Este documento além de levantar informações pertinentes a administração e operação de pontos de troca de tráfego, também sugere a criação de PTTs nas principais cidades brasileiras, como: Rio de Janeiro, Brasília, Belo Horizonte e Porto Alegre.

A situação atual entre o *backbone* da Rede Nacional de Pesquisa (RNP) e os PTTs existentes no Brasil é apresentada na Figura 3-10.

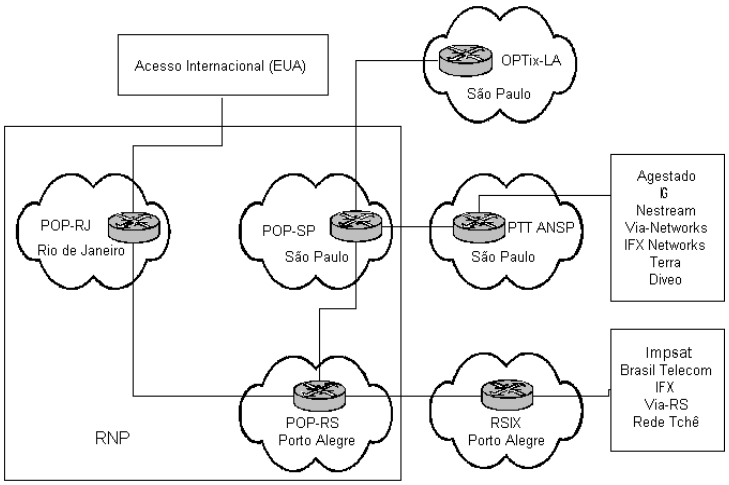

Figura 3-10 - Conexões entre os backbones da RNP e os PTTs nacionais existentes.

Nesta ilustração, o *backbone* da RNP é representado pelos Pontos de Presença dos estados do Rio Grande do Sul, São Paulo e Rio de Janeiro. Na realidade, cada estado brasileiro possui um Ponto de Presença (POP), mas estes três POPs apresentam as principais conexões para comunicação com outros *backbones* através de pontos de troca de tráfego. Também é possível verificar que a conexão aos 3 PTTs no Brasil melhorou muito a conectividade da RNP com outros *backbones* comerciais. Ao lado dos PTTs RSIX e ANSP é apresentada uma breve listagem de alguns participantes que fazem troca de tráfego juntamente com a RNP. No caso do OPTix-LA, não são listados os participantes pois essa informação não foi veiculada no momento.

3.1.5 PTT RSIX (RS *Internet Exchange*)

O RSIX iniciou suas operações no ano de 2000, tendo a participação inicial dos sistemas autônomos da UFRGS, POP-RS, Rede Tchê e RNP. Posteriormente, foram convidados outros sistemas autônomos com ponto de presença no estado, estendendo o convite a outros possíveis interessados. A sede deste PTT público é no próprio Centro de Processamento de Dados da UFRGS.

Em termos técnicos, o RSIX é um PTT baseado na utilização de *Route Servers*, tendo uma rede local utilizando um *Switch* de alta capacidade de processamento. Dessa forma, podem ser estendidos todos os conceitos e ilustrações apresentados anteriormente sobre PTTs desta natureza. Também é utilizado um roteador como *Looking Glass*.

A topologia do RSIX é basicamente a mesma que foi abordada nos exemplos de PTTs com *Route Servers* e *Looking Glass*. O sistema operacional utilizado nos *Route Servers* é FreeBSD, sendo utilizados dois PCs para tal tarefa, visto que o poder de processamento para estes componentes não é alto. Em um dos *Route Servers* foi instalada a ferramenta Zebra e na outra, Gated. Ambos *softwares* foram instalados para evitar que um mesmo *bug* pudesse comprometer os dois *Route Servers* e por conseqüência, comprometer o funcionamento do PTT.

Atualmente o RSIX conta com 9 sistemas autônomos. Maiores informações sobre este PTT podem ser encontradas em [RSI01].

4. GERÊNCIA DE REDES

O gerenciamento tem como objetivo auxiliar os procedimentos de controle e monitoração em uma rede através do acesso às informações que refletem o comportamento e a configuração de um dispositivo, seja ele *software* ou *hardware* [STA 99]. O gerenciamento da rede é muito importante, visto que hoje em dia apresenta maior complexidade, equipamentos heterogêneos e uma maior quantidade de serviços. Devido à necessidade de automatizar o gerenciamento, surgiu a aplicação e os protocolos de gerenciamento.

Com o crescimento da Internet e a complexidade de administrá-la, a *Internet Architecture Board* (IAB) estudou três propostas para a gerência de redes:

- High-Level Entity Management System (HEMS);
- Simple Network Management Protocol (SNMP);
- CMIP sobre TCP/IP (CMOT).

O HEMS é uma generalização do primeiro protocolo de gerência usado na Internet, o *Host Monitoring Protocol* (HMP). O SNMP é uma versão melhorada do *Simple Gateway Management Protocol* (SGMP). E o CMOT é uma versão do protocolo usado no modelo OSI, o CMIP sobre a pilha TCP/IP.

A IAB optou por investir no SNMP como uma solução a curto prazo e no CMOT como uma solução a longo prazo. Isso foi definido, pois se pensava que o modelo OSI iria substituir a arquitetura TCP/IP, então investindo em protocolos que usam as camadas mais acima iria facilitar a transição. Como a adoção do modelo OSI nunca foi feita, e a maior complexidade que CMOT tem em relação a SNMP, o protocolo que é usado atualmente na indústria é o SNMP. A arquitetura de gerenciamento é apresentada na Figura 4-1.

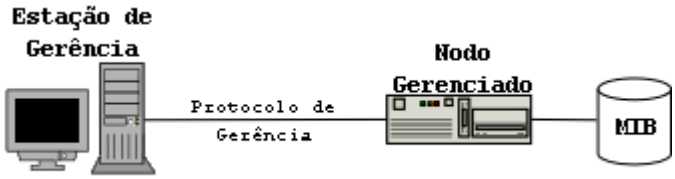

Figura 4-1 Arquitetura de Gerência

A estação de gerência é o gerente da arquitetura. Nela o administrador da rede irá interagir com os nodos gerenciados. O nodo gerenciado, por sua vez, possui um agente, este tem a capacidade de responder as requisições do gerente e coletar os objetos gerenciados do nodo e gravar na MIB.

Existem três versões do protocolo SNMP, as principais RFC que as definem são:

- RFC 1155: define a SMI (Structure of Management Information).
- RFC 1156: define a MIB-I.
- RFC 1213: define a MIB-II.
- RFC 1157: define a primeira versão de SNMP.

- RFC 1901 – 1908: definem a segunda versão de SNMP.
- RFC 2271 – 2275: definem a terceira versão de SNMP.
- RFC 1513, 1757, 2021, 2074: definem a RMON.

4.1 Structure of Management Information

As informações de gerenciamento são gravadas numa base de dados chamada MIB (*Management Information Base*), essa base contém objetos referentes à gerência de redes. Para isso, criou-se uma especificação chamada de SMI (*Structure of Management Information*) que define regras que descrevem as informações de gerenciamento de tal maneira que seja independente de detalhes de implementação [ROS 96].

A SMI prove uma especificação para definir a estrutura da MIB, a sintaxe de objetos e técnicas para codificar esses valores. Para a sintaxe de dados é definido a ASN.1 (*Abstract Syntax Notation One*) e para a transferência de dados a BER (*Basic Encondig Rules*).

4.1.1 Abstract Syntax Notation One

A ASN.1 é uma linguagem formal desenvolvida pela CCITT (X.208) e ISO (ISO 8824). Ela define uma sintaxe para definir dados que é independente de implementação. No modelo SNMP, a ASN.1 é usada para definir a estrutura da aplicação e as PDU (*Protocol Data Unit*).

4.1.1.1 Tipos de Dados Abstratos

Um tipo pode ser visto como uma coleção de dados. Em ASN.1 podemos separar os tipos de dados em 3 categorias: Simples (Primitivos), Estruturados e Definidos.

O tipo Simples define: BOOLEAN, INTEGER, BIT STRING, OCTET STRING, REAL, ENUMERATED, OBJECT IDENTIFIER e NULL. Todos os outros tipos são construídos a partir deles. O tipo Estruturado serve para definir tabelas e listas. Para isso são usados quatro tipos: SEQUENCE, SEQUENCE OF, SET e SET OF. O tipo de dados Definido representa uma derivação dos tipos anteriormente citados.

4.1.1.2 Módulos

A ASN.1 é uma linguagem usada para definir estruturas de dados. Essa estrutura é chamada de módulo. A partir do nome do módulo, podemos usar essa estrutura nos mais diversos documentos. Um módulo tem a seguinte sintaxe:

```
<modulereference> DEFINITIONS ::=
   BEGIN
      EXPORTS
```

```
    IMPORTS
    AssignmentList
End
```

O *modulerereference* é usado para identificar o módulo, um nome. O *exports* indica quais definições podem ser importadas para outros módulos e o *imports* quais outros módulos são usados por este. Os módulos são então como uma identificação. A MIB-I, MIB-II e RMON possuem um módulo que a representam que serão explicados nas seções adiante.

4.1.1.3 Macros

As macros permitem que a ASN.1 seja estendida para definir novos tipos e seus valores [STA 99]. Em SNMP é usada para definir os objetos gerenciados, por exemplo o *sysDescr*.

```
SysDescr OBJECT-TYPE
   SYNTAX    DisplayString (SIZE (0..255))
   ACCESS    read-only
   STATUS     mandatory
   DESCRIPTION
   "A textual description of the …. ASCII characteres."
   ::= { system 1 }
```

No exemplo acima foi utilizado a macro OBJECT-TYPE.

4.1.2 Basic Encondig Rules

A BER é uma especificação de codificação desenvolvida pela CCITT (X.209) e ISO (ISO 8825). Esta especificação descreve métodos de codificação de dados descritos com ASN.1 em uma seqüência de bytes. Isso é necessário pois em uma rede podemos ter várias arquiteturas de computadores, cada um representando seus dados de uma maneira diferente. BER é baseado em uma estrutura TLV (*type-length-value*, tipo-tamanho-conteúdo). Essa estrutura é recursiva, ou seja, um TVL pode conter uma ou mais TVL com mostra a Figura 4-2.

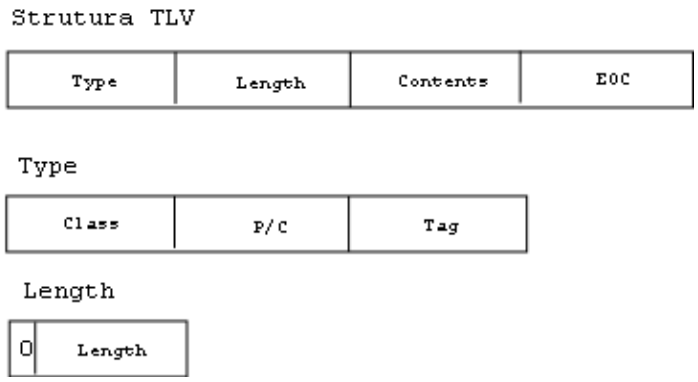

Figura 4-2 – TLV (Type Length Value)

A estrutura TLV é composta por:

- Type: indica o tipo, a classe e se a codificação é primitiva ou estruturada.
- Length: o tamanho do dado.
- Value: o valor do dado.

O *Type* ocupa, no mínimo 1 byte, e é obrigatório. Os primeiros 2 bits definem a classe do tipo, que são: *Universal*, aqueles tipos mais usados; *Application-wide*, usado por uma aplicação em particular; *Context-specific*, também aplicado por uma determinada aplicação; e *Private* que são definidos pelo usuário. O próximo bit define se é um tipo primitivo ou construído, e os últimos 5 bits definem qual é o tipo. Podemos ter desde tipos *boolean*, inteiros, strings a outros. Se a *Tag* do tipo tem mais que 5 bits, esse bit é 1, e a *Tag* é representada em bytes adicionais.

O campo de *length* determina o tamanho do TLV. O primeiro bit é 0, e o restante representa o valor. Se o valor for maior que 7 bits, o primeiro bit deve ser 1 e os 7 bits restantes dizem quantos bytes são usados para definir o tamanho, e a partir dele, o tamanho é representado.

O valor do dado é definida em *Contents*. Cada tipo tem um processo diferente de codificação.

O BER transforma os dados em uma seqüência de bytes que possam ser interpretados por varias arquiteturas. Logo, é usado em duas ocasiões, na codificação e decodificação dos dados. Possui duas funções principais: transformar uma seqüência de bytes para algum modo compreensível, no caso, para duas classes (Messages/Pdu) e transformar essas classes em bytes. A Figura 4-3 mostra como é feita a conversão de um determinado dado em pelo menos 2 bytes.

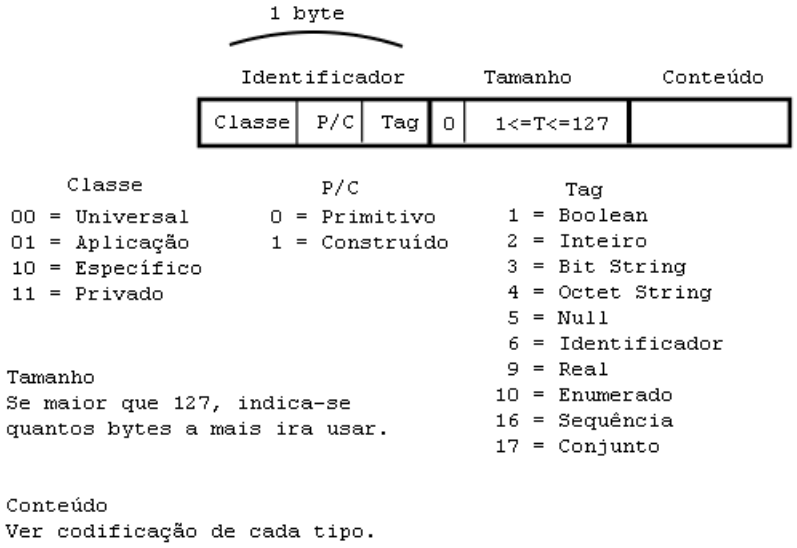

Figura 4-3- BER

A descrição da transformação de cada um dos tipos de dados é feita a seguir:

- *BOOLEAN*: O tipo booleano é Universal e primitivo. O verdadeiro pode ser representado como "01 01 FF" e o falso como "01 01 00".

- *INTEGER*: Números inteiros são representados usando a notação de complemento de dois, assim, quando o número for negativo o bit mais significante é 1 e quando o número é positivo é 0. Assim a representação do número 1 em BER é: "02 01 01". Ou seja, Universal, Primitivo de tamanho de 1 byte.

- *OCTET STRING*: Uma *String* pode ser construída ou primitiva, a diferença é que se for construída ela será dividida em pequenas *Strings*.

- *NULL*: São primitivos e são simples de representar: "05 00".

- *OBJECT ID*: O identificador dos objetos é uma seqüência de inteiros separados por ponto. O primeiro byte dessa seqüência de inteiros representa os dois primeiros números do identificador, isso é feito pois o primeiro número será sempre 0, 1 ou 2, e o segundo número será menor que 40, se o primeiro número for 0 ou 1. A Figura 4-4 mostra como os dois primeiros números são codificados.

$$Z = (X + 40) + Y$$

Valor	Primeiro Número	Segundo Número
0<=Z<=39	0	Z
40<=Z<=79	1	Z - 40
80<=Z	2	Z - 80

Figura 4-4 - Codificação de OID

Os restantes dos números são tratados normalmente como inteiros. Os tipos de enumerados, seqüência e conjunto, provém uma maneira de definir tipo em seqüência.

4.2 Management Information Base

Os objetos gerenciados na arquitetura SNMP estão agrupados em uma árvore hierárquica. Essa árvore forma a MIB, onde os nodos folhas representam cada objeto. Começando no nodo *root* da árvore temos três no primeiro nível: *iso*, *ccitt* e *join-iso-ciitt*. Sobre o nodo *iso* temos uma sub-árvore para ser usadas pelas organizações, uma delas é o departamento de defesa americano (*dod*). Então nessa sub-árvore tem um nodo que é alocado para a IAB. No nodo *internet* quatros nodos são definidos, sendo um deles, o *mgmt* que contém a MIB. A Figura 4-5 mostra a árvore formada até a MIB.

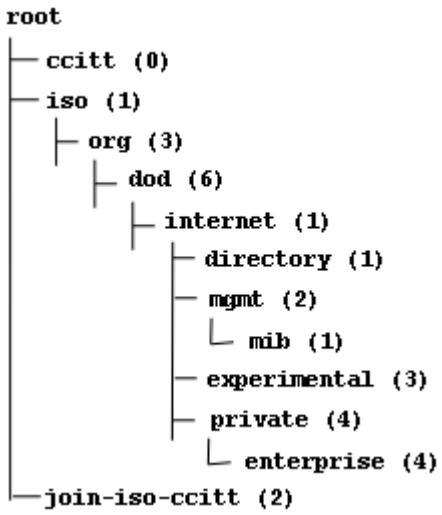

Figura 4-5 – Árvore da MIB

Como é mostrado na Figura 4-5, a MIB é uma árvore onde cada nodo folha representa um objeto. Além da MIB-II, temos outras MIBs, como por exemplo as proprietárias que ficam no grupo *enterprise*. O grupo *experimental* contém MIBs que estão sendo desenvolvidas, com será feito no trabalho. A principal MIB de roteamento é a MIB BGP que será explicada a seguir.

Existem duas versões da MIB, a MIB-I descrita na RFC 1156 e a MIB-II descrita na RFC 1213 que é uma extensão da primeira versão. A MIB-I definiu 114

objetos e a MIB-II adiciona mais 57 objetos gerenciados na definição. A MIB-II então é dividida em 10 grupos mostrados na Figura 4-6.

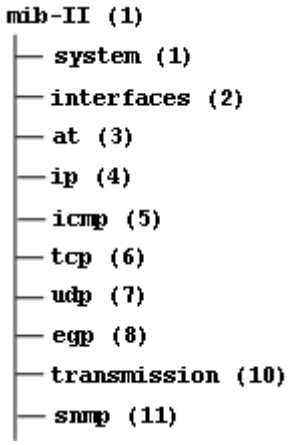

Figura 4-6 - MIB-II

As funções de cada uma das sub-árvores apresentadas na Figura 4-6 são:

- *System*: informações sobre o sistema.
- *Interfaces*: informações de cada interface do sistema.
- *At* (*address translation*; *deprecated*): descrição das tabelas de tradução de endereços.
- *Ip*: informações relativas à implementação IP do sistema.
- *Icmp*: informações relativas à implementação ICMP do sistema.
- *Tcp*: informações relativas à implementação TCP do sistema.
- *Udp*: informações relativas à implementação UDP do sistema.
- *Egp*: informações relativas à implementação EGP do sistema.
- *Transmission*: informações sobre os esquemas de transmissão e protocolos de acesso de cada interface do sistema.
- *Snmp*: informações relativas à implementação SNMP do sistema.

Cada nodo possui uma identificação (OID). A identificação é feita da seguinte maneira, todos os objetos possuem um número, esse número é atribuído em ordem crescente pelo nodo pai. Logo, a identificação de um objeto é feita percorrendo-se o nodo *root* até o nodo escolhido, concatenado a identificação. Assim, por exemplo, a MIB-II possui como identificação { 1.3.6.1.2.1 }.

4.2.1 MIB BGP

A MIB BGP, definida na RFC 1657 de 1994, define objetos que serão útil para o gerenciamento BGP. A Figura 4-7 mostra essa MIB.

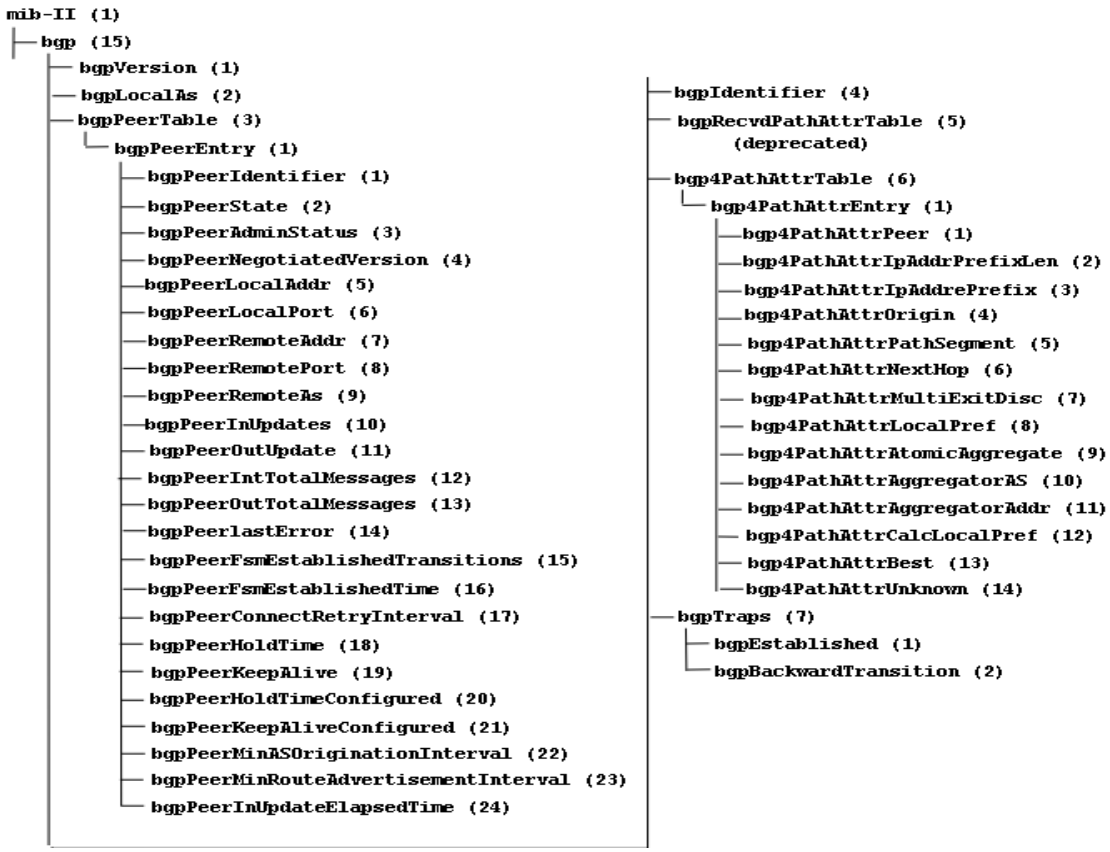

Figura 4-7 - MIB BGP

Essa MIB é divida em três tabelas. A primeira contém objetos sobre os *peers*, a segunda sobre a tabela de roteamento do protocolo BGP de versões anteriores e a terceira é sobre a tabela de roteamento da quarta versão do BGP. Ela possui ainda a definição de *traps* e mais alguns objetos sobre a versão do protocolo, número do AS e o identificador do AS. O trabalho se concentrará em analisar objetos da MIB BGP que estejam na tabela de estados das conexões e as rotas da quarta versão do protocolo.

4.3 SNMPv1

A RFC 1157 diz "SNMP explicitamente minimiza o número e a complexidade de funções de gerência realizada pelo agente". SNMP foi designado para ser simples, e a versão 1 reflete bem isso. Somente quatro operações podem ser realizadas nos objetos:

- *GetRequest*: retorna o valor de um objeto da MIB.
- *GetNextRequest*: retorna o valor do próximo objeto da MIB.
- *SetRequest*: atribui um valor a um objeto.
- *Trap*: o agente manda para o gerente um valor de objeto que não foi requisitado.

Também não é possível mudar a estrutura da MIB, somente atribuir e retornar a instância de um objeto. Somente podem ser acessados os nodos folhas, ou seja, não é possível acessar uma tabela inteira com somente uma operação. Isso contribui para a simplicidade de SNMP, mas limita a capacidade do sistema de gerência de redes [STA 99].

4.3.1 Segurança

Como SNMP é uma aplicação distribuída, com agentes possuindo suas próprias MIB, mecanismos de controle sobre o que os gerentes podem acessar em cada agente é necessário. Para isso, o SNMP define o conceito de comunidade, onde o acesso só é permitido a quem for da mesma comunidade. Ela é local ao agente sendo que ele pode ter várias comunidades para servir a diversos gerentes.

O serviço de autenticação garante para o receptor da mensagem que a origem é quem a mensagem diz ser. Para isso, toda mensagem possui uma *string* que representa a comunidade. Se a mensagem contém a comunidade que o receptor está esperando, então ele assume que a mensagem é autêntica. Isso torna esse serviço fraco, pois alguém pode capturar uma mensagem SNMP na rede, e copiar a comunidade.

A comunidade também é responsável pela política de acesso, ou seja, o agente pode ter diversas visões da MIB para diferentes comunidades. Também é definido o modo de acesso de cada comunidade, se é permitida leitura e escrita ou somente leitura. A combinação da visão com o modo de acesso é referida como SNMP *Community Profile*.

4.3.2 Protocolo

As operações da primeira versão do protocolo são apresentadas a seguir.

4.3.2.1 GetRequest

O gerente usa o *GetRequest* quando quer ler algum valor da MIB do agente. Ele manda um *GetRequest* para o agente que retornará os valores solicitados, se estiverem corretamente especificado. Se algum erro acontecer no momento da operação, nenhum valor será retornado, somente a mensagem vazia indicando o erro. Os seguintes erros podem acontecer:

- se um objeto indicado nas variáveis não existe na MIB do agente, ou é um nodo não-folha;
- se o tamanho da resposta de mensagem ultrapassar o limite; e,
- se o valor do objeto não pode ser retornado por qualquer outro motivo.

Um exemplo da operação de *GetRequest* é apresentado na Figura 4-8.

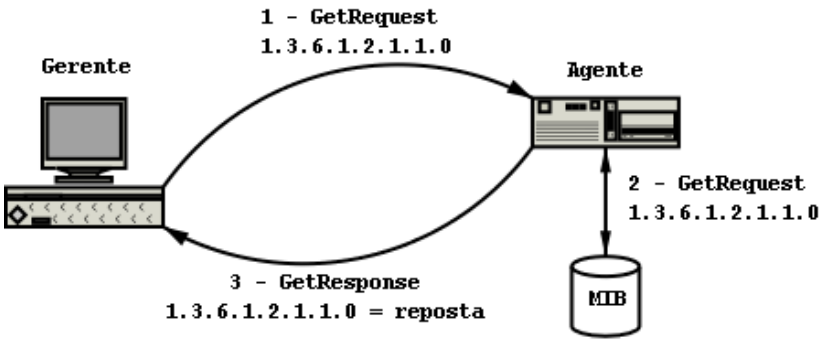

Figura 4-8 – Operação *GetRequest*

As seguintes etapas foram seguidas na Figura 4-8:

1. O gerente requisita a variável 1.3.6.1.2.1.1.0 para o agente.
2. O agente busca o valor na MIB.
3. O agente responde ao gerente o valor da variável.

A operação de *GetRequest* pode retornar somente uma instância dos objetos folhas. Se mais de uma instância for necessário, como por exemplo uma coluna ou toda a tabela de roteamento (*ipRouteTable*), cada instância deverá ser especificada.

4.3.2.2 GetNextRequest

Essa operação é parecida com o *GetRequest*, a diferença é que o *GetNextRequest* retorna o valor da próxima instância do objeto especificado na solicitação.

Para o agente, a diferença está em como ele processa a operação. Ao invés de retornar o valor da instância especificada, ele deve retornar a próxima instância. Para isso há uma ordem chamada de lexicográfica [STA99], onde começando do primeiro objeto da MIB, fazendo a operação sobre ele e então sobre o objeto que chegou na resposta e assim por diante, toda a MIB será retornada. A Figura 4-9 demonstra um exemplo de utilização do *GetNextRequest*.

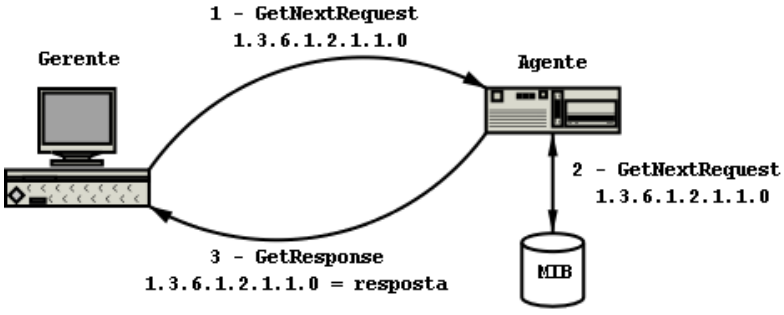

Figura 4-9 – Operação *GetNextRequest*

O exemplo é muito parecido com o *GetRequest*, a diferença é que o agente retorna a próxima instância. No caso, o gerente definiu a instância como zero e o

que foi retornado para ele foi o um. É importante lembrar, no caso de não ter mais uma instância um, ele ira retornar a primeira instância do próximo objeto, ou seja, 1.3.6.1.2.1.2.0.

A grande vantagem dessa operação é permitir que o administrador descubra objetos dinamicamente [STA 99], pois os índices dos objetos da MIB podem não seguir uma ordem seqüencial. A partir da primeira instância de um objeto de uma MIB, aplicando o *GetNextRequest*, todos a instância de todos objetos será visitada.

4.3.2.3 SetRequest

O gerente usa o *SetRequest* quando quer atribuir um valor a determinada instância de um objeto da MIB do agente. Essa operação é mostrada na Figura 4-10.

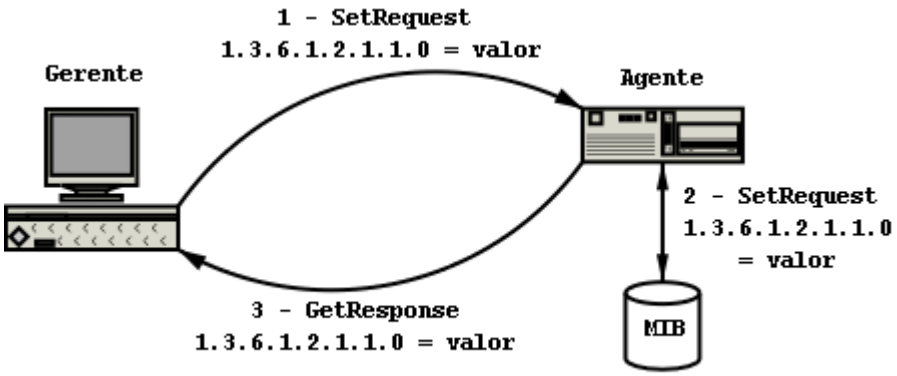

Figura 4-10 - Operação *SetRequest*

As seguintes etapas foram executadas na Figura apresentada:

1. O gerente atribui um valor a variável 1.3.6.1.2.1.1.0 e manda para o agente
2. O agente recebe e grava na MIB o valor.
3. O agente então retorna a confirmação para o gerente.

Deve-se ter cuidado com essa operação, pois o único mecanismo de segurança provido por SNMPv1 é a comunidade. Alguém que tenha conseguido a comunidade por algum modo pode executar operações de *SetRequest* em agentes causando sérios problemas.

4.3.2.4 Trap

A operação de *TRAP* o agente usa para notificar o gerente de algum evento significativo. Essa notificação parte do agente, o gerente não precisa requisitar para o gerente sobre alguma informação. Dessa forma, é desnecessário que o gerente fique comunicando-se com agente muitas vezes, pois o agente tem a capacidade de gerar esses *TRAPS* sobre determinados eventos. Um exemplo dessa operação é apresentado na Figura 4-11.

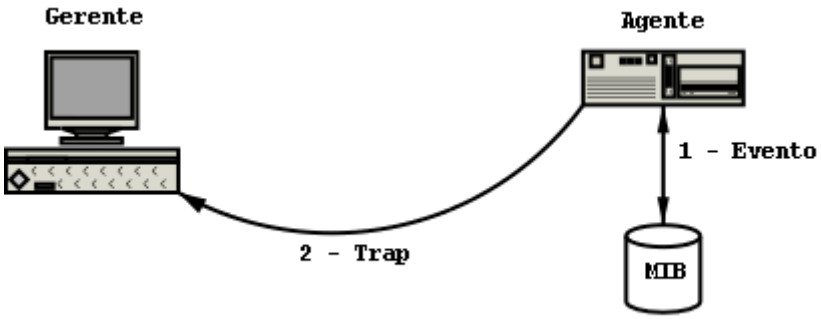

Figura 4-11 - Operação *Trap*

No *TRAP* é definido um evento que se quer monitorar, por exemplo, quando um objeto tenha em seu conteúdo o valor zero. Quando este evento acontecer um *TRAP* é enviado pelo agente para o gerente avisando.

4.3.2.5 Message/Pdu

As mensagens são encapsuladas através das SNMP *Messages*. Essas mensagens contêm a versão do protocolo, a comunidade e a PDU. Existem 3 tipos de PDU. A Figura 4-12 mostra a mensagem e suas PDUs.

Message v1

Version	Community	PDU

GetRequest, GetNextRequest, SetRequest PDU

PDU Type	Request ID	0	0	VariableBindings

GetResponse PDU

PDU Type	Request ID	Error-Status	Error-Index	VariableBindings

Trap PDU

PDU Type	Enterprise	Agent-Addr	Generic-Trap	Especic-Trap	Time-Stamp	VariableBindings

VariableBindings

Name1	Value1	Name2	Value2	...	NameN	ValueN

Figura 4-12 - Message e PDU SNMPv1

Os seguintes campos estão presentes nas mensagens.

- *Version*: Versão SNMP.
- *Community*, Comunidade.

- *Request-id*: Identificação única entre os pacotes.
- *Error-status*: Indica o erro que ocorreu no processamento da mensagem (*noError*(0), *tooBig*(1), *noSuchName*(2), *badValue*(3), *readOnly*(4), *genErr*(5)).
- *Error-index*: Indica qual variável causou o erro.
- *VariableBindings*: Lista de variáveis que serão usados na operação, com os seus respectivos valores.
- *Enterprise*: Tipo do objeto.
- *Agent-Addr*: Endereço do objeto que gerou o *trap*.
- Generic-trap: Tipo do trap (coldStart(0), warmStart(1), linkDown(2), linkUp(3), autentication-Failure(4), egpNeighborLoss(5), enterprise-Specific(6)).
- *Especific-trap*: Código do *trap* específico.
- *Time-stamp*: Tempo entre a última (re)inicialização da entidade e a geração do *trap*.

4.4 SNMPv2

Devido à falta de segurança da primeira versão de SNMP, o IETF começou a trabalhar para o melhoramento do SNMP. Os pontos fracos da primeira versão são a baixa segurança, o conjunto reduzido de operações e o *polling* entre o agente e gerente. Para resolver essas questões, a segunda versão propõem:

- Um modelo de segurança baseado no uso de criptografia.
- Um modelo de hierarquia, conhecido como MoM (*Manager of Manager*).
- Novas operações como *GetBulk, InformRequest* e *Report*.

Infelizmente, a versão que suportava segurança em SNMPv2 não foi aceito pela indústria. Isso acabou na publicação de um novo conjunto de RFCs em 1996, onde os aspectos de segurança foram retirados. Logo, os principais melhoramentos que SNMPv2 tem em relação a SNMPv1 diz respeito a SMI, comunicação gerente-gerente e operações no protocolo.

Outra diferença entre a primeira e a segunda versão de SNMP é a atomicidade das operações. Enquanto que na primeira versão, por exemplo, o *GetRequest* retornava todas as instâncias ou nenhuma, na segunda versão o mesmo método retorna aquilo que puder retornar. Se algum erro ocorrer, ele será reportado na PDU, mas as outras instâncias irão retornar também.

4.4.1 Protocolo

A primeira versão do protocolo previa somente a comunicação gerente-agente ou agente-gerente. A segunda versão prevê mais uma, a comunicação gerente-

gerente. Outra diferença no protocolo foi a adição de mais algumas operações como o *GetBulkRequest.*

Enquanto que a primeira versão do protocolo SNMP previa somente dois tipos de comunicação, onde o gerente manda informações para o agente ou o agente manda informações para o gerente. SNMPv2 adiciona mais uma, o gerente pode mandar uma mensagem para outro gerente. Outra diferença entre as duas versões é a adição da operação *GetBulkRequest*, *InformRequest* e *Report*.

4.4.1.1 GetBulkRequest

Um dos grandes avanços de SNMPv2 é a operação *GetBulkRequest*. Ele permite que o gerente SNMPv2 requisite que a resposta seja a maior possível dada o tamanho da mensagem [STA 99]. O *GetBulkRequest* funciona igual ao *GetNextRequest*, operando na próxima estância do objeto na ordem lexicográfica. A diferença é que se pode definir quantas instâncias serão retornadas.

Essa operação possui dois campos particulares, o *non-repeaters* e o *max-repetitions*. O primeiro define o número de variáveis da *variablebindings* que não serão processadas repetidamente. O segundo define o número de sucessores que serão retornados para o resto da lista.

4.4.1.2 InformRequest

O *InformRequest* é usado em comunicações gerente-gerente, não agente-gerente [STA 99]. É usado para prover informações de gerência por uma aplicação, através dos gerentes.

4.4.1.3 Report

Não há definição do como usar o *Report*. Isso aconteceu pois o uso dele estava nos documentos de segurança que foram retirados de SNMPv2, mas a referência dessa PDU continuou nos documentos de especificação.

4.4.1.4 Messages/PDU

SNMPv3 também encapsula a PDU em *SNMPv2 Messages* que tem a sintaxe igual do SNMPv1. Os formatos das PDU usados em na segunda versão são apresentadas na Figura 4-13.

Message v2

Version	Community	PDU

GetRequest, GetNextRequest, SetRequest , Trap, InformRequest PDU

PDU Type	Request ID	0	0	VariableBindings

GetResponse PDU

PDU Type	Request ID	Error-Status	Error-Index	VariableBindings

GetBulk PDU

PDU Type	Request ID	Non-repeaters	Max-repetitions	VariableBindings

VariableBindings

Name1	Value1	Name2	Value2	...	NameN	ValueN

Figura 4-13 - Message e PDU SNMPv2

Em SNMPv2, as PDUs das operações foram propostas de maneira que seja fácil sua implementação. Podemos notar que operações que antes tinham suas próprias PDUs, como o *Trap*, compartilham a mesma que outras. Foram definidos também dois campos em zero na primeira PDU para que a implementação dessas PDU possam usar as mesmas formas que as outras.

4.5 SNMPv3

Devido à falta de segurança das primeiras versões de SNMP, foi proposta a terceira versão do protocolo. Como a primeira versão praticamente não tinha preocupação com segurança, e a segunda versão teve muita dificuldade em padronizar a parte de segurança, a terceira versão de SNMP possui efetivos mecanismos de segurança. A Figura 4-14 mostra a evolução de SNMP.

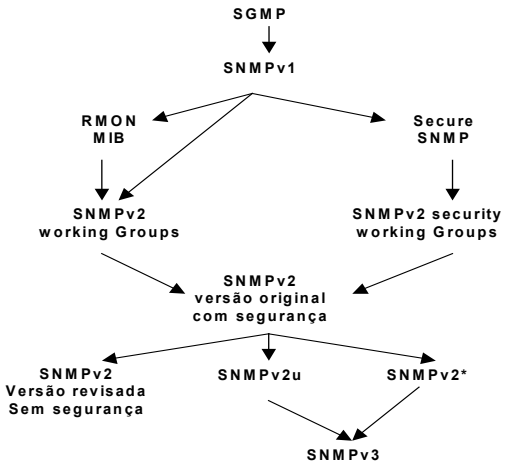

Figura 4-14 - Evolução SNMP

A terceira versão do protocolo deriva-se das informações relevantes e aplicabilidade de SNMPv1 e SNMPv2, mais um conjunto de mecanismos de segurança.

As RFCs 2271 até 2275 são insuficientes para definir a implementação de SNMP. Então, esses documentos indicam outros documentos que em conjunto formam a completa definição de SNMPv3. A seguir é apresentado esses documentos.

- *Document roadmap, Applicability statement, Coexistence and transition*.
- *Message Handling: Transport mappings, Message processing and dispatcher, Security*.
- *PDU Handling: Protocol operations, Applications, Access control*.
- *Information Model: Structure of management information, Textual conventions, Conformance statements*.
- *MIBs: Standard v1 RFC 1157 format, Standard v1 RFC 1212, Historic RFC 1441, 1443 and 1444 format, Draft RFC 1902, 1903 and 1904 format*.

Os três primeiros documentos serão usados em futuras aplicações. *Document Roadmap* lista todos as RFCs não obsoletas e indica como eles formam o *Frameworks* de SNMP, *Applicability Statement* descreve o ambiente em que SNMP é apropriado e *Coexistence and Transition* descreve a evolução entre as versões de SNMP e a sua existência entre eles.

Os documentos da *Message Handling* definem como as mensagens de SNMP são tratadas entre o agente e gerente. *Transport Mappings* define a maneira como SNMP é mapeado em protocolos de baixo nível, como UDP ou os protocolos de transporte de OSI. *Message Processing and Dispatcher* define o formato da mensagem. O documento *Security* prevê as funções de segurança, como autenticação e privacidade.

Os documentos *PDU Handling* definem como uma PDU é encapsulada em uma mensagem SNMP. *Protocol Operations* cria um conjunto de PDU e diz como processá-las. *Applications* define operações de gerência e *Access Control* determina se o acesso a determinado objeto é permitido.

Os documentos *Information Model* definem a estrutura e a sintaxe da informação de gerência. *Structure of Management Information* é um documento SMI que estabelece a sintaxe usada para construir MIBs. *Textual Convention* define novos tipos de dados e *Conformance Statements* estabelece o baixo nível na implementação para garantir a compatibilidade.

Os documentos da MIB definem a coleção dos objetos gerenciados. Esses objetos são aproveitados de SNMPv1 e SNMPv2, incluindo algumas versões de SNMPv2.

4.5.1 Segurança

A terceira versão do protocolo SNMP realmente se preocupa com segurança. Para a criptografia, pode-se usar 4 algoritmos: criptografia com o DES, MD5

secure hash function, SHA *secure hash function* e ainda autenticação de mensagens com HMAC.

4.5.2 Arquitetura da Entidade SNMPv3

Como SNMPv3 tem efetivos mecanismos de segurança, a sua arquitetura é mais complexa que as versões anteriores. Mas um dos princípios seguidos na modelagem das entidades é manter o mais simples possível, dentro das possibilidades. Pensando nisso, modelou-se módulos com funções bem específicas, onde, implementando esses módulos uma entidade pode atuar como um agente, ou um gerente, ou os dois. A Figura 4-15 apresenta a entidade SNMPv3.

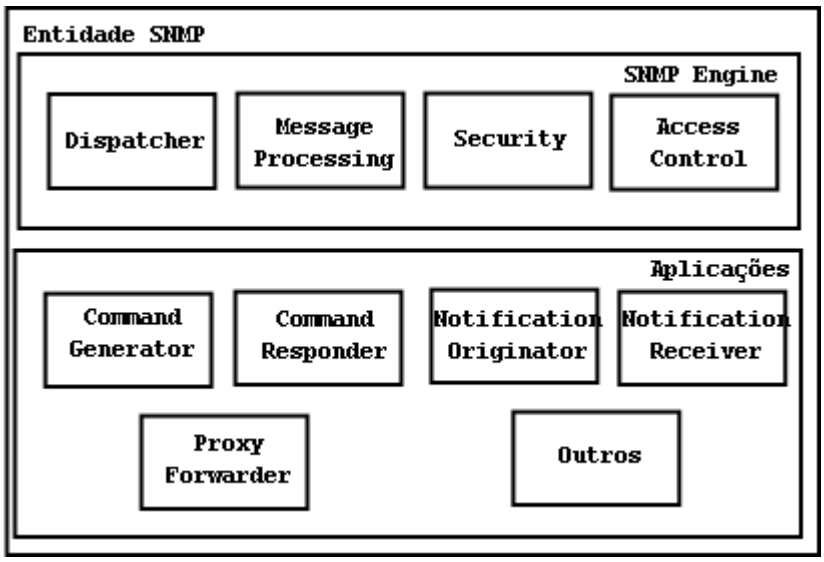

Figura 4-15 - Entidade SNMPv3

A entidade chamada de *Entidade SNMP* é formada por dois elementos. *SNMP Engine* e um ou mais *Applications*. Este *SNMP Engine* é responsável em mandar e receber mensagens, autenticar e criptografar/descriptografar mensagens, e controlar acesso a objetos gerenciados. A seguir a explicação de todos os módulos da entidade SNMPv3.

- *Dispatcher*: suporta múltiplas versões de mensagens, é responsável em mandar e receber PDU.
- *Message Processing Subsystem*: responsável em encapsular e desencapsular mensagens.
- *Security Subsystem*: prove serviço de segurança.
- *Access Control Subsystem*: prove serviço de controle de acesso.
- *Command Generator*: inicia *Get*, *GetNext*, *GetBulk* e *Set* PDU, a processar a reposta dessas operações.
- *Command Responder*: recebe *Get*, *GetNext*, *GetBulk* e *Set* PDU, e gera uma resposta para a operação.

- *Notification Originator*: monitora por um determinado evento no sistema e gera um *Trap* e/ou *Inform*.
- *Notification Receiver*: gera uma resposta quando uma mensagem de *Inform* PDU é recebida.
- *Proxy Forwarder*: Repassa mensagens SNMP.

A terceira versão do protocolo SNMP não apresenta novas operações, mas define novos tipos de mensagens para suportar segurança. Esses tipos são mostrados na Figura 4-16.

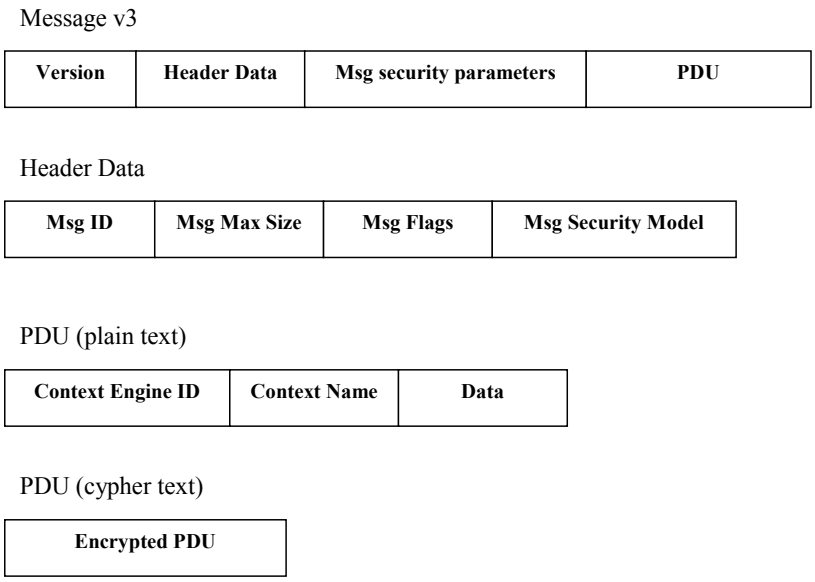

Figura 4-16 - SNMPv3 Messages

Esses tipos de mensagens continuam usando *Messages*/*PDU*, mas agora também define campos criptografados, bem como campos com informações sobre a segurança usada.

4.6 RMON

A RMON (*Remote Monitoring*) é uma MIB usada para monitorar e gerenciar segmentos remotos de redes distribuídas [STA 99]. A característica marcante de RMON é que apesar de ser uma definição de MIB, traz para SNMP grandes funcionalidades sem mudar nada na definição do protocolo. Os objetivos de RMON são os seguintes:

- operações *off-line*: coletar informações por um determinado tempo sem que seja necessário fazer o *polling* por parte do gerente;
- monitoração pró-ativa: monitoração contínua de objetos a fim de fornecer informações úteis no momento da ocorrência de problemas na rede;

- detecção de problemas: possibilidade de monitoração com ou sem *polling* de possíves características de problemas. No caso de alguma característica anormal ser detectada, comunicar imediatamente a estação de gerência;
- delegação de papéis: possibilidade de atribuição de tarefas a agentes situados em diferente pontos da rede, possibilitando um controle mais completo de características que a estação de gerência não teria acesso direto;
- múltiplos gerentes: possibilidade de relacionamento com mais de uma estação de gerência, de forma concorrente, a fim de prover uma maior confiabilidade e distribuir a tarefa de gerência a diferentes estações.

A MIB da RMON é um nodo filho da MIB-2, ela possui 10 grupos definido na RMON1 e mais 9 definidos na RMON2, como mostra a Figura 4-17.

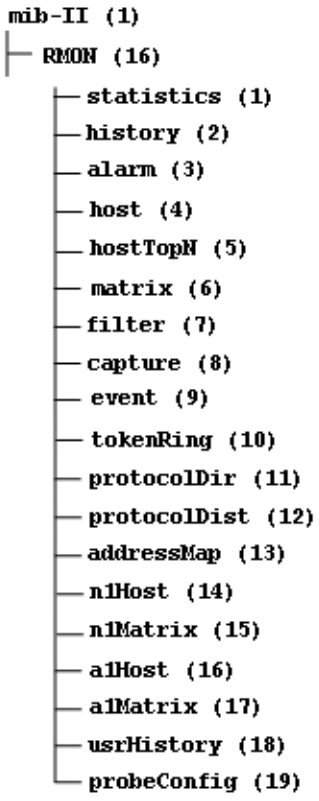

Figura 4-17 - RMON

Esses grupos têm como funções:

- *Statistics*: contém informações básicas de cada subrede monitorada.
- *History*: define funções de histórico das interfaces do sistema.
- *Alarm*: define alarmes referentes ao desempenho da rede.
- *Host*: usado para coletar estatísticas de determinado *hosts*.
- *HostTopN*: contém estatísticas sobre um conjunto de *host*.

- *Matrix*: contém informações sobre o tráfego entre dois *hosts*.
- *Filter*: define filtros que são aplicados aos pacotes de determinada interface.
- *Capture*: define um buffer para a captura de pacotes.
- *Event*: define eventos.
- *TokenRing*: informações para gerenciar redes *token ring*.
- *ProtocolDir*: contém informações sobre protocolos.
- *ProtocolDist*: contém a quantidade de bytes e pacotes enviados por cada um dos protocolos.
- *AddressMap*: relaciona cada um dos endereços de rede com um endereço MAC e uma porta.
- *N1Host*: decodifica pacotes baseados no nível de rede.
- *N1Matrix*: contém informações sobre o tráfego entre dois *hosts* sobre o nível de rede.
- *A1Host*: decodifica pacotes baseados no nível de aplicação.
- *A1Matrix*: contém informações sobre o tráfego entre dois *hosts* sobre o nível de aplicação.
- *UsrHistory*: verifica por uma determinada estatística ou variável e grava esse dado.
- *ProbeConfig*: troca informações com outros agentes RMON e gerentes.

Em 1997, duas RFCs definiram a RMON2. Esta especificação inclui monitoramento de tráfego de protocolos acima do nível MAC, ou seja, RMON2 decodifica pacotes dos níveis 3 até 7 do modelo OSI [STA 99].

Um grupo da RMON2 que será bastante utilizado no trabalho será o *matrix*. Ele retorna informações sobre o tráfego entre dois *hosts* na rede. Isso será importante para monitorar a quantidade de tráfego entre os *hosts* para gerar essas informações para o gerente.

4.7 Considerações

Para a implementação do sistema que será proposta, precisa-se de um agente que implemente a MIB BGP e a MIB-II. Essas MIBs contém importantes informações que serão processadas. O agente que foi escolhido que disponibilizará essas MIBs é o da UCD-SNMP [UCD01] (nas versões mais novas chama-se NET-SNMP). Ele é implementado por um conjunto de pessoas na Internet e é distribuído gratuitamente.

Foi escolhida então essa ferramenta, pois ela, além de ser livre, o seu agente implementa a MIB BGP pois tem condições de fazer uma ligação com o *software* de roteamento utilizado em PTTs.

Com essa integração será possível acessar dados da tabela de roteamento e sobre as sessões estabelecidas entre os *peers*. Tais informações são fundamentais para alcançar o objetivo do trabalho.

5. SISTEMA DE GERENCIAMENTO DO PROTOCOLO BGP EM PTTs

Neste trabalho é apresentado um sistema de gerenciamento SNMP para monitoração do tráfego BGP em PTTs. O objetivo deste trabalho é fornecer informações relevantes para o gerenciamento das operações de roteamento entre sistemas autônomos, através da monitoração, coleta e processamento de informações a respeito do protocolo BGP e características da rede.

No gerenciamento de um PTT, as informações que as MIBs existentes fornecem são importantes, mas encontram-se em diferentes componentes do PTT. Também, as informações fornecidas por essas MIBs não são específicas para a efetiva administração do PTT. Assim sendo, este trabalho propõe a implementação de um processo de gerenciamento que facilite a monitoração do roteamento BGP em PTTs. Para alcançar esse objetivo, o processo proposto:

- Procura coletar as informações importantes dessas MIBs nos diferentes componentes do PTT.
- Fornece uma nova MIB contendo apenas as informações específicas e importantes ao gerenciamento do BGP no contexto de PTTs. Adicionalmente, nessa MIB são disponibilizadas informações que são obtidas também no *Route Server*.
- Fornece um sistema de *traps* que observa e reporta comportamentos anormais no dia-a-dia de um PTT.

Dessa forma, com este sistema, a administração do PTT baseia-se em uma única MIB, que fornece todas as informações úteis, sendo algumas delas simplesmente coletadas das MIBs já existentes e outras processadas a partir de informações primitivas.

A seguir, são apresentadas a arquitetura do sistema, a descrição das entidades que a compõem e as informações de gerenciamento, bem como as MIBs resultantes para o gerenciamento proposto.

5.1 Arquitetura do sistema

A arquitetura desenvolvida neste trabalho é apresentada na Figura 5-1. Nesta arquitetura são representadas todas as entidades das quais obtém-se informações para o gerenciamento do sistema e as demais entidades existentes.

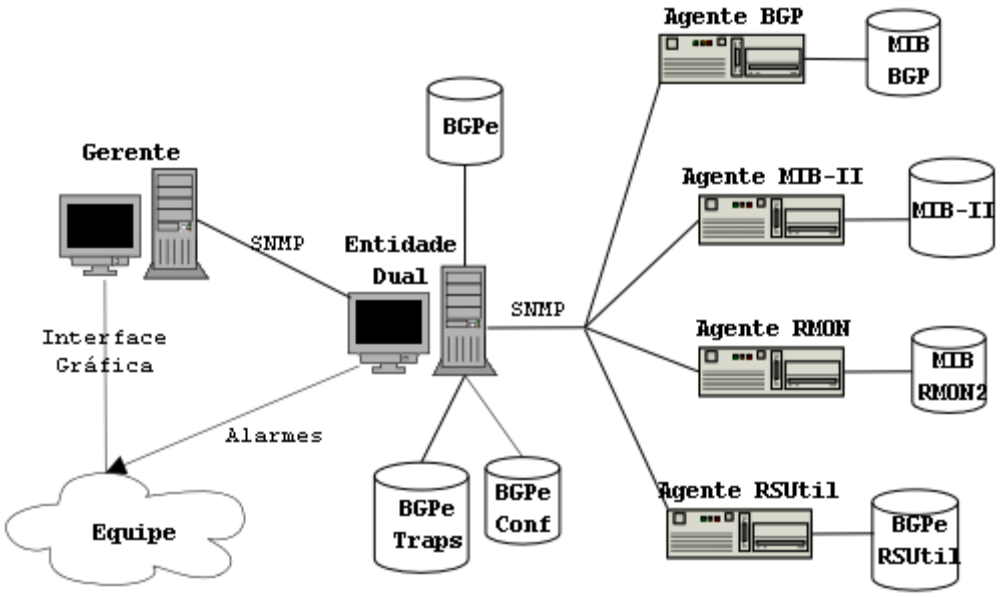

Figura 5-1 – Arquitetura do Sistema

Como pode-se verificar na Figura 5.1, a entidade Dual é quem fornece acesso às MIBs resultantes do sistema, buscando para isso as informações de 4 agentes localizados em diferentes componentes no PTT via protocolo SNMP. Os resultados obtidos são fornecidos via SNMP para uma estação de Gerência, chamada de entidade Gerente. Através da entidade Gerente a equipe do suporte ao PTT poderá efetivamente fazer uso das informações fornecidas pelo sistema. A função de cada uma das entidades é descrita a seguir:

- Entidade Gerente: Esta entidade é quem dispara solicitações e obtém as informações do sistema. O gerente faz a interface entre a equipe do PTT e o sistema implementado. Esta entidade geralmente utiliza informações de diversos sistemas que podem ser utilizados em conjunto. A localização desta entidade geralmente é uma estação de trabalho que tem como função principal o próprio sistema de gerência da rede. O gerente já está implementado utilizando recursos/ferramentas já existentes.

- Entidade Dual: Esta é a entidade principal do sistema, pois faz a união das informações que se encontram dispersas pelos diversos agentes existentes. Desempenha o papel de gerente para fazer requisições e agente para atender requisições. Essa entidade implementa três MIBs (BGPe, BGPe Conf e BGPe Traps) propostas pelo trabalho e faz acesso a diversas MIBs, entre elas a MIB RSUtil. Com estas informações, esta entidade faz o processamento dos dados. A localização desta entidade poderia ser em uma estação ou computador localizados em qualquer rede, desde que tenha permissões para acessar os agentes. Também poderia ser definido como local de utilização deste sistema o

próprio gerente, ou ainda, em caso extremo, em algum dos *Route Servers* com alguns ajustes necessários. No protótipo esta entidade está presente em uma máquina exclusiva.

- Agentes: Estas entidades possuem a capacidade de coletar os dados necessários para o sistema. Cada um dos agentes representa um grupo de informações necessárias para a entidade dual. As MIBs BGP, RMON e MIB II já se encontram implementadas nos próprios equipamentos utilizados em um PTT. Já o agente BGPe RSUtil foi totalmente implementado. O agente BGP está localizado nos *Route Servers*. Este módulo é baseado na integração entre o *software* responsável pelo roteamento BGP e o *software* NET-SNMP que implementa um servidor SNMP. A MIB BGP, utilizada neste agente, já é definida e suportada por estas ferramentas. O agente BGPe RSUtil também está presente nos *Route Servers*, sendo implementado para fornecer os *logs* provenientes do *software* de roteamento BGP e algumas informações do próprio sistema operacional dos *Route Servers*. Já os agentes RMON e MIB II estão presentes no próprio *Switch* do PTT, onde todos os participantes estão conectados.

No conjunto destas entidades, um dos benefícios é a unificação de configuração e obtenção de informações para o gerenciamento através da entidade Dual. A visão a partir do gerente e da equipe de suporte é simplificada aos objetos presentes nas MIBs BGPe, BGPeTraps e BgpeConf.

O restante dos agentes pelo qual as informações são obtidas são consultados pela entidade Dual que, além de coletar, processa e fornece apenas as informações definidas nas MIBs. Uma das facilidades desta solução é a configuração através da MIB BGPeConf, que pode ser configurada via SNMP a partir de qualquer outro *host* da Internet. Como desvantagem pode-se citar a entidade Dual como ponto único de falha do sistema de gerência, já que se esta falhar, todo o sistema de gerência estará comprometido.

5.2 Informações para Gerenciamento

Como já foi citado, diversas MIBs foram utilizadas para obter informações para gerenciamento do PTT. As principais MIBs já existentes onde essas informações são extraídas são a MIB II, MIB BGP e MIB RMON. Tais MIBs são implementadas na maior parte de equipamentos de rede utilizados atualmente. Parte das informações importantes ao gerenciamento do protocolo BGP já existem, mas para atender as necessidades específicas de um PTT as informações são abrangentes demais e não chegam ao nível de mostrar apenas as informações efetivamente úteis, discriminando-as.

Sendo assim, algumas informações são coletadas das MIBs já existentes e apenas organizadas de acordo com a organização do PTT, enquanto outras sofrem

alguma espécie de análise ou contabilização para depois serem apresentadas na forma de uma nova MIB. As MIBs que são utilizadas como base de informações ao sistema são apresentadas a seguir:

5.2.1 MIB II

Esta MIB será utilizada pelo agente MIB-II. Sua principal utilização é a coleta de informações sobre o tráfego de entrada e saída de cada participante do PTT. Esta informação é importante pois a troca de tráfego é a objetivo principal do PTT e alterações nessa informação podem ser provocadas por problemas no PTT. Estas informações são obtidas no *Switch* onde cada participante encontra-se obrigatoriamente conectado a rede do PTT. Entre os subgrupos existentes, é utilizado o grupo Interfaces que possui os objetos *IfInOctets* e *IfOutOctets* que retornam tais informações de cada interface.

Além do tráfego de entrada e saída de cada participante, é importante coletar o tráfego global do PTT. A forma mais simples para obter tal informação seria a soma do tráfego de todas as interfaces do *Switch*. Entretando é possível criar no *Switch* as chamadas *Virtual Lans* (VLANs), possibilitando que várias interfaces do *Switch* possam ser vistas como uma única interface. Isso permitirá obter de forma direta o tráfego global do PTT, se incluídas todas as interfaces dos participantes do PTT em uma VLAN.

Para ilustrar o conceito de VLAN, é mostrada através da Figura 5-2 um *Switch* onde os roteadores 1, 2 e 3 formam uma VLAN. Através desta VLAN a estação de gerência poderá obter o tráfego total entre estes roteadores obtendo apenas os dados da VLAN.

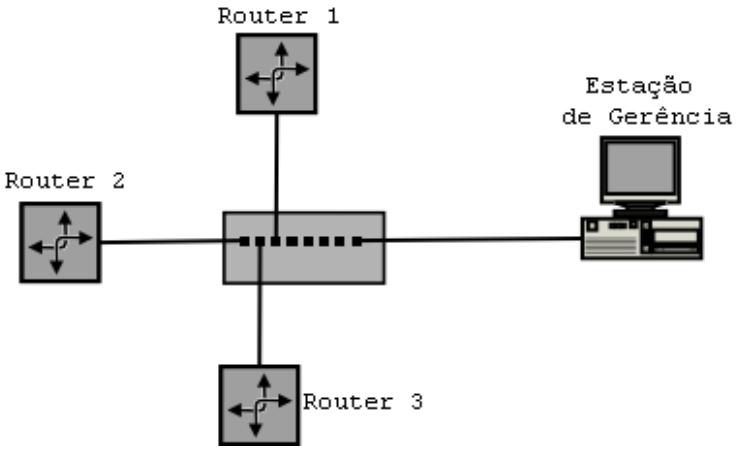

Figura 5-2 – Exemplo de aplicação do conceito de VLAN

Para obter os dados da VLAN, como em qualquer outra interface do *Switch* é necessário saber qual índice a VLAN pertence, consultando os índices do próprio *Switch* como é mostrado na Figura 5-3.

```
[root@gerencia]# snmpwalk -v 2c -c public 10.0.0.1
```

```
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifIndex.4 = INTEGER: 4
IF-MIB::ifDescr.1 = STRING: Port 1 on Unit 1
IF-MIB::ifDescr.2 = STRING: Port 2 on Unit 1
IF-MIB::ifDescr.3 = STRING: Port 3 on Unit 1
IF-MIB::ifDescr.4 = STRING: VLAN 00001 (Default)
...
```

Figura 5-3 – Exemplo de consulta aos índices das interfaces do *Switch*

Neste contexto, é obtida a informação desta VLAN que representa o tráfego total entre os três roteadores como mostra a Figura 5-4.

```
[root@gerencia]# snmpget 10.0.0.1 community interfaces.ifTable.ifEntry.ifInOctets.4
interfaces.ifTable.ifEntry.ifInOctets.4 = 30330722
```

Figura 5-4 – Exemplo de *Get*

De acordo com a Figura 5.4, o tráfego de entrada na interface de índice 4 que representa a VLAN é pouco mais de 30MBps.

5.2.2 MIB RMON II

Como já foi visto que o tráfego de entrada e saída de cada participante é importante, da mesma forma é importante saber para onde flui o tráfego entre os participantes. Mediante tal necessidade será usado o grupo *AlMatrix* da MIB RMON para mostrar o tráfego entre pares de participantes. Este grupo possui três tabelas, uma de controle e duas de dados. Essas tabelas de dados são usadas para armazenar informações sobre o tráfego de um *host* origem para um número de *hosts* destinos. A outra tabela contém as mesmas informações, mas é indexada pelo *host* destino. Com essa informação, é possível montar gráficos mostrando o tráfego entre cada um dos participantes do PTT.

5.2.3 MIB BGP

Esta MIB é utilizada pelo agente BGP presente em cada *Route Server*. Com a integração do *software* para roteamento BGP (Zebra) e o pacote NET-SNMP é possível obter os dados desta MIB. As informações provenientes desta MIB são as mais importantes do sistema, visto que refletem o estado do protocolo BGP e suas operações em relação a cada participante.

Nesta MIB, diversos objetos são utilizados. As informações constantes neste agente são utilizadas para verificar todas as informações do protocolo BGP pertinentes a cada participante do PTT. Alguns destes objetos são utilizados no processamento dos dados. Alguns exemplos de objetos utilizados seriam:

- *bgpPeerState*;
- *bgpPeerRemoteAS*;
- *bgpPeerInUpdates*;
- *bgpPeerOutUpdates*;
- *bgpPeerLastError*;
- *bgp4PathAttrPeer*;
- *bgp4PathAttrIpAddrPrefixLen*;
- *bgp4PathAttrIpAddrPrefix*;
- *bgp4PathAttrOrigin*;
- *bgp4PathAttrASPathSegment*;
- *bgp4PathAttrNextHop*.

Um exemplo para ilustrar essa MIB é apresentada pela Figura 5-5 que ilustra um roteador (10.0.0.2) com sessões BGP com três outros sistemas autônomos.

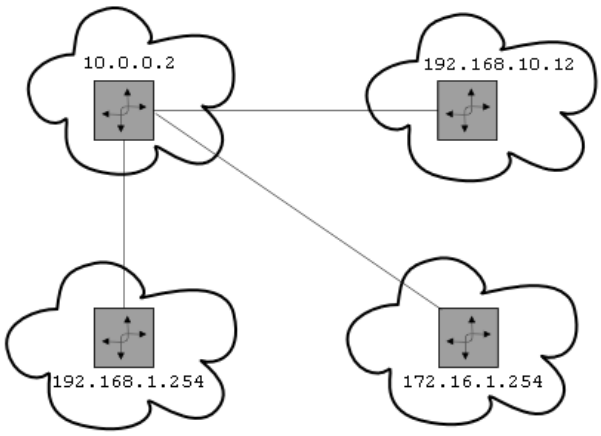

Figura 5-5 - Topologia criada para teste da MIB BGP

A partir deste roteador, é verificado através do objeto *bgpPeerState* o *status* dessas conexões BGP, como é apresentado abaixo pela Figura 5-6.

```
[root@gerencia]# snmpwalk 10.0.0.2 community .1.3.6.1.2.1.15.3.1.2
15.3.1.2.192.168.1.254 = 6
15.3.1.2.172.16.1.254 = 6
15.3.1.2.192.168.10.12 = 6
```

Figura 5-6 - Saída do *snmpwalk*

Como se pode verificar, o *status* das sessões BGP com os quatro *peers* é de *Established*, definido pelo código 6, de acordo com a seguinte codificação: (1) *Idle*, (2) *Connect*, (3) *Active*, (4) *OpenSent*, (5) *OpenConfirm* e (6) *Established*.

É desejável que as sessões BGP estejam no estado *Established*, visto que nesse estado a sessão BGP está ativa e ocorre de fato a troca de anúncios, e por conseqüência a troca de tráfego.

5.3 MIBs resultantes do trabalho

Como resultado do processamento do sistema são disponibilizadas quatro MIBs: BGPe, MIB BGPe Conf, MIB BGPe Traps e MIB RSUtil. Todas elas estão localizadas no grupo *experimental*, que é reservada para MIBs em teste.

5.3.1 MIB BGPe

Essa MIB fornece informações sobre o estado global do PTT, bem como o estado de cada participante. As informações fornecidas por essa MIB são o resultado do processamento dos dados obtidos através dos agentes. O resultado dessas informações pode ser analisado pelo gerente na forma mais adequada que ele desejar. A MIB BGPe, tem os seguintes objetos, com mostra a Figura 5-7.

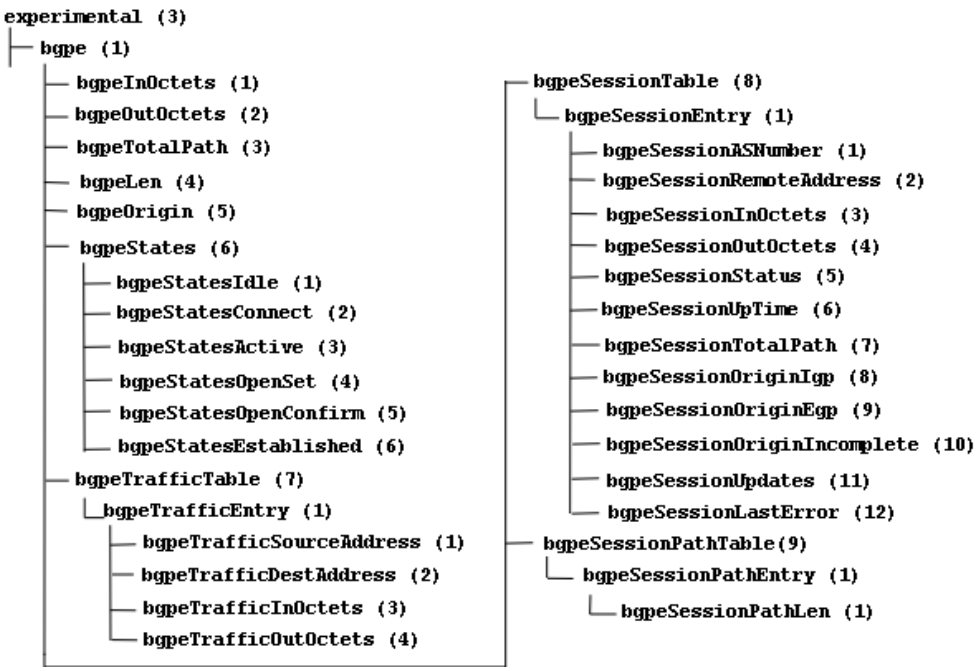

Figura 5-7 - MIB BGPe

Em resumo, a MIB BGPe pode ser dividida em duas partes principais. A primeira delas possui informações globais ao PTT, ou seja, informações que refletem a situação e comportamento de todos os participantes agrupados, sem distinção. Já a segunda parte agrupa informações particulares para cada participante, sendo útil na análise particular de informações de determinado participante.

A explicação dos objetos da primeira parte, ou seja, dos dados globais do PTT é dada a seguir:

- *BgpeInOctets*: Fornece o tráfego de entrada. Essa informação é obtida através do objeto *ifInOctets* do grupo interfaces. Como esse objeto busca o tráfego total em todas as interfaces do *Switch* do PTT, é necessário utilizar uma VLAN (*Virtual Lan*) para agrupar todas as interfaces e coletar essa informação. Tal informação é útil para saber o total de tráfego trocado no PTT.

- *BgpeOutOctets*: Fornece o tráfego de saída. Essa informação é obtida através do objeto *ifOutOctets* do grupo interfaces. Sua obtenção e origem são iguais ao objeto anterior, ou seja, coletados da mesma VLAN que engloba todas as interfaces no *Switch*. A regra é que o tráfego de saída seja muito semelhante ao tráfego de entrada do objeto anterior, já que todos os pacotes trocados entre dois participantes necessariamente saem de uma interface para entrar em outra do mesmo *Switch*.

- *BgpeTotalPath*: Fornece o total de prefixos anunciados por todos os participantes do PTT. Esse dado é importante para verificar possíveis anomalias, já que o padrão é que cada participante

anuncie apenas rotas de seu *backbone*. Mudanças abruptas podem significar problemas na configuração ou queda de algum participante do PTT. Tal dado é obtido através do agente BGP, contabilizando as instâncias do objeto *bgp4PathAttrIpAddrPrefix* que é um tabela com todos os prefixos anunciados no PTT.

- *BgpeLen*: Fornece a classificação dos anúncios quanto ao tamanho das redes anunciadas. Este tamanho de redes também é chamado de "profundidade" do bloco anunciado. No caso do BGP, os blocos são representados na forma de blocos CIDR. As fontes dessas informações são as instâncias do objeto *bgp4PathAttrIpAddrPrefixLen*. Sabendo que o tamanho de redes poderá variar entre /8 e /32 em notação CIDR, para cada possível tamanho será mostrado o número de ocorrências, inclusive se o número de redes de determinado tamanho for zero. Tal dado é importante afim de verificar por anúncios muito específicos, já que o normal é que os tamanhos sejam entre /8 e /20.

- *BgpeOrigin*: Fornece a classificação dos anúncios quanto a sua origem. Esse dado reflete a confiabilidade dos anúncios.Os tipos possíveis de anúncios são *IGP*, *EGP* ou *Incomplete* (quando não é possível determinar sua origem). Sendo assim, as rotas são classificadas dentro destes três tipos possíveis e contabilizadas, indexando pelos códigos 0 (*IGP*), 1 (*EGP*) ou 2 (*Incomplete*), seguido pelo número de ocorrências. É retirado do objeto *bgp4PathAttrOrigin* presente no agente BGP.

- *BgpeStates*: Fornece o número de participantes em cada estado da máquina de estados do BGP. Esse dado é importante pois demonstra facilmente se algum participante não se encontra no estado *Established* (conexão estabelecida) com o *Route Server*. Espera-se que esse número seja sempre igual ao número total de participantes, ou seja, que todos tenham sua conexão no estado *Established*. Os demais estados podem ser vistos através da Figura 2.3. Tal informação é obtida pelo objeto *bgpPeerState* da MIB BGP.

- *BgpeTrafficTable*: Fornece o tráfego entre pares de participantes do PTT. Tal informação é útil pois mostra para onde flui o tráfego de cada AS e não somente seu tráfego total com o PTT. A obtenção dessa informação é feita pelo agente RMON presente no *Switch* do PTT, por intermédio da MIB RMON. A função de cada um dos objetos é:
  - *BgpeTrafficSourceAddress*: Endereço origem do equipamento.
  - *BgpeTrafficDestAddress*: Endereço destino do equipamento.
  - *BgpetrafficInOctets*: Quantidade total de bytes recebidos.

- *BgpeTrafficOutOctets*: Quantidade total de bytes enviados.

As demais informações apresentadas abaixo representam as informações particulares de cada participante, formando a segunda parte dessa MIB. Serão mostrados os objetos que formam a base de dados de um participante, ou seja, para cada participante existe uma instância desses objetos. O nome da tabela que representa esse grupo é *bgpeSessionTable*. Essas informações são:

- *BgpeSessionASNumber*: Fornece o número do sistema autônomo.

- *BgpeSessionRemoteAddress*: Fornece o endereço IP do *peer* da sessão BGP remoto. Da mesma forma que objeto anterior, usado para identificar qual o participante.

- *BgpeSessionInOctets*: Fornece a quantidade de octetos que foram recebidos, esse dado é obtido através do objeto *ifInOctets* do grupo interfaces da MIB-II. Idem ao objeto global *BgpeInOctets*.

- *BgpeSessionOutOctets*: Fornece a quantidade de bytes que foram enviados, esse dado é obtido através do objeto *ifOutOctets* do grupo interfaces da MIB-II. Idem ao objeto global *BgpeOutOctets*.

- *BgpeSessionStatus*: Fornece o status atual da sessão BGP. Essa informação é importante para determinar problemas de conexão entre o roteador do participante aos *Route Servers*. Será obtida pela MIB BGP através do objeto *bgpPeerState*. Idem ao objeto global *BgpeStatus*.

- *BgpeSessionUpTime*: Fornece o tempo decorrente da sessão BGP entre o *Route Server* e o participante. Obtida através da MIB BGP pelo objeto *bgpPeerFscmEstablishedTime*. Dado importante para avaliar se a conexão do participante apresenta alguma instabilidade e fornece o tempo decorrente da última transição.

- *BgpeSessionTotalPath*: Fornece o número total de prefixos anunciados. Idem ao objeto global *BgpeTotalPath*.

- *BgpeSessionPathLen*: Fornece a classificação dos anúncios pelo tamanho das redes. Idem ao objeto global *BgpePathLen*.

- *BgpeSessionOrigin*: Fornece a quantidade de prefixos segundo o atributo origin. Cada prefixo possui um atributo ORIGIN que reflete diretamente na confiabilidade da rota. Idem ao objeto global *BgpeOrigin*.

- *BgpeSessionUpdates*: Fornece o número de mensagens tipo UPDATE trocadas com o *Route Server*. Obtido pela MIB BGP, através do objeto *bgpPeerInUpdates*, mostrando a estabilidade dos anúncios do participante, ou seja, se ocorre uma grande variância em seus anúncios. Em alguns casos, uma grande variância neste objeto reflete em *dampening* de alguma rede do participante.

- *BgpeSessionLastError*: Fornece o último erro ocorrido na sessão BGP entre o participante e o *Route Server*. Obtido através da MIB BGP pelo objeto *bgpPeerLastError*. Tal informação é útil para diagnosticar problemas de conectividade ou incompatibilidade entre o equipamento do participante e o *Route Server*.

- *BgpeSessionPathTable*: Fornece o tamanho dos prefixos de cada participante. Idem ao objeto global *BgpePathTable.*

A descrição desta MIB resultante, bem como a especificação dos tipos e estruturas de dados a serem retornados em linguagem ASN.1 encontra-se no Anexo A. A relação dos nomes de objetos e seus respectivos OIDs encontra-se no Anexo G.

5.3.2 MIB BGPe Traps

Este módulo é responsável pelo controle de comportamento anormais no funcionamento do protocolo BGP e dos *Route Servers*. A comunicação será feita mediante o envio de *traps* ao gerente e, se desejado, envio de e-mail. Neste caso, a equipe do PTT receberá o aviso do evento tão logo ele ocorra, sem atrasos e com a agilidade necessária. Além disso, também é possível registrar em arquivo de LOG todos os eventos reportados por esse módulo para análise posterior e também programar o envio de um relatório periódico dos dados mais significativos. A definição de quais eventos serão reportados é feita na MIB de configuração BGPe Conf, que será apresentada adiante. A Figura 5-8 mostra os objetos dessa MIB.

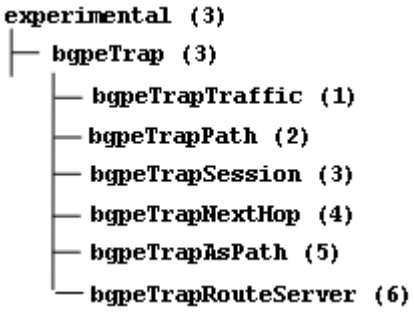

Figura 5-8 - MIB BGPeTrap

A função de cada um dos objetos é:

- *BgpeTrapTraffic*: Reporta eventos referentes a medidas de tráfego de determinado participante, sinalizando se o limite de tráfego máximo foi ultrapassado ou mínimo não alcançado. Esse dado será obtido pela comparação dos limites estabelecidos na MIB de configuração e o tráfego do participante mostrado pelos objetos *BgpeSessionInOctets* e *BgpeSessionOutOctets*. Tal controle é importante para garantir que a conexão até o PTT de determinado participante não seja superutilizada, aumentando a latência da conexão e por conseqüência, diminuindo a qualidade no acesso. Será enviado o número do AS do participante que apresenta comportamento anormal, bem como o tráfego atual e o limite do tráfego máximo. O mesmo procedimento será aplicado ao tráfego mínimo.

- *BgpeTrapPath*: Reporta anomalias nos anúncios de rota. Esta *trap* permite que seja controlado o número total de prefixos anunciados. Isso é obtido acessando o agente BGPe RSUtil ou ainda, através da MIB de configuração e a MIB BGPe. Através do *software* Zebra é possível limitar o número máximo de anúncios e, no caso de alcançar esse limite, a sessão BGP é derrubada e registrado em *log* tal prática. Isso é coletado através da MIB RSUtil pelo objeto *bgpeRSUtilMaxAnnounc*. No caso do *Route Server* não implementar tal controle, o sistema pode fazer essa verificação comparando o número de rotas esperados do participante através do objeto *BgpeConfSessionMaxPath* e do objeto *BgpeConfSessionMinPath*, presentes na MIB BGPeConf e o objeto *BgpeSessionTotalPath* da MIB BGPe que representa o número total de anúncios do participante em dado momento. Tal controle, evita que algum participante acidentalmente anuncie mais prefixos do que foi planejado ou faça *Full Routing* a partir de seu AS. Este objeto retorna o AS problemático seguido pelo número de anúncios esperado e número de anúncios atual.

- *BgpeTrapSession*: Reporta problemas nas sessões BGP, permitindo que sejam reportados eventos referentes a quedas de sessões BGP de algum participante. As quedas ou problemas na sessão BGP são facilmente encontrados analisando se o status da conexão assume algum valor diferente de *Established*. Tal controle é feito pela MIB BGP, através dos objetos do *status* da conexão de cada participante. No caso de ocorrer esse evento, as informações que acompanham a *trap* são a descrição e o AS que apresentou problemas, além do último código de erro (*BgpeSessionLastError* da Bgpe) e seu significado, afim de auxiliar na verificação do problema.

- *BgpeTrapNextHop*: Reporta se o atributo NEXT_HOP de todos os anúncios corresponde ao IP de algum dos participantes do PTT.

Isso é feito para evitar que algum anúncio seja feito de forma incorreta no PTT. Caso isso ocorra, é retornado o IP definido como NEXT_HOP, a rota anunciada e o AS_PATH.

- *BgpeTrapAsDeny*: Reporta problemas decorrentes do atributo AS_PATH. Evita que algum participante repasse anúncios de grandes provedores de *backbone*. Uma situação exemplo seria algum participante anunciar uma rota cujo AS_PATH possui o número 4230, que é o identificador do AS da Embratel. Sabendo que a Embratel não está no PTT, significa que o participante pode estar anunciando um prefixo da Embratel e a conseqüência disso é que este poderá servir de trânsito para o tráfego destinado a Embratel. Baseado nos ASs registrados no objeto *BgpeConfASDeny* da MIB BGPeConf, todos os anúncios são verificados para evitar que os ASs considerados "perigosos" estejam no AS_PATH de algum anúncio. No caso de ocorrer tal evento, é repassado qual o AS que anunciou, a rota e qual o AS_PATH.

- *BgpeTrapRouteServer*: Reporta os problemas nos *Route Servers*, como alto grau de utilização de CPU, memória, processos ou ainda possíveis tentativas de acesso indevido. Essa verificação é feita pela comparação dos dados esperados informados pela MIB BGPeConf e as informações disponíveis na BGPeRSUtil. Retorna a descrição do evento, bem como as informações coletadas que possam auxiliar na resolução do problema. Esse cuidado com os *Route Servers* deverá ser contínuo visto que se o *Route Server* parar, o PTT acaba parando também.

A descrição desta MIB resultante, bem como a especificação dos tipos e estruturas de dados a serem retornados em linguagem ASN.1 encontra-se no Anexo B. A relação dos nomes de objetos e seus respectivos OIDs encontra-se no Anexo G.

5.3.3 MIB BGPe Conf

Esta MIB é responsável pela determinação dos dados de cada participante e do PTT de forma global. Através desses, dados são repassados ao sistema informações necessárias para coleta e processamento dos dados para serem disponibilizados pela BGPe. Também serão utilizados essas informações para a determinação dos parâmetros que definem comportamentos anormais ou aceitáveis, permitindo que a BGPe Trap gere *traps* ou alarmes. Para setar tais informações, pode ser utilizado um MIB *Browser*, utilizando para isso uma *community* com permissão de escrita e leitura.

A Figura 5-9 mostra os objetos dessa MIB.

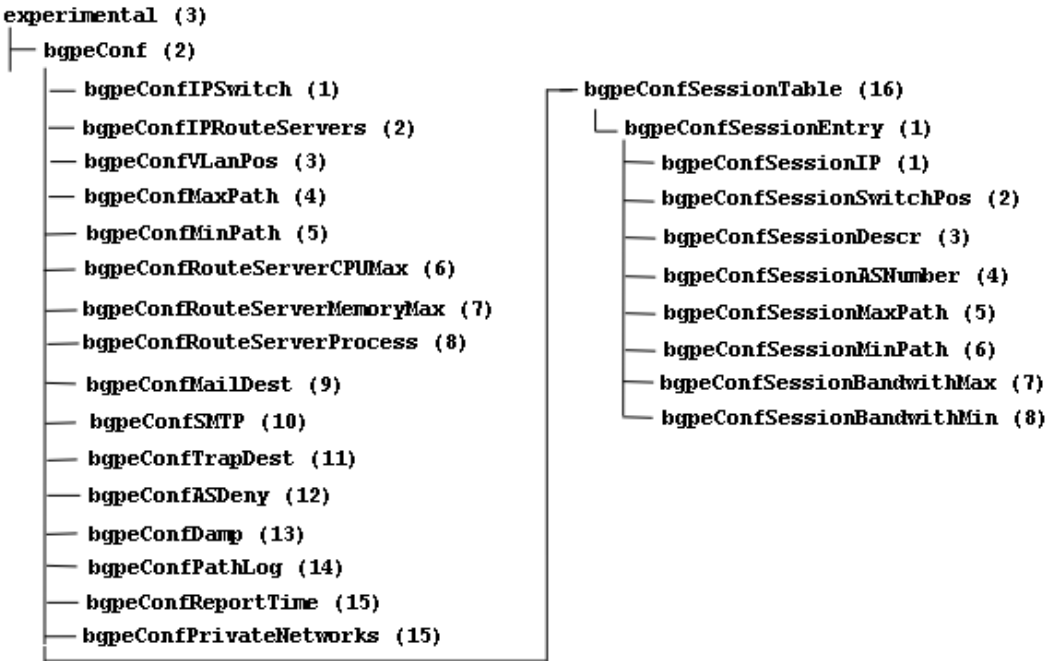

Figura 5-9 - Ilustração da MIB de configuração do sistema

Os objetos contidos nessa MIB são:

- *BgpeConfIPSwitch*: Determina o endereço IP do *Switch* do PTT.
- *BgpeConfIPRouteServers*: Determina o endereço IP dos *Routes Servers*.
- *BgpeConfVlanPos*: Determina a referência da Vlan no *Switch* que engloba todas as portas dos participantes do PTT.
- *BgpeConfMaxPath*: Determina o número máximo de prefixos aceitável no PTT.
- *BgpeConfMinPath*: Determina o número mínimo aceitável de prefixos do PTT.
- *BgpeConfRouteServerCPUMax*: Determina a utilização de CPU máxima esperada dos *Route Servers*.
- *BgpeConfRouteServerMemoryMax*: Determina a utilização de memória máxima esperada dos *Route Servers*.
- *BgpeConfRouteServerProcessMax* Determina o número máximo de processos esperados dos *Route Servers*.
- *BgpeConfMailDest*: Determina e endereço de e-mail de destino da equipe de suporte ao PTT.

- *BgpeConfSMTP*: Determina e endereço IP do SMTP para postagem dos e-mails de alarmes.

- *BgpeConfTrapDest*: Determina e endereço IP da estação de gerência que recebe as *TRAPs*.

- *BgpeConfASDeny*: Determina o número dos ASs que não deverão constar em nenhum campo AS_PATH de anúncios provenientes de participantes do PTT.

- *BgpeConfDamp*: Ativa o envio de alarmes quando alguma rota entrar em *dampening*.

- *BgpeConfPathLog*: Determina a localização do arquivo geral de *log* do sistema no *host* que a ferramenta estiver sendo executada.

- *BgpeConfReportTime*: Determina o intervalo de tempo que deve ser enviado um relatório sobre as atividades mais importantes do PTT.

- *BgpeConfPrivateNetworks*: Determina as redes privadas que não devem estar incluídas nos anúncios dos participantes dos PTTs.

- *BgpeConfSessionTable*: Tabela que tem os dados importantes de cada participante. Estes dados são importantes para o acesso a informações e envio de possíveis alarmes. Para cada um dos participantes, serão preenchidos os objetos *SessionIP*, *SwitchPos*, *Descr*, *AsNumber*, *MaxPath*, *MinPath*, *BandwithMax* e *BandwithMin*, que significam respectivamente: IP do roteador utilizado na sessão BGP, posição no *Switch*, descrição para identificar o participante, número do AS, número máximo e mínimo de anúncios esperados e banda do circuito até o PTT.

A descrição da MIB BGPe Conf resultante, bem como a especificação dos tipos e estruturas de dados a serem retornados em linguagem ASN.1, encontra-se no Anexo C. A relação dos nomes de objetos e seus respectivos OIDs encontra-se no Anexo G.

Os objetos da BGPeConf são gravados em um arquivo de texto. A Figura 5-10 mostra a configuração atual da BGPeConf.

```
#MIB BgpeConf
#Tue Oct 01 17:16:19 BRT 2002
bgpeConfIPSwitch=10.30.143.7
bgpeConfIPRouteServers.0=10.30.143.6
bgpeConfVLanPos=13
bgpeConfMaxPath=50
bgpeConfMinPath=10
bgpeConfRouteServerCPUMax=50
bgpeConfRouteServerMemoryMax=100000
bgpeConfRouteServerProcessMax=99
bgpeConfMailDest.0=ji202379@inf.pucrs.br
bgpeConfMailDest.1=andrey@penta.ufrgs.br
bgpeConfMailOrig=bgpesystem@inf.pucrs.br
bgpeConfSMTP=smtp.inf.pucrs.br
bgpeConfTrapDest=10.30.143.254
bgpeConfASDeny=4230
bgpeConfDamp=yes
bgpeConfPathLog=/var/log/bgpelog
bgpeConfReportTime=100
bgpeConfPrivateNetworks.0=10.0.0.0
bgpeConfPrivateNetworks.1=192.168.0.0

bgpeConfSessionIP.0=10.30.143.2
bgpeConfSessionSwitchPos.0=1
bgpeConfSessionDescr.0=Participante 1
bgpeConfSessionASNumber.0=1
bgpeConfSessionMaxPath.0=5
bgpeConfSessionMinPath.0=0
bgpeConfSessionBandwithMax.0=1000
bgpeConfSessionBandwithMin.0=100

bgpeConfSessionSwitchPos.1=2
bgpeConfSessionIP.1=10.30.143.3
bgpeConfSessionDescr.1= Participante 2
bgpeConfSessionASNumber.1=2
bgpeConfSessionMaxPath.1=5
bgpeConfSessionMinPath.1=0
bgpeConfSessionBandwithMax.1=1200
bgpeConfSessionBandwithMin.1=100

bgpeConfSessionIP.2=10.30.143.4
bgpeConfSessionSwitchPos.2=1
bgpeConfSessionDescr.2= Participante 3
bgpeConfSessionASNumber.2=3
bgpeConfSessionMaxPath.2=10
bgpeConfSessionMinPath.2=0
bgpeConfSessionBandwithMax.2=1000
bgpeConfSessionBandwithMin.2=100

bgpeConfSessionSwitchPos.3=2
bgpeConfSessionIP.3=10.30.143.5
bgpeConfSessionDescr.3= Participante 4
bgpeConfSessionASNumber.3=4
bgpeConfSessionMaxPath.3=9
bgpeConfSessionMinPath.3=0
bgpeConfSessionBandwithMax.3=1800
bgpeConfSessionBandwithMin.3=100
```

Figura 5-10 - Arquivo da BgpeConf

Este arquivo de configuração dará subsídios ao sistema para seu funcionamento. O arquivo apresentado foi baseado nas configurações do protótipo de PTT apresentado na seção 5.6.1 através da Figura 5-28.

5.3.4 MIB BGPe RSUtil

Essa MIB será totalmente implementada e será baseada nos *logs* extraídos dos *Route Servers*. Os *Route Servers* tem o poder de controlar diversas informações, como por exemplo, o número máximo e mínimo de anúncios de determinado participante, entradas/saídas de anúncios de rotas que podem resultar em *dampening*, aspectos de segurança e estabilidade dos próprios *Route Servers*, entre outros. Esses *logs* são explorados e reportados por essa MIB. A Figura 5-11 apresenta os objetos da MIB.

Figura 5-11 - BGPeRSUtil

Os objetos desta MIB são:

- *bgpeRSUtilDampennedNetworks*: Representa as redes que estão incluídas em *dampening*.

- *bgpeRSUtilMaxAnnounc*: Representa o número do AS cujo número de anúncios máximos foi excedido. Essa informação é obtida mediante a análise do LOG do software Zebra. Nos casos que o *software* está configurado para limitar o número máximo de conexões, se o limite for ultrapassado a sessão BGP será derrubada e será registrado LOG. A partir daí, é fornecido o número do AS envolvido nesse evento.

- *bgpeRSUtilRouteServerCPU*: Representa informações sobre a utilização de CPU nos *Route Servers*.

- *BgpeRSUtilRouteServerMemory*: Representa informações sobre a utilização de memória nos *Route Servers*.

- *BgpeRSUtilRouteServerProcess*: Representa informações sobre o número de processos que estão sendo utilizados nos *Route Servers*.

- *bgpeRSUtilRouteServerLogGeneral*: Representa os *logs* que devem ser reportados a equipe do PTT referentes a segurança do *Route Server*, como *logs* de TCPWrappers, tentativas de acesso SSH, entre outras.

No anexo D encontra-se a ASN.1 dessa MIB. A relação dos nomes de objetos e seus respectivos OIDs encontra-se no Anexo G.

5.4 Modelagem de Entidades SNMP

Como o sistema irá definir novas MIBs, um agente SNMP será implementado para suportar essas MIBs. Para isso, será utilizada a modelagem que foi apresentada na Figura 4-15, que define uma entidade SNMPv3. Mesmo que não seja implementada uma entidade da terceira versão, é bom que seja seguido esse esquema, pois ele tem suporte às versões anteriores de SNMP. A entidade é definida em módulos, sendo que alguns são necessários somente no agente ou no gerente. Isso facilita também a criação de entidades Dual, apenas implementando módulos específicos.

A seguir, na Figura 5-12, é apresentada a comunicação entre os módulos de um agente SNMP.

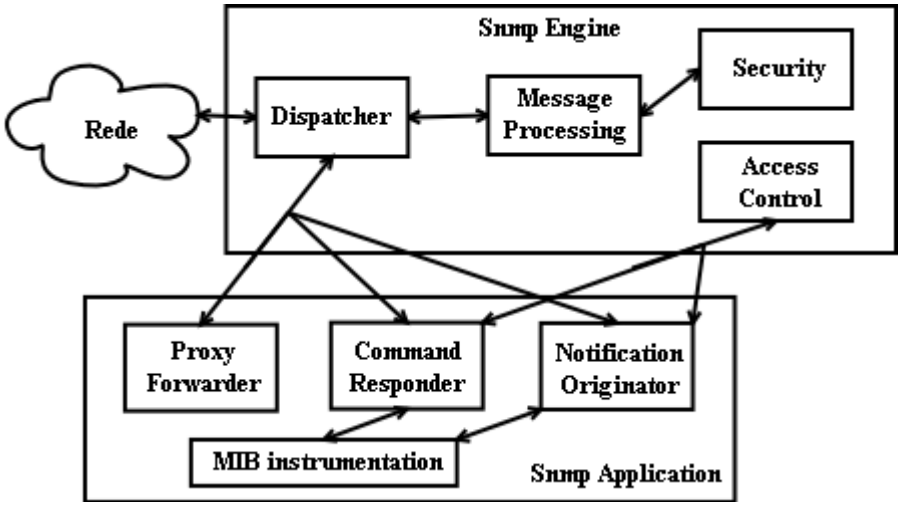

Figura 5-12 – Comunicação entre os módulos de um Agente SNMP

Existem módulos que são exclusivos ao agente. Tais módulos são apresentados abaixo:

- *ProxyForwarder*: Atua no repasse de mensagens SNMP.
- *CommandResponder*: Atua na resposta à requisições SNMP. Além do mais, o *CommandResponder* necessita do módulo *AccessControl* para controlar o acesso as MIBs.

Na Figura 5-13 é apresentado o gerente.

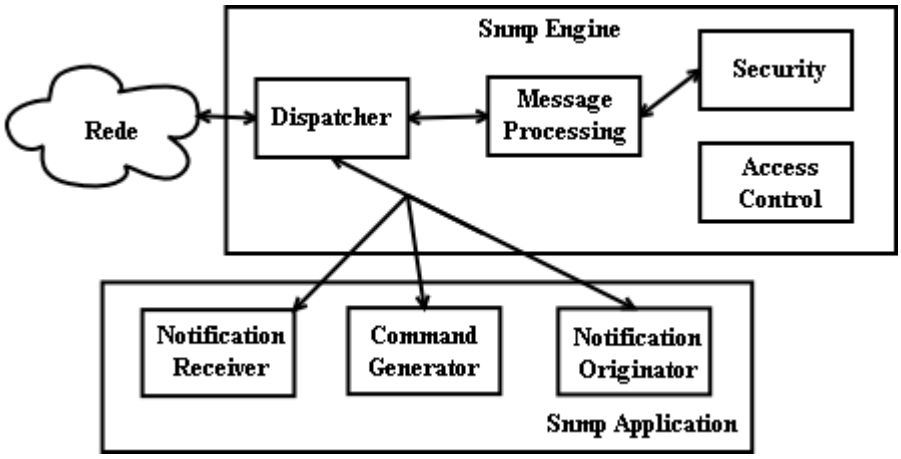

Figura 5-13 – Comunicação entre os módulos de um Gerente SNMP

Existem módulos que são exclusivos ao gerente. Tais módulos são apresentados abaixo:

- *Notification Receiver*: Atua no recebimento de mensagens do tipo *notification*.
- *CommandGenerator*: Atua na geração de requisições SNMP.

Como os módulos são bem definidos, a criação de uma entidade Dual, que tenha o papel de gerente e agente é feita incluindo todos os módulos de agente e gerente ao nível *Snmp Application*. Já o nível *Snmp Engine*, que é o mesmo para ambos, nenhuma modificação é necessária.

5.5 Implementação do sistema

A modelagem das entidades SNMP baseou-se em [STA99], que aborda detalhadamente este tipo de entidade. A implementação foi feita usando a linguagem de programação Java, utilizando a versão 1.4 do *Java Development Kit*. Utilizou-se a segunda vesão do protocolo SNMP visto que a diferença para a primeira versão é mínima, e em relação a versão três toda a parte de criptografia deveria ser implementada.

Para a implementação do sistema, foram criadas duas entidades, a citar: entidade Dual e Agente. Como a diferença de operações entre elas é mínima, como foi visto na seção 5.4, implementou-se um modelo orientado a objetos, onde a diferença é de apenas algumas classes, sendo que a estrutura básica das entidades pode ser considerada a mesma. O diagrama de classes completo está no Anexo H.

A seguir serão apresentadas as principais classes que formam as entidades Dual e Agente.

5.5.1 Classe Transport

A comunicação das entidades SNMP é feita usando o protocolo UDP. Este protocolo não é confiável, ou seja, não garante que a mensagem chegue ao seu destino e nem confirma seu recebimento. Entretando, como a troca de mensagens SNMP sempre tem uma resposta, no caso do não recebimento desta, a requisição será enviada novamente.

A porta de comunicação padrão do SNMP é a 161 para o recebimento das mensagens, exceto a operação *Trap* que usa a porta 162. Para isso, criou-se uma *Thread* que irá mandar e receber os bytes das mensagens. Para a implementação foram criadas três classes que cuidam da comunicação. Estas classe estão representadas no diagrama da Figura 5-14 com o seus relacionamentos.

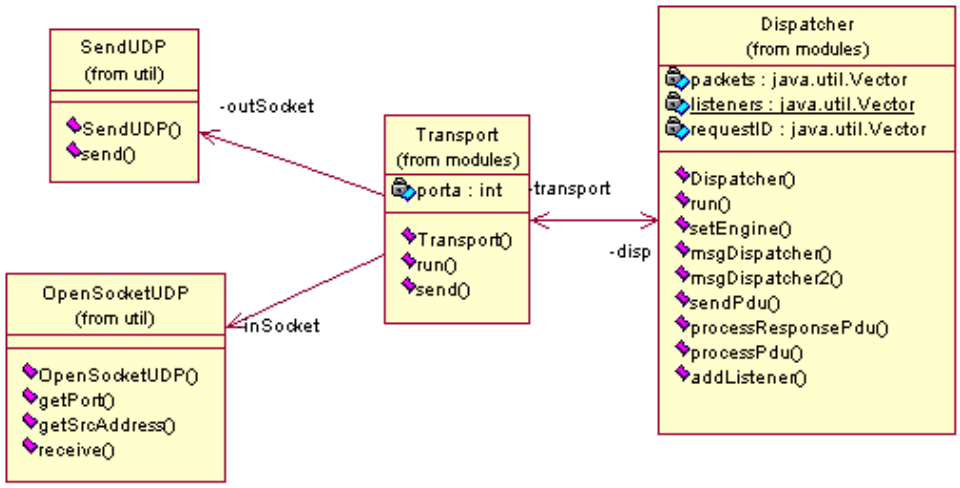

Figura 5-14 – Classe Transport

A classe *Transport* possui uma *Thread* que fica recebendo os bytes que chegam pela rede através da chamada no método *receive()* da classe *OpenSocketUDP*. Quando uma mensagem chegar, esta é enviada para a classe *Dispatcher*. Esta classe será detalhada na seção 5.5.2. Para enviar uma mensagem, é chamado o método *send()* da classe *Transport* que irá repassar para a classe *SendUDP* através do método *send()* que tem como parâmetro o IP destino, porta destino e os bytes da mensagem a serem enviados.

5.5.2 Classe Dispatcher

A classe *Dispatcher* é responsável por receber as mensagens da classe *Transport*, além de colocá-las em uma fila para esperar pelo tratamento e também em receber as mensagens que devem ser enviadas para a rede. A Figura 5-15 mostra como a classe *Dispatcher* se relaciona com as outras classes.

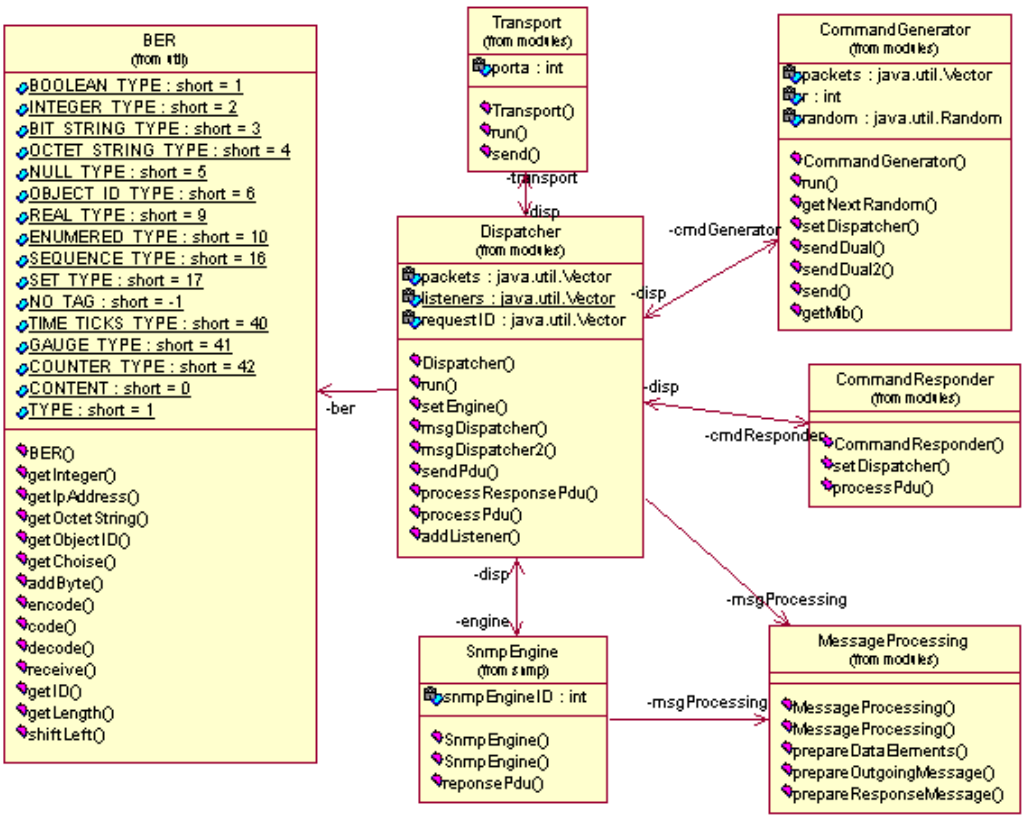

Figura 5-15 – Classe Dispatcher

No momento do recebimento de uma mensagem SNMP, a classe *Dispatcher* recebe da *Transport* os bytes dessa mensagem, enfileirando-os para posterior tratamento. Enquanto isso, uma *Thread* na classe *Dispatcher* fica verificando se existe alguma mensagem na fila para ser tratada. Em caso positivo, a classe *Dispatcher* retira a mensagem da fila para tratá-la.

O tratamento da mensagem pela classe *Dispatcher* inicia com os bytes da mensagem sendo decodificados usando a classe *BER*. Como resultado, o *Dispatcher* tem agora os valores na forma de duas classes, *Message* e *PDU*. Ele então faz a extração da *PDU* que esta contida na mensagem através da classe *MessageProcessing*. Neste momento a *PDU* pode tomar dois caminhos, se ela for uma requisição, ou se for uma resposta.

Se for uma requisição, a *PDU* é enviada para a classe *CommandResponder*. Se for uma resposta, a *PDU* é enviada para *SnmpEngine* e para a classe *Pendente*. O papel da classe *SnmpEngine* é comunicar ao gerente o recebimento de uma *PDU* de resposta. As classes CommandResponder e Pendente serão explicadas nas seções 5.5.3 e 5.5.5.

A classe *Dispatcher* também participa do envio de mensagens. Ela recebe *PDUs* da classe *Pendente* e da classe *CommandGenerator*. Então ela cria uma classe *Message* que irá encapsular essa classe *PDU*. A última tarefa da *Dispatcher* é enviar as classes *Message* e *PDU* para serem codificadas de acordo com o padrão *BER*. Assim, a

mensagem estará totalmente codificada em uma seqüência de bytes que será repassada para *Transport* juntamente com o IP e a porta destino.

5.5.3 Classe Pendente

Esta classe, apesar de não estar prevista na modelagem presente em [STA 99], foi necessária pela razão que a solicitação de algumas variáveis exige um processamento especial, ou seja, fazer o envio de outras requisições para então processar tais valores e formar a resposta.

A solução encontrada foi criar a classe *Pendente* que implementa as seguintes filas:

- Uma fila que possui todas as *Pdus* vazias que estão esperando para serem preenchidas. Essas *Pdus* são criadas pela MIB, no momento em que o *CommandResponder* faz a requisição por um valor.
- Outra fila chamada de *Listeners* que contém os métodos que serão chamados quando todos os valores que um objeto necessite chegarem.
- E também uma fila que contém as respostas para as requisições feitas por essa entidade.

A Figura 5-16 mostra o funcionamento básico da classe *Pendente*.

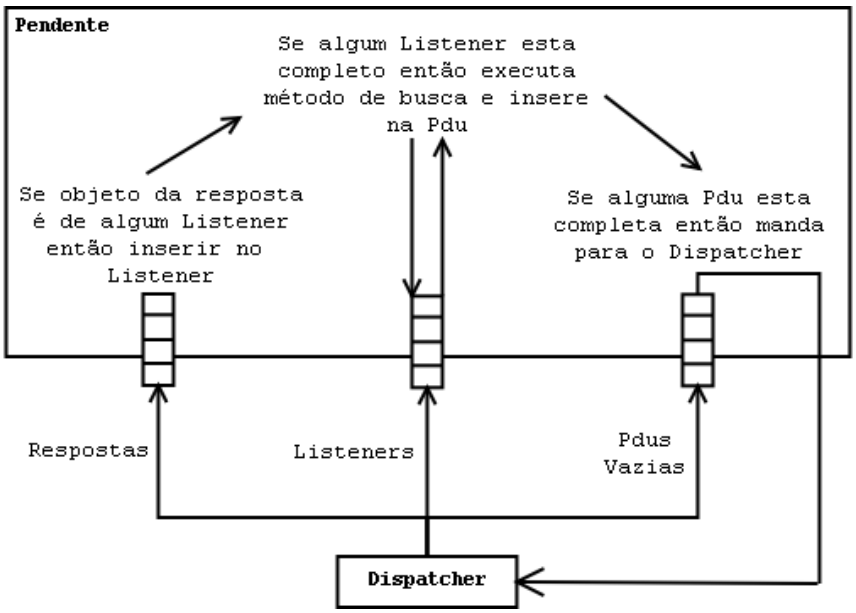

Figura 5-16 – Filas da classe Pendente

A classe *Pendente* então verifica se algum método já possui todas as variáveis que precisa. Em caso positivo, as processa e grava o valor na *PDU* que o requisitou. Quando uma *PDU* estiver com todos os objetos preenchidos, esta *PDU* é enviada para o *Dispatcher* que a envia para a origem.

5.5.4 Classe CommandGenerator

Esta classe é responsável em iniciar as mensagens de *Get*, *GetNext*, *GetBulk* e *Set*. Ela também vai processar as respostas que vierem dessas requisições. O seu uso mais evidente é no caso de uma aplicação Gerente, pois acessa diretamente o *CommandGenerator* para mandar requisições.

Um outro uso é feito pela MIB, pois alguns objetos necessitam de objetos de outros agentes, então a MIB repassa uma requisição para esta classe que inicia o tratamento da mensagem para ser enviada. Esta classe é justamente a diferença entre a entidade Dual e a entidade Agente.

5.5.5 Classe CommandResponder

A classe *CommandResponder* recebe de *Dispatcher* as seguintes operações: *Get*, *GetNext*, *GetBulk* e *Set*. A Figura 5-17, mostra a classe e seus relacionamentos.

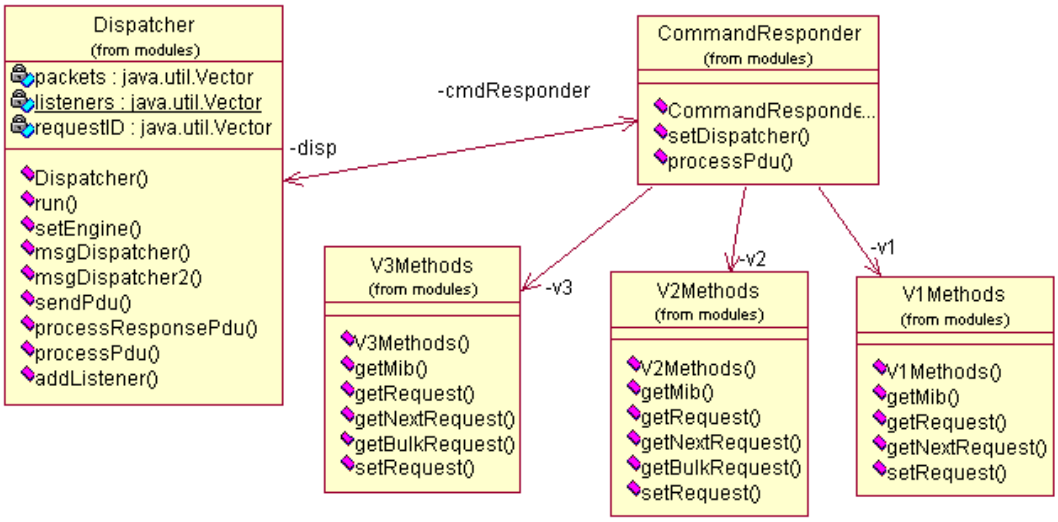

Figura 5-17 – Classe CommandResponder

A classe *CommandResponder* recebe as solicitações SNMP da classe *Dispatcher* e primeiro verifica qual operação está sendo solicitada. Repassa, então, as informações para a classe responsável. Se for SNMPv1, repassa para *V1Methods*, se for SNMPv2, repassa para *V2Methods* e se for SNMPv3 repassa para *V3Methods*. Essas classes recebem os dados da *PDU* e executam as operações necessárias, de acordo com a versão especificada.

É importante lembrar aqui, que apesar do trabalho estar baseado na segunda versão do protocolo SNMP, também possui suporte para a primeira versão. Já a terceira versão possui as classes modeladas, faltando apenas a implementação, que não será feita nesse trabalho.

5.5.6 Classes V1Methods/V2Methods/V3Methods

Para a realização das operações SNMP, foi criada uma classe para cada versão do protocolo. A Figura 5-18 mostra o diagrama de classes dessas classes.

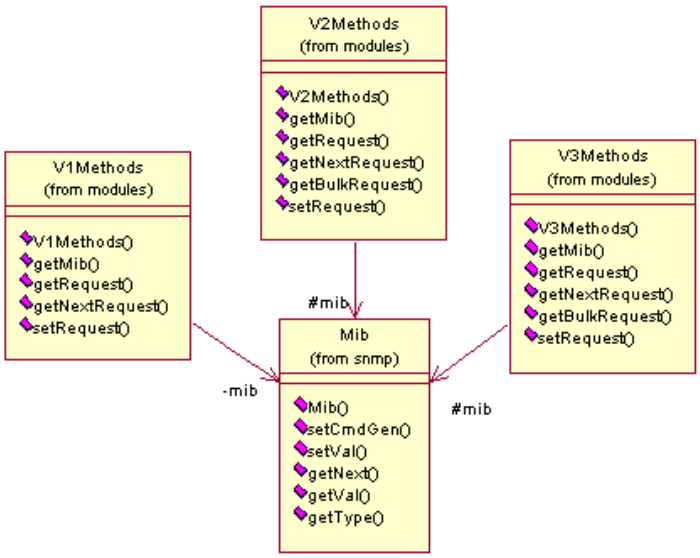

Figura 5-18 – Métodos da classe VXMethods

Como o programa implementado usa a segunda versão do protocolo, não é explicado o funcionamento das demais versões, apesar de estar funcionando também a primeira versão. Os métodos implementados por *V2Methods* estão representados na Figura 5-19.

```
1  package snmp.modules;
2
3  import snmp.*;
4  import snmp.pdu.*;
5  import snmp.util.*;
6
7  /**
8   * Class V2Methods, implementa os metodos v2
9   */
10 public class V2Methods {
11         private Mib mib;
12
13         public Pdu getRequest(Pdu pdu) { ... }
14         public Pdu getNextRequest(Pdu pdu) { ... }
15         public Pdu getBulkRequest(Pdu pdu) { ... }
16         public Pdu setRequest(Pdu pdu) { ... }
17 }
```

Figura 5-19 – Métodos da classe V2Methods

O *CommandResponder* chama o método adequado da classe *V2Methods* de acordo com a operação requisitada na mensagem SNMP. Deve-se passar no método a *PDU* da mensagem. Esta operação vai criar uma cópia dessa *PDU*, pois a resposta muda pouco em relação à requisição. As informações alteradas são a identificação da operação e os valores referentes aos objetos especificados na *PDU* a ser preenchidas. Essas operações iniciam a busca dos objetos através da MIB. Enquanto isso, a *PDU* de

retorno, que é a resposta, estará vazia e deve ser colocada em uma fila da classe *Pendente* para ser preenchida posteriormente.

5.5.7 Classe MIB

A classe MIB implementa os métodos que são necessários para a busca dos valores nos objetos. Para isso, implementou-se primeiro um compilador de MIBs, que recebe como entrada as RFCs das MIBs e tem como saída um conjunto de classes para aquela MIB. A explicação de como o compilador de MIBs é implementado é feita na seção 5.5.8.

A primeira classe a ser apresentada é a classe MIB, que faz a interface entre o programa e as MIBs. Ele possui um relacionamento com o grupo Internet e, a partir desse, consegue chegar em qualquer grupo abaixo dele.

5.5.7.1 Estrutura da MIB

Inicialmente, pensou-se em implementar a MIB como sendo apenas um método, onde teria uma estrutura de seleção, que faria a comparação e chegaria ao método que implementa esta variável. Mas isto se mostrou ineficaz, pois uma MIB, em geral, possui muitos objetos e essa estrutura ficaria grande e lenta.

Então se projetou o modelo estruturado em forma de árvore onde cada nodo folha é um objeto e os ramos seriam os caminhos até chegar ao nodo. Assim, chega-se facilmente ao objeto requerido através do seu ID. A Figura 5-20 mostra um exemplo de busca.

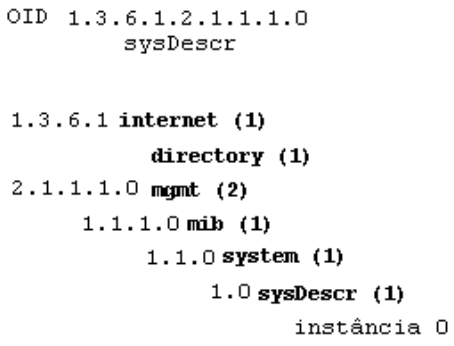

Figura 5-20 - MIB Exemplo

Neste exemplo, requisita-se o objeto *sysDescr* da MIB-II. O OID desse objeto é 1.3.6.1.2.1.1.1.0, para facilitar, a pesquisa começa no nodo *Internet* da MIB, assim, o começo 1.3.6.1 é cortado. O próximo número do OID é 2, então *Internet* olha para o seu filho número 2, *Mgmt*, então repassa a requisição para ele e o resto do OID. *Mgmt* repassa até o filho número 1, que é a MIB-II. MIB-II então repassa para o grupo *system,* pois o próximo número é 1. Então chega-se ao primeiro filho do grupo *system* que é o *sysDescr,* que descarta a sua identificação, no caso o número 1. O que sobrou de OID é a instância do objeto que foi requerida, no caso a instância zero.

Como já foi dito anteriormente, chegando na classe do nodo folha, um simples método de *Get,* por exemplo, poderia retornar o objeto requisitado preenchendo a *PDU* e enviando de volta a sua origem. Mas como o trabalho tem objetos que precisam fazer um processamento em objetos de outras MIB em diferentes agentes, criou-se a classe *Pendente* para resolver tal problema.

O método de *Get* é então chamado, enviando as requisições para os outros agentes se for preciso, e cria um *Listener*. Enquanto isso, é criada uma *PDU* de resposta, ainda vazia, que é inserida na fila de *PDUs*. Este *Listener* possui os OIDs dos objetos que são necessários para fazer o processamento e o método que irá processar esses valores. Esse *Listener* é inserido em uma fila da classe *Pendente*.

Quando chegarem todos os objetos necessários, a classe *Pendente* então chama o método *getValue*, pertencente a classe *MibListener*, que faz o processamento e tem como retorno o valor do objeto. Esse valor é inserido na *PDU* que foi criada anteriormente. Então, *Pendente* verifica se a *PDU* pode ser enviada pela rede, pois uma *PDU* pode requisitar mais que uma variável. No momento que ela for totalmente preenchida, esta será enviada para a sua origem.

5.5.7.2 Implementação de MIBs

Na implementação de MIBs utilizadas no sistema, temos dois tipos de classes. As classes que pertencem aos grupos da MIB, ou seja, os ramos da árvore, e as classes dos nodos folhas, ou seja, aqueles que precisam retornar um valor para a variável. Todo grupo possui uma classe com o mesmo nome e que estende a classe *MibModel*. Na Figura 5-21 é mostrada a classe da MIB BGPe.

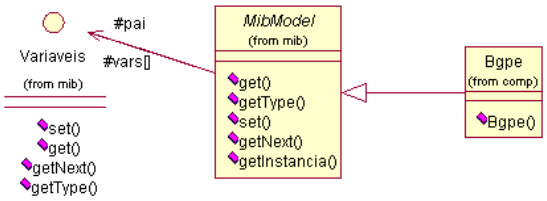

Figura 5-21 – Classes empregadas na implementação de grupos (exemplo Bgpe)

Surge também a *interface Variáveis*, implementada por todas as classes pertencentes as MIBs, possuindo métodos de acessos a MIB como mostra a Figura 5-22.

```
1 package snmp.util.mib;
2
3 public interface Variaveis {
4         public void set( String instancia , Object val );
5         public void get( String oid );
6         public Object[] getNext( String oid );
7         public Object getType( String oid );
8 }
```

Figura 5-22 –Métodos da Interface Variáveis

Quando for chamado o *get()* em uma classe de algum grupo, como por exemplo, a BGPe, é feita uma busca entre os seus nodos filhos. Encontrando o nodo,

será repassada a chamada para ele. Como essa busca é genérica, foi criada a classe abstrata *MibModel* que implementa esses métodos.

A implementação dos objetos folhas, ou seja, aqueles que possuem alguma implementação é um pouco diferente, a Figura 5-23 mostra, por exemplo, como é implementado o primeiro objeto da *BGPe*, o *BgpeInOctets*.

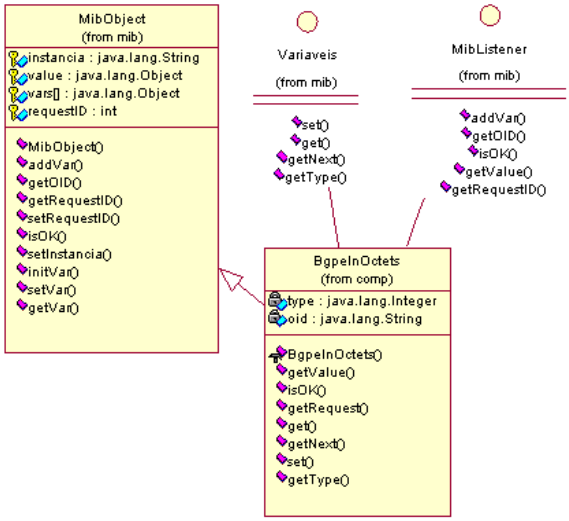

Figura 5-23 – Exemplos de classes tipo Nodo Folha

Os nodos filhos estendem a classe *MibObject,* que por sua vez, implementa a *interface Variaveis* para fazer as operações no objeto.

Os nodo folhas são manipulados na classe *Pendente* através da interface MibListener. A Figura 5-24 mostra as operações definidas em *MibListener*.

```
1  package snmp.util.mib;
2
3  public interface MibListener {
4          /**
5           * add alguma variavel q esta faltando para o processamento
6           */
7          public void addVar( String id , Object val );
8          /**
9           * retorna as variaveis, com os seus valores logo em seguinda,
10          * q sao necessarias ao processamento
11          */
12         public Object[] getOID();
13         /**
14          * verifica se todas as variaveis ja estao preenchidas
15          */
16         public boolean isOK();
17         /**
18          * retorna o valor processado
19          */
20         public Object[] getValue();
21         /**
22          * retorna o request ID
23          */
24         public int getRequestID();
25 }
```

Figura 5-24 – MibListener

Essa *interface* é usada da seguinte maneira pela classe *Pendente*: quando se faz alguma busca em algum objeto, este objeto cria uma *MibListener* e coloca na fila

dos objetos *Listener* da classe *Pendente*. A partir daí, a classe *Pendente* começa a operar nos seus métodos. A seguir é dada a explicação de cada um dos métodos:

- *addVar()*: insere uma variável que foi requisitada por esse objeto no *Listener*
- *getOID()*: retorna uma array com os objetos que já foram inseridos com o seu valor logo após.
- *isOK()*: retorna verdadeiro se esse *Listener* não precisa de mais nenhuma variável para processar ou falso se precisa.
- *getValue()*: retorna o valor processado, só deve ser chamado depois de chamar *isOK()* retornando verdadeiro.
- *getRequestID()*: se o objeto precisa de uma variável fora da sua MIB, precisa mandar um *Get*. Para saber quando a resposta chegar, guarda-se o seu *requestID* para comparar com as respostas que forem chegando. Este método apenas retorna o *requestID* que esta esperando.

A classe *MibObject* possui a implementação desses métodos de forma padrão. Se a variável precisar implementar alguns métodos com características diferentes, deve-se sobrepor o método.

5.5.8 Classe MIB Compiler

As MIBs usadas em SNMP são definidas nas RFCs. No momento da implementação de um sistema de gerência de redes pode-se optar por passar as MIBs que serão implementadas diretamente pelo programador, ou implementar um compilador de MIBs que irá criar as MIBs diretamente. Essa criação é apenas da estrutura da MIB, o método que fará a busca dos objetos deve ser feito pelo programador nas duas situações.

Foi criado um compilador de MIBs que, com as RFCs das MIBs, cria a estrutura da MIB como foi descrita na seção 2.1.5.1, deixando apenas a ser preenchido os métodos de busca dos valores dos objetos. Para o trabalho, foram compiladas as seguintes MIBs: MIB-II, BGP, BGPe, BGPeConf, BGPeRSUtil e BGPeTrap.

A criação das classes é feita da seguinte maneira: cada objeto da MIB vai ser transformado em uma classe. Quando o objeto for um grupo, como a *BGPe*, essa classe irá estender algumas classes e *interfaces,* como já foi demonstrado, e terá uma referência para o seu nodo pai, no caso, *ExperimentaI.* Essa classe terá ainda um *array* das instâncias dos seus objetos, como por exemplo, *BgepInOctets*. Para não criar muitos arquivos de pequenas classes, alguns só com um pequeno construtor, optou-se em colocar junto com a classe do grupo, os objetos desse grupo na forma de classes internas. A Figura 5-25 mostra o começo da classe *Bgpe*.

```
1  package snmp.util.mib.comp;
2
3  public class Bgpe extends MibModel {
4          public Bgpe( Variaveis pai ) {
5                  this.pai = pai;
6                  vars = new Variaveis[8];
7                  vars[0] = new BgpeInOctets();
8                  vars[1] = new BgpeOutOctets();
9                  vars[2] = new BgpeTotalPath();
10                 vars[3] = new BgpeLen();
11                 vars[4] = new BgpeOrigin();
12                 vars[5] = (Variaveis) new BgpeStates( (Variaveis) this );
13                 vars[6] = (Variaveis) new BgpeTrafficTable( (Variaveis) this );
14                 vars[7] = (Variaveis) new BgpeSessionTable( (Variaveis) this );
15         }
16 }
17 class BgpeInOctets extends MibObject implements Variaveis , MibListener {
18         private Integer type = new Integer( BER.COUNTER_TYPE );
19         private String oid = "1.3.6.1.3.1.1";
20
21         public Object[] getValue( ) { ... }
22         public void getRequest( String instancia ) { ... }
23         public void get( String instancia ) { ... }
24         public Object[] getNext( String instancia ) { .. }
25         public void set( String instancia , Object val ) { .. }
26         public Object getType( String instancia ) { .. }
27 }
28 class BgpeOutOctets extends MibObject implements Variaveis , MibListener {
29 // arquivo continua com as outras variaveis
```

Figura 5-25 – Exemplo de classe de MIB (Exemplo da Bgpe.Java)

Pode-se ver através da Figura 5-25 da classe do *Bgpe*, que a classe pública (em Java, se houver mais do que uma classe no arquivo, apenas uma deve ser pública) é a *Bgpe*. Ele possui um *array* que está na classe *MibModel*, onde estão instanciadas todas as suas variáveis. Logo após, começa a definição dos outros objetos. Se esse objeto for um nodo folha, como a B*gpInOctets*, a classe será no mesmo arquivo. É nessa classe que será implementado o método de busca da variável, no caso o *getValue()*. Se tivermos uma nova definição de uma tabela, ou um novo grupo, como por exemplo, *BgpeStates*, *BgpeTrafficTable* e *BgpeSessionTable,* será criado um novo arquivo de classe seguindo esse padrão apresentado.

5.5.9 Classes Message/Pdu

As Figuras 4-12 e 4-13, anteriormente apresentadas, mostram respectivamente a estrutura das mensagens da primeira e segunda versão do protocolo SNMP. Na implementação, a Figura 5-26 mostra como foram implementadas essas mensagens.

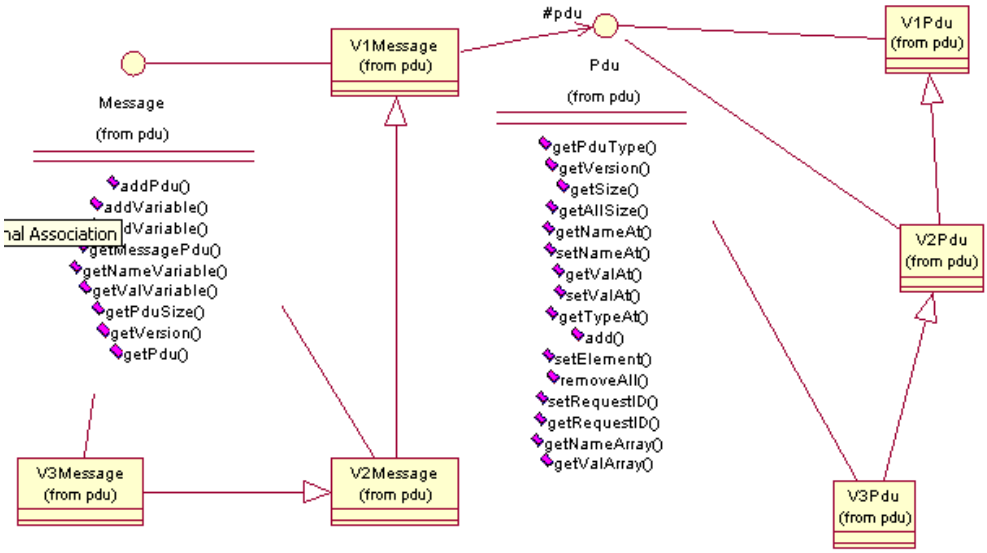

Figura 5-26 – Estrutura das classes Message e Pdu

A explicação da Figura 5-26 é a seguinte: o programa irá manipular diretamente somente as duas interfaces *Messages* e *PDU*. A interface *Messages* possui os métodos necessários para manipular a mensagem, temos então 3 classes de mensagem das 3 versões do protocolo. Todas essas 3 mensagens implementam a *Message* e estendem a classe da sua versão anterior, ou seja, a primeira versão não estende nenhuma classe, a segunda versão estende a classe da primeira versão e a terceira versão estende a classe da segunda versão.

Isso faz um bom reuso de código pois as mensagens possuem algumas similaridades na sua definição. Pode-se notar que a primeira, a *V1Message,* possui um relacionamento com a *PDU*, ou seja, a mensagem encapsula a Pdu. A implementação da *PDU* seguiu o mesmo esquema da *Message*, ou seja, cada um implementando a interface *PDU* e as versões estendendo a sua versão anterior.

É essa estrutura de classes que o BER transforma depois que processa os bytes que chegaram pela rede.

5.5.10 Entidade Agente

O agente no modelo definido pelo SNMP possui a capacidade de responder a requisições de gerentes. Logo ele não precisa gerar requisições, por isso, ele não possui implementado a classe responsável em começar as requisições, a *CommandGenerator*. Esta entidade terá como MIB a *BGPeRSUtil*.

5.5.11 Entidade Dual

A entidade Dual chama-se assim pois possui a capacidade tanto de responder requisições como gerar requisições. Por isso, a diferença entre a Dual e o Agente será justamente na classe *CommandGenerator*. Essa entidade terá como MIBs a BGPe, BGPeConf e BGPeTraps.

5.6 Simulação do sistema

Tendo o intuito de buscar uma forma de simulação de um PTT para testes e implementação deste projeto, observou-se que o Laboratório de Redes da Faculdade de Informática/PUCRS poderia ser utilizado para tal propósito. Atualmente, este laboratório conta com 7 computadores com sistema operacional Linux. Buscando uma ferramenta *freeware* e com os recursos necessários para utilizar o protocolo BGP, optou-se pelo *software* Zebra, descrito anteriormente.

5.6.1 Arquiteturas de validação do sistema

A primeira arquitetura de PTT planejada neste ambiente teve como objetivo principal apresentar todos os componentes de um PTT baseado em acordos multilaterais, como dois *Route Servers*, *Looking Glass* e participantes, que no caso, eram 4. Essa arquitetura serviu para os testes iniciais e início da implementação. A representação desta arquitetura é apresentada na Figura 5-27.

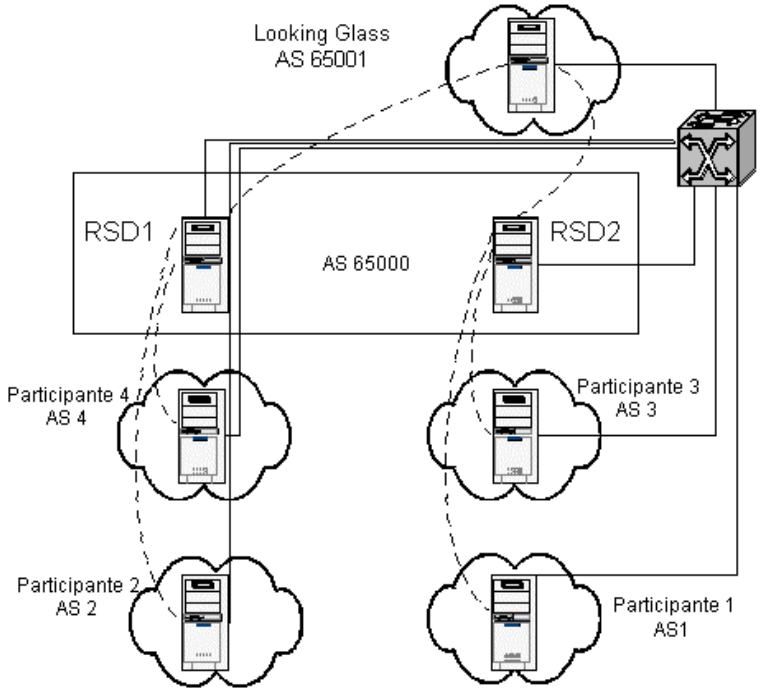

Figura 5-27 - Ilustração do protótipo de PTT no laboratório de redes

Tratando-se de um PTT baseado em *Route Servers* que utiliza acordos multilaterais (ATM), todos os participantes estabelecem sessões BGP com os dois *Route Servers*, que neste protótipo, são representados na ilustração como RSD1 e RSD2, formando o AS 65000.

Adicionalmente, o componente chamado *Looking Glass* finaliza a apresentação do protótipo. Este componente possui sessões com os 2 *Route Servers* e, como já foi explicado anteriormente, demonstra as redes divulgadas pelos participantes.

Todos os detalhes de implementação da arquitetura citada anteriormente como arquivos de configuração do software Zebra dos participantes, *Route Servers* e *Looking glass*, tabelas de sessões BGP, tabelas de rotas BGP, entre outros detalhes, encontram-se no Anexo E.

Uma segunda arquitetura foi implementada com os mesmo recursos, mas com a preocupação de simplificar a arquitetura e manter apenas os componentes básicos e necessários ao funcionamento do PTT afim de facilitar a compreensão do sistema proposto. Também aliou-se o fator de performance de processamento, já que os computadores, em sua maioria são Pentium II. Algumas das simplicações feitas foram:

- Reduzir de dois para apenas um *Route Server*, já que a colocação de um segundo *Route Server* tem o objetivo de redundância, que pode ser dispensável para o protótipo de validação.
- Utilizar um PC apenas para atuar como entidade Dual do sistema, pois é o componente principal da solução.
- Tornar o *Looking Glass* como apenas estação de gerência, visto que o papel do *Looking Glass* em PTTs é exclusivamente para consultas por parte dos participantes sobre seus anúncios e estudos de viabilidade por interessados em ingressar em um PTT.

Tais modificações podem ser visualizadas através da Figura 5-28:

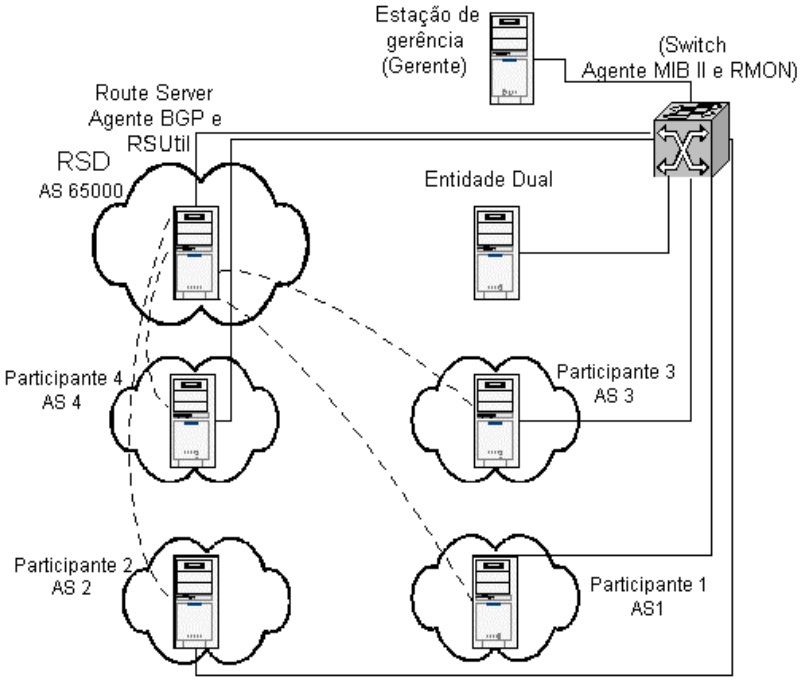

Figura 5-28 - Arquitetura do protótipo final para validação

Uma observação importante é que a entidade Dual e a estação de Gerência encontram-se na mesma rede, mas poderiam estar conectados em qualquer outra rede da Internet, visto que sua comunicação com os agentes é feita via SNMP.

Outro detalhe importante é que as entidades MIB II e RMON, indicadas como presentes no *Switch* do protótipo não são de fato utilizadas, visto que o equipamento atual do presenta no laboratório de redes é um HUB não gerenciável que suporta estas entidades. Em suma, essa deficiência faz com que não sejam utilizados os recursos do agente RMON.

Detalhes de configuração do software Zebra, tabelas de sessões BGP, tabelas de rotas BGP e os blocos de endereços atribuídos a cada participante podem ser encontrados no anexo F.

5.6.2 Softwares utilizados no protótipo

Além do software Zebra, citado como uma das ferramentas utilizadas em PTTs na seção 2.3.3, os seguintes *softwares* foram instalados para possibilitar a implementação do protótipo como um PTT:

- *JDK*: O *Java Development Kit* trata-se de um pacote que contém, entre outras ferramentas, compilador e interpretador para a linguagem Java. Utilizado para rodar os componentes implementados no trabalho.

- *NET-SNMP*: Conhecido também como UCD-SNMP, é uma ferramenta que implementa o daemon SNMPD, utilizado para o acesso e obtenção de informações em determinado *host* via SNMP. Também implementa um cliente para consultas SNMP em outros *hosts*. Possui também outras ferramentas, como um receptor de *traps*. Permite que sejam incorporadas outras MIBs no *daemon* SNMPD, como é o caso da MIB BGP.

- *RRDTool* e *NRG*: Sucessor do popular MRTG, é utilizado para representar graficamente informações obtidas via SNMP. Amplamente utilizado na gerência de redes para verificação de objetos importantes que refletem características de redes ou servidores, facilitando a identificação de eventos anormais.

- *JOpenEyes*: Implementado em Java com recursos gráficos, este programa tem funcionalidades semelhantes a sistemas de gerência comerciais como *Openview*, *Netview*, entre outros. Utilizado principalmente para o recebimento de *traps*.

Nos PCs que representam participantes do PTT, foi instalado apenas a ferramenta Zebra, responsável pela implementação do protocolo BGP. Esse *software*

permite que os participantes se comuniquem com o *Route Server*, possibilitando que sejam trocadas informações referentes ao roteamento BGP.

No *Route Server*, da mesma forma, foi instalado o *software* Zebra, além do *software* NET-SNMP, compatível com o *software* Zebra e consegue disponibilizar, via SNMP, as informações da MIB BGP.

Na Entidade Dual foi instalado apenas a máquina virtual Java para ativá-la.

Por fim, na estação de gerência foi instalado o *software* NET-SNMP para fazer as consultas as MIBs resultantes do trabalho, bem como algumas ferramentas de gerência *freeware* largamente utilizadas em NOCs (*Network Operations Center*), como é o caso do RRDTool, NRG e JOpenEyes.

5.6.3 Ativação do protótipo e sistema de gerência

Como foi visto anteriormente, o gerente é formado por diversos pacotes que permitem que sejam utilizados recursos como gráficos, recebimento de *traps* e verificação de e-mails.

5.6.4 Utilização da MIB BGPe Conf

A MIB BGPe Conf tem como objetivo principal a entrada de informações dos participantes para o sistema organizar e agrupar os resultados de forma concisa. Para alcançar tal objetivo, esta MIB permite tanto a leitura como escrita, ao contrário das demais que possuem apenas o atributo de leitura.

Internamente, em sua inicialização, esta MIB faz a leitura de um arquivo com as configurações iniciais, permitindo que posteriormente sejam feitas as alterações necessárias de duas formas. A primeira delas, mais simples embora mais limitada, pode ser feita através do arquivo de configuração. A outra forma é alterar diretamente na MIB, através de *snmpset*, permitindo que o *SetRequest* seja feito a partir de qualquer local conectado a Internet.

Um exemplo de *snmpset* é dado na Figura 5-29:

```
> snmpset -v 2c localhost public 1.3.6.1.3.2.4.0 i 10
1.3.6.1.3.2.4.0 = 10
```

Figura 5-29 – Exemplo de Set no objeto BgpeConfMaxPath da MIB BgpeConf

A relação entre os objetos existentes e seus respectivos OIDs é dada no Anexo G.

5.6.5 Utilização da MIB BGPe

A MIB BGPe é formada pelos objetos mais importantes para a administração do sistema, possuindo informações globais do PTT e também informações particulares de cada participante. Foi definido que esta MIB possui apenas permissão de leitura, pois todos os objetos são coletados e processados a partir de objetos das entidades definidas, não necessitando qualquer modificação pelo usuário através da MIB.

A maioria dos dados fornecidos por essa MIB são quantitativos, ou seja, fornecem números resultantes que podem ser utilizados, por exemplo, para a geração de gráficos. No gerente, o pacote RRDTool em conjunto com NRG é utilizado para tal recurso. Uma das vantagens de utilizar essa forma de visualização é que através do gráfico é possível visualizar facilmente uma espécie de histórico, permitindo que sejam comparados os valores atuais com os anteriores.

A relação dos objetos e seus respectivos OIDs é apresentada no Anexo G. A partir dessas informações, é possível obter um dado desta MIB, como mostra a Figura 5-30:

```
> snmpget -v 2c localhost public 1.3.6.1.3.1.3.0
1.3.6.1.3.1.3.0 = 19.
```

Figura 5-30 – Exemplo de SnmpGet no objeto BgpeTotalPath da MIB Bgpe

Dessa forma, todos os demais objetos podem ser obtidos da entidade Dual, que é onde a MIB BGPe estará disponível.

5.6.6 Utilização da MIB BGP Traps

Esta MIB é parte importante do trabalho e diferencia-se das demais MIBs pois não fornece apenas as informações para o gerente processá-las, mas a partir das informações disponíveis pela BGPeConf, são geradas *TRAPs*, enviando-as para o gerente. Algumas das informações utilizadas para essa MIB da BGPeConf seriam: IP do gerente a serem enviadas as *TRAPs*, IP, número de AS, número de anúncios e outras informações de cada participante.

O recurso de envio de *TRAPs* é importante pois atua exatamente no momento da ocorrência de comportamentos estranhos. O recebimento das *TRAPs* no protótipo é feito por uma ferramenta instalada no gerente, chamada de *JopenEyes*, que mostra o recebimento de *TRAPs* de forma gráfica, em tempo real.

Adicionalmente, pode ser configurado um endereço de e-mail para que a entidade Dual faça o envio do conteúdo das *traps* ao gerente, via e-mail. Isso pode ser útil caso a equipe de suporte não esteja junto ao gerente. Para isso, existem objetos na MIB BGPeConf para configurar o endereço do SMTP e quais os destinatários dos e-mails.

Por fim, vale lembrar que foram incluídas todas as informações úteis para a solução dos problemas que as *TRAPs* comunicam, pois de nada valeria avisar a equipe de suporte sobre a ocorrência de um comportamento estranho sem fornecer informações úteis para a solução e diagnóstico do problema.

5.6.7 Simulação de problemas em PTTs

Alguns problemas são considerados mais comuns em PTTs devido a sua freqüência e na maior parte das vezes, não são prontamente detectados. Nesse ponto o sistema proposto tem o objetivo de auxiliar na detecção destes problemas. A seguir serão analisados alguns problemas comuns, relacionando a forma normal de detecção do problema e a forma que o sistema oferece para isso.

5.6.7.1 Queda da sessão BGP de algum participante

Relevância e contextualização: A queda da sessão BGP de algum participante faz com que não sejam mais feitos anúncios das redes do participante e por conseqüência, não trocar mais tráfego no PTT. Em ocasiões normais, a verificação do *status* dos participantes seria feita através da console de um *Route Server* ou pela monitoração de objetos da MIB BGP. Ambos os métodos não são eficientes pois apenas sinalizam o evento, necessitando que a equipe de suporte analise continuamente tais valores para diagnosticar problemas.

Solução: O sistema propõe para este problema a monitoração dos participantes configurados na BGPeConf, e no caso de queda de uma das sessões, fazer o envio imediato e uma TRAP sinalizando o sistema autônomo, horário de queda e informações adicionais sobre o problema. Essa solução fornece flexibilidade já que basta incluir o participante na MIB BGPeConf e ele já passará a ser monitorado. Em complemento, a solução também faz o papel de verificação contínua, avisando o gerente quando o evento ocorre.

Exemplo: Na Figura 5-31 é mostrado um exemplo de trap que sinaliza a detecão de queda da sessão BGP de um dos participantes do PTT, no caso o participante descrito como “Participante 2”.

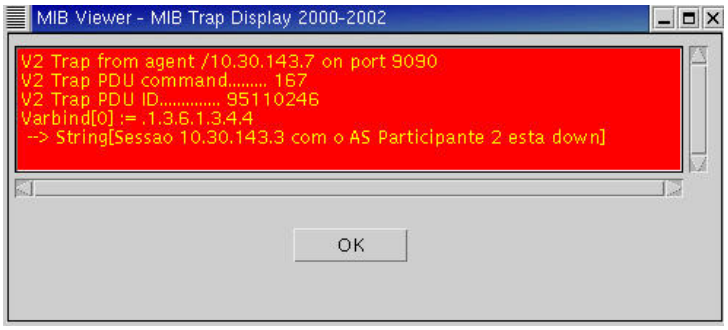

Figura 5-31 – Trap de sessão BGP down do participante 2, recebida pelo software JopenEyes.

Em seguida, a sessão BGP do participante 2 voltou ao estado normal, como sinaliza a trap na Figura 5-32.

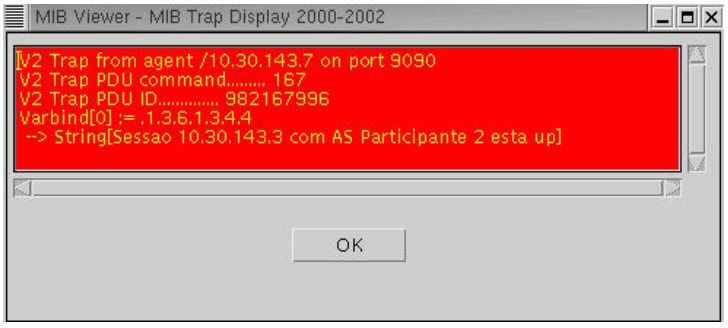

Figura 5-32 – Trap de sessão BGP up do participante 2, recebida pelo software JopenEyes.

Baseado nessas *TRAPs*, o gerente pode facilmente detectar eventos sobre as sessões BGP. Adicionalmente, pode ser enviado via e-mail a notificação, como ilustrado na Figura 5-33.

```
To: andrey@penta.ufrgs.br
From: bgpe@inf.pucrs.br
Reply-to: bgpe@inf.pucrs.br
Apparently-from: bgpe@pop3.inf.pucrs.br
Date: Mon, 20 Nov 2002 13:47:38
Subject: Sessão BGP – Participante 2 (10.30.143.3) up
X-ECS-MailScanner: Found to be clean

> Sessao 10.30.143.3 com AS Participante 2 esta up
```

Figura 5-33 – Corpo do e-mail que sinaliza a normalização da sessão BGP do participante 2.

5.6.7.2 Número de anúncios fora do intervalo esperado

Relevância e contextualização: Este problema pode ser muito comum em PTTs e deve ser rapidamente detectado para verificar o que está provocando tal ação. Em geral, quando um participante entra em um PTT, já se planeja quantos blocos estarão sendo anunciados e, a partir daí, podem ser aplicados filtros nos *Route Servers* para evitar que um número não esperado de anúncios seja repassado aos demais participantes. Mesmo com esses filtros, o problema acaba sendo apenas remediado, visto que os anúncios em excesso continuarão a ocorrer, sendo descartados pelo *Route Server*.

Solução: O que é proposto é o estabelecimento de um número mínimo e máximo de anúncios a cada participante através da MIB BGPeConf. Supondo então que não existam filtros no *Route Server*, a entidade Dual fará a verificação, comunicando por intermédio de uma *TRAP* ao gerente a ocorrência de um número de anúncios fora do intervalo esperado. Adicionalmente, caso o *Route Server* possuir filtros e de fato não permitir que os anúncios sejam propagados, através da MIB RSUtil presente no *Route Server* será possível obter a informação de qual AS está ultrapassando o número de anúncios e tomar as devidas providências. Essa informação é feita através do envio de uma TRAP.

Exemplo: Afim de ilustrar este tipo de problema, o parâmetro BgpeConfSessionMaxPath.1 recebeu o valor 5, ou seja, que o AS 2 (cujo índice na MIB BgpeConf é 2), deveria fazer no máximo 5 anúncios de blocos. Por outro lado, o AS 2 foi configurado para anunciar 8 blocos afim de provocar o envio de uma T*rap* pelo sistema avisando que o número de anúncios ultrapassava o máximo de 5.

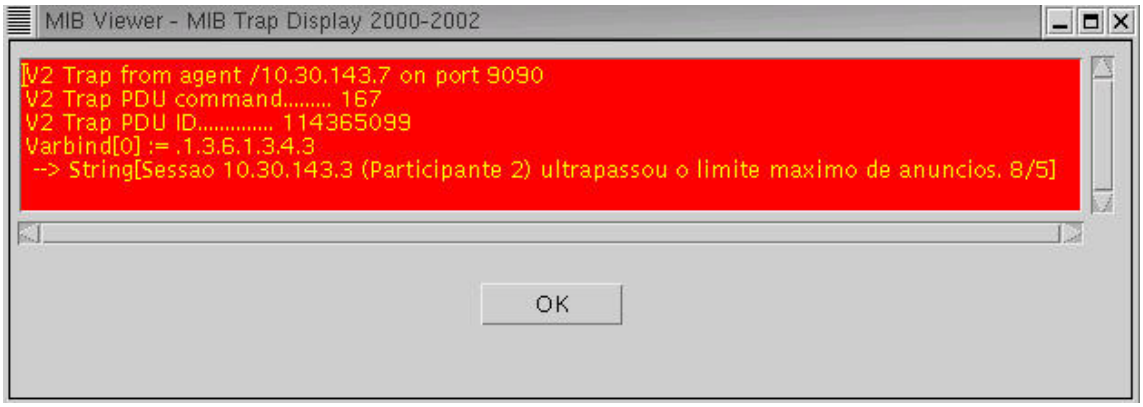

Figura 5-34 - Exemplo de trap enviada quando o número de anúncios ultrapassa o esperado.

De acordo com a Figura 5-34, a *Trap* avisa o gerente no momento que o excesso de anúncios ocorre, informando o IP do participante, a descrição deste, o número de anúncios atual e o número de anúncios esperado.

5.6.7.3 Número de anúncios e histórico de anúncios de participantes

Relevância e contextualização: O número blocos anunciado por cada participante é um dado importante para analisar o comportamento de um AS em um PTT. Através da MIB BGP não é possível obter o total de blocos anunciados por determinado AS, sendo possível obter, no máximo, o conjunto total de anúncios no PTT por todos os participantes. A única forma de obtenção dessas informações, seria através da console do software Zebra que através do comando "show ip bgp summary", fornece informações, entre estas, o total de blocos anunciados. Mesmo com isso, não é possível manter nenhum histórico sobre essa informação de cada participante.

Solução: O que é proposto, nesse caso, é o fornecimento de objetos da MIB BGPe, chamados BGPeTotalPath e BGPeSessionTotalPath que fornecem, respectivamente, o total de anúncios global e de cada participante do PTT. Isso resolve tanto o problema da acessibilidade, já que o dado poderá ser obtido via SNMP, quanto o problema de histórico, visto que com ferramentas de gerência como o RRDTool é possível fazer consultas via SNMP e gerar gráficos a partir destes dados diários, semanais, mensais e até anuais.

Exemplo: Através da Figura 5-35 é ilustrado um gráfico demonstrativo do número total de anúncios dividido por cada participante do PTT. Já na Figura 5-36 é ilustrado um gráfico mais simples que demonstra o número total de anúncios global do PTT. Tais gráficos são úteis pois, além de mostrar de forma direta o número de anúncios, podem ser utilizados como histórico de anúncios de cada participante e auxiliar na identificação de eventos anormais.

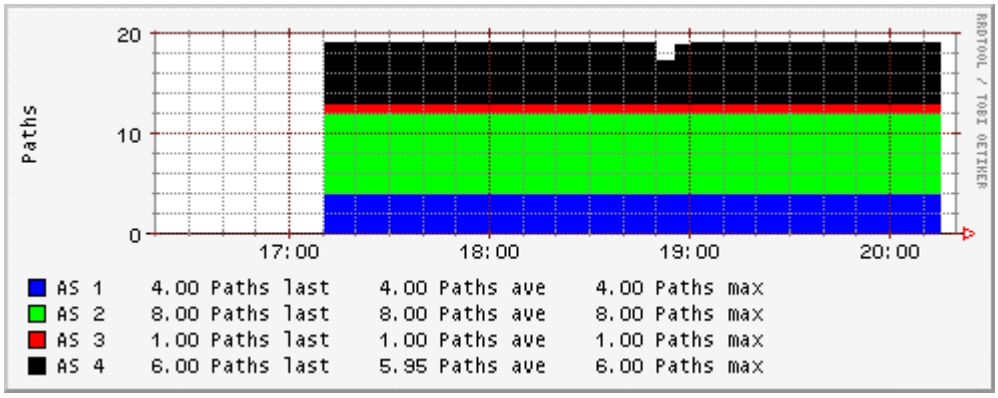

Figura 5-35 - Gráfico que ilustra o número total de anúncios de cada participante

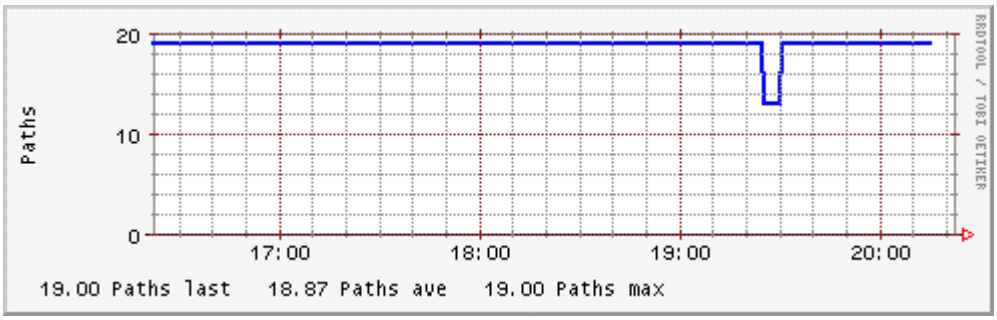

Figura 5-36 - Gráfico que ilustra o total de anúncios global no PTT

5.6.7.4 Quantificação da profundidade de anúncios

Relevância e contextualização: Mesmo sabendo do número de anúncios de determinado participante, não se sabe a quantidade de anúncios de acordo com sua profundidade, ou seja, se os anúncios são mais ou menos específicos. Em geral, são atribuídos a sistemas autônomos blocos CIDR de tamanho 20 (ou seja, /20 em notação CIDR), mas é muito comum que estes blocos sejam anunciados em sub-blocos menores. Esse detalhe pode parecer simples, mas faz com que o número de blocos anunciados em toda a Internet aumente ainda mais, exigindo mais recursos de memória e CPU de roteadores, sem contar que com a tabela de rotas maior, mais lento se torna o processamento de pacotes pelos roteadores.

Solução: O que é proposto é um conjunto de objetos com o nome de BgpeLen e BGPeSessionLen, que fornece a quantidade de blocos anunciados global e de cada participante, classificados por sua profundidade. A partir dessa informação é possível fazer um gráfico que represente tais informações, permitindo que a informação seja facilmente analisada e acompanhada pela administração do PTT.

Exemplo: Baseado em um participante do PTT, foi gerado um gráfico exemplo que contabiliza os anúncios baseados em seu tamanho. Dessa forma, através da Figura 5-37 são contabilizados os anúncios, em notação CIDR, cujo tamanho seja /16, /18,/ 20 e /24. Poderiam ser incluídas outras contabilizações, de acordo com a forma que o gerente queira visualizar melhor tais dados.

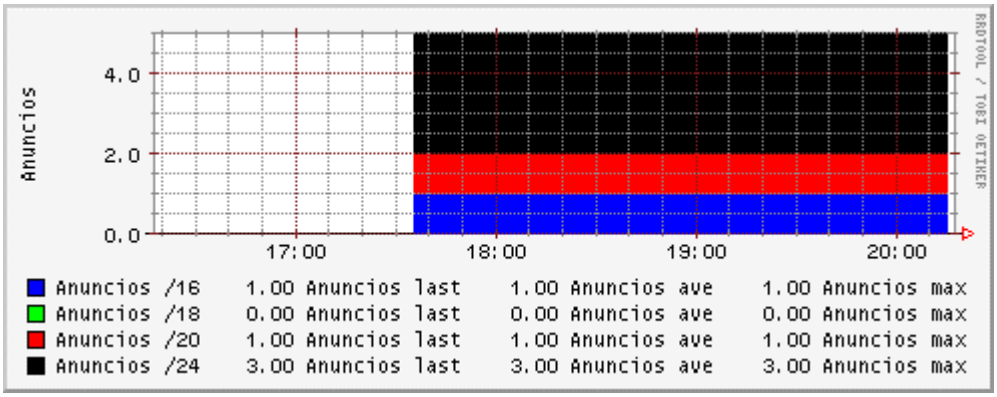

Figura 5-37 - Classificação de anúncios por tamanho de um dos participantes do PTT

5.6.7.5 Anúncio de redes inválidas

Relevância e contextualização: Uma situação que pode ocorrer é quando internamente a um sistema autônomo, geralmente via protocolos IGP, é propagado algum anúncio de redes privativas ao BGP e este propagar estas rotas em um PTT. Isso pode provocar problemas em outros sistemas autônomos, podendo até modificar o fluxo de pacotes, caso outros sistemas autônomos também utilizem algum bloco de rede privada.

Solução: O que é proposto neste caso é a inclusão das redes consideradas privadas no objeto BgpeConfPrivateNetworks, fazendo com que o sistema passe a monitorar todos os blocos anunciados e, no caso de encontrar algum anúncio que faça parte desse objeto, será gerada uma TRAP informando qual o AS que está fazendo o anúncio.

5.6.7.6 Propagação de anúncios de outros sistemas autônomos

Relevância e contextualização: Caso algum AS não faça a devida configuração de seus roteadores de borda, os anúncios de outros sistemas autônomos poderão ser propagados pelo participante a outros ASs que estão no PTT, tornando o anunciante como "caminho" até as redes anunciadas. Isso pode ficar ainda mais grave quando são propagados anúncios de sistemas autônomos grandes como Embratel, Terra, Brasil Telecom, entre outros. Tal falha faz com que a conexão do participante seja mal utilizada e se torne saturada facilmente, além de alterar o roteamento de todos os participantes do PTT.

Solução: O que é proposto é fazer uma monitoração do AS_PATH dos anúncios do PTT, observando a existência de números de ASs cadastrados no objeto BgpeConfASDeny, onde podem ser incluídos todos os números de ASs "grandes" que não deveriam estar presentes no AS_PATH de anúncios no PTT. Caso isso ocorra, será gerada uma TRAP ao gerente.

5.6.7.7 Saturação de *link* de participante do PTT

Relevância e contextualização: Em geral, as conexões de sistemas autônomos em PTT são de alta velocidade, até porque o fluxo de tráfego direto com outros participantes do PTT pode aumentar de forma muito considerável. Em algumas vezes, o AS se torna muito bem conectado em termos de critérios de avaliação dos melhores anúncios do BGP, mas se sua conexão não acompanhar o crescimento, tornará o acesso por parte dos demais participantes lentos e ineficientes.

Solução: Neste ponto, o sistema faz um acompanhamento contínuo sobre o tráfego de cada participante, comparando com o tamanho máximo de seu *link*, declarado através da MIB BgpeConf pelo objeto BgpeConfSessionBandwith para cada participante. Caso o limite de volume de tráfego ultrapasse os 80%, a equipe do PTT será notificada por intermédio de uma TRAP. Isso poderá fazer com que o participante seja avisado, sugerindo então o aumento da conexão do AS até o PTT.

6. CONCLUSÕES

Os objetivos propostos neste trabalho foram alcançados e resultaram nas seguintes contribuições:

- Implementação de um agente SNMP versão 2, preparado para ser estendido para a versão 3;
- Implementação de um gerente SNMP versão 2, também preparado para ser estendido para a versão 3;
- Implementação de um MIB compiler, que pode agregar novas MIBs em sua base de dados;
- Definição de informações importantes ao gerenciamento do protocolo BGP em PTTs;
- Especificação de MIBs experimentais que podem ser utilizadas em outros PTTs para o gerenciamento do protocolo BGP;

Adicionalmente, o trabalho possibilitou um aprendizado profundo por parte dos componentes do grupo no protocolo BGP, funcionamento de Pontos de Troca de Tráfego e no gerenciamento SNMP.

Como trabalhos futuros, pode-se citar:

- Utilização da implementação de gerente e agente em outros trabalhos na área de gerência de redes através de SNMP.
- Testes de performance sobre a capacidade de processamento que o sistema proposto atinge.
- Utilização do sistema de forma didática para auxiliar os alunos da disciplina de redes sobre os tópicos abordados pelo trabalho, a citar: roteamento BGP, gerência através de SNMP e Pontos de troca de tráfego.
- Disponibilização do código fonte do sistema a comunidade acadêmica para dar continuidade ao trabalho.
- Adaptação e aplicação do sistema em PTTs nacionais, como é o caso do RSIX.

REFERÊNCIAS BIBLIOGRÁFICAS

[ANS01] Rede ANSP - An Academic Network at São Paulo – on line – 2001 – http://www.ansp.br

[CAR93] CARVALHO, Tereza C. M. B et all. Gerenciamento de Redes de Computadores - Uma abordagem de Sistemas Abertos. São Paulo: Makron Books Ltda, 1993.

[CID02] Today CIDR Report – on line – 2002 – http://www.employees.org/~tbates/cidr-report.html

[CIS97] *LEWIS, Chris (Christopher S.)* Cisco TCP/IP routing professional reference. New York, McGraw-Hill, Inc. 1997

[CIS00] SAM HALABI, DANNY MCPERSON. Internet Routing Architectures, Second Edition. Indianapolis – USA : Cisco Press, 2000

[COM98] DOUGLAS E. COMER. Interligação em rede com TCI/IP. Rio de Janeiro : Campus, 1998. Volume I e II.

[DAN98] DANIELE, M., WIJNEN, B., FRANCISCO, D. Agent Extensibility Protocolo (Agent X). RFC 2257. IETF, Janeiro, 1998.

[GAT01] *Software* GATED – on line – 2002 - http://www.nexthop.com/products/gated.shtml

[MAN97] *BALLEW, Scott M.* Managing IP Networks with Cisco Routers. Canada : O´Reilly & Associates, Inc. 1997

[OPT01] Optiglobe – OPTix-LA – on line – 2001 – http://www.optiglobe.com

[RAD02] Router Arbiter Project – on line – 2002 - http://www.ra.net/

[REC00] Comitê Gestor Internet/BR – Grupo de Trabalho em Engenharia de Redes - Operação e Administração de PTTs no Brasil – on line – 2000 - http://www.cg.org.br/grupo/operacao_ptt_v1.1.htm

[ROS90] ROSE, Marshall, McCLOGHRIE, Keith. Structure and Identification of Management of TCP/IP based internets. Request for Comments 1155, DDN Network Information Center, SRI International, Maio, 1990.

[RSI01] RSIX - Ponto de Troca de Tráfego Internet – on line - 2000 – http://penta.ufrgs.br/rsix

[STA99] STALLINGS, Willian. SNMP, SNMPv2, SNMPv3 and RMON 1 and 2. Addison-Wesley, 3ª Edition, 1999.

[UCD01] *Software* UCD-SNMP – on line – 2002 - http://www.net-snmp.org

[ZEB01] *Software* Zebra – on line – 2002 - http://www.zebra.org

ANEXO A

É apresentado no anexo A, a definição da MIB Bgpe em ASN.1.

```
bgpe DEFINITIONS ::= BEGIN

IMPORTS;

bgpe ::= { experimental 1  }

bgpeInOctets OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Trafego de entrada"
 ::= { bgpe 1 }

bgpeOutOctets OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Trafego de saida"
 ::= { bgpe 2 }

bgpeTotalPath OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Total de prefixos anunciados por todos os
  participantes do PTT"
 ::= { bgpe 3 }

bgpeLen OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade de cada tamanho dos prefixos anunciados"
 ::= { bgpe 4 }

bgpeOrigin OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade de prefixo anunciados de acordo com origin"
 ::= { bgpe 5 }

bgpeStates OBJECT-TYPE
 SYNTAX SEQUENCE OF bgpeStatesEntry
 ACCESS not-accessible
```

```
 STATUS mandatory
 DESCRIPTION
 "Status, indica quantos participantes esta em cada estado"
 ::= { bgpe  6}

bgpeStatesIdle OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantos sessoes estao no estado Idle"
 ::= { bgpeStates 1 }

bgpeStatesConnect OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantos sessoes estao no estado Connect"
 ::= { bgpeStates 2 }

bgpeStatesActive OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantos sessoes estao no estado Active"
 ::= { bgpeStates 3 }

bgpeStatesOpenSet OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantos sessoes estao no estado OpenSet"
 ::= { bgpeStates 4 }

bgpeStatesOpenCofirm OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantos sessoes estao no estado OpenConfirm"
 ::= { bgpeStates 5 }

bgpeStatesEstablished OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantos sessoes estao no estado Established"
 ::= { bgpeStates 6 }


bgpeTrafficTable OBJECT-TYPE
 SYNTAX SEQUENCE OF bgpeTrafficEntry
 ACCESS not-accessible
```

```
 STATUS mandatory
 DESCRIPTION
 "Tabela com o trafego entre pares de hosts"
 ::= { bgpe  7 }

bgpeTrafficEntry OBJECT-TYPE
 SYNTAX bgpeTrafficEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabele de trafego"
 INDEX { bgpeSessionAsNumber ? }
 ::= { bgpeTrafficTable  1 }

bgpeTrafficEntry ::=
    SEQUENCE {
        bgpeTrafficSourceAddress
            IPAdress,
        bgpeTrafficDestAddress
            IPAddress,
        bgpeTrafficInOctets
            INTEGER,
        bgpeTrafficOutOctets
            INTEGER,
    }

bgpeTrafficSourceAddress OBJECT-TYPE
 SYNTAX IpAddress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereco IP do host origem"
 ::= { bgpeTrafficEntry 1 }

bgpeTrafficDestAddress OBJECT-TYPE
 SYNTAX IpAddress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereco IP do host origem"
 ::= { bgpeTrafficEntry 2 }

bgpeTrafficInOCtets OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade de trafego recebido"
 ::= { bgpeTrafficEntry 3 }

bgpeTrafficoOutOCtets OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade de trafego enviado"
 ::= { bgpeTrafficEntry 4 }
```

```
bgpeSessionTable OBJECT-TYPE
 SYNTAX SEQUENCE OF bgpeSessionEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabela com informacoes de cada participante do PTT"
 ::= { bgpe  8 }

bgpeSessionEntry OBJECT-TYPE
 SYNTAX bgpeSessionEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabela"
 INDEX { ? }
 ::= { bgpeSessionTable  1 }

bgpeSessionEntry ::=
    SEQUENCE {
        bgpeSessionASNumber
            DisplayString,
        bgpeSessionRemoteAddress
            IpAdress,
        bgpeSessionInOctets
            INTEGER,
        bgpeSessionOutOctets
            INTEGER,
        bgpeSessionStatus
            INTEGER,
        bgpeSessionUpTime
            INTEGER,
        bgpeSessionTotalPath
            INTEGER,
        bgpeSessionOriginIgp
            INTEGER,
        bgpeSessionOriginEgp
            INTEGER,
        bgpeSessionOriginIncomplete
            INTEGER,
        bgpeSessionUpdates
            INTEGER,
        bgpeSessionLastError
            INTEGER,
        bgeSessionPathTable
            bgpeSessionPathEntry
    }

bgpeSessionASNumber OBJECT-TYPE
 SYNTAX DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero do sistema automato"
 ::= { bgpeSessionEntry 1 }
```

```
bgpeSessionRemoteAddress OBJECT-TYPE
 SYNTAX IpAdress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereco IP do peer da sessao BGP remoto"
 ::= { bgpeSessionEntry 2 }

bgpeSessionInOctets OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade, em bytes, de trafego que foram recebidos"
 ::= { bgpeSessionEntry 3 }

bgpeSessionOutOctets OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade, em bytes, de trafego que foram enviados"
 ::= { bgpeSessionEntry 4 }

bgpeSessionStatus OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Status atual da conexao BGP"
 ::= { bgpeSessionEntry 5 }

bgpeSessionUpTime OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Tempo decorrente da sessao BGP com os Route Servers"
 ::= { bgpeSessionEntry 6 }

bgpeSessionTotalPath OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero total de prefixos anunciados"
 ::= { bgpeSessionEntry 7 }

bgpeSessionOriginIgp OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Classificacao do prefixo"
 ::= { bgpeSessionEntry 8 }

bgpeSessionOriginEgp OBJECT-TYPE
```

```
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Classificacao do prefixo"
 ::= { bgpeSessionEntry 9 }

bgpeSessionOriginIncomplete OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Classificacao do prefixo"
 ::= { bgpeSessionEntry 10 }


bgpeSessionUpdates OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero de mensagens tipo UPDATE periodico de cada participante"
 ::= { bgpeSessionEntry 11 }

bgpeSessionLastError OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Ultimo erro ocorrido na sessao BGP"
 ::= { bgpeSessionEntry 12 }

bgpeSessionPathTable OBJECT-TYPE
 SYNTAX SEQUENCE OF bgpePathEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabela com informacoes de cada rota"
 ::= { bgpeSessionEntry  13 }

bgpeSessionPathEntry OBJECT-TYPE
 SYNTAX bgpeStatesEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabela com informacoes de cada rota"
 INDEX { bgpeSessionPathIndex }
 ::= { bgpePathTable  1 }

bgpeSessionPathEntry ::=
    SEQUENCE {
        bgpeSessionPathIndex
        bgpeSessionPathLen
            INTEGER,
    }

bgpeSessionPathLen OBJECT-TYPE
```

```
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Identificação do Tamanho"
 ::= { bgpePathEntry 1 }

bgpeSessionPathLen OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Quantidade de prefixo anunciados para cada tamanho"
 ::= { bgpePathEntry 2 }

END
```

ANEXO B

É apresentado no anexo B, a definição da MIB BgpeTrap de configuração em ASN.1.

```
bgpeTrap DEFINITIONS ::= BEGIN

IMPORTS;

bgpeTrap   OBJECT IDENTIFIER ::= { experimental 4 }

bgpeTrapTraffic NOTIFICATION-TYPE
 OBJECTS {
       bgpeConfSessionBandwithMax
       bgpeConfSessionBandwithMin
       bgpeInOctets
       bgpeOutOctets
      }
 STATUS  current
 DESCRIPTION
 "Gera o trap quando o bgpeInOctets ou bgpOutOctets sair do
  limite imposto por bgpeConfSessionBandwithMax e
  bgpeConfSessionBandwithMin"
 ::= { bgpeTrap 1 }

bgpeTrapPath NOTIFICATION-TYPE
 OBJECTS {
        bgpeTotalPath
        bgpeConfMaxPath
        bgpeConfMinPath
      }
 STATUS  current
 DESCRIPTION
 "Gera o trap quando bgpTotalPath sair do limite imposto por
  bpeConfMaxPath e bgpeConfMinPath"
 ::= { bgpeTrap 2 }

bgpeTrapSession NOTIFICATION-TYPE
 OBJECTS {
        bgpeSessionStatus
      }
 STATUS  current
 DESCRIPTION
 "Quando o link estiver down, conseguido através do bgpeSessionStatus"
 ::= { bgpeTrap 3 }

bgpeTrapNextHop NOTIFICATION-TYPE
 OBJECTS {  bgpeConfASDeny }
 STATUS  current
 DESCRIPTION
 " Quando alguém tiver como nexthop algum de bgpeConfASDeny "
 ::= { bgpeTrap 4 }

bgpeTrapAsPath NOTIFICATION-TYPE
 OBJECTS { bgpeConfASDeny      }
```

```
 STATUS  current
 DESCRIPTION
 " Quando alguém tiver como path algum de bgpeConfASDeny "
 ::= { bgpeTrap 5 }

bgpeTrapRouteServer NOTIFICATION-TYPE
 OBJECTS {
      bgpeConfRouteServerCPUMax
      bgpeConfRouteServerMemoryMax
      bgpeConfRouteServerProcess
      bgpeLogRouteServerCPUMax
      bgpeLogRouteServerMemoryMax
      bgpeLogRouteServerProcess

}
 STATUS  current
 DESCRIPTION
 " Quando alguém tiver como path algum de bgpeConfASDeny "
 ::= { bgpeTrap 56}

END
```

ANEXO C

É apresentado no anexo C, a definição da MIB BgpeConf de configuração em ASN.1.

```
bgpeConf DEFINITIONS ::= BEGIN

IMPORTS;

bgpeConf ::= { experimental 2  }

bgpeConfIPSwitch OBJECT-TYPE
 SYNTAX IpAddress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereço IP do Switch do PTT"
 ::= { bgpeConf 1 }

bgpeConfIPRouteServers OBJECT-TYPE
 SYNTAX IpAddress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereco IP dos Routes Servers"
 ::= { bgpeConf 2 }

bgpeConfVLanPos OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Referencia da VLan no Switch que engloba todas as
  portas dos participantes do PTT"
 ::= { bgpeConf 3 }

bgpeConfMaxPath OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero maximo de prefixos do PTT"
 ::= { bgpeConf 4 }

bgpeConfMinPath OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero minimo de prefixos do PTT"
 ::= { bgpeConf 5 }


bgpeRouteServerCPUMax OBJECT-TYPE
 SYNTAX INTEGER
```

```
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Utilizaçao maxima de CPU esperada dos Route Servers"
 ::= { bgpeConf 6 }

bgpeRouteServerMemoryMax OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Utilizaçao maxima de memoria esperada dos Route Servers"
 ::= { bgpeConf 7 }


bgpeRouteServerProcessMax OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero maximo de processos esperado dos Route Servers"
 ::= { bgpeConf 8 }

bgpeConfMailDest OBJECT-TYPE
 SYNTAX DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 ""
 ::= { bgpeConf 9 }

bgpeConfSMTP OBJECT-TYPE
 SYNTAX IpAddress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereco IP do SMTP para postagem dos e-mails de alarmes"
 ::= { bgpeConf 10 }

bgpeConfTrapDest OBJECT-TYPE
 SYNTAX IpAddress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Endereço IP da estaçao de gerencia que recebe as Traps"
 ::= { bgpeConf 11 }

bgpeConfASDeny OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "ASs que nao deve constar em nenhum anuncio de participantes do PTT"
 ::= { bgpeConf 12 }

bgpeConfDamp OBJECT-TYPE
 SYNTAX INTEGER
```

```
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Ativado ou nao o alarme de damping"
 ::= { bgpeConf 13 }

bgpeConfPathLog OBJECT-TYPE
 SYNTAX  DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Localizacao do Log no sistema"
 ::= { bgpeConf 14 }

bgpeConfReportTime OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Intervalo de tempo que deve ser enviado relatorio sobre as
  atividade mais importantes do PTT"
 ::= { bgpeConf 15 }


bgpeConfSessionTable OBJECT-TYPE
 SYNTAX SEQUENCE OF bgpeConfSessionEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabela com dados de configuracao de
  cada participante"
 ::= { bgpeConf  16 }

bgpeConfSessionEntry OBJECT-TYPE
 SYNTAX bgpeConfSessionEntry
 ACCESS not-accessible
 STATUS mandatory
 DESCRIPTION
 "Tabela"
 INDEX { ? }
 ::= { bgpeConfSessionTable  1 }

bgpeConfSessionEntry ::=
    SEQUENCE {
        bgpeConfSessionIP
            IpAdress,
        bgpeConfSessionSwitchPos
            INTEGER,
        bgpeConfSessionDescr
            DisplayString,
        bgpeConfSessionASNumber
            DisplayString,
        bgpeConfSessionMaxPath
            INTEGER,
        bgpeConfSessionMinPath
            INTEGER,
        bgpeConfSessionBandwithMax
```

```
          INTEGER,

        bgpeConfSessionBandwithMin
          INTEGER,

    }

bgpeConfSessionIP OBJECT-TYPE
 SYNTAX IpAdress
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "IP"
 ::= { bgpeConfSessionEntry 1 }

bgpeConfSessionSwitchPos OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Posicao no Switch"
 ::= { bgpeConfSessionEntry 2 }

bgpeConfSessionDescr OBJECT-TYPE
 SYNTAX DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Descricao"
 ::= { bgpeConfSessionEntry 3 }

bgpeConfSessionASNumber OBJECT-TYPE
 SYNTAX DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero do AS"
 ::= { bgpeConfSessionEntry 4 }

bgpeConfSessionMaxPath OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero maximo de rotas"
 ::= { bgpeConfSessionEntry 5 }

bgpeConfSessionMinPath OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Numero minimo de rotas"
 ::= { bgpeConfSessionEntry 6 }

bgpeConfSessionBandwithMax OBJECT-TYPE
 SYNTAX INTEGER
```

```
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Banda da conexao"
 ::= { bgpeConfSessionEntry 7 }


bgpeConfSessionBandwithMin OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 "Banda da conexao"
 ::= { bgpeConfSessionEntry 8 }


END
```

ANEXO D

É apresentado no anexo D, a definição da MIB BgpeRSUtil em ASN.1.

```
bgpeRSUtil DEFINITIONS ::= BEGIN

IMPORTS;

bgpeRSUtil::= { experimental 3  }

bgpeRSUtilDampennedNetworks OBJECT-TYPE
 SYNTAX DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 " Representa as redes que estão incluídas em dampenning "
 ::= { bgpeRSUtil 1 }

bgpeRSUtilMaxAnnounc OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 “Registro do AS cujo número de anúncios máximos foi excedido."
 ::= { bgpeRSUtil 2 }

bgpeRSUtilRouteServerCPUUser OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 " Registra informações da utilização da CPU do sistema operacional dos Route Servers
 pelo usuário "
 ::= { bgpeRSUtil 3 }

bgpeRSUtilRouteServerCPUSystemOBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 " Registra informações da utilização da CPU do sistema operacional dos Route Servers
 pelo sistema"
 ::= { bgpeRSUtil 4 }

bgpeRSUtilRouteServerCPUIdle OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 " Registra informações do utilização da CPU do sistema operacional dos Route Servers
 em Idle"
 ::= { bgpeRSUtil 5 }

bgpeRSUtilRouteServerMemoryUsed OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
```

```
 STATUS mandatory
 DESCRIPTION
 " Registra informações do uso de memoria do sistema operacional dos Route Servers "
::= { bgpeRSUtil 6 }

bgpeRSUtilRouteServerMemoryFree OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 " Registra informações do uso de memoria do sistema operacional dos Route Servers "
 ::= { bgpeRSUtil 7 }

bgpeRSUtilRouteServerProcess OBJECT-TYPE
 SYNTAX INTEGER
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 " Registra informações do número de processos do sistema operacional dos Route Servers"
 ::= { bgpeRSUtil 8 }

bgpeRSUtilRouteServerLogGeneral OBJECT-TYPE
 SYNTAX DisplayString
 ACCESS read
 STATUS mandatory
 DESCRIPTION
 ""
 ::= { bgpeRSUtil 9 }

END
```

ANEXO E

Segue abaixo os detalhes de configuração e utilização do primeiro protótipo de PTT criado no laboratório de redes:

O endereçamento de cada participante foi definido como mostra a Tabela A-1:

Tabela A-1 - Endereçamento de cada participante

Descrição / utilização do PC	Endereço na rede local utilizado
Looking Glass	10.30.143.254
*Route Server*1 (RSD1)	10.30.143.7
*Route Server*2 (RSD2)	10.30.143.6
Participante do PTT – AS 4	10.30.143.5
Participante do PTT – AS 3	10.30.143.4
Participante do PTT – AS 2	10.30.143.3
Participante do PTT – AS 1	10.30.143.2

Tendo 4 participantes na topologia inicial de PTT, foi atribuída uma rede para cada um divulgar no PTT, simulando uma situação real onde os participantes divulgam as redes de seus sistemas autônomos.

A rede atribuída a cada participante é definida pela Tabela A-2.

Tabela A-2 - Definição de rede/bloco CIDR para cada participante

Participante / Sistema autônomo	Rede / bloco CIDR a ser divulgado
1	192.168.1.0/24
2	192.168.2.0/24
3	192.168.3.0/24
4	192.168.160/20

Em termos de configuração, cada participante deve estabelecer sessões com os *Route Servers* e ter o cuidado de anunciar apenas prefixos pertencentes a seus ASs e seus possíveis clientes. Através da Figura A-1 é mostrado um exemplo de configuração de cada participante.

```
router bgp 1
 bgp router-id 10.30.143.2
 network 192.168.1.0/24
 neighbor 10.30.143.6 remote-as 65000
```

```
 neighbor 10.30.143.6 filter-list PTT out
 neighbor 10.30.143.7 remote-as 65000
 neighbor 10.30.143.7 filter-list PTT out
!
ip as-path access-list PTT permit ^$
```

Figura A-1 - Configuração do participante 1

Entre cada participante, o que muda é o identificador do AS (router bgp 1), o router-id (bgp router-id 10.30.143.2) e o bloco divulgado (network 192.168.1.0/24).

Com os *Route Servers*, a configuração deve possibilitar as seguintes funcionalidades:

- Estabelecimento de sessões BGP com todos os participantes;
- Estabelecimento de sessão BGP com o *Looking Glass*;
- Implementação de filtros por AS-PATH para garantir anúncios válidos no PTT;
- Não alterar os atributos AS_PATH e NEXT_HOP de cada prefixo anunciado, conforme explicado anteriormente;

Para fornecer tais funcionalidades, a configuração determinada é ilustrada na Figura A-2 abaixo:

```
router bgp 65000
 bgp router-id 10.30.143.6
 neighbor 10.30.143.2 remote-as 1
 neighbor 10.30.143.2 filter-list ATM out
 neighbor 10.30.143.2 transparent-as
 neighbor 10.30.143.2 transparent-nexthop
 neighbor 10.30.143.3 remote-as 2
 neighbor 10.30.143.3 filter-list ATM out
 neighbor 10.30.143.3 transparent-as
 neighbor 10.30.143.3 transparent-nexthop
 neighbor 10.30.143.4 remote-as 3
 neighbor 10.30.143.4 filter-list ATM out
 neighbor 10.30.143.4 transparent-as
 neighbor 10.30.143.4 transparent-nexthop
 neighbor 10.30.143.5 remote-as 4
 neighbor 10.30.143.5 filter-list ATM out
 neighbor 10.30.143.5 transparent-nexthop
 neighbor 10.30.143.254 remote-as 65001
 neighbor 10.30.143.254 filter-list ATM out
 neighbor 10.30.143.254 transparent-as
 neighbor 10.30.143.254 transparent-nexthop
!
 ip as-path access-list ATM permit ^1_
 ip as-path access-list ATM permit ^2_
```

```
ip as-path access-list ATM permit ^3_
ip as-path access-list ATM permit ^4_
```

Figura A-2 - Configuração do *Route Server* 1

Ao *Looking Glass*, para obter as rotas anunciadas no PTT, basta ativar sessões BGP com os *Route Servers*, como é ilustrado na Figura A-3:

```
router bgp 65001
 bgp router-id 10.30.143.254
 neighbor 10.30.143.6 remote-as 65000
 neighbor 10.30.143.6 transparent-as
 neighbor 10.30.143.6 transparent-nexthop
 neighbor 10.30.143.7 remote-as 65000
 neighbor 10.30.143.7 transparent-as
 neighbor 10.30.143.7 transparent-nexthop
```

Figura A-3 - Configuração do *Looking Glass*

Com as configurações necessárias aplicadas em cada componente do PTT, é possível verificar o *status* das conexões BGP. Na visão de cada participante, a Figura A-4 apresenta o *status* da sessão BGP do participante 1. Um resultado muito semelhante é obtido em cada um participante.

```
AS1# sh ip bgp sum
BGP router identifier 10.30.143.2, local AS number 1
4 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.30.143.6     4 65000     194     212        0    0    0 00:05:19        3
10.30.143.7     4 65000     216     222        0    0    0 00:05:19        3

Total number of neighbors 2
```

Figura A-4 - *Status* das conexões BGP com os *Route Servers* do participante 1

Pelo lado dos *Route Servers*, a tabela de sessões BGP mostra as sessões com todos os participantes, inclusive com o *Looking Glass*, conforme a Figura A-5 pode demonstrar:

```
AS65000-RSD1# show ip bgp sum
BGP router identifier 10.30.143.6, local AS number 65000
4 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
```

```
10.30.143.2     4     1   185     238     0    0    0 00:13:32        1
10.30.143.3     4     2   186     211     0    0    0 00:13:29        1
10.30.143.4     4     3   168     210     0    0    0 00:13:23        1
10.30.143.5     4     4   205     205     0    0    0 00:13:26        1
10.30.143.254   4 65001   155     218     0    0    0 00:27:38        0

Total number of neighbors 5
AS65000-RSD1#
```

Figura A-5 - Sessões BGP de *Route Server* 1

Uma vez que as sessões BGP estejam ativas, o importante nessa altura é verificar a tabela BGP a fim de obter os prefixos que estão sendo divulgados e suas informações. A Figura A-6 ilustra a visão das rotas do *Route Server* 1. Ao total são listados 4 prefixos, cada um proveniente de um participante, com o atributo NEXT_HOP atribuído a própria interface na rede local de cada participante e AS_PATH apenas com o AS de quem anunciou.

```
AS65000-RSD1# sh ip bgp
BGP table version is 0, local router ID is 10.30.143.6
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0      10.30.143.2                            0 1 i
*> 192.168.2.0      10.30.143.3                            0 2 i
*> 192.168.3.0      10.30.143.4                            0 3 i
*> 192.168.160.0    10.30.143.5                            0 4 i

Total number of prefixes 4
AS65000-RSD1#
```

Figura A-6 - Tabela BGP do *Route Server* 1

Por parte de cada participante, obtém-se uma tabela de prefixos semelhante, como podemos verificar na Figura A-7, que refere-se ao AS 1. Algo importante ser observado é que todos os prefixos, com exceção do prefixo do próprio AS estão com dois anúncios cada. Isso ocorre pois cada prefixo aprendido é divulgado pelos dois *Route Servers*. Caso um dos *Route Servers* falhe, o anúncio do outro manterá o fluxo de dados em estado normal.

```
AS1# sh ip bgp
BGP table version is 0, local router ID is 10.30.143.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0      0.0.0.0                            32768 i
*  192.168.2.0      10.30.143.3                            0 2 i
```

```
*>                10.30.143.3              0 2 i
*  192.168.3.0    10.30.143.4              0 3 i
*>                10.30.143.4              0 3 i
*  192.168.160.0  10.30.143.5              0 4 i
*>                10.30.143.5              0 4 i

Total number of prefixes 4
AS1#
```

Figura A-7 - Tabela BGP do participante 1 (AS 1)

Por fim, também pode-se ter uma visão dos prefixos do PTT por intermédio do *Looking Glass*. Esta visão é apresentada na Figura A-8. Podem ser vistas por esta visão todos os prefixos do PTT. Pode-se utilizar tal elemento para testar a alcançabilidade de determinadas rotas de "dentro" do PTT por seus participantes e por outros interessados.

```
LG# sh ip bgp
BGP table version is 0, local router ID is 10.30.143.254
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0      10.30.143.2                           0 1 i
*> 192.168.2.0      10.30.143.3                           0 2 i
*> 192.168.3.0      10.30.143.4                           0 3 i
*> 192.168.160.0    10.30.143.5                           0 4 i

Total number of prefixes 4
```

Figura A-8 - Tabela BGP do *Looking Glass*

Um dos aspectos que devem ser muito cuidados em PTTs que pode ser ilustrado no protótipo é a aplicação correta de filtros por parte dos *Route Servers*. A Figura A-9 apresenta a tabela BGP do *Route Server* 1 sem a utilização de filtros corretos.

```
65000-RSD2# show ip bgp
BGP table version is 0, local router ID is 10.30.143.7
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0      10.30.143.2                           0 1 i
*                   10.30.143.3                           0 2 1 i
*                   10.30.143.4                           0 3 1 i
*  192.168.2.0      10.30.143.2                           0 1 2 i
```

```
*>                  10.30.143.3              0 2 i
*                   10.30.143.4              0 3 2 i
* 192.168.3.0       10.30.143.2              0 1 3 i
*                   10.30.143.3              0 2 3 i
*>                  10.30.143.4              0 3 i
* 192.168.160.0     10.30.143.2              0 1 4 i
*                   10.30.143.3              0 2 4 i
*                   10.30.143.4              0 3 4 i
*>                  10.30.143.5              0 4 i

Total number of prefixes 4
AS65000-RSD2#
```

Figura A-9 - Tabela BGP do *Route Server* 2, sem filtros por AS_PATH

Como se pode ver, os participantes acabam anunciando as mesmas rotas que aprenderam no PTT de outros participantes, incluindo seu NEXT_HOP e AS_PATH nos prefixos. Supondo que um participante apresente problemas, nada será útil que outros participantes estejam anunciando os prefixos deste participante, pois todas serão removidas quando o participante apresentar problemas. Essa sobreposição também acaba aumentando a tabela de rotas divulgada no PTT desnecessariamente, exigindo mais CPU dos roteadores de cada AS.

ANEXO F

A segunda arquitetura de PTT modelada no laboratório de redes cujo propósito é validar de fato o sistema possui algumas diferenças em seus componentes, conforme o que foi citado na seção.

O endereçamento e associação a cada componente são mostrados na Tabela A-3:

Tabela A-3 - Endereçamento dos PCs

Descrição / utilização do PC	Endereço na rede local utilizado
Estação de gerência	10.30.143.254
Route Server (RSD)	10.30.143.7
Entidade Dual	10.30.143.6
Participante do PTT – AS 4	10.30.143.5
Participante do PTT – AS 3	10.30.143.4
Participante do PTT – AS 2	10.30.143.3
Participante do PTT – AS 1	10.30.143.2

Em relação aos anúncios de blocos feitos por cada participante, o número destes foi aumentado para permitir que sejam melhor explorados os recursos que o sistema proposto oferece. A divisão dos anúncios de cada participante foi definida como mostra a Tabela A-4:

Tabela A-4 - Definição de rede/bloco CIDR para cada participante

Participante / Sistema autônomo	Rede / bloco CIDR a ser divulgado
1	151.18.10.0/24 151.18.11.0/25 151.18.11.0/26 151.18.11.128/26
2	152.10.0.0/15 152.15.0.0/24 152.15.1.0/25 152.16.10.0/26 152.16.11.0/27 152.16.12.0/28 152.16.12.128/29 152.16.13.0/30
3	153.16.0.0/16
4	154.1.0.0/20 154.1.2.0/24 154.1.3.0/27

	154.1.4.0/27 154.1.5.0/27 154.2.6.0/28

Uma vez que os blocos a serem anunciados estejam definidos, a configuração dos participantes pode ser completada. Como diferença da arquitetura anterior, existe apenas um *Route Server* e por isso, cada participante estabelece apenas uma sessão BGP com o RSD. Na Figura A-10 é mostrada a configuração do participante 1:

```
hostname AS1
password zebra
!
router bgp 1
bgp router-id 10.30.143.2
network 151.18.10.0/24
network 151.18.11.0/25
network 151.18.11.0/26
network 151.18.11.128/26
!
neighbor 10.30.143.6 remote-as 65000
neighbor 10.30.143.6 filter-list PTT out

ip as-path access-list PTT permit ^$
```

Figura A-10 - Configuração do participante 1

Em relação a configuração dos demais participantes, o que muda é apenas a identificação do AS (router bgp x), o identificador do roteador (bgp router-id x.x.x.x) e os anúncios de blocos, geralmente feitos através do comando network. O restante das configurações não é alterado.

Já a configuração do RSD, que nesta arquitetura será único, são estabelecidas sessões com os 4 participantes, conforme segue a configuração:

```
! -*- bgp -*-
!
! BGPd sample configuratin file
!
! $Id: bgpd.conf.sample,v 1.19 1999/02/19 17:17:27 developer Exp $
!
hostname AS65000-RSD
password zebra
!enable password please-set-at-here
!
!bgp mulitple-instance
!
router bgp 65000
bgp router-id 10.30.143.6

neighbor 10.30.143.2 remote-as 1
neighbor 10.30.143.2 filter-list ATM out
```

```
neighbor 10.30.143.2 transparent-as
neighbor 10.30.143.2 transparent-nexthop

neighbor 10.30.143.3 remote-as 2
neighbor 10.30.143.3 filter-list ATM out
neighbor 10.30.143.3 transparent-as
neighbor 10.30.143.3 transparent-nexthop

neighbor 10.30.143.4 remote-as 3
neighbor 10.30.143.4 filter-list ATM out
neighbor 10.30.143.4 transparent-as
neighbor 10.30.143.4 transparent-nexthop

neighbor 10.30.143.5 remote-as 4
neighbor 10.30.143.5 filter-list ATM out
neighbor 10.30.143.5 transparent-as
neighbor 10.30.143.5 transparent-nexthop


ip as-path access-list ATM permit ^1_
ip as-path access-list ATM permit ^2_
ip as-path access-list ATM permit ^3_
ip as-path access-list ATM permit ^4_

log file bgpd-rsd.log
!
log stdout
bash-2.05#
```

Figura A-11 – Configuração do RSD

Com o RSD e participantes configurados, o próximo passo é verificar o *status* das sessões BGP entre eles. A visão a partir do RSD destas sessões pode ser verificada através da Figura A-12:

```
AS65000-RSD> sh ip bgp sum
BGP router identifier 10.30.143.6, local AS number 65000
4 BGP AS-PATH entries
0 BGP community entries

Neighbor      V  AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down
State/PfxRcd
10.30.143.2   4   1    1384 1409      0    0    0 00:06:08        4
10.30.143.3   4   2    1381 1408      0    0    0 00:03:53        8
10.30.143.4   4   3     247  276      0    0    0 00:02:40        1
10.30.143.5   4   4    1382 1405      0    0    0 00:01:09        6

Total number of neighbors 4
```

Figura A-12 – Visão das sessões

Neste momento, as sessões com os 4 participantes está estabelecida e um total de 19 anúncios estão sendo recebidos (4+8+1+6). Da mesma forma, a verificação a partir de um dos participantes, no exemplo o participante 1, através da Figura A-13:

```
AS1> sh ip bgp sum
BGP router identifier 10.30.143.2, local AS number 1
4 BGP AS-PATH entries
0 BGP community entries

Neighbor      V  AS MsgRcvd MsgSent   TblVer InQ OutQ Up/Down
State/PfxRcd
10.30.143.6   4 6500   5       4        0    0    0 00:00:09       15

Total number of neighbors 1
AS1>
```

Figura A-13 – Participante 1

Como pode-se ver na Figura anterior, o participante 1 está recebendo um total de 15 anúncios. Somando este número com o total de anúncios que o AS 1 possui em seu AS (4), chega-se ao total de 19 anúncios que confere com o número recebido pelo RSD. A demonstração dos anúncios recebidos por parte do AS 1 é mostrada na Figura A-14:

```
AS1> sh ip bgp
BGP table version is 0, local router ID is 10.30.143.2
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 151.18.10.0/24   0.0.0.0                  0         32768 i
*> 151.18.11.0/25   0.0.0.0                  0         32768 i
*> 151.18.11.0/26   0.0.0.0                  0         32768 i
*> 151.18.11.128/26 0.0.0.0                  0         32768 i
*> 152.10.0.0/15    10.30.143.3                           0 2 i
*> 152.15.0.0/24    10.30.143.3                           0 2 i
*> 152.15.1.0/25    10.30.143.3                           0 2 i
*> 152.16.10.0/26   10.30.143.3                           0 2 i
*> 152.16.11.0/27   10.30.143.3                           0 2 i
*> 152.16.12.0/28   10.30.143.3                           0 2 i
*> 152.16.12.128/29 10.30.143.3                           0 2 i
*> 152.16.13.0/30   10.30.143.3                           0 2 i
*> 153.16.0.0       10.30.143.4                           0 3 i
*> 154.1.0.0/20     10.30.143.5                           0 4 i
*> 154.1.2.0/24     10.30.143.5                           0 4 i
*> 154.1.3.0/27     10.30.143.5                           0 4 i
*> 154.1.4.0/27     10.30.143.5                           0 4 i
*> 154.1.5.0/27     10.30.143.5                           0 4 i
*> 154.2.6.0/28     10.30.143.5                           0 4 i

Total number of prefixes 19
AS1>
```

Figura A-14 – Anúncios recebidos pelo AS1

Da mesma forma que o participante 1, a verificação das rotas dos participantes pode ser conferida. Na Figura A-15 é representada a listagem da rotas da tabela BGP do participante 1:

```
AS65000-RSD> sh ip bgp
BGP table version is 0, local router ID is 10.30.143.6
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 151.18.10.0/24   10.30.143.2              0             0 1 i
*> 151.18.11.0/25   10.30.143.2              0             0 1 i
*> 151.18.11.0/26   10.30.143.2              0             0 1 i
*> 151.18.11.128/26 10.30.143.2              0             0 1 i
*> 152.10.0.0/15    10.30.143.3              0             0 2 i
*> 152.15.0.0/24    10.30.143.3              0             0 2 i
*> 152.15.1.0/25    10.30.143.3              0             0 2 i
*> 152.16.10.0/26   10.30.143.3              0             0 2 i
*> 152.16.11.0/27   10.30.143.3              0             0 2 i
*> 152.16.12.0/28   10.30.143.3              0             0 2 i
*> 152.16.12.128/29 10.30.143.3              0             0 2 i
*> 152.16.13.0/30   10.30.143.3              0             0 2 i
*> 153.16.0.0       10.30.143.4              0             0 3 i
*> 154.1.0.0/20     10.30.143.5              0             0 4 i
*> 154.1.2.0/24     10.30.143.5              0             0 4 i
*> 154.1.3.0/27     10.30.143.5              0             0 4 i
*> 154.1.4.0/27     10.30.143.5              0             0 4 i
*> 154.1.5.0/27     10.30.143.5              0             0 4 i
*> 154.2.6.0/28     10.30.143.5              0             0 4 i

Total number of prefixes 19
AS65000-RSD>
```

Figura A-15 – Rotas da tabela do participante 1

Comparando a tabela BGP dos participantes da arquitetura anterior, pode-se verificar um detalhe importante onde cada rota possui duas entradas, ou seja, uma entrada para cada *Route Server*, com o intuito de redundância, pois no caso de um *Route Server* deixar de anunciar alguma rota, o outro mantém os anúncios. Já nesta arquitetura pode-se verificar que a redundância não existe mais visto que o número de *Route Server* foi reduzido para apenas 1.

ANEXO G

No Anexo G é apresentado uma listagem com a identificação dos objetos das seguintes Mibs: *BGPe*, *BGPeConf*, *BGPeRSUtil*, *BGPeTraps*.

BGPe

Tabela A-5 - Objetos BGPe

Nome do objeto	OID
//globais	//globais
BgpeInOctets	1.3.6.1.3.1.1.0
BgpeOutOctets	1.3.6.1.3.1.2.0
BgpeTotalPath	1.3.6.1.3.1.3.0
BgpeLen	1.3.6.1.3.1.4.0
BgpeOrigin	1.3.6.1.3.1.5.0
BgpeStatesIdle	1.3.6.1.3.1.6.1.1.0
BgpeStatesConnect	1.3.6.1.3.1.6.1.2.0
BgpeStatesActive	1.3.6.1.3.1.6.1.3.0
BgpeStatesOpenSet	1.3.6.1.3.1.6.1.4.0
BgpeStatesOpenCofirm	1.3.6.1.3.1.6.1.5.0
BgpeStatesEstablished	1.3.6.1.3.1.6.1.6.0
//traffic table	//traffic table
BgpeTrafficSourceAddress	1.3.6.1.3.1.7.1.1
BgpeTrafficDestAddress	1.3.6.1.3.1.7.1.2
BgpeTrafficInOCtets	1.3.6.1.3.1.7.1.3
BgpeTrafficOutOCtets	1.3.6.1.3.1.7.1.4
//cada participante	//cada participante
BgpeSessionASNumber	1.3.6.1.3.1.8.1.1.
BgpeSessionRemoteAddress	1.3.6.1.3.1.8.1.2.
BgpeSessionInOctets	1.3.6.1.3.1.8.1.3.
BgpeSessionOutOctets	1.3.6.1.3.1.8.1.4.
BgpeSessionStatus	1.3.6.1.3.1.8.1.5.
BgpeSessionUpTime	1.3.6.1.3.1.8.1.6.
BgpeSessionTotalPath	1.3.6.1.3.1.8.1.7.
BgpeSessionOriginIgp	1.3.6.1.3.1.8.1.8.
BgpeSessionOriginEgp	1.3.6.1.3.1.8.1.9.
BgpeSessionOriginIncomplete	1.3.6.1.3.1.8.1.10.
BgpeSessionUpdates	1.3.6.1.3.1.8.1.11.
BgpeSessionLastError	1.3.6.1.3.1.8.1.12.
BgpeSessionPathLen	1.3.6.1.3.1.9.1.1.

BGPeConf

Tabela A-6 - Objetos BGPeConf

Nome do Objeto Definido	OID
//globais	//globais
BgpeConfIPSwitch	1.3.6.1.3.2.1.0
BgpeConfIPRouteServers	1.3.6.1.3.2.2.0
BgpeConfVLanPos	1.3.6.1.3.2.3.0
BgpeConfMaxPath	1.3.6.1.3.2.4.0
BgpeConfMinPath	1.3.6.1.3.2.5.0
BgpeConfRouteServerCPUMax	1.3.6.1.3.2.6.0
BgpeConfRouteServerMemoryMax	1.3.6.1.3.2.7.0
BgpeConfRouteServerProcessMax	1.3.6.1.3.2.8.0
BgpeConfMailDest	1.3.6.1.3.2.9.0
BgpeConfSMTP	1.3.6.1.3.2.10.0
BgpeConfTrapDest	1.3.6.1.3.2.11.0
BgpeConfASDeny	1.3.6.1.3.2.12.0
BgpeConfDamp	1.3.6.1.3.2.13.0
BgpeConfPathLog	1.3.6.1.3.2.14.0
BgpeConfReportTime	1.3.6.1.3.2.15.0
BgpeConfPrivateNetworks	1.3.6.1.3.2.16.0
//cada participante	//cada participante
BgpeConfSessionIP	1.3.6.1.3.2.17.1.1
BgpeConfSessionSwitchPos	1.3.6.1.3.2.17.1.2
BgpeConfSessionDescr	1.3.6.1.3.2.17.1.3
BgpeConfSessionASNumber	1.3.6.1.3.2.17.1.4
BgpeConfSessionMaxPath	1.3.6.1.3.2.17.1.5
BgpeConfSessionMinPath	1.3.6.1.3.2.17.1.6
BgpeConfSessionBandwithMax	1.3.6.1.3.2.17.1.7
BgpeConfSessionBandwithMin	1.3.6.1.3.2.17.1.8

BGPeRSUtil

Tabela A-7 - Objetos BGPeRSUtil

Nome do Objeto	OID
BgpeRSUtilDampennedNetworks	1.3.6.1.3.3.1
BgpeRSUtilMaxAnnounc	1.3.6.1.3.3.2
BgpeRSUtilRouteServerCPUUser	1.3.6.1.3.3.3
BgpeRSUtilRouteServerCPUSystem	1.3.6.1.3.3.4
BgpeRSUtilRouteServerCPUIdle	1.3.6.1.3.3.5
BgpeRSUtilRouteServerMemoryUsed	1.3.6.1.3.3.6
BgpeRSUtilRouteServerMemoryFree	1.3.6.1.3.3.7
BgpeRSUtilRouteServerProcess	1.3.6.1.3.3.8
BgpeRSUtilRouteServerLogGeneral	1.3.6.1.3.3.9

BGPeTraps

Tabela A-8 - Objetos BGPeTraps

Nome do Objeto	OID
BgpeTrapTraffic	1.3.6.1.3.4.1
BgpeTrapPath	1.3.6.1.3.4.2
BgpeTrapSession	1.3.6.1.3.4.3
BgpeTrapNextHop	1.3.6.1.3.4.4
BgpeTrapAsPath	1.3.6.1.3.4.5
BgpeTrapRouteServer	1.3.6.1.3.4.6

ANEXO H

É apresentado no Anexo H, o diagrama de classes completo do formato das mensagens SNMP, do *MibCompiler*, do Agente e da entidade Dual.

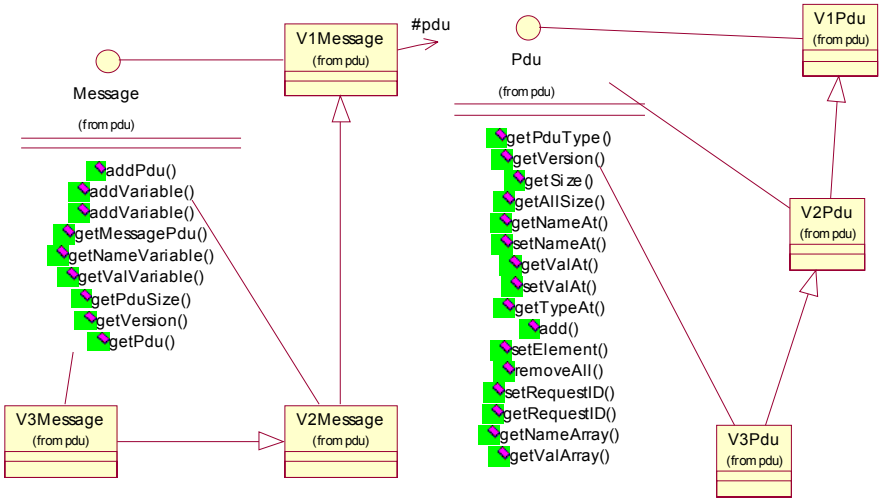

Figura A-16 - Diagrama de classes das mensagens SNMP

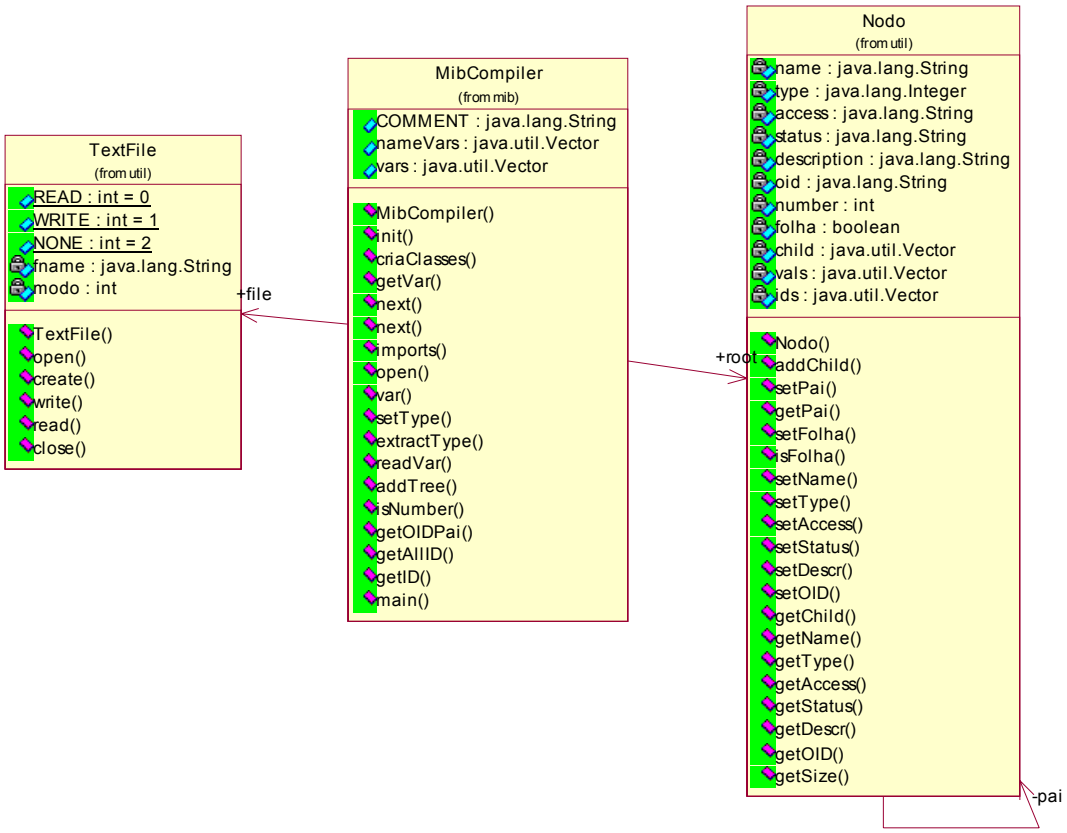

Figura A-17 - Diagrama de classes do *MibCompiler*

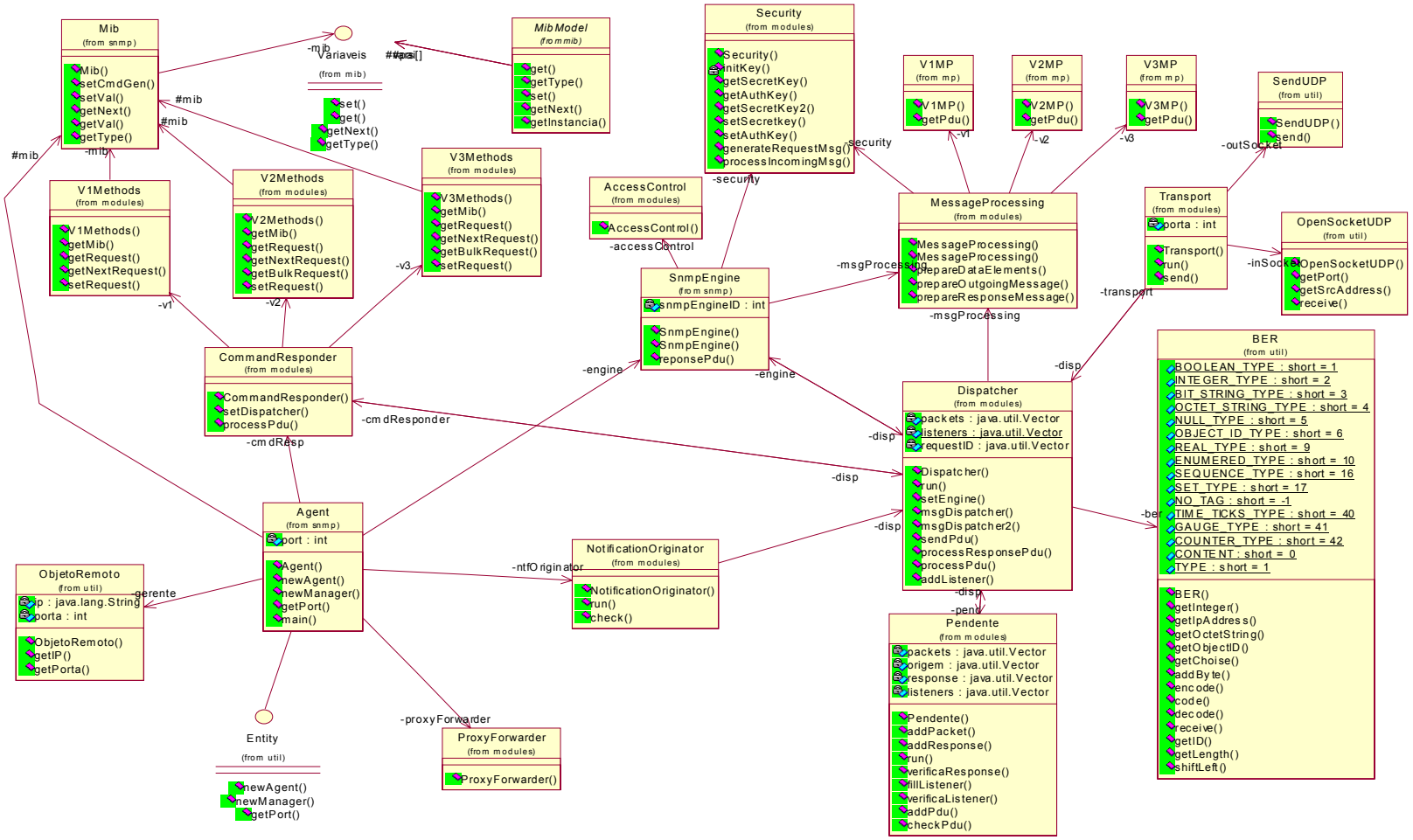

Figura A-18 - Diagrama de classes da entidade Agente

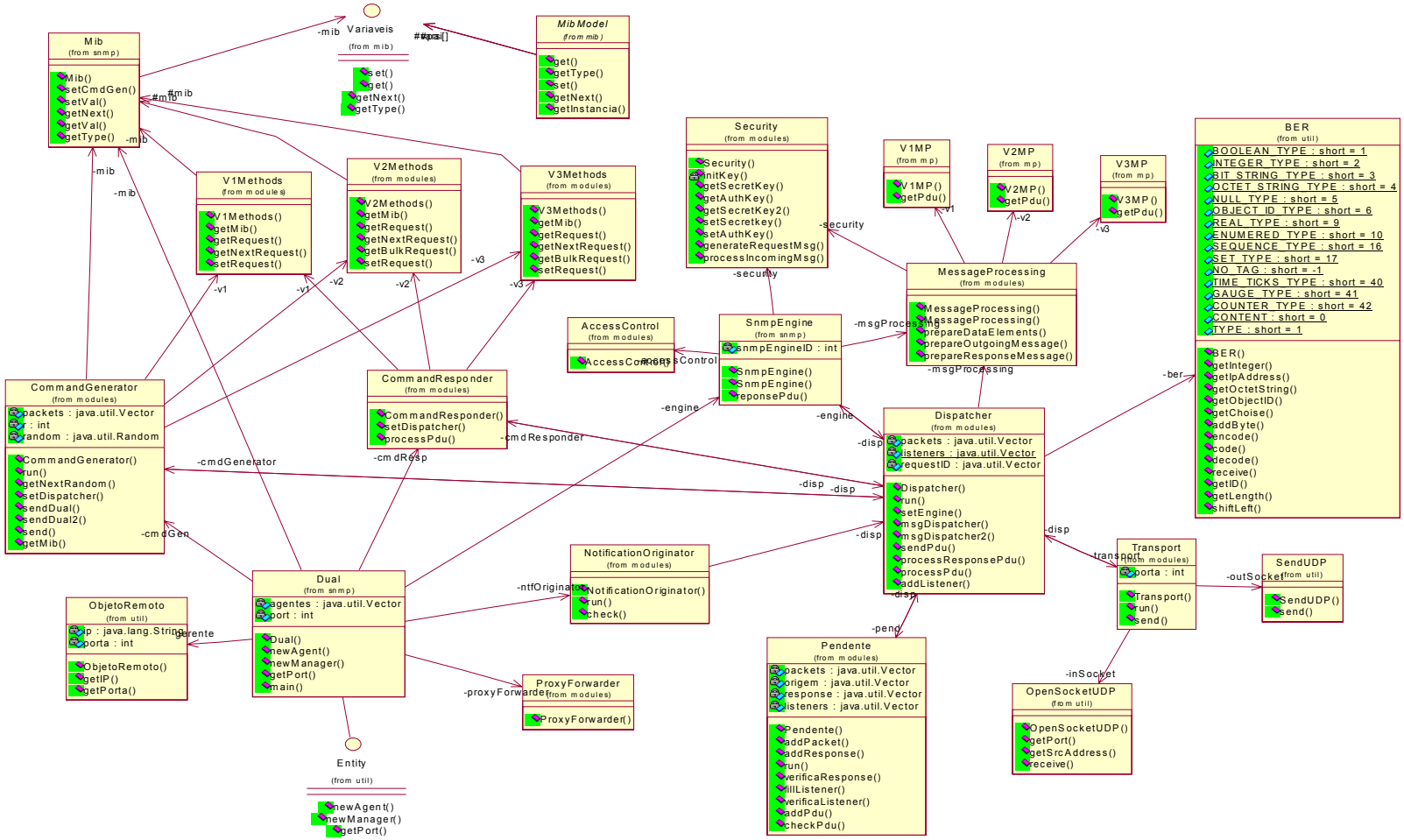

Figura A-19 - Diagrama de classes da entidade Dual