動的ホスト設定プロトコル (DHCP) の実装と評価 †

冨永 明宏‡
慶應義塾大学 環境情報学部

寺岡 文男§
ソニーコンピュータサイエンス研究所

村井 純¶
慶應義塾大学 環境情報学部

計算機をネットワークに接続する際に必要となる種々の設定を自動化するプロトコルとして、「動的ホスト設定プロトコル (DHCP) 」が提案された。本論文では、最初に現在のインターネットにおいて DHCP が重要な役割を果たすことを明らかにし、次に DHCP の実装上の問題点を考察する。そしてそれらの問題点の解決法を提案し、それに基づいた実装を行なう。この結果、我々は機種独立性の高い DHCP の実装を行なうことができた。この実装は現在 BSD/386 および NEWS-OS 上で稼働している。

1 はじめに

この数年で計算機の小型軽量化・高性能化が非常に進み、そのオペレーティングシステムとして BSD/386 、NetBSD 等の UNIX が利用できるようになった。それに加えて、非常に小型のネットワーク機器が登場したことにより、ラップトップやノートブックと呼ばれる携帯型計算機でも、デスクトップワークステーションと同等の環境を実現できるようになった。このような計算機を持ち運び、移動先でインターネットに接続する場合、いかにして IP アドレスを取得するかという問題が発生する (「一時アドレス割り当て問題」) 。その場合、移動先のネットワークの管理者に直接連絡を取ってアドレスを割り当ててもらったり、適当に空いているアドレスを不当に使用したりするか、さもなければそのサイト独自のメカニズムによってアドレスを割り当てるというようなことが行なわれている。

さらに携帯型計算機や新しいコンピュータをネットワークに接続する際の設定作業は従来手動で行なわれており、ネットワーク規模の拡大に伴って作業量の増大や誤りの発生などの問題が生じている。したがって、計算機をネットワークに接続するだけで自動的に計算機の設定が行なわれるような技術が必要になってきている (「計算機自動設定問題」) 。

またこれとは別に、昨今では IP アドレスの枯渇が叫ばれている (「IP アドレス枯渇問題」) 。しかし実際には割り当てられていても未使用のままのアドレスが多数存在し、それらのアドレスを回収する技術や複数のノードで一つの IP アドレスを共有する技術があれば、IP アドレス枯渇問題はある程度まで回避できる。しかし、このための標準的な技術は存在していない。

これら 3 つの問題を解決するため、動的ホスト設定プロトコル (DHCP: Dynamic Host Configuration Protocol)[4] が提案された。DHCP は、IP アドレスなどの共有資源の統一的かつ自動的な管理機構を提供するものである。次節では、まずこれらの問題に対する既存の手法を紹介し、その問題点を明らかにする。その後の節では、DHCP の概要解説、実装設計、実装解説、実装評価、結論の順に話を進めていく。

2 既存の技術とその問題点

「一時アドレス割当問題」と「計算機自動設定問題」を解決するための既存の方法としては RARP (Reverse Address Resolution Protocol)[6] と BOOTP (Bootstarp Protocol)[3] があげられる。RARP は Ethernet 上で動作し、データリンク層アドレス (Ethernet アドレス) からネットワーク層 アドレス (IP アドレス) への変換機構を提供するもので

† Implementation and Evaluation of Dynamic Host Configuration Protocol, by Akihiro Tominaga(Keio University), Fumio Teraoka(SONY CSL), Jun Murai(Keio University)

‡ tomy@sfc.wide.ad.jp
§ tera@csl.sony.co.jp
¶ jun@sfc.wide.ad.jp

あり、ディスクレスワークステーションの起動時などに利用される。BOOTP は UDP の上に構築されたプロトコルで、やはりディスクレスワークステーションの起動に利用されるのが一般的である。BOOTP は RARP と同様にデータリンク層アドレスからネットワーク層アドレスへの変換機構を提供すると同時に、サブネットマスク、ブートイメージの置かれているサーバマシンのアドレス、ルータのアドレスなどの情報を提供する機構をも含んでいる。

これらのプロトコルが考案されたときには、現在のような小型高性能の携帯型計算機は存在せず、IP アドレスの枯渇も予想できなかった。そのためこれら 2 つのアプローチでは前述した問題に対応しきれない点が存在するが、その主な欠点として次の 2 つがあげられる。

- 静的な割り当てしかできない。
- 計算機の必要とする情報の一部しか提供できない。

静的割り当てのみでは、計算機の移動に対応できない。動的割り当てを試みているものとして、NIP (Network Information Protocol)[7] と DRARP (Dynamic RARP)[2] があげられる。NIP は Ethernet アドレスに対して IP アドレスを割り当てるものであり、Ethernet のフレームを直接読み書きすることで動作している。NIP は問い合わせに対して、空いていそうなアドレスの範囲を返す。割り当てを受けるホストは、その中からあるアルゴリズムにしたがってアドレスを一つ選びだし、「本当にそのアドレスが使われていないか」を調べるために ARP (Address Resolution Protocol) 要求を出す。もし ARP 応答があれば、次の一つを試すようになっている。

これを "Polling/Defense mechanism" と呼ぶが、この方法では必ずしもアドレスの重複した割り当てを避けることができない。というのは、そのアドレスを使用中の計算機がたまたま負荷が高くて ARP 応答を出すのが遅れたのかもしれないし、その計算機が動作中でない可能性もある。また、どのホストがどのアドレスを使っているかという情報は明示的に保持されておらず、起動するたびに別のアドレスを割り当てられる可能性がある。起動するたびにアドレスが異なるのは混乱の元である。加えて NIP が取り扱うのは IP アドレスの割り当てだけであり、他のさまざまな情報は取得できない。

DRARP は名前の通り RARP を拡張したもので、やはり動的なアドレスの割り当てを可能にする。しかし割り当てられた IP アドレスの有効期限が 1 時間に固定されていることや、NIP と同様に他のさまざまな情報を取得できない点などに問題がある。加えて、DRARP はまだ完全に仕様が固まっていない。以上をまとめると、表 1 のようになる。

表 1: 各方式の比較

	静的割当	動的割当	情報量	完成度
RARP	○	×	×	○
BOOTP	○	×	△	○
NIP	×	△	×	△
DRARP	×	△	×	×
DHCP	○	◎	◎	○

これらの既存のアプローチの持つ問題点を解決するべく考案されたのが「動的ホスト設定プロトコル (Dynamic Host Configuration Protocol)」である。DHCP は BOOTP と同様に UDP の上で動作する。図 1 にプロトコル階層を示す。

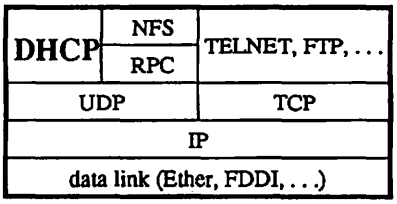

図 1: プロトコル階層

3 DHCP の概要

3.1 設計目標

DHCP はクライアントサーバモデルを用いており、一時アドレス割当問題、計算機自動設定問題、IP アドレス枯渇問題に対処するため、以下に示す目標の下に設計されている。

- 一時的にネットワークに接続される計算機にも動的にアドレスを割り当てられるようにする。その際アドレスには使用期限を設け、使われなくなったアドレスの回収を可能にする。
- ネットワーク管理者が個々の計算機を設定する必要をなくし、クライアントもユーザの介入なしに動作する。
- サーバ間のプロトコルを規定し、貴重な資源となっている IP アドレスを無駄無く割り当てられるようにする。
- 特定のホストに対する固定的な、あるいは恒久的な設定情報の割り当てを可能にする。

- DHCP はあくまでも割り当ての機構 (mechanism) を提供するだけであり、割当の方針 (policy) は管理者が自由に組み込めるようにする。

さらに使いやすさや既存のプロトコルとの互換性も考慮に入れ、以下のような目標も掲げられている。

- 従来のホストやネットワークプロトコルと共存できるようにする。特に従来の BOOTP クライアントもサポートする。
- 各サブネットにサーバがなくても動作するように、BOOTP 互換のリレーエージェントを経由したアドレス割り当てができるようにする。
- クライアントのリブートやサーバのリブートなどがあっても、可能な限り各ホストの設定が変わらないようにする。

しかしまだサーバ間のプロトコルは規定されておらず、IP アドレス枯渇問題に対しては課題を残している。

3.2 アドレス割り当てモデル

DHCP は BOOTP を拡張したものである。BOOTP はアドレスの静的な割り当てしかできない。それに対し DHCP は BOOTP のこの欠点を解消するように設計されており、次の 3 つのアドレス割り当てモデルを実現している。

- 動的割り当て (Dynamic allocation) ‥ サーバがアドレスを選択し、期限つきで割り当てる。
- 自動割り当て (Automatic allocation) ‥ サーバがアドレスを選択し、無期限で割り当てる。
- 手動割り当て (Manual allocation) ‥ あらかじめネットワーク管理者が割り当てておいたアドレスを通知する (静的な割り当て)。

最初の「動的割り当て」モデルは「一時的なアドレスをどうやって割り当てるか」という問題に対する解の 1 つであり、次の「自動割り当て」モデルは「計算機の設定をどのように自動化するか」という問題に対する解である。最後の「手動割り当て」モデルは、既存の BOOTP にほぼ対応する。割り当ての期限は 32 ビットの符号なし整数を使って、現在の時刻からの秒数として相対的に表されるため、サーバとクライアントの時刻が一致している必要はない。無期限を表すためには 0xffffffff が用いられる。

3.3 動作モデル

DHCP は、UDP 上にクライアントサーバモデルを用いて構築されている。DHCP には、サーバ、クライアント、リレーエージェントという 3 種類のモジュールが存在する。サーバの動作は単純で、各々のサーバは各種の設定情報を保持した「アドレスプール」と呼ばれるデータベースを持つ。基本的にこのデータベースの内容は変化しない。さらにサーバは、クライアントとアドレスプールのエントリの対応づけを管理する「バインディング (binding) データベース」を保持している。こちらは常に更新されるデータベースである。現在の DHCP ではアドレスプールの補充はできず、複数のサーバ間でアドレスプールの内容が重複することは想定されていない。クライアントはサーバに対するリクエストをブロードキャストし、それに対する応答を受けとることで動作するようになっている。

3.4 メッセージフォーマット

DHCP はプロトコルのすべてにわたって、BOOTP と上位互換性を保つように注意深く設計されており、従来の BOOTP のサーバやクライアントに悪影響を与えないだけでなく、DHCP サーバと BOOTP クライアントの組合せやその逆の組合せでも正常に動作するように工夫されている [5]。

op (1)	htype (1)	hlen (1)	hops (1)
xid (4)			
secs (2)		flags (2)	
ciaddr (4)			
yiaddr (4)			
siaddr (4)			
giaddr (4)			
chaddr (16)			
sname (64)			
file (128)			
options (312)			

図 2: DHCP のメッセージフォーマット

図 2 に DHCP のメッセージフォーマットを示す。括弧内はそのフィールドのオクテット長を表す。メッセージのフォーマットは BOOTP とほとんど同じであり、BOOTP と異なるのは次の 5 点だけである。

- "unused" であった 2 オクテットのフィールドが "flags" フィールドになった。
- "Vendor Extensions" フィールドの名称が "options" フィールドに変更された。

- その "options" フィールドの大きさが最小で312オクテットに拡張された(BOOTPでは64オクテット)。
- "options" フィールドを "file" フィールドと "sname" フィールドにまで拡張できる。
- メッセージの長さは可変長になり、"options" フィールドの拡張にともなって最小の長さは548オクテットになった。

"flags" フィールドを使用することについては、従来のBOOTPの仕様書[3]でも「メッセージを組み立てる前にパケットのバッファを全て0にクリアするのはよい考えである」と書かれてあるので、これを守っていない古いクライアントが多少問題となるだけである。しかも現在の仕様ではサーバが応答をブロードキャストするべきかどうかを示す1ビットが使用されているだけで、もし古いBOOTPクライアントがそのフラグを立てていたとしても、ユニキャストの代わりにブロードキャストが一回送信されるだけなので、さほど実害はない。なお、BOOTPの補足をしている文献[9]の中では、従来 "unused" だったフィールドをDHCPと同じように使用する旨が記述されている。

DHCPはオプションの種類を約60種類に増やし、ベンダ独自やサイト独自のオプションも扱えるようにしている(付録参照)。クライアントは自身の求める設定情報がどれであるかを示すのに "Parameter Request List" オプションを使うことができるため、サーバはすべての情報を返さなくてすむ。このようにDHCPはRARPやBOOTPが持っていた「計算機の必要とする情報の一部しか提供できない」という欠点に対処している。このためDHCPを通じて各種設定情報を得れば、それ以後ネットワーク上の他のホストと通信ができるようになっている。

3.5 プロトコルの動作

前述したように、DHCPはクライアントサーバモデルを用いて構築されている。クライアントの初期化は二段階に分かれており、第一段階では使用することのできるアドレスをサーバから通知してもらうためにブロードキャストを行なう。サーバからの応答は複数存在してもよい。次の段階でクライアントは、それらの複数のアドレスの中から一つ選びだし、サーバに対して割り当てを求める。サーバはそれに対して許可あるいは不許可を出すようになっている。

このように、元々のBOOTPでは「要求」と「応答」の2つでプロトコルが成り立っていたのに対し、DHCPでは表2に示すように7種類のメッセージを用いている。BOOTPと同様にクライアントがサーバへ送るメッセージの "op" フィールドにはBOOTREQUEST(= 1) という値が設定され、その逆方向の場合にはBOOTREPLY(= 2) という値が含まれる。それに加えて "options" フィールドに、DHCPで新たに追加された "DHCP message type" というオプションが含まれており、これを使って7種類のメッセージを区別することが可能となる。

表2: DHCPのメッセージ

メッセージ	内容
DHCPDISCOVER	サーバ群を特定するためにクライアントがブロードキャストする
DHCPOFFER	サーバがDHCPDISCOVERに応えて送る各種情報の提案
DHCPREQUEST	クライアントが一つのサーバにパラメータを要求するブロードキャスト(他のサーバに対しては提案の辞退を示す)
DHCPACK	サーバがクライアントにアドレスを割り当てる場合に送る
DHCPNAK	サーバがクライアントの要求を断る場合に送る
DHCPDECLINE	クライアントが割り当てられた情報やアドレスに何か問題を発見した時に送る
DHCPRELEASE	クライアントが期限が切れる前にアドレスを放棄する場合に送る

DHCPでは新たにアドレスを割り当てる場合(図3)と、以前に割り当てられたアドレスを再度割り当ててもらう場合(図4)で流れが異なる。まず最初のケースについて説明する。

1. DHCPクライアントは、そのサブネットに対してDHCPDISCOVERメッセージをブロードキャストする。メッセージには、希望するIPアドレスや希望する割り当て期限を示す値を含めても良い。このとき、リレーエージェントが別のサブネットのサーバへメッセージを転送するかもしれない。
2. 受信したサーバ(複数存在するかもしれない)はDHCPOFFERメッセージを返送してもよいし、しなくてもよい。返送する場合は割り当てても良いアドレスを "yiaddr" フィールドに含めてブロードキャストする。ここで注意すべきことは、この時点ではサーバとクライアントの間でアドレス割り当ての合意は形成されていないということである。DHCPOFFERは、あくまでも「提案」を行なうだけである。
3. クライアントは1つないし複数のDHCPOFFERを受けとり、何らかの方針に基づいてサーバを1つ選び出す。クライアントは選んだサーバのIPアドレスを "Server identifier" オプションに含めてDHCPREQUESTをブロードキャストする。

4. DHCPREQUEST を受けとったサーバは、"Server identifier" オプションをチェックし、自分宛てでない場合にはクライアントが提供を断ったものとみなす。自分宛てだった場合、そのアドレスを割り当てても良ければ DHCPACK を返す。何らかの理由で割り当てることができない場合 (例えば、別のクライアントにそのアドレスを割り当ててしまった場合) には DHCPNAK を返す。
5. DHCPACK を受けとったクライアントは、そのアドレスの使用を開始する。逆にもしクライアントが DHCPNAK を受けとった場合には初期化を最初からやり直す。
6. クライアントが DHCPACK で受けとったアドレスや情報に問題を発見した場合は、DHCPDECLINE をサーバに送信して初期化を最初からやり直す。
7. クライアントが停止する場合や、別のサブネットへ移ろうとするときには DHCPRELEASE を送信して、割り当てを放棄する旨を通知しても構わない。ただしこれを行なわなくても DHCP は正常に動作するようになっている。

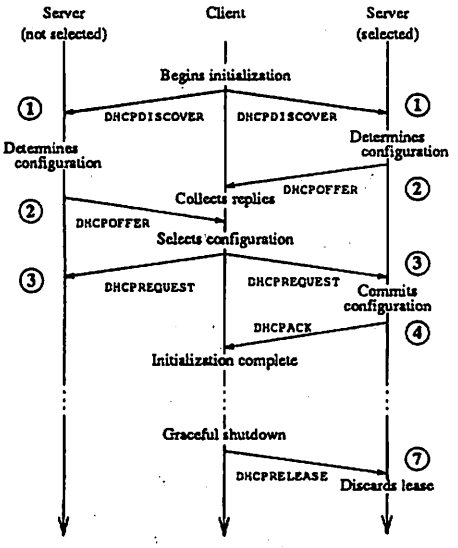

図 3: アドレスの新たな割り当て

これに対し、クライアントが以前に割り当てられたアドレスを再度割り当ててもらう場合や期限の延長をする場合には、いくつかのステップが省略できる。

1. クライアントは以前のアドレスを DHCPREQUEST メッセージの "ciaddr" フィールドに入れてブロードキャストする。期限の延長の場合には明示的に希望する期限を示す "IP Address Lease Time" オプションを含める。
2. そのアドレスについての情報を持つサーバが DHCPACK か DHCPNAK で応える。
3. 後は図 3の場合と同様。

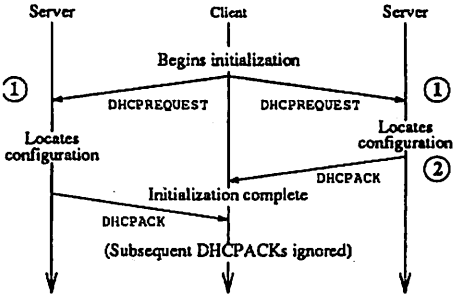

図 4: アドレスの再度割り当て

3.6 リレーエージェント

前述したように DHCP の動作時にはリレーエージェントが介在することがある。リレーエージェントは、サーバとクライアントが別々のサブネットに存在する場合にメッセージの転送を行なうものであり、クライアントと同じサブネットに存在する。したがって、各サブネットにはサーバかリレーエージェントのどちらかが存在しなければならない。リレーエージェントは次のように動作する。

1. クライアントが DHCP メッセージのブロードキャストをする。
2. クライアントのブロードキャストを受けとったリレーエージェントは、それを受けとったネットワークインターフェースの IP アドレスを "giaddr" フィールドに設定し、サーバに向けて転送する。
3. サーバは "giaddr" フィールドが 0 でないことから、それがリレーエージェントによって中継されたメッセージであることが分かる。サーバは応答を "giaddr" フィールドに含まれる IP アドレスに向けて送信する。その際に "giaddr" フィールドは受信したメッセージと同じ値に設定する。

4. サーバからの応答を受けとったリレーエージェントは、そのメッセージの "giaddr" フィールドに含まれるアドレスが示すネットワークインタフェースへ受信したメッセージをブロードキャストする。
5. クライアントはリレーエージェントのブロードキャストを受信する。

ただし、サーバは何らかの方法によってリレーエージェントの存在するサブネットのサブネット番号を知る必要がある。

4 実装の基本設計

DHCP を実装するにあたって目標としたことを以下に示す。括弧内はサーバ、クライアント、リレーエージェントのうちのいずれに対する目標なのかを示す。

1. 一般的なオペレーティングシステム上で動くようにする (すべて)。
2. オペレーティングシステムに対する独立性を高くする (すべて)。
3. 実際の大規模な環境で使えるようにする (サーバ、リレーエージェント)。
4. ユーザの使いやすさを考慮する (すべて)。
5. ソースコードを全て公表し、自由に変更できるようにする (すべて)。

以降、サーバ、クライアント、リレーエージェントのそれぞれについて上に示した全体の目標を具体化し、さらに各々に固有の目標を示す。また、それらを実現するための方針や問題点についても述べる。

4.1 サーバの設計

今回は「一般的なオペレーティングシステム」として BSD 系の UNIX を選んだ。また DHCP 自体はデータリンク層を特に仮定していないが、今回はデータリンク層として Ethernet を仮定する。次の「オペレーティングシステムに対する独立性を高くする」を達成するためには、オペレーティングシステムのユーザ空間におけるアプリケーションプログラムとして実装すればよい。オペレーティングシステムのユーザ空間で実装をする際に生ずる最大の問題点は、通常の *socket* インタフェースでは、宛先 IP アドレスが 255.255.255.255 のブロードキャスト (リミテッドブロードキャスト) ができないことである。BOOTP サーバのユーザ空間における実装として CMU で作られたものが知られているが、これはリミテッドブロードキャストを行なわず、あらかじめ ARP テーブルを設定することによってクライアントに割り当てるアドレスに対するユニキャストを行なっている。ところが自分のハードウェアアドレス宛ての Ether フレームに含まれる IP データグラムであっても、自分自身に正当な IP アドレスが設定されるまではそれを受けとることのできないクライアントが存在する。したがって、CMU の実装方法ではこのタイプのクライアントはうまく動作しない。いわゆる「鶏が先か卵が先か」問題である。

またルータ上で BOOTP サーバが動作している場合、割り当てる IP アドレスはあらかじめ登録されているので、その IP アドレスが示すサブネットに応答を返せばよい。仮にクライアントが別のサブネットに接続されていても、応答が本来クライアントの存在すべきサブネットに向けて無駄に送信されるだけである。ところが DHCP サーバの場合には、クライアントがどのサブネットに存在するかを認識することによって割り当てるアドレスを決定する。もしリレーエージェントが介在する場合には比較的容易にサブネットを認識できる。しかしサーバが直接ブロードキャストを受信した場合、どのネットワークインタフェースからかということをユーザ空間で認識することは通常不可能である。これらの問題にいかに対処したかについては次の第 5 節でくわしく解説する。

「実際の大規模な環境での使用」に耐えるためには、アドレスプール量が増えてもメモリ使用量は $O(AddressPool)$ 以下にするする必要がある。また、複数のクライアントがほぼ同時に要求を出してもそれに応えられるようなパフォーマンスを達成する必要がある。さらに混雑したネットワークであっても動作するように、効率的に DHCP メッセージを選択する必要もある。「長時間の運用に耐えるようにする」というのは、DHCP の設計目標に掲げられている「サーバのリブートなどがあっても継続して動作する」という目標に関わってくるので重要である。「ユーザの使いやすさを考慮する」ために、複雑なサーバ設定ファイルを記述しなくてもすむようにすること、アドレスプールデータベースやバインディングデータベースを容易に記述できるようにすることを目標とする。DHCP サーバは必ずしも BOOTP クライアントをサポートする必要はないが、今回の実装では BOOTP のサポートも目標の一つとする。

4.2 クライアントの設計

クライアントもサーバと同様に BSD 系 UNIX のユーザ空間で実装する。その際にはやはり、どのようにしてリミテッドブロードキャストを行なうかと

いう問題が生じる。

先にも述べたように、DHCP は現在約 60 種類の設定情報を取り扱うことができ、その内容も非常に多岐にわたっている (付録参照)。"Parameter Request List" オプションが存在することからも明らかなように、クライアントがこれらのすべての情報を必要とするとはまず考えられない。したがって、クライアントがすべての情報を取り扱うように一括してプログラミングすることは現実的でないし、プログラムやオブジェクトコードのサイズ、メモリ使用量から見ても無駄が多い。そこで今回は、ライブラリルーチンの形でクライアントを提供することにした。つまり各ユーザは自分の要求にしたがってライブラリルーチンを使用したプログラミングをするものとする。その代わりにライブラリルーチンの使用例としてサンプルプログラムを添付することにする。その際に「ユーザの使いやすさの考慮」として、ライブラリルーチンにはアドレスの取得、アドレスの再確認、アドレスの期限延長、アドレスの解放といった DHCP の各アクションに対応した比較的抽象度の高いルーチンを用意し、DHCP の詳細についてはほとんど知る必要がないようにする。

4.3 リレーエージェントの設計

これもサーバやクライアントと同様である。サーバ同様にリレーエージェントも複数のネットワークインタフェースを持つ場合には、それらのうちのどれから要求を受信したかを知る必要がある。リレーエージェントも非常に簡潔な設定ファイルで動作するようにする。

5 実装

本節では実際の実装を、サーバ、クライアント、リレーエージェントの順に解説していく。その際に、前節で掲げた目標をどのような方法で解決したかを示していく。

5.1 サーバの実装

今回の実装は C 言語にして約 7400 行になった。もちろん目標にも掲げたように完全にアプリケーション層で実装を行なった。前節で繰り返し述べたように、いかにしてリミテッドブロードキャストをするかという問題に関しては Berkeley Packet Filter(BPF) を使用することで対処した。BPF は BSD 系の UNIX にはほとんど実装されていて、ネットワークインタフェースのドライバに組み込まれている。このためデータリンク層 (既に述べたように今回は Ethernet に限定) のパケットを直接送受信できるようになっている。したがってどのネットワークインタフェースからパケットを受けとったか識別することも容易であるし、リミテッドブロードキャストを行なうこともできる。その代わりに、プログラムがトランスポート層、ネットワーク層、データリンク層のすべてを自身で処理する必要がある。ただし、通常の UDP を使用できる部分についてはオペレーティングシステムの *socket* インタフェースを通して処理している。図 5 に、クライアント、リレーエージェントも含めたサーバーの実装構造を示す。

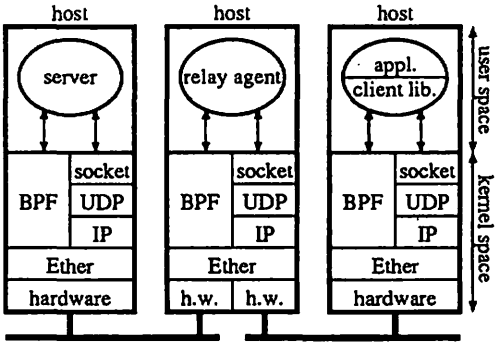

図 5: 実装構造

BPF は名前の通り強力なフィルター機能を持っていて、フィルターの記述は疑似アセンブラのような独自のコマンド体系を使用することで行なう。もちろんサーバプログラム自身でフィルタリングを行なうことも可能だが、そうするとカーネルモードからユーザモードへの切替え、ユーザ空間へのメモリコピーなどのオーバーヘッドが増えてしまう。そのため、混雑しているネットワークでも効率的に動作するように、今回の実装では BPF の中でフィルタリングをおこなっている。具体的には、各々の Ethernet フレームに対して次のようなチェックを行なっている。

1. IP データグラムであるか?
2. UDP データグラムであるか?
3. UDP のポート番号が DHCP のものであるか?

さらにサーバプログラムで、次のようなチェックも行なう。

1. IP データグラムの長さが正常か?

2. IP の宛先がブロードキャストあるいは自分の IP アドレスか?
3. UDP データグラムの長さが正常か?
4. IP チェックサムは正しいか?
5. UDP チェックサムは正しいか?
6. DHCP クライアントからの要求か("op" フィールドの中身が BOOTREQUEST か)?
7. "Options" フィールドの先頭に正しい Magic Cookie が入っているか?

もちろん、すべてのチェックを BPF 内で行なうことも不可能ではないが、疑似アセンブラによる実現には手間がかかることや、サーバプログラムでフィルタリングを行なう方が柔軟性が高いことなどを考慮した。

BPF を使ったプログラムというと *tcpdump* などが有名であるが、多くの方々は BPF は遅いという印象を持たれていると思う。しかし実際は *tcpdump* などのプログラムが、自分のハードウェアアドレス宛てでないパケットも受けとれるように、ネットワークインタフェースを PROMISC モード*にしてしまうために遅くなっているだけである。DHCP が受けとる必要のあるのは、自身のハードウェアアドレスか、ハードウェアブロードキャストアドレス宛てのパケットだけであるため、ネットワークインタフェースを PROMISC モードにする必要はない。それに加えて若干のフィルタリングを行なっているために、計算機に与える負荷はさらに低い。

サーバは、*inetd* のように要求を受けとると自分自身の複製を作って処理するのではなく、自分ですべて処理を行なうようになっている。データベースの更新という機構があるために、なんらかのロック機構を導入しない限り複製を作って処理を行なうことは不可能であるし、仮にそれを実現するとしても、それに要する努力の割に効果は薄いと考えたためである。したがってカーネルやネットワークインタフェースも含めたバッファリングの能力を越える勢いでパケットを受信すると処理できなくなってしまう。

データベースの記述を容易にするために、アドレスプールデータベースの記述は、いわゆる *termcap* と同じ形式にした。ただし若干の文法上の変更を加えて機能を拡張してある。これは CMU の BOOTP サーバに含まれるソースを変更することで実現した。これに対しバインディングデータベースは単純に 1 行 1 エントリの、データの羅列である。また、リレーエージェントとサブネット番号の対応を記述したデータベースも必要であるが、これは不当なリレーエージェントからの転送を拒否するためのアクセスコントロールリストとしても使用する。

*イーサネット上のすべてのパケットを受信するモード

```
global.dummy:snmk=255.255.254.0:tmof=32400:
subnet68.dummy:tblc=global.dummy:\
               :rout=133.27.68.1:\
               :dht1=500:dht2=850:\
               :brda=133.27.69.255:\
:dfll=600:maxl=600:
  684:     :ipad=133.27.68.4:tblc=subnet68.dummy:
  685:     :ipad=133.27.68.5:tblc=subnet68.dummy:
  686:     :ipad=133.27.68.6:tblc=subnet68.dummy:
```

図 6: アドレスプールデータベースの記述例

表 3: ライブラリルーチン一覧

dhcp_init	DHCP クライアントの初期化
dhcp_discover	DISCOVER を送り OFFER を集める
dhcp_request	REQUEST を送り ACK を待つ
dhcp_verify	アドレスの再確認
dhcp_extlease	期限の延長
dhcp_decline	DECLINE を送る
dhcp_release	アドレスの解放
clean_param	構造体のクリア
arp_reply	ARP 応答の送信
config_if	ネットワークインタフェースの設定
reset_if	ネットワークインタフェースのリセット

メモリの使用量を抑えるために動的にメモリを確保するのはもちろん、ポインタを利用してメモリの節約に努めた。その結果、各ホストエントリに対して最初にとられる領域はわずか 208 バイトのサイズしかない。また長時間の運用を行なっても問題は発生していない。図 6 にアドレスプールデータベースの記述例を示す。

5.2 クライアントの実装

ライブラリは C 言語にして約 3000 行である。やはり BPF を利用したユーザ空間のプログラムであり、サーバ同様のフィルタリング方法を使っている。

用意した関数の一覧を表 3 に示した。このようにプログラミングインタフェースは単純であり、得たい情報の種類であるとか、要求する期限といった情報を含む構造体を渡して呼び出すと、各種設定情報が設定された構造体が返されるようになっている。基本的には、最初に *dhcp_init*、*reset_if* を呼び出しておいて *dhcp_discover* を呼び出す。そうすると各サーバからの応答が構造体になって返されるので、そのうちの一つを選んで *dhcp_request* を呼び出せば、最終的な設定情報の入った構造体が返されるというものである。

クライアントの実装目標も達成できたが、実際に使用するにはルーティングテーブル設定などの処

```
int main(int argc, char **argv)
{
  initialize();
  dhcp_init();

  /* main loop */
  while (1) {
    /* get OFFER from server */
    construct_request(req, &ipaddr);
    dhcp_discover(req, &netif, &param);

    /* get ACK from server */
    choose_offer(&param);
    dhcp_request(req, &netif, param);

    /* got ACK and configure */
    config_if(&netif, &param->yiaddr,
      param->subnet_mask,
            param->brdcast_addr);
    set_route(param);

    while (1) { /* extend lease loop */
      construct_extlease(req, param);
      if (dhcp_extlease(req, &netif, param) < 0) {
        reset_if(&netif);
        break;
      }
    }
  }
}
```

図 7: クライアントの実装例

理も行なわなければならない。このような処理は機種依存性が高いために、今回のライブラリには含めなかった。その結果、プログラマにはルーティングテーブルに関する知識など要求される。したがって「カーネルなどに関する知識は不要」という目標は、完全には達成できていない。図 7 にクライアントライブラリの使用例を示す。ここではエラー処理などは一切省略してある。

5.3 リレーエージェントの実装

やはり BPF を使用していて、C 言語で約 880 行である。信頼性のために、複数のサーバに転送を行なえるようになっている。設定ファイルには DHCP サーバの IP アドレスのリストと、(信頼性のために) そのうちのいくつのサーバに転送をするかを示す数字を記述するだけである。このように設定ファイルの記述が容易であるし、安定して動作することも確認している。複数のサーバをサポートする際には、クライアントが送信した DHCP メッセージを元にハッシュを行なって転送先を分散させることにより、サーバの負荷分散を狙っている。これにより大規模なネットワークでもサーバがうまく動作すると考えるが、現在のところは確認できていない。

6 実装評価

サーバ実装の全体目標として掲げた項目は一つを除いてすべて達成できた。残りの一つ「大規模環境でも動作する」は実験を行なっていないために確認できていない。現在のところ、我々の実装は BSD/386 と NEWS-OS という 2 種類の BSD 系 UNIX オペレーティングシステム上での動作を確認している。これらは Intel80486/80386、MC68030 および R3000 という全く異なったアーキテクチャに基づく CPU 上で動作している。このことが、今回の実装は機種独立性を高くすることに成功したことを立証している。当然、他の BSD 系 UNIX への移植も容易であり、SunOS に移植することも現在計画中である。SunOS には、BPF とほぼ同等の機能を持つ NIT(Network Interface Tap) と呼ばれる機能が存在するため、移植は容易であると思われる。また MS-DOS 上の BOOTP クライアントとの相互運用性のテストでも、良好な結果が得られている。Termcap 形式のデータベースを導入したことにより、設定の柔軟性も確保できた。

クライアントライブラリに限定すれば、すべての目標を達成できている。しかし実際のクライアント実装においては、プログラマにはシステムに関する知識が要求される。本質的に、DHCP のクライアントは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など、システムに深くかかわる項目を操作するので、これはやむを得ない。クライアントライブラリも、BSD/386 と NEWS-OS 上での動作を確認している。

リレーエージェントについても目標は達成できた。しかし設定の柔軟性に欠ける部分があるので、今後の課題としたい。

7 おわりに

機種独立性を高めた DHCP の実装という今回の目標は達成できたが、依然として課題も残されている。サーバやクライアントには仕様を完全に満たしていない部分が存在し、さらに設定ファイルの記述のしやすさについても改善する必要がある。また、さまざまな DHCP の実装の相互運用テストが 1993 年 10 月 29 日に米国のシアトルで計画されている。我々もこのテストに参加し、我々の実装の問題点を洗い出してくる予定である。それらを実装に反映させて改善していきたい。

今後は慶應義塾大学湘南藤沢キャンパスにおいて大規模な実験を行ない、今回は未評価のままになってしまった大規模環境での耐久性などについて評価する予定である。また DHCP の応用を考えていき

たい。移動計算機のためのプロトコル[8]への応用などは非常に興味深いといえる。なお今回実装したプログラムについては、無償で配布を行なう計画であることを最後に付け加えておく。

謝辞

論文の内容等について相談にのっていただいた慶應義塾大学の徳田英幸助教授に感謝します。また常に相談に応じてくれた共同研究者の方たち、特に実装の際に直面した問題の解決法を助言してくださった電気通信大学計算機科学科前期博士課程の植原啓介氏とソニー(株)スーパーマイクロ事業本部の尾上淳氏に感謝します。最後にさまざまな議論・意見を提供して下さった慶應義塾大学徳田・村井研究室の諸氏とWIDEプロジェクトの研究者の方々、そしてIETFのDHCワーキンググループのメンバーにも感謝します。

参考文献

[1] Alexander, S. and Droms, R.: *DHCP Options and BOOTP Vendor Extensions*, RFC1533, October 1993.

[2] Brownell, D.: *Dynamic RARP Extensions for Automatic Network Address Acquisition*, Sun Microsystems, Inc., September 1989.

[3] Croft, B. and Gilmore, J.: *Bootstrap Protocol (BOOTP)*, RFC951, September 1985.

[4] Droms, R.: *Dynamic Host Configuration Protocol*, RFC1531, October 1993.

[5] Droms, R.: *Interoperation Between DHCP and BOOTP*, RFC1534, October 1993.

[6] Finlayson, R., Mann, T., Mogul, J., and Theimer, M.: *A Reverse Address Resolution Protocol*, RFC903, June 1984.

[7] Schiller, J. and Rosenstein, M.: *A Protocol for the Dynamic Assignment of IP Addresses for use on an Ethernet*, MIT Athena Project, 1989.

[8] Uehara, K., Teraoka, F., Sunahara, H., and Murai, J.: Enhancement of VIP and Its Evaluation, In *Proceedings of INET'93*, August 1993.

[9] Wimer, W.: *Clarifications and Extensions for the Bootstrap Protocol*, RFC1532, October 1993.

付録: DHCPのオプション一覧

(名称を一部省略しているところがあるため、正しくは文献[1]を参照のこと。)

Pad	Subnet Mask
Time Offset	Router
Time Server	Name Server
Domain Name Server	Log Server
Cookie Server	LPR Server
Impress Server	RLS server
Host Name	Boot File Size
Merit Dump File	Domain Name
Swap Server	Root Path
Extensions Path	IP Forwarding
NonLocal Source Routing	Policy Filter
Max Dgram Reassm Size	Default IP Time-to-live
PathMTU Aging Timeout	Path MTU Plateau Table
Interface MTU	All Subnets are Local
Broadcast Address	Perform Mask Discovery
Mask Supplier	Perform Router Discovery
Router Solicitation Addr	Static Route
Trailer Encapsulation	ARP Cache Timeout
Ethernet Encapsulation	TCP Default TTL
TCP Keepalive Interval	TCP Keepalive Garbage
NIS Domain	NIS servers
NTP Servers	Vendor Spec Info
NetBIOS Name Server	NetBIOS Dgram Dist Srv
NetBIOS Node Type	NetBIOS Scope
X Font Server	X Display Manager
Requested IP Address	IP Address Lease Time
Option Overload	DHCP Message Type
Server Identifier	Parameter Request List
Message	Max DHCP Message Size
Renewal (T1) Time	Rebinding (T2) Time
Class-identifier	Client-identifier
End	