論　文

経路表を利用したIPパケット受信処理

神明　達哉[†]　尾上　淳[††]　山本　和彦[†††]　萩野純一郎[†††]
江崎　浩[††††]　村井　純[†††††]

A method of processing inbound IP packets with a routing table

Tatuya JINMEI[†], Atsushi ONOE[††], Kazu YAMAMOTO[†††], Jun-ichiro HAGINO[†††], Hiroshi ESAKI[††††], and Jun MURAI[†††††]

あらまし　インターネットの普及が進み，また機器の性能が向上するにつれて，一台のノードに多数のIPアドレスを割り当てるという運用が一般化してきた．このようなノードにおいては，受信パケットが自ノード宛であることを効率よく判定する必要がある．従来，BSD系システムでは，自ノードのアドレスを線形に探索するアルゴリズムを用いてきた．一方，本論文で述べるように，WIDEプロジェクトでは，経路表のアルゴリズムを利用する改良手法を研究開発した．その後，NetBSDやBSD/OSのIPv4実装において，ハッシュ表による独自の改良も導入された．本論文では，この三つの方法について説明し，その性能を実機によって評価する．その評価結果から，経路表及びハッシュ表による判定手法が，ともに線形探索による従来の方法に対する優れた改良であることを明らかにする．また，経路表のアルゴリズムを利用した判定法が，一般の経路検索時と同様に，アドレス数が増加しても性能が劣化しにくい特性を持つことも示す．

キーワード　BSD，WIDEプロジェクト，KAMEプロジェクト，経路表，IPv6

1. 多数のアドレスを持つIPノード

インターネット普及の初期の段階では，一つのノードに割り当てられるアドレスの個数は少数であった．現在インターネットの標準プロトコルとして使用されているIPv4では，一つのネットワークインタフェース(IF)には高々一つのIPアドレスのみを割り当てるのが原則であり，また，一般にホストの持つIF数は少ないためである．

†株式会社東芝 研究開発センター，川崎市
Research and Development Center, Toshiba Corporation, 1 Komukai Toshiba-cho, Saiwai-ku, Kawasaki-shi, 212-8582 Japan
††ソニー株式会社，東京都
IT Development Division, NSC, Sony Corporation 6-7-35 Kitashinagawa, Shinagawa-ku, Tokyo, 141-0001 Japan
†††IIJ技術研究所，東京都
Research Laboratory, Internet Initiative Japan Inc. Takebashi Yasuda Bldg., 3-13 Kanda Nishiki-cho Chiyoda-ku, Tokyo, 101-0054 Japan
††††東京大学 大学院 情報理工学系研究科，東京都
Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
†††††慶應義塾大学 環境情報学部，藤沢市
Faculty of Environmental Information, Keio University, 5322 Endo, Fujisawa-shi, 252-8520, Japan

ルータは複数のIFを持つが，初期のルータにおいては，性能上の制約から，あるいは扱うネットワークの規模が小さいことから，IF数は多くても数十であった．このため，ルータに割り当てられるIPアドレスの個数も同程度に抑えられていた．

しかし，インターネットの普及が進み，また機器の性能が向上するにつれて，ホストかルータかを問わず，一台のノードに多数のIPアドレスを割り当てるという運用が一般化してきた．

例えば，一台のサーバに複数のIPアドレスを割り当て，外部に対しては複数台のサーバのように振る舞う，仮想ホスト機能を考える．仮想ホストの目的の一つは，一台の物理ノードに複数サーバの機能を集約させることであり，集約が進むほど，ノードには多くのアドレスが割り当てられることになる．実際，BSD/OSバージョン4(以後，単にBSD/OSと表記する)では，仮想ホスト機能を想定して，出荷時の設定で最大8000個のIPv4アドレスを持つ環境に対応している．

ルータにおいても，高機能化やネットワークの大規模化に伴い，数百以上の物理IFを持つ製品が存在する．また，物理的には少数のネットワークデバイスしか持たないノードであっても，多数のIFとアドレスを

持つ場合もある．例えば，仮想 LAN 機能により，少数の物理 IF を持つ一台のルータが数千個のサブネットを管理し，同程度の数の IP アドレスを持つこともあり得る．

商用ルータ以外で，多数の IF 上に多数のアドレスを持つルータの例としては，カナダの Viagénie 社が提供する Freenet6 (注1) と呼ばれるサービスのサーバが挙げられる．このサーバは，不特定多数のユーザに対して，IPv6 パケットを IPv4 パケットにカプセル化するトンネル機能 [1] によって IPv6 ネットワークへの接続性を提供している．

Freenet6 のサーバは，KAME プロジェクト [2] の IPv6 実装を持つ BSD 系システムの PC ルータである．この実装では，ソフトウェアによってトンネル用の仮想 IF を実現する．Freenet6 のサービスで必要となるトンネル数を確保するため，このサーバは数千個の仮想 IF を持つ．また実際に，そのすべての IF に常時グローバルアドレス [3] が割り当てられ，トンネルサービスを提供している．

本論文では，多数のアドレスを持つノードにおいて，受信パケットが自ノード宛であるかどうかを効率よく判定するための手法について述べる．まず，BSD 系のシステムにおける従来のアルゴリズムである線形探索，NetBSD や BSD/OS で採用されたハッシュ表による方法，及び，筆者らが WIDE プロジェクトで開発した，経路表を利用する方法のそれぞれについて説明する．次に，各判定方法の性能を実機を用いて評価した結果を示す．

2. 受信パケットの宛先判定手法

BSD 系システムの実装では，パケットを受信した際に，それが実際にそのノードを宛先としているかどうかを判定する．その結果，そのノード宛のパケットであれば受理し，更に上位の層で処理する．そうでない場合，ノードがホストとして動作するならパケットを破棄する．ルータとして動作する場合には，次ホップノードのアドレスを経路表から求め，そのノードに向けてパケットを転送する．

したがって，前節で述べたような，多数のアドレスを持つノードにおいては，受信したパケットが自ノード宛であるかどうかの判定アルゴリズムが，受信や転送処理の性能に与える影響が大きくなる．

(注1)：http://www.freenet6.net/

BSD 系のシステムでは，従来，この判定方法として，自ノードの持つアドレスを線形に探索するアルゴリズムを用いてきた．一方，1997 年 10 月に，筆者らが WIDE プロジェクトにおいて開発していた BSD 系システム上の IPv6 実装では，経路表による判定手法を導入した．この実装は KAME プロジェクトに引き継がれ，現在，BSD 系システムの正式な IPv6 スタックとして，WIDE プロジェクトが運用する IPv6 ネットワークをはじめ，世界的に広く利用されている．IPv4 スタックについては，1998 年に，NetBSD や BSD/OS においてハッシュ表によるアルゴリズムを用いた独自の改良が加えられた．

この節では，これら三つの方法について，その特徴を考察する．

2.1 線形探索による判定法

BSD 系システムは，ネットワーク層のアドレスをリスト構造で管理する．先に述べたように，従来の BSD 系システムの IPv4 実装では，受信パケットが自ノード宛であるかどうかを判定するために，このリストを線形に探索し，パケットの宛先アドレスと比較するという方法を用いてきた．

この方法の場合，自ノード宛の場合には最悪でアドレスの個数回の比較が必要となる．ルータとして動作する場合，ほとんどのパケットは他のノード宛であるため，多くの場合に最長回数の比較が必要になる．また，ホストとして動作する場合，宛先アドレスに偏りがなければ，平均的にはアドレスの個数に比例した比較回数を要する．

2.2 ハッシュ表を利用した判定法

NetBSD や BSD/OS では，ノードの持つアドレスを，リスト構造と同時にハッシュ表にも保存することによって，受信したパケットに対する判定を高速化している．これらの実装では，受信パケットの宛先アドレスに対するハッシュ値を求め，その値に属する各アドレスと宛先アドレスを比較する．

この方法の場合，受信アドレスが自ノード宛であることを判定するまでに必要な時間は，ハッシュ値の計算と，そのハッシュ値に属する各アドレスと受信パケットの宛先との比較にかかる時間の合計である．ここで，ハッシュ値の計算に必要な時間は，宛先アドレスやノードの持つアドレス数に依存しない一定値であるとみなせる．また，ハッシュ計算のアルゴリズムを工夫してハッシュ値の偏りを少なくすれば，アドレスの比較回数はアドレス数をハッシュのエントリ数で

割った値に近づく．更に，ハッシュ表のサイズをアドレスの数に比べて十分に大きく取ることで，比較回数を一回以下に近づけられる．

2.3 経路表を利用した判定法

2.3.1 縮退二分木による経路検索

BSD系のシステムでは，ネットワーク層の経路表として縮退二分木 (radix tree) [4] を用いる．縮退二分木では，各葉ノードが経路表の一つのエントリに相当する．また，木の中間ノードには，検索の鍵となる宛先アドレスに対する，ビット判定のためのオフセット値が格納される．縮退二分木による経路検索では，与えられた宛先に対して，根ノードから葉ノードに至るまでの各中間ノードにおけるビット判定と，葉ノードにおけるアドレス比較によって対応するエントリを決定する．

ここで，葉ノードにおける比較に失敗した場合には，上位のノードまで遡り，宛先アドレスに対して適切なマスクを施した上で検索を継続する．この操作はバックトラックと呼ばれ，集約された経路から派生した経路の存在によって一時的に失敗した検索を修正する働きを持つ．特に，デフォルト経路，すなわちプレフィクス長が0の経路にのみ適合する宛先に対して検索した場合には，通常最低一回のバックトラックが発生する．

送信ノードは，パケットの宛先アドレスに対する送信IFと，更に必要なら転送先のゲートウェイを決定するために経路表を検索する．これは宛先が送信ノード自身である場合にも適用され，この場合，送信IFにはループバックIFと呼ばれる仮想的なIFが利用される．ループバックIFに出力されたパケットはそのまま同じノードの受信処理に渡る．

2.3.2 経路検索の受信処理への応用

BSD系のシステムにおいて，縮退二分木による経路表は，十分な実績を持つ安定した実装である．筆者らは，これを受信処理に応用すれば，新たな機能を導入することなく受信処理の効率を向上させられると考え，実際にIPv6の実装に適用した．この方法では，受信処理において，宛先アドレスに対して経路表を検索し，送信IFがループバックIFかどうかによって自ノード宛であるか否かを判定する．

この場合，判定に必要な操作は経路表検索に必要な操作に等しい．つまり，葉ノードに至るまでの中間ノード数分のビット判定と，葉ノードにおけるアドレス比較である．更に，バックトラックが発生した場合には，その分のビット判定とアドレス比較が必要となる．

ここで，葉ノードに至るまでのビット判定回数は，高々アドレスのビット長分，すなわち，IPv6においては高々128回である．また，二分木が縮退した構造を持つことによって，この回数は実際上は上限値の数分の一程度に収まる場合が多い（3.2節参照）．葉ノードにおけるアドレス比較は，重複した検索鍵を持つ特殊なエントリの場合を除けば一度でよい．更に，自ノード宛の場合は普通バックトラックは発生しないため，判定に必要な時間はアドレス数や外部経路に依存しない一定の上限に収まると期待できる．

ルータとして動作する場合には，受信パケットの多くは他ノード宛であり，転送処理を必要とする．この場合は，受信処理で求めた経路表のエントリをそのまま転送用途に利用できるため，転送時に再度経路表を検索する必要はない．このときの経路表検索では，バックトラックによって，ビット判定とアドレス比較の回数が増加し得る．ただし，これはルータにとって本来必要な処理であり，受信パケット処理で経路表を検索したために発生した負荷ではない．

3. 判定方法の評価

この節では，前節で述べた各判定手法について，その性能を実機を用いて評価した結果について説明する．

3.1 評価方法

評価に用いたノードは，Pentium III 866Mhz のCPUを持ち，オペレーティングシステムとしてFreeBSD 4.4を用いたデスクトップパソコンである．2.節の冒頭で述べたように，FreeBSD 4.4のIPv6スタックには，すでに経路表による判定手法が採用されている．

今回の評価のために，FreeBSD 4.4のIPv6スタック上に，線形探索による従来の方法と，ハッシュ表による改良手法を追加で実装し，三種類の判定手法を切り替えて使用できるようにした．更に，それぞれの方法で判定に要した時間をPentiumプロセッサのクロックカウンタによって測定及び記録するコードを追加した．

ハッシュ表による判定手法は，BSD/OSのIPv4実装での方式を踏襲し，以下のように実装した．与えられたIPv6アドレスに対するハッシュ値は，アドレスの上位から32ビットずつ4つの部分に分けてその排他論理和を取り，それをハッシュ表のエントリ数で割っ

て求める．128 ビット全体の値を利用するのは，一部のビットを共有する複数のアドレスが同一のハッシュ値を持つことを防ぐためである．エントリ数としては，ハッシュ値の偏りを防ぐために，一般に素数を用いる．ここでは，BSD/OS の大規模サーバ向け規定値である 997 を流用した．

WIDE プロジェクトの実装では，経路表の検索結果をキャッシュとして保存する．受信処理では，経路表を検索する前にパケットの宛先アドレスとキャッシュされた経路エントリの宛先アドレスを比較し，これが一致していればキャッシュされた経路をそのまま利用する．このため，同一宛先のパケットを連続して受信している場合には，経路表の検索は最初の一度のみで済む．一方，NetBSD や BSD/OS でのハッシュ表による実装では，ハッシュ表の検索結果は保存されず，すべての受信パケットに対してハッシュ値を計算し，ハッシュ表を検索する．そこで，評価結果が公平になるよう，評価用に実装したハッシュ表による判定手法では，ハッシュ表の検索結果もキャッシュとして保存するように変更した．

BSD 系のシステムは，ホストとしてもルータとしても動作するが，ここでは，以下の理由から，ホストとしての動作時に限定して性能を測定した．ルータの受け取るパケットの多くはルータ自身を宛先とすることはなく，他のノードへ転送される．したがって，線形探索やハッシュ表による判定手法を用いた場合，まず自ノード宛でないことを判定した上で，転送先を求めるために経路表を検索する，という手順が典型的な操作となる．一方，経路表を利用した場合には，受信時の検索結果を転送時に流用できるため，自ノード宛でないことを判定するための処理は実質的には不要である．つまり，ルータの場合の典型的な挙動においては，経路表による方法の性能が最善であることは自明である．

評価対象ノードには，ループバック IF 上に複数のグローバルアドレスを割り当てた．これは，仮想ホスト機能を実現するための典型的な設定方法である．アドレスの個数による性能の変化を調べるために，割り当てるアドレスの数を 100 から 10000 まで 100 個おきに変化させ，それぞれの場合について下記の方法で測定した．

割り当てたグローバルアドレスの中から一つを乱数で選び，外部ノードからそのアドレス宛にパケットを送信する．これを 10000 回繰り返し，各回について，評価対象ノードがそのパケットを自ノード宛であると判定するまでの時間を記録する．この測定では，経路やハッシュ表検索結果のキャッシュ効率がゼロになる環境での性能を比較するため，同一の宛先が連続しない送信パケット列を用いた．

次に，経路やハッシュ表検索結果のキャッシュ効率が性能に与える影響を調べた．ここでは，アドレス数が 100，500，1000，5000，10000 の各場合について，送信パケット列において同じ宛先が連続する確率を 0%から 10%おきに 100%まで変化させ，同様に測定した．評価対象のホストは，評価用以外のパケットはほとんど受信しないので，この確率をキャッシュの的中率とみなせる．

3.2 評価方法に対する考察

測定に際し，どの判定手法にとっても評価条件が妥当であることを確認するために，評価に用いた各設定について，対応するハッシュ表と経路表の構造を調べた．

経路表による方法については，縮退二分木における葉ノードの深さの分布を求め，木構造の偏りを調べた．図 1 は，自ノードのアドレスに対応する経路エントリの，経路表における深さについて，その最大（Max），最小（Min），平均値（Ave）をアドレス数を変化させて調べた結果である．

図からわかるように，最大値と最小値には開きがあるが，平均的にはほぼ最大の深さに等しい構造となっている．したがって，パケットの宛先に偏りがなければ，評価結果は公平であると考えて良い．

ハッシュ表による判定手法については，ハッシュ値の分布に偏りがあるかどうかを調べるために，各ハッシュ値に属するアドレス数の最大値（Max）と最小値（Min），更に空でないハッシュ値に属するアドレス数の平均（Ave）を求めた．

図 2 に，アドレス数ごとのこの結果をグラフで示す．また，この図では，理想的なハッシュ表，すなわち各エントリに同じ個数のアドレスが属し，空のエントリのないハッシュ表における一エントリ辺りのアドレスの個数（Ideal）も併せて示してある．この図によれば，最大値と最小値，及び平均値の間には差が見られる．しかし，平均値が理想的なハッシュ表の値に近いことから，パケットの宛先に偏りがなければ公平な結果を得られるはずである．

3.3 評価結果

3.1 節で述べた評価方法によって得られた各結果に

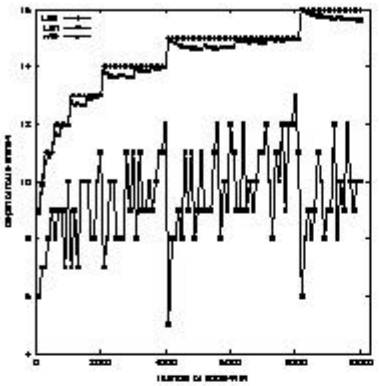

図1 範退二分木の深さの変化
Fig. 1 The depth of the radix tree.

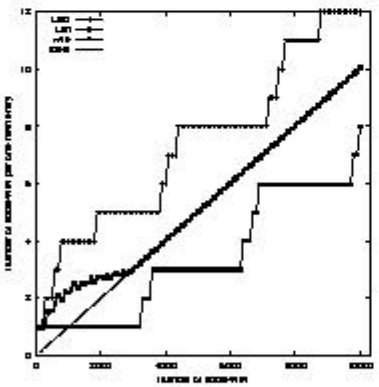

図2 ハッシュ表の構造
Fig. 2 The structure of the hash table.

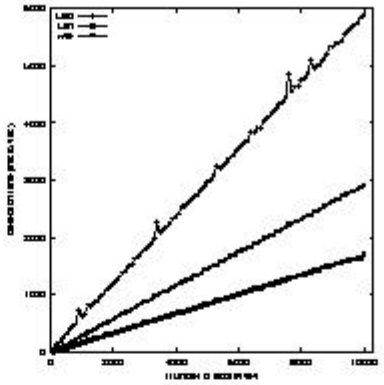

図3 線形探索の場合の評価結果
Fig. 3 Evaluation results using the linear search algorithm.

ついて，その最大値と平均値，標準偏差を求めた．なお，判定時間は，測定値をCPUのクロック数で割ることで実時間（マイクロ秒）に変換している．

判定手法として線形探索を用いた場合の結果を図3にグラフで示した．この図からわかるように，この場合には，最大値（Max），最小値（Min），平均値（Ave）のいずれもアドレスの個数にほぼ比例した判定時間を必要とする．

一方，図4には，同一の宛先が連続しない，すなわち経路をキャッシュする効果が得られない送信パケット列に対して，ハッシュ表及び経路表による判定手法を利用した場合の結果をまとめた．"Routing table" で示される折れ線は経路表の場合の結果を，"Hash table" で示される折れ線はハッシュ表の場合の結果を，それぞれ表す．また，"ave"，"max"，"std. deviation" は，それぞれ平均，最大，標準偏差を意味する．

まず，図3と図4の比較により，線形探索に比べると，ハッシュ表及び経路表を利用する判定手法の性能は明らかに良いことがわかる．

次に，図4の結果から，ハッシュ表及び経路表による方法の性能は平均的にはほぼ同等であると言える．

厳密には，アドレス数に対するハッシュ表のエントリ数が十分大きい場合には，ハッシュ表の方がやや性能が高い．ただし，経路表による判定手法の場合には，アドレス数の増加に対して判定時間の増加速度が緩やかであるという特徴がある．実際，経路表の二分木としての性質と，ホスト上に設定された経路表のエントリ数がそのホストに割り当てられたアドレスの個数にほぼ等しいこととから，経路検索に必要な時間，すな

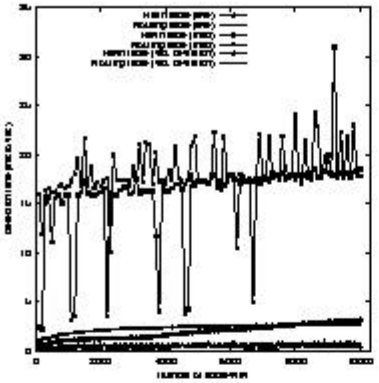

図4 ハッシュ表と経路表利用の場合の評価結果
Fig.4 Evaluation results using the hash table and the routing table.

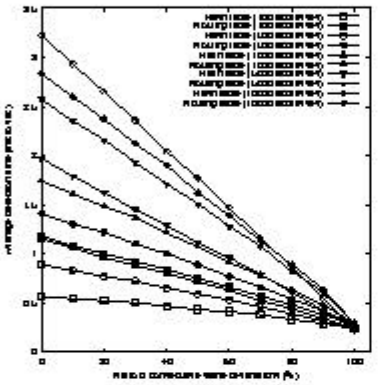

図5 同一宛先アドレスが連続する確率を変化させた場合の評価結果（平均値）
Fig.5 Evaluation results per the ratio of same consecutive destination addresses in the test packets (average).

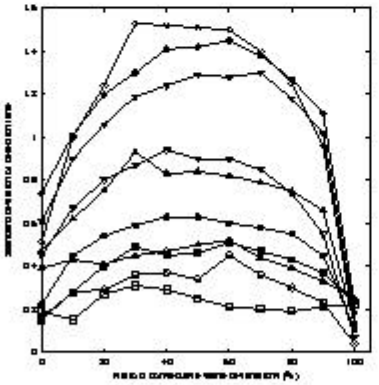

図6 同一宛先アドレスが連続する確率を変化させた場合の評価結果（標準偏差）
Fig.6 Evaluation results per the ratio of same consecutive destination addresses in the test packets (standard deviation).

わち判定時間はアドレス数に対する対数オーダで増加すると予想される．図4は，この予想を裏付ける結果を示している．

図4によれば，最大値にはばらつきが見られるが，標準偏差が小さいことから，ほとんどの場合は平均に近い値を取っていることがわかる．すなわち，平均値がノードの全体的な性能を代表していると考えて良い．

最後に，送信パケット列中で同一宛先が連続する確率を変化させた場合の結果を図5及び図6に示す．図5は判定手法とアドレス数ごとの判定時間の平均値を，図6は対応する標準偏差を表したグラフである．図6の各折線は，図5における同じ型の折線に対応している．

図5により，キャッシュの効果は判定手法によらずほぼ同等に得られていると言える．すなわち，キャッシュの効果を考慮しても，2つの手法の性能はほぼ同等と考えて良い．

ところで，図6から，特にアドレス数が大きいときに，同一宛先が連続する確率が中程度であると，判定時間の分布に広がりがあることがわかる．これは，キャッシュが的中して判定時間が小さい場合と，それ以外の場合で検索に時間がかかる場合との値に開きがあったためと考えられる．ただし，図6が示す範囲であれば，図5の結果，すなわち平均値が全体的な性能の代表と考えて良いであろう．

4. 結　　論

筆者らがWIDEプロジェクトで開発したIPv6の実装では，受信したパケットが自ノード宛であるかどうかを判定するために，縮退二分木による経路表検索を利用している．すなわち，宛先アドレスに対する送信IFがループバックインタフェースであれば自ノード宛のパケットであると判定する．

本論文では，従来のBSD系システムで採用されてきた線形探索による判定法と，NetBSDやBSD/OSによる改良であるハッシュ表を用いた判定法，及び経路表による方法のそれぞれについて説明した．

次に，ホストとして動作する設定のもとで，実機を用いて各判定手法の性能を評価し，比較した．評価結果から，線形探索による従来の方法と比較すると，ハッシュ表及び経路表を用いた改良手法が十分に優れていることを示した．一方，ハッシュ表及び経路表による判定手法の性能はほぼ同等であった．この結果は，ハッシュ表や経路表の検索結果をキャッシュする実装

において，キャッシュ効率を変化させても同様である．

厳密には，ハッシュ表による方法は，ノードの持つアドレス数に対してハッシュ表のエントリ数を十分に大きく取れば，経路表による判定手法よりもやや良い性能を持つ．これに対し，経路表を利用した方法には，ノードの持つアドレス数が増えても性能が劣化しにくい特性がある．

一方，ルータとしての動作時には，経路表による方法が最善であることは明らかである．ルータ宛のパケットのほとんどは他のノードへ転送される．この場合，自ノード宛であるかどうかの判定手法の種類によらず，経路表を検索する必要がある．したがって，自ノード宛であるかどうかの判定に経路表を利用すれば，その判定にかかるコストは実質的に無視できるためである．

また，経路表を利用する場合には，実装コストの上でも利点がある．縮退二分木による経路表は，BSD系システムにおいては十分に実績のある安定した実装であり，この実装を受信時にも利用することで，新たな機能を導入せずに宛先アドレスの判定効率を向上させられるからである．

以上の点から，経路表を宛先アドレス判定に利用する手法は，線形探索を用いた従来の方法に対する改良として妥当であると言える．

謝辞 WIDEプロジェクトIPv6分科会のメンバー，特にKAMEプロジェクトのメンバーには，実装へのコメント，運用による効果の検証を通じて協力していただきました．また，Marc Blanchet氏からは，Freenet6サーバの構成を紹介する許可をいただきました．これらの方々のご協力に感謝致します．

文　献

[1] R. Gilligan and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers," RFC 2893, August 2000.

[2] 神明達哉，山本和彦，萩野純一郎，江崎浩，村井純，"KAMEプロジェクトによるIPv6基本ソフトウェア開発，" 情報処理，第41巻，第12号，pp. 1367–1372, December 2000.

[3] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 2373, July 1998.

[4] K. Sklower, "A Tree-Based Packet Routing Table for Berkeley Unix," USENIX Winter 1991, pp. 93–104, Dallas, USA, January 1991.