GateD

UNIX ルーティングデーモン
ユーザマニュアル

GATED コンソーシアム

January 22, 1997

Contents

Chapter 1

序章

GateD は核となるサービスとルーティングデータベースと複数の経路制御プロトコル (RIP バージョン 1 と 2、DCN HELLO、OSPF バージョン 2、EGP バージョン 2、BGP バージョン 2 から 4) をサポートしたプロトコルモジュールからなるソフトウェアです。GateD は NSFNET と各地域のネットワークを相互接続し、ポリシルーティングによりフィルタリングを実装するために最初に使用されました。GateD はネットワーク管理者が、送り元と destination の自律システム (AS) や受け取るネットワークインタフェースと送り出すネットワークインタフェースや一つ前のホップルータや特定の destination アドレスによりプロトコルごとに経路情報の import や export を制御することが可能です。ネットワーク管理者は柔軟なフィルタリングを使用して import された経路情報の組合せごとに優先レベルを指定することができます。優先レベルが割り当てられると、GateD はプロトコルにかかわらずどの経路情報を使用するかを決定します。

GateD は経路制御プロトコルにより交換された情報から作成された経路情報データベースにより動的経路を扱うように設計されています。

1.1 文法表記法

このドキュメントで参照するファイル名は `typewriter` 形式で表示します。例えば、`/etc/gated.conf` は **gated** が使用する設定ファイルです。ファイル名がフルパスとなっていない場合は、ソースコードのルートディレクトリからのパス名となります。gated が `/usr/local/gated` で展開されている場合、`src/task.c` というファイルは、`/usr/local/gated/src/task.c` を意味します。OS のコマンドラインから起動するコマンドは同様に `typewriter` 形式で表示します。`echo "Hello world"` は画面に `Hello world` と表示します。

構文解析時に正確に表記されなければならないキーワードや特殊文字は **boldface** 形式で表示します。パラメータは *italic* 形式で表示します。角括弧 (‘[’ と ‘]’) 中のパラメータは省略可能なキーワードやパラメータが入ります。縦棒 (‘|’) はオプションの選択を示すために使用します。丸括弧 (‘(’ と ‘)’) は必要なキーワードやパラメータがひとまとまりの場合に使用します。

以下の文法表現を例に説明すると、

autonomoussystem *autonomous_system* [**loop** *number*]

角括弧は **loop** パラメータはオプションであることを意味しています。キーワードは **autonomoussystem** と **loop** です。*autonomous_system* は *variable definition* スタイルとなっているため、設定されなければならないパラメータであることを意味しています。

C ライブラリは、システムコールと同様に *italic* スタイルで示します。

1.2 最初にすべきこと

GateD は新しいユーザにとってはしばしばフラストレーションの元になってしまう非常に複雑なシステムで

す。このセクションでは **gated** に慣れていない方へのいくつかのガイドラインを示し、GateD をどのように運用し始めるかを説明します。GateD は UNIX マシンをインターネットルータを構成することになり、正しくかつ有効な経路制御スキームを設計するために IP と IP のルーティングプロトコルに慣れておく必要があることに留意してください。

最初にするべきことは、まだインストールしていなければ **gated** をシステムにインストールすることです。chapter 3 では gated をどこから入手し、どのように作成しインストールするかを説明します。同時にそこではいくつかの OS についてのアドバイスも示します。

そして **gated** の設定が次のステップです。どのように動くのかを得るために **gated** は設定ファイル (通常は `/etc/gated.conf`) を読み込みます。**gated** はルーティングプロトコルを処理し、UNIX マシンをルータにするようになっているため、設定パラメータは非常に複雑になり得ます。Chapter 4 では設定ファイルの文法やセマンティックスの詳細を説明します。この章は **gated** がサポートしている多くのプロトコルと機能を説明しているため非常に長くなっています。この章は全部読む必要は無く、導入部分、経路情報、preference、トレースや実際に使用するプロトコルだけでも構いません。ポリシーセクション (Chapter 4.4) も非常に重要でほとんどすべての設定に応用されています。

次に、**gated** の実行を開始し制御します。**gated** のコマンドラインの文法とオプションや **gated** を制御 (起動、停止、再設定) するツールは Chapter 5 で説明します。この章では RIP と OSPF の動作をモニタするツールも説明します。

ここまででうまくいかなかったり、gated が期待どおりに動作していない場合は、gated を使用しての経験から得た共通の問題は Chapter 6 により解決することができます。これによりすべての問題に対する対処法が得られるわけではありませんが、問題を解決するための良い指針が得られます。4.29 ではいくつかのもっとも共通な設定ファイルを紹介します。その設定ファイルでは異なる statement を使用しています。標準的なルータのセットアップのいくつかもそこで示しますが、それは実際の設定での基本として使用することもできます。

GateD を使用している方はいくつかのメーリングリストに参加することができます。GateD コンソーシアムのオフィシャルなメーリングリストには **GateD** の一般的な質問用として `gated-people@gated.merit.edu` が、**GateD** のアルファリリースについての質問用として `gated-alpha@gated.merit.edu` があります。GateD の開発者や技術者が利用するのに `gated@gated.merit.edu` あるいはバグレポートとして `gated-bugs@gated.merit.edu` もあります。

Chapter 7 は **gated** 開発者や **gated** に新しいプロトコルをポーティングに従事している方のためのものです。そこでは **gated** の基本的な内部構造をいくつか詳細に説明します。

1.3 謝辞

Scott Brim `<swb@gated.cornell.edu>` はコンセプトと継続的な管理的な支援、全米科学財団は財政的な支援、Mark Fedor `<fedor@psi.com>` はリリース 1.3.1 までの開発とメンテナンスと継続的なアドバイス、Dennis Ferguson `<dennis@canet.ca>` は BGP と gated の再設計箇所での多大なる支援、Art Harkin `<ash@hpindda.cup.hp.com>` は拡張テストや多くのバグフィックス、David Robinson `<david@elroy.jpl.nasa.gov>` は拡張テストや多くのバグフィックス、Rob Coltun `<rcoltun@ni.umd.edu>` は UMd OSPF のコードの作者であり継続的な支援、Mike Petry `<petry@ni.umd.edu>` はオリジナルの HELLO のコードの作者、Thomas Pusateri `<pusateri@netedge.com>` は DVMRP、IGMP、PIM のコードの作者、Igor E. Chechik `<chech@kaija.spb.su>` は SystemV への変更、カリフォルニア大バークレー校の方は gated が最初にベースにした routed のコードと継続的な支援、IBM Austin の方には SMUX インタフェースの最初の実装、IBM Raleigh の方には (ゴルフをしていないときに) 多大な拡張のテスト、Robert Hagens `<hagens@ans.net>` と Dan Schuh `<schuh@cs.wisc.edu>` は最初の Wisconsin IS-IS のコードの作者、Steve Heimlich `<heimlich@watson.ibm.com>` は gated での IS-IS の修正と現在の開発でそれぞれ貢献していただきました。

このドキュメントは ISI からのいくつかの (GateD の内部についての) ドキュメントと同様に 3.5 の HTML ドキュメントを編集したもので、GateD コンソーシアムでの多くの再フォーマットと処理を行っています。

1.4 日本語ドキュメントについて

このドキュメントは gated-R3_6Alpha_2/doc/tex 以下を kondou@uxd.fc.nec.co.jp がボランティアで日本語に翻訳したものです。翻訳者は英語にそれほど精通しているわけではありませんが、翻訳者の判断で意訳し、日本語としてできる限り読みやすいようにしたつもりです。一部原文に明らかな誤りがありましたので正しい内容で翻訳したつもりです。説明として意味不明な点がありましたら、原文を参照し、原文から内容を理解していただけると助かります。原文とてらし合わせても説明が不明だとか正しく翻訳できていないと思われましたら、kondou@uxd.fc.nec.co.jp 宛にご連絡いただけると助かります。ドキュメントに反映するよう努力したいと思います。

日本語訳のドキュメントの配布条件は gated-R3_6Alpha_2/Licensing にしたがうこととします。

Chapter 2

GateD コンソーシアム

2.1 GateD コンソーシアムの概要

Cornell Gatedaemon Project ではネットワークを相互接続するために世界中で使用されている経路制御ソフトウェアである "GateD"(ゲートディーと読みます) を製作しています。GateD は IP と ISO ネットワークの複数のプロトコルをサポートしています。GateD は研究者や実装者や企業で以下のように使用されています。

- 新たなルーティングプロトコルのプラットフォーム
- ルーティングプロトコルのテスト実装のリファレンスソース
- 新たなプロトコルの実装機構の配布
- 商用製品の開発の核となる機構の提供
- 時代をリードするネットワーク研究と開発へのアクセス手段

GateD コンソーシアムは GateD の機能を開発するために存在している現状の成功した共同作業を支援し、拡張するための正式な機構となっています。コンソーシアムでは以下のことを行います。

- キーとなるテクノロジー間のコミュニケーションと交換の改善
- 必要とされている機能がよりタイムリーに配布されるようにする
- 将来を確実にするためのより広い基金の創設
- ライセンス処理の効率化
- 研究機関と企業間の技術提供の改善

コンソーシアムメンバはインターネットのルーティングプロトコルの開発をサポートし、かかわっているすべての機関に開かれています。コンソーシアムメンバになることが GateD のソースコードライセンスの取得に必要となっているわけではありません。

コンソーシアムメンバの特典には以下のことがあります。

- 継続的に GateD がインターネット上で自由に利用可能となる保障
- 機能開発や目標の設定での活発な議論
- 新たな GateD の機能や可能性についての詳細な技術集会

- メンバが従業員を送り込んだり、GateD の開発スタッフとルーティングソフトウェアについて学ぶというようなインターン制度
- GateD に貢献している従業員がいる機関がその作業に対してコンソーシアムからの基金をうけることができるという開発者の身分の保障

メンバはスライド制によって財政的な支援をすることになります。現在次の 4 つのレベルのメンバがあります。

- スポンサー: U.S. $50,000
- 準会員: U.S. $25,000
- 会員: U.S. $15,000
- 大学生またはスタートアップ: U.S. $5,000

Merit GateD コンソーシアムに加わるためにはコンソーシアム契約にサインし、メンバ料を支払わなければいけません。詳しくは下記に問い合わせてください。

```
GateD Consortium
Merit Network, Inc.
4251 Plymouth Road
Ann Arbor, Mi 48105-2785

phone: (313) 764-9430
fax:   (313) 747-3185
email: consort-interest@gated.merit.edu
```

この情報は `http://www.merit.edu` の GateD Consortium Welcome Page からも利用できます。

2.2 活動の概要

Merit GateDaemon プロジェクトは複数のルーティングプロトコルとプロトコルファミリをサポートしているソフトウェアである **GateD** を製作しています。プロジェクトはもともと全米科学財団が 1987 年からサポートを開始しました。GateD は初期の NSFNet と新たな地域ネットワークを相互接続するために最初に使用され、その後、地域ネットワーク内のサイト間を接続するために使用されました。現在 GateD は世界中の何百もの機関で使用されています。

Merit の目的はインターネット上の最新のルーティングプロトコルをサポートするために「すでに開発された」ソフトウェアプラットフォームを提供することです。このソフトウェアはルーティングプロトコルの研究開発として便利で有効なプラットフォームを提供します。インターネットの急速な成長にともなって拡大する問題の解決法に現在のインターネットコミュニティからの注目が注がれるようになると、研究やプロトタイププラットフォームとしての GateD の必要性が引続き増大するでしょう。現在、GateD は多くのルーティングプロトコルの事実上のレファレンス実装として使用されています。

GateDaemon は相互運用可能なネットワーク製品の製造を行い、支援するために使用することができます。また、多大な初期投資無しで、時代をリードするネットワーク研究と開発へのアクセス手段を提供します。GateD はリリースされて製品として提供されるまでにインターネット上で広範囲にテストされています。GateD はパブリックドメインソフトと多くの商用製品の主要部分のいずれにも広く行き渡っているため、インターネットの今の成長に耐えることが必要なルーティングの拡張に対する強力な機構となります。

GateD は個々の使用において Web、FTP、gopher 経由で入手できます。一般の利益となるようないかなる機能強化も再配布されるためにプロジェクトにそれをフィードバックすることに同意する限り無償で誰に対しても再配布のライセンスが与えられます。この方式は圧倒的な成功をおさめ、特に変更やあらたなプロトコルの実装に対するユーザの貢献を助長しています。プロトコルへの拡張や実装の貢献は一般配布パッケージに

統合されています。他の多くの企業が同様なことをしていることを知っている場合、企業は従業員に企業の成果をインターネット全体に利用可能となるように強力に指示します。実際、研究機関から私企業あるいはその逆への技術移転はこのプロジェクトによりうまい具合に実証されています。

ユーザからはプロジェクトで対応できる以上のことを要望されています。Cornell は GateDaemon の開発を促し、拡大するために GateDaemon コンソーシアムを創設しました。Merit は 1995 年 9 月からこの作業を促進するようになりました。ライセンスや契約料の形式を採る代わりに、Merit GateD コンソーシアムは引続きコンソーシアムメンバから財政支援により資金援助と供託金を増やし、2000 年までインターネットに供給します。Merit はコンソーシアムメンバの利益を拡張し、ルータソフトウェアの開発とテストを支援するためのテストツールのような特別な利益を与えるつもりです。

コンソーシアムは新たな GateD の機能や可能性についての詳細な技術集会を主催します。コンソーシアムのメンバは GateD の開発者とともにあらたな機能とリリースの目的についての活発な議論に参加します。開発チームとともに作業するために Merit にスタッフを送り込みたいコンソーシアムメンバにはインターン制度や身分の保障が用意されています。プロジェクトやコンソーシアムの活動についての改善された情報もハードコピーと電子媒体で用意されています。あらたな機関が参加しやすいようにすることによって、コンソーシアムは研究者と開発者のこれまでの協調を奨励強化しています。

Merit は引続き NSF などのような研究や国家的な情報インフラやインターネット全体としての福祉を支援することが目的の政府機関や機構からの支援を求めています。コンソーシアムのメンバはインターネットやインターネットのルーティングプロトコルに興味を持つ国のいかなる機関にも開かれています。コンソーシアムメンバになることは再配布ライセンスにあたって強く奨励されてはいますが、必要ではありません。GateD のソースコードに固有な拡張をしたい企業は考慮の上個々に開発プロジェクトを起こすことができます。このようなプロジェクトは個々に評価されます。

コンソーシアムの中でのメンバとしての貢献により、優れたルーティングプロトコルの実装がインターネットや運用者や開発者やユーザへの重要な恩恵を通じて利用可能になります。ここ数年以上 GateD は評価に値することを示してきています。次の 4 年間に起こり得るであろう問題はこれまで以上に複雑なものでしょう。インターネットは引続き爆発的な成長を遂げるでしょう。企業と研究機関の協調は成長がもたらす挑戦をあきらかにするために重要なことです。GateD こそが行われなければいけない解決の道なのです。

2.3 GateD ライセンス

GateD は自由に利用可能ですが、その再配布は制限されています。GateD を再配布したい方は Merit との同意のもとにおいて行うことができます。いかなる第三者に GateD の一部あるいはすべてを渡したい方は最初に再配布しようとするバージョンについての GateD 再配布ライセンスを得る必要があります。GateD 再配布ライセンスはライセンス料を必要としません。その代わりライセンスを受けた方は GateD への変更や拡張はユーザに配布するために Merit に渡す必要があります。ライセンスを受けた方が開発した拡張の配布は GateD ユーザに対する利益であり、GateD のソースファイルの制限と可能ならば GateD の再配布ライセンスにのみ従います。

Chapter 3

インストール

GateD の Makefile システムは同じソースを使用して異なるアーキテクチャの gated のコンパイルが可能となっています。個々のアーキテクチャごとにマシン固有のオブジェクトファイルを保持するためにオブジェクトディレクトリを別々に使用するようになっています。オブジェクトディレクトリはいくつかの設定ファイルと大部分が GateD ファイルへのシンボリックリンクから構成されます。ディレクトリの名前は `obj.<arch>` になっていて、個々のシステムのアーキテクチャに関連づけられています。GateD バージョン 3.6 については新たに自動設定処理が設けられています。この機能は限定されたアーキテクチャのみ使用可能なため、(バージョン 3.6 より前の) 古い gated インストール処理を使用することが必要となるかもしれません。

3.1 自動設定

この処理は SunOS4.1.x、SunOS5.x、BSD/OS、Ultrix4.x でサポートされています。これ以外のシステムで gated を使用する場合は 3.2 で説明するこれまでの手動による設定を使用することができます。

1. カレントディレクトリをソースディレクトリ (src) にします
   ```
   cd src
   ```
2. 自動設定プログラムを実行します
   ```
   make autoconf
   ```
3. すべての問い合わせに答えて、それが終了すれば gated をコンパイルします
   ```
   make
   ```
4. gated をインストールします
   ```
   make install
   ```
5. 最後にゴミを消去します
   ```
   make clean
   ```

処理の詳細については section 7.2 で設定処理の個々の過程を説明します。要するに、`make autoconf` はオブジェクトディレクトリを作成します。ディレクトリ名は `obj.<arch>` で、`<arch>` は `util/configure -a` の出力結果から得られたものです。このオブジェクトディレクトリには `Config` というファイルがすべての必要なオプションとパラメータとともに作成されています。

オプションを変更するために、この `Config` を変更することができます。その後、オブジェクトディレクトリのファイルの更新を行うために、`make config` を実行する必要があります。

`make autoconf` ではソース (src) ディレクトリの `Config` というファイルを参照します。このファイルは `make autoconf` 処理で問い合わせられていない、ユーザ指定の優先度を設定するために使用されます。例え

ば、`make autoconf` では **cc** よりも GNU **gcc** コンパイラを選択します。**cc** を強制的に使用したい場合は `src/Config` に以下の一文がなければいけません。

```
cc       cc
```

最終的な `src/obj.<arch>/Config` ファイルは `src/Config` と `src/configs` 中のシステム固有なファイルの組み合わせになります。`src/configs` 中のファイルは編集すべきではありません。

3.2 手動設定

これはバージョン 3.6 より前までの **GateD** の配布では通常の作業です。この作業を 3.6 以上の配布で行う場合は、ソースファイル中の `Makefile` を置き換える必要があります。

```
cd src
cp Makefile.org Makefile
```

その後、以下にしたがって作業を行います。

1. オブジェクトディレクトリ `src/obj.‘src/util/archtype‘` を作成します。アーキテクチャを表示させるためには直接 `src/util/archtype` を実行します。GateD を別のアーキテクチャでコンパイルすることがないならば、`obj` を使用してもかまいません。例えば、*Sun SPARCstation* で *SunOS 4.1.1* が動いている場合は次のようになります。

   ```
   sun% src/util/archtype
   SunOS-4.1.1-sun4
   sun% mkdir src/obj.‘src/util/archtype‘
   sun% ls -d src/obj.*
   src/obj.SunOS-4.1.1-sun4/
   ```

   BSD 4.3 Reno やそれ以降のバージョンでは、オブジェクトディレクトリが、システムソースが使用しているフォーマットでサポートされています。これは、ハッキングにより可能です。このようなシステムでは、ディレクトリへのフルパスを指定する必要があります。例として、`src/configs/vangogh` を参照してください。

2. 設定ファイルを作成します。このファイルは Makefile とシステム固有のヘッダファイルを構築するための設定スクリプトへの入力に使用されます。

   設定ファイルでのオプションの説明は `src/configs/README` で行っています。

   ごく小数ですが設定ファイル例が `src/configs` にあるので、最も近いものを選んで、固有の設定に作り変えてください。その後、`Config` という名前で先に作成したオブジェクトディレクトリにインストールしてください。

3. ’`make config`’ と入力してこのアーキテクチャの GateD を作成してください。これにより、`obj.‘src/util/archtype‘/Config` をもとに awk スクリプトを実行させ、sed スクリプトを作成します。この sed スクリプトは Makefile のテンプレートを編集するのに使用されます。Makefile により、アーキテクチャ固有のファイルを設定し、シンボリックリンクを作成し、依存関係を調べます。`make config` は `src` ディレクトリで実行されなければいけません。

4. `src` またはオブジェクトディレクトリで `make` と入力して GateD を作成します。

5. root になり、ソースディレクトリまたはオブジェクトディレクトリで’`make install`’ により GateD をインストールします。さらに必要ならば同じディレクトリで `make install-man` と入力することによりマニュアルをインストールします。

6. 設定ファイルを作成します。conf ディレクトリに設定ファイル例がありますし、マニュアルに設定オプションがすべて説明されています。設定ファイルを `/etc/gated.conf` にインストールします。
7. ふたをして... 火をつけます。

3.3 注意事項

3.3.1 UDP チェックサム

RIP はカーネルでの UDP のチェックサムが無効となっていると判断した場合、動作しません。UDP チェックサム無しで動作した場合、特にシリアルリンクの場合、不正確な経路情報が伝搬するおそれがあります。このチェックは RIP パケットがチェックサム無しで受信したかどうかを判断するためのものでは無く、少なくともチェックサム無しのパケットを発生しないようにし、このような問題点があることを認識してもらうためのものです。

UDP チェックサムを有効にするためには二通りの方法があります。デフォルトで、OS としてチェックサムを有効にすることが可能なようにソースコードが用意されているかもしれません。*SunOS* では `/sys/netinet/in_proto.c` により可能です。この場合、ソースを変更し、カーネルの再構築を行ってください。

OS で関連するソースコードが用意されていない場合、(root で) 起動中の OS とディスクイメージに以下と同様な方法でパッチを当てることができます。

```
sun# adb -k -w /vmunix /dev/mem
_udp_cksum/W 1
_udp_cksum:     0x0             =       0x1
_udp_cksum?W 1
_udp_cksum:     0x0             =       0x1
^D
sun#
```

UDP チェックサムの変数名はおそらく以下のようにして見つけることができます。

```
sun# nm -o /vmunix | grep udp_c
/vmunix:f80fa35c D _udp_cksum
/vmunix:f801bd10 T _udp_ctlinput
sun#
```

3.3.2 IP マルチキャストサポート

OSPF や RIP の実装 (将来の希望としてはルータディスカバリも) では IP マルチキャスト機能を使用します。この機能が無い場合にはプロトコルの機能が制限されます。

BSDI の *BSD/386 1.0*、*Sun* の *Solaris 2.0* や シリコングラフィックスの *IRIX* などのいくつかのシステムには IP マルチキャストをサポートして出荷されています。他のシステムについては (例えば *SunOS 4.1.** やいくつかのバージョンの *Ultrix*)、IP マルチキャストのサポート `ftp://gregorio.stanford.edu/vmtp-ip` や `gregorio.stanford.edu` や `ftp://parcftp.xerox.com` や `ftp://ftp.isi.edu/mbone` の FTP サイトをチェックしてみてください。

RIP-II や OSPF では point-to-point のネットワークインタフェースでの IP マルチキャストの使用を指定します。多くの IP マルチキャストの実装のバグのため、GateD はこのようなネットワークインタフェースでのマルチキャストパケットの使用ができません。そのため、GateD では自動的に point-to-point のリンクの destination アドレスを使用するようにしてあります。OSPF の場合には機能低下はありませんが、RIP-II の場合にはこのようなネットワークインタフェース経由で任意のサブネットマスクを通すことができなくなります。

IP マルチキャストのもう一つのバグによって、IP マルチキャストのデフォルトルートや指定したグループの経路が無い場合に、自分が属しているマルチキャストアドレスへのパケットの送信が失敗します。これを回避するため、GateD では自分が送信しようとするマルチキャストアドレスへの経路情報がループバックインタ

フェース経由となるように設定します。この設定は実際には使用されませんが、このグループへのパケットの送信のカーネルバグを回避します。

IP マルチキャストルーティングプロトコル

GateD ではマルチキャストルーティングプロトコルをサポートしています。この実装は IGMP のルータ部分となっていて、これによりローカルのグループメンバの情報を処理します。現時点では、PIM(Dense Mode) と Distance Vector Multicast Routing Protocol (DVMRP) をサポートしてます。これは Xerox Parc からの IP マルチキャスト配布のリリース 3.3 と 3.5 と互換性があるようにしようとしています。

GateD は以下の三つのマルチキャストカーネルのうちのいずれかで設定できます。これはカーネル内のマルチキャストフォワーディングテーブルがどのように作成されているかにより決まります。

KRT_IPMULTI_NOCACHE これはカーネル内にプロトコルに依存しないマルチキャストフォワーディングキャッシュが無い場合です。そのため、このタイプのカーネルでは GateD で IP マルチキャストルーティングを使用することはできません。これにはすべての Xerox Parc やスタンフォード大学の IP マルチキャストソフトウェアのバージョン 1.x や 2.x が含まれます。使用するカーネルがこれに当てはまる場合、残念ながら GateD で IP マルチキャストルーティングを使用することはできません。

KRT_IPMULTI_RTSOCK リリース 1.x や 2.x では DVMRP に依存していたため、何人かの GateD 開発者がプロトコルに依存しないカーネルを設計しました。その設計は BSD 4.3 reno や (ルーティングソケットを持つ) それ以降のシステムに備わっている機能があることを前提にしています。あるグループのマルチキャストパケットを受けた場合、カーネル内のフォワーディングキャッシュエントリにソースアドレスとの対でそのアドレスが存在していない場合、その対のテーブルを解決するためにルーティングソケット上にその要求を行います。(ルーティングソケットで待っている) ルーティングデーモンは、その要求を得て、カーネルにフォワーディングキャッシュエントリを作成します。

新しいカーネルのもう一つの目的は仮想ネットワークインタフェース (vif) を使用しないようにすることでした。代わりに、それ自身の ifnet 構造体を持つ仮想トンネルインタフェースが実装されました。これにより、トンネルネットワークインタフェースはあたかも別の point-to-point のネットワークインタフェースに見せることができたため、カーネルのソフトウェアをかなりシンプルにすることができました。

この実装では BSD 4.3 Reno やそれ以降のシステムでサポートされている radix tree を使用することができます。グループアドレスとソースアドレスの対は BSD 4.3 でのハッシュベースのテーブルとは対象的に radix tree の単一キーとして設定されます。ほとんどの BSD 4.4 Lite に対して適用可能なカーネルのソースパッチは `ftp://ftp.gated.cornell.edu/pub/multicast` から入手できます。

注意: 未実装です

KRT_IPMULTI_TTL0 この場合もカーネルはプロトコルに関係なくマルチキャストフォワーディングキャッシュを持っています。フォワーディングキャッシュのエントリはマルチキャストパケットをフォワードするときに要求されて作成されます。この場合、ルーティングデーモンとデータの交換をするために IGMP のソケット上のメッセージを使用し、ファワーディングキャッシュのエントリを求めるように要求します。この場合の使用例は Xerox Parc の IP マルチキャスト 3.3 と 3.5 のカーネルリリースに含まれています。

3.3.3 SunOS 4.0 での GateD の起動

SunOS 4.0 上での GateD が *sendto()* network unreachable でエラーになる場合、`/etc/rc.local` で `‘hostname‘` を `ifconfig` コマンドで `ie0/le0/ec0` に追加してください。そうしないと *SunOS* は各ネットワークインタフェースへの経路を誤って解釈してしまいます。

異なるカーネルアーキテクチャ間でバイナリでのカーネルのメモリ読み込みに互換性を持たせるため、Sun では libkvm.a が用意されています。残念なことに、このライブラリのダイナミックリンク版は *SunOS 4.** ではうまく動かないため、GateD はスタティックリンクされていなけけばいけません。これにより、あるカーネルアーキテクチャ(*sun4m* とします) でコンパイルされた GateD のバイナリが別のアーキテクチャ(*sun4c* とします) では動作することができません。

3.3.4 AIX 3.2 での GateD の起動

AIX 3.2 は可変長サブネットマスクとルーティングソケットとともに *BSD 4.3 Reno* をベースにしたネットワークコードを使用しています。*BSD 4.3* 互換モード (`compat_43` という変数が `no` になっているかで確認できます) で起動していない場合に拡張機能を使用することができます。これには (GateD が **ripquery** に応答するときに使用する) RIP パケットの destination アドレスを決めることができるというのも含まれます。GateD は機能がやや劣った *BSD 4.3* 互換モードでも動作することもできます。この場合は `-D_BSD=44` を付けてコンパイルするようにしてください。

AIX 3 上でのデバッグをしやすくするために core を生成する場合は、デフォルトの core の大きさの制限値を大きくしておく必要があります。これはシェル経由で設定するか、**gdc** で GateD を起動した場合に可能です。また、これがうまく動作するためには以下のコンパイル時の設定が必要です。`obj/Config` で `GDC_CORESIZE=RLIM_INFINITY` を定義するか、`GDC_RESOURCE` を定義して起動時に `gdc` のオプションに`-c` を付けてください。

3.3.5 シェアードライブラリがあるシステムでの GateD のコンパイル

`task_stdio_read()` で assertion failure が起きるのはファイルディスクリプタが正常にクローズされていないためです。これは *named resolver* ライブラリがシステムのシェアードライブラリに正常にインストールされていない時に起きます。シェアードライブラリで使用されているソケットがスタティックに -1 に初期化されていない場合、GateD が `endhostent()` を呼んだ時にファイルディスクリプタ 0 がクローズされてしまいます。この解決法はシェアードライブラリを正しくすることです。回避するためには、設定ファイルでシンボル名を一切使わないようにし、かつ `options noresolv ;` を指定します。

3.3.6 AIX 3.1 と 3.2 での GateD の起動

少なくとも *AIX 3.1* と *3.2* のいくつかのバージョンでは `yacc` に問題があることが報告されています。その問題のうちの一つは、`yacc` が呼出し側に解析エラーを報告しないということです。そのため、GateD が不正確な設定で起動しようとしてしまいます。その代わりに、*GNU* `bison` を入手することを強く勧めます。`ftp://prep.ai.mit.edu/pub/gnu` から入手可能です。

3.3.7 BSD 4.2 ベースでの HELLO や EGP の起動

Ultrix 1.2, 2.0, 2.1 や *SunOS 3.x* のような BSD 4.2 ベースのシステムで HELLO や EGP を動かしたい場合、以下のコードをそれぞれのソースコードに追加して、カーネルの再構築を行ってください。

```
/sys/netinet/in.h:

#define IPPROTO_EGP 8 /* exterior gateway protocol */

#define IPPROTO_HELLO 63 /* Fuzzball HELLO protocol */
```

SunOS 3.x での /sys/netinet/in_proto.c:

```
{ SOCK_RAW,PF_INET,IPPROTO_HELLO,PR_ATOMIC|PR_ADDR,
  rip_input,rip_output,0,0,
  raw_usrreq,
  0,0,0,0,
},
{ SOCK_RAW,PF_INET,IPPROTO_EGP,PR_ATOMIC|PR_ADDR,
  rip_input,rip_output,0,0,
  raw_usrreq,
```

```
  0,0,0,0,
},
```

Ultrix 1.2, 2.0 と *2.2* での /sys/netinet/in_proto.c:

```
{ SOCK_RAW,     &inetdomain,    IPPROTO_HELLO,    PR_ATOMIC|PR_ADDR,
  rip_input,    rip_output,     0,               0,
  raw_usrreq,
  0,            0,              0,               0,
  0,            0,              0,
},
{ SOCK_RAW,     &inetdomain,    IPPROTO_EGP,     PR_ATOMIC|PR_ADDR,
  rip_input,    rip_output,     0,               0,
  raw_usrreq,
  0,            0,              0,               0,
  0,            0,              0,
},
```

3.4 ISODE SMUX インタフェース

このバージョンの GateD *ISODE 7.0* SMUX インタフェース経由により、SNMP を使用することができます。*ISODE 7.0* をまだ入手していなければ、多くの SMUX/SNMP パッチが当たったバージョンを `gated.cornell.edu` の `pub/gated/isode-snmp-7.0.tar.Z.` を `ftp://gated.cornell.edu/pub/gated/isode-snmp-7.0.tar.Z` として入手することができます。

ISODE ソースには多くのディスク領域を必要としますが、すべての *ISODE* を作成インストールする必要は無く、核となる部分と snmp コードがあれば十分です。ただし、部分的なインストール (`inst-partial` と `inst-snmp`) は最低必要です。

GateD は *AIX 3* での *ISODE 6.8* ベースの SMUX インタフェースもサポートしています。次のようなエラーに遭遇した場合、

```
Building:       gated-mib.c
mosy -s -c gated-mib.c ../mib/smi.my ../mib/mib.my ../mib/rt.my ../mib/bgp.my ..
/mib/ospf.my ../mib/rip.my
mosy 6.8 of Sat Nov 23 12:30:27 CST 1991
line 56: Warning: EXPORTS but no ModuleIdentifier
last token read was ";"
object enterprises: Warning: IMPORTS but no ModuleIdentifier
object snmpEnableAuthenTraps: Warning: IMPORTS but no ModuleIdentifier
object snmpEnableAuthenTraps: syntax error
make: *** [gated-mib.c] Error 1
```

デフォルトの *ipForwardInfo* の二つ目の 0 を以下のように削除してください。

```
 DEFVAL  { { 0 } } -- 0.0
```

これは *ISODE 6.8* での mosy はこの文法が正しくても解析することができないためです。

Chapter 4

設定

4.1 GateD 設定ファイルについて

4.1.1 文法

GateD の設定ファイルはセミコロン (';') で終了する一連の statement から構成されます。statement はブランクやタブや改行からなるホワイトスペースで区切られたトークンで構成されます。この構造により、相互に関連した設定部やプロトコルにかかわる設定部の定義を簡素化しています。コメントには二つの形式があります。一つはパウンド記号 ('#') で始まり行の最後までのものです。もう一つは *C* スタイルの '/*' で始まり '*/' まで続くものです。

4.1.2 statement の配置

設定 statement により `gated.conf` を *options statement* 部、*interface statement* 部、*definition statement* 部、*protocol statement* 部、*static statement* 部、*control statement* 部や *aggregate statement* 部の順序で区切っています。statement の順序が正しくないと設定ファイルを解析時にエラーとなります。

他の二種類の *directive statement* と *trace statement* はこれには当てはまりません。この statement は設定ファイルの解析時に用いられ、設定ファイルによりトレースを制御します。この statement はプロトコルの設定には関係無く、`gated.conf` のどこにあっても構いません。

4.2 preference

preference は GateD が一つのプロトコルあるいは複数のプロトコルから得た経路の優先順位として使用する値です。preference はネットワークインタフェースやプロトコルやゲートウェイ毎に指定できます。

igp の経路選択を制御するために preference を使用しなくても構いませんが、これはメトリックベースで動作するプロトコルが自動的に行うためです。個々の peer や個々の AS からの同じ **egp** から得た経路を選択するために preference を使用することができます。

preference は設定ファイル中のいろいろな所で指定できますが、各経路にはそれぞれ一つしか preference の値はありません。単純に、各経路に指定された最も新しいあるいは最適な preference の値が使用される値となります。preference の値はある経路データベース中の同じ destination への経路情報の優先順位を決めるために使用される、比較のための値です。実際に使用される経路は最も低い preference 値を持つものが選択されます。

プロトコルによっては第二の preference (*preference2*) を持つものもあり、タイブレーク (同じ preference を持つ場合にさらにその優先度を決める) を行う場合に参照されることがあります。

4.2.1 経路情報選択

GateD は destination への最適な経路 (あるいは次 hop) を求めるために次のような規則を使用します。

- 最適な (数字が小さい) preference を持つ経路が選択されます。
- 二つの経路が同じ preference を持つ場合は最適な (数字が小さい) preference2 を持つ経路が選択されます。
- **egp** より得た経路よりは **igp** で得た経路の方が選択されます。**egp** により運ばれる間接的な **igp** の経路は最も優先順位が低くなります。
- AS 経路情報が使用可能な場合は、最適な経路の決定のために使用されます。
  - AS 経路を持つ経路は AS 経路を持たない経路よりも優先されます。
  - AS 経路と発信元が同じ場合、小さいメトリックの方が選択されます。
  - **egp** が発信元の AS 経路よりも **igp** が発信元の AS 経路の方が選択されます。最も優先されない AS 経路は **unknown** な発信元の場合です。
  - より短い AS 経路の方が選択されます。
- 二つの経路が同じプロトコルで同じ AS からの場合、小さいメトリックを持つ方が選択されます。
- 次 hop のアドレスが小さい方の経路を使用します。

4.2.2 preference の指定

デフォルトの preference は GateD が受信する経路の各発信元ごとに設定されます。preference 値の範囲は 0 から 255 で、最小値は最適な経路を示します。

以下の Table 4.1 にデフォルトの preference 値を示します。表では preference を指定する statement (このうちのいくつかは statement 内で指定されます) と、各 statement が指定している経路情報の種類を示します。各経路のタイプ毎にデフォルトの preference とプロトコル間の優先順位を示します。statement のレベルが小さくなるほど preference 値は高いものとなりますが、それに関係する経路情報は少なくなります。

preference	定義される箇所	デフォルト
ネットワークインタフェースのネットワークへの経路情報	interface	0
OSPF からの経路情報	ospf	10
IS-IS レベル 1 からの経路情報	isis level 1	15
IS-IS レベル 2 からの経路情報	isis level 2	18
GateD 内部で作成したデフォルトの経路情報	gendefault	20
リダイレクトされた経路情報	redirect	30
ルーティングソケットから得た経路情報	kernel	40
設定による静的経路情報	static	60
ANS SPF (SLSP) からの経路情報	slsp	70
HELLO からの経路情報	hello	90
RIP からの経路情報	rip	100
point-to-point のネットワークインタフェースへの経路情報		110
ダウンしたネットワークインタフェースへの経路情報	interfaces	120
aggregate/generate した経路情報	aggregate/generate	130
OSPF AS external からの経路情報	ospf	150
BGP から経路情報	bgp	170
EGP からの経路情報	egp	200

Table 4.1: GateD でのプロトコルごとのデフォルトの preference

4.2.3 preference の指定例

```
interfaces {
    interface 138.66.12.2 preference 10 ;
} ;

rip yes {
    preference 90 ;
} ;

import proto rip gateway 138.66.12.1 preference 75 ;
```

これらの statement では 138.66.12.1 から RIP 経由で得た経路情報の preference は 75 としています。138.66.12.1 というゲートウェイからの RIP 経由で得た経路情報に最終的に適用可能な preference は **import** statement で指定されます。他の RIP から得た経路情報に適用される preference は rip statement で指定しています。interface statement で設定した preference はそのインタフェースのみに適用されます。

4.3 プロトコル概要

4.3.1 プロトコル概要

経路制御プロトコルは各 destination に対する 最適の」経路情報を決め、ネットワーク上のシステムに経路情報を渡します。経路制御プロトコルは大きく二つのグループに分けられ、それぞれ *AS* 内部プロトコルと *AS* 外部プロトコルです。GateD では AS 内部と AS 外部の経路制御プロトコルを組み合わせた管理を一つのデーモンとして行っています。

4.3.2 AS 内部経路制御プロトコル

AS 内部経路制御プロトコルは自律システム (AS) 内の到達可能情報を交換するために使用されます。これは名前の頭文字を取って **igp** と呼ばれます。以下に AS 内部経路制御プロトコルをいくつか示します。

RIP

Routing Information Protocol (バージョン 1 とバージョン 2 があります) はもっともよく使用される AS 内部経路制御プロトコルです。RIP はもっとも低いメトリックを持つ経路情報を最適な経路情報として選択します。メトリックはデータが destination にたどり着くまでに通過しなければいけないゲートウェイの数を表すホップカウントのことです。RIP が受け取ることができる最大のホップカウントは 15 です。メトリックが 15 よりも大きくなると、destination は到達不可能と見なされ、GateD はその経路情報を廃棄します。RIP では最適な経路情報はもっとも少ないゲートウェイを通過する経路情報、すなわち経路中の混雑度や遅延を考慮しない最短経路だとしています。

RIP バージョン 1 は [RFC 1058] で、RIP バージョン 2 は [RFC 1723] で説明されています。

HELLO

もう一つの AS 内部経路制御プロトコル である HELLO では最適な経路情報を選択する要素として遅延時間を使用します。ラウンドトリップタイムはパケットが発信元から destination にたどり着くのにかかる時間です。NSFNET のプロトタイプの元となったバックボーンの *fuzzball* というゲートウェイ間で使用されていた経路制御プロトコルであるため、HELLO はインタネットにとっては歴史的に重要です。現在では fuzzballs でもそうであるように HELLO はほとんど使用されなくなった経路制御プロトコルです。

初期のバージョンの HELLO は [RFC 891] で説明されています。

OSPF

Open Shortest Path First は link-state を使用する経路制御プロトコルです。多くのゲートウェイが存在しているような複雑なネットワークでは RIP に比べると OSPF はより適しています。OSPF では同じコストを持つ経路情報を複数扱うことができます。

OSPF は [RFC 1583] で説明され、MIB は [RFC 1253] で定義されています。その他の関連文献は [RFC 1245]、[RFC 1246] および [RFC 1370] です。

IS-IS

Intermediate System to Intermediate System (ISIS) はもともと ISO/CLNP (International Organization for Standardization/ConnectionLess Network Protocol) のパケットの経路制御のために開発された link state を使用する interior gateway protocol (IGP) です。このバージョンの GateD では IP パケットと同様にルーティングすることができます。

4.3.3 AS 外部経路制御プロトコル

AS 外部経路制御プロトコルは AS 間で経路情報を交換するために使用されます。AS 外部経路制御プロトコルは AS が他の AS と経路情報を交換する場合にのみ必要となります。AS 内のルータは RIP のような AS 内部経路制御プロトコルが動作しています。他の AS と接続しているゲートウェイでのみ AS 外部経路制御プロトコルを動作させなければいけません。以下に GateD が現在サポートしている AS 外部経路制御プロトコルを示します。

EGP

Exterior Gateway Protocol (EGP) の経路到達可能性情報はもともと ARPANET/MILNET の "core" ゲートウェイに渡されていました。このゲートウェイでは最適な経路情報が選択され、その情報が接続されたすべての AS に返されます。インタネットが階層構造的ではなくなってくると、階層構造を前提とした EGP はそれほど効果的ではなくなってきました。

EGP は [RFC 827] と [RFC 904] で説明されています。

BGP

Border Gateway Protocol は AS 外部経路制御プロトコルの選択肢の一つとして EGP に取って代わるものです。BGP は AS 間で到達可能性情報を交換しますが、EGP よりも多くの機能があります。BGP では最適な経路情報を選択するために各経路情報についてより多くの情報を持つ経路属性を使用します。経路属性には例えば経路情報の決定で政治的、組織的あるいはセキュリティ(ポリシ) 的な考慮に基づく優先度を含むこともできます。BGP では階層的でないトポロジをサポートしていて、対等な AS 間構造のネットワークを構成するのに使用することができます。

BGP バージョン 1 は [RFC 1105] に、バージョン 2 は [RFC 1163] に、バージョン 3 は [RFC 1267] に、バージョン 4 は [RFC 1771] で説明されています。バージョン 3 の MIB は [RFC 1269] で説明されています。[RFC 1164] と [RFC 1268] と [RFC 1772] の三つはバージョン 2 と 3 と 4 のインタネットでの利用法を説明しています。BGP バージョン 3 のプロトコル解析と実験は [RFC 1265] と [RFC 1266] で説明されています。[RFC 1397] では BGP バージョン 2 と 3 でのデフォルトの経路情報のアナウンスについて説明しています。そして最後に [RFC 1403] で BGP と OSPF との間の経路情報の交換について説明しています。

4.3.4 その他の経路制御プロトコル

router discovery

router discovery protocol はホストにパケットを送ることが可能なルータを知らせ、静的に設定したデフォルトのルータを補うために使用されます。これはホスト側で動作させることを期待していて、これにより経路制御プロトコルのパケットを傍受する必要が無くなります。router discovery protocol は [RFC 1256] で説明されています。

4.4 ルーティングポリシ

ルーティングポリシは経路情報の交換やプロトコルの相互作用によりルータや外部との関係を定義する一連の規則をあらわします。要約すると、ルーティングポリシを選択することによりユーザは以下のことを定義できます。

- システムが peer から得る経路情報
- システムが peer に伝搬する経路情報
- **GateD** がサポートしているプロトコル間での経路情報の交換

ルーティングポリシは GateD の設定ファイルの非常に重要な箇所で、特に同じシステム上で (**IGP** や **EGP** や静的経路、ネットワークインタフェースのネットワークへの経路等) いくつかのプロトコルを使用する場合に無視してはいけません。ポリシ定義は経路情報を import したり、export したりする前に適用する一連のフィルタを定義することと等価なため、ルーティングポリシはしばしば経路情報のフィルタリングともいわれます。

4.4.1 経路情報の import

import statement では import する経路情報を定義します。経路情報を import する範囲には、経路情報を受け取ったネットワークインタフェースや、経路情報を送ってきた peer のアドレスや peer AS 等のようなパラメータも含まれます。section 4.26 では、受け取った経路情報のフィルタを定義するための **import** statement の詳細を説明します。

4.4.2 経路情報の export

export statement ではすべてのプロトコルにより外部に export される経路情報を定義します。経路情報を export する範囲は *import* statement と同じです。

export ポリシはシステム上で動作している異なるプロトコル間での経路情報の相互の交換も定義します。デフォルトでは、プロトコルでの経路情報の再配布は同じプロトコルから経路情報についてのみしか行いません。例えば、RIP では RIP から得た経路情報のみアナウンスし、BGP では他の BGP peer から得た経路情報のみをアナウンスします。export statement ではプロトコル間で経路の交換ができるようになっていて、AS 内プロトコルが再配布可能な AS 外部の経路情報を使用することができ、同じようにその逆もまた可能です。export statement はどのプロトコル経由からも (デフォルトルートのような) 静的経路情報をアナウンスするためにも非常に有用です。

section 4.27 では **export** statement の詳細を説明します。

4.4.3 経路情報のフィルタリング

すべてのルーティングポリシでは経路情報のフィルタリングを、受け取ったりアナウンスする経路情報を明示的に定義するために利用します。経路情報のフィルタリングは経路情報がマッチするための単にアドレス (プリフィックス) とマスクの経路情報のリストです。経路情報のフィルタリングは **martians** statement でも使用されます。

4.5 trace statement

4.5.1 trace statement

trace statement ではトレースオプションを制御します。GateD のトレースオプションはいろいろなレベルで設定することができます。トレースオプションにはファイルの設定、制御オプション、グローバルオプションとプロトコル固有のオプションがあります。トレースオプションはあらためて指定しない限りその statement 内部に引き継がれます。これは例えば BGP peer でのトレースオプションは BGP group でのトレースオプションを引き継ぎ、BGP group でのオプションは GateD のグローバルトレースオプションを引き継ぐということです。各レベルでトレースの設定をすると上位レベルから引き継いだオプションは無効になります。

4.5.2 トレースオプション記述法

traceoptions [”*trace_file*” [**replace**] [**size** *size* [**k** | **m**] **files** *files*]]
[*control_options*] *trace_options* [**except** *trace_options*] ;
traceoptions none;

trace_file

トレース情報を書き込むファイル名を指定します。このファイルがスラッシュ (/) で始まっていない場合は GateD が起動されたディレクトリ以下のファイル名となります。

replace

すでに存在しているファイルに上書きする形でトレースを開始します。デフォルトではすでに存在しているファイルに追加されます。

size *size* [**k** | **m**] **files** *files*

トレースファイルの大きさを指定した大きさ (最小で 10K バイト) に制限します。トレースファイルの大きさが指定した大きさになると、`file.0` という名前にリネームされ、次は `file.1` にリネームされ、次は `file.2` にリネームされ、最大のファイル数 (最小で 2) までリネームされていきます。

control_options

トレースの表示を制御するオプションを指定します。有効なオプションは以下の通りです。

nostamp すべてのトレース表示行にタイムスタンプが付かないように指定します。

trace_options

トレースオプションについては次の section で説明します。

except *trace_options*

広範囲にトレースが行えるトレースオプションを指定した場合に、あるオプションだけを無効にするために使用します。

none

指定したプロトコルや peer についてのトレースは行わないようにします。

4.5.3 グローバルトレースオプション

グローバルオプション (あるいは先に説明した *trace_options*) には 2 種類あります。一つは全体的な動作に使用可能なもので、もう一つは各プロトコル固有なものです。

全体的にのみ有効なオプション

全体的にのみ有効なトレースフラグは以下の通りです。

parse 文字解析と構文解析をトレースします。おもに GateD 開発用のデバッグに用いられます。

adv 経路情報制御用の領域の割り当てや解放をトレースします。おもに GateD 開発用のデバッグに用いられます。

symbols 起動時にカーネルからシンボルを読む時のトレースを行うために使われます。このレベルでのトレースを有効にするためには、コマンドラインで `-t` オプションを指定します。これは、設定ファイルを解析する前にカーネルからシンボルを読み込むためです。

iflist ネットワークインタフェースの情報の読み込みのトレースを行うために使用します。最初のインタフェースの情報の読み込みは設定ファイルを解析する前に行われますが、コマンドラインで `-t` オプションを指定すると、この場合のトレースも行います。

プロトコル固有なオプション

プロトコル固有なオプションは以下の通りです。

all 以下のオプションを全て有効にします。

general `normal` と `route` の両方を有効にします。

state プロトコルの状態遷移をトレースします。

normal プロトコルの通常の動作をトレースします。プロトコルの異常な動作は常にトレースされます。

policy プロトコルが行う設定と import や export した経路情報に対してユーザが指定した設定をトレースします。

task 指定したネットワークインタフェースやプロトコルや peer とのデータの交換をトレースします。

timer 指定したプロトコルや peer とのデータの交換で使用するタイマのトレースをします。

route 指定したプロトコルや peer とのデータの交換で使用される経路情報のトレースをします。

注意

上記のすべてのオプションがすべてのプロトコルに適用できるわけではありません。ある場合においてはオプションの使用は意味のないこともあり (例えば、RIP には状態遷移は無いので)、またある場合では指定したトレースオプションが (RIP での `policy` オプションのように) 実装されていない場合もあります。

注意

現状ではコマンドラインからパケットのトレースをとることはできません。これはパケットトレースを行うグローバルオプションは非常に多くの出力を発生することが予想されるためです。

プロトコル statement でグローバルオプションを引き継ぐ場合は、(**parse**、adv やパケットトレースが無効になってしまうのと同じ様に) トレースレベルは無効になります。

グローバルトレース statement は直ちに有効となり、特に設定ファイルの解析に意味のあるような parse オプションはそうなっています。設定ファイルで各プロトコルが引き継ぐトレースフラグは特にオプションで設定しない限りもともとはグローバルなオプションを実際に解析して引き継いだものです。したがって、設定ファイルを読み込んだ後、明示的に指定していないトレースオプションは実際には設定ファイルを最後まで読んだ所でのグローバルオプションを引き継いだものとなります。

4.5.4 パケットトレース

パケットのトレースには非常に柔軟性があります。それぞれのプロトコルに対して一つ以上のパケットトレースオプションがあります。すべてのプロトコルに対して **packets** というキーワードが使用可能であることにより、プロトコルが送受信する全パケットをトレースすることが可能になります。ほとんどのプロトコルでパケットの種類によってトレースを制限するために他にオプションがあります。このトレースオプションは以下のオプションでさらに制御することができます。

detail **detail** は **send** や **recv** より前に指定されなければいけません。通常のパケットは一行あるいは二行の簡潔な形でトレースが出力されます。**detail** が指定されているとパケットの内容をより詳しく伝えるためにより多くの行数を用いた形で出力されます。

send, recv これらのオプションで送信あるいは受信したパケットにトレースを制限します。これらのオプションが無ければ送受信したパケットの両方をトレースします。

注意

detail が指定されている場合は、**send** や **recv** の前になければいけません。プロトコルでいくつかの異なるタイプのパケットトレースが可能な場合は各オプションはそれぞれのタイプに適用することができます。しかし、トレース指定の範囲内ではトレースオプションは一つにまとめられることに注意しなければならず、したがって **detail** は全パケットの全トレースをとることになります。

4.6 directive statement

directive statements は GateD 設定ファイルの解析時に読み込みファイルとそれがどこにあるかを示すために用います。directive statements は設定ファイル解析時に直ちに作用します。他の statements はセミコロン (;) で終了しますが、directive statements は改行で終了するようになっています。二種類の directive statements があり、以下にそれを示します。

%directory *directory*

読み込みファイルが置かれているディレクトリを指定します。これが使用されると GateD はフルパス指定しないすなわち ”/” で始まらない読み込みファイルはこのディレクトリ以下を検索します。この statement は実際にはカレントディレクトリを変えるわけではなく、単に読み込みファイルのプリフィックスを指定しているだけです。

%include *filename*

> 読み込みファイルを指定します。読み込みファイルの内容は gated.conf 中で **%include** の指定があった時点で gated.conf に取り込まれます。指定したファイル名がフルパスで指定されていない、すなわち "/" で始まっていない場合は、**%directory** で定義したディレクトリからの相対だと判断します。**%include** では設定ファイルを取り込む前に指定したファイルをすべて解析します。**%include** は最大 10 個までネストできます。ネストする最大値は `parse.h` 中の **FI_MAX** の定義を変えることにより変更することができます。

複雑な環境下では、大きな設定ファイルを小さく分かりやすく分けることにより使い易くなります。しかし GateD の大きな利点の一つとしてはいくつかの経路制御プロトコルを一つのファイルにまとめることができることでもあります。不必要に小さなファイルに分けると設定を複雑にするだけです。

4.7 options statement

options statement によってグローバルなオプションの指定が可能になります。**options** を使用する場合は gated.conf のどの設定 statement よりも前になければいけません。

options statement の使用法は以下の通りです。

options [**nosend**]
[**noresolv**]
[**gendefault** [**preference** *preference*] [**gateway** *gateway*]]
[**syslog** [**upto**] *log_level*]
[**mark** *time*]
;

options は一つ以上の以下のオプションから成ります。

gendefault [**preference** *preference*] [**gateway** *gateway*]

> **gendefault** が指定されると、BGP あるは EGP の neighbor が立ち上がっている場合に **default** というデフォルト経路を作成します。これは BGP あるいは EGP の group 指定毎に **nogendefault** オプションで無効にすることもできます。デフォルトでは、この経路の preference は 20 です。この経路は通常カーネルのフォワーディングテーブルに設定されることは無く、他のプロトコルにアナウンスできるようにするためにだけ存在しています。gateway が指定されていると、次 hop を gateway としてカーネルのフォワーディングテーブルに設定されます。
>
> **gendefault** オプションを使用するよりももっと一般的なオプションを使用した方が良いことに留意して下さい。これは、**gendefault** オプションは将来のリリースでは無くなるかもしれないからです。**generate** statement に関しては section 4.28 を参照して下さい。

nosend

> パケットの送信を行いません。このオプションにより、実際には経路制御プロトコルに加わること無く GateD を実際のネットワークで動作させることができます。GateD のログでパケットのトレースを調べることにより GateD が適切に機能していることを確認することができます。これは RIP や HELLO やあるいは SMUX SNMP インタフェースを使用している場合に有用です。このオプションは今のところ BGP には適用できませんし、EGP や OSPF 場合に有用となることはほとんどありません。

noresolv

デフォルトでは GateD は *gethostbyname()* や *getnetbyname()* のライブラリを使用してホスト名を IP アドレスに変換しようとします。このライブラリは通常はローカルホストや変換テーブルの代わりに Domain Name System (DNS) を使用します。ところが、DNS 要求を送るのに十分な経路情報無いと、GateD は起動時にデッドロックに陥ってしまします。このオプションはこの状況を避けるために使用され、そのためこのオプションを指定した場合は設定ファイルでのホスト名指定はエラーとなります。

syslog [**upto**] *log_level*

setlogmask() がサポートされているシステムで syslog 経由で GateD がログを出す場合にそのレベルを制御します。使用可能なログのレベルや用語は `setlogmask(3)` に定義されています。デフォルトでは、**info** レベルまでです。

mark *time*

GateD が指定した間隔でトレースファイルにメッセージを出力するように指定します。これにより、GateD が動作し続けているかどうかを判断する方法の一つとして使うこともできます。

4.8 interface statement

4.8.1 interfaces の使用法

```
 interfaces {
           options
                      [ strictinterfaces ]
                      [ scaninterval time ]
                      ;
           interface interface_list
                      [ preference preference ]
                      [ down preference preference ]
                      [ passive ]
                      [ simplex ]
                      [ reject ]
                      [ blackhole ]
                      ;
           define address
                      [ broadcast address ] | [ pointtopoint address ]
                      [ netmask mask ]
                      [ multicast ]
                      ;
} ;
```

ネットワークインタフェースはシステムとネットワークを接続しています。ネットワークインタフェースはインタフェース名、IP アドレスあるいは (unnumbered point-to-point ネットワークでなければ) ドメイン名で指定することができます。設定ファイル中の記述では複数のレベルで、ワイルドカードやネットワークインタフェースのタイプ名を用いることによりネットワークインタフェースの指定が可能でまた、アドレスでの指定や、アドレスの削除が可能です。いくつかの UNIX システムでは一つのネットワークインタフェースに複数のアドレスが使用可能なため、ネットワークインタフェース名を使用時には注意が必要です。interface_list は複数のワイルドカードの名前 (番号の無い名前) を含むネットワークインタフェース名のリストと、複数のネットワークインタフェースやアドレスのリストあるいはすべてのネットワークインタフェースを示す **all** というトークンから構成されます。

options

interfaces に関するいくつかのグローバルオプションを有効にします。以下にそれらを示します。

strictinterfaces　GateD 起動時に設定ファイルに *define* 宣言文に無いネットワークインタフェースを参照しようとすると致命的エラーとなるように指定します。このオプションが無い場合にはワーニングメッセージを出しますが、GateD はそのまま動作します。

scaninterval *time*　GateD がネットワークインタフェースの状態が変化しているかどうかを調べる間隔を指定します。デフォルトではほとんどのシステムでは 15 秒おきで、インタフェースの状態変化をルーティングソケット経由で知らせるシステム (例えば 4.4BSD) の場合は 60 秒おきとなります。GateD は *SIGUSR2* を受けてもネットワークインタフェースの状態を調べに行くことにも留意して下さい。

interface *interface list*

オプションを設定するネットワークインタフェースを指定します。ネットワークインタフェースのリストは **all**、ネットワークインタフェース名 (ネットワークインタフェース名についての注意を参照してください) のリスト、ドメイン名あるいは IP アドレスから成ります。この宣言文で使用可能なオプションは以下の通りです。

preference *preference*　ネットワークインタフェースが動作状態で正しく動作しているような場合のそのネットワークインタフェースに対する経路情報の preference を指定します。デフォルトの preference は *0* です。

down preference *preference*　GateD がネットワークインタフェースが正しく動作していないと認識していてカーネルが動作状態に無いことを示していない場合のそのネットワークインタフェースに対する経路情報の preference を指定します。デフォルトの preference は *120* です。

passive　経路情報を受信していないためにネットワークインタフェースが正しく機能していないと GateD が認識している場合に、そのネットワークインタフェースに対する経路情報の preference を変えないように指定します。GateD はネットワークインタフェースに対する経路情報が経路制御プロトコルに加わっている場合にのみこのチェックを行います。

simplex　ネットワークインタフェースが自分の送信したブロードキャストパケットを受信することができない場合に指定します。IFF_SIMPLEX というフラグが立っている場合はそのネットワークインタフェースは simplex であると認識されますが、そうでない場合ば設定ファイルで指定する必要があります。simplex ネットワークインタフェースでは自分自身が送信したパケットはソフトウェア的にループバックされてきたものだと見なされ、ネットワークインタフェースが正しく機能していることを示すためには用いられません。

reject　ネットワークインタフェースリストのうち、カーネルに *reject* 経路を設定する場合に、ネットワークインタフェースのアドレスがローカルアドレスとして使用される場合に指定します。この指定は reject/blackhole という仮想的なネットワークインタフェースを設定するような 4.3BSD Tahoe 版かそれ以降のものがベースとなっているシステムでのみ使用すべきものです。

blackhole　ネットワークインタフェースリストの内、カーネルに *blackhole* 経路を設定する場合に、ネットワークインタフェースのアドレスがローカルアドレスとして使用される場合に指定します。この指定は reject/blackhole という仮想的なネットワークインタフェースを設定するような 4.3BSD Tahoe 版かそれ以降のものがベースとなっているシステムでのみ使用すべきものです。

define *address*

GateD が起動時に存在していない可能性のあるネットワークインタフェースに **strictinterfaces** を指定していると設定ファイル中で参照できるように指定します。使用可能なオプションは以下の通りです。

broadcast *address*　ネットワークインタフェースが (Ethernet や Token Ring のように) ブロードキャストタイプのネットワークインタフェースであることを指定するとともにブロードキャストアドレスを指定します。

pointopoint *address* ネットワークインタフェースが (SLIP や PPP のように) point-to-point タイプのネットワークインタフェースであることを指定するとともに自側のアドレスを指定します。*define* statement の最初の *address* には相手側のネットワークインタフェースを指定し、この *pointopoint* オプションの後の *address* は自側のネットワークインタフェースを指定します。

ブロードキャストタイプあるいは point-to-point タイプで定義されないネットワークインタフェースは X.25 ネットワークのように非ブロードキャストマルチアクセスタイプ (NBMA) のネットワークと見なされます。

netmask *mask* このネットワークインタフェースで使用されるサブネットマスクを指定します。この指定は point-to-point タイプのインタフェースでは無視されます。

multicast このネットワークインタフェースでマルチキャストが使用可能な場合に指定します。

4.8.2 ネットワークインタフェースリスト

ネットワークインタフェースリストはネットワークインタフェースやネットワークインタフェース群を表します。ネットワークインタフェースは四つの形式で表されます。一般的な形式からより具体的な形式という順番で以下に示します。

all

使用可能なすべてのネットワークインタフェースを意味します。

ネットワークインタフェース名のワイルドカード

同じタイプのすべてのネットワークインタフェースを表します。UNIX 上でのネットワークインタフェース名は *ie* のようなデバイスドライバ名と 0 とか *5* とか *22* のようなユニット番号から構成されます。名前の照合はアルファベット文字のみで行い、アルファベット文字部が同じすべてのネットワークインタフェースに一致するネットワークインタフェースを表します。

例えば、Sun での *ie* はすべての Interlan Ethernet のネットワークインタフェースを表し、*le* はすべての Lance Ethernet のネットワークインタフェースを表します。しかし、*ie* は *iel0* には当てはまりません。

ネットワークインタフェース名

特定のネットワークインタフェースで、通常は一つのネットワークインタフェースを表します。これはアルファベット文字部とそれに続く数字の部分で表されます。これにより一つのネットワークインタフェースに一致させます。しかし、ネットワークインタフェースに複数のプロトコル (例えば *IP*) アドレスを持ち得るシステムも多いということに留意して下さい。

例えば、*ef1* は *ef1* というネットワークインタフェース名に一致しますが、*ef10* というネットワークインタフェース名には一致しません。

ネットワークインタフェースアドレス

これにより、特定のネットワークインタフェースを指定することができます。これは、プロトコルアドレス (例えば *10.0.0.51*) 指定やホスト名 (例えば *nic.ddn.mil*) 指定で表すことができます。ホスト名指定はそのホスト名から一つのアドレスが得られる場合にのみ有効となることに留意して下さい。ホスト名指定はあまり推められません。

設定ファイル中に複数のオプションと一緒にいくつかネットワークインタフェースリストがある場合は、このオプションは GateD の起動時に取り込まれ、各ネットワークインタフェースのオプションリストを作成します。複数のネットワークインタフェースリストに同じオプションが指定されている場合は、最も具体的なインタフェース名で表されたネットワークインタフェースのオプションが使用されます。

例えば、*le0* と *le1* と *du0* の三つのネットワークインタフェースを持つシステムを想定します。

```
rip yes {
    interface all noripin noripout ;
    interface le ripin ;
    interface le1 ripout ;
} ;
```

RIP パケットは *le0* と *le1* というネットワークインタフェースからのみ受け付けられ、*du0* からは受け付けられません。RIP パケットは *le1* というネットワークインタフェースからのみ送信されます。

4.8.3 ネットワークインタフェースアドレスと経路情報

4.3BSD やそれ以後のネットワーク実装では四種類のネットワークインタフェースのタイプが使用できます。一つのネットワークインタフェースで複数のプロトコルアドレスが使用可能なものもありますが、これらはほとんど *4.3BSD Reno* あるいはそれ以後のものがベースとなっています。

loopback このネットワークインタフェースは **127.0.0.1** というアドレスでなければいけません。このネットワークインタフェースに送信されるパケットは送信元に送り返されます。このネットワークインタフェースは *reject* 経路や *blackhole* 経路のような他の機能を持つネットワークインタフェースとしても使用されます。このネットワークインタフェースではネットマスクが設定されていますが無視されます。このネットワークインタフェースに OSPF や BGP の *router id* と同じアドレスをさらに割り当てると便利です。これにより、ネットワークインタフェースがダウンしていても *router id* に基づいて動作しているシステムへのルーティングが可能となります。

broadcast このネットワークインタフェースは *Ethernet* や *Token Ring* や *FDDI* のような物理レベルでのブロードキャストが可能なマルチアクセスネットワークインタフェースです。このネットワークインタフェースにはサブネットマスクとブロードキャストアドレスがあります。*broadcast* するネットワークへのネットワークインタフェースへの経路情報はまさにサブネットへの経路情報となります。

point-to-point このネットワークインタフェースは他のホストへのトンネルとしてのネットワークインタフェースで、通常は *serial* リンク上で使用されます。このネットワークインタフェースには自側アドレスと相手側アドレスがあります。一つの *point-to-point* のネットワークインタフェースに複数のアドレスを指定することは可能ですが、そのようにする有効な理由は無いように思われます。

point-to-point のネットワークインタフェースの相手側アドレスはシステムの持つすべての *point-to-point* のネットワークインタフェースの相手側のアドレスの中で唯一でなければいけません。*point-to-point* のネットワークインタフェースの自側アドレスは一つの non-*point-to-point* のネットワークインタフェースと他の *point-to-point* のネットワークインタフェースで同じであっても構いません。これは技術的にはアドレスの無い links のために使用する *router id* とするためです。サブネットを新たに使用しない方法としてこの手法が使用されます。

point-to-point のネットワークインタフェースでサブネットマスクが指定されると、RIP バージョン 1 と HELLO のみが相手側のルータにどのサブネットを伝えるのかを決めるためにネットマスクを使用します。

non-broadcast multi-access or nbma このネットワークインタフェースはマルチアクセスではあるけれどもブロードキャストタイプでは無いタイプのネットワークインタフェースです。この例としては、*frame relay* や *X.25* があります。このタイプのネットワークインタフェースには自側アドレスとサブネットマスクを持ちます。

GateD は設定済みの使用可能な各ネットワークインタフェースへの経路情報があることを保証します。通常これはネットワークインタフェースを設定する *ifconfig* コマンドによって行われますが、GateD はこれを保証するために同じことを行います。

point-to-point のネットワークインタフェースでは GateD は特別な経路をいくつか設定します。*point-to-point* のネットワークインタフェースの自側アドレスがどの non-*point-to-point* のネットワークインタフェースのアドレスとも同じでない場合、GateD は *loopback* ネットワークインタフェースを示す自側アドレスへの経

路の preference を 110 として設定します。これによりこのホストが送信した自側アドレス宛のパケットがローカルに処理されることを保証します。ところが、OSPF ではローカルネットワークインタフェースに対するパケットが相手側のルータから *point-to-point* link 経由で戻ってくるようになります。これは link の動作確認のために使用されます。OSPF は preference 10 で経路を設定するため、preference 110 で設定される経路もこの preference で設定されます。

複数の *point-to-point* のネットワークインタフェースの自側アドレスが non-*point-to-point* のネットワークインタフェースのアドレスと同じ場合、GateD は自側アドレスへの経路情報を preference 0 で設定しますが、カーネルのフォワーディングテーブルには設定しません。これはこのシステムがルータではなく、ホストとして機能している場合に OSPF のようなプロトコルが *point-to-point* ネットワークインタフェース経由でルーティングしないようにするためです。

ネットワークインタフェースの状態が変化した場合は GateD はすべてのプロトコルにそれを知らせ、それにより各プロトコルは適切な処理を行います。GateD は *UP* 状態でないネットワークインタフェースは存在していないネットワークインタフェースだと見なします。これは正しい処理でないかもしれませんが、現状ではこのように動作します。

GateD は自側や相手側やブロードキャストアドレスやサブネットマスクの値が無効な場合、そのネットワークインタフェースは無視します。無効な値には 0 という値も含みます。GateD は自側アドレスと相手側アドレスが同じ *point-to-point* のネットワークインタフェースも無視します。この場合 GateD はループバックテストモード状態の一種だと見なします。

4.9 definition statement

definition statement は GateD のすべてあるいは少なくとも二つ以上のプロトコルに関連する一般的な definition statement です。これには三つの definition statement があり、それぞれ **autonomoussystem** と **routerid** と **martians** です。**autonomoussystem** や **routerid** や **martians** を使用する場合は gated.conf 中の他の設定 statement よりも前になければいけません。

4.9.1 AS 番号の設定

autonomoussystem *autonomous_system* [**loops** *number*] ;

このルータが属する自律システムの自律システム (AS) 番号を指定します。このオプションは BGP や EGP を使用する場合に必要です。AS 番号は Network Information Center (NIC) が割り当てています。

loops は BGP のように AS 経路をサポートしているプロトコルのためだけのものです。これは AS 経路中にこの AS が現れる回数を指定し、デフォルトでは 1 です。

4.9.2 ルータ ID の設定

routerid *host* ;

BGP や OSPF で使用するルータの識別子を指定します。デフォルトでは GateD が最初に見つけたネットワークインタフェースのアドレスとなります。この識別子は point-to-point のネットワークインタフェースの自側のアドレスよりも non-point-to-point のネットワークインタフェースのアドレスの方が良いですし、ループバックネットワークインタフェースがループバックアドレス (127.0.0.1) でないならその方が最も良いです。

4.9.3 martians の設定

martians {
    host *host* [**allow**] ;
    network [**allow**] ;
    network **mask** *mask* [**allow**] ;
    network **masklen** *number* [**allow**] ;

default [**allow**] ;

} ;

無視する経路情報を *martians* アドレスとして指定します。設定を誤ったシステムが明らかに不正な経路情報を送ってくることがあります。この経路情報は martians と呼ばれ、経路制御のソフトウェアが受け付けないようにします。この statement により martians アドレスの指定ができます。指定する範囲は、section 4.25.1 を参照して下さい。また、設定した範囲の一部を明示的に無効にすることができる **allow** オプションも指定することができます。

4.9.4 definition statement の例

```
autonomoussystem 249 ;

martians {
    0.0.0.26
};
```

上記の statement では次の機能を果たします。

- autonomoussystem statement では GateD に EGP や BGP で使用する AS 番号として 249 を指定しています。
- martians statement では 0.0.0.26 への経路情報は受け付けないように指定しています。

4.10 RIP statement

4.10.1 Routing Information Protocol

Routing Information Protocol (RIP) はもっとも広く使用されている interior gateway protocols です。RIP は distance-vector あるいは Bellman-Ford の経路制御プロトコルをローカルネットワークで実現したものです。RIP ではルータをアクティブとパッシブ (静的) に位置付けます。アクティブルータは他のホストやルータに経路情報 (到達可能情報) をアナウンスします。パッシブルータではアナウンスを待ち、アナウンスされた内容をもとに経路情報を更新しますが、アナウンスはしません。通常、ルータではアクティブモードで RIP を動作させ、ホストではパッシブモードで RIP を動作させます。

アクティブモードで RIP が動作しているルータでは一定間隔ごとに経路情報を更新したものをブロードキャストします。更新した経路情報には destination のネットワークアドレスとそのネットワークへの距離の整数値との対で構成される値が含まれています。RIP では destination への距離を示すのにホップカウントメトリックを使用します。RIP のメトリックでは、直接接続しているネットワークをメトリック 1 でアナウンスします。他のゲートウェイを一つ通過して到達可能となるネットワークはホップ数は 2 というようになります。このように、ある送信元から到達点までの経路でのホップ数またはホップカウントはその経路でデータグラムが通過するゲートウェイの数となります。しかし、最短経路を求めるのにホップカウントを使用することが常に最適なものとなるわけではありません。例えば、三つの Ethernet を通過してホップカウントが 3 となるような経路の方が二つの低速シリアルラインを通過してホップカウントが 2 となるような経路のよりも実際は早くなります。技術的にこれを補うために、多くのルータでは低速回線については意図的に大きなホップカウントでアナウンスします。

ほとんどの UNIX システムで配布されているように、RIP は経路制御デーモンである *routed* (ルートディと発音します) により動作します。RIP による経路制御では受信した RIP の更新情報をもとに動的に経路情報を作成します。起動時には経路情報を取得するために REQUEST パケットを送信し、その要求への応答を待ちます。RIP を供給するように設定されたシステムがその要求を受けると、そのシステムが持つ経路情報をもとに RESPONSE パケットで応答します。RESPONSE パケットには destination のネットワークアドレスとそこへのメトリックが含まれています。

RIP の RESPONSE パケットを受けると経路制御デーモンは情報を採取し、新たな経路やすでにデータベース中にある destination へのよりよい (より低いメトリック) 経路情報を追加して経路情報のデータベースを再構築します。RIP では destination ネットワークへのゲートウェイへのホップ数が 15 を越えていたり、経路情報が削除されている場合にはデータベースからその経路情報を削除します。また、一定の期間ゲートウェイから更新情報を受けないとそのゲートウェイ経由の経路情報はすべて削除します。一般的には、経路情報の更新は 30 秒おきに行われます。多くの実装では、180 秒間ゲートウェイからの更新情報が受信されないと経路情報のデータベースから、そのゲートウェイから受け取った経路情報をすべて削除します。この 180 秒は特定の経路情報の削除についてもあてはまります。

RIP のバージョン 2 (RIP II の方が一般的ですが) では RIP にさらに機能を追加しています。このうちいくつかは RIP I と互換性がありますが、そうでないものもあります。間違って RIP I の経路情報として解釈してしまうようなことを避けるために、RIP II ではマルチキャストパケットの場合にのみ非互換の機能を使用することができます。IP マルチキャストができないネットワークインタフェースでは、RIP I 互換のパケットを使用し、紛らわしい情報を含まないようになっています。

RIP II の重要な機能拡張のいくつかは以下のとおりです。

次ホップ 経路情報の更新をアナウンスするルータ以外に次ホップのルータをアナウンスすることができます。これは、RIP を動作させていない沈黙したルータへの静的な経路情報をアナウンスする場合に有効です。これによりこのルータを通過するパケットが同じネットワークを二度通過することを避けています。

RIP I のルータは RIP II パケットの次ホップの情報は無視します。これはネットワークに同じパケットが二度通過することになりますが、RIP I では必ず発生することです。このようにこの情報は RIP I と互換性のある RIP II パケットに提供されています。

ネットマスク RIP I ではあるネットワークのすべてのサブネットマスクは同じネットマスクであることを仮定しています。この仮定を受信したすべての経路情報のネットマスクを求める場合について適用します。この仮定では RIP パケットでは異なるネットマスクを持つサブネットを持つことができません。RIP II ではパケット中に各ネットワークのネットマスクを明示的に指定することができるようになっています。

RIP I では RIP II パケットのネットマスクを無視するようになっているのにかかわらず、RIP II でのネットマスクの算出は非常に不正確なものとなり得ます。このため、RIP I 互換の RIP II パケットでは間違って解釈してしまうようなネットワークを含んではいけません。このようなネットワークの経路情報はマルチキャストパケットを使用する RIP II パケットでのみ提供されなければいけません。

認証 RIP II のパケットでは受信した経路情報の正当性を確認するために使用することができる二つのうちいずれかの認証を行うことができます。認証は RIP I 互換の RIP II パケットでも使用することができますが、RIP I のルータでは無視されることに注意してください。

一つ目の方法はパケット中に含まれる 16 文字までの認証キーによるパスワードです。これが予期したものに一致しない場合、そのパケットは無視されます。この方法は RIP パケットを監視することにより認証キーを知ることができるため、セキュリティは非常に低いものとなります。

二つ目の方法は未だ実験段階で、将来のリリースでは非互換となるかもしれません。この方法では、RIP パケットと 16 文字までの認証キーの暗号化チェックサムを作成するために MD5 アルゴリズムを使用します。送信したパケット自身には認証キーは含まれませんが、代わりにダイジェストと呼ばれる暗号化チェックサムを含みます。パケットを受信したルータは正しい認証キーを使用して計算を行い、ダイジェストが一致しなければそのパケットを無視します。さらに、古いパケットへの応答を避けるために連続番号を保持しています。この方法では、ルータが送信した正しい認証キーを持つ経路情報には非常に強い保証があります。

RIP I では受け取ったネットワークやホストへの経路へのネットマスクをそのパケットを受け取ったネットワークインタフェースのネットマスクより得ます。受け取ったネットワークやホストが自分と同じネットワーク番号であり、そのパケットを受け取ったネットワークインタフェースがサブネット化されている (ネットマスクが通常ののネットワークで決まるネットマスクよりも長い) 場合、その destination ネットワークにそのサブネットマスクが適用されます。マスクのホスト部分がセットされている場合はホストとし、そうでない場合はサブネットワークとみなします。

point-to-point のネットワークインタフェースではネットマスクは相手アドレスに対して適用されます。このようなネットワークインタフェースでは相手ネットワークと同じネットワークであったりネットマスクがすべて 1 である場合、ネットワークインタフェースのネットマスクは無視されます。

これまでのリリースとは違い、ゼロのサブネットマスク (ネットワークインタフェースのネットワークには一致するがネットマスクは長い) は無視されます。無視されないようにする場合は経路情報のフィルタリングを使用して実現することができます。

4.10.2 RIP Statement

```
rip yes | no | on | off [ {
          broadcast ;
          nobroadcast ;
          nocheckzero ;
          preference preference ;
          defaultmetric metric ;
          query authentication [ none | [ [ simple | md5 ] password ] ] ;
          interface interface_list
                    [ noripin ] | [ ripin ]
                    [ noripout ] | [ ripout ]
                    [ metricin metric ]
                    [ metricout metric ]
                    [ version 1 ] | [ version 2 [ multicast | broadcast ] ]
                    [ [ secondary ] authentication [ none | [ [ simple | md5 ] password ] ] ;
          trustedgateways gateway_list ;
          sourcegateways gateway_list ;
          traceoptions trace_options ;
} ] ;
```

rip statement で RIP を使用したりしないようにします。**rip** statement が指定されていない場合はデフォルトでは **rip on ;** となります。RIP を使用する場合、ネットワークインタフェースが一つしかないと **nobroadcast** とし、ネットワークインタフェースが二つ以上では **broadcast** としています。

オプションは以下の通りです。

broadcast

ネットワークインタフェースの数に関係なく RIP パケットをブロードキャストするように指定します。これは静的経路情報や他のプロトコルから得た経路情報を RIP でアナウンスする場合に有効です。ただし、ネットワークインタフェースが一つしかない場合に **broadcast** を使用するとネットワーク上を同じパケットが二回通過するような原因となる場合があります。

nobroadcast

ネットワークインタフェースが二つ以上あっても RIP パケットをブロードキャストしないように指定します。**sourcegateways** がある場合には、経路情報をそのゲートウェイに直接送ります。

nocheckzero

受信した RIP I のパケットの reserved フィールドがゼロであるかどうかの確認はしないように指定します。通常 RIP では reserved フィールドがゼロでないパケットは受け付けません。

preference *preference*

RIP によって得た経路情報の preference を指定します。デフォルトの preference は 100 です。この preference は import statement であらためて指定することができます。

defaultmetric *metric*

他のプロトコルから得た経路情報を RIP でアナウンスする場合のメトリックを指定します。指定しない場合は、デフォルトで 16 (到達不能) になります。この値の選択するには他のプロトコルから RIP へ経路情報を export する場合のメトリックを明示的に指定する必要があります。このメトリックは export statement であらためて指定することができます。

query authentication [**none** | (**simple** | **md5**) *password*] **;**

ルータ以外からの RIP の要求パケットに必要な認証を指定します。デフォルトでは **none** です。

interface *interface_list*

個々のネットワークインタフェースで RIP を送信する場合の属性を制御します。*interface_list* の説明については interfaces statement を参照してください。

同じネットワークに複数のネットワークインタフェースを設定している場合、RIP の更新パケットは最初に RIP パケットの出力をするように設定された方から送信されます。この制限は UNIX のカーネルの動作によるものです。将来のリリースではこの制限がなくなることが期待されます。

使用可能なオプションは以下の通りです。

noripin 指定したネットワークインタフェースで受信した RIP パケットを無視するように指定します。デフォルトではループバックインタフェース以外のすべてのネットワークインタフェースからのパケットを受け取ります。

ripin これがデフォルトです。このオプションはネットワークインタフェースの表記でワイルドカードを指定して **noripin** を使用する場合に必要です。

noripout 指定したネットワークインタフェースからは RIP パケットを送信しないように指定します。デフォルトでは **broadcast** を指定しているとすべてのブロードキャストや非ブロードキャストのネットワークインタフェースで RIP を送信します。point-to-point のネットワークインタフェースで RIP を送信する場合は逐一設定する必要があります。

ripout これがデフォルトです。このオプションは point-to-point のネットワークインタフェースで RIP を送信したり、ネットワークインタフェースの表記でワイルドカードを指定して **noripout** を使用する場合に必要です。

metricin *metric* 受信した経路情報をルーティングテーブルに取り込む前に付け加える RIP メトリックを指定します。デフォルトではカーネルの持つネットワークインタフェースのメトリック (デフォルトの RIP のホップカウント) に 1 を足したものです。この値が指定されると、絶対値として使用されカーネルの持つメトリックは足されません。このオプションはルータ上で指定したネットワークインタフェースから得た RIP の経路情報の方を他のネットワークインタフェースから得た RIP の経路情報の方よりも優先させる場合に使用されます。

metricout *metric* 指定したネットワークインタフェースから送信される経路情報に足される RIP メトリックを指定します。デフォルトではゼロです。このオプションは別のルータの RIP の経路情報を優先させる場合に使用されます。

version 1 指定したネットワークインタフェースから送信される RIP パケットはバージョン 1 のパケットとなるように指定します。これがデフォルトです。

version 2 指定したネットワークインタフェースから送信される RIP パケットはバージョン 2 のパケットとなるように指定します。このネットワークインタフェースで IP マルチキャストがサポートされている場合、デフォルトでは完全なバージョン 2 のパケットを送信します。そうでなければバージョン 1 互換のバージョン 2 のパケットが送信されます。

multicast このネットワークインタフェースから送信される RIP バージョン 2 のパケットはマルチキャストとなるように指定します。これがデフォルトです。

broadcast IP マルチキャストが使用可能であっても、このネットワークインタフェースから送信される RIP バージョン 1 互換のバージョン 2 のパケットはブロードキャストとなるように指定します。

[**secondary**] **authentication** [**none** | [**simple** | **md5**] *password*] このオプションでは使用する認証を指定します。これは RIP バージョン 2 でのみ適用され、RIP I では無視されます。デフォルトの認証は **none** です。パスワードを指定した場合、認証タイプは **simple** になります。パスワードは 16 文字までの引用符付きの文字列でなければいけません。

secondary が指定されている場合、二つ目の認証を指定することになります。これが無い場合は、一つ目の認証を指定したことになります。デフォルトでは一つ目の認証は **none** で、二つ目の認証はありません。

trustedgateways *gateway_list*

どのゲートウェイからの RIP の更新パケットを受け入れるかを指定します。*gateway_list* はホスト名または IP アドレスのリストです。デフォルトでは同じネットワーク上のすべてのルータの経路情報を受け取るようになっています。しかし、**trustedgateways** を指定するとリストで指定したゲートウェイからの更新のみを受け取るようになります。

sourcegateways *gateway_list*

RIP パケットをマルチキャストやブロードキャストではなく、直接送るルータのリストを指定します。これは指定したゲートウェイに別の経路情報を送る場合に使用することができます。指定したゲートウェイへの更新パケットの送信は **noripout** とは無関係です。

traceoptions *trace_options*

RIP で使用されるトレースオプションを指定します (trace statements や以下の RIP 固有のトレースオプションを参照して下さい)。

4.10.3 トレースオプション

policy オプションでは新しい経路情報がアナウンスされたり、メトリックの変更がアナウンスされたり、経路情報を保持したり放棄するとトレースを行います。

パケットトレースオプション (**detail** や **send** や **recv** で変更可能です) は以下の通りです。

packets すべての RIP パケットのトレースを行います。

request *REQUEST* や *POLL* や *POLLENTRY* のような RIP の情報要求パケットのトレースを行います。

response RIP の *RESPONSE* パケットのトレースを行います。このパケットは実際に経路情報を含むようなパケットのタイプです。

other その他のタイプのパケットのトレースを行います。有効なパケットのタイプは *TRACE_ON* と *TRACE_OFF* ですが、処理はされません。

4.11 HELLO statement

4.11.1 HELLO プロトコル

特別に必要でないのでしたら、HELLO を使用しない方がよいです。GateD 4.0 あたりで HELLO を外す予定です。

HELLO は source と destination の間でパケットが到達するまでにかかる時間長をベースにしたルーティングメトリックを使用する AS 内部プロトコルです。HELLO パケットでは受信側で destination への最も短い遅延経路を求めるためのタイムスタンプ情報を伝達します。「もっとも良い」経路は最も短い遅延時間となる

ホップ数	HELLO メトリック
0	0
1	100
2	148
3	219
4	325
5	481
6	713
7	1057
8	1567
9	2322
10	3440
11	5097
12	7552
13	11190
14	16579
15	24564
16	30000

Table 4.2: Hello メトリック

経路です。HELLO で使用される時間の単位はミリセカンドです。HELLO の更新パケットがルータ間を 100 ミリセカンド以下で伝搬する場合、100 のうちの最小値がホップとして使用されます。このように高速のネットワークで構成されていると HELLO は本質的にデフォルトでホップカウントを使用します。HELLO では他のルーティングアルゴリズムと同様に、経路情報を高速に変更することはできません。そうでないと経路情報が不安定になります。経路情報の不安定を避けるために、HELLO の実装ではヒステリシスを使用し、経路情報の変更が続くであろうと決定するまで経路情報を変更することを「ためらい」ます。

デフォルトで HELLO は RIP と同様に、ルーティングテーブルにインストールされる (*metricin*) 時に経路情報に追加されたメトリックに反映するために *ifconfig* で設定されたネットワークインタフェースのカーネルでのメトリックを使用します。カーネルでのネットワークインタフェースのメトリックはホップであらわされるため、HELLO のミリセカンドのメトリックに変換されなければいけません。これをおこなうためには table 4.2 を使用します。

4.11.2 HELLO とネットマスク

HELLO では受信したネットワークやホストのネットマスクをパケットを受信したネットワークインタフェースのネットマスクから得ます。受信したネットワークやホストがサブネット化 (ナチュラルネットワークのネットマスクよりマスクが長い) されたネットワークインタフェースと同じナチュラルネットワークと同じ場合、そのサブネットマスクを使用します。マスクの箇所以外のビットがセットされているとホストとみなし、それ以外はサブネットされているとみなします。

point-to-point のネットワークインタフェースではネットマスクは相手アドレスに対して適用されます。このようなネットワークインタフェースでは相手ネットワークと同じネットワークであったりネットマスクがすべて 1 である場合、ネットワークインタフェースのネットマスクは無視されます。

これまでのリリースとは違い、ゼロのサブネットマスク (ネットワークインタフェースのネットワークには一致するがネットマスクは長い) は無視されます。無視されないようにする場合は経路情報のフィルタリングを使用して実現することができます。

4.11.3 HELLO statement

hello yes | **no** | **on** | **off** [**{**

```
            broadcast ;
            nobroadcast ;
            preference preference ;
            defaultmetric metric ;
            interface interface_list
                        [ nohelloin ] | [ helloin ]
                        [ nohelloout ] | [ helloout ]
                        [ metricin metric ]
                        [ metricout metric ] ;
            trustedgateways gateway_list ;
            sourcegateways gateway_list ;
            traceoptions trace_options ;
} ] ;
```

hello statement ではHELLOを使用したりしないようにします。**hello** statement が指定されていないと、デフォルトでは **hello off;** となります。HELLOを使用する場合、一つしかネットワークインタフェースがなければ **nobroadcast** とし、ネットワークインタフェースが二つ以上では **broadcast** とします。

broadcast

ネットワークインタフェースの数に関係なくHELLOパケットをブロードキャストするように指定します。これは静的経路情報や他のプロトコルから得た経路情報をHELLOでアナウンスする場合に有効です。ただし、ネットワークインタフェースが一つしかない場合に **broadcast** を使用するとネットワーク上を同じパケットが二回通過するような原因となる場合があります。

nobroadcast

ネットワークインタフェースが二つ以上あってもHELLOパケットをブロードキャストしないように指定します。**sourcegateways** がある場合には、経路情報をそのゲートウェイに直接送ります。

preference *preference*

HELLOによって得た経路情報のpreferenceを指定します。デフォルトのpreferenceは90です。このpreferenceはimport statementであらためて指定することができます。

defaultmetric *metric* 他のプロトコルから得た経路情報をHELLOでアナウンスする場合のメトリックを指定します。指定しない場合は、デフォルトで30000 (到達不能) になります。この値の選択するには他のプロトコルからHELLOへ経路情報をexportする場合のメトリックを明示的に指定する必要があります。このメトリックはexport statementであらためて指定することができます。

interface *interface_list*

個々のネットワークインタフェースでHELLOを送信する場合の属性を制御します。*interface_list* の説明についてはsection 4.8.2のネットワークインタフェースリストを参照してください。

同じネットワークに複数のネットワークインタフェースを設定している場合、HELLOの更新パケットは最初にhellooutが設定された方から送信されます。この制限はUNIXのカーネルの動作によるものです。将来のリリースではこの制限がなくなることが期待されます。

使用可能なオプションは以下の通りです。

nohelloin 指定したネットワークインタフェースで受信したHELLOパケットを無視するように指定します。デフォルトではループバックインタフェース以外のすべてのネットワークインタフェースからのパケットを受け取ります。

helloin これがデフォルトです。このオプションはネットワークインタフェースの表記でワイルドカードを指定して **nohelloin** を使用する場合に必要です。

nohelloout 指定したネットワークインタフェースからは HELLO パケットを送信しないように指定します。デフォルトでは **broadcast** を指定しているとすべてのブロードキャストや非ブロードキャストのネットワークインタフェースで HELLO を送信します。point-to-point のネットワークインタフェースで HELLO を送信する場合は逐一設定する必要があります。

helloout これがデフォルトです。このオプションは point-to-point のネットワークインタフェースで HELLO を送信したり、ネットワークインタフェースの表記でワイルドカードを指定して **nohelloout** を使用する場合に必要です。

metricin *metric* 受信した経路情報をルーティングテーブルに取り込む前に付け加える HELLO メトリックを指定します。デフォルトではカーネルの持つネットワークインタフェースのメトリック (デフォルトの HELLO のホップカウント) に 1 を足したものです。この値が指定されると、絶対値として使用されカーネルの持つメトリックは足されません。このオプションはルータ上で指定したネットワークインタフェースから得た HELLO の経路情報の方を他のネットワークインタフェースから得た HELLO の経路情報の方よりも優先させる場合に使用されます。

metricout *metric* 指定したネットワークインタフェースから送信される経路情報に足される HELLO メトリックを指定します。デフォルトではゼロです。このオプションは別のルータの HELLO の経路情報を優先させる場合に使用されます。

trustedgateways *gateway_list*

どのゲートウェイからの HELLO の更新パケットを受け入れるかを指定します。*gateway_list* はホスト名または IP アドレスのリストです。デフォルトでは同じネットワーク上のすべてのルータの経路情報を受け取るようになっています。しかし、**trustedgateways** を指定するとリストで指定したゲートウェイからの更新のみを受け取るようになります。

sourcegateways *gateway_list*

HELLO パケットをマルチキャストやブロードキャストではなく、直接送るルータのリストを指定します。これは指定したゲートウェイに別の経路情報を送る場合に使用することができます。指定したゲートウェイへの更新パケットの送信は **nohelloout** とは無関係です。

traceoptions *trace_options*

HELLO のトレースオプションを指定します (以下のトレースオプションと HELLO 固有のトレースオプションを参照してください)。

デフォルトの preference は 90 です。デフォルトのメトリックは 30000 です。

4.11.4 トレースオプション

policy パラメータでは新たな経路情報がアナウンスされたり、アナウンスされたメトリックの変更や経路情報が holddown 状態になったり終了すると常にログにとられます。

パケットのトレースオプション (`detail`、`send` や `recv` で変更することができます) を指定します。

packets すべての HELLO パケットのトレースを行います。

4.12 OSPF statement

4.12.1 OSPF プロトコル

Open Shortest Path First (OSPF) は *Shortest Path First*(SPF) または *link-state* プロトコルです。OSPF は一つの自律システム内のルータ間で経路情報の交換を行う interior gateway protocol です。OSPF では最小の

コストを最良の経路として選択します。OSPF は非常に多くのルータが存在する複雑なネットワークに適していて、同時に一つ以上のネットワークインタフェース経由で destination にパケットを送信できるというような等価コストを持つ複数経路を提供します。link-state プロトコルでは AS 内の全体トポロジを表すデータベースを各ルータごとに保持していて、そのトポロジはすべてのルータからの link state 通知を集めたものから作成しています。OSPF を使用するルータは自分の持つ状態 (例えば、ルータで使用可能なネットワークインタフェースや到達可能な隣接ルータ) を AS 内に flooding により配布します。各 multiaccess ネットワークでは少なくとも二つルータが存在していて、それぞれ指定ルータとバックアップ指定ルータがあります。指定ルータには multiaccess ネットワークの link state 通知を flooding させたりその他の責務があります。指定ルータにより、mutiaccess ネットワーク上に必要な隣接関係の数を抑えるようになっています。

OSPF ではネットワークをエリアにまとめることができます。エリア間での経路情報はまとめられ、経路情報トラフィックを大きく抑制することができます。OSPF では 4 つのタイプの経路情報を使用し、それぞれ優先度の順に、エリア内経路、エリア間経路、AS 外部経路タイプ 1、AS 外部経路タイプ 2 です。エリア内経路はエリア内への経路情報、エリア間経路は他の OSPF エリアへの経路情報、そして Autonomous System External (ASE) は AS 外部への経路情報となっています。OSPF に import されるタイプ 1 の AS 外部経路情報では AS 内部の経路制御プロトコルからのものであるとしていて、そのメトリックは OSPF メトリックと直接比較できるようになっています。経路情報を選択する場合、OSPF は AS 境界ルータへの外部メトリックに AS 内部のコストを足します。タイプ 2 の ASE では AS 外部メトリックを使用していて、そのメトリックは OSPF のメトリックとは比較できません。この場合、経路選択時には AS 境界ルータへの内部コストのみが使用されます。

トポロジデータベースより、各ルータは自分自身を根として最短経路の木構造を作成します。この最短経路の木構造は AS 内の各 destination への経路情報を提供します。AS 外部から得た経路情報は木構造上は葉として現われます。AS 外部からの経路情報と AS 内部からの経路情報では link-state 通知フォーマットが異なるため、それらが不明確になることはありません。AS 外部からの経路情報 (例えば EGP や BGP から得た経路情報) は AS 内にそのまま渡され、OSPF の持つ AS 内部の経路情報とは区別されています。AS 外部の各経路情報は通知ルータにより AS 外部の経路情報と位置付けをすることも可能で、AS 境界ルータ間での追加情報を渡すこともできます。

OSPF では *type of service* (TOS) の経路制御を選択することもでき、管理者がある destination への各 type of service 毎 (例えば low delay や high mthroughput) の複数経路を設定することができます。OSPF を動かしているルータでは destination への最適な経路を選択するために destination アドレスと type of service を使用します。

OSPF のエリア内とエリア外の経路情報は GateD の経路情報に preference 10 で常に import されます。OSPF ルータがエリアの OSPF に完全に属していない場合はプロトコル違反となり、preference をあらためて指定することはできません。他の経路情報に明示的に低い preference を指定することはできますが、賢明ではありません。

ネットワークインタフェースのマルチキャスト機能も使用され、それにより link-state メッセージをアナウンスすることができます。

OSPF エリアはバックボーンエリアに接続していて、バックボーンエリアの識別子は 0.0.0.0 です。すべてのエリアは論理的に連続していなければならず、バックボーンも例外ではありません。柔軟性を最大限に持たせるために、OSPF では仮想リンクの設定によりバックボーンエリアを実際の物理的な接続にかかわらず連続に見せることができます。

エリア内のすべてのルータはエリアパラメータが同じでなければいけません。エリア毎に link-state アルゴリズムが動いています。このため設定パラメータのほとんどはエリア毎に決まっています。エリアに属するすべてのルータはそのエリアの設定にしたがわなければいけません。設定ミスをすると隣接したルータとの adjacency を確立できなくなり、経路情報が伝わらないかループするかもしれません。

4.12.2 認証

OSPF のすべての情報の交換は認証されます。認証により、インタネットとそのユーザを保護するために経路情報を信頼できるルータからのみ import することを保証します。いろいろな認証方式を使用することができますが、エリア内では一つの方式となっていなければいけません。これにより、あるエリアではそれ以外のエリアに比べてより厳しい認証を使用することができます。

この場合、二つの認証方式を使用することができます。一つ目は8文字までの認証キーを使用するもので標準化されています。二つ目はまだ実験段階で、MD5アルゴリズムで16文字までの認証キーを使用します。

単なるパスワードだけではほとんど保護することはできません。これはネットワーク上のパケットをトレースして認証キーを得るのが容易であることが多いためです。実験段階のMD5アルゴリズムでははるかに保護することができます。これはパケット中に認証キーが含まれないためです。

現状のOSPFの仕様では、認証方式はエリア毎に指定し、ネットワークインタフェース毎にパスワードを指定することができるようになっています。これは認証方式とネットワークインタフェース毎にキーを指定できるように拡張されています。さらにネットワークインタフェース毎に*primary*と*secondary*の認証を指定することができます。送信パケットはprimaryの認証を使用しますが、受信パケットはprimaryまたはsecondaryの認証方式とキーに一致させることができます。

4.12.3 OSPF statement

```
ospf yes | no | on | off [ {
        defaults {
                preference preference ;
                cost cost ;
                tag [ as ] tag ;
                type 1 | 2 ;
        } ;
        exportlimit routes ;
        exportinterval time ;
        traceoptions trace_options ;
        monitorauthkey authkey ;
        monitorauth none | ( [ simple | md5 ] authkey ) ;
        backbone | ( area area ) {
                authtype 0 | 1 | none | simple ;
                stub [ cost cost ] ;
                networks {
                        network [ restrict ] ;
                        network mask mask [ restrict ] ;
                        network masklen number [ restrict ] ;
                        host host [ restrict ] ;
                } ;
                stubhosts {
                        host cost cost ;
                } ;
                interface interface_list; [ cost cost ] {
                        interface_parameters
                } ;
                interface interface_list nonbroadcast [ cost cost ] {
                        pollinterval time ;
                        routers {
                                gateway [ eligible ] ;
                        } ;
                        interface_parameters
                } ;
                Backbone only:
                virtuallink neighborid router_id transitarea area {
                        interface_parameters
                } ;
```

```
                } ;
} ] ;
```

上で示した *interface_parameters* でのパラメータは以下の通りです。これはネットワークインタフェースの属性を指定し、**interface** statement で記述することができます。

enable | **disable ;**
retransmitinterval *time* **;**
transitdelay *time* **;**
priority *priority* **;**
hellointerval *time* **;**
routerdeadinterval *time* **;**
[**secondary**] **auth** (**none** | **simple** | **md5** | *auth_key*) **;**
authkey *auth_key* **;**

defaults

以下のパラメータは OSPF の ASE 経路情報を GateD が持つ経路情報に import したり、GateD が持つ経路情報を OSPF の ASE へ export する場合のデフォルトを指定します。

preference *preference* preference は GateD が持つ経路情報内で OSPF の経路情報が他のプロトコルからの経路情報に対してどれぐらいの優先度となるのかを指定します。デフォルトでは 150 です。

cost *cost* cost は GateD の経路情報のうち OSPF 以外から得た経路情報を OSPF ASE に export する場合に使用します。デフォルトは 1 です。これは export statement であらためて指定することができます。

tag [**as**] *tag* OSPF の ASE の経路情報には OSPF では使用しない 32 ビットのタグフィールドがありますが、経路情報のフィルタリングをするために export statement で使用することができます。OSPF が egp と相互に経路情報の交換を行う場合 tag フィールドを使用することができます。そのような場合は as を使用し、タグは 12 ビットまでに制限されます。指定していない場合は、タグはゼロになります。

type *1* | *2* GateD が持つ経路情報を OSPF に export する場合、デフォルトでタイプ 1 の ASE となります。このデフォルトはここで明示的に変更するか export statement であらためて指定することができます。

ASE の flooding の割合

OSPF の性質上 ASE の flooding を行う割合は制限されなければいけません。以下の二つのパラメータによりこの割合を調整することができます。

exportinterval *time* ASE link-state 通知が行われ、OSPF に flooding される間隔を指定します。デフォルトでは一秒に一回です。

exportlimit *routes* 一度にどれだけの ASE を作成し flooding させるかを指定します。デフォルトでは 100 です。

traceoptions *trace_options*

OSPF で使用されるトレースオプションを指定します (trace statements や以下の OSPF 固有のトレースオプションを参照して下さい)。

monitorauthkey *authkey*

OSPF の状態は *ospf_monitor* を使用して要求することができます。このコマンドは非標準の OSPF パケットを送信し、OSPF からの応答をテキスト形式で作成するようになっています。デフォルトではこの要求の認証はされていませんが、認証キーを設定していれば受信した要求が指定した認証キーに一致していないといけません。このパケットによって OSPF の状態を変更させることはできませんが、OSPF への要求動作はシステム資源を使用するかもしれません。

backbone

area *area*

各 OSPF ルータは少なくとも一つの OSPF エリアに属するように設定されなければいけません。複数のエリアを設定する場合、少なくともそのうちの一つは *backbone* でなければいけません。backbone は **backbone** を指定した場合のみ使用することができ、*area 0* として指定することはできません。backbone のネットワークインタフェースは *virtuallink* とすることができます。

authtype *0* | *1* | *none* | *simple* OSPF ではエリア毎に認証方式を指定します。エリア内の各ネットワークインタフェースは別の *authenticationkey* を使用することができますが、同じ認証方式を使用しなければいけません。現在有効なものは認証を行わないことを示す **none** (*0*) または単純なパスワード認証を示す **simple** (*1*) です。

stub [**cost** *cost*] *stub* エリアでは ASE の経路情報が通知されません。*cost* が指定されている場合、デフォルトの経路情報をこのエリア内にこのコストでアナウンスします。

networks *networks* はエリアの範囲を表します。指定した範囲内に一致するエリア内の LSA はエリア間の経路情報として他のエリアにアナウンスされることはありません。その代わり、指定した範囲は *summary network* LSA としてアナウンスされます。**restrict** が指定されている場合は、summary network LSA はアナウンスされません。指定した範囲内に一致しないエリア内の LSA も summary network LSA としてアナウンスされます。このオプションはエリア間を伝搬する経路情報量を抑えるように十分考慮されたネットワークを構築する場合に非常に有用です。このリストのエントリはネットワークまたは対になったサブネットワークとマスクのいずれかです。範囲の指定の詳細は section 4.25.1 を参照してください。

stubhosts 指定したコストで到達可能であることをアナウンスしなければいけないような直接接続しているホストを指定します。OSPF を動作させることが望めない point-to-point のネットワークインタフェースはここで指定すべきです。

ループバックインタフェースに別のアドレス (127 のネットワークでない) を割り当てる場合にも有用で、その場合 stub ホストとしてアナウンスされます。このアドレスがルータ ID として使用されているものと同じ場合、ネットワークインタフェースのアドレスの代わりにルータ ID を用いて他の OSPF ルータへの経路制御が可能となります。これは常に到達可能ではないかもしれないネットワークインタフェースのアドレスに向けての経路制御よりも信頼することができます。

interface *interface_list* [**cost** *cost*]

ここでの interface 文ではブロードキャストタイプ (IP マルチキャストがサポートされていることが必要) または *point-to-point* タイプのネットワークインタフェースについての設定を行う場合に使用します。*interface_list* の指定の説明については section 4.8.2 を参照してください。

ネットワークインタフェースにはそれぞれ *cost* があります。cost には destination に到達するためにパケットが通過しなければいけないすべてのネットワークインタフェースの cost が合計されています。デフォルトでは cost は 1 ですが、0 以外の値を指定することができます。

ネットワークインタフェースのパラメータはすべてのタイプのネットワークインタフェースに共通で以下の通りです。

retransmitinterval *time* 指定したネットワークインタフェースでの隣接ルータへの LSA の再送間隔の秒数を指定します。

transitdelay *time* 指定したネットワークインタフェースで link state update を送信するのに必要な秒数を指定します。transitdelay は送信時間と伝搬遅延時間を考慮し、0 以上でなければいけません。

priority *priority* 指定したネットワークで指定ルータになる優先度を指定し、0 から 255 までの値です。ネットワーク上に二つのルータが存在している場合、どちらも指定ルータになろうとしますが、高い優先度を持つ方が指定ルータになります。priority が 0 となっているルータは指定ルータになる資格はありません。

hellointerval *time* 指定したネットワークインタフェースから Hello パケットを送信する間隔を秒数で指定します。

routerdeadinterval *time* ルータからの Hello パケットを受信しなくなってからそのルータがダウンしているとするまでの秒数を指定します。

[**secondary**] **auth** (**none** | **simple** | **md5** | *auth_key*) 指定したネットワークインタフェースで使用する認証方式を指定します。

authkey *auth_key* OSPF の認証を作成し、OSPF ヘッダでの認証フィールドを確認するための認証キーを指定します。認証キーはネットワークインタフェース毎に指定することができます。ピリオドで区切られた 1 つから 8 つまでの 10 進数か、*0x* で始まる 1 から 8 バイトまでの 16 進数か、ダブルクォートで囲まれた 1 から 8 つの文字列のいずれかを指定します。

point-to-point のネットワークインタフェースにはさらに以下のパラメータがあります。

nomulticast デフォルトでは OSPF パケットは IP マルチキャストを使用して point-to-point のネットワークインタフェースの隣接ルータに送られます。しかし、UNIX の IP マルチキャストの実装のうちいくつかにはバグがあり、このようなネットワークインタフェースでの IP マルチキャストができません。GateD ではこの状態を見つけ、point-to-point の隣接ルータにはユニキャストの OSPF パケットを送信するようになっています。

相手のルータが IP マルチキャストサポートしていないため IP マルチキャストを使用したくない場合、ユニキャストの OSPF パケットを使用させるように **nomulticast** パラメータを指定することができます。このオプションは GateD が先に述べたバグを見つけた場合に警告を発しないようにするためにも使用することができます。

interface *interface_list* **nonbroadcast** [**cost** *cost*]

ここでの interface 文では非ブロードキャストのマルチアクセス (NBMA) メディアに接続されている非ブロードキャストタイプのネットワークインタフェースについての設定を行う場合に使用します。OSPF ではブロードキャストタイプのメディアは IP マルチキャストをサポートしている必要がありますが、Ethernet のようにブロードキャストすることが可能ではあるようなメディアであっても IP マルチキャストをサポートしていないものは非ブロードキャストタイプのネットワークインタフェースとして設定しなければいけません。

非ブロードキャストタイプのネットワークインタフェースは先に示した普通の interface 文で示しているものをサポートしていますが、さらに以下の二つの非ブロードキャストタイプのネットワークインタフェース固有なオプションもサポートしています。

pollinterval *time* 隣接ルータと adjacency が確立するまでに OSPF パケットを **pollinterval** で指定した間隔で送信します。

routers 非ブロードキャストのネットワークでは OSPF の隣接ルータを見つけるためにブロードキャストアドレスを送信することはもともとできないため、すべての隣接ルータを指定しなければいけません。このリストには一つ以上の隣接ルータと指定ルータになる資格があるかどうかを指定します。

virtuallink neighborid *router_id* **transitarea** *area* virtuallink はバックボーンエリアとの接続を確立したり追加する場合に使用します。**neighborid** には virtuallink の先の *router_id* を指定します。通過する *area* もここで指定しなければいけません。virtuallink では先に示した普通の **interface** 文で示しているオプションも指定することができます。

4.12.4 トレースオプション

以下に示す OSPF 固有のトレースオプションに加えて、OSPF ではネットワークインタフェースと隣接ルータとの状態遷移をトレースする state をサポートしています。

lsabuild LSA の作成

spf Shortest Path First (SPF) の算出

lsatransmit Link State Advertisement (LSA) の送信 (未実装)

lsareceive LSA の受信 (未実装)

state 状態遷移 (未実装)

パケットトレースオプション (**detail** や **send** や **recv** で変更可能です) は以下の通りです。

hello 隣接ルータとの到達可能性を調べるために使用する OSPF の *HELLO* パケットのトレースを行います。

dd OSPF のデータベースを合わせるために使用する OSPF の Database Description パケットをトレースを行います。

request OSPF のデータベースを合わせるために使用する OSPF の Link State Request パケットのトレースを行います。

lsu OSPF のデータベースを合わせるために使用する OSPF の Link State Update パケットのトレースを行います。

ack OSPF のデータベースを合わせるために使用する OSPF の Link State Ack パケットのトレースを行います。

4.12.5 あらたな OSPF の実装

GateD のバージョン 3_6Alpha_2 以上には OSPF のあらたな実装が含まれています。設定の文法は以下以外は同じです。

1. **export limit**、**exportinterval**、**monitorauthkey**、**monitorauth**、**networks** や **stubhosts** statements はすでに使用できません (将来復活するかもしれません)。
2. **simple** の認証は実装されていません。
3. OSPF 固有のトレースオプションで使用できるのは、**state**、**drelection**、**db**、**spf** と **flood** です。パケットのトレースについて使用できるのは **hello**、**dd**、**lsr**、**lsu** と **lsa** です。

4.13 IS-IS statement

4.13.1 IS-IS Domain 間プロトコル

IS-IS の設定の詳細は GateD で配布されている関連するドキュメント (ISIS-config.ps) で説明しています。一般的な設定ではモード (dual や iso)、システムレベル、トレースオプション、circuit やそのメトリックが指定できます。

IS-ISはもともとISO/CLNP(International Organization for Standardization/Connectionless Network Protocol)パケットでのルーティングをおこなうために開発されたlink stateを使用したinterior gateway protocol (igp)です。GateDで配布しているIS-ISではIPパケットも同様にルーティングできます。ISOの用語ではルータは「Intermediate System」(IS)と呼ばれます。IS-ISのドメイン間ルーティングは階層的に構成されていて、大きなドメインは管理によってエリア内のISではレベル1を、エリア間のISではレベル2を使用して小さいエリアに分割することができます。管理下のドメイン間でのルーティングはIDRP(inter-domain routing protocol)を使用してBorder Intermediate Systems(BISs)によっておこなわれます。レベル1のシステムでは同じエリア内へのシステムへは直接ルーティングし、destinationのシステムが別のエリアにある場合は、レベル2のシステムに対してルーティングします。レベル2のシステムではエリア間へルーティングし、destinationのエリアへの経路を保持するようになっています。レベル2のサブドメイン内のシステムはdestinationのエリアか他の経路ドメインに対してルーティングします。インターネットの経路制御プロトコルと同様に、IS-ISでは巨大な経路ドメインをサポートしていてさまざまなタイプのサブネットワークを持つことができます。このようなサブネットワークにはpoint-to-pointのネットワークや、マルチアクセスなネットワークや、X.25でのネットワークのように動的にデータリンクを張るようなネットワークやISO 8802のようなブロードキャストタイプのネットワークを含みます。

IS-ISでのすべてのサブネットワークのタイプは、必要なサブネットワークの集中機能を使用して、コネクションレスなサブネットワークであるかのようにサブネットワークの個々の機能として扱われます。OSPFと同様にIS-ISではSPFアルゴリズムを使用して経路を決定します。経路情報の過剰な流れを制御する部分では各ISをモニタし、バッファのデッドロックが起きないようにしています。GateDの設定文法では起こり得るエラーの発生をできる限り押えるように自動設定が可能です。また、BGP、EGP、RIPやIDRPなどのようにGateDで動作する他のプロトコルと経路情報の交換をするためのポリシを指定することもできます。新しいIS-ISでは複数経路(load-split)でのフォワーディングや外部ネットワークのプリフィックスのすべての取り込みや内部BGPや同様なプロトコルの必要性を除外するための属性をサポートしています。IS-ISではレベル2のISでの静的なルーティングドメイン情報をサポートしています。

到達可能なアドレスプリフィックスはそのプリフィックスが関連するSubnet Point of Attachment (SNPA)経由で到達可能かもしれないプリフィックスに一致するNetwork Service Access Points (NSAPs)を示します。このSNPAが接続しているサブネットワークに動的なデータリンクの確立がサポートされている一般的なサブネットワークがあるような場合には、プリフィックスは必要なサブネットワークアドレス情報やdestinationのNSAPアドレスから得られることができる情報にも関連づけられます。レベル2の経路制御アルゴリズムによるアドレスプリフィックスは、ドメイン内でレベル1の情報が扱われるのと同様に扱われます。

4.13.2 IS-IS statement

```
isis no | dual | ip | iso {
        level 1 | 2 ;
        [ traceoptions isis_traceoptions ; ]
        [ systemid 6_digit_hexstring ; ]
        [ area hexstring ; ]
        [ set isis_parm value ; ... ]
        circuit string
                metric [ level 1 | 2 ] 1..63;
                ...
                priority [ level 1 | 2 ] 0..127;
                ...
                ;
        ...
} ;
```

このstatementによりGateDでIS-ISが使用できます。デフォルトではIS-ISは使用しません。**dual**オプションにより、IS-ISがISOとIPアドレスの双方を扱うように指定します。**isis** statementは、ISの初期

指定と指定した circuit と管理するネットワークの設定する定義文により構成されます。定義文の順番は関係無く、次のものから構成されます。

level

GateD がレベル 1(intra-area) の IS かレベル 2 (inter-area) の IS で動作するかを指定します。デフォルトではレベル 1 です。

traceoptions

以下で説明するトレースオプションを参照してください。

systemid *6_digit_hexstring*

(ネットワークインタフェースのアドレスとそのネットマスクから決まる) 自動的に設定されるシステム ID を再設定します。システム識別子が指定されていないと、circuit の NSAP の最初のシステム ID が使用されます。システム ID が設定されると、すべての IS-IS を無効にして再び設定しない限り、変更することはできません。

area *hexstring*

IS-IS のエリアのアドレスは IS-IS が動作している実際の circuits をもとに、自動的に設定されます。この定義文で指定されるアドレスは circuits から自動的に設定されたアドレスとともに保持されます。この定義文は基本的にはテストのために使用されます。

circuit *string*

各 **circuit** 定義文により、システムで管理しようとする circuits を指定します。circuits は通常 UNIX のネットワークインタフェース名の文字列に関連づけられていますが、疑似的なデバイス名も指定することができます。string が ”simN” という形式 (N は整数) になっている場合、circuit はネットワークの疑似的な疑餌針 (?) に管理された疑似的な circuit とされます。circuit の属性は circuit 定義文で指定できるオプションのリストからなります。

metric

各 circuit についてのレベル 1 とレベル 2 のメトリックを指定します。デフォルトのメトリックのタイプのみ指定できます。IS-IS のメトリックは 1 から 63 の間でなければいけません。circuit にメトリックが指定されていないと、デフォルトの 63 が使用されます。

priority

指定ルータの選出に関する優先度を指定します。高い値を指定すると、より指定ルータになる優先度が上がります。レベルのデフォルトはレベル 1 です。priority が指定されていないと、0 から 127 の任意の値になります。

レベル 2 の IS では、レベル 1 の circuit はメトリック 10 に、レベル 2 のメトリックは 20 に設定し、circuit 定義文にそのメトリックを追加します。

デフォルトのレベルは 1 で、メトリックは 63 です。レベル 1 の IS-IS のデフォルトの priority は 15 で、レベル 2 の IS-IS では 18 です。

4.13.3 トレースオプション

IS-IS の設定は、ほとんど完全に GateD の配布で含まれている関連ドキュメント (ISIS-config.ps) で説明しています。トレースオプションは次のうちの一つあるいは複数指定することができます。

all

iih

lanadj

p2padj

lspdb

lspcontent

lspinput

flooding

buildlsp

csnp

psnp

route

update

paths

spf

events

4.14 IGMP statement

4.14.1 Internet Group Management Protocol (IGMP)

IGMP はもともと、マルチアクセス可能なネットワーク上のホストがそのネットワーク上のルータにホストが属するグループ情報を知らせるために設計されました。これは、IGMP の Host Membership Report をマルチキャストすることによりおこなわれます。マルチキャストルータはこのメッセージを聞き、他のマルチキャストルータとグループ情報を交換することができます。これにより、マルチキャストのデータグラムを配送するための tree 構造を形成することができます。オリジナルバージョンの IGMP は [RFC 1112] で定義されています。

IGMP の拡張のための開発がおこなわれてきていてその後、Xerox PARC から IP マルチキャストがリリースされています。この拡張は IGMP バージョン 2 として知られていて、より速い pruning のための明示的な leave message やマルチキャストでの traceroute メッセージが含まれています。GateD では IGMP のバージョン 1 や 2 の双方のメッセージを解釈します。しかし、マルチキャストの traceroute の機能は未だ実装されていません。もちろん「ごく近い将来」実装されるでしょう。

4.14.2 IGMP statement

igmp yes | **no** | **on** | **off** [**{**
queryinterval *sec* **;**
timeoutinterval *sec* **;**
interface *interface_list* **enable** | **disable**;
traceoptions *trace_options* **;**
}] ;

igmp statement により IGMP を有効にしたり無効にすることができます。**igmp** statement が指定されていない場合、デフォルトでは **igmp off;** になります。有効にしている場合、デフォルトでブロードキャストとマルチキャストの双方が可能なすべてのネットワークインタフェースで使用します。このようなネットワークインタフェースは IFF_BROADCAST や IFF_MULTICAST のネットワークインタフェースのフラグで識別されます。IP マルチキャストルーティングプロトコルを有効にする前には IGMP を有効にしておかなければいけません。

オプションは次のとおりです。

queryinterval *sec*

ルータがその LAN 上で指定 querier となっている場合の IGMP Host Membership Query を送信する間隔を指定します。指定していなければ、デフォルトの 125 秒が使用されます。

timeoutinterval *sec*

Host Membership Report を受信していない場合に、ローカルグループデータベースにそのグループを保持する期間を指定します。指定していなければ、デフォルトの 270 秒が使用されます。これは、2 × (queryinterval + 10) で計算されます。10 秒は、Host Membership Report を送信する際の変動の割合からきています。queryinterval の値が指定されていて、timeoutinterval が指定されていなければ、GateD はこの計算式を使用して適切な timeoutinterval を求めます。しかし、timeoutinterval が指定されていれば、計算値は使用されず、その値が使用されます。

interface *interface_list*

指定したネットワークインタフェースあるいはネットワークインタフェース群で IGMP を有効にしたり無効にしたりすることができます。現状の GateD の IGMP の実装では、同じ物理ネットワークインタフェースに複数のアドレスが指定されていてもそれを見つけることはできません。したがって、論理的なネットワークに IGMP のメッセージを送信します。これは将来のリリースで修正されるべきことです。

使用可能なパラメータは次のとおりです。

disable 指定したネットワークインタフェースから受信した IGMP のパケットは無視するように指定します。デフォルトではすべてのネットワークインタフェース上からのパケットを受信します。

enable これがデフォルトです。この指定は、ワイルドカードでネットワークインタフェースを指定した場合に `disable` が必要な場合に指定することができます。

traceoptions *trace_options*

IGMP のトレースオプションを指定します (以下のトレースオプションと IGMP 固有のトレースオプションを参照してください)。

4.14.3 トレースオプション

パケットのトレースオプション (`detail`、`send` や `recv` で変更することができます) を指定します。

packets すべての IGMP パケットのトレースを行います。

query IGMP Host Membership Query のパケットのトレースを行います。

report IGMP Host Membership Report と New Host Membership Report のパケットのトレースを行います。

leave IGMP Host Leave messages のパケットのトレースを行います。

mtrace IGMP マルチキャストの traceroute(`request` と `response`) パケットと Cisco multicast trace message のパケットのトレースを行います。

4.15 DVMRP statement

4.15.1 Distance Vector Multicast Routing Protocol (DVMRP)

DVMRP は IP マルチキャストルーティングプロトコルのオリジナルとなっています。これは、マルチキャストできないルータ経由でも同様にマルチキャスト可能な (Ethernet のような)LAN 上で動作するように設計されていました。このような場合、IP マルチキャストパケットはユニキャストパケットとしてルータを「トンネリング」して送られています。これはパケットを複製して、パフォーマンス上効果が得られますが、ルータベンダが本来の IP マルチキャストルーティングをサポートするまでの間、IP マルチキャストルーティングをインタネット上でおこなうための暫定的な解決策です。

DVMRP の初期のバージョンは RFC 1075 で述べられていますが、現在使用されているプロトコルを説明しているものではありません。GateD での DVMRP は Xerox PARC リリース 2.x や 3.x と互換があります。上流の neighbor がサポートしていれば、pruning や grafting は機能します。また、Xerox PARC リリース 3.5 で導入された capability flag も理解します。IP マルチキャストの traceroute や DVMRP の MIB はまだサポートされていません。これらの機能はもうしばらく待ってください。

GateD での DVMRP は最初のリリースです。PIM のような別の IP マルチキャストルーティングプロトコルが動作している他のネットワークから source となっているネットワークを DVMRP のユニキャストルーティングプロトコルに取り込む方法は現状ではありません。この機能はテストされている方法が安全であることがわかり次第リリースされるでしょう。

4.15.2 DVMRP statement

```
dvmrp yes | no | on | off [ {
        interface interface_list [ {
                enable | disable ;
                metric metric ;
                threshold threshold ;
                ratelimit rate ;
                advertise network metric metric ;
                advertise network mask mask metric metric ;
                advertise network masklen number metric metric ;
        } ] ;
        tunnel host lcladdr local_address [ {
                enable | disable ;
                metric metric ;
                threshold threshold ;
                ratelimit rate ;
                advertise network metric metric ;
```

```
                    advertise network mask mask metric metric ;
                    advertise network masklen number metric metric ;
          } ] ;
          traceoptions trace_options ;
} ] ;
```

dvmrp statement では DVMRP を有効にしたり無効にしたりします。**dvmrp** statement が指定されていないと、デフォルトでは **dvmrp off;** になります。指定されていると、デフォルトでマルチキャスト可能なすべてのネットワークインタフェースで有効になります。このようなネットワークインタフェースはネットワークインタフェースフラグの IFF_MULTICAST で識別されます。

同一ネットワークインタフェース上では同時に一つのみマルチキャストルーティングプロトコルを設定することができます。

オプションは次のとおりです。

interface *interface_list*

指定したネットワークインタフェースあるいはネットワークインタフェース群で DVMRP を有効にしたり無効にしたりすることができます。現状の GateD の DVMRP の実装では、同じ物理ネットワークインタフェースに複数のアドレスが指定されていてもそれを見つけることはできません。したがって、論理的なネットワークに DVMRP のメッセージを送信します。これは将来のリリースで修正されるべきことです。

使用可能なパラメータは次のとおりです。

disable 指定したネットワークインタフェースから受信した DVMRP のパケットは無視するように指定します。デフォルトではマルチキャストが可能なすべてのネットワークインタフェース上からの DVMRP パケットを受信します。

enable これがデフォルトです。この指定は、ワイルドカードでネットワークインタフェースを指定した場合に `disable` が必要な場合に指定することができます。

metric ネットワークインタフェースやトンネルでのメトリックを指定するために使用します。このメトリックはこのネットワークインタフェースから得たすべての経路情報に加算されます。

threshold ネットワークインタフェースやトンネルでのスレショルドを指定するために使用します。パケットの TTL がこのスレッショルドを越えなければこのネットワークインタフェースからはフォワードされません。

ratelimit ネットワークインタフェース上でのマルチキャストパケットのトラフィックを制限するために使用します。これは Xerox PARC の IP マルチキャストカーネル (リリース 3.3 以上) でのオプションです。

advertise `advertise` により、指定したネットワークをこのネットワークインタフェースからのみアナウンスします。これは多少危険ですが、非 DVMRP エリアが DVMRP エリアに join する場合に必要です。ネットワーク番号とネットマスクあるいはネットワーク番号とプリフィックス長により指定することができます。ネットマスクが指定されていない場合、クラスのデフォルトのマスクが使用されます。

tunnel *host* **lcladdr** *local_address*

tunnel 定義文は **interface** 定義文と同じオプションを使用します。

traceoptions *trace_options*

DVMRP のトレースオプションを指定します (以下のトレースオプションと DVMRP 固有のトレースオプションを参照してください)。

4.15.3 トレースオプション

パケットのトレースオプション (`detail`、`send` や `recv` で変更することができます) を指定します。

packets すべての DVMRP パケットのトレースを行います。

probe DVMRP Router Probe パケットのトレースを行います。

report DVMRP Route Report パケットのトレースを行います。

mapper DVMRP neighbor と neighbor 2 パケットのトレースを行います。

prune DVMRP Prune パケットのトレースを行います。

graft DVMRP Graft と Graft Ack パケットのトレースを行います。

4.16 PIM statement

4.16.1 Protocol Independent Multicast (PIM)

PIM は既存のマルチキャストルーティングプロトコルである DVMRP と CBT を利用するために設計されました。これは、Core Based Trees での sparse グループの働きを利用する一方で、dense グループエリアでの Reverse Path Multicasting を使用してマルチキャストパケットの flooding させるプロトコルからなります。したがって、プロトコルには *dense* と *sparse* の二つのモードがあります。GateD では現状、dense モードのみ実装されています。PIM の概要は [PIM Arch] で述べられています。詳細なプロトコル仕様は [PIM Spec] で述べられています。こらのドキュメントは執筆中のものとされています。

4.16.2 PIM statement

```
pim yes | no | on | off [ {
        queryinterval sec ;
        routertimeout sec ;
        prunetimeout sec ;
        inactivitytimeout sec ;
        graftacktimeout sec ;
        preference {
                direct preference ;
                kernel preference ;
                ospf preference ;
                ospfase preference ;
                rip preference ;
                hello preference ;
                static preference ;
                bgp preference ;
                egp preference ;
                idrp preference ;
                isis preference ;
                slsp preference ;
        } ;
        interface interface_list [ [ enable ] | [ disable ] ] [ mode [ dense | sparse ] ] ;
        traceoptions trace_options ;
} ] ;
```

pim statementではPIMを有効にしたり無効にしたりします。**pim** statementが指定されていないと、デフォルトでは**pim off;**になります。指定されていると、デフォルトでマルチキャスト可能なすべてのネットワークインタフェースで有効になります。このようなネットワークインタフェースはネットワークインタフェースフラグのIFF_MULTICASTで識別されます。

オプションは次のとおりです。

queryinterval *sec*

マルチキャスト可能なLAN上でPIM Router Queryを送信する間隔です。PIM Router QueryはPIMルータがマルチアクセスネットワーク上の他のPIMルータを見つけるためにAll Routers Multicast Groupに対して送信されます。指定していないと、デフォルトで30秒になります。

routertimeout *sec*

PIM Router Queryを受信していないためにPIMルータの存在を記憶している間隔です。この時間を過ぎてもPIM Router Queryを受信しないとそのルータはローカルなデータベースから削除されます。この値はデフォルトではqueryintervalの3倍かデフォルトのqueryintervalが使用去れていれば90秒になります。

prunetimeout *sec*

ルータから送信されるpruneを受信してからそのpruneが有効であるとする期間です。デフォルトでは180秒です。

inactivitytimeout *sec*

フォワーディングキャッシュエントリが無効になってから、すなわちsourceとgroupの対がフォワードされなくなってから有効になるまでの期間です。デフォルトでは180秒です。

graftacktimeout *sec*

PIMルータでgraftの確認のパケットを受信されなくてgraftが再送されるまでの期間です。デフォルトでは60秒です。

preference

preference宣言文ではマルチアクセスLAN上でマルチキャストパケットをフォワードするルータを決めるためのプロトコルを比較するために使用する優先度の値を変更することができます。この値は、一つ以上のルータがマルチキャストパケットをフォワードでき、他のルータがパケットを複製してフォワードしていることを見つけると、assertメッセージで送信されます。この値はマルチアクセスLAN上のすべのルータで同じでなければいけません。そのため、デフォルトの値は他の実装に合わせられています。デフォルトの値はネットワークインタフェースへのネットワークへは0、静的経路は1、OSPFやOSPFASEは110、ISISは115、RIPは120、SLSPは130、EGPは140、BGPは200、IDRPは210、カーネルに存在しているルーティングテーブルは220、helloは255になっています。

interface *interface_list*

指定したネットワークインタフェースあるいはネットワークインタフェース群でPIMを有効にしたり無効にしたりすることができます。現状のGateDのPIMの実装では、同じ物理ネットワークインタフェースに複数のアドレスが指定されていてもそれを見つけることはできません。したがって、論理的なネットワークにPIMのメッセージを送信します。これは将来のリリースで修正されるべきことです。

使用可能なパラメータは次のとおりです。

disable 指定したネットワークインタフェースから受信したPIMのパケットは無視するように指定します。デフォルトではマルチキャストが可能なすべてのネットワークインタフェース上からのPIMパケットを受信します。

enable これがデフォルトです。この指定は、ワイルドカードでネットワークインタフェースを指定した場合に `disable` が必要な場合に指定することができます。

mode ネットワークインタフェースでの PIM のデフォルトのモードを指定します。これは Router Query メッセージで他の PIM ルータと交換され、ルータで明示的な join を上流側に送る (**sparse**) か、flooding や pruning をおこなう (**dense**) かを決めることができます。

traceoptions *trace_options*

PIM のトレースオプションを指定します (以下のトレースオプションと PIM 固有のトレースオプションを参照してください)。

4.16.3 トレースオプション

パケットのトレースオプション (`detail`、`send` や `recv` で変更することができます) を指定します。

packets すべての PIM パケットのトレースを行います。

query PIM Router Query パケットのトレースを行います。

register PIM Register、Register Stop と RP パケットのトレースを行います。

join PIM Join/Prune パケットのトレースを行います。

assert PIM Assert パケットのトレースを行います。

assert PIM Graft と Graft Ack パケットのトレースを行います。

4.17 EGP statement

4.17.1 Exterior Gateway Protocol

Exterior Gateway Protocol (EGP) は他の AS のゲートウェイと経路情報を交換するために使用される AS 外部経路制御プロトコルです。AS 内部経路制御プロトコルとは異なり、EGP は到達可能情報のみを伝搬し、正確なメトリックは伝搬しません。EGP のデータには 0 から 255 までの *distances* と呼ばれるメトリックが含まれています。GateD は同じ AS から得た経路情報の distances の比較しかしません。

EGP では経路情報を他のルータに送る前に、そのルータと adjacency を確立する必要があります。これは *Hello* (HELLO プロトコルや OSPF の HELLO メッセージとは混同しないようにして下さい) と *I Heard You* (I-H-U) をそのルータと交換することにより行います。EGP でメッセージの交換をするルータを EGP *neighbor* と呼び、HELLO や I-H-U のメッセージの交換のことを *neighbor* の確立と呼びます。neighbor が確立されると、システムは neighbor の持つ経路情報を調べるために *poll* を送信します。この場合、neighbor は経路情報を含む *update* を送り返します。システムが neighbor から poll を受信すると、システムの持つデータを送り返します。システムがデータを受信すると、そのデータをシステムの経路情報のデータベース中に取りこみます。neighbor が三回連続して poll に応答しなかった場合、GateD は neighbor がダウンしたと判断し、システムのデータベースからその neighbor に関する経路情報を削除します。

4.17.2 EGP Statement

egp yes | **no** | **on** | **off**
{
    preference *preference* ;
    defaultmetric *metric* ;
    packetsize *number* ;
    traceoptions *trace_options* ;

```
        group
                [ peeras autonomous_system ]
                [ localas autonomous_system ]
                [ maxup number ]
        {
                neighbor host
                        [ metricout metric ]
                        [ preference preference ]
                        [ preference2 preference ]
                        [ ttl ttl ]
                        [ nogendefault ]
                        [ importdefault ]
                        [ exportdefault ]
                        [ gateway gateway ]
                        [ lcladdr local_address ]
                        [ sourcenet network ]
                        [ minhello | p1 time ]
                        [ minpoll | p2 time ]
                        [ traceoptions trace_options ]
                        ;
        } ;
} ] ;
```

preference *preference*

EGP から得た経路情報の preference を指定します。デフォルトの preference は 200 です。この preference は **group**、**neighbor** または **import** statments あらためて設定することができます。

defaultmetric *metric*

EGP 経由でアナウンスする経路情報で使用されるメトリックを指定します。指定していない場合は、システムによっては到達不可能を示すことにもなる 255 というデフォルトの値になります。neighbor に export する場合にメトリックを明示的に指定するにはこの値を指定する必要があります。このメトリックは neighbor、 group あるいは export statement であらためて設定することができます。

packetsize *maxpacketsize*

neighbor から受信する EGP パケットの最大値を指定します。受信した EGP パケットの大きさがこの値を越えた場合、このパケットは不完全となり、廃棄されます。この場合、このパケットの大きさは GateD に通知され、この大きさのパケットが受信できるように最大値は拡大されます。このオプションを設定することにより、最初の EGP パケットが捨てられることはなくなります。指定していない場合は、デフォルトで 8192 バイトになります。すべてのパケットの大きさはシステムのページサイズの倍数に切り上げられます。

traceoptions *trace_options*

EGP のトレースオプションを指定します。デフォルトでは、グローバルトレースオプションを引き継ぎます。この値は group や neighbor で指定するとその部分ではそちらの方が使用されます (trace statement や以下の EGP 固有のトレースオプションを参照して下さい)。

group

neighbor は **group** のメンバとして指定しなければいけません。group は通常、neighbor がすべて一つの AS 内にある場合に使用します。group 文で指定したオプションは明示的に neighbor 文であらためて指定しない限り、その group に属する neighbor に適用されます。**group** 文の数は **neighbor** 文の数を意味する場合があります。

neighbor 文内のオプションは **group** 文で指定することができ、これは group 文内でのデフォルトとして使用され (これは各 neighbor 文であらためて指定することができます) ます。さらに、**group** 文は以下の属性が指定することができます。

peeras group 内の neighbor の AS 番号を指定します。指定しない場合は動的に得ます。

localas group に対して GateD が示す AS 番号です。デフォルトでは **autonomoussystem** statement でグローバルに指定した値となります。このオプションは通常は、他の AS として見せる場合にのみ使用し、その使用は勧められません。

maxup group 内で GateD が経路情報の交換が可能な neighbor の数を指定します。デフォルトでは group 内のすべての neighbor を受け付けます。GateD はリストされた順番で **maxup** の数だけ neighbor を受け付けようとします。neighbor のうちの一つが動いていない場合は、リスト中の次 hop を受け付けます。GateD 起動後にさらに neighbor を受け付けるためには、今受け付けている neighbor のうちもっとも優先度の低いものを切り離します。

neighbor *neighbor_address*

各 neighbor 文では group 内の neighbor を指定します。この文で最低必要な引数は *neighbor_address* で、これは neighbor のホスト名か IP アドレスです。他のものはすべて省略できます。

preference *preference* この neighbor から得た経路情報の preference を指定します。**egp** statement で指定した EGP のデフォルトの preference と異なることがあり、GateD は別の neighbor や別の neighbor グループからの経路情報より優先させることができます。この preference は import statement で明示的にあらためて指定することができます。

preference2 *preference* *preference* が同じ場合の二つ目の preference で、*preference2* をそれを決めるのに使用することができます。デフォルトでは 0 です。

metricout *metric* この neighbor に送信するすべての経路情報に対して使用されるメトリックを指定します。この値は **egp** statement でのデフォルトのメトリックや export statement で指定したメトリックをあらためて指定することになりますが、指定した neighbor や neighbor グループに対してだけ適用されます。

nogendefault EGP でデフォルトの経路情報を正しく受信しても、GateD がデフォルトの経路情報を生成しないように指定します。デフォルトの経路情報は **gendefault** オプションが有効の場合のみに生成されます。

importdefault EGP の経路情報にデフォルトの経路情報 (0.0.0.0) が含まれていた場合、GateD がデフォルトの経路情報を受け入れるように指定します。指定していないと EGP の経路情報にデフォルトの経路情報が含まれていても、無視されます。効率を考慮して、EGP の外部境界ルータに EGP の莫大な経路情報を送信させないためにデフォルトの経路情報をアナウンスするネットワークもあります。

exportdefault GateD に、neighbor に送る EGP の経路情報にデフォルトの経路情報 (0.0.0.0) を含めるように指定します。これにより、EGP 経由でデフォルトの経路情報をアナウンスすることができます。通常は、EGP の経路情報にはデフォルトの経路情報は含まれません。

gateway *gateway* neighbor と同じネットワークに存在していない場合、**gateway** で neighbor から経路情報を得るために次 hop への同じネットワーク上のゲートウェイを指定します。このオプションはほとんど使用しません。

lcladdr *local_address* neighbor と接続する際に使用する自側のアドレスを指定します。自側のアドレスは neighbor と同じネットワークか、**gateway** を指定している場合は、そのルータへのゲートウェイと同じネットワークにあるネットワークインタフェースでなければいけません。neighbor との接続が可能になるのは関連する自側のネットワークインタフェース (ここを通して neighbor やそこへのゲートウェイと直接到達可能となる) が使用可能になった場合です。

sourcenet *network* EGP POLL パケットで使用するネットワークを指定します。デフォルトでは、これは neighbor と同じネットワークです。neighbor と同じネットワークで無い場合は、neighbor のネットワークのうちの一つを指定すべきです。このオプションは neighbor と同じネットワークにいない場合にその neighbor のネットワークを指定するために使用することもできます。このオプションは通常は不要です。

p1 *time*

minhello *time* EGP の *HELLO* パケットの最小送信間隔を指定します。デフォルトの HELLO の間隔は 30 秒です。neighbor が三回連続して HELLO パケットに応答しないと、GateD はそのルータから経路情報の取得を停止します。長い間隔を指定すると neighbor に応答する機会をより多く与えることになります。minhello は EGP で定義された *p1* の別名です。

p2 *time*

minpoll *time* EGP の POLL の間隔を指定します。デフォルトでは 120 秒です。三回連続して POLL に応答が無いと、peer は"ダウン"していると見なされ、そのルータから得た経路情報は経路情報のデータベースから削除されます。POLL の間隔を長くすると、経路情報データベースはそれだけ安定しますが、経路情報が変わった場合の反応はそれほどよくありません。minpoll は EGP で定義された ¡code¿P2¡/code¿ の別名です。

ttl *ttl* デフォルトでは GateD はローカルな neighbor への IP の TTL は *1* とし、ローカルで無い neighbor への TTL は 255 とします。このオプションは TTL が 1 のパケットを無視するような正しく機能していないルータと接続する場合に指定します。

traceoptions *trace_options* この neighbor に関するトレースオプションを指定します。デフォルトでは、EGP のグローバルオプションを引き継ぎます (trace statement や以下の EGP 固有のトレースオプションを参照して下さい)。

4.17.3 トレースオプション

EGP では、**state** と **policy** オプションが有効です。

パケットトレースオプション (**detail** や **send** や **recv** で変更可能です) は以下の通りです。

packets すべての EGP パケットのトレースを行います。

hello neighbor への到達可能性を決めるために使用される *HELLO/I-HEARD-U* パケットのトレースを行います。

acquire EGP の接続の確立や終了を行うために使用される *ACQUIRE/CEASE* パケットのトレースを行います。

update 経路情報の要求や受信を行うために使用される *POLL/UPDATE* パケットのトレースを行います。

4.18 BGP statement

4.18.1 BGP

Border Gateway Protocol (BGP) は AS 間で経路情報を交換するのに使用される exterior routing protocol です。BGP は transit AS 間や stub AS 間と同様に複数の transit AS 間と経路情報の交換をするのに使用されます。BGP は EGP と関連はしていますが、能力的に優れていて、より高い柔軟性を持ち、より少ない通信帯

域で済みます。BGP は各経路情報に対してより多くの情報を含んでいる経路属性を使用します。特に、経路情報が通過して来た各 AS 番号を含む *AS* 経路を保持し、これには絶えず変化するトポロジーでの経路のループを防ぐために十分な情報が含まれています。経路属性は管理上の優先順位を決める場合に経路情報を区別するために使用することもできます。これによりいろいろな管理上の目的を果たすための経路情報の優先順位の決定に柔軟性を持たせることができます。

BGP は peer との接続において、AS 内部 (IBGP として表されることもあります)(internal) と AS 外部 (external) との二つの基本的なタイプをサポートしています。external な接続は異なる AS に存在する peer 間で行われるのに対して、internal な接続は同じ AS 内での peer 間で行われます。経路情報を external peer へ送信する場合は、自分の AS 番号が AS 経路の一番前に追加されます。これは external peer から受信した経路情報にその peer の AS 番号が AS 経路の先頭にあることを保証するためです。internal peer から受信した経路情報には普通は自分の AS 番号は先頭に追加されません。これは internal peer が external peer から経路情報を受信した場合の経路情報の AS 経路と普通は同じであるためです。AS 経路に AS 番号が無い経路情報は internal peer から正しく受信したものとすることができます。この場合は、受信した経路情報は自分の AS 内部への経路情報であるとするべきです。

GateD の BGP の実装ではバージョン 2、3、4 の三つをサポートしています。BGP 2 と 3 の能力と機能はほぼ同じです。また、クラスのネットワークのみ伝搬することができ、AS 経路は AS 番号の簡単な配列です。BGP 4 はアドレスとマスクを伝搬することができ、AS 経路には異なる経路情報を集約したことを表すいくつかの構造体があります。

external BGP との接続ではメトリックを一つ持つものや持たないものがあるかもしれません。これは BGP では経路属性の *Multi-Exit Discriminator* と呼んでいます。BGP 2 や 3 ではこのメトリックは符号無し 16 ビットの大きさで、BGP 4 では符号無し 32 ビットの大きさです。いずれの場合も小さい値のメトリックの方が優先されます。現状では、このメトリックは同じ AS からの等しい preference を持つ経路情報のうちどちらを採用するかを決定するために使用されます。internal peer との接続では経路属性に少なくとも一つメトリックが含まれ、これを BGP では *LocalPref* と呼びます。このメトリックの大きさは *Multi-Exit Discriminator* の場合と同じです。BGP 2 や 3 ではこのメトリックは値が小さいほどよいとされ、BGP 4 では大きいほどよいとされています。BGP 4 での接続では internal 接続にオプションで *Multi-Exit Discriminator* の内部版として二番目のメトリックを使用することもできます。これらのメトリックの使用は指定した内部処理のタイプによって決まります。

BGP は同じような経路属性を持つ経路情報をアナウンスするために経路情報を一つにまとめます。一つにまとめられた経路情報は一つにまとめられたまま、再びアナウンスされます。peer がいなくなることによって生じる混乱は最小限に抑えられ、peer との接続で送信される最初のアナウンスは必要最小限となります。BGP はカーネルからメッセージ毎に情報を得るのではなく、まず入力バッファに入れます。そして、再度読み込むようなことはせずにバッファ中の完全なメッセージを処理します。BGP はソケットにキューイングされたすべての入力データをクリアするために同時に読み込むこともします。この特性により BGP の接続によってその間他のプロトコルの処理をブロックするかもしれません。

すべての到達不能メッセージは一つのメッセージにまとめられ、経路情報の更新時に到達可能経路よりも先に送られます。このような到達不能アナウンスにおいてはその経路情報でのゲートウェイは接続時の自側アドレスになり、メトリックは送られず、path origin は incomplete に設定されます。external peer との接続では到達不能なアナウンスでの AS 経路は自 AS に設定され、internal peer との接続では AS 経路は長さ 0 に設定されます。

GateD が実装している BGP では external peer への接続は同じネットワークで直接接続していることを想定していて、また、この peer がアナウンスする次 hop はそのネットワーク上のホストであることを想定しています (しかし、この制限はテスト用の設定を使用することにより軽減されます)。しかし、internal peer への接続では、group で指定することによっていくつか選択肢があります。*internal* group タイプでは peer はすべて同じネットワーク内にあるものとしていて、そのため external peer と同様に BGP の経路情報での次 hop をパケットをフォワードするゲートウェイとして直接使用することができます。*routing* group タイプではその代わりに、フォワードするアドレスとして peer から得た経路情報中の次 hop を使用したり、IGP からの経路情報中の次 hop を見付けるための次 hop を使用することによって、経路情報中の直接の次 hop を決めようとします。これらのグループでは異なるネットワーク上の peer との接続をサポートしていますが、直接の次 hop を決定するために使用される IGP からの経路情報を知っている必要があります。最後に *igp* group タイプでは同じグループの peer からの経路情報をフォワードのためには使用しないものとしています。その代

わりに、BGP からの経路情報は IGP からも受信され、BGP からの経路情報は IGP からの経路情報に関係する経路属性を決定するために使用されるものとしています。これらのグループでも異なるネットワーク上の peer をサポートしていますが、同時に動作している IGP からも経路情報を知っている必要があります。

共通のポリシーに基づいてすべての peer に対する送信メッセージを作成する internal BGP グループ (とテストグループ) では、グループ内のすべての peer へは各ルータに応じて次 hop を変えながらそのメッセージが送られます。これによって、このタイプのグループ中の膨大な数の peer に対する処理負荷を最小に抑えます。BGP statement では *allow* 宣言文で適切なグループを設定しておけば peer の指定をしていなくても構いません。

4.18.2 BGP statement

```
bgp yes | no | on | off
[ {
        preference preference ;
        defaultmetric metric ;
        traceoptions trace_options ;
        group type ( external peeras autonomous_system )
                | ( internal peeras autonomous_system )
                | ( igp peeras autonomous_system proto proto )
                | ( routing peeras autonomous_system proto proto
                        interface interface_list )
                | ( test peeras autonomous_system )
        {
                allow {
                        network
                        network mask mask
                        network masklen number
                        all
                        host host
                } ;
                peer host
                        [ metricout metric ]
                        [ localas autonomous_system ]
                        [ nogendefault ]
                        [ gateway gateway ]
                        [ preference preference ]
                        [ preference2 preference ]
                        [ lcladdr local_address ]
                        [ holdtime time ]
                        [ version number ]
                        [ passive ]
                        [ sendbuffer number ]
                        [ recvbuffer number ]
                        [ indelay time ]
                        [ outdelay time ]
                        [ keep [ all | none ] ]
                        [ analretentive ]
                        [ noauthcheck ]
                        [ noaggregatorid ]
                        [ keepalivesalways ]
                        [ v3asloopokay ]
```

[**nov4asloop**]
[**logupdown**]
[**ttl** *ttl*]
[**traceoptions** *trace_options*]
;
} ;
}] ;
external | **internal** | **igp** | **test**

bgp statement で BGP をどのように使用するかを指定します。デフォルトでは BGP は使用しません。BGP 経由でアナウンスする経路情報のデフォルトのメトリックではメトリックを送信しないようになっています。

preference *preference*

BGP 経由で得た経路情報の優先度を指定します。デフォルトの preference は 170 です。この preference は **group** statement や **peer** statment あるいは import statement であらためて指定することができます。

defaultmetric *metric*

BGP 経由で経路情報をアナウンスする場合に使用するメトリックを指定します。指定しない場合はメトリックは伝搬されません。このメトリックは neighbor 宣言文や group 宣言文あるいは export statement であらためて指定することができます。

traceoptions *trace_options*

BGP で使用されるトレースのオプションを指定します。これはデフォルトではグローバルトレースオプションから引き継がれます。この値は group 宣言文や neighbor 宣言文であらためて指定することができます (trace statement や以下の BGP 固有のトレースオプションを参照して下さい)。

4.18.3 グループ

peer はそのタイプや peer が属する AS によってグループ化されます。グループの数はいくつでも指定しても構いませんが、それぞれ唯一のタイプと AS でなければいけません。以下に五つの使用可能なタイプを示します。

group type external peeras *autonomous_system*

標準的な external BGP のグループでは送受信するすべてのアナウンスには完全なポリシーチェックが行われます。external peer はネットワークインタフェースのネットワークと同じネットワーク上でなければいけません。デフォルトでは、外部アナウンスにはメトリックは含まれていなくて、次 hop は同じネットワークから求められます。

group type internal peeras *autonomous_system*

例えば SMDS ネットワークや MILNET のように IP レベルの IGP が無いような internal BGP グループです。このグループの peer はすべて一つのネットワークインタフェース経由で直接つながっていなければいけません。次 hop への情報はこのネットワークインタフェースから求められます。このグループのアナウンスでは import statement や export statement を使用することができます。external peer あるいは EGP から受信した経路情報はデフォルトでは受信したメトリックで再びアナウンスされます。

group type igp peeras *autonomous_system* **proto** *proto*

IGP の ASE 機構を使用する internal BGP グループです。このグループでは IGP が export する経路情報を調べ、IGP タグに経路属性が全く無い場合にのみアナウンスします。AS 経路、経路の発信元および伝搬されるべきオプションの属性のみが経路情報とともに送られます。メトリックは送られず、次 hop は peer との接続時に使用したローカルアドレスが設定されます。internal peer からの経路情報はアナウンスには再び使用されることはありません。その代わりに、IGP から得た経路情報に AS 経路情報を付加し、それをアナウンスするために使用します。このグループでは IGP が export した経路情報の一部のみを送るので IGP からの経路の export には export statement が使用されます。同一グループ内の peer からの経路情報をアナウンスしないようにする必要はありません。これはアナウンスされた経路情報は IGP がすでに export している経路情報だからです。

group type routing peeras *autonomous_system* **proto** *proto* **interface** *interface_list*

フォワードするゲートウェイのアドレスを見付けるために IGP の経路情報を使用する internal BGP のグループです。このタイプのグループでは external の経路を同じネットワーク上に無いルータとの間で交換し、BGP でのその経路に対する次 hop を IGP の経路情報によって見付けたフォワードすべきゲートウェイと同じものにしています。本来 IGP が AS 内部の経路情報のみを伝搬してきたのに対して、internal BGP は AS external の経路情報を伝搬するために使用され、IGP を使用して次 hop を見付けます。

proto は BGP 経路での次 hop を見付けるために使用する IGP の名前で、IGP の設定で使用した名前を使用することができます。デフォルトでは BGP 経路情報の次 hop は peer との接続時に使用したローカルアドレスとなります。これはこのアドレスへの経路情報は IGP 経由で伝搬されるものとしているためです。*interface_list* はネットワークインタフェースリストで、そこへの経路情報は IGP で伝搬され、別のルータが次 hop として使用することができます。

group type test peeras *autonomous_system*

テスト用の peer を用いたポリシー固定の external BGP の拡張です。このポリシー固定の特別な拡張を用いて、peer のテストを比較的容易に行うことができます。テストする peer は同じネットワーク上にある必要はありません。テストする peer が同じネットワーク内にある場合は、アナウンスされた次 hop はそのネットワークから求められ、そうでない場合は次 hop は自ホストが知っている次 hop となります。テストする peer からのすべての経路情報は破棄され、すべての BGP からのアナウンス可能な経路情報はそのテスト peer に送り返されます。EGP あるいは BGP からの経路情報からのメトリックはアナウンスされますがそうでなければメトリックが無かったということになります。

グループオプション

BGP statement 内には **group** 宣言文があり、その中には **peer** 宣言文があります。peer 宣言文は group 宣言文内にいくつでも指定することができます。group 宣言文では普通、peer グループ毎のデフォルトのオプションを指定し、このオプションはその中の peer 宣言文すべてに適用されます。peer 宣言文内のオプションはグループ全体でのデフォルト (これは各 peer 毎にあらためて指定することができます) とするために group 宣言文で指定することができます。

4.18.4 peer の指定

グループ内では peer は二つのうちのいずれかの方法で指定します。これは明示的に **peer** 宣言文で指定するか、あるいは **allow** 宣言文で暗に指定することができます。以下にそれを示します。

allow allow 宣言文により指定したネットワークとマスクの範囲内のアドレスからの *peer* からの接続が可能となります。この場合の peer のオプションは group 宣言文で指定しなければいけません。peer に対する内部的な管理領域は open 要求を受信した時に作成され、接続が切れた時に解放されます。ネットワーク/マスクの指定の詳細は、経路情報のフィルタリングの項を参照して下さい。

peer *host* peer 宣言文により個々の peer を指定します。peer 宣言文は group 宣言文でデフォルトとして指定したすべてのオプションを引き継ぎます。このデフォルトは peer 宣言文で明示的に指定することによりあらためて指定することができます。

group 宣言文内では個々の peer や、あるいは **allow** 宣言文を使用して接続可能な peer のグループを指定することができます。**allow** 宣言文はアドレスとマスクを指定して使用します。GateD が指定した一連のアドレスから接続要求を受けると、GateD はそれを受け入れ、peer との経路情報の交換のための準備を行います。

peer オプション

peer 宣言文によって以下のオプションを使用することができます。なお、このオプションは **group** 宣言文でも指定することができます。すべて省略可能です。

metricout *metric* 指定した場合は、指定した peer に送られる経路情報すべてにこのメトリックが最初のメトリックとして使用されます。このメトリックによって group 宣言文や export statement で指定したメトリックをあらためて指定することになります。

localas *autonomous_system* GateD がこのグループの peer に示す AS を指定します。デフォルトは **autonomoussystem** statement でグローバルに指定したものとなります。

nogendefault BGP の有効な経路情報を受信しても GateD がデフォルトの経路情報を生成しないように指定します。デフォルトの経路情報は **gendefault** オプションが有効の場合のみに生成されます。

gateway *gateway* peer と同じネットワークに存在していない場合、*gateway* で peer から経路情報を得るために次 hop への同じネットワーク上のゲートウェイを指定します。このオプションはほとんど必要ありません。

preference *preference* peer から得た経路情報の preference を指定します。**bgp** statement で指定した BGP のデフォルトの preference と異なることがあり、GateD はある peer やある peer グループからの経路情報を他のものよりも優先することができます。この preference は import statement で明示的にあらためて指定することができます。

preference2 *preference preference* が同じ場合の二つ目の preference で、*preference2* をそれを決めるのに使用することができます。デフォルトでは 0 です。

lcladdr *local_address* peer と TCP 接続する際に使用する自側のアドレスを指定します。*external* peer にとっては自側のアドレスは peer か *gateway* を指定している場合は、そのルータへのゲートウェイと同じネットワークにあるネットワークインタフェースでなければいけません。指定した自側のネットワークインタフェース (ここを通して peer やその peer へのゲートウェイと直接到達可能となる) が使用可能になると、external peer との接続が可能となります。指定した自側のネットワークインタフェースのうちいずれかが使用可能になると、その他のタイプの peer との接続が可能となります。いずれの場合も指定した peer に一致すると指定した自側のアドレス宛に来た接続要求は認識されます。

holdtime *time* peer と接続する場合に peer と調整する BGP の Hold Time を秒で指定します。BGP では GateD が keepalive や update や notification メッセージを BGP OPEN メッセージの Hold Time フィールドで指定した間受信しないと、BGP の接続を切ります。値は 0 (keepalives を送信しません) か少なくとも 3 以上でなければいけません。

version *version* この peer とで使用する BGP のバージョンを指定します。何も指定しない場合は、はじめに大きいバージョンが使用され、バージョンの調整を行います。指定した場合は、指定したバージョンのみ調整時に使用されます。現在サポートしているバージョンは 2、3 および 4 です。

passive peer へは自発的に OPEN を発行しないように指定します。GateD は peer が OPEN を発行するのを待つことになります。デフォルトでは明示的に指定したすべての peer に対しては自発的に OPEN を発行し、peer が応答するまで OPEN メッセージを送信し続けます。

sendbuffer *buffer_size*

recvbuffer *buffer_size* カーネルに要求する送受信バッファの大きさを制御します。最大値は 65535 バイトですが、カーネルによってはもっと低い制限値があります。デフォルトでは、GateD は最大値を指定します。このオプションは通常に動作しているシステムでは必要ありません。

indelay *time*

outdelay *time* 経路情報が不安定になるのを弱めるために指定します。**indelay** は peer から得た経路情報を GateD の経路情報データベースに取り込むまでの間その経路情報が安定していなければいけない時間です。**outdelay** は経路情報を GateD の経路情報データベースから BGP に export する間その経路情報が存在していなければいけない時間です。デフォルトの値はそれぞれ 0 で、これはこの機能を使わないことを示します。

keep all 経路情報の AS 経路に自分が export した AS 番号が含まれていても、peer から得た経路情報を保持するように指定します。

analretentive 経路情報の重複があったり存在していない経路情報の削除を行うようなあやしい BGP UPDATE メッセージを受信すると、GateD にウォーニングメッセージを出すように指定します。通常はこのようなメッセージは無視されます。

noauthcheck 通常、GateD は入力パケットの認証フィールドはすべて 1 であることを確認します。このオプションは他の認証形式を使用しているような peer との通信を可能にするために使用することができます。

noaggregatorid GateD に aggregator 属性のルータ ID を (そのルータ ID の代わりに) 0 とするように指定します。これは AS 内の別のルータが AS 経路の異なる集約経路を作成しないようにするためです。

keepalivesalways update が正常に keepalive の役割をしていても、GateD が keepalive を常に送信するように指定します。これにより、この点でプロトコルに完全にしたがっていないルータと接続することができます。

v3asloopokay デフォルトでは AS 経路がループしている (すなわち AS 経路中に同じ AS が 2 回以上ある) 経路情報はバージョン 3 の external peer にはアナウンスしません。このオプションを設定することにより、この制限が無くなります。これは internal BGP グループや peer 宣言文での指定では無視されます。

nov4asloop バージョン 4 の external peer には AS 経路がループした経路情報はアナウンスしないように指定します。これは経路情報を誤ってバージョン 3 の peer に送ってしまうような peer へこのような経路情報をアナウンスしないようにする場合に有用となるでしょう。

logupdown peer が *ESTABLISHED* 状態になったり、その状態から抜けたりすると syslog 経由でログを取るように指定します。

ttl *ttl* デフォルトでは GateD は IP の TTL をローカルな peer に対しては **1**、ローカルな peer 以外に対しては 255 を設定します。このオプションは TTL が 1 のパケットを無視するような正しく機能していないルータと接続しようとする場合に主に指定します。すべてのカーネルが指定した TCP 接続の TTL を指定できるわけではありません。

traceoptions *trace_options* この peer に関するトレースオプションを指定します。デフォルトでは、BGP のグローバルオプションを引き継ぎます (trace statement や以下の BGP 固有のトレースオプションを参照して下さい)。

4.18.5 トレースオプション

state オプションは BGP でのみ有効ですが、正確な状態遷移情報を示すわけではないことに留意して下さい。

パケットトレースオプション (**detail** や **send** や **recv** で変更可能です) は以下の通りです。

packets すべての BGP パケットのトレースを行います。

open peer との接続を確立するために使用される BGP *OPEN* パケットのトレースを行います。

update ネットワーク到達可能性情報を渡すために使用される BGP *UPDATE* パケットのトレースを行います。

keepalive peer への到達可能性を確認するために使用される BGP *KEEPALIVE* パケットのトレースを行います。

4.19 ICMP statement

BSD のルーティングソケットが無いシステムでは、GateD はシステムが受信する ICMP メッセージを受け入れます。現在の GateD は ICMP リダイレクトのみ処理をしますが、ルータディスカバリメッセージの処理を行うような機能を将来追加するかもしれません。ICMP リダイレクトメッセージの処理は **redirect** statement で行います。

現状では **icmp** statement を指定する理由は GateD が受信する ICMP メッセージのトレースを行うことができるようにするためです。

4.19.1 ICMP statement

icmp {

traceoptions *trace_options* ;

}

traceoptions *trace_options* ;

ICMP のトレースオプションを指定します (trace statement や以下の ICMP 固有のトレースオプションを参照して下さい)。

4.19.2 トレースオプション

パケットトレースオプション (**detail** や **recv** で変更可能です) は以下の通りです。

packets すべての ICMP パケットのトレースを行います。

redirect ICMP *REDIRECT* パケットのみのトレースを行います。

routerdiscovery ICMP *ROUTER DISCOVERY* パケットのみのトレースを行います。

info mask 要求/応答、info 要求/応答、echo 要求/応答および time stamp 要求/応答の ICMP パケットのみトレースを行います。

error time exceeded、parameter problem、unreachable および source quench の ICMP パケットのみトレースを行います。

4.20 redirect statement

4.20.1 リダイレクト処理

リダイレクト処理部にはICMPパケットをモニタして得られるICMPやISOのリダイレクトパケットあるいはシステムがサポートしているルーティングソケット経由からの情報が渡されます。そこではリダイレクト要求を処理し、そのリダイレクトを受け入れるかどうかを決めます。リダイレクトを受け入れた場合は、その経路情報はGateDのルーティングテーブルに*redirect*というプロトコルで設定されます。リダイレクトによってできた経路情報は3分後に削除されます。

GateDがリダイレクトを受け入れないとした場合には、カーネルのフォワーディングテーブルが変更されていないかどうかを調べようとします。ICMPメッセージをモニタしているようなシステムでは、カーネルがリダイレクトパケットをどのように処理しているかを考慮することによりこれを行います。ルーティングソケットがあるシステムでは、リダイレクトを受け入れたかどうかをカーネルが示してくれます。GateDではリダイレクトの処理がされなかったものは無視します。

GateDがカーネルのフォワーディングテーブルの状態が変わっていると判断すると、正しい状態に戻すために必要な要求をカーネルに行います。

現在のシステムではルータとして機能していてもICMP処理をしないようにすることはできないことに留意して下さい。リダイレクトを無視するために、GateDはリダイレクト毎に状態が変更されているカーネルのフォワーディングテーブルをもとに戻さなければいけません。これらのことを行うためにリダイレクトの影響がカーネルにどう現れるかを示すインタフェースがあります。

デフォルトではGateDは(RIP、HELLO、OSPFやIS-ISのような) interior gateway protocolに加わっていない場合はリダイレクトによる経路情報を削除します。リダイレクトを自動的に受け入れないようにしてしまうと受け入れるようにすることはできません。*nobroadcast*モードではRIPやHELLOではリダイレクトを無視させることはできませんし、EGPやBGPでも同じです。この場合はリダイレクト処理をしないように指定しなければいけません。

最新のIETFのRouter Requirementsにしたがって、GateDはすべてのICMPのネットワークリダイレクトはホストリダイレクトとして処理するようになっていることに留意して下さい。ICMPのネットワークリダイレクトを受けると、GateDはカーネルのフォワーディングテーブルがネットワークリダイレクトの代わりに、ホストリダイレクトによる変更になるようにカーネルに要求します。

redirect statementではシステムがリダイレクトを送信しないようにはできません。リダイレクトを受け入れないようにするだけです。

4.20.2 redirect statement

redirect yes | **no** | **on** | **off**
[{
    preference *preference* ;
    interface *interface_list*
        [**noredirects**] | [**redirects**] ;
    trustedgateways *gateway_list* ;
    traceoptions *trace_options* ;
}] ;

preference

リダイレクトによって作成された経路情報のpreferenceを指定します。デフォルトでは30です。

interface *interface_list*　**interface** statementによってネットワークインタフェース毎にリダイレクトを受け入れるかどうかを指定できます。*interface_list*の表記法の説明に関してはネットワークインタフェースリストの項を参照して下さい。指定可能なオプションは以下の通りです。

noredirects 指定したネットワークインタフェースから受信したリダイレクトを無視するように指定します。デフォルトではすべてのネットワークインタフェースからのリダイレクトを受け入れます。

redirects これがデフォルトです。このオプションはネットワークインタフェースの表記でワイルドカードを指定して **noredirects** を使用する場合に必要です。

trustedgateways *gateway_list*

どのゲートウェイからのリダイレクトを受け入れるかを指定します。*gateway_list* は単にホスト名かアドレスのリストです。デフォルトでは同じネットワーク上のルータがリダイレクトを送信しても受け入れるようになっています。しかし、**trustedgateways** が指定されていると、ここで指定されたゲートウェイからのリダイレクトのみが受け入れられます。

traceoptions *trace_options*

リダイレクト固有のトレースオプションはありません。*normal* のトレースではエラー以外のトレースが行われます。

4.20.3 トレースオプション

リダイレクト固有のトレースオプションはありません。*normal* のトレースではエラー以外のトレースが行われます。

4.21 Router Discovery statement

4.21.1 router discovery protocol

router discovery protocol はルータが存在していることを知らせるために使用されている IETF の standard protocol です。これは、ホストで RIP のような経路制御プロトコルのパケットを傍受する代わりに使用されます。ホストに静的に設定されたデフォルトの経路情報の代わり、あるいはそれに追加して使用されます。

このプロトコルは二つの部分から構成されていてそれぞれ、ルータ側で動作するサーバ部とホスト側で動作するクライアント部から成ります。GateD ではこれらを二つの別のプロトコルとして扱い、同時にはどちらか一つしか使用することができないようになっています。

4.21.2 router discovery サーバ

router discovery サーバはルータ側で動作し、自分が存在していることをホストにアナウンスします。これは周期的に **Router Advertisement** を使用可能なネットワークインタフェースにマルチキャストあるいはブロードキャストすることにより行います。この Router Advertisement にはそのネットワーク上のすべてのルータのアドレスとそれをデフォルトのルータとして使用する場合の優先順位が含まれています。

最初はこの Router Advertisement は数秒毎に行われ、その後数分毎に行われるようになります。さらに、ホストではルータが (マルチキャストあるいはブロードキャストが使用できなければ) ユニキャストの Router Advertisement で応答することになる **Router Solicitation** を送信することができます。

Router Advertisement にはアナウンスしたアドレスがどの期間有効であるかを示す *Advertisement Lifetime* フィールドが含まれています。この有効期限はこの有効期限が切れる前に次の Router Advertisement が送信されるような時間が指定されます。有効期限に 0 が指定されると一つあるいはそれ以上のアドレスがすでに無効となっていることを示しています。

IP マルチキャストがサポートされているシステムでは Router Advertisement はデフォルトで all-host のマルチキャストアドレスである *224.0.0.1* に送信されます。しかし、ブロードキャストアドレスの使用も指定することができます。Router Advertisement が all-host のマルチキャストアドレスで送信されたり、ネットワークインタフェースが limited-broadcast アドレスである *255.255.255.255* を使用するように指定されている場合は、そのネットワークインタフェースに指定されたすべての IP アドレスが Router Advertisement に含まれます。Router advertisement がネットワークやサブネットワークのブロードキャストで送信される場合は、そのネットワークやサブネットワークに係わるアドレスのみが含まれます。

4.21.3 router discovery server statement

```
 routerdiscovery server yes | no | on | off [ {
                traceoptions trace_options ;
                interface interface_list
                                [ minadvinterval time ]
                                [ maxadvinterval time ]
                                [ lifetime time ]
                                ;
                address interface_list
                                [ advertise ] | [ ignore ]
                                [ broadcast ] | [ multicast ]
                                [ ineligible ] | [ preference preference ]
                                ;
} ] ;
```

traceoptions *trace_options*

router discovery protocol のトレースオプションを指定します (trace statement や以下の router discovery protocol 固有のトレースオプションを参照して下さい)。

interface *interface_list*

ネットワークインタフェースに対するパラメータを指定します。GateD の他の statement での指定とは少し異なっていることに留意して下さい。**interface** ではネットワークインタフェース (例えば *le0*、*ef0* や *en1* など) 名のみを指定し、**address** ではプロトコルアドレス (この場合は IP アドレス) を指定します。

interface のパラメータは以下の通りです。

maxadvinterval *time* ネットワークインタフェースからブロードキャストあるいはマルチキャストの Router Advertisement を送信する間隔の最大値を指定します。この値は、*4* 秒以上 *30:00*(30 分または 1800 秒) 以下でなければいけません。デフォルトでは **10:00**(10 分 または 600 秒) です。

minadvinterval *time* ネットワークインタフェースから自発的にブロードキャストあるいはマルチキャストの Router Advertisement を送信する間隔の最小値を指定します。この値は *3* 秒以上で **maxadvinterval** を越えてはいけません。デフォルトでは **0.75** × **maxadvinterval** です。

lifetime *time* Router Advertisement の有効期限を指定します。この値は **maxadvinterval** 以上で、*2:30:00* (2 時間 30 分または 9000 秒) 以下でなければいけません。デフォルトでは **3** × **maxadvinterval** です。

address *interface_list*

このネットワークインタフェースに指定されたアドレスに対するパラメータを指定します。GateD の他の statement での指定とは少し異なっていることに留意して下さい。**interface** ではネットワークインタフェース (例えば *le0*、*ef0*、*en1* など) 名のみを指定し、**address** ではプロトコルアドレス (この場合は IP アドレス) を指定します。

advertise Router Advertisement に指定したアドレスが含まれるように指定します。これがデフォルトです。

ignore Router Advertisement に指定したアドレスが含まれないように指定します。

broadcast システムでIPマルチキャストをサポートしていないか、ネットワーク上の他のホストがIPマルチキャストをサポートしていない場合に指定したアドレスがブロードキャストのRouter Advertisementで送信されるように指定します。一つのネットワークインタフェースからブロードキャストのRouter AdvertisementやマルチキャストのRouter Advertisementを混ぜて送信することも可能です。IPマルチキャストをサポートしていない場合はこれがデフォルトとなります。

multicast 指定したアドレスがマルチキャストのRouter Advertisementで送信されるように指定します。システムでIPマルチキャストがサポートされていない場合は、指定したアドレスは送信されません。システムでIPマルチキャストがサポートされている場合は、デフォルトではネットワークインタフェースがIPマルチキャストをサポートしていればマルチキャストのRouter Advertisementで送信され、ネットワークインタフェースがIPマルチキャストをサポートしていなければブロードキャストのRouter Advertisementで送信されます。

preference *preference* 指定したアドレスのデフォルトのルータのアドレスとしての優先度を指定します。これは同じサブネット上の他のルータのアドレスとの相対的なものです。これは32ビットの符号付きの2の補数の整数で、大きな値ほどより優先されます。*0x80000000*は**ineligible**としてのみ指定することができることに留意して下さい。デフォルトでは**0**です。

ineligible 指定したアドレスは0x80000000の優先度であることを指定し、これはどのホストからもデフォルトのルータとなることが不適当だということを意味しています。

これは指定したアドレスがデフォルトのルータとすべきでは無くても、ICMPリダイレクトで次hopとして指定されるような場合に便利です。これにより、ホストは指定したアドレスが有効であることを調べることができます。

4.21.4 router discovery クライアント

IPマルチキャストが使用可能で有効であれば、ホストではall-hostのマルチキャストアドレス(*224.0.0.2*)経由で、そうでなければネットワークインタフェースのブロードキャストアドレス経由でのRouter Advertisementを待ちます。GateD起動時あるいは再設定時にホストはRouter Solicitationsを何度かall-routerのマルチキャストアドレスである*224.0.0.2*あるいはネットワークインタフェースのブロードキャストアドレスに対して送信します。

0以外の有効期限を持つRouter Advertisementを受信すると、ホストはデフォルトの経路情報をアナウンスされた各アドレスに設定します。優先度が**ineligible**であったり、アドレスが同じネットワーク内に無い場合は、その経路情報は使用できないものとなりますが、保持はされます。優先度がineligibleで無い場合、優先度の働きとしてメトリックが設定され、もっともよい優先度を持つルータが使用されます。同じ優先度を持つ複数のアドレスを受信した場合は、もっとも小さいIPアドレスを使用します。これにより得たデフォルトの経路情報は他のプロトコルにexportすることはできません。

有効期限が0のRouter Advertisementを受信した場合、ホストはそのルータから得た経路情報をすべて削除します。さらにICMPリダイレクトで示されたアドレスがこれらのアドレスに含まれる場合もすべて削除されます。有効期限が切れる前にあらたにRouter Advertisementを受信しない場合、同様のことが生じます。

4.21.5 router discovery client statement

```
routerdiscovery client yes | no | on | off [ {
            traceoptions trace_options ;
            preference preference ;
            interface interface_list
                          [ enable ] | [ disable ]
                          [ broadcast ] | [ multicast ]
                          [ quiet ] | [ solicit ]
                          ;
} ] ;
```

traceoptions *trace_options*

router discovery protocol のトレースオプションを指定します (trace statement の項や以下の *router discovery* protocol 固有のトレースオプションを参照して下さい)。

preference *preference*

router discovery で得たすべてのデフォルトの経路情報の preference を指定します。デフォルトでは **55** です。

interface *interface_list*

ネットワークインタフェースに対するパラメータを指定します。GateD の他の statement での指定とは少し異なっていることに留意して下さい。**interface** ではネットワークインタフェース (例えば *le0*、*ef0* や *en1* など) 名のみを指定します。router discovery クライアントにはパラメータは無く、ネットワークインタフェースのアドレスのみに適用することができます。

enable 指定したネットワークインタフェースで router discovery が動作するように指定します。これがデフォルトです。

disable 指定したネットワークインタフェースで router discovery が動作しないように指定します。

broadcast 指定したネットワークインタフェースで Router Solicitation をブロードキャストで送信するように指定します。このホストやネットワークインタフェースで IP マルチキャストが使用することができない場合はこれがデフォルトとなります。

multicast 指定したネットワークインタフェースで Router Solicitation をマルチキャストで送信するように指定します。このホストやネットワークインタフェースで IP マルチキャストが使用することができない場合は Router Solicitation は送信されません。デフォルトでは、ホストやネットワークインタフェースが IP マルチキャストをサポートしていればマルチキャストで Router Solicitation を送信し、そうでなければ、Router Solicitation はブロードキャストで送信されます。

quiet router discovery は動作しますが、このネットワークインタフェースでは Router Solicitation を送信しないように指定します。

solicit 最初の Router Solicitation はこのネットワークインタフェースで送信するように指定します。これがデフォルトです。

4.21.6 トレースオプション

router discovery のクライアントとサーバでは **state** のトレースオプションをサポートしていて、これによりプロトコルのさまざまなトレースを行うことができます。

state 状態遷移のトレースを行います。

router discovery のクライアントとサーバではパケットのトレースを直接サポートしていません。router discovery パケットのトレースは icmp statement により行うことができます。

4.22 SNMP statement

Simple Network Management Protocol (SNMP) は経路制御プロトコルではなく、ネットワーク管理プロトコルです。**snmp** statement では GateD が変数群を渡すために SNMP の多重化デーモンと接続するかどうかを制御します。SNMP デーモン (通常は *smuxd* です) は GateD とは独立に動作させなければいけません。**snmp** statement は GateD が管理ソフトウェアにその状態を知らせるかどうかを制御するだけです。

GateD は RFC 1227 で説明されている SMUX プロトコル経由で SNMP デーモンとデータの交換をします。

4.22.1 snmp statement

snmp yes | **no** | **on** | **off**
[{
    port *port* ;
    debug ;
    traceoptions *traceoptions* ;
}] ;

SNMP とのデータの交換は **yes** あるいは **on** で行うようになり、**no** あるいは **off** で行わないようになります。デフォルトでは **on** です。

port *port* デフォルトでは SMUX デーモンは 199 のポート番号で要求を待ちます。GateD は明示的にポート番号を指定することにより、SMUX デーモンと別のポート番号でデータの交換をするように指定できます。

debug このオプションを指定すると、ISODE SMUX のデバッグが可能となります。デフォルトではデバッグはしないようになっています。

traceoptions *trace_options* SMUX のトレースオプションを指定します (trace statement や以下の SMUX 固有のトレースオプションを参照して下さい)。

4.22.2 トレースオプション

SNMP 固有のトレースオプションはありません。SNMP デーモンから SMUX プロトコル経由で受信した SNMP の要求は通常のパケットとしては扱われません。**detail**、**send** および **recv** オプションはサポートしていません。

receive SMUX デーモンから受信した SNMP の要求とそれに対する応答のトレースを行います。

register 変数を登録するプロトコル要求をトレースします。

resolve 変数名を解決するプロトコル要求をトレースします。

trap プロトコルからの SNMP トラップ要求をトレースします。

4.23 kernel statement

カーネルとのインタフェースは技術的には経路制御プロトコルではありませんが、さまざまな特徴を有していて、GateD では経路制御プロトコルと同様に扱います。GateD がカーネルのフォワーディングテーブルに設定すると選択した経路情報はカーネルが実際にパケットのフォワードに使用されます。

カーネルのフォワーディングテーブルを更新するために GateD が使用しなければならない追加、削除および変更には結構時間がかかります。これは特に時間にクリチカルではなく、また多くの経路情報を容易に扱っていなかった比較的古い経路制御プロトコル (RIP や EGP) の場合には問題となっていませんでした。比較的新しい経路制御プロトコル (OSPF や BGP) ではより厳密なタイミングが要求され、多くの経路情報を処理するために使用されます。このようなプロトコルを使用する場合には、カーネルインタフェースの速さがクリチカルになります。

GateD が多くの経路情報を設定するのにかなり長い間動かなくならない (実際 1 分以上動かないことがあった) ようにするためにこれらの経路情報の処理は今のところまとめて (バッチで) 行われています。このバッチの大きさは次に説明するオプションを変えることにより制御することができますが、通常は適度に機能するようにデフォルトのパラメータがあります。

通常の終了処理時に GateD は **retain** となっているものを除いてカーネルのフォワーディングテーブルに設定したすべての経路情報を削除します。オプションで、GateD はカーネルのフォワーディングテーブルを削

除せず、すべてそのままにしておくこともできます。この場合、**retain** の経路情報がフォワーディングテーブルに設定されていることを保障するためにフォワーディングテーブルの変更がされるかもしれません。これは多くの経路情報を持つシステムで GateD を再起動する場合にこの経路情報を再設定しないようにするような場合に便利です。これにより再起動時のリカバリの時間を大きく抑えることができます。

4.23.1 フォワーディングテーブルとルーティングテーブル

パケットのフォワードを制御するカーネル内のテーブルはフォワーディングテーブルであり、ISO では *forwarding information base* あるいは *FIB* として知られています。経路制御プロトコルから得た GateD が内部的に使用する経路情報を保持するテーブルはルーティングテーブルであり、ISO では *routing information base* あるいは *RIB* と呼ばれています。ルーティングテーブルはさまざまなプロトコルから得た経路情報を集め、保持するために使用されます。ネットワークとマスク毎に使用可能な経路情報が選択され、これがもっともよい (数字が小さい) *preference* を持つ経路情報となります。すべての使用可能な経路情報は、カーネルのフォワーディングテーブルに設定されます。このテーブルにはカーネルが実際にパケットをフォワードする場合に使用する情報となります。

4.23.2 フォワーディングテーブルの更新

カーネルの *FIB* を更新する主な方法は二つあり、それぞれ *ioctl()* インタフェースとルーティングソケットインタフェースです。これらのさまざまな特徴をここで説明します。

ioctl インタフェースによるフォワーディングテーブルの更新

フォワーディングテーブルに対する *ioctl* インタフェースは *4.3 BSD* で導入され、*4.3 BSD* で広く使用されました。これは一方通行型のインタフェースで、GateD はカーネルフォワーディングテーブルの更新のみ可能です。この他にもいくつか制限があります。

サブネットマスク固定 4.3 BSD のネットワークの実装ではあるネットワークのすべてのサブネットは同じサブネットマスクであることを想定していました。これはカーネルによって制限されています。ネットマスクはカーネルのフォワーディングテーブルには保持されていませんが、パケットをフォワードする時に同じネットワークのネットワークインタフェースを調べて決めています。

一方通行型のインタフェース GateD はカーネルのフォワーディングテーブルを更新することはできますが、フォワーディングテーブルが変更されたことはわかりません。しかし、GateD は ICMP を受信して、カーネルが ICMP リダイレクトによってフォワーディングテーブルがどのように更新されたかを判断することはできます。

フォワーディングテーブルの更新がわからない GateD はシステム管理者が *route* コマンドを使用したためのカーネルのフォワーディングテーブルの変更を見付けることはできません。*ioctl()* インタフェースを使用しているシステムで *route* コマンドを使用することは GateD の動作を大きく妨げることになります。

フォワーディングテーブルの変更がサポートされていない すべての実装において、カーネル内にある経路情報を変更するためのインタフェースがサポートされていません。この場合は、経路情報を削除して新たに追加しなければいけません。

ルーティングソケットインタフェースによるフォワーディングテーブルの更新

カーネルのフォワーディングテーブルに対するルーティングソケットインタフェースは *4.3 BSD Reno* で導入され、*4.3 BSD Net/2* で広く配布され、*4.4 BSD* で改善されました。このインタフェースは UDP に似た単なるソケットであり、これによりカーネルと GateD がメッセージを交換します。これには *ioctl()* に比べていくつか利点があり、以下に示します。

可変長サブネットマスク ネットマスクを明示的にカーネルに渡します。これにより、同じネットワーク内のサブネットで異なるマスクを使用することができます。また、natural ネットマスクよりももっと短いマスクの経路情報を使用することもできます。これはクラスレスルーティングとして知られています。

双方向型のインタフェース GateD はこのインタフェースを通してカーネルのフォワーディングテーブルを変更することができるだけでなく、カーネルは GateD にフォワーディングテーブルの変更を通知することができます。これのもっとも興味深いところはリダイレクトによってカーネルのフォワーディングテーブルが変更されたことが通知されるということです。これは GateD がもはやリダイレクトを知るために ICMP メッセージをモニタする必要が無いということを意味しています。それに加えて、カーネルがリダイレクトを処理したかどうかが通知されるので、GateD は安心してカーネルが処理しなかったリダイレクトメッセージを無視することができます。

フォワーディングテーブルの更新がわかる *route(8)* コマンドも含めて、他のプロセスが行ったフォワーディングテーブルの変更がルーティングソケット経由でわかります。これにより、GateD はカーネルのフォワーディングテーブルがルーティングテーブルと同期していることを確実にします。それに加えて、GateD が動作中でもシステム管理者が *route(8)* コマンドでフォワーディングテーブルの操作をすることができます。

フォワーディングテーブルの変更がサポートされています カーネルの経路情報を一回で変更できる変更メッセージがあります。ルーティングソケットの初期のバージョンには変更メッセージの処理にバグがありました。コンパイル時と設定時に変更メッセージを使用する代わりに削除して追加するようにできます。

拡張可能 新しいメッセージのタイプを追加することにより、カーネルと GateD のあらたなメッセージの交換方式を追加することができます。

4.23.3 フォワーディングテーブルの読み込み

GateD は起動するとカーネルのフォワーディングテーブルを読み込み、関連する経路情報をルーティングテーブルに設定します。この経路情報は *remnant*(残りもの) と呼ばれていて、指定した間 (デフォルトでは 3 分) の後タイムアウトするか、あるいはより優先度の高い経路情報があればそれを保持します。これにより、経路制御プロトコルが経路情報を取得し始めるまでの間フォワーディングを行います。

カーネルからフォワーディングテーブルを読み込む主な方法は三つあります。

kmem 経由によるフォワーディングテーブルを読み込み

多くのシステム、特に 4.3 BSD ベースのシステムでは GateD はカーネル内のデータの構造を知り、現在のフォワーディングテーブルの状態を読み込むためにカーネル内をあちこち捜しまわらなければいけません。この方法は遅く、GateD が読み込みの途中でフォワーディングテーブルが更新されるとエラーを生じることになります。これは、GateD が起動中にシステム管理者が *route* コマンドを使用したり、ICMP リダイレクトメッセージを受けたりすると起こり得ます。

OSF/1 などのような 4.3 BSD Reno やそれ以降をベースにしていても問題のあるようなシステムには以下で示すカーネルの内部構造を知らなくてもカーネルから経路情報を読み込むことができる *getkerninfo()* というシステムコールはありません。このようなシステムではカーネルメモリをあちこち捜してカーネルから radix tree を読み込む必要があります。この場合は hash ベースのフォワーディングテーブルを読み込むよりもエラーが多い可能性があります。

getkerninfo/sysctl システムコールによるフォワーディングテーブルの読み込み

ルーティングソケットとは別に 4.3 BSD Reno では *getkerninfo()* システムコールを導入しました。このシステムコールにより、ユーザプロセス (GateD もその一つですが) がカーネル内のデータ構造を知らなくてもカーネルからいろいろな情報を読み込むことができます。フォワーディングテーブルの場合は GateD に一連のルーティングソケットメッセージとしてひとつひとつ返ってきます。これにより、GateD がフォワーディングテーブルを読み込み中にフォワーディングテーブルの変更に関する問題が無くなります。

4.4 BSD では *getkerninfo()* のインタフェースを *sysctl()* に変更しました。これはパラメータが違いますが、機能は同じです。

OS 固有の方法によるフォワーディングテーブルを読み込み

例えば SunOS 5 のようなシステムではカーネルのフォワーディングテーブルを読み込むのに独自の方法を定義しています。SunOS 5 では *getkerninfo()* の方法と概念的に同じです。

4.23.4 ネットワークインタフェースリストの読み込み

GateD の機能の一部であるカーネルインタフェースはネットワークインタフェースの状態を周期的に読み込みを行います。GateD はネットワークインタフェースの変化を見つけ、peer との経路情報の交換を開始したり停止したりできるように各プロトコルに知らせます。ネットワークインタフェースのリストは以下の二つの方法で読み込まれます。

SIOCGIFCONF によるネットワークインタフェースリストの読み込み

4.3 BSD、4.3 BSD Reno、4.3 BSD Net/2 をベースにしたシステムでは *SIOCGIFCONF* の *ioctl* インタフェースを使用してネットワークインタフェースのリストを読み込みます。この方法を用いるとネットワークインタフェースのリストと基本的な情報が SIOCGIFCONF によって返ります。その他の情報はネットワークインタフェースのネットマスク、フラグ、MTU、メトリック、相手アドレス (point-to-point のネットワークインタフェースのみ) およびブロードキャストアドレス (ブロードキャスト型のネットワークインタフェース) を得るための別の ioctl() を発行することにより得なければいけません。

変更がないかどうかを調べるために GateD はこのリストを 15 秒毎に読み込みます。ルーティングソケットが使用されている場合は、経路情報が変更されたことを示すメッセージを受けると常にこのリストを読み込みます。また、*SIGUSR2* のシグナルを受けても GateD はネットワークインタフェースのリストを読み込みに行きます。読み込む間隔は interface statement で明示的に指定することができます。

sysctl によるネットワークインタフェースリストの読み込み

4.4 BSD では *sysctl* システムコール経由でネットワークインタフェースリストを読み込むことができるようになりました。ネットワークインタフェースの状態はルーティングソケットのメッセージとしてひとつひとつ返ってきて、GateD では必要な情報を得るために解析します。

4.4 BSD ではネットワークインタフェースの状態の変化を直ちに通知するルーティングソケットメッセージも追加しています。これにより、GateD はネットワークインタフェースの状態の変化にすぐに反応することができます。

この方法を使用する場合は、GateD はネットワークインタフェースを一分に一回しか読み込まなくなります。また、*SIGUSR2* のシグナルを受けても GateD はネットワークインタフェースのリストを読み込みに行きます。読み込む間隔は interface statement で明示的に指定することができます。

4.23.5 ネットワークインタフェースの物理アドレスの読み込み

getkerninfo() と *sysctl()* インタフェースの最近のバージョンでは、ネットワークインタフェースの情報の一部として物理アドレスを返すようになっています。この情報が返ってこないようなシステムでは、GateD はこの情報を得るために IFF_BROADCAST がセットされたネットワークインタフェース (この場合 Ethernet ドライバと同じように扱えるものと見なしています) を捜してネットワークインタフェースリストを検索します。*SunOS 4* や *SunOS 5* のようなシステムでは、この情報を得るためにシステム固有のインタフェースを使用します。

ネットワークインタフェースの物理アドレスは IS-IS で有用です。IP では今のところ使用されていませんが、将来使用されるかもしれません。

4.23.6 カーネル変数の読み込み

起動時に、GateD はカーネルからある変数を読み込みます。これは通常、*nlist* (あるいは *kvm_nlist*) のシステムコールで行われますが、別の方法を使用するシステムもあります。

変数読み込みには *UDP* チェックサム、*IP forwarding* およびカーネルのバージョン (動作環境の情報を得るため) があります。フォワーディングテーブルをカーネルメモリから直接読み込むようなシステムでは、ハッシュテーブルあるいは *radix tree* のフォワーディングテーブルが読み込まれます。ネットワークインタフェースの物理アドレスを他の方法で読み込むことができないようなシステムでは、ネットワークインタフェースリストが読み込まれます。

4.23.7 特殊な経路情報のフラグ

最近の *BSD* ベースのカーネルでは以下に示すような特殊な経路情報フラグをサポートしています。

RTF_REJECT 通常の経路情報ではパケットをフォワードしますが、*RTF_REJECT* フラグの付いた経路情報ではパケットを廃棄し、発信者に到達不可能メッセージを送るようになります。このフラグはループバックのネットワークインタフェースへの経路情報でのみ有効です。

RTF_BLACKHOLE フラグの付いた経路情報ではパケットを廃棄しますが、発信者に到達不可能メッセージは送りません。このフラグはループバックのネットワークインタフェースへの経路情報でのみ有効です。

RTF_STATIC GateD が起動すると、GateD はカーネルのフォワーディングテーブルに設定されているすべての経路情報を読み込みます。ネットワークインタフェースへの経路情報を除いて、通常はすべて GateD の起動前からの *remnant(*残りもの*)* として数分後には削除されます。これは *route* コマンドで追加された経路情報が GateD が起動後には保持されないということになります。

このようにならないために *RTF_STATIC* フラグが追加されました。ネットワークインタフェースに対するものではない経路情報が *route* コマンドを使用して追加された場合、*RTF_STATIC* フラグがセットされます。これは GateD にこの経路情報がシステム管理者によって追加され、保持されるべきものであることを知らせています。

4.23.8 カーネルコンフィギュレーション

```
kernel {
        options
                [ nochange ]
                [ noflushatexit ]
                ;
        remnantholdtime time ;
        routes number ;
        flash
                [ limit number ]
                [ type interface | interior | all ]
                ;
        background
                [ limit number ]
                [ priority flash | higher | lower ]
                ;
        traceoptions trace_options ;
} ;
```

options *option_list*

カーネルのオプションを指定します。有効なオプションは以下の通りです。

nochange ルーティングソケットをサポートしているようなシステムでは経路情報の変更処理をせずに、削除と追加のみを使用するように指定します。これは経路情報の変更処理が正しくないような初期バージョンのルーティングソケットを使用している場合に有用です。

noflushatexit 通常の終了処理では GateD は **retain** の指定がない経路情報はカーネルのフォワーディングテーブルから削除するようになっています。**noflushatexit** では終了処理時に経路情報の削除を行わないように指定します。その代わり、**retain** するように指定されたすべての経路情報が設定されていることを保証するために経路情報は変更されたり追加されたりします。

これは何千もの経路情報を保持するシステムにおいては便利です。起動時に GateD はどの経路情報がカーネルのフォワーディングテーブルに設定されているかを認識し、もとには戻しません。

remnantholddimte *time*

通常、GateD の起動時にカーネルのフォワーディングテーブルから読み込まれる *remnant*(残りもの) の経路情報は 180 秒後にタイムアウトするか、あるいは即座に経路情報を上書きします。このオプションではこの間隔を 0 から 900 秒の間に指定できます。0 に指定するとこの経路情報は直ちに削除されます。

routes *number*

システムによってはカーネルメモリが貴重な場合があります。このような場合、このパラメータで GateD がカーネルに設定する経路情報の数を制限するように指定することができます。通常、GateD は経路情報の追加、変更や削除をネットワークインタフェース、AS 内部経路そして AS 外部経路の順で行います。すなわち、ネットワークインタフェースの経路情報をはじめにキューに入れ、その次に AS 内部経路、AS 外部経路の順でキューに入れ、そのキューを始めから処理して行きます。このオプションが指定されていて、経路情報数の制限に達すると、GateD はそのキューリストを二度検索します。はじめの検索では削除を行い、そこではキューに入れた変更の情報を追加の情報にしながら変更された経路情報をすべて削除します。次の検索では制限に達するまでネットワークインタフェース、AS 内部経路そして AS 外部経路の順で経路情報を追加します。これは AS 外部経路よりも AS 内部経路が好ましくなるようになっています。デフォルトでは、カーネルのフォワーディングテーブル内の経路情報の数の制限はありません。

flash

経路情報が変更された場合に、それを各プロトコルに知らせる処理は *flash update* と呼ばれます。カーネルのフォワーディングテーブルのインタフェースへは最初に知らされます。通常は一つの flash update で最大 20 の経路情報を処理することができます。**flash** ではこのパラメータの調整を行います。

limit *number* 一つの flash update で処理する経路情報の最大数を指定します。デフォルトでは *20* です。*-1* という値を指定すると、指定したタイプのすべての変更中の経路情報を flash update で処理します。

type interface | interior | all flash update で処理する経路情報のタイプを指定します。**interior** は AS 内部経路情報 (section 4.3.2 での interior gateway protocols の定義を参照して下さい) も設定するように指定します。**all** は AS 外部経路情報 (section 4.3.3 での exterior gateway protocols の定義を参照して下さい) も含んだ経路情報を設定するように指定をします。デフォルトではネットワークインタフェースへの経路情報を設定するように指定する **interface** が flash update で使用されます。

flash limit -1 all を指定すると、*flash update* ではすべての経路情報が設定されます。この場合が以前のバージョンの GateD と同じ動作になります。

background

flash update では通常ネットワークインタフェースへの経路情報しか設定されていなくて、その他の経路情報はバックグラウンド、これは経路制御プロトコルのパケットを受信していない時にまとめて処理されます。通常、一度に 120 の経路情報を設定することによって他の処理も働くようになっています。このバックグラウンド処理は flash updates よりも低い優先度で行われます。以下のパラメータではこれらの調整を行います。

limit *number* バッチで一度に処理する経路情報の数を指定します。デフォルトでは 120 です。

priority flash | higher | lower カーネルの経路情報の更新のためのバッチ処理と *flash update* 処理との優先順位を指定します。デフォルトでは **lower** となっていて、flash updates が先に処理されます。カーネルの経路情報の更新と flash updates を同じ優先度で処理する場合は **flash** を指定します。低い優先順位で処理する場合は **lower** を指定します。

4.23.9 トレースオプション

カーネルとのインタフェースは技術的には経路制御プロトコルではありませんが、多くの場合同じように扱います。以下の二つのシンボルはコマンドラインで入力した場合に意味があります。これは設定ファイルを解析する前にそのトレースにかかわる処理を行うためです。

symbols *nlist()* あるいは同等なインタフェースによってカーネルからシンボルを読み込みます。

iflist ネットワークインタフェースリストを検索します。これは設定ファイルを解析する前にネットワークインタフェースのリストの検索処理を行うのでコマンドラインで指定すると有効となります。

以下のトレースオプションは設定ファイルでのみ指定することができます。コマンドラインでの指定は無効となります。

remnants GateD 起動時のカーネルからの経路情報の読み込みをトレースします。

request GateD がカーネルのフォワーディングテーブルに対して行った経路情報の追加、削除や変更をトレースします。

以下の通常のオプションやパケットトレースオプションはカーネルと経路情報を交換するために使用するルーティングソケットを使用しているシステムでのみ適用することができます。カーネルに対して 4.3 BSD の古い *ioctl()* インタフェースを使用するシステムに対しては適用することはできません。

info 経路情報の参照の失敗や経路情報の参照要求のようなルーティングソケットから得た情報メッセージのトレースを行います。GateD は現状ではこのメッセージの処理は行いませんが、指定されればその内容をログに書き込みます。

パケットトレースオプション (**detail** や **send** や **recv** で変更可能です) は以下の通りです。

routes 追加、削除や変更メッセージや他のプロトコルからの追加、削除や変更メッセージを含むカーネルと交換されたメッセージのトレースを行います。

redirect カーネルからのリダイレクトメッセージのトレースを行います。

interface カーネルからのネットワークインタフェースの状態を示すメッセージのトレースを行います。これは 4.4 BSD がベースのシステムでのみサポートされています。

other 上記 **info** で述べたものを含む、その他のカーネルからのメッセージのトレースを行います。

4.24 static statement

4.24.1 static statement

static statement では GateD が使用する静的経路情報を指定します。一つの **static** statement でいくつも経路情報を指定することができます。**static** statement は gated.conf 内では protocol statement 後でかつ、control statement の前になければいけません。**static** statement はいくつでも設定でき、それぞれいくつでも経路情報を指定することができます。この経路情報のうち好ましい preference 値を持つ経路情報の方が優先されます。

```
static {
        ( host host ) | default |
        ( network [ ( mask mask ) | ( masklen number ) ] )
                gateway gateway_list
                [ interface interface_list ]
                [ preference preference ]
                [ retain ]
                [ reject ]
                [ blackhole ]
                [ noinstall ] ;
        ( network [ ( mask mask ) | ( masklen number ) ] )
                interface interface
                [ preference preference ]
                [ retain ]
                [ reject ]
                [ blackhole ]
                [ noinstall ] ;
} ;
```

host *host* **gateway** *gateway_list*

network [(**mask** *mask*) | (**masklen** *number*)] **gateway** *gateway_list*

default gateway *gateway_list*

これは静的経路情報の指定のもっとも一般的な形式です。これは静的経路情報に対して一つ以上のゲートウェイを指定します。静的経路情報は指定した一つ以上のゲートウェイがネットワークインタフェースと同じネットワークの場合に設定されます。複数の適切なゲートウェイが利用可能な場合、複数経路がサポートされているかどうかで制限されます (これは UNIX ではコンパイル時のパラメータで常に一つになるようになっています)。

静的経路情報のオプションは以下の通りです。

interface *interface_list* このオプションが指定されていると、ゲートウェイがこのネットワークインタフェースのうちの一つと同じネットワークの場合にのみ有効です。*interface_list* の表記法の詳細は section 4.8.2 のネットワークインタフェースリストを参照して下さい。

preference *preference* この静的経路情報の preference を指定します。preference によってこの経路情報と他のプロトコルからの経路情報との優先度を決定します。デフォルトの preference は 60 です。

retain 通常、GateD は終了時にカーネルのフォワーディングテーブルからネットワークインタフェースに対するもの以外のすべての経路情報を削除します。**retain** オプションはこのように静的経路情報が削除されないようにするために使用することができます。これは GateD が動作していない場合に経路情報がいくつかあることを保証するのに有用です。

reject 通常の経路情報にしたがってパケットをフォワードする代わりに、*reject* で示された経路情報ではパケットを破棄し、発信者に到達不能メッセージを送るようにします。このオプションを指定すると reject で示される経路情報を設定することができます。すべてのカーネルのフォワーディング機構が reject で示される経路情報をサポートしているわけではありません。

blackhole *blackhole* で示される経路情報は到達不能メッセージを送らない以外は *reject* で示される経路情報と同じです。

noinstall 通常もっとも低い preference を持つ経路情報がカーネルのフォワーディングテーブルに設定され、他の経路制御プロトコルに export されます。**noinstall** を指定するとその経路情報が有効でもカーネルのフォワーディングテーブルには設定されませんが、他の経路制御プロトコルには export されます。

(*network* [(**mask** *mask*) | (**masklen** *number*)]) **interface** *interface*

この形式では一つのネットワークインタフェースに複数のアドレス使用をサポートしているような場合の静的なネットワークインタフェースへの経路情報を指定します。**preference**、**retain**、**reject**、**blackhole** および **noinstall** オプションは上の説明と同じです。

4.25 経路フィルタと AS 経路

4.25.1 経路情報のフィルタリング

経路情報のフィルタはある経路情報がマッチすることになる一連の経路です。マッチした結果の処理は経路情報のフィルタリングの使われ方によります。アドレスやアドレスとマスクを指定することにより経路情報をフィルタリングします。`martians` や **import** や **export** の statement 中で経路情報のフィルタリングは使用されます。

もっとも当てはまるフィルタにより経路情報をフィルタリングします。同じマスクやオプションを持つフィルタリングアドレスを複数指定するとエラーとなります。

4.25.2 filtering syntax

経路情報のフィルタリングは以下のような一つあるいは複数のリストからなります。

network [**exact** | **refines**]
network **mask** *mask* [**exact** | **refines**]
network **masklen** *number* [**exact** | **refines**]
all
default
host *host*

上記が経路情報のフィルタリングで指定することができるフォーマットです。すべてのフォーマットがどこでも指定することができるわけではなく、例えば **host** や **default** は **martians** では無効です。多くの場合、フィルタの内容に関連したパラメータを追加指定することができます。例えば、**martians** statement では **allow** というキーワードを指定することができますし、**import** statement では preference を指定することができますし、また **export** statement ではメトリックを指定することができます。

network [**exact** | **refines**]

network **mask** *mask* [**exact** | **refines**]

network **masklen** *number* [**exact** | **refines**]

フィルタリング時には通常はアドレスとマスクが必要です。マスクには次の三つの形式を使用します。この三つの形式はマスクをどのように指定するかで変わります。一つ目の形式ではマスクはネットワークのネットマスクです。二つ目の形式ではマスクは明示的に指定されます。三つ目の形式ではマスクは連続したビットの数で指定されます。

オプションパラメータが指定されないと、ネットワークとマスクの範囲内に当てはまるすべての経路情報に一致し、経路情報のマスクは無視されます。通常のネットワークを指定している場合は、ネットワークとそのサブネットとそのホストに一致することになります。また、以下の二つのオプションにより経路情報のマスクをどのように扱うかを指定することもできます。

exact このオプションにより経路情報のマスクと指定したマスクが正確に 一致した場合に一致するように指定します。これはネットワークには一致するがサブネットやネットワーク内のホストに一致させないために使用します。

refines 経路情報のマスクが指定したマスクよりもより一致する (すなわちより長い) 場合に一致するように指定します。これはサブネットや (あるいは) ネットワーク内のホストに一致するが、ネットワークには一致させないために使用します。

all

このオプションは全てに一致させる場合に指定します。これは以下と等価です。

```
0.0.0.0 mask 0.0.0.0
```

default

デフォルト経路に一致させる場合に指定します。一致させるには、アドレスはデフォルトアドレスでありマスクはすべて 0 でなければいけません。これは以下と等価です。

```
0.0.0.0 mask 0.0.0.0 exact
```

host *host*

指定したホストに一致させる場合に指定します。一致させるには、アドレスは指定したホストのアドレスでありネットマスクはホストマスク (すなわちすべて 1) でなければいけません。これは以下と等価です。

host **mask** 255.255.255 **exact**

4.25.3 AS 経路の一致

AS 経路は経路情報がルータへ到着するのに通過する autonomous_systems のリストであり、AS 経路の発信元も示しています。これは destination ネットワークへの経路を選択するために使用されます。GateD でこれを行うための主な方法は経路情報の *import* や *export* 時の AS 経路に適用するパターンリストを指定することです。

経路情報が AS を通過すると AS 経路の最初に通過した AS 番号が追加されます。

origin により、AS 経路情報を区別します。origin が **igp** の場合は、interior routing protocol から得たものであり、AS 経路はほとんど完全である場合です。origin が **egp** の場合は (例えば EGP のように)AS 経路をサポートしていない exterior routing protocol から得たものであり、AS 経路は完全とはいえない場合です。この場合 AS 経路情報がまったく完全でないと **incomplete** の origin が使用されます。

AS 経路の表記法は RFC 1164 の section 4.2 で定義されています。

4.25.4 AS 経路の表記法

AS 経路に一致する指定は以下の表記法を用いて行います。

aspath *aspath_regexp* **origin any** | ([**igp**] [**egp**] [**incomplete**])

これにより指定した origin で *aspath_regexp* に一致する AS 経路を指定します。

4.25.5 AS 経路の正規表現

技術的には AS 経路の正規表現は AS 番号とアルファベットがセットになった正規表現です。AS 経路の正規表現は一つ以上の AS 経路表現により構成されます。AS 経路表現は AS 経路記述子 (aspath_term) と AS 経路記号 で構成されます。

AS 経路記述子

AS 経路記述子 は以下の三つのうちのいずれかです。

autonomous_system
.
(*aspath_regexp*)

autonomous_system 1 から 65534 までの有効な AS 番号です。

. いずれの AS 番号にも一致します。

(*aspath_regexp*) 括弧により括弧内の表現をまとめます。*** や *?* のような記号は括弧内の一つの要素や正規表現に作用します。

AS 経路記号

AS 経路 operator は以下のうちのいずれかです。

aspath_term **m,n**
aspath_term **m**
aspath_term **m,**
aspath_term *****
aspath_term **+**
aspath_term **?**
aspath_term | *aspath_term*

aspath_term **m,n** AS 経路記述子に続く *m,n* (m と n はいずれも負でない整数であり $m \leq n$ です) は、少なくとも m 回、多くても n 回 AS 経路記述子を繰り返すことを表します。

aspath_term **m** AS 経路記述子に続く *m* (m は正数です) は、ちょうど m 回 AS 経路記述子を繰り返すことを表します。

aspath_term **m,** AS 経路記述子に続く *m,* (m は正数です) は、m 回以上 AS 経路記述子を繰り返すことを表します。

aspath_term ***** AS 経路記述子に続く *** は、*0* 回以上 AS 経路記述子を繰り返すことを表します。これは *0,* の省略形です。

aspath_term + AS 経路記述子に続く *+* は、*1* 回以上 AS 経路記述子を繰り返すことを表します。これは *1,* の省略形です。

aspath_term ? AS 経路記述子に続く *?* は、*0* 回あるいは *1* 回 AS 経路記述子を繰り返すことを表します。これは *0,1* の省略形です。

aspath_term | *aspath_term* 左右いずれかの AS 経路記述子に一致します。

4.26 経路情報の import

経路制御プロトコルから経路情報を import したり、GateD の経路情報のデータベースに経路情報を取り込む場合に **import** statement で制御します。**import** statement の形式は import するプロトコルにより変化します。

4.26.1 preference の指定

経路情報を import する場合にはその経路情報に preference を指定することができます。この preference は複数の次ホップがある場合に最適な次ホップを選択するために使用されます。preference は次の二つのオプションのうちのいずれかとともに指定することができます。

restrict ルーティングテーブルにこの経路情報を取り込みたくない場合に指定します。これはルーティングテーブルにこの経路情報を設定しないことを意味することもあります。そうでない場合は負の preference で設定されます。これにより、この経路情報は使用可能になっていないためルーティングテーブルに設定されたり、他のプロトコルに export されることはありません。

preference *preference* 他のプロトコルから得た経路情報と比較する場合の preference 値を指定します。最も低い preference を持つ経路情報が使用可能な経路情報であり、カーネルのフォワーディングテーブルに設定され、他のプロトコルに export されるものとして選択されます。デフォルトの preference は各プロトコルで設定されます。

4.26.2 経路情報のフィルタリング

経路情報のフィルタはプロトコルが import する経路情報を明示的に指定するために使用されます。経路情報をフィルタリングする場合の形式を以下に示します。この詳細な働きの説明ついては section 4.25.1 を参照して下さい。経路情報のフィルタリングが指定されていない (すなわち **restrict** が statement の最初の行に指定されている場合) と、import した経路情報すべてが statement に一致することになります。経路情報のフィルタが指定されている場合は、指定したフィルタに一致する経路情報だけが import されます。経路情報のフィルタが別々に指定されている場合は **all restrict ;** が指定したリストの最後にあるものとして見なされます。

network [**exact** | **refines**]
network **mask** *mask* [**exact** | **refines**]
network **masklen** *number* [**exact** | **refines**]
default
host *host*

4.26.3 BGP や EGP からの経路情報の import

import proto (**bgp** | **egp**) **autonomoussystem** *autonomous_system*
　　restrict ;
import proto bgp | **egp autonomoussystem** *autonomous_system*
　　[**preference** *preference*] **{**

route_filter [**restrict** | (**preference** *preference*)] **;**
} ;

import proto bgp aspath *aspath_regexp*
origin any | ([**igp**] [**egp**] [**incomplete**])
restrict ;
import proto bgp aspath *aspath_regexp*
origin any | ([**igp**] [**egp**] [**incomplete**])
[**preference** *preference*] **{**
route_filter [**restrict** | (**preference** *preference*)] **;**
} ;

最初の形式では EGP や BGP からのすべての経路情報の import を制限します。経路情報は AS によって制御されます。二つ目の形式は受け入れるべき一連の経路情報を明示的に指定するための *route_filter* を使用します。**preference** は指定したすべての経路情報のデフォルトの preference を指定するために使用することができます。

次の二つの形式は BGP 固有のものです。これは一つ目の形式が経路情報は AS 経路を使用して制御されるということ以外は同じになっています。AS 経路のマッチについては section 4.25.3 で説明しています。

EGP や BGP のバージョン 2 や 3 では *natural* ネットワークの経路情報しか伝搬することができないということに留意して下さい。そのためホストルートやデフォルトルートのフィルタリングは意味がありません。BGP のバージョン 4 では連続したビット列のネットマスクによってどのアドレスの経路情報も伝搬することができます。

EGP や BGP はいずれも経路情報のフィルタリングで指定されていなくて暗黙のうちに除外された経路情報や、明示的に **restrict** オプションで経路情報に負の preference を持たせたりして受け付けなかった経路情報を保持しています。負の preference により、経路情報は使用可能にはいけません。これによりカーネルのフォワーディングテーブルに設定されたり他のプロトコルに export されることは無くなります。このようにして、import した内容が変わっても再設定によるセッションの中断や確立が起きないようにします。

4.26.4 RIP や HELLO や リダイレクトからの経路情報の import

import proto (**rip** | **hello** | **redirect**)
[(**interface** *interface_list*) | (**gateway** *gateway_list*)]
restrict ;
import proto (**rip** | **hello** | **redirect**)
[(**interface** *interface_list*) | (**gateway** *gateway_list*)]
[**preference** *preference*] **{**
route_filter [**restrict** | (**preference** *preference*)] **;**
} ;

RIP や HELLO や ICMP リダイレクトによる経路情報の import はプロトコルや入力ネットワークインタフェースや経路情報を送ってきたゲートウェイにより制御することができます。複数指定されている場合は、もっとも一般的 (プロトコル) なものからもっとも具体的 (ゲートウェイ) なものへという順番で処理されます。

RIP や HELLO は同じプロトコル内から得た経路情報を選択するのに preference を使用することはできません。その代わりにメトリックを使用します。

これらのプロトコルでは経路情報の更新間隔は短いため、受け付けられなかった経路情報を保持することはありません。

4.26.5 OSPF からの経路情報の import

import proto ospfase [**tag** *ospf_tag*] **restrict ;**

import proto ospfase [**tag** *ospf_tag*]
    [**preference** *preference*] **{**
    route_filter [**restrict** | (**preference** *preference*)] ;
} ;

OSPF の性質上 ASE 経路情報の import のみを制御することができます。OSPF のエリア内とエリア間の経路情報はつねに preference は 10 で GateD のルーティングテーブルに import されます。tag が指定されている場合は、import statement は指定したタグを持つ経路情報にのみ適用されます。

AS 境界ルータとして機能している場合は OSPF の ASE 経路情報の import を制限することだけを行うことができます。これは **export ospfase** 文を指定することにより行われます。中身が無い export statement の指定は、ASE 経路情報が export されない場合の ASE 経路情報の import を制限するために使用することができます。

他の IGP と同様に OSPF の ASE 経路情報を選択するために preference を使用することはできません。これは、preference は OSPF のコストとして使用されるためです。設定により、受け付けないようにした経路情報は負の preference を持つ経路情報として保持されます。

4.27 経路情報の export

4.27.1 export statement

export statement はゲートウェイ上で動作しているプロトコル間でどの経路情報を再配布するのかと同様に、GateD が他のシステムにどの経路情報を伝えるかを制御します。**import** statement と同様に **export** statement の表記法はプロトコル毎に少しずつ違います。**export** statement の表記法は各オプションの意味はほとんど同じですが、**import** statement の表記法と少し異なっています。両者の主な違いは、経路情報の import は import したプロトコル毎に制御されるのに対して、経路情報の export は export するプロトコル毎とその中でどのプロトコルから import したのかによって制御されるということです。いいかえると、**import** statement ではプロトコルが *source* から import する一連の経路情報を指定します。**export** statement では *source* から得た経路情報が *destination* へアナウンスされるように指定します。例えば、RIP で受信したすべての経路情報を AS 番号が AS123 の peer にアナウンスするように BGP を設定することができます。これはプロトコル間での経路情報の再配布にあたります。

基本的に **export** statement での外側の定義では export しようとする経路情報の destination を指定します。中間の定義では考慮したい経路情報の source の制限を指定します。そして、もっとも内側の定義では個々の経路情報を選択するための経路情報フィルタを指定します。

さらに、すべての export される経路情報にメトリックを指定します。このメトリックはプロトコルによっては peer に送られ、peer が destination へのよりよい経路情報を選択するのに使用されます。以下のセクションでは、メトリックの使用法とともに各プロトコル毎の **export** statement の文法の詳細を説明します。

4.27.2 メトリックの指定

経路情報をアナウンスする場合に経路情報とメトリックを関連づけるプロトコルがあります。**export** statement では経路情報とともに送られるメトリックを指定したり、経路情報のアナウンスをしないようにするような省略可能なキーワードを指定することができます。

restrict 何も export しないように指定します。**export** statement に指定すると、このプロトコルには何も export しないように指定することになります。経路情報の source の箇所に指定すると、この source からは destination に export されないようにすることになります。経路情報のフィルタリング箇所で指定すると、そのフィルタに一致する経路情報は export されることはありません。

metric *metric* 指定したプロトコルに export する場合に使用するメトリックを指定します。

4.27.3 経路情報のフィルタリング

経路情報のフィルタリングは明示的に export する経路情報を指定するために **export** statement で使用することができます。経路情報のフィルタリングで使用可能な形式を以下に示します。これの詳細な働きの説明については section 4.25.1 を参照して下さい。経路情報のフィルタリングが指定されていない (すなわち **restrict** が statement の最初の行に指定されている場合) と、受け付けた経路情報すべてが statement に一致することになります。経路情報のフィルタが設定されている場合は指定したフィルタに一致する経路情報だけが export されます。経路情報のフィルタが別々に指定されている場合は **all restrict ;** が指定したリストの最後にあるものと見なされます。

network [**exact** | **refines**]
network **mask** *mask* [**exact** | **refines**]
network **masklen** *number* [**exact** | **refines**]
default
host *host*

4.27.4 EGP と BGP への export

export proto bgp | **egp as** *autonomous system*
        restrict ;
export proto bgp | **egp as** *autonomous system*
        [**metric** *metric*] **{**
        export_list ;
} ;

EGP や BGP への export は AS によって制御され、同じ設定が AS 内のすべてのルータに適用されます。

EGP のメトリックは 0 から 255 までで、0 がもっとも良い値です。BGP のメトリックは 16 ビットのアンサインドの大きさで、すなわち 0 から 65535 までの値で 0 が最も良い値です。なお、BGP バージョン 4 では 32 ビットのアンサインドの大きさをサポートしていますが、GateD はまだこれをサポートしていません。

export_list には export しようとする経路情報を指定します。この *export_list* の文法は経路情報をどのプロトコルから得たかにより異なり、それについては section 4.27.7 で説明しています。

export statement を指定しない場合は、ネットワークインタフェースに関する経路情報のみが **export** されます。何らかの設定が指定されている場合はその指定がデフォルトの設定よりも有効となり、**export** すべき経路情報は明示的に指定する必要があります。

EGP と BGP バージョン 2 と 3 は *natural* ネットワークのみサポートしていることに留意して下さい。したがって ホストやデフォルト の経路情報のフィルタリングは意味がありません。BGP バージョン 4 では連続したビット列のネットマスクを持つアドレスを伝えることができます。

4.27.5 RIP や HELLO への export

export proto rip | **hello**
        [(**interface** *interface_list*) | (**gateway** *gateway_list*)]
        restrict ;
export proto rip | **hello**
        [(**interface** *interface_list*) | (**gateway** *gateway_list*)]
        [**metric** *metric*] **{**
        export_list ;
} ;

RIP や HELLO への export はどのプロトコルやネットワークインタフェースや経路情報を送ってきたゲートウェイから制御することができます。何か指定されている場合はもっとも一般的 (プロトコル) なものからもっとも具体的 (ゲートウェイ) なものへという順番で処理されます。

RIP から得た経路情報を RIP に export する場合にメトリックを設定したり、HELLO から得た経路情報を HELLO に export する場合にメトリックを設定することはできません。そのような設定を行っても無視されます。

export_list には export しようとする経路情報を指定します。この *export_list* の文法は経路情報をどのプロトコルから得たかにより異なり、それについては section 4.27.7 で説明しています。

export statement を指定しない場合は、**RIP** には **RIP** から得た経路情報とネットワークインタフェースに関する経路情報が **export** され、**HELLO** には **HELLO** から得た経路情報とネットワークインタフェースに関する経路情報が **export** されます。何らかの設定が指定されている場合はその指定がデフォルトの設定よりも有効となり、**export** すべき経路情報は明示的に指定する必要があります。

RIP バージョン 1 と HELLO は同じクラス内ではすべて同じサブネットマスクであることを前提としていて、そのサブネットしか伝搬することができません。RIP バージョン 2 ではその制限が取り除かれ、バージョン 1 と互換のあるデータを送信しない場合はすべての経路情報を伝搬することができます。

自分の次 hop を指定した経路情報 (すなわち静的経路情報や内部で作成されたデフォルトの経路情報) を RIP や HELLO でアナウンスするためには **export** statement 内でメトリックを指定する必要があります。これは RIP や HELLO ではデフォルトのメトリックを設定するだけでは十分ではないためです。これはアナウンスしようとする経路情報を正確にするためのものです。

4.27.6 OSPF への export

export proto osfpase [**type 1** | **2**] [**tag** *ospf_tag*]
    restrict ;
export proto osfpase [**type 1** | **2**] [**tag** *ospf_tag*]
    [**metric** *metric*] **{**
    export_list ;
} ;

GateD の持つ経路情報を OSPF に export することによって OSPF のエリア内やエリア間の経路情報を作成することはできません。OSPF ASE 経路情報にだけ GateD の持つ経路情報を export することができます。また、OSPF プロトコル内で OSPF から得た経路情報の伝搬を制御することもできません。

OSPF ASE 経路情報には二種類あり、タイプ *1* とタイプ *2* があります。この二つのタイプの詳しい説明は OSPF statement で行っています。デフォルトのタイプは **ospf** statement 中の **defaults** 内で指定します。これは **export** statement であらためて指定することができます。

OSPF ASE 経路情報には *tag* を付けることができます。これは OSPF ルータ上で経路情報をフィルタリングするために使用する任意の 32 ビットの数字です。OSPF のタグの詳しい説明は OSPF statement でおこなっています。**ospf** 中の **defaults** 内で指定したデフォルトのタグは **export** statement であらためて指定することができます。

export_list には export しようとする経路情報を指定します。この *export_list* の文法は経路情報をどのプロトコルから得たかにより異なり、それについては section 4.27.7 で説明しています。

4.27.7 経路情報の source の指定/プロトコル間の経路情報の再配布

先の **export** statement の説明で使用した *export_list* ではどの source から得たプロトコルであるかのようなパラメータを基準にした一連の経路情報の source を指定しました。間接的には *export_list* はプロトコル間での経路情報の再配布を制御します。この文法は経路情報を得たプロトコルごとにわずかに変わります。

BGP と EGP から得た経路情報の export

proto bgp | **egp autonomoussystem** *autonomous_system*

```
                restrict ;
proto bgp | egp autonomoussystem autonomous_system
                [ metric metric ] {
                route_filter [ restrict | ( metric metric ) ] ;
} ;
```

BGP や EGP から得た経路情報はどの AS から得たかにより指定することができます。AS 経路によってすべての経路情報を export することもできます。詳しくは以下を参照して下さい。

RIP と HELLO から得た経路情報の export

```
proto rip | hello
                [ ( interface interface_list ) | (gateway gateway_list ) ]
                restrict ;
proto rip | hello
                [ ( interface interface_list ) | (gateway gateway_list ) ]
                [ metric metric ] {
                route_filter [ restrict | ( metric metric ) ] ;
} ;
```

RIP と HELLO から得た経路情報はどのネットワークインタフェースから得たかとか、どのゲートウェイから得たかにより export することができます。

OSPF から得た経路情報の export

```
proto ospf | ospfase restrict ;
proto ospf | ospfase [ metric metric ] {
                route_filter [ restrict | ( metric metric ) ] ;
} ;
```

OSPF と OSPF ASE から得た経路情報は他のプロトコルに export することができます。タグによる export は以下を参照して下さい。

ネットワークインタフェースの経路の export

```
proto direct | static | kernel
                [ ( interface interface_list ) ]
                restrict ;
proto direct | static | kernel
                [ ( interface interface_list ) ]
                [ metric metric ] {
                route_filter [ restrict | ( metric metric ) ] ;
} ;
```

プロトコル毎に export したり、その場合に使用するネットワークインタフェースによって export することができます。ここで言うプロトコルとは以下の通りです。

direct ネットワークインタフェースのネットワークに対する経路情報です。

static static statement で指定した静的経路情報です。

kernel ルーティングソケットを持つシステム上ではルーティングソケットから得た経路情報はGateD のルーティングテーブルに **kernel** というプロトコル名で登録されています。この経路情報はこのプロトコルから得たものとして export されます。これは *route(8)* コマンドで経路情報を設定し、他の経路制御プロトコルに伝搬させる場合に有用です。

集約した経路情報の export

proto default | **aggregate**
        restrict ;
proto default | **aggregate**
        [**metric** *metric*] **{**
        route_filter [**restrict** | (**metric** *metric*)] ;
} ;

このプロトコルはプロトコル毎に export することができます。

default **gendefault** オプションで作成された経路情報です。この代わりに経路情報の集約を使用することを勧めます。

aggregate **aggregate** statement や **generate** statement が使用された場合の集約した経路情報です。詳しくは経路情報の集約を参照して下さい。

AS 経路による export

proto *proto* | **all aspath** *aspath_regexp*
        origin any | ([**igp**] [**egp**] [**incomplete**])
        restrict ;
proto *proto* | **all aspath** *aspath_regexp*
        origin any | ([**igp**] [**egp**] [**incomplete**])
        [**metric** *metric*] **{**
        route_filter [**restrict** | (**metric** *metric*)] ;
} ;

BGP が設定されている場合、すべての経路情報にはルーティングテーブルに追加される時に AS 経路とともに追加されます。すべての IGP の経路情報の AS 経路の origin は IGP となっていて AS 経路には AS は含まれません (現状では AS は経路情報が export される時に付け加えているためです)。EGP の経路情報の AS 経路の origin は EGP となっていて、AS 経路がどの AS から受け入れたかを表します。BGP の経路情報にとっては AS 経路は BGP から得たものとして保持されます。

AS 経路の正規表現は AS 経路の一致で説明しています。

経路情報のタグによる export

proto *proto* | **all tag** *tag* **restrict ;**
proto *proto* | **all tag** *tag*
        [**metric** *metric*] **{**
        route_filter [**restrict** | (**metric** *metric*)] ;
} ;

OSPF と RIP バージョン 2 はいずれもタグをサポートしており、他のプロトコルでは常に 0 のタグを持つものとしています。export される経路情報はこのタグをもとに選択されます。これは経路情報がタグで分類されていて経路情報が他の経路制御プロトコルに export される場合に有用です。

4.27.8 経路情報のフィルタリングと export の例

ここでの設定例では BGP peer へ経路情報を export する場合のものです。AS 番号 AS123 の BGP peer へ以下の経路情報をアナウンスします。

- AS 番号 AS999 の (別の) BGP peer から得た BGP のすべての経路情報
- **static** statement で設定した静的なデフォルトの経路情報
- RIP から得た内部ネットワークの一連の経路情報

設定ファイルは次のようになります。

```
# AS 番号 AS123 についての export を制御します

export proto bgp as AS123 {

        # AS999 から得た BGP の経路情報

        proto bgp as AS999 { all; };

        # 静的デフォルトルート

        proto static { default; };

        # 内部の RIP からの経路情報

        proto rip {
                1.2.3 masklen 24;
                1.2.4 masklen 24;
                192.1.0;
        };
};
```

section 4.29 で他の例をあげています。

4.28 経路集約

aggregate statement. 経路集約は特定の経路情報の存在によって、より一般的な経路情報を生成する手法です。例えば、RIP から得た一つまたは複数のサブネットワークの経路情報が存在すると ASBR では EGP 経由でアナウンスすることになる一つのネットワークとしての経路情報を作成するために経路集約を使用します。GateD の古いバージョンでは *natural* ネットワークのサブネットワークのネットワークインタフェースから (クラス A、B、C の概念を用いて) *natural* ネットワークへの経路情報を作成するようなことを自動的に行っていました。しかし、これは常に正しいわけではなく、CIDR の登場でより正しくないことが多くなり、経路集約を明示的に指定しなければならなくなりました。このように **aggregate** statement で明示的に指定しないと経路集約は行わないようになっています。

経路情報の集約は地域的なネットワークや全国ネットワークでの経路情報のトラフィックの量を抑えるためにも使用されます。IP アドレスを十分考えて割り当てることにより地域的なネットワークへは数百もの経路情報の代わりに一つの経路情報をアナウンスすることができます。

集約経路情報を作成したゲートウェイでは集約経路情報は実際にパケットのルーティングで使用されることはなく、(望むならば) その情報を受けた側でのみ使用されます。したがって、ルータが集約経路情報に当て

はまらないパケットを受信した場合、ICMP の *network unreachable* のメッセージで応答することになります。これは destination ネットワークに対する経路情報が無いパケットがデフォルトの経路情報にしたがうと境界ルータにルーティングするネットワークに戻ってしまい、TTL が 0 になるまでその間を往復してしまわないようにするためです。集約経路情報の中に目的の経路情報が無い場合に unreachable のメッセージを送るのは reject route をサポートしているシステムでのみ可能です。

経路情報のもとになる条件により経路集約は微妙に変わります。これは最後の楽園の経路情報として知られていることもあります。この経路情報は次 hop や AS 経路をもっとも低い (好ましい) preference にして経路集約の生成条件となる経路情報から受け継ぎます。これは peer からその peer のバックボーンに対する経路があるかどうかによってデフォルトの経路情報を作成するということにもっともよく使用されます。

4.28.1 経路集約と経路生成の表記法

```
 aggregate default
                | ( network [ ( mask mask ) | ( masklen number ) ] )
                [ preference preference ] [ brief ] {
                proto [ all | direct | static | kernel | aggregate | proto ]
                                [ ( as autonomous system ) | ( tag tag )
                                                | ( aspath aspath_regexp ) ]
                                restrict ;
                proto [ all | direct | static | kernel | aggregate | proto ]
                                [ ( as autonomous system ) | ( tag tag )
                                                | ( aspath aspath_regexp ) ]
                                [ preference preference ] {
                                route_filter [ restrict | ( preference preference ) ] ;
                } ;
} ;

generate ( default | network [ ( mask mask ) | ( masklen number ) ] )
                [ preference preference ] {
                                [ ( as autonomous system ) | ( tag tag )
                                                | ( aspath aspath_regexp ) ]
                                restrict ;
                proto [ all | direct | static | kernel | aggregate | proto ]
                                [ ( as autonomous system ) | ( tag tag )
                                                | ( aspath aspath_regexp ) ]
                                [ preference preference ] {
                                route_filter [ restrict | ( preference preference ) ] ;
                } ;
} ;
```

経路情報のフィルタに当てはまる経路情報は集約経路情報の構成要因と呼ばれます。この経路情報は適用される集約経路情報の preference の順番に並べられます。同じ preference を持つ経路情報の構成要因が複数ある場合は、各経路情報の preference を用いて順番に並べます。集約経路の preference はもっとも小さい preference を持つ経路情報の構成要因の preference となります。

preference *preference*

集約経路の preference を指定します。デフォルトの preference は 130 です。

brief

AS 経路を短くする時にもっとも長い共通の AS 経路になるように指定します。デフォルトではすべての AS 経路の SET と SEQUENCE から構成される AS 経路です。

proto *proto*

列記してある特別なプロトコル名に加えて、経路集約に関係しているプロトコルを GateD がサポートしているプロトコルの中から選択指定することができます。

as *autonomous_system*

指定した AS から得た経路情報から選択するように制限します。

tag *tag*

指定したタグを持つ経路情報から選択するように制限します。

aspath *aspath_regexp*

指定した AS 経路に一致した経路情報を選択するように制限します。

restrict

指定した経路情報が集約経路の構成要因とは見なされないように指定します。GateD がサポートしているプロトコルのうちのいずれかがプロトコルとして指定可能です。

route_filter

以下を参照して下さい。

集約してできる経路情報が構成要因の経路情報よりも一般的な場合のみにその経路情報は集約経路の構成要因となることができます。すなわち集約経路情報のマスクで集約経路に一致しなければいけません。どの経路情報ももっとも詳しく指定した一つの集約経路にのみの構成要因となり得ますが、集約経路をより一般的な経路集約の要因とすることもできます。

4.28.2 経路情報のフィルタリング

経路情報のフィルタリングで使用可能な形式を以下に示します。どのように作用するかの詳細な説明は section 4.25.1 を参照して下さい。経路情報のフィルタリングが何も指定されていない場合 (すなわち **restrict** が statement の最初の行に指定されている場合) は、指定した source からの経路情報はすべてこの statement に一致することになります。何かフィルタが指定されている場合は、指定したフィルタに一致する経路情報のみが集約経路の構成要因と見なされます。経路情報のフィルタが別々に指定されている場合は、**all restrict ;** が指定したリストの最後にあるものと見なされます。

network [**exact** | **refines**]
network **mask** *mask* [**exact** | **refines**]
network **masklen** *number* [**exact** | **refines**]
default
host *host*

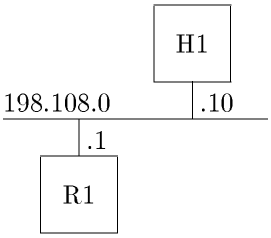

Figure 4.1: 受動的な *RIP* の受信ホスト

4.29 設定例

4.29.1 受動的な RIP 受信

ネットワークに単一な接続をして使用するホスト H1 は RIP を受信しますが、経路情報のアナウンスはしません。デフォルトの経路情報を R1 に対して設定していて、安全のために gated が終了しても (**retain** オプションを使用して) カーネル内に保持するようになっています。ログ情報は `/tmp/gated.log` というファイルに最大 10KB、ログの置き換えを一つのファイルにのみ行います。

```
#
#   この設定は RIP を quiet モード (受動) で起動し、いくら複数のネットワーク
#   インタフェースが設定されていてもパケットの受信のみを行います
#

traceoptions "/var/spool/gated/gatedlog" size 10k files 2 general;

rip yes {
        nobroadcast ;
};

static {
        0.0.0.0 gateway 198.108.0.1 retain;
};
```

4.29.2 AS 内部の RIP ルータ

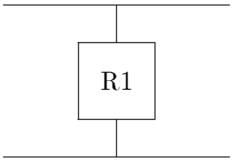

Figure 4.2: *AS* 内部の *RIP* ルータ

これは、複数のネットワークインタフェースを持つルータで RIP のみ動作させる例です。この RIP の設定ではすべてのルータからの RIP の経路情報を受け入れ、ネットワークインタフェースが属しているネットワークと同時にすべての RIP の経路情報をすべてのネットワークインタフェースに export します。

```
#
# RIP を有効にするだけです
#
rip yes;
```

4.29.3 サブネットワーク化された RIP バージョン 2 の設定

この設定例は、静的なデフォルトの経路情報を設定し、それを他の RIP ルータへアナウンスし、192.168.1 のサブネットがクラス C のナチュラルネットワークに集約され他の RIP ルータにアナウンスする以外は先の例と同じです。

```
rip yes;

# デフォルトルートの定義

static {
        default gateway 192.168.1.18; #ether address of WORLD router
};

# サブネットワーク 192.168.1.0/28 と 192.168.1.16/28 に属するネットワーク
# インタフェースから 192.168.1.0/24 の集約経路を作成

aggregate 192.168.1.0 mask 255.255.255.0 {
        proto direct {
                192.168.1.0 mask 255.255.255.240;
                192.168.1.16 mask 255.255.255.240;
        };
};

# RIP の export

export proto rip metric 2 {
        # 集約経路(クラス C のネットワーク)
        proto aggregate {
                192.168.1.0 mask 255.255.255.0;
        };
        # 静的経路
        proto static {
                default;
        };
};
```

4.29.4 BGP-RIP ルータ

(AS 1234 にある) ルータ R2 は (AS 2021 にある) ルータ R1 とセッションを張ります。R2 は R1 からのデフォルトの経路情報のみ受け、RIP から得た AS 内部の経路情報をすべて R1 にアナウンスします。また、R2 は RIP で AS 内部のルータにデフォルトの経路情報を再アナウンスします。

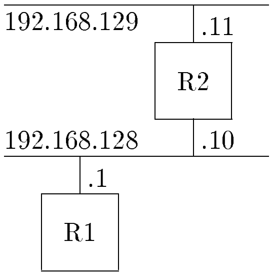

Figure 4.3: *RIP-BGP* ルータ

```
#
# 自分の AS
#
autonomousystem 1234;

#
# DMZ(192.168.128) 以外では RIP を使用
#
rip yes {
        interface le1 noripin noripout;
};

#
# ISP との BGP セッション
#
bgp yes {
        group type external peeras 2021 {
                peer 192.168.128.1;
        };
};

#
# ISP からはデフォルトの経路情報のみ import する
#
import proto bgp {
        default;
};

#
# ISP に AS 内部の経路情報を export する
#
export proto bgp {
        proto rip {
                all;
        };
};
```

```
#
# AS 内部に RIP でデフォルトの経路情報を再アナウンスする
#
export proto rip metric 2 {
        proto bgp as 2021 {
                default;
        };
        proto direct {
                all;
        };
};
```

4.29.5 経路集約を行う BGP-RIP ルータ

この設定例は、R2 が AS 内部の経路情報を集約をし、ISP には集約した経路情報のみアナウンスする以外は先の例と同じです。

```
#
# 自分の AS
#
autonomousystem 1234;

#
# DMZ(192.168.128) 以外では RIP を使用
#
rip yes {
        interface le1 noripin noripout;
};

#
# ISP との BGP セッション
#
bgp yes {
        group type external peeras 2021 {
                peer 192.168.128.1;
        };
};

#
# ISP からはデフォルトの経路情報のみ import する
#
import proto bgp {
        default;
};

#
# ISP に AS 内部の経路情報を export する
#
export proto bgp {
        proto aggregate { all; };
```

```
};

#
# AS 内部に RIP でデフォルトの経路情報を再アナウンスする
#
export proto rip metric 2 {
        proto bgp {
                default;
        };
        proto direct {all; };
};

#
# AS 内部のクラス C の 4 つのネットを /22 のネットに集約する
# 二つはネットワークインタフェースが属するネットワーク
#
aggregate 192.168.128.0 masklen 22 {
        proto direct {
                192.168.128.0 masklen 24 ;
                192.168.129.0 masklen 24 ;
        };
        proto rip {
                192.168.130.0 masklen 24 ;
                192.168.131.0 masklen 24 ;
        };
};
```

4.29.6 マルチキャストルータの設定例

ユニキャストルーティングと同様に、いくつかのマルチキャストルーティングプロトコルがあり、それぞれ利点欠点があります。GateD では、DVMRP と MOSPF と dense モードと sparse モードの PIM のサポートが期待されています。現時点では、dense モードの PIM と DVMRP のみ提供されています。これらのプロトコルはすべて IGMP に依存しています。

この設定では IGMP と PIM をすべてのネットワークインタフェースで単に有効にしています。PIM のデフォルトのモードは dense モードです。

```
#
# IGMP を使用する
#
igmp yes ;

#
# PIM を使用する
#
pim yes ;
```

トンネリングした DVMRP と同時に PIM を動作させ、GateD でマルチキャストルーティングしている場合、PIM はトンネリングした仮想ネットワークインタフェース上で動作しようとします。これは次のように無効にさせます。

```
igmp yes ;
```

```
pim yes {
        interface tu disable ;
};
```

一つのトンネルを持つホストの DVMRP の設定例です。このホストでは RIP も動作し、多くのログをとるようにしています。

```
traceoptions "/tmp/gated.log" replace  all;

rip yes;

kernel {
        traceoptions all request;
};

igmp on {
        traceoptions all packets;
};

dvmrp on {
        #
        # トンネルの定義
        # 始めに相手のアドレス、そして自側のアドレス
        #
        tunnel 198.108.60.64 lcladdr 192.231.253.50;
        traceoptions all packets;
};

static {
        default gateway 192.231.253.2  retain;
};
```

Chapter 5

運用法

5.1 gated の起動

5.1.1 システムブート時の GateD 起動

GateD は通常、システムの起動 (ブート) 時に `/etc/rc.local` で起動されます。GateD はそれが存在しているパーティションがマウントされ、*syslog(8)* が起動後に起動されることが望まれます。ネットワークインタフェースが *ifconfig(8)* で使用可能になる前に起動することも可能ですが、望むなら後で起動することもできます。GateD は例えば別のネットワーク上のシステムからの NFS マウントや *xntpd(8)* のように経路情報が必要なプログラムが起動する前に起動されなければいけません。

5.1.2 GateD の直接起動

通常は次のように数行を `/etc/rc.local` に追加することにより GateD を起動します。

```
if [ -f /etc/gated.conf -a -f /usr/local/sbin/gated ]; then
    /usr/local/sbin/gated ; echo -n "gated "
fi
```

5.1.3 gdc による GateD の起動

gdc による GateD の起動は gated を起動するのに十分なリソースが無いシステムで必要となります。同様に、`/etc/rc.local` に数行の追加が必要です。

```
if [ -f /etc/gated.conf -a -f /usr/local/bin/gdc ]; then
    /usr/local/bin/gdc [flags] start ; echo -n "gated(via gdc) " ;
fi
```

flags は **gdc** のオプションです。

5.1.4 GateD の制御

GateD は通常、**gdc** により制御されますが、直接制御することもできます。section 5.2 と section 5.3 でより詳しく説明しています。

5.1.5 GateD の状態確認

gdc を使用することにより、GateD のダンプを取得することができます。RIP や OSPF についても **ripquery** や **ospf_monitor** により、GateD に情報を要求することができます。section 5.5 と section 5.4 でより詳しく説明しています。

5.2 gated コマンド

5.2.1 コマンドラインオプション

gated [**-c**] [**-C**] [**-n**] [**-N**] [**-t** *trace_options*] [**-f** *config_file*] [*trace_file*]

コマンドラインのオプションは次のとおりです。

-c

文法エラーのチェックするために設定ファイルを読み込み、その後、gated は終了するように指定します。エラーが無い場合には gated は `/usr/tmp/gated_dump` というメモリダンプファイルを作成します。-c オプションを使用する場合は gated をスーパーユーザで起動する必要はありませんが、スーパーユーザで起動していない場合にはカーネル内のルーティングテーブルやネットワークインタフェースの設定状況を読み込むことができないかもしれません。-c オプションは *-tgeneral* と等価です。設定ファイル中の *traceoption* 文はすべて無視されます。

-C

文法エラーのチェックするために設定ファイルを読み込みます。gated はエラーがあれば 1 の終了ステータスで終了し、エラーが無ければ 0 の終了ステータスで終了します。-C オプションを使用する場合は gated はスーパーユーザで起動する必要はありませんが、スーパーユーザで起動していない場合にはカーネル内のルーティングテーブルやネットワークインタフェースの設定状況を読み込むことができないかもしれません。

-n

gated がカーネル内のルーティングテーブルを書き換えないように指定します。これは gated の設定を実際の経路情報でテストする場合に使用します。

-N

gated がデーモンとしては動作しないように指定します。通常、標準出力へのトレース出力が指定されていなくて親のプロセス ID が 1 でない場合に、gated はデーモンとして動作します。これによって `/etc/inittab` を使用するような方法で親の PID が 1 では無い gated の起動ができます。

-t *Trace_options*

コンマで区切られたトレースオプションのリストを起動時に有効となるように指定します。トレースオプションが何も指定されていない場合は、general が指定されたものと見なします。オプションリストやその引数の間にスペースがあってはいけません。

このオプションはネットワークインタフェースの設定状況やカーネルからルーティングテーブルを読み込むというような設定ファイルを解析する前に行われることをトレースするために使用しなければいけません。

有効なトレースオプションやトレースに関するより詳しい説明は section 4.5 を参照してください。

-f *config_file*

別の設定ファイルを使用する場合に指定します。デフォルトでは gated は `/etc/gated.conf` を使用します。

コマンドラインでトレースファイルを指定しているか、コマンドラインにトレースオプションが指定されていない場合は、gated は標準出力を使用しなくなり、バックグラウンドで動作します。トレースファイルの指定無しでトレースオプションが指定されている場合は gated は標準エラー出力にトレース結果を出力し、フォアグラウンドで動作します。

5.2.2 シグナル処理

gated を制御するために次のシグナルを使用することができます。

SIGHUP 設定ファイルを再度読み込みます。

SIGHUP によって **gated** は設定ファイルを再度読み込みます。この場合、**gated** は最初に設定に関して割り当てたすべての領域を一掃します。BGP や EGP で経路交換する peer に関するものはすべて削除されるようにマークされ、設定ファイルを再度読み込みます。

再読み込みが成功すると、EGP や BGP で経路交換する peer に関するものが設定ファイル中に無ければその peer との交換を停止し、新たに経路交換する相手との交換を開始します。**gated** は経路交換中の peer が変わった場合に交換の停止と開始が必要かどうかを判断しようとします。OSPF ではこのような再設定はできなくて、再設定中は交換を停止しその後再起動します。これは経路制御システムにとっては不利なことになり得ます。

これにより **gated** の再起動をしなくても経路制御プロトコルの使用開始や使用停止を行うことができます。

SIGINT 現在の状態をダンプします。

gated の現在の動作状況やタイマやプロトコルやテーブルの状態を`/usr/tmp/gated_dump` に書きます。

これは、*fork()* をサポートしているシステムでは **gated** の経路制御機能に影響を与えないようにテーブルをダンプするプロセスを fork して行います。メモリ管理でコピー時にライトしないシステムでは *fork()* は **gated** の使用しているアドレスを複製します。これはシステムに大きな影響を与えるかも知れません。*fork()* をサポートしていないシステムでは gated 自身が直ちにダンプを行います。そのため **gated** の経路制御機能に影響があるかもしれません。

SIGTERM gated を終了させます。

SIGTERM を受けると **gated** は終了しようとします。すべての動作とプロトコルが停止するようになります。大部分は直ちに終了しますが、確認を待つために EGP で経路交換する peer に関しては例外です。この処理に時間がかかる場合は **SIGTERM** を 1、2 回繰り返して送る必要があるかもしれません。

プロトコルによって設定された経路情報は **SIGTERM** を受けるとカーネルのルーティングテーブルから削除されます。ネットワークインタフェースに関するルーティング情報や (route コマンドがサポートしていれば付く) *RTF_STATIC* のフラグが付いた経路情報や *retain* が指定された静的経路情報はそのままです。経路情報をそのままにして **gated** を終了させる場合は、**SIGKILL** を使用して下さい。

SIGUSR1 トレースのスイッチです。

SIGUSR1 を受けると **gated** はトレース用のファイルをクローズします。その次の SIGUSR1 ではそのトレース用のファイルを再度オープンします。これによりトレースファイルの移動を正常に行うことができます。

トレースファイルを指定してなかったり、トレースを標準エラー出力に出力するようにしていた場合は、**SIGUSR1** を使用することはできません。

SIGUSR2 ネットワークインタフェースの状態を調べます。

SIGUSR2 を受けると **gated** はネットワークインタフェースの状態が変わっていないかどうかを調べます。

5.2.3 ファイル

以下に示すデフォルトのファイル名には `%s` という文字が含まれますが、これは gated が起動時に使用された名前に置き換えて下さい。通常、これは *gated* ですが、*gated-test* で起動された場合、*gated* はデフォルトで `/etc/gated-test.conf` というファイルを探します。このパス名はコンパイル時にすべて変えることができます。

`/usr/tmp/gated_dump` **gated** がダンプ情報を書き込むファイルです。デフォルトでは `/usr/tmp/%s_dump` です。別の同じパス名は `/var/tmp/%s_dump` です。

`/etc/gated.conf` **gated** が使用する設定ファイルです。デフォルトでは `/etc/%s.conf` です。

`/etc/gated.pid` **gated** のプロセス ID (PID) を書き込むファイルです。デフォルトでは、`/etc/%s.pid` ですが、`/var/run/%s.pid` と同じです。

5.3 gdc

5.3.1 形式

gdc [**-q**] [**-n**] [**-c** *coresize*] [**-f** *filesize*] [**-m** *datasize*] [**-s** *stacksize*] [**-t** *seconds*] *command*

gdc はユーザが *gated (8)* を操作するインタフェースを提供します。このコマンドでは gated の起動と停止を行ったり、gated が動作中に gated を操作するためにシグナルを送ったり、設定ファイルのメンテナンスや文法チェックをしたり gated の状態のダンプや core を作成、削除することができます。

gdc は *gated* を操作するシェルスクリプトを使用するよりも便利で、*gated* の動作状態を確実に決めることができ、エラーが起きた場合に信頼できる終了ステータスを出力します。gated で audit trail を実行していれば、**gdc** によって行ったコマンドやその実行により生じたエラーメッセージは *gated* が使用している *syslogd (8)* 経由でログを取ります。

root の setuid ビットを立ててインストールされていると同じグループ (デフォルトでは **gdmaint** です) の root で無いユーザが gated を操作でき、それ以外のグループの場合には操作ができないようになっています。audit のために、コマンドが起動される毎に *syslogd (8)* にユーザ名が記録されます。

コマンドラインのオプションは以下の通りです。

-n

カーネルのルーティングテーブルを変更せずに gated を起動します。これはテストやパケットのフォワードをしない経路情報のサーバとして動作する場合に便利です。

-q

実行時の表示出力をしません。このオプションにより、通常標準出力に出力されるメッセージは出なくなり、エラーメッセージは標準エラー出力に出力される代わりに *syslogd (8)* 経由で記録されます。これは **gdc** がシェルスクリプトから実行される場合に便利です。

-t *seconds*

gdc が *gated* に指定した操作 (特に終了や起動) が終了するまでにどれだけ待つのかを秒で指定します。デフォルトでは 10 秒です。

以下のコマンドラインオプションは **gdc** のコンパイル時に使用されたオプションにより使用可能です。

-c *coresize*

gated が **gdc** から起動された場合の core の最大の大きさを指定します。これはデフォルトの最大 core サイズが *gated* がエラー時に作成する core よりもはるかに小さいようなシステムで便利です。

-f *filesize*

gated が **gdc** から起動された場合のダンプファイルの最大の大きさを指定します。これはデフォルトの最大のダンプサイズが *gated* が作成要求を受けて作成するダンプよりもはるかに小さいようなシステムで便利です。

-m *datasize*

gated が **gdc** から起動された場合のデータセグメントの最大の大きさを指定します。これはデフォルトの最大のデータセグメントサイズが *gated* が使用するデータセグメントサイズよりはるかに小さいようなシステムで便利です。

-s *stacksize*

gated が **gdc** から起動された場合のスタックの最大の大きさを指定します。これはデフォルトの最大のスタックの大きさが *gated* が使用するスタックサイズよりはるかに小さいようなシステムで便利です。

以下のコマンドによって、さまざまな目的のために *gated* にシグナルを送ります。

COREDUMP

gated に停止シグナルを送り、core を生成して終了させます。

dump

gated に現在の状態を `/usr/tmp/gated_dump` にダンプさせるシグナルを送ります。

interface

gated にネットワークインタフェースの状態が変わっていないかどうかを調べに行くようにシグナルを送ります。*gated* は通常、いつも周期的にこれを行いますが、ネットワークインタフェースの状態が変わったことをすぐに知るためにこの機能によって、gated にネットワークインタフェースの状態を調べに行かせることができます。

KILL

gated を強制終了させます。通常は gated がハングアップした場合に使用します。

reconfig

gated に設定ファイルを読み込みに行くようにシグナルを送り、適宜現在の状態を再設定します。

term

gated に経路制御プロトコルの処理を終了させて停止するようにシグナルを送ります。このコマンドを 2 回行うと、*gated* は経路制御プロトコルの処理が終了していなくても停止します。

toggletrace

gated がトレースをファイルに取っている場合は、トレースを中止し、トレースファイルをクローズします。*gated* のトレースが止まっている場合は、トレースファイルを再びオープンし、トレースを開始します。これはトレースファイルを移動する場合に便利です。

デフォルトでは *gated* は通常 `/etc/gated.conf` というファイルからその設定を得ます。*gdc* はこの他にもいくつか設定ファイルを保持していて、以下にそれを示します。

`/etc/gated.conf+` 新しい設定ファイルです。**gdc** に新しい設定ファイルで設定するようにした場合は、このファイルは `/etc/gated.conf` に変更されます。

`/etc/gated.conf-` 古い設定ファイルです。**gdc** に新しい設定ファイルで設定するようにした場合は、前の `/etc/gated.conf` がこの名前に変更されます。

`/etc/gated.conf--` さらに古い設定ファイルです。**gdc** は前の古い設定ファイルをこの名前に変更します。

以下のコマンドは設定ファイルに関する操作のものです。

checkconf `/etc/gated.conf` の文法エラーをチェックします。これは設定ファイルを変更後、動作中の *gated* に **reconfig** のシグナルを送る前に行うと便利です。これにより、動作中の *gated* が再設定により停止しないように設定ファイル中にエラーが無いことが確認します。このコマンドが使用された場合は、**gdc** は解析エラーがあるかどうかのメッセージを表示し、エラーがある場合にはそれを調べることができるようにファイルにエラーを出力します。

checknew 新しい設定ファイル (`/etc/gated.conf+`) をチェックする以外は **checkconf** と同じです。

newconf 古い設定ファイルは先に示したように保持しながら、`/etc/gated.conf+` を `/etc/gated.conf` にします。新しい設定ファイルが無かったり、正しくないようであれば **gdc** は何もしません。

backout 設定ファイルを新しいものに移し、古い設定ファイルを`/etc/gated.conf` に移動します。このコマンドは `/etc/gated.conf-` が無かったり、あっても大きさが 0 であったり、0 でない大きさの `/etc/gated.conf+` を削除することになる場合には何もしません。

BACKOUT `/etc/gated.conf+` があって、0 でない大きさの場合でも **backout** を行います。

modeconf すべての設定ファイルのモードを 644、オーナを root、グループを gdmaint とします。これにより root 以外のユーザが設定ファイルを変更することができます。

createconf `/etc/gated.conf+` が無い場合に大きさが 0、モードが 644、オーナが root グループが gdmaint のものを作成します。これにより root 以外のユーザが新しい設定ファイルを設定することができます。

以下のコマンドで *gated* の起動および停止や動作状態を決めることができます。

running *gated* が動作中かどうかを調べます。これは、*gated* がその pid を中身とするロックファイルがあるかどうか、そのファイル中の pid が正しいかどうかやその pid で動作中のプロセスがあるかどうかを調べることによって行います。*gated* が動作中ならば 0 の終了ステータスで終了し、それ以外は 0 以外の値で終了します。

start *gated* を起動します。すでに *gated* が動作中の場合はエラーを返します。そうでなければ、*gated* を実行し、新たに起動したプロセスが pid ファイルからロックを得るまで遅延間隔 (デフォルトでは 10 秒で、**-t** で指定できます) の間待ちます。*gated* を実行中にエラーを検出したり、指定待機時間内に pid ファイルからロックを得られなかった場合は 0 以外のエラーステータスが返ります。

stop 可能な限り正常に *gated* を停止します、そうでなければ強制的に停止します。*gated* が動作中で無い場合は (0 以外の終了ステータスで) エラーを返します。そうでない場合は *gated* に停止シグナルを送り、プロセスが終了するまでの間遅延間隔 (デフォルトでは 10 秒で、**-t** オプションで指定できます) になるまで待ちます。*gated* が遅延間隔内の終了に失敗した場合は 2 回目には terminate シグナルを送る必要があります。*gated* が 2 回目の遅延間隔内の終了に失敗した場合は 3 回目には kill シグナルを送る必要があります。何も異常が無い場合はすぐに終了するはずです。このコマンドは *gated* が終了したことが分かった時点で 0 の終了ステータスで終了し、それ以外の場合は 0 以外の終了ステータスを返します。

restart *gated* が動作中の場合は上記 **stop** コマンドで使用したのと同じ処理で停止します。*gated* が停止した時点、あるいはこのコマンド起動時に *gated* が動作していなければ、上記 **start** コマンドで使用したのと同じ処理で新たに *gated* が起動されます。いずれかの処理で失敗とみなされるようなことが生じると 0 以外の終了ステータスを返します。

以下のコマンドでは上記のコマンドを実行することにより作成されたファイルを削除することができます。

rmcore *gated* の core を削除します。

rmdump *gated* のダンプを削除します。

rmparse **checkconf** や **checknew** コマンドにより設定ファイルをチェックして文法エラーが見つかった時に作成された構文解析エラーファイルを削除します。

5.3.2 関連ファイル

以下に示したデフォルトのファイル名には `%s` という文字が含まれますが、これは gated が起動時に使用された名前に置き換えて下さい。通常は **gated** ですが、*gated-test* で起動された場合、**gated** はデフォルトで `/etc/gated-test.conf` というファイルを探します。このパス名はコンパイル時にすべて変えることができます。

`/etc/gated` *gated* の実行ファイルです。よく使用される別の場所は `/usr/local/sbin/gated` です。

`/etc/gated.conf` *gated* の設定ファイルです。

`/etc/gated.conf+` 新しい設定ファイルです。

`/etc/gated.conf-` 古い設定ファイルです。

`/etc/gated.conf--` 最も古い設定ファイルです。

`/etc/gated.pid` *gated* のプロセス PID を書き込むファイルで、デフォルトでは、`/etc/%s.pid` です。よく使用される別の場所は `/var/run/%s.pid` です。

`/usr/tmp/gated_dump` `gated` がダンプ情報を書き込むファイルで、デフォルトでは`/usr/tmp/%s_dump` です。よく使用される別の場所は `/var/tmp/%s_pid` です。

`/usr/tmp/gated_parse` 設定ファイルの解析エラーを格納するファイルで、デフォルトでは`/usr/tmp/%s_parse` です。よく使用される別の場所は `/var/tmp/%s_parse` です。

`/usr/tmp` *gated* が core ファイルを作成するディレクトリです。よく使用される別の場所は `/var/tmp` です。core ファイル名には通常 `core` が使用されますが、`core.gated` を使用するシステムもあります。

5.3.3 注意

ここで説明した多くのコマンドは *gated* が設定されたシステムディレクトリにインストールされている場合にのみ動作します。

gdc にシステムの core ファイル名に `core` (`core.gated` はそれほど共通ではありません) 以外の名前を伝える方法はまだありません。

5.4 ospf_monitor

5.4.1 形式

ospf_monitor *mon_db_file*

OSPF ルータに問い合わせるために **ospf_monitor** コマンドを使用します。**ospf_monitor** コマンドは対話モードで動作します。これにより、いろいろな OSPF ルータにパケットの入出力の統計、エラー、link-state データ、AS external データ、OSPF のもつ経路情報、OSPF を使用するネットワークインタフェース、OSPF の neighbors などに関する詳細な情報を問い合わせることができます。

mon_db_file は **ospf_monitor** のリモートコマンドを行う相手を指定したデータベースの絶対パス名です。各相手へのレコードは相手の IP アドレス、相手のホスト名や (相手の認証が有効になっていれば) OSPF の認証キーを並べた一行のエントリです。相手のレコードに認証キーを入れることができるため、このデータベースへの一般のアクセスは制限することを勧めます。

OSPF のデータベースとそのパケットフォーマットの詳細は [RFC 1583] (OSPF Specification, version 2) を参照して下さい。

5.4.2 コマンド

対話モードに入ると、**ospf_monitor** は `'[ # ] dest command params >'` というプロンプトを表示し、その状態では **ospf_monitor** のどの対話コマンドも使用することができます。対話コマンドはいつでもキーボード割り込みで割り込むことができます。コマンドラインの文字数は 200 文字以下でなければならないことに注意して下さい。

ローカルコマンド

? すべてのローカルコマンドとその機能を表示します。

?R すべてのリモートコマンドとその機能を表示します。

d 指定したすべての相手を表示します。このコマンドでは *mon_db_file* で指定した **ospf_monitor** コマンドのすべて相手の *dest_index*、IP アドレスやホスト名を表示します。

h 過去 30 回までに入力した対話コマンドのコマンド履歴を表示します。

x **ospf_monitor** を終了します。

@ *remote_command* 同じ (前の) 相手に *remote_command* を送ります。

@*dest_index remote_command* *dest_index* で指定した相手に *remote_command* を送ります。

F *filename* **ospf_monitor** の結果を *filename* に出力します。

S **ospf_monitor** の結果を標準出力に出力します。

リモートコマンド

a *area_id type ls_id adv_rtr* LSA を表示します。*area_id* は問い合わせを行う destination OSPF エリアです。*adv_rtr* は この LSA を出すルータのルータ ID です。*type* は要求する LSA のタイプで以下を指定することになります。

1 ルータリンク通知を要求します。この通知はルータのネットワークインタフェースを表します。このタイプの要求を行うためには、*ls_id* にはこの LSA の発信元のルータ ID を指定しなければいけません。

2 ネットワークリンク通知を要求します。この通知はルータが属しているネットワークを表します。このタイプの要求を行うためには、*ls_id* には指定ルータのネットワークインタフェースの IP アドレスを指定しなければいけません。

3 ネットワークへの経路を表すサマリーリンク通知を要求します。この通知はエリア間の経路情報を表し、エリア境界で経路情報をまとめることができます。このタイプの要求を行うためには、*ls_id* には destination ネットワークの IP アドレスを指定しなければいけません。

4 AS 境界ルータへの経路を表すサマリーリンク通知を要求します。この通知はエリア間の経路情報を表し、エリア境界で経路情報をまとめることができます。このタイプの要求を行うためには、*ls_id* には AS 境界ルータのルータ ID を指定しなければいけません。

5 AS 外部リンク通知を要求します。この通知は AS の外部への経路情報を表します。このタイプの要求を行うためには、*ls_id* には destination ネットワークの IP アドレスを指定しなければいけません。

c 累積ログを表示します。このログには、モニタ要求、HELLO、data base description、link-state 要求、link-state 更新および link-state 確認の各パケットの入出力の統計が含まれています。エリアの統計では、neighbor の数と OSPF が使用しているネットワークインタフェースの数の合計を表示します。経路情報テーブルの統計はエリア内の経路情報、エリア間の経路情報および AS 外部への経路情報をまとめて表示されます。

e 累積エラーを表示します。このログは OSPF の neighbor 間で発生したいろいろなエラーを表示し、それぞれ何回発生したかも表示します。

h 次 hop のリストを表示します。これは主に SPF により求めた有効な次 hop のリストを表示します。

l [*retrans*] (ASE を除く) link-state データベースを表示します。このデータベースは AS を構成するルータとネットワークを表します。*retrans* が 0 でない場合、各 link-state データベースが持つ再送した neighbor のリストも表示されます。

A [*retrans*] AS 外部の link-state データベースを表示します。各 AS 外部への経路情報について通知ルータ、フォワードするアドレス、LS age、長さ、 LS シーケンス番号、経路情報のタイプ、メトリックを表示します。*retrans* が 0 でない場合、各 link-state データベースが持つ再送した neighbor のリストも表示されます。

o [*which*] OSPF の経路情報テーブルを表示します。このテーブルでは OSPF で管理している AS 境界ルータへの経路情報、エリア境界ルータへの経路情報、AS 境界ルータサマリー、ネットワーク、ネットワークサマリーおよび AS 外部ネットワークを表示します。*which* を省略すると上記すべてが表示されます。指定する場合は、*which* の値 (1 から 63 の値) により表示するテーブルを指定します。これは以下で示すテーブルの値の和を指定することによりテーブルを指定します。

1 エリア内の AS 境界ルータへの経路情報

2 エリア内にあるエリア境界ルータへの経路情報

4 別のエリアにある AS 境界ルータサマリー

8 エリア内にあるネットワークへの経路情報

16 別のエリアにあるネットワークサマリー

32 OSPF を使用していないネットワークへの AS 外部ネットワーク

I すべてのネットワークインタフェースを表示します。OSPF が使用するネットワークインタフェースを表示します。これにはエリア、ネットワークインタフェースの IP アドレス、ネットワークインタフェースのタイプ、ネットワークインタフェースの状態、コスト、優先順位およびそのネットワークの DR と BDR の IP アドレスが含まれています。

N OSPF のすべての neighbor を表示します。これにはエリア、ローカルネットワークインタフェースのアドレス、ルータ ID、neighbor の IP アドレス、状態およびモードが含まれています。

V GateD のバージョン情報を表示します。

5.5 ripquery

5.5.1 形式

ripquery [**-1**] [**-2**] [**-** [**a5**] *authkey*] [**-n**] [**-N** *dest* [**/***mask*] [**-p**] [**-r**] [**-v**] [**-w** *time*] *gateway ...*

ripquery は RIP ゲートウェイが知っているすべての経路情報を RIP の REQUEST か POLL コマンドを送信して要求するために使用します。送り返された経路情報はアドレスと名前で表示されます。**ripquery** はネットワーク管理のためでは無く、ゲートウェイのデバッグ用のツールとして使用されます。ネットワーク管理のプロトコルとしては SNMP の方が適しています。

ripquery はデフォルトでは RIP の POLL コマンドを使用し、*SunOS 3.x* 上の *routed* やそれ以降のものや *gated* 1.4 以降でサポートしていますが、どこにも記述されていない RIP の拡張です。RIP の REQUEST コマンドよりも RIP の POLL コマンドの方が適しています。これは Split Horizon や Poisoned Reverse に関係しないからです。詳しくは [RFC 1058] や [RFC 1723] を参照して下さい。

5.5.2 オプション

-1

RIP バージョン 1 で要求パケットを送信します。

-2

RIP バージョン 2 で要求パケットを送信します (デフォルト)。

-[**a5**] *authkey*

要求時に使用する認証パスワードを指定します。*-a* を指定した場合は、SIMPLE の認証タイプが使用され、*-5* を指定した場合は、MD5 の認証タイプが使用され、そうでなければデフォルトの NONE の認証タイプが使用されます。入力パケットの認証フィールドは表示されますが、チェックはされません。

-n

応答ホストのアドレスから名前を参照しないように指定します。

-N *dest* [/*mask*]

すべての経路情報ではなく、dest/mask で指定したものについての経路情報を要求します。オプションの mask を指定するとバージョン 2 の要求を行うことになります。一つの要求で 23 個まで指定することができます。

-p

経路情報を要求するのに RIP の POLL コマンドを使用します。これはデフォルトですが、*SunOS 3.x* 以降のバージョンや最近の *gated* でしかサポートされていない、どこにも記述されていない拡張です。RIP POLL コマンドで応答が無い場合は、RIP REQUEST コマンドを使用します。*GateD* は POLL コマンドには RIP から得たすべての経路情報を返します。

-r

経路情報を要求するのに RIP の REQUEST コマンドを使用します。RIP の POLL コマンドとは異なり、すべてのゲートウェイは RIP の REQUEST をサポートしています。RIP REQUEST コマンドで応答が無い場合は、RIP POLL コマンドを使用します。*gated* は REQUEST コマンドにはそのネットワークインタフェースへ送出したすべての経路情報を返します。UDP インタフェースの制限により、*4.3BSD Reno* かそれ以前のシステムでは REQUEST には送信するネットワークインタフェースから送出する経路情報を返します。これは要求されたホスト上で **ripquery** を動かすことによって回避できます。

-v

ゲートウェイに要求する前に、**ripquery** のバージョンを表示します。

-w *time*

ゲートウェイからの最初の応答を待つ時間を秒数で指定します。デフォルトは 5 秒です。

5.6 Gated Interactive Interface

Gated Interactive Interface (GII) では起動中の gated へのインタラクティブなインタフェースを提供し、gated の内部変数を調べるために使用することができます。このインタフェースは gated での他のプロトコルと同様に実装されていて、ポート 616 への *telnet* コネクション ([RFC 854]) を受け入れ、ユーザ認証の後、ASCII 文字で送られたさまざまなコマンドに応答します。コマンドには、メモリ、経路情報、ネットワークインタフェースリストやその他の内部パラメータについての問い合わせとなっています。

5.6.1 設定

GII ではシステム上の *gii* という ID の UNIX のパスワードを使用してユーザを認識します。すなわち、GII インタフェースを使用する場合、'gii' というアカウントを作成し、GII へ接続する際に使用するパスワードを設定しなければいけません。

5.6.2 GII への接続

利用者は gated が起動しているマシンの 616 の TCP ポートに telnet コネクションを行います。利用者が簡単なパスワード方式を利用して利用者の認識をおこなうと、**gated** のプロンプトがあらわれます。このようにして GII が要求を受け付ける準備が整います。以下にセッションの例を示します。

```
(lpj@obelix: 53) telnet radb2.ra.net 616
Trying 198.108.0.9...
Connected to radb2.ra.net.
Escape character is '^]'.
Password?
100 Gated Interactive Interface. Version R3_5Alpha_10
GateD> sh ip
100 HELP: The possible subcommands are:
100     route [x.x.x/len]: Show info about IP routes
100     walkup x.x.x/len: Show less specific routes
100     walkdown x.x.x/len: Show more specific routes
GateD>
```

5.6.3 ヘルプ

トップレベルのコマンドの **help** によって使用できるコマンドのリストが得られます。コマンドにはサブコマンドをつづけることもできます。サブコマンドについてのヘルプを得るためには、サブコマンド無しでメインコマンドを入力します。これにより GII は使用可能なサブコマンドの一覧を表示します。コマンド名を短縮して同じ名前にならない限り、短縮することができます。

5.6.4 gated 内部

以下のコマンドは gated の内部構造に関するパラメータを表示します。

show version 起動中の gated のバージョンを表示します。

show kernel (reject route、マルチキャストや UDP のチェックサム等) どのような機能がサポートされているかを含むカーネルのタイプを表示します。

show memory メモリブロックごとのメモリの使用状況を表示します。

show task gated で動作している処理を表示します。

5.6.5 ネットワークインタフェース

以下のコマンドはネットワークインタフェースに関するパラメータを表示します。

show interface [name | index] パラメータ無しでこのコマンドを使用すると、システム上のすべてのネットワークインタフェースを表示します。name あるいは index を引数として指定するとこのネットワークインタフェースに関するすべてのパラメータが表示されます。

5.6.6 ルーティングテーブル

以下のコマンドはルーティングテーブルに関するパラメータを表示します。

show ip route [x.x.x / len] パラメータ無しでこのコマンドを使用すると、IP のルーティングテーブルの大きさを表示します。引数は IP アドレスのプリフィックスとネットマスク長を指定しなければいけませんが、指定された経路情報がアナウンスされた回数、次 hop、AS 経路や有効な経路情報等の情報を表示します。

show ip walkup x.x.x / len *x.x.x/len* より一般的な、すなわち *x.x.x/len* を含む集約経路のような経路情報をすべて表示します。

show ip walkdown x.x.x / len *x.x.x/len* より詳細な、すなわち集約経路 *x.x.x/len* の要因となる経路情報をすべて表示します。**show ip walkdown 0/0** はすべての経路情報を表示します。**^c** で表示を途中で停止させることができます。

いくつか例を示します。

```
GateD> sh ip walkdown 0/0
100 Sta         0.0.0.0/0  192.231.253.2   IGP (Id 1)
100 Sta             127/8  127.0.0.1       IGP (Id 1)
100 RIP         141.211/16 192.231.253.2   IGP (Id 1)
100 RIP         141.212/16 192.231.253.2   IGP (Id 1)
100 RIP         141.213/16 192.231.253.2   IGP (Id 1)
100 RIP         141.214/16 192.231.253.2   IGP (Id 1)
100 Dir     192.231.253/24 192.231.253.50  IGP (Id 1)
100 Dir         198.108/24 198.108.0.9     IGP (Id 1)
100 RIP      198.108.60/24 192.231.253.2   IGP (Id 1)
GateD>
GateD> sh ip walkdown 141/8
100 RIP         141.211/16 192.231.253.2   IGP (Id 1)
100 RIP         141.212/16 192.231.253.2   IGP (Id 1)
100 RIP         141.213/16 192.231.253.2   IGP (Id 1)
100 RIP         141.214/16 192.231.253.2   IGP (Id 1)
GateD>
GateD> sh ip walkup 192.231.253/24
100 Dir     192.231.253/24 192.231.253.50  IGP (Id 1)
100 Sta         0.0.0.0/0  192.231.253.2   IGP (Id 1)
GateD>
GateD> sh ip ro 198.108.60/24
100 Route 198.108.60 - 255.255.255 entries 1 Announced 1 Depth 0 <>
100 * RIP 192.231.253.2   192.231.253.2   4/0     0   10:25:13 IGP (Id 1)
```

5.6.7 DVMRP

show dvmrp mfc 定義されているマルチキャストのフォワーディングキャッシュを表示します。

show dvmrp targets すべての DVMRP の destination (ネットワークインタフェース) を表示します。

5.6.8 その他

quit GII セッションを終了します。

5.6.9 OSPF

以下のコマンドは **gated** のバージョン **3_6Alpha_2** あるいはそれ以上で導入された **OSPF** でのみ使用することができます。

show ospf global ルータ ID、preference やフラグのような OSPF の一般的なパラメータを表示します。

show ospf interface [**x.x.x.x**] パラメータ無しでは OSPF を使用するネットワークインタフェースのリストやそのパラメータ (interval や状態等) を表示します。パラメータとして IP アドレスを指定すると、そのネットワークインタフェースでの neighbor のリストとそのパラメータと状態を表示します。

show ospf area [**x.x.x.x**] パラメータ無しではそのルータ上で設定されているエリアのリストを表示します。パラメータとしてエリア番号 (エリア番号は IP アドレスと同じフォーマットです) を指定すると、link-state のデータベースのフルダンプと同時にそのエリア向けのネットワークインタフェースを表示します。

Chapter 6

トラブルシューティング

- 設定ファイルではデフォルトの経路情報をしていますが、*IGP* で *export* することができません。

  export statement を使用して明示的に **igp** がデフォルトの経路情報を export するように指定してください。例えば、**igp** に RIP を使用する場合は、設定ファイルは次のようにするべきです。

  ```
  export proto rip {
          proto static {
                  default;
          };
  };
  ```

- *AS* 内部がクラス *C* のネットワークをサブネット化されています。*IGP* や *EGP* ではクラス *C* のネットワークを *export* しないようにしていますが、個々のサブネットワークのみ *export* します。

  igp や **egp** が (RIP 2, BGPv4 のような) クラスレスなプロトコルである場合、明示的にそのように指定しない限り、プロトコルではクラス C のネットワークをアナウンスしようとはしません。**aggregate** statement と **export** statement を組み合わせて使用するとアナウンスさせることができます。詳細な設定例は section 4.29.3 を参照してください。

- *SunOS 4* のマルチキャストパッチをインストールしたのですが、*gated* はネットワークインタフェースがマルチキャストできないものと判断しているようです *(*あるいは *RIP* や *OSPF* をマルチキャストモードで起動できません*)*。

  この問題としては、gated を生成するために gcc を使用していると、C のヘッダファイルに正しくパッチがあたっていないためです。gcc は `/usr/include` の代わりに `/usr/local/lib/gcc/<architecture>/<version>/include` をヘッダに使用しているため、マルチキャストのためのパッチがあたっていないのです。そのため、gated はどのネットワークインタフェースもマルチキャストできないものと判断してしまいます。

  解決策としては `/usr/include` から `/usr/local/lib/gcc/<architecture>/<version>/include` にパッチをあてたヘッダをコピーすることです。

- *gated* の起動時にログファイルに *'VARIABLE MASKS NOT SUPPORTED'* という奇妙なメッセージが出ます。

  これはカーネルで可変長ネットワークマスク (CIDR とも言われます。[RFC 1518] を参照してください) をサポートしていないことを意味します。例えば、SunOS4.1 ではサポートしていません。クラスの IP アドレス (クラス A、B そして C のネットワーク、RFC 791 を参照してください) や RFC791 でのサブネットを使用する限りは問題は無く、無視してください。CIDR を利用したい場合、他の OS を使用する必要があります。これはカーネルの問題であって、gated の問題ではありません。

- コンパイル時に *C* コンパイラが *'objects.h'* と *'smux.h'* が無いためにエラーになります。

  ISODE SMUX をサポート(すなわち gated に SNMP/SMUX により情報を得ることができます)するように gated を生成しようとしているためです。このためにはまず ISODE パッケージを最初にインストールする必要があります(詳細は INSTALL を参照してください)。SNMP を必要としない場合、Config ファイルの protocols リストから 'isode_snmp' を削除してください(gated で配布されている Config ファイルのいくつかはデフォルトで SNMP が指定されています)。

- *gated* が起動後数分間は正常に動作していますが、その後 **if_rtdown: DOWN route for interface xxx0 1.2.3.4/255** というようなエラーを出力します。

  これは GateD で IGP が動作しているネットワークインタフェースで(ネットワークインタフェースが正常に動作していないか、そのセグメント上に他にルータが存在していなくて) IGP のパケットを受け取らなかったために、タイムアウトがおきてしまったためである可能性があります。この問題の解決策としては設定ファイルの interface statement で **passive** の属性を使用してネットワークインタフェースを *passive* で定義することです。例えば、次のようにします。

```
interfaces {
        interface xxx0 passive;
};
```

Chapter 7

開発者のための手引

7.1 概要

ルーティングソフトウェアの中でも、GateD は移植可能なルーティングプロトコルの開発プラットフォームで、ほとんどのプロトコル実装に共通な機能を分業して (そして個々に提供して) います。この機能の分業には次の三つの重要な利点があります。一つ目はルーティングプロトコルの実装での処理を単純化できます。二つ目は複数の (すなわち経路情報の交換に加わっている) プロトコルが同一ルータ上で動作している場合、プロトコル処理とメモリ資源のつりあいのとれた共有が可能です。三つ目は移植性を高めるためにプロトコル実装でのシステム依存の箇所を分離できます。

GateD は (安定した状態で) 次のような基本的な機能から構成されているため、理論的には、人的な設定の操作が可能なルータとなります。この機能とは (ある順序で組み合わせて)、neighbor からのプロトコルメッセージの受信と、その処理と、forwarding information base (FIB) の更新や neighbor へのプロトコルメッセージの送信です。

このモデルをもとに、すべてのルーティングプロトコルの実装に沿った、必要な以下の一般的な機構を抽出することができます。

- プロトコル処理のスケジューリング機構
- メモリ管理機構
- 情報交換管理機構
- 経路情報保持機構
- 設定と再設定

これらの機構は (UNIX のような) これまでの OS でも提供されています。しかし、基礎的な OS の機構ではルーティングプロトコルの実装には不十分となり得るのです。

このため GateD では内部的に高く最適化したこのような機構を実装しています。GateD ではもともと BSD ベースのシステムでのルーティングデーモンとしてシステム内で複数のプロトコルを動作させるように作られています。GateD の共通ルーチンを使用しながら、これらのプロトコル実装箇所ではより有効的にプロセッサやメモリ資源を共有するように作られています。

次に個々の機構の概要を説明します。

7.1.1 プロトコル処理のスケジューリング機構

GateD 単体で複数のルーティングプロトコルを動作させることができます。そのため、(例えば BGP や OSPF のように) 異なるルーティングプロトコル処理のための資源を確実に確保することができなければいけません。この目的のために、GateD は *task*、*timer* や *job* の機能を持っています。

task は (メッセージの読み込みとその処理のように) 一連の関連した基本的な機能の実行の流れです。このように関連した機能は他のプロトコルの機能や同じプロトコル内での他の機能とは独立して (すなわち任意に) スケジューリングすることができます。

プロトコルの実装毎に task をそれぞれで使用します。task を使用する最も簡単な方法は、プロトコル内で発生するすべての処理のために単一の task を確保することです。個々の peer とのコネクションを保持する BGP のように peer に関連したプロトコル処理に task を割り当てるプロトコルもあります。

GateD の **timer** ではプロトコルでの周期的あるいは時期をずらした処理を行います。このような処理例には、peer の状態のモニタや BGP での keep-alive メッセージの送信があります。

GateD の **job** ではプロトコルでの都合の良いタイミングで処理をおこなうようにスケジューリングすることができます。例えば、OSPF の実装では job を利用して SPF 処理のスケジュールを行います。

7.1.2 メモリ管理機構

GateD では必要なメモリの獲得のために OS のメモリ割り当て機構を使用します。しかし、(例えば、task やプロトコル制御などの) 個々の処理のために自分自身の高く最適化した割り当て機構も使用します。

7.1.3 情報交換機構

ルーティングプロトコルメッセージの (unreliable なデータグラムや reliable なバイトストリーム型のコネクションなどの) トランスポートプロトコルとしてはほとんどいつも OS のインタフェースを利用します。初期のころのプロトコル実装では GateD の unreliable なデータグラムトランスポートを使用していました。

プロトコルの実装を簡単にするために、GateD では物理的なネットワークインタフェースと neighbor に概念的なデータタイプを提供しています。このようなデータタイプではプロトコル毎に使用されるこれらの物理的な情報をまとめて保持しています。

7.1.4 経路情報保持機構

ルーティングプロトコルの実装でのもうひとつの共通機構は、経路情報の保持と処理そしてルータの forwarding information base の更新です。GateD では集約したルーティングデータベースを提供していて、ルータ上で起動しているプロトコルが到達性の情報を保持しています。この機能を提供しているモジュールではカーネルのルーティングテーブルも直接更新します。

BGP での AS 経路の保持と処理を有効におこなうための GateD の経路モジュールもここに含められます。このモジュールでは BGP の AS 経路を容易に処理できる表記機能も提供しています。

7.1.5 設定機構

最後に、GateD ではほとんどのプロトコル実装に共通な設定情報を集約する機構を提供しています。この機構ではプロトコルが動作している AS 番号やポリシ指定というようなことを集約しています。

7.2 コンパイル設定処理

7.2.1 構成

GateD の生成処理では同じソースツリーを使用して異なるアーキテクチャの gated のコンパイルが可能です。ソースファイルをアーキテクチャ依存のオブジェクトファイルにコンパイルするために、個々のシステムで `src` ディレクトリのサブディレクトリを使用することができます。

さらに、コンパイルされるべきファイルやその内容は `Config` ファイル中での protocols に記述されたリストによって変わります。

'make config' により、GateD の生成処理で使用するファイルの生成と同様に、GateD の生成に必要な一連のファイルオブジェクトディレクトリ配下の設定を行います。

7.2.2 *make config* 処理

'make config' 処理では次のようなファイルを生成します。

- `obj/defines.h` は `#define` 文から構成され、すべての C のソースコードに include されます。`#define` の値は **protocol** 文と同様に `Config` ファイルの **options** 文から生成されます。
- `obj/paths.h` には、設定ファイル、ダンプファイルのような *GateD* が使用するファイルのパス名が入っています。これは `src/paths.h` を置換して生成されます (section 7.2.4 参照)。
- `obj/version.c` には *GateD* のリリースバージョンと同様にソースファイル毎にバージョン番号が入っています。バージョン番号は文字定数に代入され、*GateD* のバイナリにハードコーディングされます。
- `obj/parser.y` は *GateD* で設定すると選択したプロトコルで使用する parser を生成するための本物の *yacc* のソースファイルです。これは `src/parser.y` を置換して生成されます (section 7.2.4 参照)。
- `obj/signames.c` では `sys_signame[]` という C の変数をシステムがサポートしていなければ宣言します。`Config` ファイルで NEED_SIGNAME オプションが定義されている場合にのみ `signames.c` は `<signal.h>` から生成されます。
- `obj/Makefile` は *GateD* を生成するために使用する **make** のルールファイルです。これは `util/Makefile.temp` を置換して生成され (section 7.2.4 参照)、設定を指定したソースファイルのみについてのルールを含みます。
- `obj/*.c obj/*.h obj/*.y obj/*.l` は `src/` 配下のソースファイルへのシンボリックリンクです。

上記処理で必要なファイルは以下のとおりです。

- `util/configure.pl` は設定をおこなう **perl** スクリプトです。
- `util/Makefile.templ` Makefile のテンプレートです。
- `util/files` はコンパイルすべきファイルのリストです。このリストはプロトコルごとに分類されています。
- `parser.y` はすべてのプロトコルで使用する yacc の全ソースファイルです。
- `*.c *.h` は `src/` ディレクトリ配下のすべてのソースファイルです。

まとめると、`make config` を実行すると `util/configure.pl` が最初に起動されます。このスクリプトは次のことを行います。

- カレントディレクトリとそのシステムで使用するオブジェクトディレクトリを見つけます
- 先にリストしたすべてのファイルをオブジェクトディレクトリに生成します。

その後、`make config` をオブジェクトディレクトリで実行します。これにより次のことを行います。

- オブジェクトディレクトリを一掃します。
- **yacc** あるいは **bison** を `parser.y` に対して行い、`parser.c` と `parser.h` を生成します。
- シンボリックリンクを生成します。
- `mkdep` コマンドを実行してファイルの依存関係を調べます。

7.2.3 make でのターゲットリスト

オブジェクトディレクトリでの *Makefile* では実行プログラムが生成されるために次のようなターゲットを定義します。それ以外は、ケースバイケースで追加することができます。

- *all::* `make` を実行するときのデフォルトのターゲットです。
- *lint::* **lint(1)** でソースコードのチェックをおこなう場合に使用します。
- *install::* 実行プログラムをインストールする場合に使用します。
- *install-man::* man page やドキュメントをインストールする場合に使用します。
- *clean::* ディレクトリ中のオブジェクトファイルを一掃する場合に使用します。
- *cleanall:: clean::* よりも好ましいです。*clean::* ではオブジェクトファイルのみ削除します。*cleanall::* では配布パッケージでもともと存在しているファイルをすべて削除します。

7.2.4 ファイルのフィルタ

配布パッケージ中のファイルのうちいくつかは、フィルタを通して実行されるようになっていて、指定した protocols のみの gated を得るようになっています。例えば、**rip** と **bgp** のみ必要であれば、そのためのファイルは RIP と BGP のみのソースコードを含むものになり、その結果最終的なバイナリはより小さくなります。フィルタをおこなうための一部のコードの一般的な文法は C 言語と似ています。文法表記は次のとおりです。

@BEGIN [**NOT**] *SYMBOL* [| *SYMBOL2*...]
        text...
@END

@BEGIN と **@END** の間のテキストは、*SYMBOL*(あるいは *SYMBOL2* 等) が定義されているとそのままになります。このシンボルは Config ファイルで定義された変数、プロトコルあるいはオプションのいずれかです。これは (**cc**、**lex** 等の) プログラム名やディレクトリあるいは (**path_config**、**mandir** 等の) ファイル名や (**NEED_STRERROR** や **POSIX_SIGNALS** 等の)) オプションや (**bgp** や **rip** 等の) プロトコル名からなります。**@BEGIN** の後に付けるシンボルは **bgp** なら **PROTO_BGP**、**rip** なら **PROTO_RIP** 等のようになります (プロトコル名とシンボルの関係については `util/configure.pl` 内の `%proto` の定義を参照してください)。

(ダンプファイル名や設定ファイル名等の) いくつかのジェネリックな名前も設定で指定した実際の値に置き換わります。ソースファイルに **@(symbol)** というトークンが含まれている場合、このトークンは `Config` ファイルで定義した **symbol** の値に置き換わります。例えば、`Config` ファイルに次のような内容が含まれていて、

```
path_config            /etc/\%s.conf
```

`paths.h` というファイルに次のような内容が含まれていると、

```
#define _PATH_CONFIG   "@(_path_config)"
```

結果として以下のようになります。

```
#define _PATH_CONFIG   "/etc/\%s.config"
```

[注意: 最初の '_' は無視され、置き換えは大文字も小文字も同じに扱われます]

7.2.5 新たなプロトコルの追加

新たなプロトコルを GateD に導入する場合には以下のことを推奨します。

- Config ファイルで使用するプロトコル名を定義し、この名前を `util/configure.pl` の `%proto` の定義に追加します。Config ファイルで新たなオプションが必要な場合、`%options` にもデフォルトの値を追加してください。(Makefile を生成するのに使用するディレクトリとファイルを示す) `%dirPath`、(`paths.h` にディレクトリやファイル名をハードコーディングしている) `runtimePath`、(Makefile でのパラメータを指定する) `%miscParam`、(Makefile での UNIX コマンドを指定する) `%unixCmd` や (UNIX コマンドのフラグを指定する) `%cmdFlags` を参照することもできます。
- (C や yacc 等の) 新たなソースファイルを section 7.2.4 で定義した文法を使用して `util/files` に追加します。
- 新たなターゲットを追加するために `util/Makefile.temp` を変更します。
- GateD の parser に新たなプロトコルの設定のサポートを追加するために `parser.y` を更新します。section 7.2.4 で定義した文法を使用して使用状況に応じたコンパイルが可能となります。

7.3 プロトコルスケジューリング

理論的には、ルーティングプロトコルでの経路情報の交換とその処理にかかわっている機能を次の三つの基本的な機能からなる一連の処理としてモデル化できます。一つ目はプロトコルメッセージを受信します。二つ目は経路情報の処理を行います (この処理は他の副次的な処理のトリガとなったり、しばらくして実行するようにスケジューリングするための処理をするための要因となるかもしれません)。三つ目はプロトコルメッセージを送信します。

GateD ではこれらの機能を有効的に実装するために task、timer や job のような機能からなる共通のサブルーチンを提供しています。*task* は個々にスケジューリングが可能な一連のプロトコル処理です。*timer* は後で実行するようにスケジューリングされる処理で、これによりそのときに占有できなかった処理を行います。この処理は特定の task に関係しています。(フォアグランドにもバックグランドにもなり得る)*job* は後で処理されるようにスケジュールされているその時点で占有できなかったいくつかのプロトコル処理です。

7.3.1 GateD での task

GateD では、(例えばルータとして) 単体でプロトタイププラットフォームと複数のルーティングプロトコルの機能を提供しているため、(例えば、BGP と OSPF のように) 異なるルーティングプロトコルに属する基礎的な機能をつりあいをたもちながら、管理することができなければいけません。これをおこなうために、GateD では、基礎的な機能に関連した一連の処理の流れである **task** を提供しています。このように関連した機能は他のプロトコルの機能や同じプロトコル内での他の機能とは独立して (すなわち任意に) スケジューリングすることができます。大雑把に言えば、task は共通ルーチンと同じです (task は優先的なスケジュールはされません)。

プロトコルが変われば使用する task も変わります。task を使用するもっとも簡単な方法はプロトコル内で発生するすべての処理に一つの task を割り当てることです。OSPF や RIP ではすべてのプロトコル処理は単一の task でおこなわれます。個々の peer 毎にコネクションを持つ BGP では peer (あるいは peer グループ) 毎に (メッセージ処理やその処理等の) プロトコル処理の task を割り当てます。

現時点での GateD の実装では、(GateD のダンプ task を除いて) task は他の UNIX のプロセスに関係していません。

task データ構造体

task モジュールの核となる構造体は task 構造体です。これは、一連の基礎的な処理に関するすべての処理状態を示します。このような状態には、(例えば、関連している *bgpPeer* 構造体のようなプロトコル処理状態等

の) プロトコルに固有な情報へのポインタが含まれています。task は通常 UNIX のソケットと関連づけられていて、task 構造体にはソケットのディスクリプタが含まれています。

さらに、task 構造体には task モジュールがサポートする個々の基礎的な処理をおこなうために、プロトコル固有のルーチンへのポインタも含まれています。この基礎的な処理を次に示します。

- プロトコルメッセージを一つあるいはそれ以上読み込みます。
- プロトコルメッセージを一つあるいはそれ以上書き出します。
- 経路情報の変更をフォワーディングテーブルへ反映します。
- 指定した時刻に処理をおこなうようにスケジューリングします。
- 空いたときに処理をおこなようにスケジューリングします。

例えば、peer からの更新のメッセージを読み込む準備ができると、BGP の実装では peer に関連した task に *bgp_recv_v4_update()* へのポインタをハンドラとして登録します。そして、実際にソケット上にデータが到着すると、task モジュールはプロトコル固有の読み込みをおこなうためにこのハンドラを呼びます。

GateD の再設定をプロトコルに知らせるために、task ではプロトコル固有の通知ハンドラ (例えば、再設定の前や後などに呼ばれるようにするプロトコル固有のハンドラのように) へのポインタも保持しています。

GateD では独自にこれらの処理の優先づけをしています。どのような場合でも、ペンディング状態の timer が最も優先度は高く、フォアグランドの job、ソケットからの読み込み、プロトコル固有の例外処理、ソケットへの書き込み、そしてバックグランド job へと続きます。GateD には (*task_read_queue* や *task_write_queue* 等の) 処理が終るのを待つために task で使用するグローバル変数があります。task 構造体にはそれぞれのキューがどこにあるかを示す linked list ポインタも保持しています (実際には、job と timer は別のデータ構造体に入れられています。これは task が複数の timer や待機中の job を持っているかもしれないからです)。

task エントリポイント

プロトコル処理では通常、最初に task のためのデータ構造体を (*task_alloc()* を呼んで) 割り当てて、task 構造体のフィールドを初期化し、task のグローバル変数のリストにこの task 構造体を挿入するために *task_create()* を呼ぶことにより task を作成します。この挿入は優先順位にしたがっておこなわれます (task には固定の優先度が決まっています)。最後のこの機能では task の read や write で使用するソケットも作成します。

task が保持するソケット上での処理には (task が保持するソケットの close や read や write のグローバルキューから削除するための) *task_close()* や、*task_set_socket()* や *task_reset_socket()* があります。

task_delete() が呼ばれると、task 構造体で保持しているポインタが示す領域は削除されます。これは task に関連したすべての (timer や job などの) 資源を解放し、task を属していたかもしれないリストから削除します。

task モジュールの関数のうちいくつかは個々の task 処理にグローバルな指示を通知するために使用されます。このうちのいくつかは task の優先度で処理されます。このようにしてすべてのプロトコルで設定ファイルを再読み込みさせるために *task_reconfigure()* が呼ばれます。同様に、ネットワークインタフェースの状態の変更をプロトコルに通知させるために *task_ifachange()* や *task_iflchange()* が使用されます。そして、(例えば、管理者の操作の結果として) 処理を終了させるように GateD のプロトコルに通知するために *task_terminate()* が使用されます。また、関連するプロトコルにルーティングポリシが変更されたことを知らせるために *task_newpolicy()* が使用されます。最後に、GateD のグローバルなルーティングデータベースに変更が生じた場合、*task_flash()* は各プロトコルの処理ハンドラを呼びます。

ひとたび task が作成されると、個々のプロトコルでは task 用のソケットから *task_set_recv()* を使用して、読み込むためのスケジューリングや *task_set_write()* を使用して、ソケットに同期書き込み (プロトコルにソケットから送信データがある場合に直接書き込もうとして、それがブロックされるとこの関数を呼びます) のスケジューリングをおこなうことができます。いずれのプロトコルでもソケットの例外機能を使用しないようになっています。

OS からデータを読み込むと、task はグローバルな *task_recv_buffer* を読みにいきます (task は占有されないため、GateD はこれをおこなうことができます)。そして、プロトコルはバッファ中のすべてのデータをとり

こむことになります。同様に、データを書き込む前に、プロトコルではグローバルな *task_send_buffer* にデータを入れます。write がブロックされると、プロトコルではそのデータを別のところにスプールします。

task 処理と GateD の main での loop

GateD でのすべてのプロトコル処理は *main()* 内での loop でスケジュールされます。先に述べたように main での loop では個々の処理の優先度を決め、最後に行った loop の後でどの処理が有効になったかを見つけるために繰り返し loop を行います。

基本的に main での loop では read、write や例外処理で使用するすべてのソケットの識別子について *select()* を行います。timer でスケジューリングされていなかったり、job のキューが空の場合は、*select()* で待ちます。GateD は、つぎにスケジューリングされている timer が動くか、8 秒までのどちらか短い方の間は *select()* で待ちます。select から抜けると、「高い優先度」の timer の処理 (timer の section を参照してください) を行います。(UNIX のシグナル経由で) 管理者が操作をした場合、(*task_process_signals()* により) GateD はこれを処理し、*select()* に戻る前にキューに入れられたフォアグラウンドの job を実行します。それ以外の場合は、待機中の timer やバックグラウンドの job のソケットの read や write を (*task_process_sockets()* により) 順番に行います。

このようにして、timer や job と同様にプロトコルのメッセージの read や write の処理は GateD の main の loop からスケジューリングされます。(read の結果として) GateD のルーティングデータベースが更新されるときのみ flash 用のルーチンが呼ばれます。

7.3.2 GateD での timer

timer は指定したタイミングで特定の処理をおこなうように GateD にスケジューリングします。GateD の timer モジュールでは一度限りの timer と間欠 timer の双方の指定が可能です。時間の指定は 1 秒単位になります。timer による処理の開始のタイミングはシステムの負荷に左右されます。したがって、最後の timer による処理から次の timer による処理までの間隔は正確でなくなることもあり得ます。

GateD ではプロトコルによる優先度の高い timer の指定が可能です (このような timer では「通常の」timer よりも正確な処理タイミングが期待できます)。GateD では timer 処理のタイミングの変更も可能です。すなわち、timer がどのように初期化されたかにもよりますが、次の timer 処理のタイミングが、最後に実際に timer 処理されたのがいつなのかと timer の間隔をもとに計算されます。また、タイミングを変更しない timer の設定も可能です。この場合、timer が最初にスケジュールされてからのある値の整数倍の間隔で正確にスケジューリングされます。

プロトコルでは、コネクションの状態やコネクション開始のタイムアウトなどの周期的な task のために timer を使用します。

現在の GateD の実装では、timer による処理の開始をするかどうかは、すべてのアクティブな timer を調べることによりチェックしています。timer による処理の開始の通知には OS の機能は使用していません。

timer データ構造体

核となるデータ構造体は timer 構造体 *task_timer* です。これには timer によって開始される処理とともに、(タイマの間隔やタイマの開始時刻などの) timer のパラメータが含まれます。

task_timer 構造体は task モジュールが保持している三つのグローバルキューのうちのいずれかにキューイングされます。これは、高い優先度の timer と、アクティブな timer とアクティブでない timer です (最後の timer はプロトコルが一時的にタイマによって開始される処理をしないことが必要な場合に使用されます)。最初の二つは、処理を容易にするために処理の開始の順に並べられます。

高い優先度の timer は HELLO を送信する OSPF のようにいくつかのプロトコルで使用されます。BGP では、keepalive、コネクション再送等タイマが必要な処理にはすべて「通常の」timer を使用します。

timer エントリポイント

task_timer_create() では指定したパラメータで timer を作成し、timer の適切なグローバルなリストに挿入します。逆に、*task_timer_delete()* は timer をキューから削除し、timer が保持しているすべての資源を解放し

ます。

また、timer によって開始される処理をしないよう (すなわち、アクティブでないリストに移動します) に *task_timer_reset()* を使用します。

timer 処理

timer 処理は最初に *task_timer_dispatch()* と *task_timer_hiprio_dispatch()* でおこなわれます。ここでは、単に timer のリストをサーチし、適切なハンドラを呼び出し、次の処理までの間隔を再計算します。その間隔で timer が起動し、それまでの間の timer を再スケジューリングします。これは main の loop から呼ばれます。

7.3.3 GateD での job

GateD の *job* は処理の単位を意味し、それは今よりも後で処理されるようにスケジューリングすることができます。job はフォアグラウンドにでもバックグラウンドにでもなります。フォアグラウンドの job は処理にかかる時間が制限されていて、GateD のルーティングデータベースを安全に変更することができるようになったときに処理され、多くは高い優先度の timer のハンドラや flash ハンドラから呼ばれます。バックグラウンドの job は待機中の timer や I/O がなければそれだけ長く処理できるようにスケジューリングされます。個々のバックグラウンド job には 0 から 7 までの優先度を持っています。バックグラウンド job は優先度の順にスケジューリングされます。

例えば、BGP では peer の経路情報を一掃する場合、エラーがあると peer とのコネクションを close するためにフォアグラウンド job を使用します。read キューに残っていたり、task を処理するために timer がスケジューリングされているかもしれないため、task の write ルーチンからはこの job は正しく処理されることができません。同様に、OSPF では SPF 処理の後 timer を設定して、SPF 処理がその期間おこなわれないようにします。timer の終了時には、次の SPF 処理をするためのバックグラウンド task を設定します。

job データ構造体

単一の job への識別子をもつ *task_job* 構造体にはその job がスケジューリングされる task へのポインタや処理するルーチンへのポインタやそのルーチンへの一個の引数が含まれます。

GateD では (FIFO の順で) フォアグラウンドの job のグローバルなリストや、(優先度順で) バックグラウンドの job を保持します。さらに、バックグラウンドの job には *task_job* 構造体の最後の優先度と同じになっている、その場所を示すポインタの単一な配列があります。

job エントリポイント

プロトコルでは *task_job* 構造体を割り当て、適切なリストに挿入するために *task_job_create()* を呼びます。その逆のことをするためには *task_job_delete()* を呼びます。フォアグラウンドの job は終了すれば自動的に削除されます。バックグラウンドの job は明示的に削除しなければいけません。

job 処理

GateD の main の loop ではフォアグラウンドの job の処理のスケジューリングをおこなうために *task_job_fg_dispatch()* を呼びます。これはすべてのフォアグラウンドの job を終了させます。可能になれば、GateD の main の loop ではバックグラウンドの job の処理のスケジューリングをおこなうために *task_job_bg_dispatch()* を呼びます。先の関数と違い、この関数では最初にバックグラウンドの job のリストから最も優先度の高いバックグラウンド job を削除し、その job の実行をスケジューリングし、元の関数に戻ります。

7.4 メモリ管理

GateD は、ある状況においては OS の提供するメモリ管理機能を使用します。しかし、ほぼ共通に制御用に割り当てる場合、メモリの使用を最適化するために、GateD では自分自身のメモリ管理ルーチンを提供しています。メモリ管理のソースコードはおそらく歴史的な理由により、task モジュールの一部となっています。

GateD において、プロトコルモジュール (と他のモジュール) は二つの異なる方法でメモリを割り当てます。固定長のメモリの動的割り当てにおいては、GateD は次で説明するメモリ割り当てを行います。(例えば、BGP の経路属性のための *as_path* 構造体などの) 可変長のメモリの割り当てについては、GateD は *malloc()* を使用します (実際には、GateD は後者の割り当てについても同様に最適化していて、*malloc()* を使用するのは共有しない場合のみで、共有している場合は固定長のメモリを割り当てます)。

固定長のメモリの割り当てではモジュールから繰り返し同じ大きさの割り当てが要求される場合に最適化されています。モジュールは、最初に (*task_block* を呼ぶことにより) メモリ管理機構から後でメモリを要求する時に使用するメモリサイズを示すことになるメモリのディスクリプタを取得します。後でメモリを割り当ててもらう場合、メモリ管理機構をこのディスクリプタを引数に呼びます。これにより、メモリ管理機構は適切な大きさのメモリへのポインタを返します。

7.4.1 メモリ管理データ構造体

固定長のメモリ割り当て機構には *task_block* と *task_size_block* の二つの主な構造体を持ちます。上記に関連したディスクリプタである *task_block* には個々の割り当て要求の大きさといくつかのデータからなります。

task_size_block は特定の大きさの割り当てのためのディスクリプタです。これにはその大きさの *task_block* のリストが含まれます。さらに、特定の大きさの free リストへのポインタも含まれます。

7.4.2 メモリ管理エントリポイント

プロトコルモジュールは *task_block_init()* を呼ぶことによってメモリ管理機構に登録し、割り当てる大きさを指定します。

task_block_alloc() と *task_block_free()* のマクロでは、要求した大きさで割り当てと解放を行います。メモリの割り当て方針は次に説明します。

プロトコルモジュールによっては直接ページ単位でメモリの割り当てを行います。例えば、task モジュールでは送受信バッファをこの方式で割り当てます。このようなモジュールでは *task_block_reclaim()* を呼ぶことにより、明示的に割り当てたページを解放することができます。このようなページはグローバルな free リストに入れられ、メモリ管理機構はここからメモリブロック割り当てをおこなうことができます。

7.4.3 メモリ管理内部

メモリ管理機構内部は次の例により説明することができます。プロトコルモジュールがメモリ管理機構に 16 バイトのブロック割り当てを登録した場合、メモリ管理機構は *task_size_block* の 16 バイトブロック識別子にリンクした後、プロトコルモジュールに *task_block* を返します。

後者の識別子では free な 16 バイトブロックのリストを保持しています。free なブロックは一方向のリストにリンクされています (free なブロックへのポインタは双方向のリストの最初に入れられていますが、free なリストでは前方ポインタのみ使用されます)。*task_size_block* での free なリストが空になると、メモリ管理機構はページを割り当て、ページを適切な大きさのブロックのリストへ入れます (*runt* と呼ばれるひとつのブロックに合わないページの切れ端はその大きさにかかわる free リストにリンクされます)。

割り当てはこの free リストからおこなわれます。プロトコルモジュールがブロックを解放すると、解放されたブロックは free なブロックへ戻されます。

7.5 neighbor との情報交換

すべてのプロトコルは一つあるいは複数の neighbor と経路情報の交換をおこなうために通信をします。GateD ではプロトコルが経路情報の送受信をおこなうためのインタフェースとこのような通信で必要な物理的な情報のためのいくつかの概念的なデータタイプ (例えば neighbor 自身や GateD がルータと通信するときに使用する物理的なネットワークインタフェース) を提供しています。

7.5.1 データ転送機構インタフェース

多くのプロトコルの実装では neighbor との経路情報の送受信をするために OS のインタフェースに直接アクセスします。このようにして、BGP では peer との TCP コネクションに関するソケットを直接 read します。

しかし、古いプロトコルの実装には *task_receive_packet()* と *task_send_packet()* という関数をそれぞれ使用しているものもあります。ルーティングプロトコルのメッセージを送信するために unreliable なデータグラムを使用するプロトコルでは理想的ですが、データストリーム型の転送を使用するプロトコルではこれらの関数では不十分です。

7.5.2 neighbor ゲートウェイモジュール

GateD ではルーティングプロトコルの neighbor と交換するすべての有用な情報をまとめるために絶対的なデータタイプを提供しています。

ここでの基本的なデータ構造体は *gw_entry* で、一つのプロトコルに (可能ならば) 一つの peer で使用し、これは neighbor のアドレス、いくつかの状態の情報、neighbor から得た経路情報のリストと neighbor に対しておこなう import statement と export statement を保持します。

設定ファイルから読み込む場合には、文法解析のために *gw_locate()* を呼び、設定した neighbor に *gw_entry* を割り当てます。

7.6 経路情報保持機構

GateD の核となる機能では、個々のプロトコルが持つ destination へのその時点で適した経路情報の核となる保持機構を提供します。BGP の経路属性の情報を保持し、処理するのに十分な方式も提供しています。

7.6.1 GateD 経路情報データベース

GateD では一つあるいはそれ以上のプロトコルが持つある destination への経路情報を保持する核となる機構を提供しています。経路情報データベースモジュールは同じ destination に対してどのプロトコルの経路情報 (そして、プロトコル内で複数の経路情報を保持している場合にどれ) を選択するかの一連の規則を形成します。

プロトコルモジュール毎にそれぞれの方式で経路情報データベースを使用します。OSPF では LSDB を個々に保持し、link state 処理を行う時に、経路情報を経路情報データベースに取り込みます。OSPF からはある destination への経路情報は唯一となります。RIP と BGP では peer から受信した経路情報の更新を追加しているため、peer の数だけ destination への BGP の経路情報の経路情報データベースがあるかもしれません。

経路情報データベースへの一連の変更は明示的にデータベースをロックすることによりおこなわれます。データベースがロックされていない場合は、経路情報データベースモジュールはすべてのプロトコルに、おこなわれたすべての変更を通知します。これにより、経路情報の更新メッセージや不到達性を送信させることができます。

データタイプの詳細な説明、マクロや経路情報データベースの機能の定義を次に説明します。

経路情報データベースデータ構造体

経路情報データベースの主なデータ構造体は *rt_entry* です。この構造体は特定の destination とマスクへの個々の経路への次 hop、メトリック、優先度や AS 経路等の情報を保持します。

同じ destination とマスクペアへの経路情報はすべて双方向のリストでつながれています。このリストの先頭は *rt_head* 構造体です。*rt_entry* にはこの構造体への逆ポインタが含まれています。*rt_head* はこの destination とマスクへの「アクティブな」(すなわちもっとも適した) 経路情報へのポインタや以前「アクティブであった」経路情報やその他の情報から構成されています。

GateD の経路情報データベース保持機構にはデータベースとなっているすべての経路情報を単一の radix tree で (実際にはアドレスファミリごとにひとつずつ) 保持しています。tree の検索は destination とマスクの対でおこなわれます。この tree の内部 node は *radix_node* で、個々の内部 node は *rt_head* 構造体を指します。

例により経路情報データベースのデータ構造体をより詳しく説明します。

他のデータ構造体にはプロトコルに関するものもあります。*rt_list* 構造体にはデータベースに起こった変更のリストを含み、変更がおきると常にすべてのプロトコルに伝えられます。*rt_parms* はプロトコルのメッセージを解析する場合に経路情報のパラメータを一時的に保持するために使用します。*rt_entry* は経路情報用に後で作成されます。

GateD が「アクティブな」経路情報を選択すると、新たな「アクティブな」経路情報をすべてのプロトコル (これは OSPF や BGP のコネクションに関連します) に通知します。プロトコルの export statement によりその経路情報をアナウンスすることができます。*rt_bits* フィールドは指定した経路情報をその時点でアナウンスしているプロトコルを示します。このフィールドはビットマスクになっていて、各プロトコルにはグローバルなビットマスクから “route bit” として割り当てられます。*rt_bits* のあるビットがセットされていれば、そのビットのプロトコルが経路情報をアナウンスしていることをあらわします。

プロトコルでは個々に経路情報データベース内に保持している経路情報マップを保持しています。*rt_entry* について、プロトコルはどのようにして *rt_entry* への参照を含むどのデータ構造体を見つけるのでしょうか？このようなプロトコル固有のデータは経路情報のエントリの *rt_head* 内に保持されています。バイト列中のこのデータの位置はプロトコルの “route bit” により決まります。個々のプロトコルに対して、GateD はプロトコル固有のデータを保持することができるこのバイト列の中にスタティックにブロックを割り当てます。GateD では割り当てられた箇所 (*rtbit_info*) へのプロトコルの経路情報からのマップと同様にバイト列 (*rttsi_map*) 中に割り当てられたグローバルなビットマスクを保持します。このデータ構造体については以下の例を使用して詳しく説明します。

経路情報データベースエントリポイント

プロトコルは *rt_add()* を呼び、指定したパラメータで指定した destination とマスクの対を経路情報データベースに追加します。この関数では (必要ならば、同じ destination への経路情報が存在していない場合に) *rt_head* を割り当て、radix tree に挿入します。

rt_delete() という関数は経路情報データベースから論理的に削除します。経路情報に割り当てられた資源はすぐに解放されず、その経路情報をアナウンスしている可能性のあるすべてのプロトコルから削除されたという通知を待ちます。

rttsi_alloc() と *rttsi_set()* と *rttsi_get()* という関数は *rt_head* 内のプロトコル固有のデータを保持するためのそれぞれ連続したブロックを割り当て、指定した *rt_head* からプロトコル固有のデータを保持し、取得します。

プロトコルが peer から先に送られた経路情報の更新を受信すると、新しい情報で経路情報のエントリを更新するために *rt_change_aspath()* を呼びます。最適にするため、処理中の経路情報がその時点で「有効な」場合、GateD では変更を別に保持し、経路情報が再びアナウンスされるとそれを適用します。

rthlist_all() という関数は指定したアドレスファミリの radix tree を動きまわって tree 中のすべての *rt_head* のリストを返します。tree 中のすべての *rt_entry* のリストあるいは指定した範囲のリストを返すためにいろいろな関数があります。

最後に、再設定をおこなうと *rt_new_policy()* が呼ばれ、*rt_close* によって *rt_flash_update()* がキューに入れられ、カーネルのルーティングテーブルとすべてのプロトコルのいちばん最後の経路情報の変更を通知するために task にスケジューラから呼ばれます。

経路情報データベース内部

rt_table_add() と *rt_table_locate()* と *rt_table_delete()* という関数はそれぞれ、radix tree に対してある destination アドレスとマスクについての追加、検索、削除を行います。つぎに標準的な radix tree の操作を説明します。

*rt_event_* ではじまる関数は、指定した状態変化を *rt_entry* に示します。(例えば経路情報が削除されて、新たな有効な経路情報を求めなければいけなくなるように) 状態変化を伴う処理もここでおこなわれます。

rt_insert() という関数は「有効な」経路情報を選択する場合の経路情報データベースの規則をとりこみます。もっとも高い優先度の経路情報は「有効な」経路情報として選択されます。タイブレークの場合は、短い AS 経路を持つ経路情報が選択されます。この場合のタイブレークには次 hop のアドレスがもっとも小さいものが選択されます。

7.6.2 GateD の AS 経路モジュール

GateD の AS 経路モジュールでは AS 経路の解析機能あるいは BGP 属性の操作を提供します。(IS-IS のような) 他のプロトコルでも同じような機能を将来必要とするかもしれない状況では、このモジュールは BGP の実装とは分離されるでしょう。

理論的には、(origin や AS 経路やオプションの transit 属性のような) 一連の属性を渡すと、この機能はこれらの属性の識別子を返します。このモジュールでは同じ識別子を共有する等価な属性の定数も保持します。

詳細なデータタイプ、マクロ、AS 経路モジュールの関数定義を次に説明します。

AS 経路データ構造体

AS 経路モジュールの主なデータ構造体は属性識別子のブロックである *as_path* です。これは origin の属性と AS 経路と経路情報の属性リスト中のオプションの transit 属性から構成される可変長のブロックです。

二つの経路情報が同じ属性を持つ場合、同じ *as_path* 構造体を共有します。識別子には AS 属性を共有する経路情報の数を示すリファレンスカウントが含まれます。

ひとつあるいはそれ以上の経路情報に参照される AS 経路はすぐに取得できるようにグローバルなハッシュテーブル *path_list* に保持されます。ハッシュの値は AS 経路長と origin の属性とオプションの transit 属性長から得ます。

内部に関連するもう一つのデータ構造体は *as_path_list* です。これは個々の経路情報を集約するために使用されます。この構造体のリストは集約経路にかかわる経路情報の AS 経路から構成されます。

AS 経路モジュールエントリポイント

AS 経路モジュールの最初の機能は *aspath_attr()* と *aspath_format_v4()* の二つのルーチンにより実行されます。属性リストを含む BGP の更新メッセージへのポインタを渡すと、*aspath_attr()* は同じ属性を持つ *as_path* 構造体へのポインタを返します。*aspath_format_v4()* は逆の動きをし、*as_path* 構造体を渡すと、*as_path* 識別子の内容を使用して BGP の更新メッセージの属性部分を作成します。

経路情報の更新を受信し、GateD が経路情報データベースが変更するときは、GateD はフォワーディングテーブルを更新します。これをおこなう前に、可能ならば経路情報の変更を集約しようとします。
aspath_do_aggregation() は集約経路の AS 経路を処理するのに使用されます。

最後に、*aspath_rt_build()* は AS 経路識別子を *rt_entry* に関連づけるために使用します。これはほとんどの場合、単にリファレンスカウントを増やすだけです。

AS 経路モジュール内部

AS 経路のハッシュテーブルの操作は比較的直接的です。*aspath_find()* は単一の識別子によりあらわされる一連の経路に一致するものを見つけます。

AS 経路のメモリ確保では属性ブロックの二つの共通な場合は 32 バイトか 64 バイト以下のいずれかを使用した固定ブロック割り当てを行い、それ以外では *malloc()* を使用します。

GateD の実装では GateD が起動しているルータが複数の AS に属することを認めています。AS 経路を同じルータ上の異なる AS で動作している BGP に渡すこともできます。このルータが経路情報をアナウンスする場合、経路情報にはこのアナウンスを受け取り、渡したローカルなすべての AS を正確に含んでいなければいけません。AS 経路モジュールは「ローカルな」AS 番号の配列を保持しています。一連の *aslocal_*()* ではじまる関数はこの配列を操作するために使用します。*as_path* 識別子はこの AS 経路を受信したローカルな AS のビットマスク (ビットマスク中のビットの位置はローカルな AS 配列へのインデックスを示します) を保持しています。AS 経路モジュールが更新メッセージを作成する場合、AS 経路が一連の AS として関連しているすべてのローカルな AS のリストを含むようになっています。

7.6.3 経路情報データベースモジュールのタイプ宣言

rt_changes

この構造体は *rt_head* 構造体の一部です。この構造体の後半部分には経路情報のリストとアナウンスされた有効な経路情報へのポインタがあります。この構造体には有効な経路情報に対しておこなわれた (メトリックや優先度や AS 経路等の) 変更があればそれが含まれています。

rt_aggr_entry

プロトコルで個々の destination を集約する場合、集約される経路情報は双方向のリストに保持されます。この構造体の要素にはこの双方向のリストの先頭を含まれています。双方向のリストにはこの経路情報に対する *rt_entry* へのバックポインタが含まれます。

rt_aggr_head

rt_aggr_entry のリストの先頭を含みます。これには個々の経路情報が持つ AS 経路も含みます。

rt_head

GateD での経路情報データベースは radix tree で *rt_head* 構造体へのポインタを含む個々の内部ノードになっています。この構造体には destination アドレスとその destination への *rt_entry* の双方向リンクを含みます。この構造体には、radix tree での内部ノードへのバックポインタも含まれています。

rt_entry

この構造体には、GateD が持つ destination への経路情報データベースのうちのひとつの経路情報から構成されています。この識別子を構成する情報には、経路情報を (IGP や EGP 等) どのように得たのかや、destination への次 hop やメトリックあるいは preference の値や AS 経路などを含んでいます。

rt_parms

すべての正当性のチェックが終り、経路情報がデータベースに追加されるまで経路情報のパラメータを保持するために使用する一時的な構造体です。

rt_tsi

それぞれの *rt_head* には target specific information (tsi) のリストが含まれます。これは経路情報やプロトコルに固有 (例えば、BGP 経路の tsi データはこの経路情報を含む関連する bgp_rto_entry を指します) な大量の 16 バイトのデータリストです。各プロトコルの tsi は順番に保持され、*rttsi*()* で始まる一連のルーチンはこのデータにアクセスするために使用されます。

rt_list

処理すべき経路情報の flash 更新はこの構造体に保持されます。*rt_list* 構造体はページの先頭にある識別子ブロックです。処理すべき経路情報へ識別子ブロックはこの識別子ブロックの後のページに保持されます。それに続くページは識別子の *rtl_next* フィールドを用いてリストでつながれています。

7.6.4 経路情報データベースモジュールのマクロ

*RTBIT_**

各プロトコルや (peer があるような) プロトコルでの peer にはビットが割り当てられています。*rtbit_map* 配列には各プロトコルでの固有の状態が含まれています。各 *rt_entry* には経路情報が得られたプロトコルを示す

ビットマスクが含まれます。このマクロはこのビットマスクを操作します。GateD での経路情報データベースモジュールではビットからモジュールに登録されているビットを持つ task へのマッピングを保持しています。

RT_ALLRT(_REV)

rt_entry のリストを最初から (最後まで) 順番に動きまわるマクロです。

RT_ALLRT(_REV)_END

loop で動き回るのを終了するのに関連したマクロです。

*RT_LIST_**

このマクロでは flash 更新を含む *rt_list* を操作します。これには、リストを割り当てるマクロと同様に一度にリストの一つの要素を動き回るマクロを含みます。

7.6.5 経路情報データベースモジュールの関数定義

rttsi_alloc()

rttsi_map 配列はその時点で割り当てられているすべての tsi の位置を示すビットマスクとなっています。割り当てられていない最初のブロックを単純に探し、それを割り当てるための処理を行います。これにより各プロトコル毎に tsi を連続して割り当てます。この関数は *rtbit_map* 配列の *index* フィールドを適宜セットします。

rttsi_get()

rt_head 構造体を渡すことにより tsi リストを動き回り、指定した *rtbit* に関するデータを見つけます。そして、指定したバッファにデータをコピーします。

rttsi_set()

rt_head 構造体を渡すことにより tsi リストを動き回り、*rtbit* で指定された tsi の位置へ指定したデータを書き込みます。

rttsi_reset()

rt_head 構造体を渡すことにより tsi リストを動き回り、*rtbit* で指定された tsi の位置のデータのリセットを行います。

rttsi_release()

rt_head 構造体を渡すことにより tsi リストを動き回り、tsi データの各ブロックを解放します。

rttsi_free()

指定した rtbit_map がビットマップ中にすでに存在していないことを反映するために tsi の割り当てのビットマップを変更します。

rth_remove()

tsi リストや *rt_head* が属する radix tree を含む指定した *rt_head* についてのすべての資源を削除します。経路情報のリスト自身は別のところで管理されます。

rth_locate()

指定された destination とマスクの対について、マッチするヌルでない外部データを持つ内部ノードを見つけるために tree の検索ルーチンを呼びます。そうでなければ、*rt_head* を割り当て、初期化し、radix tree に挿入します。

rtchanges_free()

rt_changes 構造体にかかわる資源を解放します。これにはいずれの AS 経路やアドレス情報も含まれます。

rt_free()

rt_entry 構造体の (AS 経路や必要ならば *rt_head* などの) 資源を解放します。

select_active()

destination への経路情報のリストが変わる (すなわち経路情報が追加されるか削除されるかあるいはその両方の場合) とこの関数が呼ばれます。この関数は指定された *rt_head* によって示される経路情報のリスト中の最初の経路情報を選択します。停止状態のプロトコルがその時点で有効な経路情報をアナウンスしていないならば、これは有効な経路情報として使用されます。

rt_remove()

経路情報をそれが属するリストから削除します。また、キューの先頭の経路情報の数を一つ減じます。

rt_insert()

経路情報のリストでは、経路情報は有効なものを最初にし、隠れたあるいは「削除された」経路情報がそれに続きます。「削除された」経路情報を挿入するにはリストの最後に追加します。隠れた経路情報を挿入するにはリストを最後から始まって挿入するのに適切な位置が見つかるまで探します。最後に、有効な経路情報を挿入は優先度の順により行います。同じ優先度の経路情報ではタイブレークをするために aspath 長や origin ID の数値を使用します。

rt_event_*()

この関数群は経路情報の状態が変わると呼ばれます。このような状態の変更例には有効や無効になるか、経路情報の優先度や次 hop の変更によるものが含まれます。実行される処理は状態 (あるいは状態の組み合わせ) に依存し、通常は適切な状態フラグの設定を含み、(ある場合には) 経路情報の再キューイングとあらたな有効な経路情報の処理も含まれます。

rt_new_policy()

GateD が初期化される時あるいは再設定された後に呼ばれます。この関数では存在する flash のリストを削除し、全経路情報のデータベースから完全な flash のリストを作成します。さらに、カーネルのルーティングテーブルも更新します。

rt_flash_update()

GateD が終了する時に呼ばれます。この関数では、集約経路を再処理したり引数で渡されたリストでカーネルの経路情報を flash することにより存在する flash リストを単に処理します。

rtbit_alloc()

rtbit_map 配列にはプロトコルやプロトコルでの peer に対して割り当てられたビットマップが含まれています。この関数はまだ割り当てられていない最初のビットを見つけるために配列を動き回り、このプロトコルにそれを割り当てます。また、このビットと指定したデータ長に対して *rttsi_map* から tsi ブロックも割り当てます。

rtbit_free()

指定した *rtbit_map* を解放します。

rtbit_set()

引数に *rt_entry* が渡されると、指定したプロトコルビットをセットしてこのプロトコルがこの経路情報をアナウンスしているということを示します。

rtbit_reset()

指定した経路情報の指定したプロトコルビットをリセットします。

rtbit_reset_all()

すべての存在している経路情報中の指定したプロトコルのビットをリセットします。また、経路情報が属している各経路情報のヘッダ中の tsi フィールドもリセットします。

rt_locate()*

destination とマスクが引数として渡されると、この destination に関連し、さらにある範疇にマッチする (例えば、状態やプロトコルやゲートウェイにマッチする) *rt_entry* を示します。この関数は、radix tree を動き回り、*rt_head* にマッチするものを見つけ、関連する *rt_entry* を見つけるまで *rt_list* を動き回ります。RIP や OSPF で使用し、BGP では使用しません。

rt_lookup()

rt_lookup()* と同様に、この関数は一連の範囲の destination とマスクに一致する経路情報を探します。しかし、この destination への集約経路を見つけるためにすべての経路情報を動き回り、この範囲にもっとも一致する集約経路を見つけます。この関数は SNMP のソースコードで多く使用されます。

rt_add()

rt_parms 構造体中の経路情報へのさまざまなパラメータを渡すことにより、GateD の経路情報データベースに経路情報を追加します。この関数は例えば BGP で経路情報の更新を受信すると呼ばれます。そしてこの *rt_entry* が入れられるべきキューの先頭に単に位置させるだけです。*rt_entry* を割り当て、(優先度やメトリック等の) パラメータをその中にコピーします。*rt_entry* を指定したルータによりアナウンスされる経路情報のリストに追加します。必要ならば、AS 経路を経路情報に割り当てます。最後に、優先度の順に経路情報をリストに挿入するために *rt_insert()* を呼びます。

rt_change_aspath()

適切な説明ではありませんが、存在する経路更新情報を受信したり、一つあるいは複数の (メトリックや AS 経路や優先度等) 経路情報の属性が変更されると呼ばれます。この場合のもっとも複雑な処理は、次 hop(や複数経路への一連の次 hop) が変更された場合におこなわれます。

rt_delete()

到達不可能の経路情報を受信するとこの関数が呼ばれます。これはゲートウェイのリストから経路情報を単に削除し、(例えば、flash リスト上にあれば慎重におこなわなければいけないことになる) 削除の印を付けます。

rt_default()*

設定により指定した次 hop と優先度とともにデフォルトの経路情報を指定することができます。一連のこのようなルーチンは再設定後デフォルトの経路情報を追加削除したり再設定するために使用されます。

7.7 設定

7.7.1 GateD ポリシモジュール

GateD ポリシモジュールでは GateD ポリシ記述子を保持する機構を提供しています。GateD 設定機構により import statement や export statement の指定ができます。ポリシモジュールではすばやいポリシ適合テストを可能にするようにこのような指定を保持します。

理論的には、あるプロトコルについて GateD では import statement と export statement で一連のポリシ要素 (経路情報のパラメータ (や、destination とマスクと優先度)...) によってそれぞれ指定できるようになっています。経路情報のパラメータはプロトコル固有なもので、例えば、BGP の policy statement は AS 番号や AS 経路の正規表現によりあらわすことができ、RIP の policy statement はネットワークインタフェースのリストやルータのアドレスリストによりあらわすことができ、OSPF の policy statement は経路情報のタグによりあらわすことができます。ポリシ要素のセマンティクスは次のとおりです。(例えば、AS 経路が指定した正規表現に一致したり、AS 番号が経路情報を受信した peer と一致したりする等) 経路情報にポリシ要素で指定した経路情報のパラメータが含まれていたりすると、その経路情報にはもっとも一致する destination とマスクについての優先度が割り当てられます。

ポリシモジュールではこのようなポリシを時間的にも空間的にも効率化して実装しています。ポリシモジュールで使用されているデータ構造体の詳細な説明やマクロや関数は次に説明します。

ポリシモジュールデータ構造体

GateD のポリシモジュールの基本的なデータ構造体は *adv_entry* です。*adv_entry* は単一のポリシ要素とポリシ要素の (destination とマスクと優先度の対の) リストのいずれも示すために使用されます。*adv_entry* 内の *dest_mask* 構造体は対のリスト中の destination のアドレスとマスクを保持するために使用されます。分離フィールドは *adv_entry* が何を保持しているかを区別するために使用されます。

ポリシ要素中の destination とマスクの対により速く経路情報の destination アドレスとマスクをマッチさせるために、GateD ではこのリストにより radix tree を作成しています。radix tree の内部ノードは *dest_mask_internal* 構造体になっています。

import と export のリストはこのポリシ要素のリストの順にならべられています。import と export のリストはプロトコル識別子のブロックで保持されています。例えば、BGP では peer に関する *gw_entry* データ構造体中に peer についてのポリシリストを保持しています。さらに、BGP の policy statement で AS 経路の正規表現による経路情報の import を制限するために、BGP ではグローバルなポリシリストも保持しています。

ポリシモジュールのエントリポイント

ポリシによる経路情報の優先度を求めたり、経路情報のアナウンスをフィルタするために、プロトコルモジュールでは *import()* と *export()* を使用します。ポリシリストを引数で渡すと、これらの関数では、リスト中で経路情報のパラメータのいずれかに一致する (もしあれば) 最初のポリシ要素を見つけます。そして、関連するポリシ要素の (destination とマスクと優先度の対の) リストからもっとも一致する destination とマスクをみつけます。

設定ファイルを読み込むと、GateD は (destination とマスクと優先度の) 対のリストから radix tree を作成します。このリストの要素を radix tree に挿入するためには *adv_destmask_insert()* を使用します。ポリシ

要素も含めて radix tree が作成されると、*adv_destmask_finish()* を呼んで radix tree 上での移動を速くするために radix tree のポインタを調整します。

ポリシモジュール内部

destination とマスクを渡すと、radix tree 上を移動する関数 *adv_destmask_match()* は、関連する (destination とマスクと preference の) 対を指定した対のリストから探します。これは *import()* と *export()* の双方で使用されます。

Bibliography

[PIM Arch] Internet Draft, *"Protocol Independent Multicast (PIM): Motivation and Architecture"*, S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei.

[PIM Spec] Internet Draft, *"Protocol Independent Multicast (PIM): Protocol Specification"*, S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei.

[RFC 827] Request For Comment 827, *"Exterior Gateway Protocol EGP"*, 10/01/1982, E. Rosen.

[RFC 854] Request For Comment 854, *"Telnet Protocol specification"*, 05/01/1983, J. Postel, J. Reynolds.

[RFC 891] Request For Comment 891, *"DCN local-network protocols"*, 12/01/1983, D. Mills.

[RFC 904] Request For Comment 904, *"Exterior Gateway Protocol formal specification"*, 04/01/1984, International Telegraph and Telephone Co, D. Mills.

[RFC 1058] iequest For Comment 1058, *"Routing Information Protocol"*, 06/01/1988, C. Hedrick.

[RFC 1105] Request For Comment 1105, *"Border Gateway Protocol BGP"*, 06/01/1989, K. Lougheed, Y. Rekhter.

[RFC 1112] Request For Comment 1112, *"Host Extensions for IP Multicasting (IGMP)"*, S. Deering.

[RFC 1163] Request For Comment 1163, *"A Border Gateway Protocol (BGP)"*, 06/20/1990, K. Lougheed, Y. Rekhter.

[RFC 1164] Request For Comment 1164, *"Application of the Border Gateway Protocol in the Internet"*, 06/20/1990, J. Honig, D. Katz, M. Mathis, Y. Rekhter, J. Yu.

[RFC 1245] Request For Comment 1245, *"OSPF Protocol Analysis"*, 08/08/1991, J. Moy.

[RFC 1246] Request For Comment 1246, *"Experience with the OSPF Protocol"*, 08/08/1991, J. Moy.

[RFC 1253] Request For Comment 1253, *"OSPF Version 2 Management Information Base"*, 08/30/1991, F. Baker, R. Coltun.

[RFC 1256] Request For Comment 1256, *"ICMP Router Discovery Messages"*, 09/05/1991, S. Deering.

[RFC 1265] Request For Comment 1265, *"BGP Protocol Analysis"*, 10/28/1991, Y. Rekhter.

[RFC 1266] Request For Comment 1266, *"Experience with the BGP Protocol"*, 10/28/1991, Y. Rekhter.

[RFC 1267] Request For Comment 1267, *"A Border Gateway Protocol 3 (BGP-3)"*, 10/25/1991, K. Lougheed, Y. Rekhter.

[RFC 1268] Request For Comment 1268, *"Application of the Border Gateway Protocol in the Internet"*, 10/25/1991, P. Gross, Y. Rekhter.

[RFC 1269] Request For Comment 1269, *“Definitions of Managed Objects for the Border Gateway Protocol (Version 3)”*, 10/26/1991, J. Burruss, S. Willis.

[RFC 1370] Request For Comment 1370, *“Applicability Statement for OSPF”*, 10/23/1992, Internet Architecture Board.

[RFC 1397] Request For Comment 1397, *“Default Route Advertisement In BGP2 And BGP3 Versions Of The Border Gateway Protocol”*, 01/13/1993, D. Haskin.

[RFC 1403] Request For Comment 1403, *“BGP OSPF Interaction”*, 01/14/1993, K. Varadhan.

[RFC 1518] Request For Comment 1518, *“Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy”*, September 1993, V. Fuller, T. Li, J. Yu, K. Varadhan.

[RFC 1583] Request For Comment 1583, *“OSPF Version 2”*, 03/23/1994, J. Moy.

[RFC 1723] Request For Comment 1723, *“RIP Version 2 Carrying Additional Information”*, 11/15/1994, G. Malkin.

[RFC 1771] Request For Comment 1771, *“A Border Gateway Protocol 4 (BGP-4)”*, 03/21/1995, Y. Rekhter, T. Li.

[RFC 1772] Request For Comment 1772, *“Application of the Border Gateway Protocol in the Internet”*, 03/21/1995, Y. Rekhter, P. Gross.

Index