動的ホスト設定プロトコルの実現と移動計算機への応用に関する研究

指導教官
徳田 英幸 ・村井 純

慶應義塾大学 環境情報学部
79053779
徳田英幸研究室
冨永 明宏

平成 8 年 4 月 25 日

概要

インターネット上では、ネットワークへの計算機の接続/切断という動的な変化が生じる。この場合、ネットワークアドレス等の共有資源の割り当てが不可欠であるが、これらの資源を自動的かつ集中的に管理する一般的な機構は存在しなかった。さらに移動計算機の登場にともない、共有資源の割り当てを動的に行なう必要も生じてきている。これらの機能を実現するための標準プロトコルとして Dynamic Host Configuration Protocol (DHCP)[1][2] の仕様がインターネットに提案された。インターネットにおいて DHCP のもつ共有資源の管理機能は重要であるにも関わらず、現在にいたるまで DHCP の実装は配布、公開されていない。

本研究は大きく分けて 3 つの部分から構成されている。第一に、DHCP の仕様書に規定されている全てのモジュール (サーバ、クライアント、リレーエージェント) を、仕様に完全準拠した形で実装した。その際、オペレーティングシステムを変更することなしに、アーキテクチャから独立した実装を行なうことを目標とした。そのために、Berkeley Packet Filter や Network Interface Tap などのネットワークインタフェース層を直接読み書きできる機構を利用した。これにより、オペレーティングシステムのユーザ空間で動作するアプリケーションプログラムとして実装を行なうことが可能となった。その結果として機種独立性が非常に高く、汎用性の高い実装を初めて完成することができた。 UNIX 上に構築した本実装は、テキストによる設定ファイルの記述という既存のシステム管理の手法を踏襲し、各種のデータベースを容易に記述、変更できるようにした。サーバおよびリレーエージェントに関しては、大規模な環境での使用にも耐えるように設計した。またクライアントは個々のユーザの要求に柔軟に対応できるように、ライブラリ形式で実装を行なった。

第二に、DHCP を移動計算機へ適用した。DHCP の持つ資源管理機構の一つにネットワークアドレスの動的な割り当て機能がある。これにより頻繁に接続/切断を繰り返す携帯型計算機のサポートが可能になった。本研究ではこの機構をさらに発展させて、移動計算機サポートプロトコルの一つとして提案されている Virtual Internet Protocol (VIP)[3][4] への DHCP の応用を行なった。 DHCP と VIP は非常に親和性が高く、計算機の透過的かつ円滑な移動を可能にすることができた。今後はこれら DHCP の実装を VIP と共に配布して、広域分散環境における実験を行なう予定である。

第三に、DHCP の “サーバ間プロトコル” に関する考察を行なった。現在の DHCP では複数サーバによる信頼性の向上や、階層的なアドレス管理を想定したサーバ群の構造化が行なわれていない。しかし、サーバ同士が協調して動作することにより、IP アドレスの枯渇などの問題にも対処できうる。そこで本論文では、サーバ同士が協調するための “サーバ間プロトコル” に求められる機能などについて検討し、その基本的な設計上の問題点についても考察した。

Abstract

On the Internet, many dynamic changes, such as the connection and disconnection of hosts, occur. When a host is connected to the Internet, the allocation of shared resources, such as network addresses, is indespensable. However there has been no concentrated mechanisms for managing these resources automatically. The appearance of the mobile host introduces a new requirement for dynamic allocation mechanisms for shared resources. To achive these functions, specifications of the protocol named Dynamic Host configuration Protocol (DHCP)[1][2] have been proposed to the Internet. Although the mechanisms provided by DHCP, such as the management of shared resources, are important for the Internet, implementations of the DHCP have never been released nor distributed as public domain software.

This research consists of three parts. First, all modules (server, client and relay agent) were implemented with full specifications as described in the specifications of the DHCP. The goal was to implement these modules without modifying the operating system. Thus, we could achieve better architecture independant implementation. Therefore mechanisms such as Berkeley Packet Filter and Network Interface Tap were used to directly read and write to the network interface. By using these mechanisms, it became possible to implement the modules as application programs running in the user space of the operating system. As a result, it was possible to achieve an implementation with high flexibility and with high indpendence from machine type. This system, which was implemented on UNIX, has adopted the traditional method of managing the system by writing simple text files, in order to write and maintain the database entries easily. The server and the relay agent were both designed to be used in a large-scale environment. The client was implemented as a user-level library to cope with various requirements of individual users.

Secondly, we adapted the DHCP to the mobile host. One of the resource management mechanisms of the DHCP is the dynamic allocation of the network address. This function provided the ability of supporting portable computers which repeat connections and disconnections frequently. In this work, the allocation function was extended, and was applied to the Virtual Internet Protocol (VIP)[3][4] which is one of the protocols that has been proposed to support the mobile host. The DHCP works with the VIP very well, and the combination of the two enables the mobile host to move around with both transpareny and smoothness. There are plans to distribute these programs along with implementations of the VIP, and to then experiment them in a widely distributed environment.

Finally, the “server-to-server protocol” for the DHCP was discussed. In the current DHCP, there is neither a mechanism supporting the cooperation between multiple servers for higher reliability, nor a mechanism of hierarchical address management with structured multiple servers. However, if servers could cooperate with each other, it would be possible to cope with problems such as the lack of IP addresses. We discussed the functions required for the server-to-server protocol for cooperation between multiple servers and their basic design issues.

目 次

図目次

表目次

第 1 章

序論

1.1 はじめに

現在のインターネットは計算機が相互に接続されることで成立しているが、インターネットに接続される計算機には識別のための番号として、全世界を通じて一意の値を持つ IP アドレスが割り当てられる。インターネットに接続されている計算機の集合は絶えず変化し、新たに接続される計算機には IP アドレスを割り当てる必要がある。

インターネットに接続された計算機がインターネット上の他の計算機と通信を行なうためには、IP アドレス以外にも様々な共有資源の割り当てを受け、設定情報を取得して計算機の設定に反映させる必要がある。逆にインターネットから計算機が撤去される場合には、その計算機の占有していた IP アドレスや共有資源を回収する必要がある。しかしながら現在にいたるまで、これらの作業の大半が人手に委ねられてきた。

これらの割り当て、回収および設定作業は極めて繁雑である上に誤りを犯しやすい。誤って設定された計算機は他の計算機と通信できないばかりでなく、状況によってはネットワーク上の他の計算機に悪影響を及ぼす。加えてインターネットの拡大にともなって、共有資源の管理に要する労力が増加している。結果として、資源の管理はおろそかになり、共有資源の利用効率が低下する。この端的な例としてあげられるのが、IP アドレスの枯渇問題 [5] である。 IP アドレスのとれる値の総数には限りがあるが、近い将来に割り当て可能な IP アドレスが尽きてしまうという予測がある。これを回避するためには、IP アドレスの利用効率を向上させることが望ましい。

さらに近年の携帯型計算機および小型ネットワーク機器の登場により、計算機はインターネットへの接続/切断を頻繁に繰り返すようになった。これらの移動する計算機は比較的短い期間のみ共有資源の割り当てを要求する。このような新しい種類の要求には、既存の人手による方法では対応できない。

したがって現在のように個々の計算機を手動で管理するのではなく、自動的かつ集中的に管理する機構が求められる。これらの目的の一部を解決するために考案されたプロトコルがいくつか存在する。その中でも代表的なのが、RARP[6] と BOOTP[7] である。この 2 つのプロトコルは共に、2 次記憶を持たない計算機の起動時などに利用される。OSI 7 層モデル [8] におけるデータリンク層データリンクアドレス (MAC アド

レス) からネットワーク層アドレス (IP アドレス) への変換は、RARP と BOOTP に共通する機能である。BOOTP はさらに数種類の設定情報を取得し設定する機能を持つ。

前述の移動計算機に特有の要求に応えるためには、共有資源の動的な割り当てのための機構が必要となる。しかしながら RARP や BOOTP には静的な割り当ての機構しか存在しない。それゆえ、これらの要求に応えるための機構として、Dynamic Host Configuration Protocol (DHCP)[1][2] が提案されている。 DHCP の持つ特徴として、以下のようなものがあげられる。

- 動的な IP アドレスの割り当てが可能
- 使用されていない IP アドレスの回収が可能
- 管理者やユーザの介入なしに動作する

このような特徴をもつ DHCP はインターネットにおいて重要な役割を担うと思われるが、現在までのところライセンスフリーで利用可能な実装は配布されていない。本研究の第一の目標は、DHCP の実装を完成し、評価することにある。その際、実装が一般的に利用されるためには、機種独立性の高い実装を行なう必要がある。そこで本研究では、DHCP を機種独立性の高い形で実現する際に生ずる問題点の考察と解決方法の提案を行ない、それに基づいた実装と評価を行なう。

前述したように携帯型計算機等の登場により移動計算機の概念が生まれた。移動計算機のネットワーク接続は、DHCP を利用するだけで実現できる。DHCP は移動計算機の非常に初歩的なサポートを可能にするが、設定が自動化された点と持ち運び可能な点以外は、既存の据え置き型計算機と変わらない。

移動計算機をサポートするためのプロトコルとして Virtual Internet Protocol (VIP)[3][4] が提案されている。VIP を利用することで計算機の移動透過性が実現されるが、VIP ホストは他のネットワークに移動する時には一時的な IP アドレスの取得を必要とする。本研究では、この一時 IP アドレスの取得に DHCP を適用する方法に関しても考察し、実装と評価を行なう。

現在の DHCP の仕様にはいくつかの問題点が存在する。また現在の DHCP の持つ機能的な限界に言及し、これらの問題点を解決するためには DHCP を拡張したサーバ間プロトコルが必要である。本論文ではサーバ間プロトコルに関する考察も行なう。

1.2 本論文の構成

本論文では、第 2 章で DHCP の概要について解説する。第 3 章ではまず DHCP 実装の設計目標を定義し、実装上の問題点を明らかにする。その後、設計と実装について解説する。第 4 章では実装の評価を行なう。さらに、初期の DHCP のクライアントの持つ欠点を明らかにし、クライアントの改良と実装、評価を行なう。第 5 章では VIP の概要を紹介した後に、DHCP を VIP へ適用する方法を解説する。また初期版 (Version 1) のクライアントによる実装と新版 (Version 2) のクライアントによる実装を行ない、

両者の比較評価を行なう。第6章では現在のDHCPの問題点とDHCPサーバ間プロトコルに関して考察する。最後に第7章で本論文を締めくくる。

第 2 章

Dynamic Host Configuration Protocol

この章では、Dynamic Host Configuration Protocol (DHCP) の持つ機能に着目し、概要を解説する。さらに DHCP に関連する各種のプロトコルについても触れる。

2.1 DHCP の概要

2.1.1 DHCP の設計理念

インターネットに計算機が接続される際に生ずる計算機資源の割り当てを“時間的広がり”と“設定作業に必要な技術的熟練度”の二点から分類することができる (図 2.1)。図 2.1において「長期接続」とは「計算機の新規購入・廃棄」を意味する。同様に「中期接続」は「計算機の起動・停止」を、「短期接続」は「携帯型計算機の持ち運び」を、「ごく短期の接続」は「無線 LAN での移動」を意味する。なかでも「短期・ごく短期」は近年になって生まれた概念である。

過去のインターネットにおいては、「長期~中期」の接続に「熟練」したユーザが対応するのが一般的であった。しかし近年ではユーザが「熟練」していても頻繁に生じる「短期~ごく短期」の接続要求に対応しきれなかったり、「長期~中期」の接続であってもユーザが「初心者」であるために、対応できなかったりする事例が増えている。加えてインターネットの規模の拡大がこの傾向に拍車をかけている。

分析すると、次のような要求が存在することがわかる。

- 計算機の自動設定 .. 計算機の接続と誤りがちな設定の自動化
- 計算機の集中管理 .. 共有資源の集中的な管理
- アドレスの一時的な割り当て .. 短期的な接続のための一時的な IP アドレスの割当

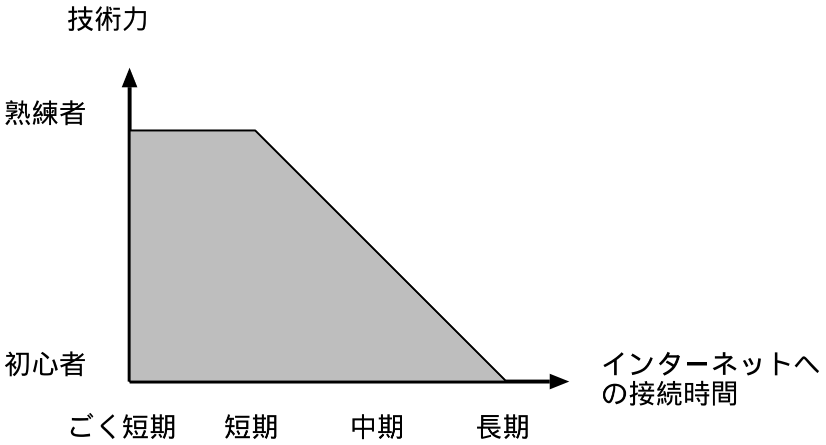


図 2.1: DHCP の有効領域

DHCP はこれらの要求を解決するものであり、とりわけ DHCP が有効なのが図 2.1 において塗りつぶした領域である。さらに、近年の懸念である IP アドレスの枯渇に対処するために、次のような要求も存在する。

- 効率的利用 .. 使用されていない IP アドレスの回収
- 共有機構 .. 複数の計算機での IP アドレスの共有

DHCP はこれら 2 つの要求に関しても取り扱っている。ただし残念ながら、完全な解決にまでは至っていない。文献 [1][2] には DHCP 全体の設計目標として、以下の項目が掲げられている。

- DHCP は方針 (policy) を提供するのではなく、機構 (mechanism) を提供する。ローカルのシステム管理者が望むように、設定パラメータを制御できなければならない。例えばローカル資源へのアクセスと割り当てに関する方針が自由に設定可能でなくてはならない。
- ホストはユーザによる介入なしで、設定情報を発見し設定を行なうことが可能でなければならない。
- ネットワーク管理者がホスト毎の設定情報を入力しなくても、ホストの設定が自動的に行なわれるべきである。
- 大規模性と経済性のために、DHCP はルータを越え、あるいは BOOTP[7] や DHCP のリレーエージェントを仲介して動作しなければならない。
- 信頼性とパフォーマンスを向上させるために複数のサーバを設置しても、クライアントはそれらのサーバからの複数の応答に対処できなければならない。

- 静的に設定された DHCP と無関係のホストや既存のネットワークプロトコルの実装と共存できなければならない。
- DHCP は RFC[1]951[7] および RFC1542[9] に述べられている BOOTP リレーエージェントと相互運用できなければならない。
- DHCP は既存の BOOTP クライアントに対してもサービスを提供しなければならない。

さらにネットワーク層のパラメータに固有の設計目標として、以下の項目があげられている。

- 複数のホストが同じネットワークアドレスを同時に使わないように保証する。
- ホストを再起動しても、設定を保つ。また、可能な限り以前と同じ設定情報 (例えばネットワークアドレス) を割り当てる。
- サーバを再起動しても、各ホストの設定を保つ。さらに DHCP の機構を再起動しても、可能な限り以前と同じ設定情報を割り当てる。
- 新しいホストに対する設定情報の自動割り当てを可能にする。
- 特定のホストに対する設定情報の固定的、恒久的な割り当てを可能にする。

2.1.2 DHCP の概念モデル

DHCP はサーバクライアントモデルの上に構築されている。設定情報を得るために DHCP を利用するホストがクライアントであり、その要求に応えるのが DHCP サーバである。これに加えて、DHCP にはリレーエージェントというものが存在する (図 2.2 および図 2.3)。

DHCP は BOOTP を拡張したプロトコルである。BOOTP は MAC アドレスから IP アドレスへのマッピングと、数種類の設定情報の通知を行なうものである。(この「設定情報」のことを BOOTP では “Vendor Extension”、DHCP では “Option” と呼ぶ)。BOOTP、DHCP ともに UDP[2]上に構築されていて、All 1 (宛先 IP アドレスが 255.255.255.255) のブロードキャストを用いる。しかし All 1 のブロードキャストがルータを越えて別のサブネットに到達することはできない。したがって DHCP の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、中継を行なうのがリレーエージェントである。リレーエージェントの持つ中継機構は通常のルータが持つ転送機構とは異なり、受けとった DHCP メッセージに変更を加えてから再度 UDP データグラムとして送り直すものである。したがってリレーエージェントはルータ上で動作していても良いし、通常のホスト上で動作していても良い。 DHCP のリレーエージェント

---

[1] インターネットにおける様々な技術的議論を行なうために Internet Engineering Task Force という組織が存在する。そこで発行される公式文書を Request For Comments (RFC) と呼び、通し番号を振って整理する。

[2] コネクションレスのデータグラム転送サービスを提供するトランスポート層のプロトコル。

と BOOTP のリレーエージェントは等価であり、リレーエージェントに関する詳細は文献 [9] に見ることができる。

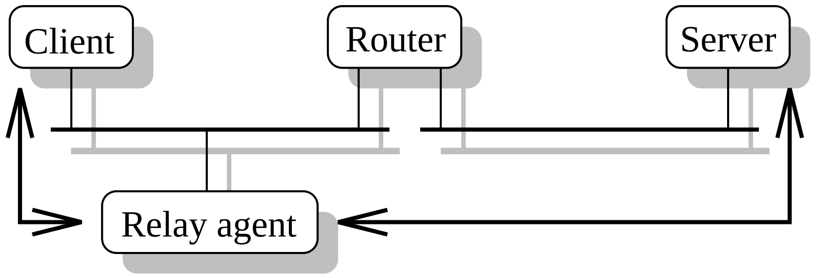


図 2.2: ルータとリレーエージェントが別の場合

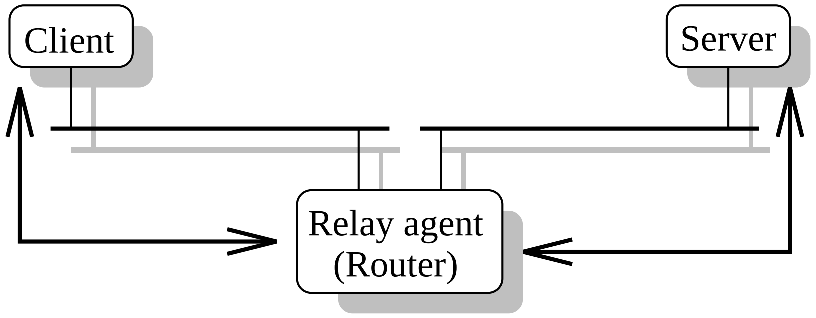


図 2.3: ルータとリレーエージェントが同一の場合

サーバ

DHCP サーバは、2 つのデータベースを管理する。1 つは「アドレスプール (Address Pool)」と呼ばれ、ネットワーク上の割り当てるべき共有資源を含んだ静的なデータベースである。このデータベースの内容は基本的に変化しない。もう 1 つは「バインディングデータベース」と呼ばれ、常に更新されるデータベースである。各種設定情報 (特に IP アドレス) の割り当ては、サーバがアドレスプールの中からエントリを選びだし、それをクライアントに通知する形で行なわれる。この際に、「どのクライアントにどのエントリを割り当てたか」という対応づけを行なう。この「対応」を DHCP ではバインディング (binding) と呼ぶ。

現在の DHCP では、アドレスプールの補充を自動的に行なうことはできないし、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。サーバはクライアントから DHCP メッセージが届かない限りは何の動作もおこさないメッセージ駆動型のモジュールである。

クライアント

クライアントは、設定情報の割り当て要求をUDPデータグラムに入れて送信し、サーバからの応答を受けとることで動作する。サーバはメッセージ駆動型なので、すべての動作をクライアントが能動的に行なわなければならない。DHCPの仕様書[1][2]の中では、有限状態機械によるクライアントのモデル化が行なわれている。

リレーエージェント

リレーエージェントは、クライアントの送信したDHCPメッセージをOSI 7層モデルにおけるアプリケーション層で受けとる。リレーエージェントは、DHCPメッセージのいくつかのフィールドを書き換え、再度UDPデータグラムにのせ、サーバに向けてユニキャスト[3]で送信する。メッセージを書き換えることからも明らかなように、ルータの持つ転送機能とは全く異なる機構を持つ。

2.1.3 割り当てモデル

DHCPでは設定情報の割り当てを「動的割り当て」、「自動割り当て」、「手動割り当て」の3種類に分類している。

動的割り当て

DHCPの最も特徴的な割り当て機能である。サーバはアドレスプールの中からIPアドレスを選択し、クライアントに期限つきで割り当てる。この際にIPアドレスに加えて各種の設定情報を通知する。有効期限は現在からの相対時刻で表されるため、サーバとクライアントの時刻が一致していなくても良い。有効期限は32ビットの符合なし整数を使って1秒単位で表され、最長約100年までの期限を表すことができる。なお最短の有効期限は、RFC1541[1]で1時間と規定されている。この動的割り当てにより、「アドレスの一時的な割り当て」という要求を満たすことが可能となる。

自動割り当て

動的割り当て同様に、サーバがアドレスプールの中からIPアドレスの選択を行なう。ただし割り当ての有効期限は無期限な点が異なる。有効期限は32ビット符合なし整数で表されるが、特に無期限は$2^{32}-1$という値を持つ。(したがって無期限をのぞく最大有効期限は$2^{32}-2$秒である)。この機構は、「アドレスを自動的に割り当てたい」という要求を満たすものであり、主に据え置き型の計算機に対する割り当てを想定している。現在のDHCPでは、(ホストの廃棄等で使用されなくなった)無期限のアドレスを回収する機構は定義されていない。

---

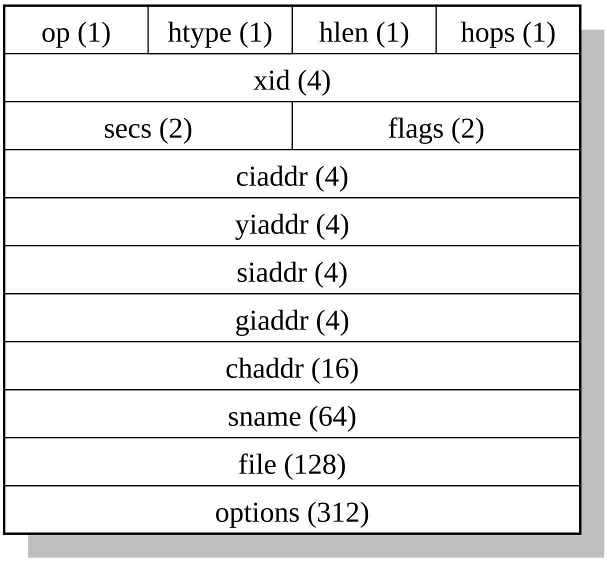
[3] ネットワーク上の全てのホストに向けた送信がブロードキャストであるのに対し、特定のホストに向けた送信をユニキャストと呼ぶ。

手動割り当て

前述の2つとは異なり、この割り当て機構ではあらかじめ管理者が割り当てておいたアドレスを通知するためだけにDHCPが利用される。いわゆる　静的割り当て」と呼ぶべきものであり、これは既存のBOOTPの機構とほぼ等価である。

2.1.4　メッセージのフォーマットおよびタイプ

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[10]。



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2.4: DHCPのメッセージフォーマット

DHCPのメッセージフォーマットは図2.4のようになっている。括弧内の数字は各フィールドのオクテット長を示す。互換性を保つため、メッセージフォーマットはBOOTPとほぼ同一であり、BOOTPと異なるのは以下の5点だけである。

1. “unused”であった2オクテットのフィールドが“flags”フィールドに変更
2. “Vendor Extensions”フィールドの名称を“options”フィールドに変更
3. “options”フィールドの大きさが最小で312オクテットに拡張(BOOTPでは64オクテット)
4. “file”フィールドと“sname”フィールドを“options”フィールドと同じ目的に使用可能

5. 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も 548 オクテットに増加

“flags” フィールドを使用することについては、従来の BOOTP に関する RFC でも「メッセージを組み立てる前にパケット用のバッファを全て 0 にクリアするのが望ましい」と記述されている。仮にこの推奨に準じていないクライアントが存在するとしても、深刻な問題とはならない。現在の仕様ではサーバがクライアントへ送る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 (図 2.5参照) ある。したがって仮に古い 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 の仕様を補足している文献 [9] の中では、従来 “unused” であったフィールドを DHCP と同じように使用する旨が記述されている。

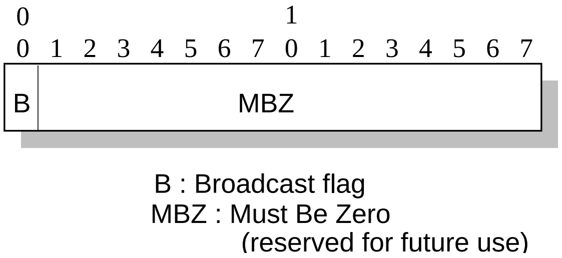


図 2.5: フラグフィールド

2.1.5 プロトコルの動作概要

BOOTP が要求と応答」のようにメッセージを一度だけ交換するプロトコルだったのに対して、DHCP は表 2.1 に示すような 7 種類のメッセージを用いている。ただし、BOOTP との互換性を保つため、“op” フィールドの使い方は BOOTP と全く同じである。すなわちサーバからクライアントへ送られるメッセージには BOOTREQUEST (= 1) が含まれ、逆方向には BOOTREPLY (= 2) が含まれている。したがって 7 つのメッセージを区別するために “options” フィールドに “DHCP message type” というオプションが新たに追加された。

これら 7 種類のメッセージを使用して設定情報の割り当てが行なわれる。本節の残りでは、新たにアドレスを割り当てる場合と、アドレスの再割り当て/確認をする場合のプロトコルの流れについてみていく。

新規アドレス割り当て

新たにアドレスを割り当てる場合のプロトコルの流れを図 2.6に示す[4]。

---

[4] 図中の数字と説明文の数字は対応している。

表 2.1: 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	クライアントが選択した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際に送る
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

1. DHCP クライアントは、自身の接続しているサブネットに DHCPDISCOVER メッセージをブロードキャストする。クライアントは、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 (実装依存) 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、既に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[5]は、DHCPDECLINE をサーバに送信して初期化を最初からやり直す。

---

[5] 例えば RFC1541 では、そのアドレスが本当に使用されていないことを確認するために ARP 要求を使ったチェックを行なうことを推奨している。Address Resolution Protocol (ARP)[11] とはネットワーク層のアドレスをデー

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。

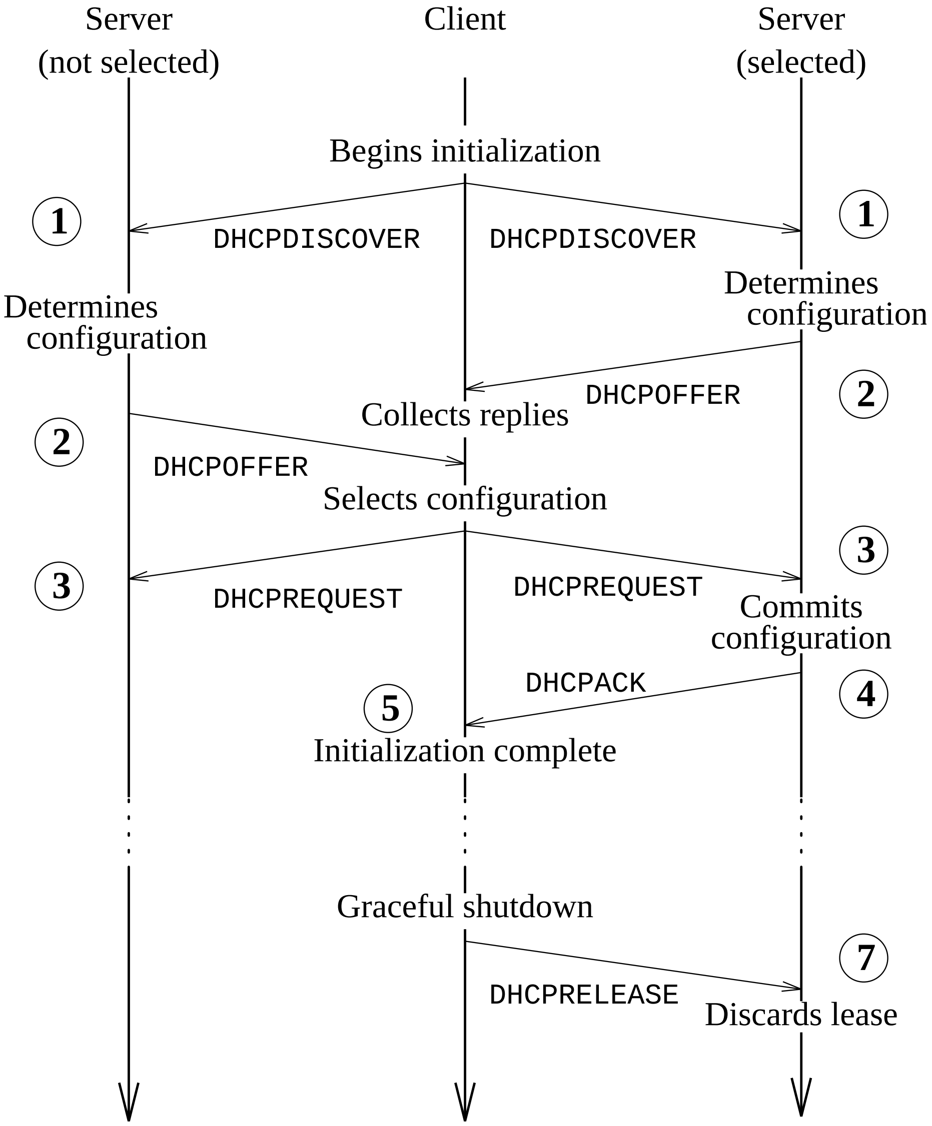


図 2.6: 新規アドレス割り当て

タリンク層のアドレスに変換するものである。要求に対する応答がなければ、そのアドレスは使用されていないと推測できる。

アドレス再割り当て/期限延長/確認

クライアントは、動的割り当てを受けたアドレスの期限が切れる前に、延長を申請することができる。期限が切れた後であっても、以前に使用していたアドレスの再割り当てを申請することが可能になっている。さらにDHCPは割り当てられた設定情報が変化した可能性のある場合には、任意の時点で「確認」を行なうことができる。確認と再割り当ての間に本質的な違いはない。唯一の違いは、確認の場合はアドレスの有効期限が切れていないということである。

これらの場合には、既に説明したプロトコルフローのうちのいくつかのステップを省略することができる。

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”オプションを含めなければならない。
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. 以後は図2.6の場合と同様。

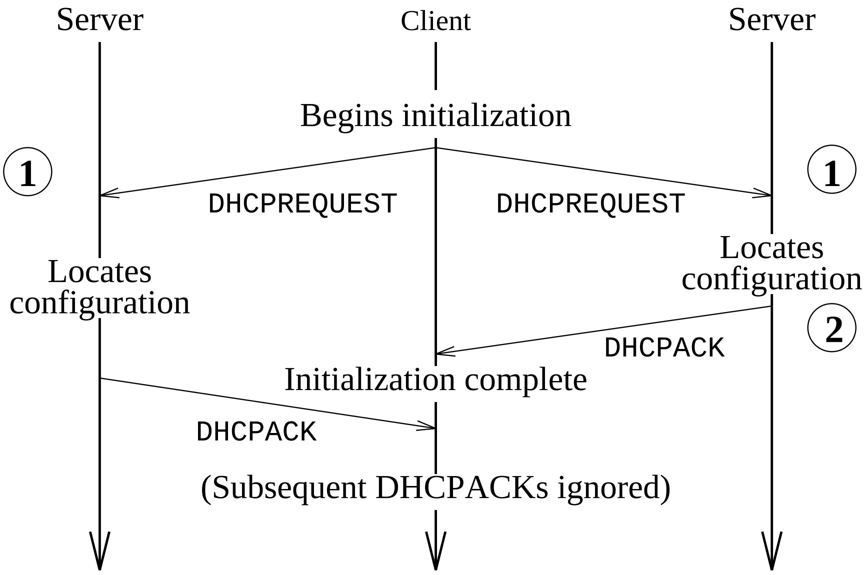


図2.7: アドレスの再割り当て/期限延長/確認

図2.8[6]はクライアントの状態遷移図である。アドレスの新規割り当てや再割り当てなども、このように有限状態機械としてモデル化されている。

---

[6] DHCPの仕様に関してはIETFのDynamic Host Configuration Working Group (DHC WG)で議論されていて、WG内部にのみ公開されている非公式な仕様書として文献[2]がある。多くの部分で仕様に細かい変更が加えられていて、この状態遷移図も例外ではない。図2.8は文献[2]に含まれるものである。

クライアントが初めてDHCPを利用する場合にはINIT状態から、再起動などDHCPを利用するのが初めてでない場合にはINIT_REBOOT状態から開始する。INIT状態からBOUND状態までの遷移は「新規アドレス割り当て」において解説した。またINIT_REBOOT状態までの遷移は「アドレス再割り当て/期限延長/確認」において解説したので説明を省略する。

BOUND状態に遷移したクライアントは、二つの時刻T1およびT2を管理する。これらの値はサーバから通知されるか、さもなければ標準値[7]が利用される。クライアントは時刻T1がくるとRENEWING状態に遷移し、期限の延長申請を開始する。最初にその時点で利用しているアドレスを割り当てたサーバに対してユニキャストで延長申請を開始する。

時刻T2まで再転送を繰り返してもDHCPACKもDHCPNAKも返ってこない場合にはREBINDING状態に遷移し、ブロードキャストで延長申請をおこなう。有効期限が切れるまでにDHCPACKを受けとることができた場合には、BOUND状態に遷移する。それ以外の場合にはINIT状態に遷移することになっている。

リレーエージェントの介在

リレーエージェントは以下のようにして中継を行なう。

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。

---

[7] RFC 1541ではT1は有効期限の0.5倍、T2は有効期限の0.875倍と定義されている。

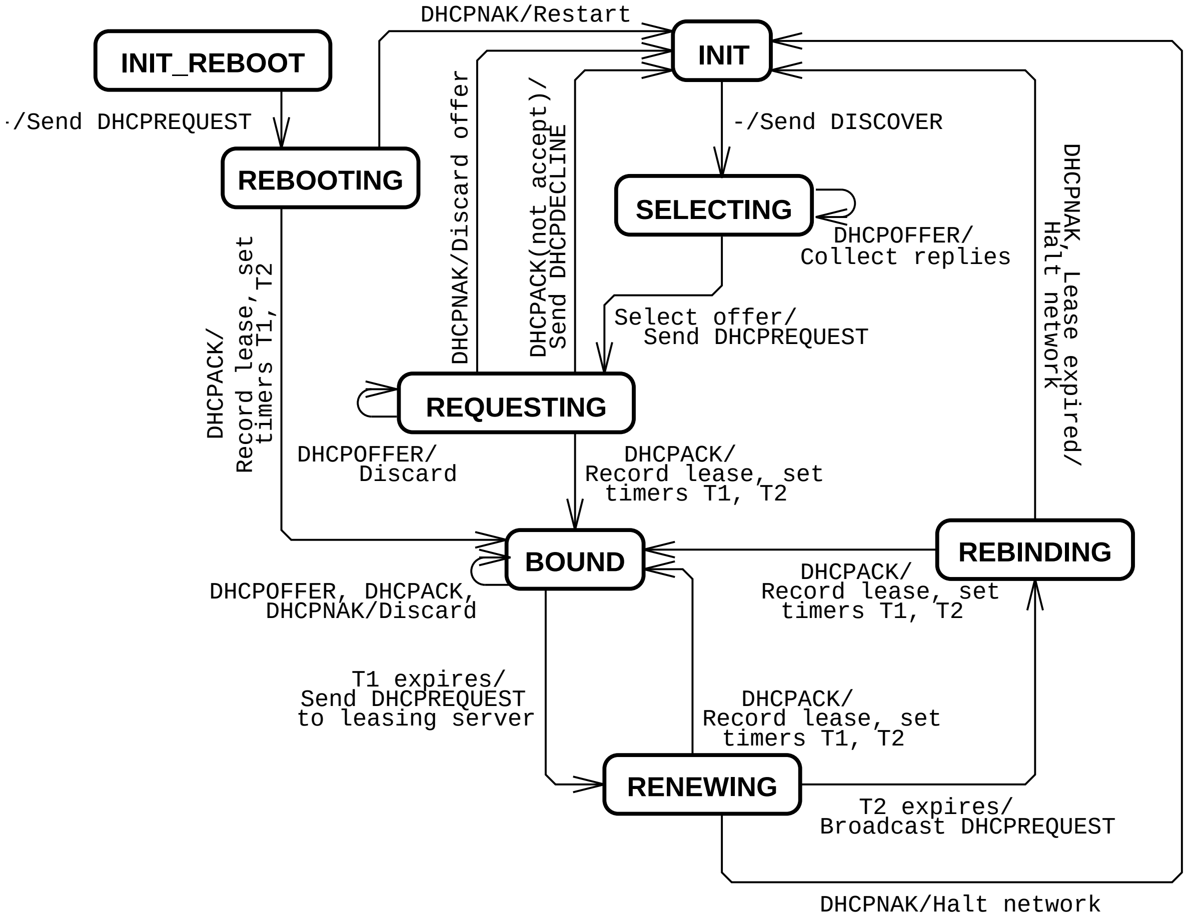


図 2.8: DHCP クライアントの有限状態機械モデル

その他

DHCP は DHCPDECLINE および DHCPRELEASE に対する確認応答 (Acknowledgement) を持たない。したがって DHCPACK によって割り当てを受けたアドレスに関してクライアントが問題を発見した場合に送信する DHCPDECLINE が、必ずしもサーバに到達しないことがある。同様にアドレス割り当ての期限を解放する DHCPRELEASE もサーバに到達したことを確認できない。

2.2 関連研究

すでに触れたように DHCP は BOOTP の拡張である。しかし他にも DHCP の取り扱う問題の一部を解決しようと試みたプロトコルがある。本節ではそれらの概要を紹介し、簡潔に DHCP と比較する。

Reverse Address Resolution Protocol (RARP)

Reverse Address Resolution Protocol (RARP)[6] は、MAC アドレスから IP アドレスへのマッピングを行なうもので、ディスクレスワークステーションや X 端末などの起動時に用いられる。それ以外の機能は持たない。RARP は Ethernet の上で Ethernet のフレームを直接読み書きすることで動作する。RARP は計算機を自動的に設定したいという要求のごく一部を実現するものといえる。

Bootstrap Protocol (BOOTP)

Bootstrap Protocol (BOOTP) は、MAC アドレスと IP アドレスのマッピングおよび数種のホスト設定情報の通知機能を持つことについては既に触れた。BOOTP も通常はディスクレスワークステーションなどの起動時に用いられる。しかしながら DHCP と異なり、BOOTP はアドレスの動的な割り当てができず、DHCP における「手動割り当て」の機構のみを提供する。したがって BOOTP は計算機の自動設定という要求に関しては、ある程度解決しているものの、一時的なアドレス割り当てや使われなくなったアドレスの回収といった要求に関しては対応できていない。

Network Information Protocol (NIP)

Network Information Protocol (NIP)[12] は、MIT の Athena プロジェクトで独自に開発されたもので、アドレスの動的な割り当てを行なうことができる。NIP は Ethernet のフレームを直接読み書きすることで動作し、MAC アドレスに対して IP アドレスを割り当てる機構を持つ。

NIP はアドレス割り当ての要求に対して、「未使用と思われる」アドレスの集合を返す。割り当てを受けるホストは、その集合の中からあるアルゴリズムに基づいてアドレスを一つ選択する。さらに「本当にそのアドレスが未使用である」ことを確認するために、ARP 要求を送信する。もし応答があった場合には、アドレスを選択し直す。

これを “Polling/Defense mechanism” と呼ぶが、この機構ではアドレスが絶対に重複しないと保証することが出来ない。なぜなら問題のアドレスを使用している計算機の負荷が高かったために ARP 応答を行なうのが遅れたのかも知れないし、あるいは計算機が動作中でないのかも知れない。さらにどの計算機がどのアドレスを使用しているかという情報が保持されていないため、起動するたびに以前と異なるアドレスが割り当てられる可能性がある。これは好ましくない。

NIP は計算機の自動設定と動的なアドレス割り当てという要求を一部解決している。しかしながら取得できる設定情報は RARP 同様に IP アドレスに限定されているし、動的な割り当てにも問題がある。

Dynamic RARP (DRARP)

DRARP[13] 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 等と同様に取得できる情報の種類に限りがあるなどの問題がある。さらに DRARP は開発の途上にある。

Point-to-Point Protocol (PPP)

PPP[14] 自体はポイント-ポイント間のリンク上で複数のプロトコルデータグラムを転送するためのものである。PPP は 3 つの要素から構成されているが、その中に Link Control Protocol というものが含まれている。これを IP に適用したものとして Internet Protocol control Protocol (IPCP)[15] が存在する。

IPCP では、PPP のコネクションを設定する際にコネクションの両端の IP アドレスを設定する方法が規定されている。文献 [14][15] および、それに基づいた具体的な実装の詳細を参照すると、以下のような手順でアドレスの設定が行なわれている。(以下の説明では、ホスト X、ホスト Y の二つが存在すると仮定する)。

1. ホスト X が Source Address を知らない場合
    (a) ホスト X は、ホスト Y にアドレスの割り当てを要求する
    (b) ホスト Y は、Destination Address(= X 側のアドレス) が設定されている場合にはそれをホスト X に返す
    (c) ホスト X は、それを自身のアドレスとして使用する
2. ホスト X が Source Address を知っている場合
    (a) ホスト X は、自身の Source Address をホスト Y に通知する
    (b) ホスト Y は、Destination Address(= X 側のアドレス) が設定されていないならば、通知されたアドレスを使用する

(c) 逆に Destination Address(= X 側のアドレス) が設定されていて、さらにそれがホスト X から通知されたアドレスと異なる場合には、その Destination Address を X に返す

(d) ホスト X は、返ってきたアドレスと自分のアドレスが違うために、接続を断念する

このように PPP は、単純なアドレスの設定機能を持っている。しかしながら、実際にアドレスをどのように割り当てるかということに関しては規定されていない。したがって PPP のアドレス設定機能は、むしろ前述の BOOTP に近い。

DHCP との比較

RARP, BOOTP, IPCP (PPP) は静的な割り当てしかできない。一方 NIP, DRARP は動的な割り当てしかできず、また動的な割り当ての機構も不完全である。提供できる情報の種類は

$$RARP, NIP, DRARP, PPP << BOOTP < DHCP$$

の順に豊富である。 DHCP では現在約 60 種類の情報を提供することができる (表 2.3 参照)。ただし常にすべての情報を必要とするわけではないため、クライアントが望む設定情報のリストを “Parameter Request List” オプションとしてサーバに通知することができる。

以上の比較評価をまとめると表 2.2のようになる。

表 2.2: 各方式の比較

	静的割当	動的割当	情報の種類	スケール	プロトコル完成度
RARP	○	×	×	×	○
BOOTP	○	×	△	△	○
NIP	×	△	×	×	△
DRARP	×	△	×	×	×
IPCP (PPP)	○	×	×	×	○
DHCP	◎	◎	◎	◎	?

表 2.3: DHCP のオプション一覧

(名称を一部省略しているため、正しくは文献 [16] 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

第 3 章

DHCP の実装

この章ではまず、DHCP を実装するにあたって設定した目標を示す。次にサーバ、クライアント、リレーエージェントのそれぞれについて、それらの目標を達成する上で生じる問題を明らかにし、その解決方法を考案する。その後、各々についてさらに具体的な設計と実装の解説を行なう。

3.1 実装目標

今回の実装で全体的な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頭文字で表している。

1. 一般的に使用されているオペレーティングシステム上で動くようにする (S,C,R)
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 (S,C,R)
3. 実際の大規模な環境で使えるようにする (S,R)
4. ユーザの使いやすさを考慮する (S,C,R)
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 (S,C,R)

現在までのところ DHCP の実装は公開、配布されていないため、ライセンスフリーの実装を行なうことは重要である。DHCP を組み込んだ計算機が販売されるようになったとしても、機種依存性が高くなると予想される上に、プログラムのソースコードは公開されない可能性が高い。したがって、一般的なオペレーティングシステムの上で動作する機種独立性の高い実装を行なうことを最大の目標とした。

3.2 サーバの設計と実装

サーバを実装する際の問題点とその解決策を示し、設計と実装の詳細について解説する。

3.2.1 設計

今回は “一般的なオペレーティングシステム” としてUNIX(BSD系およびSunOS4.1.3)を選択した。DHCP自体はデータリンク層が何であるかを特に仮定していないが、本実装はEthernetに限定する。“オペレーティングシステムに対する独立性” を高くするために、DHCPサーバを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を行なう。

しかしユーザ空間で実装する際に生ずる最大の問題点として、通常の*socket*インタフェースではAll 1のブロードキャストが不可能だということがあげられる[1]。アメリカのCarnegie Mellon大学(CMU)の公開しているBOOTPサーバもユーザ空間で実装されているが、このサーバはAll 1のブロードキャストを行なわない。その代りBOOTPメッセージをクライアントに送信する直前に、クライアントのEthernetアドレスとクライアントの(あらかじめ管理者によって割り当て済みの)IPアドレスを組としたARPテーブルを設定する。この方法により通常のユニキャストでBOOTPメッセージを返すことを可能にしている。

しかし自身のEthernetアドレスを宛先に持つEthernetフレームに含まれるIPデータグラムであっても、IPアドレスが設定されるまでは受信することのできないタイプのクライアントも存在する。したがってこの実装方法のサーバと組み合わせてもうまく動作しないクライアントが存在する。このため “IPアドレスを設定するためには、IPアドレスが設定されていなければならない” という問題が発生する。

またBOOTPサーバが、ルータ上で動作していると仮定する。この場合は割り当てるIPアドレスが既知なので、そのアドレスに対応するサブネットに応答を送信すればよい。しかしDHCPの場合には割り当てるアドレスを選択しなければならない。仮にサーバが複数のネットワークに接続されているとすると、サーバはどのサブネットから要求を受信したかによって、割り当てるアドレスを決定しなければならない。だがユーザ空間のプログラムが、ブロードキャストを受信したインタフェースを識別することは通常不可能である。これらの問題点への対処方法に関しては、第3.2.2節でくわしく解説する。

“実際の大規模な環境” で使用できるために、アドレスプールのエントリ数が増えてもメモリの使用量をO(エントリ数)以下にする必要がある。さらに多数のクライアントがほぼ同時に要求を送出したとしても、それらを処理できるようなパフォーマンスを達成しなければならない。そのためにはEthernet上を流れる雑多なデータグラムの中から、効率的にDHCPのメッセージを抽出することが重要である。また “長時間の運用” を行なっても安定して動作しなければならない。これはDHCPの実装目標として掲げられている “サーバのリブートの前後を通じて、ホストの設定を保つ” という要求にも深く関わってくる。

“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済むようにする(既存のデーモンの設定ファイルなどと同様にする)ことと、アドレスプールデータベースなどの記述を簡潔にすることの2点を具体的な目標として設

---

[1] 厳密に言えばNet2以降のBSDなどでは不可能ではないが、問題が多い。

定した。

DHCP は BOOTP の上位互換であるため、DHCP サーバが BOOTP クライアントをサポートすることが可能なことは既に説明した。BOOTP クライアントのサポートは必ずしも行なわなくても良いのだが、今回の実装ではこれも目標に含めた。

3.2.2 実装

本実装は C 言語にして約 6400 行からなる。目標にも掲げたようにオペレーティングシステムのユーザ空間で実装されている。前節で指摘した “どのようにしてユーザ空間のプログラムから All 1 のブロードキャストを行なうか” という問題に関しては、Berkeley Packet Filter (BPF)[17] または Network Interface Tap (NIT)[18] を使用することで解決した。BPF は BSD 系の UNIX に含まれるもので、ネットワークインタフェースのドライバに直接組み込まれている。そのためデータリンク層 (今回は Ethernet に限定) のパケットを直接送受信できる。したがってどのネットワークインタフェースから パケットを受信したか識別することも容易であるし、All 1 のブロードキャストを行なうことも可能である。その代わりにトランスポート層、ネットワーク層、データリンク層の処理を自力で処理する必要がある。状況によってはDHCP サーバは通常のユニキャストによる送受信を行なうが、その場合にはオペレーティングシステムの *socket* インタフェースを使用している。なお、SunOS の場合には BPF の代わりに Network Interface Tap (NIT) を利用した[2]。図 3.1 に、クライアントとリレーエージェントも含めたサーバの実装構造を示す。

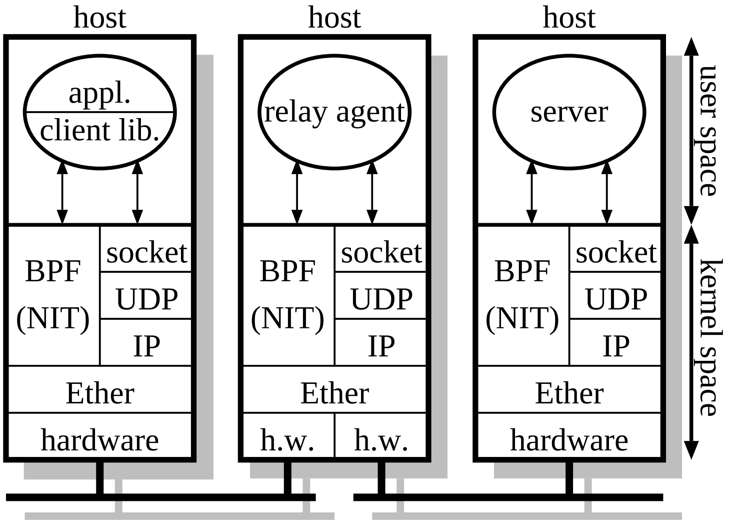


図 3.1: 実装構造

BPF は強力なフィルター機能を持ち、疑似アセンブラのような独自のコマンド体系

---

[2] SunOS でも BPF を使用できるが、NIT は OS に標準で組み込まれている。

を用いて任意のフィルターを作成できる。もちろんBPFのレベルでなくサーバプログラムの中でフィルタリングを行なうこともできるが、速度などの観点から言って非効率的である。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、BPFのレベルでフィルタリングを行なった。具体的には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[3]であるか

一方、サーバプログラムの中でも次のようなチェックを行なっている。

1. IPデータグラムの長さが正常か
2. IPデータグラム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”フィールドの値がBOOTREQUESTか)
7. “Options”フィールドの先頭に正しいMagic Cookie[4]が入っているか

もちろんすべてのチェックをBPFの中で行なうことも可能である。しかし疑似アセンブラによる実現に要する手間とサーバプログラム内でフィルタリングを行なう場合の柔軟性の高さを考慮した。なおBPFは使用しているが、PROMISCモード[5]は使用しない。またBPFでのフィルタリングを行なっているため、システムに与える負荷を低く押さえることができている。

SunOSでNITを使った場合も、ほぼ同様である。しかしNITはフィルターを記述するコマンド体系に間接アドレッシングが存在しない。したがってIPヘッダのIPオプションをスキップすることが不可能である。したがってSunOS版ではBPFのレベルで行なっていたフィルタリング処理のうち、“UDPポートのチェック”をサーバプログラム内で行なっている。BPF版はDHCPパケットが届くまで処理を中断しているのに対し、NIT版はUDPパケットが届くたびにサーバプログラムに処理が戻る。したがってNIT版は計算機パワーを余分に消費する。

サーバは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、自分ですべての処理を行なう。仮に複製を作って処理する場合、データベースを更新する際に

---

[3] BOOTPと同じ67, 68番を使用している。

[4] プロトコルとは無関係のメッセージを処理すると、そのメッセージフォーマットなどに対する仮定が崩れ、予想外の状況を引き起こす可能性がある。これを回避するために、ある特定のオクテット列を定められた位置に挿入することがあり、これをMagic Cookieと呼ぶ。DHCPの場合には99, 130, 83, 99という4桁のオクテットを使用する。無関係のメッセージがDHCPのMagic Cookieと同じ位置に偶然同じ値を持つ確立は$1/256^4$と非常に低い。

[5] Ethernet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 3.2: アドレスプールデータベースの記述例

複製同士の間で矛盾なく処理を行なえるように何らかのロックク機構を導入する必要がある。これに要する手間とそれに対する効果を考慮して、複製を使用する方法は採用しなかった。したがってオペレーティングシステムなどの性能も含めた処理能力を上回る頻度でパケットが到着すると、処理しきれなくなる。

データベースの記述を容易にするため、アドレスプールデータベースの記述はいわゆる *termcap*[19] と同形式を採用した。ただし文法に若干の変更を加えて機能を拡張してある。この部分に関しては、CMU の BOOTP サーバに含まれるソースに変更を加えることによって作成した。一方バインディングデータベースは単純にデータの羅列を 1 行に 1 エントリ記述するようになっている。リレーエージェントから DHCP メッセージが中継されてきた場合、アドレスを割り当てるためにはそこのサブネット番号を知る必要もある。今回は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を用意した。この中に記述されていないリレーエージェントからの転送は処理しないように実装したため、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の役割も持っている。

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、ポインタを可能な限り共有することでメモリの節約を目指した。その結果、各ホストエントリに対して最初に確保される領域は 208 オクテットである。

図 3.2にアドレスプールデータベースの記述例を示す。 基本的に 1 行が 1 エントリであるが、“\”(バックスラッシュ) を使用して複数行にまたがった記述をすることも可能である。各行の先頭がエントリの見出し (インデックス) であり、“ipad=” は IP アドレス、“snmk=” はサブネットマスクを表している。それぞれの値は “:” で区切られている。*termcap* と同様に他のエントリへの参照を使って、記述を簡略化することもできる (例: “tblc=” の部分)。

3.3 クライアントの設計と実装

3.3.1 設計

クライアントの場合にも、サーバ同様に UNIX 上に実装する。やはりブロードキャストの利用に関して問題が生じるが、BPF や NIT を使用して解決できる。

第2章で述べたように、DHCP では現在約 60 種類の設定情報を取り扱うことが可能で、その内容も非常に多岐にわたっている (表 2.3)。しかし通常のクライアントがこれらの情報をすべて必要とするとは考えられない。また DHCP の仕様自体が “Parameter Request List” オプションによって設定情報の種類を抑えられるようになっている。したがって、すべての情報を取り扱えるようなクライアントを実装することは非現実的であるし、プログラムやオブジェクトコードのサイズ、使用するメモリの量などの観点からいっても無駄が多い。そこで本実装では、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する。各ユーザは個人的な要求に基づいて、ライブラリルーチンを使ったプログラミングを行なう必要がある。ただし、ごく基本的な情報のみを利用するサンプル実装を添付する。ユーザの使いやすさを考慮して、ライブラリ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具合に 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ているので、ユーザ自身で記述すべきソースコードの量は多くない。

3.3.2 実装

ライブラリは C 言語で約 3000 行となった。サーバ同様に、BPF や NIT を使用したユーザ空間のアプリケーションプログラムである。サーバとほぼ等しいフィルタリングを行なっている。

表 3.1: ライブラリルーチン一覧

dhcp_init	DHCP クライアントの初期化を行なう
dhcp_discover	DISCOVER を送り OFFER を集める
dhcp_request	REQUEST を送り ACK または NAK を待つ
dhcp_verify	アドレス割り当ての再確認をする
dhcp_extlease	アドレスの割り当て期限の延長をする
dhcp_decline	サーバに DHCPDECLINE を送る
dhcp_release	DHCPRELEASE による割り当ての期限を解放する
clean_param	情報交換に使われる構造体をクリアする
arp_reply	ARP 応答を送信して明示的に ARP キャッシュを作らせる
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

用意した関数の一覧を表 3.1に示す。ライブラリのインタフェースは簡潔で、得たい情報の種類や要求する期限などの情報を含む構造体を引数にして呼ぶと、各種の設定情報を含む構造体を戻り値として返す。前章に載せた図 2.8に各々のルーチンの処理する範囲を当てはめると図 3.3のようになる。淡いグレーで囲んである部分が各ルーチンの領域をおおむね示していて、斜体で対応する関数名を記述してある。また図 3.4にク

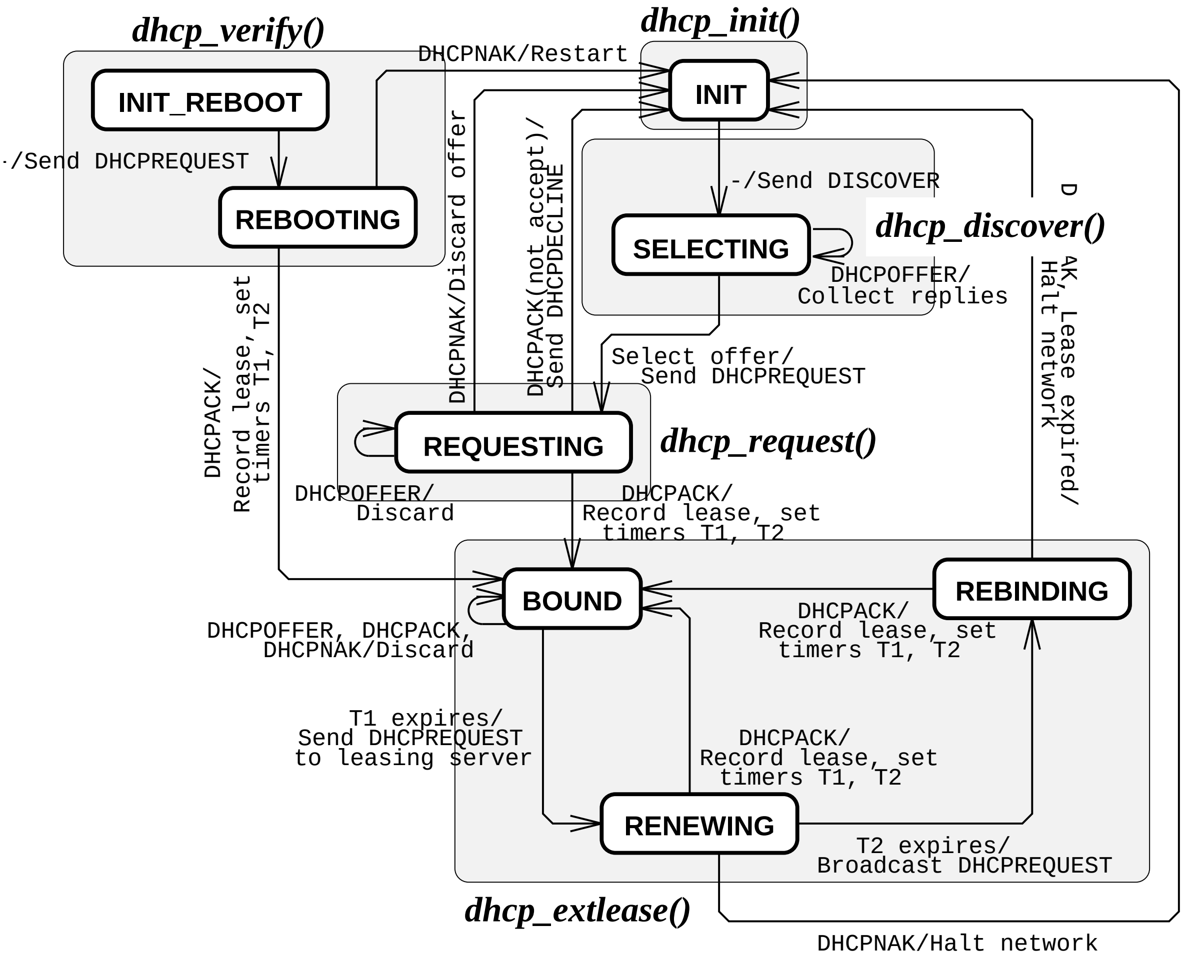


図 3.3: クライアントライブラリと有限状態機械の対応

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すべて省略して、概略を示すにとどめる。

サンプルクライアントの動作の概要を以下に示す。

1. 最初に *dhcp_init()* と *reset_if()* を呼び出す。例では *initialize()* の中で *reset_if()* を呼び出している (**3, 4** 行目)。

2. *construct_request()* で引数の構造体を作成し、*dhcp_discover()* を呼ぶ (**9, 10** 行目)。

3. 各サーバからの応答が構造体のリストとして返ってくる。

4. その中から *choose_offer()* で、一つ選択する (**13** 行目)。

5. それを引数として、*dhcp_request()* を呼び出す (**14** 行目)。ACK が返ってきた場合には、最終的な割り当て情報を含んだ構造体が戻ってくる。NAK を受けとった場合は、負の値が戻ってくるので 1からやり直す。

6. 割り当てを受けた情報に基づき、ネットワークインタフェースの設定 (*config_if()*) やルーティングテーブルの設定 (*set_route()*) などを行なう (**19** ~ **21** 行目)。

7. その後、割り当て期間 (lease) を延長するループ (**23** ~ **29** 行目) に入る。

8. *construct_extlease()* で引数を組み立て、*dhcp_extlease()* によって lease の延長を試みる。計算機資源の浪費を避けるために、 *dhcp_extlease()* は延長を開始するべき時刻までブロックしている (**24, 25** 行目)。

9. 延長に成功したら 8へ戻る。失敗したら *reset_if()* でネットワークインタフェースをリセットし (**26** 行目)、1に戻る。

3.4 リレーエージェントの設計と実装

3.4.1 設計

リレーエージェントの実装方針もサーバやクライアントと同様である。したがって All 1 のブロードキャストを用いたり、どのネットワークインタフェースから DHCP メッセージを受信したかを認識したりする必要がある。リレーエージェントも、簡潔なテキストファイルを記述することによって設定できるようにする。

```
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);
    if (dhcp_request(req, &netif, param) < 0) {
      continue;
    }

    /* 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 3.4: クライアントの実装例

3.4.2 実装

リレーエージェントはC言語で約950行である。クライアントやサーバと同様にBPFやNITを使用している。ただしクライアント・リレーエージェント間はAll 1のブロードキャストを使用するが、サーバ・リレーエージェント間は通常のユニキャストで通信すれば良いので、BPF(NIT)とsocketの双方を使用している(前掲の図3.1参照)。

複数のサーバが存在する場合に信頼性を向上させるため、リレーエージェントはクライアントから受けとった単一のDHCPメッセージを複数のサーバに送ることができる。これによりどれか一つのサーバに障害が発生していても、残ったサーバが処理をすることができる。したがってDHCP機構の信頼性が高くなる。

設定ファイルには、DHCPサーバのIPアドレスを列挙する。また信頼性を向上させるために、いくつのサーバへ転送を行なうかを指定する。もちろん複数のサーバへ転送を行ないたくない場合には、1を指定すればよい。複数サーバへ転送する場合には、クライアントから受信したDHCPメッセージを元にハッシュを行ない、転送先を選出する。このように転送先を分散させることによって、特定のサーバへ負荷が集中しないように負荷分散を試みている。

第 4 章

DHCP 実装の評価

本章では第 3章で解説した 3 つの実装について、実装目標を達成できたかという観点から評価を行なう。またクライアントライブラリの問題点を明らかにし、クライアントの設計と実装を再度行なう。

4.1　評価

4.1.1　サーバ評価

サーバ実装の “一般的なオペレーティングシステムの上で動作する” という目標は達成できた。現在のところ、本実装は BSD/386, NEWS-OS (CISC[1]版と RISC[2]版) という 2 つの BSD 系 UNIX の上で動作することを確認している。また SunOS4.1.3 の上でも動作確認を行なった。これらは Intel 80486/80386, MC68030, R3000, Sparc などの全くことなるアーキテクチャの CPU 上で動作している。これらの CPU の間には、次のような違いが一例としてあげられる。

- Little Endian と Big Endian[3]
- CISC と RISC
- ワード境界 (word alignment) を強要する/しない

これらの違いにもかかわらず、本質的な変更を加えることなしに動作することは、本実装の機種独立性が高い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る。SunOS でも BPF は利用可能であるが、標準装備の NIT を使用することにより、オペレーティングシステムを再構築せずに使用することが可能となっている。また BPF や NIT を利用することで、ネットワークインタフェースの種類や数に関係な

---

[1] Complex Instruction Set Computer の略称。複雑で高機能な命令セットを持つ CPU のこと。

[2] Reduced Instruction Set Computer の略称。命令セットを基本的なものに限定することで、プログラムの高速実行を目指している CPU のこと。

[3] 複数バイトにまたがる数をメモリ上に格納する際に、上位バイトから順に格納するか下位バイトから順に格納するかという違いがある。Endian の異なる機種間でデータを交換する場合には変換が必要となる。

く動作するようになっている。このように“オペレーティングシステムに対する独立性を高くする”という目標も達成できた。またテキストファイルを使った設定方法によって、ユーザの使いやすさも考慮した。*termcap*形式のデータベースを導入したことによって、設定の柔軟性も確保することに成功した。

サーバ実装は“大規模環境でも動作する”以外の目標は達成できた。“大規模環境でも動作する”に関しては、実験を行なっていないために確認できていない。なお今回のサーバはBOOTPクライアントのサポートも目的としていた。これに関しては、KA9QというMS-DOS上で動作するライセンスフリーのTCP/IPパッケージに含まれるBOOTPクライアントとの相互運用テストでも良好な結果が得られた。

4.1.2　クライアント評価

クライアントライブラリは、現在のところBSD/386およびNEWS-OS (RISC)での動作しか確認していない。サーバなどと同様に基本的な部分に関しての移植は容易である。しかしネットワークインタフェースにアドレスを設定する方法や、ルーティングテーブルを設定するための方法などが、同じUNIXでも系統によって異なる。現在のところSunOS用の実装は行なっていないが、将来的には実装を行ないたいと考えている。

4.1.3　リレーエージェント評価

リレーエージェントも同様にして、

- 一般的なOSで動作する
- OSに対する独立性を高くする
- ソースコードをすべて公表する

などの目標に関しては達成できた。“ユーザの使いやすさ”に関してはテキストファイルによる設定という方法で考慮したものの、設定の柔軟さの点でやや課題が残る。サーバ同様に、大規模環境での動作テストは行なっていない。

4.1.4　相互運用性テスト

DHCPに関しての議論はInternet Engineering Task Force (IETF)のInternet AreaにおけるDHC Working Groupで行なわれている。メーリングリストによる通常の議論のほかに、有志による相互運用性テストが1993年8月、1993年10月29日、1994年1月19日の3回開催されている。その目的は、独立に行なわれた各人の実装を持ち寄り、実際にテストを行なうというものである。本実装もシアトルで開催された第2回相互運用性テストにて実験を行なった。

当日は7つのサーバと12のクライアントが参加した。参加組織を以下に示す。オペレーティングの種類が特に記述されていないのは、UNIXと思われる[4]

1. Microsoft .. WINDOWS NTサーバ、WINDOWS NTクライアント、DOSクライアント
2. Sun Select .. サーバ、クライアント
3. HP .. クライアント
4. Boeing .. サーバ、クライアント
5. DEC .. クライアント
6. 当方 .. サーバ、クライアント、リレーエージェント
7. SGI .. サーバ、クライアント
8. Competitive Automation .. サーバ、クライアント
9. FTP Software .. WINDOWSサーバ、OS/2サーバ、WINDOWSクライアント、DOSクライアント

なお当初の予定では、リレーエージェントを介したテストも行なう予定であったが、時間の関係上から実施することが出来なかった。予定では我々のものの他にもう1つリレーエージェントがあるはずだったが、確認できなかった。

テストの結果、本実装は非常に良好な結果が得られた。また仕様への忠実度に関しては、極めて厳密な部類に属していた。そういった意味で、他の実装との相性は良かったといえる。

4.2 クライアントの再実装

本節では、我々の実装したクライアントライブラリ(Version 1と呼ぶ)の持つ問題点を明らかにし、新しくクライアントライブラリを設計し直す。また新たなクライアントライブラリ(Version 2と呼ぶ)の実装と評価を行なう。

4.2.1 クライアント Version 1の問題点

ライブラリVersion 1は図3.3に記述したように、有限状態機械を細かく分割している。これらはアドレスを使用するためのクライアントのアクションに対応している。だが、この分割によって以下のような問題点が生じた。

- 完全なクライアントを作る際に、やや複雑な制御構造を必要とする

---

[4] 具体的なオペレーティングシステムは確認できなかった。

- 内部構造を意識する必要がある (例 *dhcp_discover()* から返ってきた構造体のリストの内、選択したもの以外は明示的に free() してやらなければいけない)
- 関数とそれに渡す構造体の対応が、複雑で理解しづらい
- DHCP の仕様に熟知していないとライブラリが使用できない
- ライブラリの使い方を誤ると、DHCP のプロトコル仕様に違反してしまいやすい

さらに DHCP の仕様書には次のように記述されている。

> アドレスなどの設定情報が変化した可能性がある場合には、DHCP を使用して IP アドレスや設定情報の確認ないし再取得を行なうべきである。

Version 1 のライブラリではこのための関数として *dhcp_verify()* が用意されている。しかし DHCP の仕様書に掲載されている状態遷移図 (図 2.8参照) には verify に対応する状態が明確には示されていない。またサンプルクライアントも、起動時にオプション指定した場合のみ *dhcp_verify()* を呼び出すようになっている。したがって無線 LAN などのように “ネットワークが変わったも知れない[5]” という状況が頻繁に生じる場合に対応できない。

4.2.2 クライアントライブラリの改良

これらの欠点を解消するために、クライアントの改良を行なった。そのためにまず状態遷移図の見直しを行なった (図 4.1)。

まず図 2.8においては SELECTING 状態から REQUESTING 状態へ遷移するための条件が不明瞭であったため、新たに WAIT_OFFER 状態を付け加えた。これにより REQUESTING 状態へ遷移する条件が明確になった。

DHCP では以前使用していたアドレスを引き続き使用できるかどうか以前のサーバに確認できる。また、そのサーバが何らかの理由によって作動していなかった場合には、以前のアドレスを有効期限が切れるまで使い続けることができる。これらは REBOOTING 状態から BOUND 状態へ遷移することに相当する。しかし図 2.8 には DHCPACK を受信した場合、つまり確認に成功した場合についてしか記述されていない。

また実際には有効期限の残り時間によってはただちに RENEWING 状態や REBINDING 状態へと遷移する必要がある。しかし DHCP の仕様書の中では BOUND 状態に遷移する場合しか記述されていない。したがって BOUND 状態から REBINDING 状態へと遷移する矢印を付け加えた。

さらに “ネットワークが変化したかも知れない” 場合に行なう確認 (Verify) に相当する VERIFY 状態を追加した。BOUND 状態、 RENEWING 状態、REBINDING 状態のどの状態でも VERIFY 状態に遷移することができる。これにより “ネットワークが変わったかも知れない” という非同期の事象にいつでも対応できるようになった。

---

[5] 現在の UNIX のカーネルでは、ネットワークが変更されたことを確実に検知することはできない

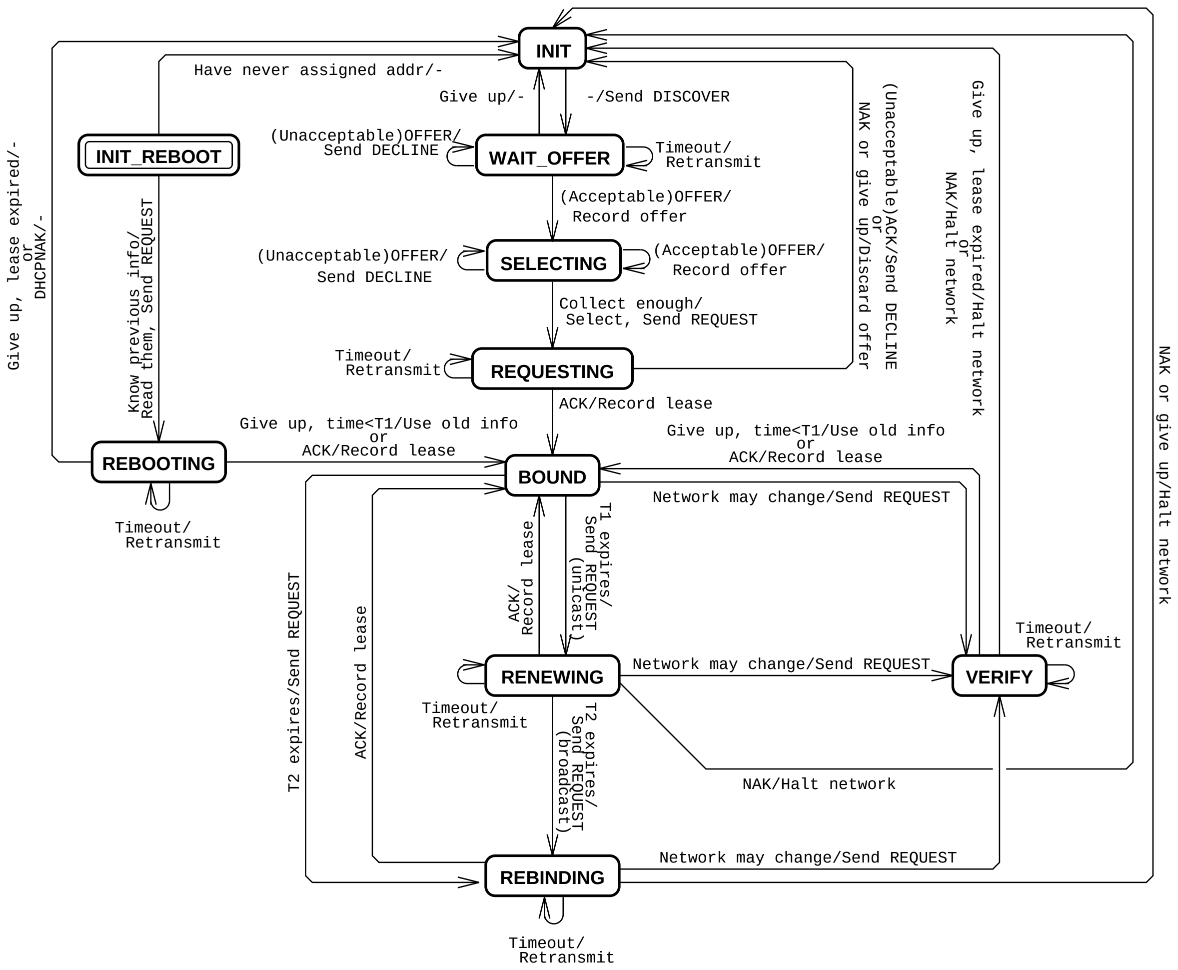


図 4.1: 改良した状態遷移図

このように変更された状態遷移図に基づいて再度クライアントライブラリの設計を行なった。まず Version 1 での問題点を解消するために、細分化されていた関数群を廃止し、単一の関数で全てを処理するようにする。また "ネットワークが変わったかも知れない" 場合、つまり確認 (verify) をしたい場合に、非同期にイベントを通知するための手段として、シグナル[6]を用いる。DHCPRELEASE をおくって割り当てを放棄するというのも非同期に発生するイベントだが、これの通知にもシグナルを用いるものとする。

4.2.3 クライアント Version 2 の実装

設計に従い、*dhcp_client()* という単一の関数を用意した。引数は次の 6個である。

1. ネットワークインタフェースの名前などを含む構造体へのポインタ
2. "Parameter Request List" オプションに含める値、DHCPOFFER を収集する秒数、 "Client Identifier" などの情報を含んだ構造体へのポインタ
3. 割り当てられた情報を利用するための関数へのポインタ。この関数はヘッダファイルに定義された構造体 dhcp_param を引数として受けとる
4. 複数の DHCPOFFER(dhcp_param のリストとして渡される) の中から、一つを選択する関数へのポインタ
5. DHCPOFFER を受けとった際に、提案された設定情報の内容に問題がないか調べる関数へのポインタ
6. DHCPACK を受けとった際に、提案された設定情報の内容に問題がないか調べる関数へのポインタ

さらに VERIFY 状態への遷移を引き起こすためには SIGUSR1[7]を、割り当てを放棄することをライブラリに通知するためには SIGUSR2[8]を利用することにした。また内部的に使用している *config_if()*(ネットワークインタフェースにアドレスを設定する) や *set_route()*(ルーティングテーブルを設定する)、*flushroutes()*(ルーティングテーブルをすべて消去する) などの関数は内部に隠蔽せず、ユーザが実際にクライアントを実装する際に利用できるようにしている。

ライブラリ Version 2 は状態遷移図に基づいて有限状態機械として実装してある。各状態に対応する関数を設けることによってモジュール化を進め、今後の仕様の変更や拡張に容易に対応できるようにした。中心部分である有限状態機械の部分のソースコードは図 4.2のように非常に簡潔にまとまっている。

またライブラリ Version 2 を使った際のクライアントの実装例は図 4.3のようになる。これは実際のソースの *main()* 部分をほぼそのままの形で掲載している。

---

[6] 別のプロセスに対して、例外事象の発生を非同期に伝達するための手段。シグナルを送られたプロセスは、実行中の処理を一時中断して例外処理のための関数に制御を移すことができる。

[7] 前述のシグナルの一つである。

[8] これも同様にシグナルである。

```
int (*fsm[MAX_STATES + 1])() = {
  init,
  wait_offer,
  selecting,
  requesting,
  bound,
  renewing,
  rebinding,
  init_reboot,
  verify,
  rebooting,
  verifying
};

prev_state = curr_state = INIT_REBOOT;
while (1) {
  if ((next_state = (*fsm[curr_state])()) < 0) {
    syslog(LOG_WARNING,
           "Some exception occurs in finite state machine");
    return(curr_state);
  }
  prev_state = curr_state;
  curr_state = next_state;
}
```

図 4.2: クライアント Version 2 の中心部分

```
int main(int argc, char **argv)
{
  int debug = 0;
  int n;
  struct dhcp_allocinfo alinfo;

  if (argc < 3) usage();
  --argc, ++argv;
  if (argv[0][0] == '-' && argv[0][1] == 'd') {
    debug = 1;
    --argc, ++argv;
  }

  if (argc < 2 || strlen(argv[0]) != strlen("ne0"))
    usage();
  strcpy(ifinfo.name, argv[0]);
  if (parse_haddr(argv[1], &ifinfo) != 0)
    exit(1);

  if ((int) signal(SIGTERM, (void *) finish) < 0) {
    syslog(LOG_ERR, "cannot set signal handler(SIGTERM)");
    exit(1);
  }
  if ((int) signal(SIGINT, (void *) finish) < 0) {
    syslog(LOG_ERR, "cannot set signal handler(SIGINT)");
    exit(1);
  }

  set_alinfo(&alinfo);
  return(dhcp_client(&ifinfo, &alinfo,
                     use_information, NULL, NULL, NULL));
}
```

図 4.3: ライブラリVersion 2によるクライアント実装例

4.2.4 クライアント Version 2 の評価

ライブラリ Version 1 の持つ問題点のうちの大半は解決できた。もちろんライブラリ Version 1 のもつ利点はすべてそのまま維持している。“複雑な制御構造を必要とする” という欠点に対しては、*dhcp_client()* という非常に大きな単位での抽象化を行なうことによってほとんど意識しなくて良くなった (図 4.3)。この抽象化により “関数と引数の対応が複雑” という欠点も解消することができた。またプログラムの構造化をうまく行なっているために、将来もし DHCP の仕様が変更されたとしても容易に対応できる。

“DHCP の仕様に熟知している必要がある”、“ライブラリの使用法を誤るとプロトコル違反してしまいやすい” などの欠点についても同様の理由により解決できた。ただ “内部構造を意識する必要がある” については、なお課題を残している。例えば Version 2 のクライアントライブラリも Version 1 同様に内部でタイマを利用している。しかもライブラリ外部で使用されているタイマをまったく意識しない。そのため *dhcp_client()* を呼び出す側がタイマを使っていても、内部で上書きされてしまい、うまく動作しないという欠点があげられる。

第 5 章

移動計算機への応用

DHCP は携帯型計算機を意識して設計されている。そのため携帯型計算機を持ち歩いた場合に、移動先のネットワークで再起動するだけで自動的に設定を終え、簡単に使用することができる。しかしこの方法では、論理的通信路 (セッション) をその度に終了しなければならないし、ネットワークにアクセスしながらの移動も不可能である。さらに問題となるのが、携帯型計算機に現在どの IP アドレスが割り当てられているかを任意の第三者が知る方法が存在しないことである。したがって、携帯型計算機から固定計算機へ新たに論理的通信路を張ることは可能だが、逆は不可能となる。そこで DHCP のほかに移動計算機をサポートするような機構が必要となる。

本章では、移動計算機サポートプロトコルの一つである Virtual Internet Protocol (VIP) への DHCP の適用について述べる。最初に VIP の概要を解説し、VIP と DHCP の機能が補完しあうことを示す。さらに DHCP を VIP へ組み込む方法を提案し、Version 1, Version 2 のクライアントライブラリそれぞれを利用した実装の解説と評価を行なう。また実際にこれらを運用した実験の結果についても解説する。

5.1 Virtual Internet Protocol の概要

IP で通信を行なう際、通常はホスト名を指定する。しかし最終的には Domain Name System[20][21] などを利用することにより、ホスト名を IP アドレスに変換して通信を行なう。IP アドレスは NetID と HostID から形成されているが、NetID はネットワークの識別子であり、ホストという存在の識別とは無関係である。つまり IP アドレスはホストを識別しているのではなく、ホストの位置を特定している。IP データグラムを配送する際の経路制御に IP アドレスが使用されるため、計算機が移動する限り IP アドレスは変化せざるを得ない。IP アドレスが変化した場合には通信を維持することが不可能なのはもちろん、相手を指定することすらも困難となる。

これらの問題点を解決するために VIP では、位置情報から独立した HostID を導入している。さらに既存のネットワーク層を 2 つに分割し、既存のネットワーク層の上にもう一つ仮想ネットワーク層なるものを設けた。仮想ネットワーク層は位置情報に依存しない純粋な HostID から構成されている (図 5.1)。

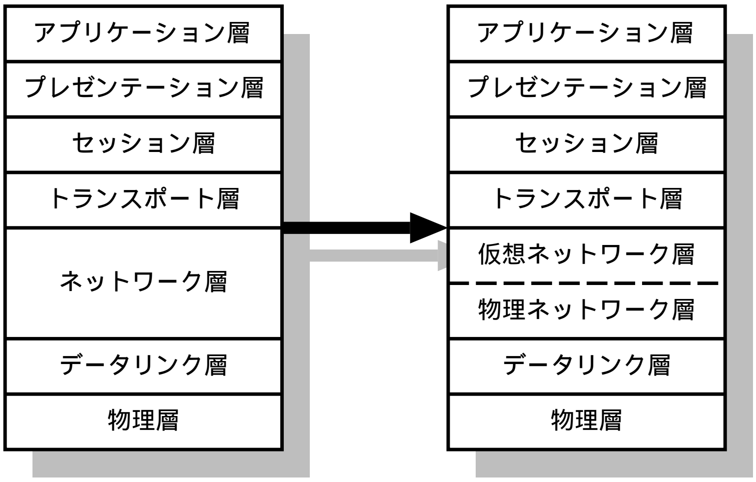


図 5.1: VIP 概念図

VIP で使用する HostID を VIP アドレスと呼ぶが、これは IP アドレスと全く同じ形式を持っている。IP を使って通信する場合には、宛先として VIP アドレスを指定するようになっている。しかし VIP アドレスと IP アドレスが同じ形式を持つために、上位プロトコルである TCP や UDP に変更を加える必要がない。これが VIP の提供する移動透過性である。

移動計算機が移動していない時、つまり通常時に接続されているネットワークをホームネットワークと呼ぶ。特にホームネットワークに接続されている場合は VIP アドレスと IP アドレスが等しくなる。逆に、VIP ホストが移動した場合には、移動先の位置情報を示す IP アドレスとホスト識別子である VIP アドレスが異なる。

移動している VIP ホストの実際に存在する物理ネットワークに IP データグラムを届けるためには、 VIP アドレスに基づいて IP アドレスと VIP アドレスの対応付けを行なう必要がある。これを役割を担うのが仮想ネットワーク層であり、VIP では特に「拡散キャッシュ法」という独自の方法を用いている。

VIP の詳細な仕様は文献 [3][4] に記述されているが、特に以下の項目が DHCP に関係する。

1. VIP ホストは、移動しても変化しない VIP アドレスと変化する IP アドレスの 2 つを必要とする
2. VIP ホストがホームネットワークに存在する場合には、VIP アドレスと IP アドレスが等しくなる
3. VIP 自体には、一時的に使用する IP アドレスを取得する機構がない

これらの項目から明らかなように、1を実現するために DHCP が応用できる。3には、DHCP のもつ動的なアドレス割り当て機構が適している。特別なケースとして VIP ホ

ストがホームネットワークに存在する場合には、2の要求が発生するが、DHCP はこれに適した手動割り当て (静的割り当て) の機構も持っている。

5.2 DHCP の VIP への応用

5.2.1 クライアント Version 1 による実装・評価

Version 1 のクライアントライブラリを使用して、VIP への応用を行なった。実装の目的は、ネットワークの変化を検出して自動的に IP アドレスを取得してくることである。

設計

現在の UNIX アーキテクチャでは、ネットワークが変化したというような情報をネットワークインタフェースからアプリケーションレイヤへ直接通知することができない。したがって BPF などを利用して、ネットワークインタフェースを監視する。もし一定時間なにも受信できなかった場合には、ネットワークインタフェースが変わった可能性を考慮して、DHCP による確認を行なうようにする。すなわちポーリングを行なうものとする。

具体的にはデーモンの形で実装を行ない、次のように動作するものとする。

1. 通常のように、DHCP により IP アドレスの割り当てを受ける
2. ポーリングを行なう。何らかのパケットを受信できた場合にはネットワークは変化していないと見なすことができるので、ある間隔休止してから 2に戻る
3. もし何のパケットも受信できなかった場合には、ネットワークが変化している可能性がある。したがって DHCP による確認を行なう
4. もしネットワークが変化していないことが確認できたら、2 に戻る
5. もしネットワークが変化している場合には、確認に失敗する。新たに IP アドレスを取得するために 1に戻る

実装

初期に行なった実装 (vipd Version 0.1 と呼ぶ) ではプログラムを起動すると最初に fork[1]する。子プロセスは DHCP を起動し、親プロセスはポーリングを行なうようになっている。親プロセスが、何のパケットも受信できなかった場合 (前掲の動作概要の 3に相当する) には子プロセスをシグナルによって終了させる。親プロセスは、*wait()*[2]の後に再度 fork するように実装されている。親プロセスの中心部分を図 5.2 に示す。

---

[1] UNIX において新たにプロセスを作成することを意味する。

[2] fork した場合に、新たにプロセスを作成したプロセスのことを親プロセス、作成された方のプロセスを子プロセスと呼ぶ。*wait()* を呼び出すことで、子プロセスの終了状況を得ることができる。

しかしVersion 0.1はうまく動作しない。まずforkが頻繁すぎるために負荷が高い。また、親プロセス・小プロセスの双方で頻繁にタイマを使用しているのだが、forkの前に設定したタイマが親プロセスと小プロセスの間で相互に影響を及ぼし合うという副作用が発生する。プロセスがforkするためにデバッグしづらいという欠点もある。

```
while ((cpid = fork())) {
  i = 1;
  sleep(6);
  while (i) {
    if (polling(netif.name)) {
      sleep(6);
    } else {
      verify_f = 0;
      if (kill(cpid, SIGHUP) != 0)
        if (errno != ESRCH) {
          syslog(LOG_WARNING, "kill() error in main()");
          exit(1);
        }
      wait(0);
      i = 0;
    }
  }
}
```

図5.2: vipd Version 0.1の中心部

このような欠陥を踏まえて、Version 1のライブラリによる実装を再度行なった(Version 0.9)。vipd Version 0.9はforkせずに単一のプロセスとして動作する(図5.3)。最初にアドレスを取得すると、ポーリングを行なう。パケットを何も受信できなかった場合には、DHCPによる確認をする。

DHCPでは割り当て有効期限の希望値をクライアントが指定できるが、有効期限の延長の場合にも希望する有効期限を指定できる。また確認と同時に有効期限の延長を申請できる。したがってvipdの用いる確認では有効期限の延長も申請している。

確認に失敗した場合には、再度アドレスを取得する。これらを繰り返してvipd Version 0.9は動作するようになっている。

実験結果

我々が実装したvipd(Version 0.9)やDHCPを用いて、第28回IETFミーティングおよびIPミーティング'93にて実験デモを行なった。実験にはDHCPサーバ、DHCPリレーエージェントの他に、VIPを実装したBSD/386マシン(vipdを使ったDHCP

```
while(dhcp(param, flag, sockfd) < 0);

while(1) {
  alarm(0);
  sleep (interval);
  if (polling(netif.name)) {
    if (dhcp(param, VERIFY, sockfd) < 0) {
      reset_if(&netif);
      ipaddr.s_addr = 0;
      while(dhcp(param, DISCOVER, sockfd) < 0);
    }
  }
}
```

図 5.3: vipd Version 0.9 の中心部

クライアント) と DOS マシン (BOOTP クライアント) を用いた。その時のネットワーク構成を図 5.4に示す。この実験デモにより DHCP サーバと DHCP リレーエージェントが安定して動作すること、KA9Q 上の BOOTP クライアントと DHCP サーバの組合せでも動作することなどが確認できた。デモでは図中の SONY NEWS から移動計算機 (BSD/386) に ico[3]のウィンドウを開き、ネットワークを移動しても ico が動き続けることを視覚的に示した。この時にキーボードには一切手を触れない。これは VIP によって移動透過性が保たれることと、DHCP によって (vipd によって) アドレスを自動的に取得できることによって初めて可能となる。 KA9Q マシンでも同様にリモートログインの論理的通信路を維持したまま移動可能なことを示した。

このように DHCP と VIP を組み合わせることにより、ユーザがネットワークアドレス等の詳細を意識せずに、しかも作業を中断する事なく移動できる[4]。また任意の第三者が移動計算機に向かって新たに論理通信路を開く場合にも、VIP アドレスさえ知ることができれば良く、相手の位置 (IP アドレス) を意識する必要がない。

VIP は既存の IP ホストとの互換性を保っている。したがって世界中のどこに移動しても一時的に使用する IP アドレスが取得できれば、通信が可能となる。vipd が実装される前は IP アドレスを取得するために、手動で設定する方法、独自のデーモンによって設定する方法、改造された BOOTP を利用して設定する方法などがとられていた。しかし手動による設定は非常に煩わしい上に、無線 LAN を使用するような場合に生ずると思われる頻繁なネットワーク変更に対応できない。また独自デーモンによる方式では、サーバの存在しないところでは利用することができない。改造された BOOTP による方法も同様である。これでは VIP の持つ IP ホストとの互換性という利点が半

---

[3] X ウィンドウ上で動作するプログラム。多面体がウィンドウ内を動く。

[4] ただしアプリケーションによっては時間切れとして処理を終了してしまうものもある。

減してしまう。だがDHCPはインターネットにおいて広く提案されているため、今後世界中で利用可能となることが予想される。したがってVIPの持つ利点をそのまま活かすことができるだろう。

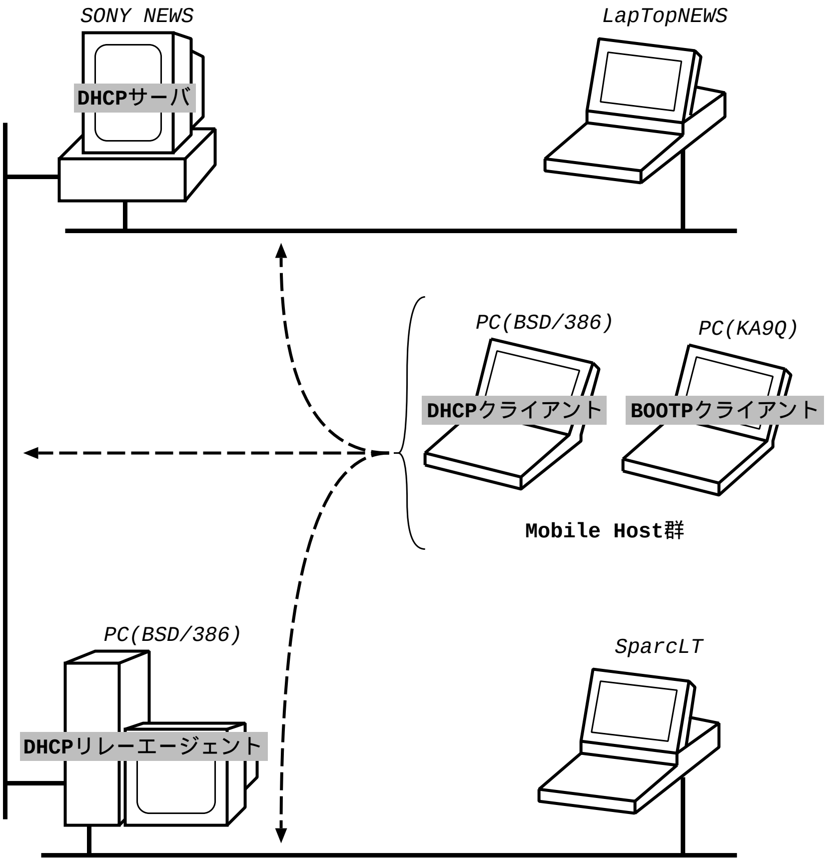


図 5.4: デモ用ネットワーク

評価

vipd Version 0.9はうまく動作したのだが、割り当ての有効期限を遵守することができないという欠点がある。普通に使用している限りにおいてはポーリングの間隔が短いために、絶えず有効期限が更新される。したがってこのような問題は発生しない。しかしDHCPサーバに障害が発生した場合には、有効期限の延長ができない。通常のDHCPクライアントであれば、有効期限が切れた場合にはただちに割り当てられたIPアドレスの使用を中止する。しかしvipd Version 0.9では実装上の構造からこれができない。有効期限を無視することはDHCPの仕様に違反しているため、非常に問題である。vipd Version 0.1はforkにより完全な形のクライアントを起動するため、有効期限を遵守できるはずであったが、うまく動作しなかった。

これらの欠点は、第 4.2.1節に指摘したクライアントライブラリ Version 1 のもつ欠陥によるところが大きい。したがってライブラリ Version 1 による vipd の実装ではなく、ライブラリ Version 2 による vipd の実装が望ましい。実際のところ、 vipd Version 0.9 のもつこれらの欠点を解消するためには新たなクライアントライブラリが必要であるというのが、ライブラリを改良した理由の一つである。

5.2.2 クライアント Version 2 による実装・評価

設計および実装

Version 1 のクライアントライブラリの欠点を考慮して、新たに設計したライブラリ Version 2 は vipd を実装するのに適した特徴を持っている。とりわけシグナルによって非同期にアドレス確認 (verify) を行なうことができるため、vipd に非常に適している。また抽象化のレベルが大きいためにクライアントの内部構造をほとんど意識する必要がなく、vipd の制御構造との切り分けが明確に行なえる。

図 5.5は vipd Version 1.0 の中心部分である。vipd Version 1.0 は vipd Version 0.1 のようにポーリングを行なう親プロセスと、DHCP のクライアントである子プロセスの二つが協調して動作する。vipd Version 0.1 と異なるのは、 fork してから子プロセスは DHCP のクライアントに exec[5]することと、 fork は一度しか行なわなず、親プロセスが子プロセスを殺さないことである。

親プロセスは子プロセスを生成すると、無限ループに入る。ポーリングを行なって何のパケットも受信できないと、子プロセスにシグナルを送る。シグナルを受けとった子プロセス (DHCP クライアント) は状態遷移を起こし、確認をおこなうための VERIFY 状態に移る。もちろん VERIFY 状態に移るべきでない場合 (例えばシグナルが届いた時に SELECTING 状態だった場合など) は、遷移しない。逆に有効期限の延長を行なっている最中であっても確認を行なえるようになっている。なお Version 2 のクライアントライブラリでは、確認の際に有効期限の延長を行なわない。

旧バージョンとの比較評価

vipd Version 1.0 は Version 0.9 と異なり、有効期限を順守することができる。加えてクライアントライブラリ Version 2 を利用することにより、プログラムの大幅な簡略化がなされている。

現在は BPF や NIT によるポーリングによってしか、ネットワークの変化を検出できないうえに、実際にネットワークが変化したかどうかは不確定である。しかしながら、クライアントライブラリ Version 2 は VERIFY 状態への遷移が非同期なシグナルにより行なえる。したがって将来的にネットワークデバイスのドライバを拡張できたような場合には、*polling()* 関数を変更するだけで容易に対応可能である。

---

[5] 前述の fork によってプロセスを生成する場合、いくつかの点を除いて子プロセスは親プロセスの複製となる。したがって fork をした後に、何らかの方法を使って新たなプログラムを実行するように指定しない限り、親プロセスと全く同じ処理を行なうことになる。新しいプログラムを実行させる方法として UNIX で提供されているのが exec() という関数であり、これを実行することを 「exec する」というように表現する。

```
if ((childpid = fork()) < 0)
  exit(1);
if (childpid == 0) {
  if (execl(dhcpc_path, dhcpc_name,
      "-d", argv[0], argv[1], (char *) 0) < 0) {
    exit(1);
  }
}
while(1) {
  sleep(INTERVAL);
  if (wait3(NULL, WNOHANG, NULL) == childpid) {
    syslog(LOG_WARNING, "DHCP client had died");
    exit(1);
  }
  if (polling(argv[0]) != 0) {
    kill(childpid, SIGUSR1);
  }
}
```

図 5.5: vipd Version 1.0 の中心部

残念ながら vipd Version 1.0 による本格的な実験は行なっていない。小規模な実験によっては既に動作を確認しているので、おそらく本質的な問題は発生しないと思われる。なお vipd もクライアントライブラリ同様に、BSD/386 でしか動作を確認していない。

第 6 章

DHCP に関する考察

本章では、現在の DHCP の持つ問題点と仕様上の限界を示す。それらの問題点や限界をクリアするためには複数の DHCP サーバ同士が協調して動作することが必要である。そのための仕組みとして DHCP サーバ間プロトコルについて考察する。

6.1 DHCP の問題点

DHCP の仕様書 (RFC1541) には、仕様上の問題点がいくつかある。本節では RFC1541 が潜在的に持つ問題点を 2 つ指摘する。ただし最初の問題は、新しい仕様書 [2] ではうまく回避されている[1]。

6.1.1 複数サーバ問題

DHCP にはリレーエージェントを設けられていて、一つのサブネットに必ずしも一つのサーバが存在する必要がない (ただし少なくともリレーエージェントかサーバのどちらかが存在しなければならない)。逆に、サーバやリレーエージェントを一つのサブネットに複数設置することによって、アドレス割り当て機構の信頼性を向上させることも可能である。しかし RFC1541 に規定されている仕様にしたがうと、サーバ同士が通信することができないために以下に説明するような問題が発生する。

DHCPOFFER と DHCPNAK の競合

RFC1541 には、DHCP サーバが DHCP クライアントからの DHCPDISCOVER メッセージを受けとった場合に次のような動作を行なうように規定されている。

> サーバは DHCPDISCOVER をメッセージを受けとると、そのクライアントに提案するアドレスを選びだす。もし割り当て可能なアドレスがない場

---

[1] IETF の DHC WG では、相互運用性テストの結果やメーリングリストでの議論を通じて、より高い完成度をもつ仕様書を完成しようと作業を続けている。我々もこれらの活動に参加し、貢献してきた。DHC WG の成果の一環として RFC を改訂したドラフト [2] などが WG 内部で公開されているが、これらは一般にも入手可能である。本節で指摘する問題点のうちの最初のものはこのような過程を通じて解決された。

> 合、サーバはシステム管理者に報告しても良いし、さらにクライアントにDHCPNAK メッセージを送ることにしても良い。

問題なのは、「DHCPNAK メッセージで応えても良い」点である。仮に2つのサーバが同一サブネット上に存在する場合 (リレーエージェントによって別サブネットへ中継される場合も、同一サブネットに存在するとみなす) に、一方のサーバは DHCPOFFER を送り返したにもかかわらず、もう一方のサーバがそれよりも先に DHCPNAK を送り返してしまう可能性がある (図 6.1)。しかも RFC1541 によると、DHCP クライアントは、DHCPNAK を受けとるとその後に届く DHCPOFFER は無視してしまう。この場合、利用できるアドレスが存在するにもかかわらず、DHCP クライアントはアドレスの割り当てを受けることを断念してしまうかも知れない。

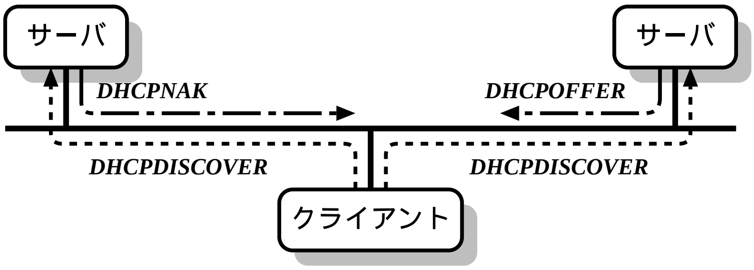


図 6.1: 複数サーバ問題 1

最新のドラフト [2] では「DHCP サーバは割り当て可能なアドレスがない場合でも DHCPNAK を返さない」と規定することによって、この問題を回避している。しかしながらその場合は逆に、両方とも割り当て可能なアドレスを持たない場合に、DHCPNAK を受けとることはない。したがってタイムアウトと再送を繰り返し、最終的に DHCP の利用を断念するまでユーザは待たされてしまう。

DHCPACK と DHCPNAK の競合

同様の問題が、DHCPREQUEST メッセージでも起こる可能性がわずかに存在する。問題となるのは次の箇所である。

> クライアントが DHCPREQUEST メッセージに含めた情報がサーバが以前に割り当てたものと等しい場合、あるいは有効期限の延長要求 (“IP address lease time” オプションによって区別できる) が受け入れ可能である場合には、サーバは DHCPACK メッセージを返す。それ以外の場合には DHCPNAK メッセージを返す。

ある種の条件が重なると、次に示すように DHCPACK と DHCPNAK の競合が発生する。

1. クライアントがサーバ 1 にアドレス A を割り当てられる (設定情報は 2 次記憶に保存される)

2. クライアントがシャットダウンされ、再起動される

3. クライアントはアドレス A を使い続けられるか確認する

4. その際にサーバ 1 が障害などにより停止していると、確認に失敗する (あるいは 2 次記憶の情報が破壊されていたため、確認ができないとする)

5. 確認に失敗すると通常はクライアントはアドレス A を期限が切れるまで使用する

6. 期限が切れるとクライアントは INIT 状態に移り、新たにアドレス B を取得する

7. クライアントはアドレス B を使い続けられるか確認する

8. サーバ 1 は DHCPNAK を返し、サーバ 2 は DHCPACK を返す (図 6.2参照)

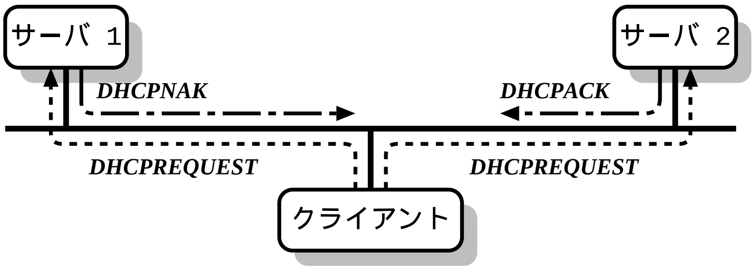


図 6.2: 複数サーバ問題 2

このような競合は稀にしか発生しないが、2 次記憶を使わない (情報を保存しておかない) クライアント[2]が存在すると、このような状況が頻繁に発生する可能性がある。

大半の DHCP クライアントは、アドレスが付け変わる場合には再起動を必要とすると思われる。少なくともセッションを終了しなければならないことは間違いない。このような状況になると 6の場合には、クライアントは再起動してしまう可能性がある。その場合には、アドレス A を使い続けるよりも 4の段階で新しくアドレス B を割り当てられる方が望ましい。さらに望ましいのはサーバ 1 と 2 が協調し合うことで、4が起こらないようにすることである。すなわちサーバ 2 が、1 の代りに DHCPACK を返すことができるように、あらかじめ互いのデータベースを交換しておく。

---

[2] 例えば X 端末などが考えられる。

6.1.2　アドレスプール枯渇問題

こちらの問題は、現在のDHCPの限界を明らかにするものである。現在のDHCPでは、アドレスプールに関して次のような仮定を行なっている。

> DHCPサーバはネットワークアドレスの集合を保持していて、クライアントから届くアドレス割り当ての要求を満たすことができると仮定する。

したがってアドレスプールに割り当て可能なアドレスがなくなった場合のことについては触れられていない。実はRFCになる前、つまりInternet Draft[3]の段階では「アドレスプールにはサーバ間プロトコルによって、常に割り当て可能なアドレスが存在すると仮定する」と記述されていた。しかしサーバ間プロトコルに関する公式の文書などは一切存在せず、そのためRFCになる段階で前掲のような記述に訂正されてしまった。このように現在のDHCPサーバはアドレスプールが枯渇しても補充することができない。また複数サーバ間でアドレスプールが重複する場合は考慮されていないため、アドレスプールは互いに独立した集合である。

近年のインターネットにおいて、IPアドレスは近い将来に確実に不足することが指摘されている。DHCPでは使われなくなったIPアドレスをサーバが回収し、別のホストに割り当てることでIPアドレスの効率的な利用ができることになっている。しかし実際にはアドレスプールは管理者の手によって設定される固定した集合である。したがって既存の人手による管理に比べて、劇的に使用率が改善されることは考えられない。

6.2　サーバ間プロトコルに関する考察

このような欠点を解決するためには、DHCPにサーバ間プロトコルを導入することが望ましい。アメリカのペンシルバニア州にあるBucknell大学では、ごく簡単なサーバ間プロトコルが考案され、試験的に実装された[4]。本節では文献[22]に述べられているBucknell版のサーバ間プロトコルについて概要を紹介し、それからサーバ間プロトコルの設計方針について考察する。

6.2.1　Bucknell版サーバ間プロトコル

Bucknell版のサーバ間プロトコルは極めて単純である。まずマスターサーバが存在する。マスターサーバはアドレスプールを一手に引き受けて管理する。マスターサーバの下には複数のスレーブサーバが存在する。またマスターサーバは、2次記憶にアドレスプールを持っている(図6.3)。

アドレスプールを読み込んだマスターサーバは、未割り当てのアドレスをスレーブサーバに対して均等に配布する。スレーブサーバは配布されたアドレスを、DHCPク

---

[3] IETFで発行される技術的メモであり、その多くは各WGが発行する。将来的にRFCになる場合もあるが、そうでない場合もある。

[4] 学部の授業の一環としてプロジェクトが実施されたようであるが、詳細は不明である

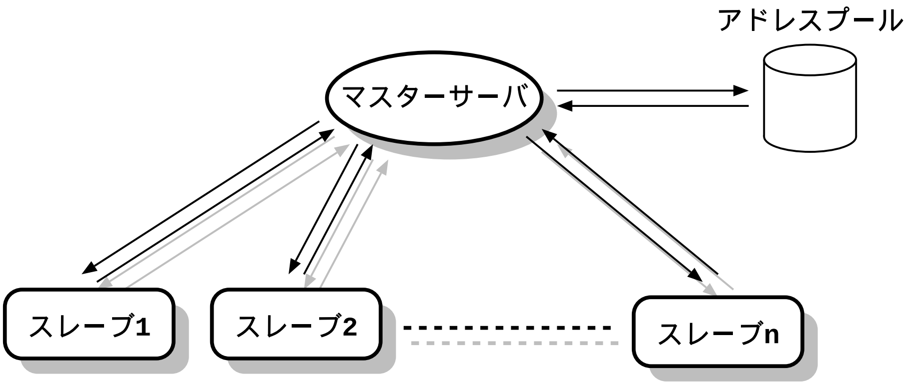


図 6.3: bucknell 版サーバ間プロトコル

ライアントに割り当てる。スレーブサーバはアドレスのバインディング、つまりどのアドレスを割り当てたかということを、マスターサーバに対して定期的に通知する。マスターサーバも、未割り当てのアドレスの数が各スレーブサーバで均等になるように、定期的に再配布を行なう。

Bucknell 版のサーバ間プロトコルによって得られる利点は「複数あるサーバのどれか一つでも動作していれば、DHCP クライアントはアドレスの割り当てを受けられる」ということだけである。したがって第 6.1.1節で上げた問題点を解決することはできない。また試験的な実装にしても、アドレスプールに含まれる情報は IP アドレスのみであるため、非実用的と言える。

6.2.2 設計に関する諸問題

複数サーバによる競合問題に関しては、簡単な peer-to-peer のサーバ間プロトコルで対処可能と思われる。しかしアドレスプール枯渇問題や、ひいては IP アドレス自体の枯渇問題に対処するためには、階層的なサーバ群の構築が必須である。構造化をする際に、あるサイト内の構造化は比較的イメージしやすい。一例として図 6.4のようなものが考えられる。

各サイトにはドメイン全体を管轄する最上位のサーバが存在する (ただし Secondary Server によるバックアップが存在するだろう)。各サブネットには DHCP サーバやリレーエージェントが最低一つ存在し、アドレスの割り当てを行なう。両者の中間レベルにサーバが存在することも考えられる。

さらに上のレベルの階層化はイメージすることが困難である。Domain Name System (DNS)[20][21] と連係して構成されるのか、それとも DHCP のサーバ群だけで階層構造が構成されるのか、今の段階では決定できていない。これを決定するためにはサーバ間プロトコルに対する、より具体的な要求を明確に把握する必要がある。

階層的なサーバ間プロトコルを設計する際に障害となることがいくつか考えられる。第一に、通常のホストの取り扱いをどのように行なうかという問題があげられる。 IP

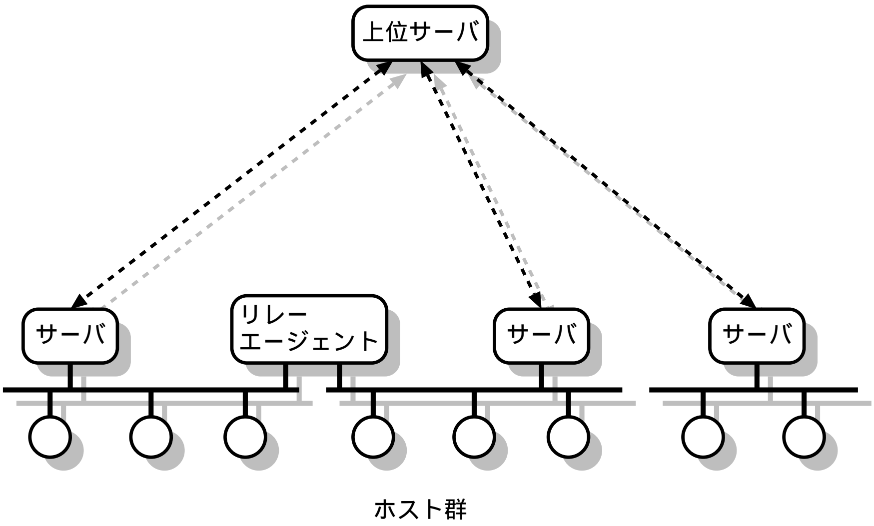


図 6.4: サーバ間プロトコルのモデル例

アドレスの効率的な利用を行なうためには、IP アドレスの付け変えが発生すると思われる。インターネット上の全ホストが DHCP クライアントであり、しかも比較的短い有効期限を持つ動的割り当てである場合には、アドレスの付け変えもある程度自動的に行なうことができるだろう。しかしながら実際には通常のホストも存在するし、有効期限の長いホストも存在する。こういったホストの IP アドレスを付け変えるためには人間の介入が必要とされる。また IP アドレスの付け変えに伴って生ずる様々な影響 (例えば複雑な経路制御の記述や、Domain Name System への登録内容をどのように変更するかなどの問題が発生するだろう) に、どのように対処するかという問題もある。

第二に、サーバ間プロトコルにおいて、どのようにアドレスプールのエントリを表現するかということが問題となる。現在の DHCP は一つのメッセージにつき一つのエントリに関する情報しか含まない。しかしながら多数のエントリを一度に交換する必要のあるサーバ間プロトコルでは、より効率的で強力な記述力を持つ表現方法を考案する必要がある。

また、階層的にアドレスプールを管理する場合に、下位のサーバにアドレスプールを配分することになる。この際に上位サーバが、下位サーバに対して割り当てなどの方針 (Policy) を伝達する必要があるかも知れない。そのような方針をどのように表現して伝達するのかが問題となる。

これらの諸問題に対する解決方法を考案することが、今後の大きな課題である。

第 7 章

結論

7.1　まとめ

インターネットにおいては絶えず計算機が接続と切断を繰り返す。インターネットに接続される計算機はIPアドレスなどのネットワーク上の共有資源を割り当てを受けて、設定に組み込まなければならない。逆にインターネットから撤去される計算機の使用していた共有資源は回収する必要がある。資源の割り当て作業や計算機の設定作業は、煩わしいものである上に間違いを犯しやすい。したがってこれらの作業を集中的・自動的に行なう機構が求められていた。そのために提案されているのがDHCPである。

本研究では第一に、初めてライセンスフリーの形でDHCPの全てのモジュールを実装した。本実装は非常に機種独立性が高く、オペレーティングシステムのユーザ空間で動作する。これにより汎用性の高さを確保することが可能となり、複数のUNIX上で動作する実装を行なった点が特徴的である。サーバやリレーエージェントについては大規模性や設定の柔軟性を確保することを目標とした。クライアントは、ユーザの個別の要求に柔軟に対応できるようにライブラリ形式の実装を行なった。本実装の完成度の高さは、アメリカで開催された第2回相互運用性テストにて実証済みである。

第二に、DHCPの応用例としてVIPへの適用を行なった。DHCPとVIPは非常に相性が良く、機能を補完し合う。DHCP単独ではごく初歩的な移動しかサポートできないが、DHCPをVIPと組み合わせることによって携帯型計算機の透過的かつ円滑な移動が可能となることを示した。その可能性を実証するために、動的にIPアドレスの取得を行なって設定を行なうvipdの実装と評価を行なった。世界中の任意の場所でIPアドレスが取得可能となるDHCPを応用することによって、VIPの有効性が向上した。

またvipd Version 0.9の実装の際に明らかになったクライアントライブラリVersion 1の欠点を改善し、新たにVersion 2の実装を行なった。クライアントライブラリVersion 2はVersion 1よりもさらにユーザの使いやすさを考慮している。vipd Version 0.9に関しては、実験ネットワークにおいて安定して動作することを確認した。vipd Version 0.9にはDHCPの仕様に違反する部分が存在したが、クライアントライブラリVersion

2 を利用した vipd Version 1.0 を実装して、この問題点を解決した。

最後に現在の DHCP のもつ矛盾を示す事例を 2 つ指摘した。この問題は、同一サブネットに複数のサーバが存在する場合に、各々のサーバからの応答が互いに矛盾したものであるために発生する。これはサーバ間での情報の授受が行なわれない限り本質的に発生するものであり、問題を解決するために、サーバ間プロトコルが必要であるということを示した。

さらに IP アドレスの枯渇問題などの、より複雑な問題に対処するためには、サーバ間プロトコルを階層的なものにする必要があることを示した。サーバ間プロトコルを設計する際には、通常のホストをどのように取り扱うのか、あるいは DNS との関係はどうなるのかという問題がある。またアドレスプールのエントリをどのように効率的に記述するかという問題も発生する。本研究では、これらの問題点についても指摘し、今後の展望とした。

7.2 今後の課題

我々の実装も、DHCP の仕様を完全にサポートしていない部分がある。以下にサポートしていない事項を列挙する。

- クライアントが “file” フィールドに希望するファイルのパス名を設定できない。またサーバもこれを処理できない。
- クライアントが “sname” フィールドに希望するホスト名を設定できない。またサーバもこれを処理できない。
- IP フラグメントの再構成 (Reassemble) を行なえない。
- サーバが “Class-identifier” オプションを処理できない。したがってこれと深い関わりのある “Vendor Specific Information” オプションも処理できない。

今後はこれらの欠点を解消する必要がある。その他にも、

- 設定ファイルの記述をさらに簡潔にする
- より大規模な環境での長期運用テストをする
- 実装を配布するために運用テストを重ねて安定性を検証する
- クライアントライブラリ Version 2 および vipd Version 1.0 のさらなるテストを行なう
- (一定時間内にどれだけの DHCP クライアントを処理できるかという) パフォーマンスのテストを行なう
- クライアントライブラリ Version 2 の移植

- DOS マシンへの DHCP クライアントの移植

などがあげられる。

また最大の懸念である “サーバ間プロトコル” の設計を完了し、それをインターネットに提案することが望ましい。そのためには “サーバ間プロトコル” の実装を行なって評価する必要がある。

謝辞

本研究を進めるにあたり、御指導頂きました慶應義塾大学環境情報学部助教授の徳田英幸博士と村井純博士に感謝します。また絶えず御助言と御指導を頂きました (株) ソニーコンピュータサイエンス研究所の寺岡文男博士に感謝します。

実装の際に直面した問題に対する解決法を提案し、助言して下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および北陸先端科学技術大学院大学情報科学研究科助教授の篠田陽一博士に感謝します。

本研究を進めていく上で、様々な議論をして下さった慶應義塾大学徳田・村井研究室の諸氏と VIP Working Group をはじめとする WIDE Project の諸氏に感謝します。最後に仕様上の疑問点や詳細に立ち入った議論に根気よく対応して下さった IETF の DHC Working Group のメンバーに感謝して謝辞と致します。

参考文献

[1] Droms, R.: *Dynamic Host Configuration Protocol*, RFC 1541, October 1993.

[2] Droms, R.: *Dynamic Host Configuration Protocol*, Working memo of host-conf WG, Janurary 1994.

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

[4] 植原 啓介, 寺岡 文男: 移動ノード」, 1992 年度 WIDE プロジェクト報告書.

[5] Gerich, E.: *Guidelines for Management of IP Address Space*, RFC 1466, May 1993.

[6] Finlayson, R., Mann, T., Mogul, J., and Theimer, M.: *A Reverse Address Resolution Protocol*, RFC 903, June 1984.

[7] Croft, B. and Gilmore, J.: *Bootstrap Protocol (BOOTP)*, RFC 951, September 1985.

[8] *Information processing systems — Open Systems Interconnection — Basic Reference Model*, ISO 7498, 1984.

[9] Wimer, W.: *Clarifications and Extensions for the Bootstrap Protocol*, RFC 1542, October 1993.

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

[11] Plummer, D.: *An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address*, RFC826, October 1982.

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

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

[14] Simpson, W.: *The Point-to-Point Protocol (PPP)*, RFC 1548, December 1993.

[15] McGregor, G.: *The PPP Internet Protocol Control Protocol (IPCP)*, RFC 1332, May 1992.

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

[17] McCanne, S. and Jacobson, V.: *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, 1993 Winter USENIX conference proceedings.

[18] *SunOS Reference Manual — Network Interface Tap*, Sun Microsystems, Inc.

[19] *SunOS Reference Manual — Termcap*, Sun Microsystems, Inc.

[20] Mockapetris, P.: *Domain Names - Concepts and Facilities*, RFC1034, November 1987.

[21] Mockapetris, P.: *Domain Names - Implementation and Specification*, RFC1035, November 1987.

[22] DeHart, Darick., Gilmore, Ryan., and Post, Bill.: *Documentation for the Server-to-Server Protocol in DHCP*, Bucknell University, 1992.

[23] 冨永 明宏, 寺岡 文男, 村井 純: 動的ホスト設定プロトコル(DHCP)の実装と評価」, 情報処理学会 マルチメディア通信と分散処理研究会 論文集 Vol.93, No.2