第 7 部

アドレスと経路制御に関して

第 1 章

Classless な経路制御

1.1 背景

急速に発展してきたインターネットも、1980 年代後半になると、IP アドレス、特に Class B 空間の枯渇が深刻になってきた。そのため、従来は経路制御上のオーバヘッドを避けるため、複数の Class C アドレスを割り当てるよりは、アドレス空間利用効率がたとえ低くても Class B アドレスを割り振ることが行われていたが、この割り当て方針を変更せざるを得なくなった。すなわち、Class B アドレスを十分に利用することが可能な場合を除き、各組織に対しては、必要個数（ただし 2 の冪乗個まで切り上げる）の Class C アドレスの割り当てを行うようになった。

このような割り当ては当然経路制御上のオーバヘッドをもたらすことになる。従来は一つの新しい組織の接続は、インターネットのルータの経路表が一つ大きくなるだけであったが、割り当て方針の改定により、一つの組織の接続に伴い、経路表が複数、場合によっては 64 もの増加に継ることになった。

経路制御上の効率を改善するため、CIDR — Classless InterDomain Routing — が提案された [14]。インターネットでは、経路制御の単位はネットワークであり、その大きさはアドレスのビットパターン自身によって決まっていた。つまり、アドレスのどのビットがネットワーク部分を表現するかは、アドレス自身の符号化に含められていた。CIDR は、どのビットがネットワーク部を表現するかという情報を別途搬送することにより、従来の Class A、Class B、Class C といわれていた固定したアドレスクラスの概念を破棄し、任意のビットまでをネットワーク部とすることを可能にした。

さらに、32bit の IP アドレスのネットワーク部は左から連続していなければならないとし、宛先アドレスに対して、何ビットがネットワーク部を表現しているかという情報（プリフィックス長）によって、経路のカバーしているアドレス領域を表現することにした。今日行われている `133.4.0.0/16` という表現は、宛先 `133.4.0.0` の先頭 16bit がネットワーク部であることを示している。この記法に従えば、従来 Class A アドレスと呼ばれていた空間は、`0/1` と表現することができるし、同様に Class B 空間、Class C 空間はそれぞれ `128/2`、`192/3` と表現することができる。

1.2 経路制御プロトコル

CIDR 環境に対応するため、宛先にプリフィックス長を添付して経路を表現し、伝搬することができるように、経路制御プロトコルが修正された。OSPF version 2 は、もともと宛先アドレスにネットワークマスクを添付していたため、若干の修正を行っただけで CIDR に対応することができた [25] が、BGP は新しいバージョン BGP-4 にすることにより対応した [26]。また、RIP はマスク情報を搬送できるように RIP-2 が開発された [27]。

1.3 経路検索

一方、ルータにおけるパケットの経路決定では、例えば、4.3BSD UNIX Tahoe Release では、経路表はハッシュを用いて構成されており、宛先アドレスに対してハッシュ値を計算し、該当する候補の検索を行う。この場合、宛先アドレスの 32bit 全体に対して検索を行い、該当する経路が発見されなかった場合には、宛先アドレスのネットワーク部を取り出して、再度検索を行っていた。ハッシュによる方法は実装は容易であるが、経路数が増えた場合、ハッシュバケツ数を増やすことができない場合には、経路検索のオーバヘッドが増加してしまう欠点がある。

Classless な経路制御を行う場合、宛先アドレスからそのネットワーク部を取り出すという操作を容易に行うことはできない。また、一つの経路でカバーされる複数の宛先に対してハッシュ関数を適用した結果が同じでなければならないという制約があるため、ハッシュを単純に適用することはできなくなる。

このような状況で有効な方法は、経路表を木状に構成することである。木で表現した場合、ある経路でカバーされるアドレスは、その経路に対応するノードの部分木に含まれることになる。基本的には、アドレスのビットパターンに対応して木を構成しておき、宛先アドレスのビットパターンを参照することによって木を検索することになる。

木状に経路表を構成した場合のもう一つの利点は、経路数 n が増加した場合でも、検索時間は $O(\log n)$ 程度しか増加しないことである。そのため、経路数が 30,000 を越えるようなインターネット環境下での経路検索には非常に利点が多い方式であるということができる。

4.3BSD Reno Release では、長い OSI アドレス検索を効率よく行うために、Radix Tree による経路表が導入された。経路表を木状に構成した場合、仮に全てのビットで比較を行ったとしても、比較は高々アドレスのビット数だけ行えば十分である。22 Octet ある OSI アドレスでは、比較演算が比較的高価であるため、ノードでの比較は全体ではなく、特定のビット部分のみを行うように工夫されている。

しかし Radix Tree では、CIDR ではもはや意味をなさなくなったアドレスマスクの 1 のビットが連続していない、いわゆる不連続マスクにも対応するため、アルゴリズムが複雑で難解になったという問題点があった。

1.4　WIDE Internet と Classless な経路制御

Cisco や Proteon、3Com、Baynetworks などの専用ルータはソフトウェアのバージョンアップにより Classless な経路制御が可能になってきた。しかし WIDE Internet では、研究上の必要性により、このような専用ルータのみを使うのではなく、ワークステーションや PC に `gated` で経路制御を行うルータ機能も実現することが行われてきた。PC UNIX の場合、ほとんどが 4.4BSD に基づいたものになっており、Radix Tree が実装されているため、classless な経路制御を行う上では問題はない。

WIDE Internet では、Sun ワークステーションをルータとして使用している箇所も多い。SunOS 4.1.x のネットワーク部分は概ね 4.3BSD Tahoe 相当であり、経路制御の部分はハッシュを用いたものになっている。これが WIDE Internet 上で classless な経路制御を行う上で問題になった。

SunOS で Classless な経路制御を行うためには、

1. 4.3BSD Reno あるいは 4.4BSD の Radix を移植する
2. Radix とは別の機構を実装する

という２つの方法がある。前者については、4.3BSD Tahoe と 4.3BSD Reno では、ネットワーク関係の部分は相当修正が行われており、`sockaddr` や `mbuf` 周りが変更されていたり、経路の操作は従来の `ioctl` から routing socket を用いるようになっていたりしており、移植は面倒であると考えられた。

Radix はネットマスクのビットが連続である必要はないため自由度は高いが、構造や、検索アルゴリズムにバックトラックを伴うなど複雑であるという難点があった。CIDR のようにネットマスクのビットは連続であるような環境では、もっと単純な構造で classless な経路制御が実現できると考え、次に述べる Radish と呼ばれるパッケージが開発された。

SunOS 4.1.x で提供されているハッシュによる経路表は、わが国のインターネットの発展による経路数が増加した場合に、経路検索のオーバヘッドが著しく増加するという問題も引き起こした。バイナリで提供されている OS のコードは、経路全体を８つのバケツにハッシュしており、そのため、経路数 n に対して、経路が経路表に含まれている場合で、平均 $n/16$ 回の比較が必要である。経路の集成を行う直前の 1995 年 8 月のわが国のインターネットの経路数は約 3,000 であったため、経路を発見するまでに平均約 190 回の比較操作が必要であった。このように、経路数が数十を越えるような環境では、経路数の大きさに比較操作の回数が比例する方法は不利であり、木構造のような経路数の対数のオーダの比較で済む方法が望まれた。

第 2 章

Radish

Radish は TRIE 構造を用いた classless な経路制御パッケージである。Radish は不連続なネットマスクをサポートしない代わりに、単純で Radix よりも高速である。以下、Radish の構造と検索、評価について述べる。

2.1 Radish の基本的な考え方

木構造の経路表を考える場合、IP アドレスは 32bit あるため、32 段の 2 分木を考える。ルートノードは最上位ビットに対応し、一番下のノードは最下位ビットに対応し、ビットが 0 の場合には左の枝、1 の場合には右の枝を選ぶことにする。この木構造には $2^{33}-1$ 個のノードが存在し、2^{32} 個の最下位ノードが存在する。各ノードには、左右の枝の他、そこに経路が存在しているかどうかを示すフラグと、次ホップアドレスやインターフェースなどの経路情報へのポインタが付いていることにする。この構造では、各 IP アドレスに対して一つの最下位ノードが対応しており、2^{32} 個の IP アドレス全てに対する、いわゆるホスト経路を表現することができる。

このような構造を作成し、全てのノードの経路フラグをクリアしておく。そして、例えば `133.4.0.0/16` という経路を加える動作は、ルートから `133.4.0.0` に対応する最下位ノードに至るパス上の 16 レベル目のノードの経路フラグを 1 にし、必要な情報へのポインタを付け加えることによって実現することができる。`0.0.0.0/0` といういわゆる default 経路がある場合には、ルートノードに経路をぶら下げる。`133.4.0.0/32` という宛先は、`133.4.0.0` に対応する最下位ノードに経路をぶら下げる。

経路の検索は、ルートから宛先アドレスに対応して木を探索することで行うことができる。パス上に経路フラグが 1 のノードがあった場合には、それは経路の候補であるため、その位置をメモしておく。そして、木を上から下に探索し、最下位ノードに達した時点で検索は終了する。検索終了時に集まった経路の候補から、適当なポリシに従って経路を選択すればよいことになる。

一般的な経路選択ポリシは、

> マッチする経路のうち、マッチしているビット数の最も多いものを選択する（best-match）

というものである。この場合、経路の候補を一つだけ管理しておき、新たな候補が発見された場合には、候補情報を更新することで、このポリシを実現することができる。

2.2 Radish の構造

上の構造は $2^{33}-1$ 個のノードがあるため、計算機上に実装することは現実的ではない。経路制御を考える場合、ほとんどのノードの経路フラグは 0 であり、国内のインターネットだけを考えると約 2,000 個、国際的なインターネットの経路制御を考える場合でも、約 35,000 個しか経路フラグは 1 にならない。そのため、

> 下位ノードに経路フラグが 1 になっているノードがない場合には、その部分木を管理する意味はないので、削除する

ことによってメモリおよび検索の手間を大幅に減らすことができ、計算機上に実装可能になる。

また、経路の分布は割り当て方式に依存しているため、ランダムではない。そのため、特定の枝に集中し、その他の枝には経路がないような部分木が現れることが多い。そのため、経路がない部分木に対する分岐を省略することもできる。省略によって、分岐のテストを行うビットがまばらになり、探索中に発見した経路フラグが 1 になるノードが対応する経路ではない場合が存在する。そのため、探索しているパスが経路に対応しているかどうかのチェックを行う必要が生じることに注意が必要である。

表 2.1: 経路表の例

route	mask
0.0.0.0	0x00000000
133.4.0.0	0xffff0000
133.5.0.0	0xffff0000
133.5.16.0	0xffffff00
133.5.23.0	0xffffff00

一例として、表 2.1 に示す経路表を考える。この経路表に対応する Radish 木は図 2.1 のようになる。図 2.1 において、“*” が付されているノードは経路フラグが 1 になっていることを示し、実際の経路に対応している。“*” の付されていないノードは、木構造の維持に必要なノードで、実際の経路には対応していないことに注意されたい。

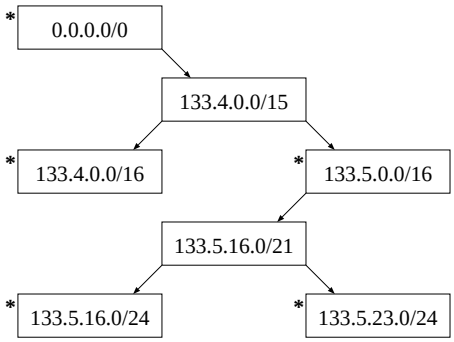

図 2.1: 表 2.1 に対応する Radish 木

2.3 探索方法と評価

Radish において、木の一部を省略しているため、探索方法に幾つかのバリエーションが存在する。それぞれについて紹介する。

2.3.1 Forward Search

Forward Search (FS) は、木を上から下へ探索する方法で、省略によってマッチしていない部分木に探索が進むのを防ぐため、各ノードで宛先アドレスのチェックを行う、もっとも直観的な方法である。

2.3.2 Skipping Forward Search

Skipping Forward Search (SFS) は、FS と同じように木を上から下へ探索する方法であるが、宛先アドレスのチェックは、経路フラグが 1 になっているノードのみで行う。このことによって、探索に失敗する場合の木の探索の手間は増加するが、アドレスの比較操作を行う回数が一般的な経路表の場合に大きく減少する（高々３回で済むことが多い）利点がある。

2.3.3 Skipping Backward Search

Skipping Backward Search (SBS) は、まず宛先アドレスの対応するビットだけをチェックして、木を上から最下位ノードまで宛先アドレスの比較を行わずに到達する。そして、経路のチェックを行う。最下位ノードが求める経路ではなかった場合には、木を上に向かって逆に辿り、経路フラグが 1 になっているノードで順次比較操作を行う。

キャンパスネットワークなどの環境では、この最下位ノードが目的の経路であることが多い。そのため、多くの場合に１回のアドレス比較で経路を得ることが可能になる。その

半面、インターネット環境で default 経路を利用している場合には、default 経路を発見するまでは木を一往復探索する操作と、数回の比較操作が必要になる欠点がある。

2.3.4 Strategic Search

Strategic Search (SS) は SFS と SBS の利点を両方生かすため、宛先アドレスに対して簡単な関数を適用し、そちらの方法を用いて探索を行うかを決定する方法である。わが国のインターネットでは、JPNIC などによる階層的アドレス割り当てが古くから行われてきたため、先頭 8bit によって経路の密度が大きく異なる。そのため、先頭 8bit をキーとするカウンタの配列を定義しておき、有効な経路それぞれに対応するカウンタを管理する。例えば、`133.4.0.0/16` という経路が追加された際には、`count[133]` を一つ増加させ、削除された際には一つ減少させる。

SS はこのカウンタ値によって、検索アルゴリズムを選択する。宛先アドレスの上位 8bit を取り出し、それに対応するカウンタ値が閾値より小さければ SFS を用い、そうでなければ SBS による。閾値が n である SS を $SS(n)$ と表現することにすると、$SS(0)$ は SBS であり、$SS(2^{24})$ は必ず SFS を選択する。

2.4 評価

Radish に於ける幾つかの探索方法を評価するため、1995 年 7 月のわが国のインターネットの経路表（BGP-4 による経路の集成が実施される前である）（JP と記す）および 1995 年 6 月の世界的なインターネットの経路表（FULL と記す）に対して評価を行った。前者は 2,855 経路、後者は 28524 経路を含んでいる。

この２通りの経路表に対して、実際に WIDE Internet で採集したパケットの宛先データから、連続する重複を除いたもの（Measured と記す）と、Class A、Class B、Class C のアドレス空間からランダムに宛先を 10,000 個生成したもの（Random と記す）の２つの宛先列を適用し、経路検索に必要な CPU 時間を測定した。ちなみに、Measured は 32,792 の長さがあり、652 の異なった宛先が含まれている。連続する重複を除いたのは、4.4BSD Lite の経路検索プログラムで、経路を一つだけキャッシュするためである。

測定は、DEC 製 HiNote Ultra CT475（Intel 486/DX4-75 を使用）の BSD/OS 2.0 のユーザプログラムとして実装された Radix および Radish に対して、JP/FULL の２通りの経路表に Measured/Random の２通りの宛先列を適用し、検索１回あたりに必要な CPU 時間を測定した。なお、表中の数字の単位はμsec である。

表 2.2 は Radix の場合および Radish FS の場合の測定結果である。JP/Random の場合には Radish FS の方が Radix より速いが、その他の場合には全て Radish FS が遅くなっている。これは、比較操作を頻繁に行うオーバヘッドのためと考えられる。

比較操作を経路フラグが 1 になっているノードに限定した Radish SFS の測定結果は表 2.3 に示す通りであるが、Measured 列を用いた場合には Radix よりわずかに遅いもの

表 2.2: Radix および Radish FS の測定時間

経路表 宛先列	JP Measured	JP Random	FULL Measured	FULL Random
Radix	13.06	18.42	12.60	8.45
Radish FS	22.29	6.32	22.24	8.66

の、Random 列に対しては高速になっている。

表 2.3: Radish SFS の場合

経路表 宛先列	JP Measured	JP Random	FULL Measured	FULL Random
Radish SFS	13.65	5.31	12.89	6.87

Radish SBS の場合、表 2.4 に示す通り、いずれの場合にも Radix より高速になっている。

表 2.4: Radish SBS の場合

経路表 宛先列	JP Measured	JP Random	FULL Measured	FULL Random
Radish SBS	10.81	6.52	10.14	7.70

Radish SS の場合、閾値によって性能は変化するため、JP/Measured の組合せに対して $SS(0) \sim SS(9)$ に対する測定値を図 2.2 に示すが、$SS(3)$ が最も高速であったため、$SS(3)$ に対して４つの組合せを実行した。その結果、表 2.5 に示すように、良好な結果が得られており、JP/Measured の場合、Radix の 1.2 倍程高速になっている。

上の全ての測定は、アドレスの比較操作を 1byte 単位で行ったものである。これは、対象プロトコルのアドレス幅が計算機の語長に依存しないようにするためであるが、現実に我々が直に必要とするのは 32bit の IPv4 アドレスを扱うものである。最近の PC やワークステーションでは、32bit のデータの比較は一命令で可能であるため、Radish FS の場合に、32bit 単位で比較を行うようにした場合の結果を表 2.6 に示す。

表 2.2 と比べて明らかなのは、各ノードで比較を行っているにも関わらず、いずれの場合でも Radix より高速であることである。測定を行っていないが、SFS などの場合、さらに短時間で検索を行うことが期待されるわけで、IPv4 アドレスを対象に考えている限り、

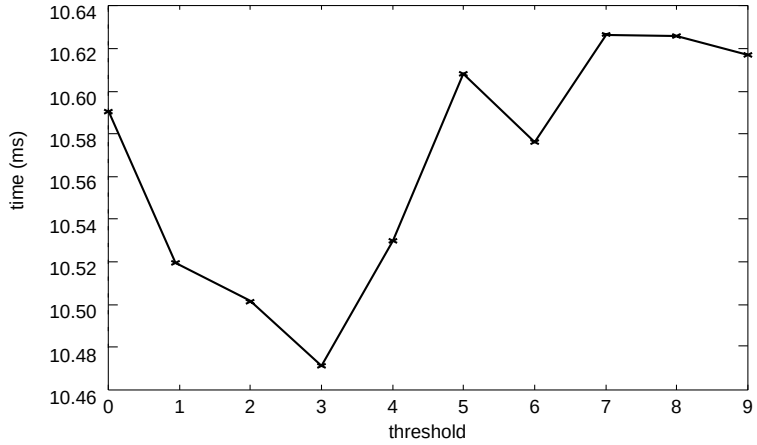

図 2.2: Radish SS の閾値による特性

表 2.5: Radish SS の場合

経路表 宛先列	JP Measured	JP Random	FULL Measured	FULL Random
Radish SS(3)	10.47	5.42	10.52	6.94

Radix を単に SunOS に移植するよりも、Radish を用いる方が経路検索オーバヘッドが少ないことになる。

表 2.6: Radish FS の４バイト比較の場合

経路表 宛先列	JP Measured	JP Random	FULL Measured	FULL Random
Radish FS-4B	9.83	3.10	10.11	4.21

2.5　単純さに関する考察

Radix の問題点は、その構造やアルゴリズムが複雑で理解が容易でない点を上げることができる。一方、Radish は不連続なアドレスマスクには対応できない半面、構造は単純であり、理解も容易である。５人の修士課程の学生に対して Radix および Radish の理解を

テストしたところ、全員が Radish を正しく理解したのに対して、完全に Radix を理解した学生はいなかった。無論、経路制御のプログラムが全ての人に理解される必要はないが、アルゴリズムの改良などを考える場合には、単純なことは好ましいことである。

また、例えば IPv6 の Path MTU discovery を考えた場合、経路は経路制御ドメインの境界で集成されるため、単純に経路表に Path MTU 情報を付加するだけでは十分ではないことが知られている。そのため、経路とは別にアドレスとマスク対から Path MTU を探索することが必要になり、Radish をこれに応用することも考えられる。

2.6 Radish の実装

Radish を SunOS 4.1.x カーネルに組み込み、Classless な経路制御を可能にするような実装が行わた。Sun Sparc では、アドレス比較を 32bit 単位で行うことが可能なため、32bit 単位で比較を行う SFS による検索が用いられている。また、sockaddr 構造体の未使用の領域を用いてマスク情報をカーネルに渡す機構も用意されてた。また、GateDaemon Consortium で開発が行われている `gated` に対するパッチも用意され、BGP-4 や OSPF を用いることによって Classless な経路制御が可能になった。

SunOS 4.1.x 上の Radish は、1995 年 7 月から WIDE Internet において運用が始まり、約一年間問題なく動作している。WIDE 福岡 NOC における 22 時間の運用において、21,291,357 パケットが通過し、48,585,537 回の比較操作が行われた。これは、一パケットにつき 2.28 回であり、また木を辿る回数としては、一パケットあたり 17.9 回ということになっている。経路数は 2,175 経路で 262KByte のメモリが Radish のノードを格納するために使用された。

SunOS 4.1.x では、mbuf 空間の上限は約 2MB という制約があるため、実際に格納できる経路数はおよそ 15,000 であり、35,000 あるインターネットの全経路を対象とした経路制御には参加することはできないが、国内の経路制御（経路数約 2,000）に対しては十分実用になる。

第 3 章

国内の経路制御

インターネットの発達がやや送れたわが国であるが、1994 年に入ると商用インターネットサービスプロバイダが運用を開始し、1994 年後半から経路数が平均して毎月 100 程度増加するようになった。そのため、JEPG/IP などの調整のもと、1995 年 9 月から経路制御ドメイン間の経路制御には BGP-4 を用いること、可能な経路は極力集成を行うことが申し合わされた。図 3.1 に国内の経路数の変遷を示すが、約 3,000 あった経路が集成により 1,600 程度に減少したことが示されている。その後新規のプロバイダの参入が相次ぎ、JPNIC のアドレス割り当て方針との関連で経路数は再び増加を見ているが、インターネットの発展が非常に急激になっている割には経路数の増加は少ないということができる。

経路の集成による効果は、経路数の減少のみに留まらず、経路の安定化にも大きく寄与している。複数の Class C アドレスの割り当てが行われるようになってから、一つの組織へのリンクがダウンした場合には、複数の経路が影響を受けることになる。また、組織内部のルータのダウンによって、その変動が国内全体に伝搬するので、経路が非常に不安定になっていることが知られていた。集成された経路は、その範囲内に一つでも経路が存在すれば、アナウンスされるため、これらの経路変動を経路制御ドメイン内に閉じ込めることができ、不安定さは大きく軽減されている。

もう一つの経路の不安定な原因は、歴史的理由により幾つかの経路制御ドメインでは、依然として RIP [28] による経路制御を行ってきた部分が多数存在したためである。経路の集成を行った場合、経路に陽にマスク情報を搬送しない RIP ではもはや正確な経路制御を行うことはできない。RIP-2 を用いればこの問題は解決するが、RIP-2 の実装は限られていたこと、RIP は 30 秒に 1 回全経路をアナウンスするため、経路数が増加した場合には送出されるメッセージ数が多くなり、受信バッファがオーバフローするという問題は依然として解消されていない。

RIP に起因していると考えられる経路の不安定さは、経路の変動が 30 秒の倍数で発生するという特長がある。経路数の増加はオーバヘッドや変動の増加に継るが、急激な変化ではないため、特に学術系のネットワークでは、遅くまでそのまま RIP による運用が行われてきた。BGP-4 による経路の集成は、質的な変化を伴うため、RIP から OSPF や BGP などの CIDR に対応し、しかもトラフィック的な影響の小さな経路制御プロトコルに移行が促進され、30 秒単位の経路の不安定さは大幅に改善された。

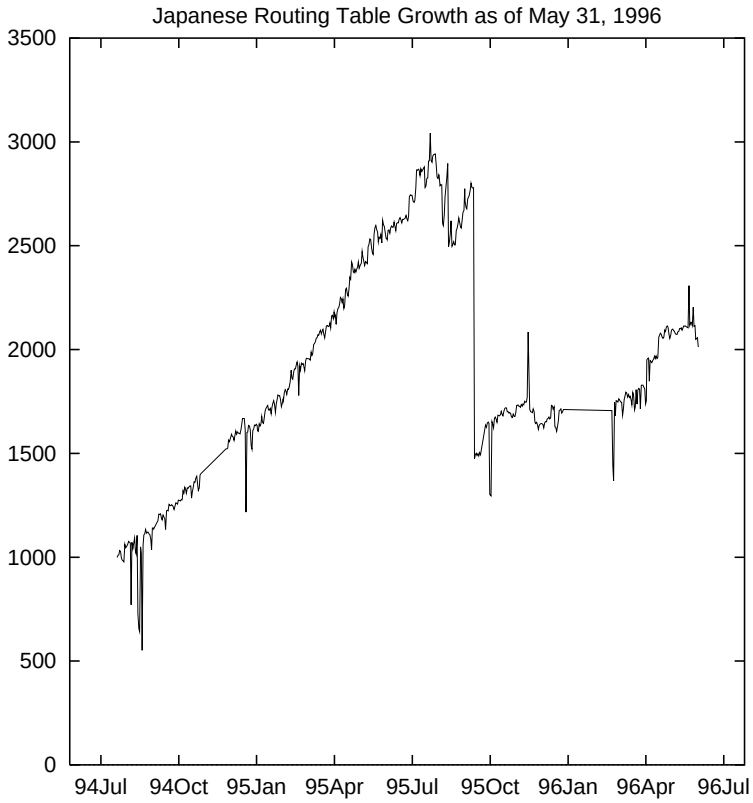

図 3.1: 国内の経路数の変遷

第 4 章

経路の診断

経路制御において、実際に経路が集成されていなかったり、集成するような設定が十分に機能しておらず、集成されていない経路がアナウンスされることもしばしばある。実時間的な経路の診断にはまだ対応していないが、BGP 経路の診断を行うプログラムが開発されている。

IGP による経路の影響を避けるため、ルータとしては機能していないワークステーションを一台割り当て、private AS 番号を割り振り、独立した AS として設定する。このワークステーション上に gated を動かし、BGP-4 によって経路を供給する。そして、適当なタイミングで、gated に対して signal を与え、内部の経路情報のダンプである gated_dump を作成する。この gated_dump を診断することにより、経路制御上の問題点の幾つかを指摘することができる。経路の検査には、Internet Routing Registry に登録された経路データベースを用い、観測された経路を対照する。

この perl で書かれた経路診断プログラムは、

- 隣接した同一 AS Path を持つ経路。この経路は集成可能である。
- 同一 AS Path をもち、かつ片方が他方にカバーされているような場合。133.4.0.0/16 と 133.4.3.0/27 のような場合であるが、これはカバーされている経路を削除することができる。
- IRR のデータベースに登録されていない経路や、IRR データベースに登録されたものとは異なった origin をもつ経路。

などを検出することができ、またプリフィックス長別の経路数の分布などの統計情報も得ることができる。

このプログラムは定期的に実行されるようにはなっていないが、毎月１回程度の割合で実装され、該当経路ドメインの管理者に警告を送り、経路制御の効率化に寄与している。

効率的な経路制御を行うためには、各プロバイダの担当者間の相互の協力関係が不可欠である。そのため、JEPG/IP の一つのワーキンググループとして経路制御に関係する担当者の情報交換のためのメーリングリスト AS-OPS が運用されており、一年に数回のミーティングも開催されている。