# Problems and Solutions of DHCP

*- Experiences with DHCP implementation and Operation -*

Akihiro Tominaga
Osamu Nakamura
Fumio Teraoka
Jun Murai

## Abstract

*An implementation of DHCP (called WIDE version) has been made and beta-tested in some practical uses since March 1994. WIDE version DHCP achieves full semantical compatibilities with the exisiting UNIX system architecture, the architectural independency, the administration efficiency on the UNIX environment, and the interoperability with BOOTP-based clients and other DHCP implementations. As the results of the experiences, several problems of DHCP have been found: a) The DHCP client is unable to perform a* verification *simultaneously with a* lease extension. *b) A design difficulty was claimed since the usage of a DHCP message is too complicated. c) There is no acknowledgement mechanism for releasing addresses. d) A concentration of* address reuse *requests were detected under some circumstances. These problems have been solved by modifying DHCP specifications. Other problems have also been found such as ; a) lack of fault-torelancy in DHCP, b) no active notification from server to client, and c) no mechanism to retrieve addresses by server. This paper describes the design and implementation, encountered problems during the beta-tests, and proposes their solutions. Requirements for the future possible changes on the protocol also are discussed.*

## 1 Introduction

Personal computers has become smaller such as notebook PCs for mobility without lack of performance. Also PCMCIA plug-on network card devices enabled the network connection of these computers. Using TCP/IP connection on PCs with UNIX or other Operating System like DOS, Windows, WindowsNT, etc. has become possible. It is very natural to say by these technological solution that made a mobile computing environment, computers which moves one network to another.

To temporary connect the mobile computer to IP network environment, IP address, subnet mask, default gateway, and other informations are required to communicate with other nodes on the network. There are conventional ways to configure these network parameters. Most traditional ways to connect computers to network, is to configure these parameters by hand. But it is liable to say, that these configuration are not easy solution for novice users. They often make mistakes during the configuring method. A mechanism of automatic configuration and a method to get a temporary IP address are desired to solve these problems. Meanwhile, it is apprehended that IP addresses will be exhausted in the near future. Of course, a fundamental solution for this problem is under development(IPv6). But efficient IPv4 address recycling structures is required.

To achieve these requirements, DHCP is proposed by Dynamic Host Configuration Working Group of an IETF (Internet Engineering Task Force) [1, 2] [1]. DHCP is an extension of Bootstrap Protocol (BOOTP) [3], though DHCP differs from BOOTP, because DHCP can allocate IP address automatically and temporarily. Currently, there are many implementation of DHCP. Most of them are commercial products. To freely distribute DHCP, implementation was made from scratch. (This DHCP implementation is called as WIDE version). In addition, this DHCP implementation has been tested for operation under our practical network environment, with addendum of new protocol supporting host mobility:Virtual Internet Protocol[7, 8]. From these experiments, some problems of the DHCP were found.

This paper consists of four parts. First, the explanation of specification and protocol flow of DHCP is introduced. Secondly, the WIDE version DHCP implementation design is discussed. WIDE version DHCP is architecture independent. It is running on BSD UNIX and SunOS architecture. Porting of DHCP to other UNIX platforms requires no difficulty by running DHCP as a daemon process. Thirdly, some details of problems about DHCP specification and solutions are indicated. Some problems could be solved by sev-

[1] There are several documents concerned with DHCP, such as [4, 5, 6]

eral changes to DHCP. And others are fundamental problems about the design of DHCP. Finally, evaluations of WIDE version DHCP and future studies are described.

## 2 DHCP Overview

In this section, summary of DHCP related host configuration protocol is introduced, including the summary of DHCP itself.

### 2.1 Related Protocol

By using DHCP, dynamically configuring the host to the network is done by a simple handshake. In history, there are also many dynamic automatic configuration protocols. RARP[9] and BOOTP can also provide the mechanism of automatic configuration. These protocols use simple interaction; the client requests and the server replies. RARP (Reverse Address Resolution Protocol) is executed on Ethernet, and converts the Ethernet address to an IP address. RARP handshake is mainly used in the diskless workstations. RARP uses an Ethernet frame directly, meanwhile BOOTP uses UDP. BOOTP returns IP addresses with the subnet mask of a network, IP addresses of routers, etc. RARP and BOOTP have two defects. First, these protocol only support static allocation (conversion) of an IP address. RARP and BOOTP protocol do not solve the requirement of dynamic allocation. Secondly, these protocol can provide only few parameters.

Another protocol called NIP (Network Information Protocol) has the mechanism of dynamic allocation[10]. It is based on the "Polling/Defense" mechanism. When the NIP client requests an IP address, NIP server returns a set of IP address which is available in the network. NIP client selects one IP address from the network, and checks its availability with ARP[2]. If ARP replies, NIP client will retry the selection phase. If there is no ARP reply to repeated ARP requests, NIP client reserves the IP address. NIP may cause IP duplication if the host does not reply to ARP request. In addition, there is another problem that a client may be allocated to a different address than before after reallocation.

### 2.2 Design of DHCP

DHCP is extension of BOOTP. DHCP uses client-server model and is constructed on UDP. DHCP has server and client model with relay agent same as BOOTP. It relays messages between the client and the server[3].

[2] Address Resolution Protocol converts an IP address to an Ethernet address.

[3] This is different from function of IP router.

In DHCP all interactions are initiated by a client, and a server replies. This paper refers to this DHCP interaction model as **Basic Model**. DHCP has following implementation goals;

- DHCP should support dynamic allocation. DHCP is capable of leasing the address, DHCP server should be able to recycle the IP address when the lease period is expired.
- A network administrator should not configure each client, and user interaction of each client should not be required.
- DHCP should also support static allocation and an infinite lease.
- DHCP should coexist with BOOTP and normal host.
- DHCP should not require a server on each subnet. Relay agent which is compatible with BOOTP can be used.

Only allocation model supported by BOOTP is static allocation. On the other hand, DHCP supports three models;

1. **Dynamic allocation**: Server chooses and allocates an IP address with finite lease.
2. **Automatic allocation**: Server allocates an IP address with infinite lease.
3. **Static allocation**: Server allocates an IP address which has been chosen by the administrator.

DHCP server manages two database to recognize the network status: **Address Pool** is the database which holds IP addresses and other network configuration parameters. **Binding database** keeps mapping between an Ethernet address and entry of Address Pool.

### 2.3 DHCP Message Format

DHCP uses the same packet format with BOOTP for its compatibility(*Figure* 1). DHCP server can respond to BOOTP client and vice versa.

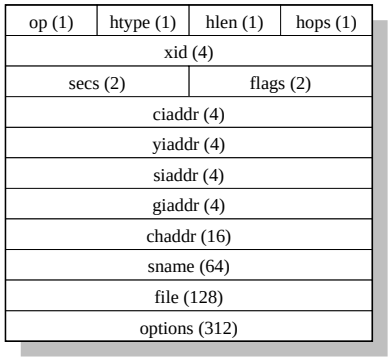


Figure 1: DHCP Packet Format

*Table* 1 is a part of the description of fields in a DHCP message. BOOTP uses only two message

| field | description |
|---|---|
| op | Message type. BootRequest/BootReply |
| xid | Transaction ID, a random number chosen by the client, used by the client and server to associate messages and responses between a client and a server. |
| ciaddr | Client IP address; only filled in if client can respond to ARP requests. |
| yiaddr | 'your' (client) IP address. |
| giaddr | Relay agent IP address, used in booting via a relay-agent. |
| sname | Optional server host name. |
| file | Boot file name. |
| options | Optional parameters field. |

Table 1: DHCP message fields' Description

types; **BootRequest** and **BootReply**[4]. These values are also used in DHCP for compatibility. However, DHCP puts another value in the *options* field to distinguish more various messages. There are seven message types for DHCP (*Table* 2).

| Message | Description |
|---|---|
| Discover | A client broadcasts to recognize servers. |
| Offer | Servers respond with proposal of parameters. |
| Request | The client broadcasts to the preferable server. An implicit decline to others. |
| Ack | The server assigns an IP address. |
| Nak | The server rejects the request from the client. |
| Decline | The client found a problem with an assigned address. |
| Release | The client returns the assigned address before its lease expires. |

Table 2: DHCP Message Type

## 2.4 Protocol Flow of DHCP

As stated above, DHCP is based on the client-server model. The process of an initial assignment can be divided into two phases; In the first step, client broadcasts a DhcpDiscover [5] to collect proposals from servers. In the second step, the client chooses one offer from the server, and requests the server to assign the address.

DHCP protocol flows are different with, getting a new address and gettting the previously assigned address. *Figure* 2 shows first case[6];

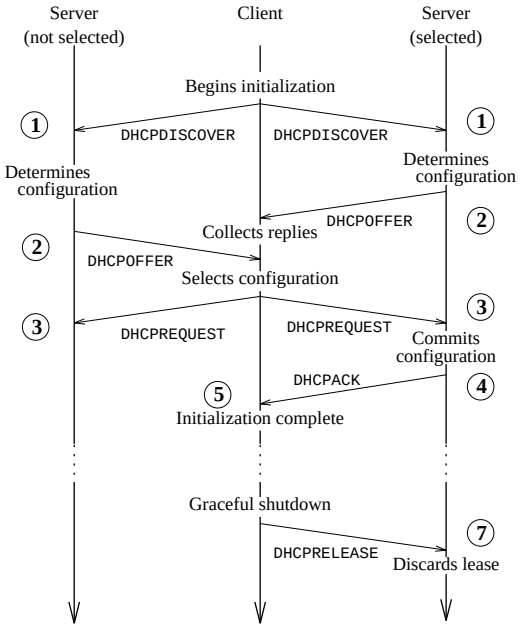


Figure 2: Initial Address Assignment

1. DHCP client broadcasts a DhcpDiscover. The client may specify preference of a lease and/or a IP address.
2. DHCP server receiving the DhcpDiscover may return DhcpOffer or may not return (Many servers may receives the DhcpDiscover). If a server decides to respond, server puts an available address into *yiaddr* field and broadcasts the DhcpOffer. At this point, there is no agreement of an assignment between the server and the client.
3. Client gets one or more DhcpOffers and choose one server from them. The client puts the IP address of the server into the *Server identifier* option of a DhcpRequest and broadcasts it to a server.
4. The server check the *Server identifier* option. If it does not match with its own address, the server consider it as an implicit decline. The selected server sends the DhcpAck (if its address is available) or the DhcpNak (for example, the address is already assigned to another client).
5. The client which gets the DhcpAck starts using the IP address. If it gets DhcpNak, it

[4] RFC951[3] refers to these types as BOOTREQUEST and BOOTREPLY.

[5] DHCPDISCOVER is an official name in [1, 2]. In this paper, we refer to DHCPDISCOVER as "DhcpDiscover" for easy to read. We also refer to other messages in the same manner.

[6] Numeral in the figure and in the descriptions are corresponded with each other.

restarts from step 1.

6. The client finds a problem with the assigned address of DhcpAck, it sends DhcpDecline to the server, and restarts from step 1.
7. The client can release the address before its lease expires by DhcpRelease. This process is not required absolutely.

The second case, the client requests previously assigned address, some steps are omitted;

1. A DHCP client broadcasts a DhcpRequest which *Requested IP address* option is the previously assigned address.
2. A DHCP server which has a binding of the address returns DhcpAck or DhcpNak.
3. Remaining are same as the first case.

### 2.5 Relay agent

The relay agent is for relaying packets between servers and clients. This makes server possible to handle the subnet which has no server available. Which means that there is no need to setup a server per subnet. The relay agent works as following;

1. A DHCP client broadcasts a message.
2. A relay agent puts its IP address of a network interface from which it receives the message into the *giaddr* field of the message. The relay agent unicasts to a server.
3. The server sends back its reply to the relay agent(unicast). The reply includes same *giaddr* as the request sent by the client.
4. The relay agent broadcasts the reply from the interface which IP address appears in *giaddr* field.

## 3 WIDE version DHCP implementation

The implementation of "WIDE Version" DHCP modules were coded from scratch. There were 5 goals to implement DHCP.

1. Should be running on major UNIX architecture.
2. Should be architecturely highly independent.
3. Has capability to be used in the actual network environment.
4. Adopts the conventional user interface.
5. Should be written from scratch to distribute it as free.

In the remaining section, details of the goals per each module: server client and relay agent is introduced with achieving the solutions per each problems. In addition, we applied DHCP to mobile support protocol named VIP[7, 8] and the last subsection describes about our experiences of this operation.

### 3.1 Server Implementation

BSD UNIX and SunOS were chozen to achieve the goal1. The Datalink will be limited to Ethernet Protocol. To achieve goal2 "*architecturely highly independent*", DHCP should be implemented in the User Area of an Operating System. DHCP requires a limited broadcast address of 255.255.255.255, but application using a normal socket cannot handle this address. If the DHCP server has multiple network interfaces, DHCP requires the identification of the network interface to correctly handle the received messages. This is because the DHCP server needs to assigns IP address dynamically from appropriate subnet in the multiple subnet connected in multiple network interfaces. This requirement is difficult to implement using the normal socket. Using BPF (Berkeley Packet Filter) for BSD UNIX and NIT (Network Interface Tap) for SunOS solved the problems. Sending messages to limited broadcast address and distinguish the network interfaces is possible by using BPF and NIT which are included in the interface driver. Also BPF and NIT has ability of filtering packets, which is very useful when using it in the actual network environment(Goal 3). It is important to save memory usage of the server to handle the large address pool and many clients. The server also uses the normal socket for unicast communication with relay agent and the already initialized client. *Figure* 3 is a layering structure of implementations of server including client and the relay agent.

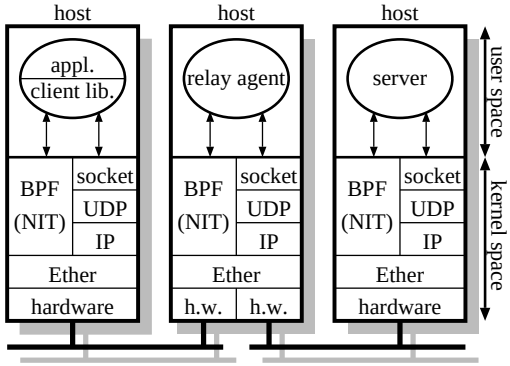


Figure 3: Layering Structure

For the goal 4, the binding database of the address pool is written in simple ASCII text file. The format of address pool file has similarity with *termcap (5)* for easy maintainability. Example of the address pool database is shown in *figure* 4.

```
global:snmk=255.255.254.0:tmof=32400:
subnet68:tblc=global:rout=133.27.68.1:\
        :brda=133.27.69.255:maxl=78000:
684:    :ipad=133.27.68.4:tblc=subnet68:
685:    :ipad=133.27.68.5:tblc=subnet68:
```

Figure 4: Example of Address Pool

### 3.2 Client and Relay agent

The client uses BPF and NIT to solve goal 1, 2, 3. Two client implementation were made. One is the **Library** version, Another version is the **Finite State Machine** version. Although DHCP offers various network parameters, clients only use few of them. In the library version, only basic functions are implemented; getting, reusing, extending the period, and releasing etc, the network address. DHCP client user needs to write the configuration using these library functions[7]. The FSM version is the full implementation of the client. The relay agent also uses BPF and NIT. FSM DHCP reads a database of server's IP address written in simple ASCII text file. For load balancing of servers, the relay agent selects server at random, and forwards messages.

### 3.3 Test Operation of WIDE version with VIP

When applying DHCP to mobile support protocol "VIP", a mobile host uses a VIP address as *Identifier* which will never change in operation, and a normal IP address as *Locater* which is gained by DHCP. In the current UNIX architecture, application layer process cannot get information from a datalink layer. So, to detect its own migration from a network to another network, VIP host has to verify repeatedly whether it can continue to use the previously allocated address or not. Repeated verifications may cause a problem, because there is no consideration about the situation of simultaneous verification with a lease extension. After section describes a solution to this problem.

## 4 Problems and Solutions

DHCP is now operating in our practical network environment used in WIDE project. In this section, Some problems and solutions of DHCP specifications which were found in our operations is introduced.

### 4.1 Verification with Lease Extension

Current DHCP does not consider clients to repeatedly verify the availability of allocated address.

The DHCP server is a packet driven module, causing easy implementations by using simple structure like;

```
while (1) {
    readmsg(buffer);
    processmsg(buffer);
}
```

On the other hand, client does the complex state transition (*Figure* 5). It is a nuisance to implement DHCP client in Operating System kernel or on a single task Operating System because of these complex state transition. This complexity becomes a problem with the implementation of a library version of a DHCP client. For example, if we defined a function to extend lease including retransmission to the DHCP library and implemented it thoughtlessly, there is the case when the function will block until lease extension completes.

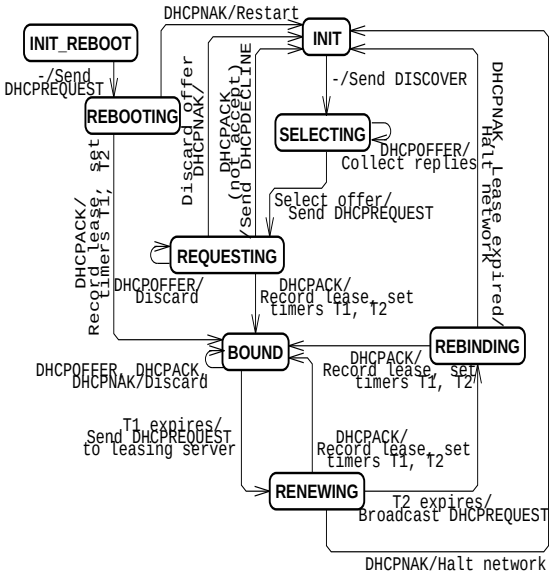


Figure 5: State Transition Diagram

In addtion, the DHCP specification considers that a client rarely moves to another network and the client is rebooted after movement. However mobile computers may move to another network without rebooting. As stated in the section 3.3, the mobile client may rapidly repeat verification of assigned address to detect changing subnet to which it is connected. If the lease extension occurs with verification simultaneously, the definition of retransmission of lease becomes obstacle.

The PCs' suspend resume feature may cause some problems to DHCP. Most of the PC turns on the clock back again after resume. The action taken by a DHCP client is strictly binded to the real time clock. Suppose the following case; The

[7] We also offer sample implementation using library.

client gets some lease, then it sleeps until half of lease is expired. Of course, other programs on the client host uses network continuously. Then, the client wakes up and begins lease extension. If the client host suspends and resumes, client host should have to check the current date. So the client has to pay attention to the clock at any moment to adapt suspend resume feature. There is an easy way to avoid this problem. Making the algorithm of retransmission free; The client can extend lease at any moment.

WIDE version DHCP can handle concurrent verification with lease extension, while following the defined algorithm of retransmission. WIDE version DHCP defines new "verify" state to support verification at any time (*Figure* 6)[8]. This

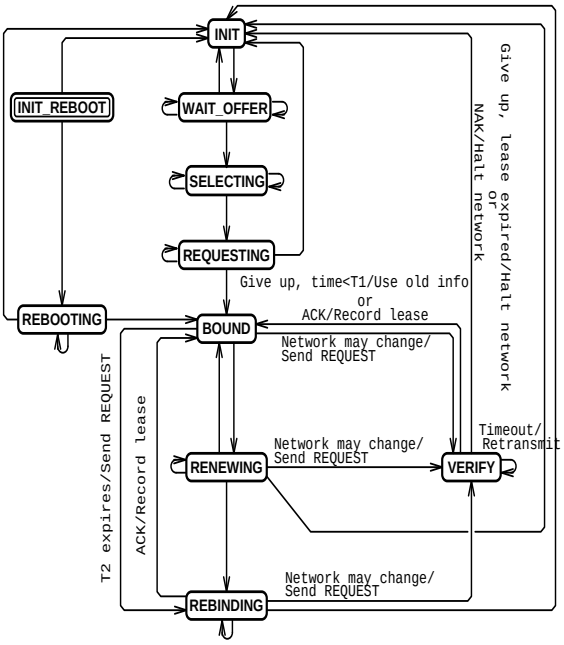


Figure 6: New State Transition Diagram

modification solves our specific problem with VIP, but the state transition is still complex.

It is clear that DHCP policy is causing these problems; **The client is active and stateful, the server is passive and message driven module.** Another solution is to change this policy, meaning to redesign a new protocol.

## 4.2 Availability of Fault-tolerant Lease Extension

There is no fault-tolerancy in DHCP. The **Basic Model** of DHCP is the server client model. If the server goes down, the client cannot extend its address lease and has to stop using the leased IP address when it expires.

The DHC Working Group proposes a backup server to solve this problem at 31st IETF[11]. The backup server has read-only copies of the address pool and the binding database. The master server is the owner of the entry. Master server can transfer the ownership to the backup server via a server-server protocol[9]. But this model cannot handle the release of an IP address. Another model was also proposed at 32nd IETF in Danvers[10]. This model defines a primary server and multiple secondary servers. The primary propagates its update of the database to secondary servers. A secondary server also can update the copy of the database, then notifies to the master. But this model only considers about lease extension. If address release or address recycle happens, this model can not handle it properly. Both models try to manage distributed databases. But essentially, current DHCP is unsuitable to the distributed database because of the following;

- The server strictly manages IP addresses.
- The server updates their databases very frequently.
- Lease expiration date does not increase monotonously.
- The client cannot work autonomously without server.
- IP addresses can be re-assigned to another client by the server.

## 4.3 Clarification of DhcpRequest

DHCP has backward compatibility with BOOTP. Message format has full compatibility with BOOTP. This feature has advantages; DHCP and BOOTP can interact with each other. Also BOOTP relay agent can forward the DHCP messages. But this feature also has disadvantages. For compatibility, DHCP cannot change packet format, only changing a usage of the *option*, *file* and *sname* fields. It would be hard to extend protocol with these restrictions, which is making DHCP more complicated. DHCP should have the field (or an option) which indicates the protocol version to easily extend protocol feature in the future release.

Another complication is the variety of purpose of DhcpRequest used in DHCP. DhcpRequest is used at following situation;

- **SELECTING**: Request new IP address.
- **INIT-REBOOT**: Request the previously assigned IP address.

[8] This figure omits many "event/action (input/output)" description about arrows.

[9] It's not defined yet.

[10] The proceedings is not available yet.

| | INIT-REBOOT | SELECTING |
|---|---|---|
| broad/unicast | broadcast | broadcast |
| server-ip | MUST NOT | MUST |
| requested-ip | MUST | MUST |
| ciaddr | zero | zero |

| .. | RENEWING | REBINDING |
|---|---|---|
| .. | unicast | broadcast |
| .. | MUST NOT | MUST NOT |
| .. | MUST NOT | MUST NOT |
| .. | IP address | IP address |

Table 3: Usage of DhcpRequest

- **RENEWING**: Extend the lease by unicast.
- **REBINDING**: Extend the lease by broadcast.

These situations are distinguished from each other by values; *Server IP address* option, *Requested IP address* option and *ciaddr* field (*Table* 3). These various DhcpRequest are very similar to each other and incur a confusion. It is desired to divide into different messages, or to add "Sub Code" to the DhcpRequest message.

It is easy to solve these two problems. Two new options, *DHCP version* and *DhcpRequest subtype* is added. The reason of introducing the subtype rather than dividing into new messages is for compatibility. On the other hand, the *DHCP version* option has no compatibility with the current DHCP. The format of these are shown in *Figure* 7. The code $x$ and $y$ should be assigned by IANA.

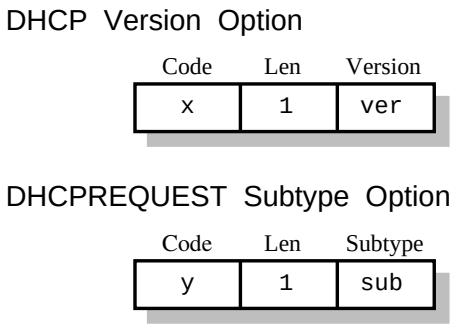


Figure 7: New DHCP Options

## 4.4 Acknowledgement of DhcpRelease

There is no acknowledgement messages corresponded with DhcpRelease. In the DHCP policy, DHCP client does not require to send DhcpRelease. It seems that a reply to DhcpRelease is unnecessary in implementation. But if a network administrator needs to reduce losses of DhcpRelease from clients as possible, acknowledgment of DhcpRelease is a good scheme. New message "DhcpReleaseAck" is implemented for handling the acknowledgement. When DHCP client no longer requires the use of its address, client sends DhcpRelase messages to the DHCP server. The server receives the DhcpRelease, marking the network address which was released as not allocated. DhcpReleaseAck is sent from server to client, indicating that the server received the DhcpRelase messages. DhcpReleaseAck follows the **Basic Model**; "Client sends a message and server replies".

The DHCP server replies whether the address of the released client is known or not via a DhcpReleaseAck messages. If the server returns the result of release in the DhcpReleaseAck, and the message was lost, it becomes troublesome to process following DhcpRelease retransmission by a client. Because, the server has already marked the address as released and can not release it again. Server will only reply it as a failure. Returning no result is easiest solution to avoid such a situation.

Client waits some period to receive the DhcpReleaseAck. If client cannot get the DhcpReleaseAck until its timeout, client should retransmit DhcpRelease. If client gets no answer after $N$ retransmission, client gives up confirmation of release and finishes its execution. The specification of retransmission algorithm and $N$ is not specified in this paper, but it should be configurable by a network administrator and be informed to the client in the *options* field.

## 4.5 Reduction of DhcpRequest Congestion

Current DHCP has no server-server protocol. IP address is managed by a single server. Multiple DHCP servers on a same subnet work appropriately, if all the database of the servers are independent to each other. Current DHCP has a scalability problem. Each server has capability to handle hundreds of clients during normal operation. But if many clients begin DHCP configuration at once, such as after power failure, may cause many DhcpRequests in short time period. Current DHCP has little consideration about such a situation[11], but it leaves place for revision. Basic idea for scalability is following;

[11] The client waits 1 - 10 seconds at **INIT-REBOOT** state before sending DhcpRequest, to avoid congestion. Client which gets no reply from the server can use previously assigned address until its lease expires.

1. Client listens advertisement (which is described after) from the server for about 1 - 10 seconds before sending a DhcpRequest in INIT-REBOOT state.
2. Client cannot get a preferable advertisement until timeout period, it will send its own DhcpRequest.

When the server receives DhcpRequest, it advertises active binding with a reply. *Figure* 8 shows *Binding Advertisement* option for this purpose. After power failure, first client sends a DhcpRequest and the server returns DhcpAck or DhcpNak with *Binding Advertisement*. Another client also listens the DhcpAck or DhcpNak responding to the first client and check whether there is active binding of its own or not. If there is, client uses it. If there isn't, client sends its own DhcpRequest after timeout. Server choose bind-

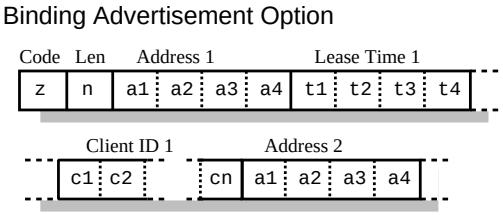


Figure 8: Binding Advertisement

ings which are not expired and are on the same subnet of the first client, with randomization to avoid that every *Binding Advertisement* becomes same one. Then, the server inserts them into the reply as many as possible.

*Binding Advertisement* reduces DhcpRequest storm after power failure. However there are some defects in this idea. One is that total size of the DhcpAck or DhcpNak should be smaller than default size defined in specification, because not all client cannot handle the large messages. Another defect is that clients (except the client sends its own DhcpRequest) cannot specify *Requested Lease* option, because client does not send its own DhcpRequest.

### 4.6 Active Notification from Server to Client

It would be natural to require the server to inform the changes of network parameters to the clients. The recent specification draft[12] deals with this requirements. If client receives a newly defined message called DhcpRevalidate, client has to move RENEWING state and send DhcpRequest to the server. Server will return DhcpAck including these new parameters.

There is a specification problems with DhcpRevalidate. First, DhcpRevalidate requires client to watch incoming messages increasing the complexity of state transition. Secondly, DhcpRevalidate may cause a DhcpRequest storm from clients, because it is sent as a broadcast. If the server requires DhcpRevalidate is received by all its clients, client must send DhcpRevalidate repeatedly. But this is excessive requirement to the server and does not follow **Basic Model**.

If server wants to retrieve the client's IP address, DhcpRevalidate can be used. Server returns DhcpNak instead of DhcpAck which informs changes of network parameters to make client to move into INIT state. But DhcpRevalidate is not suitable for this kind of situation. Because it requires three messages to retrieve; DhcpRevalidate, DhcpRequest and DhcpNak. Another new message policy, "DhcpRetrieve" is implemented to send messages from server to client. If client receives the DhcpRetrieve message, client has to release the assigned address via DhcpRelease. This DhcpRelease should have a flag which indicates that DhcpRelease is a reply to DhcpRetrieve. If the flag is set, the server must not send DhcpReleaseAck, and the client must not wait DhcpReleaseAck. DhcpRetrieve also violates the **Basic Model**. Therefore, this paper propose a new auto configuration model in the next section.

## 5 Evaluation and Future Works

This section evaluates WIDE version DHCP and discusses about future works to solve problems which requires a new automatic configuratin protocol.

### 5.1 Evaluation of Current Implementation

The WIDE version accomplishes all goals. All modules (Server, Client, Relay agent) are currently running on major UNIX architecture; BSD/OS, NEWS-OS[13], FreeBSD and SunOS. The goals of architecturely highly independency is achieved with these sample implementation. We adopt simple text file to configure the server and the relay agent. It is similar with the way of traditional UNIX configuration. So we regard that WIDE version achieve the goal of the conventional user interface.

We tested the WIDE version DHCP in the interoperability test with many vendors. The WIDE version indicated high interoperability. It can interact with almost any vendors DHCP implementations. In another words, if there is no

[12] ftp://sol.cs.bucknell/dhcwg/draft-spec-apr-95.txt

[13] NEWS is a Work Station of Sony

interoperability with the WIDE version DHCP, this means that the implementation is not fully under specification of DHCP.

BOOTP client interoperability was also tested. With countinous practical testing in the network showed high stability in implementation. Scalability was proved under operability test of hundreds' scale network connected environment.

WIDE version DHCP is suitable with mobile support protocol VIP. VIP requires temporary address, which indicates that DHCP is suitable for address distribution. DHCP with VIP can acess without disconnection from one network environment to another.

## 5.2 Future Studies

The following improvements could be accomplished with a modification of DHCP;

- Verification at any time. Especially with lease extension.
- Clarify about usage of DhcpRequest.
- Definition of a protocol version number for the future extension.
- Make DhcpRelease reliable using acknowledgement.
- More scalability in condition such as after power failure.

In addition, there are proposal to improve the following problems;

- Fault-torelancy of DHCP.
- Notification of changes on the network to a client.
- An address retrieval by the DHCP server.

But these proposals did not came to a successful completion. Because the current DHCP's **Basic Model** is not suitable for these requirements. Improvements of "*More scalability after power failure*"and "*Make DhcpRelease reliable*", does not match **Basic Model** of the DHCP. It is desired to make a new automatic configuration protocol to solve these problems radically.

The new automatic configuration mechanism should achieve following design goals;

1. Client can continue to use address if the server is down.
2. Protocol should withstand the scalability.
3. It should be easy to extend protocol to form a hierarchy.

The current DHCP is too severe about lease management. To accomplish goal 1, new protocol should use an optimistic and loose management. Optimistic management means that the lease has no explicit expiration date. Server periodically sends a message which includes address pool. This message has multi purpose; For the client which has no address, it is an advertisement of an available addresses. Client chooses one entry from the advertisement and sends requests for reservation. For client which already has address, it is a query from the server and client has to reply to use the address continuously. Client would be noticeable of the change of the connected network during these pollings because contents of the messages changes, For other servers, it would be an information exchange of the database contents. If the server goes down in case, another server can be the backup.

There is no explicit lease, and client can use the address semipermanently. Instead, client have to reply to a query from the server to use the address continuously. Client can continue to use the IP address, if the server is down or the connection path to the server is down. If the server cannot get a reply from client after $N$ advertisements, server marks address as not allocated. If network connection between server and client goes down, the server regards client to go away, although client may still uses the address. This inconsistency is to be fixed; Server makes the client to release address with a message.

We consider that optimistic and loose management fits distributed database management mechanism which is required in multi server environment for fault-torelancy. It should be easy to add new server within this management model, when number of client grows (Goal 2). It will be also easy to extend this management model to the goal 3.

# 6 Conclusion

The WIDE-version DHCP modules are designed to achieve full semantical compatibilities with the existing UNIX system architecture. The combination of the use of the Berkeley Packet Filter (BPF) and the Network Interface Tap (NIT) for the initial broadcast communication of the DHCP protocol, together with the use of the normal unicast transport mechanism for the rest of the communication contributes to the performance as well as to the architectural independency of the implementation. The compatible *termcap* format of the address pool database achieves the administration efficiency on the UNIX environment. As the result, the complete functions among clients, servers, and relay agents has been proved to be achieved. Adding to it, the functions of the server allocating resources to various implementation of the clients including the Windows NT 3.5 and many bootp-based

clients have been tested and proved to work. The implementation now is freely distributed with the full source code for evaluation purposes.

After the first distribution of the code, some protocol changes and experimental modifications based on the change have been discussed and designed. These includes the following points; in order for a client to ensure the status of the resource allocated by its server (at any moment; simultaneously with lease extension), the "verify" state has been added to the original state diagram. To achieve the clearer handling of requesting and extending the use of resources, the two new options, *DHCP version* Option and *DhcpRequset Subtype* Option are introduced. To avoid possible congestion caused from the DhcpRequest messages when the system restarted from a failure, The *Binding Advertisement* option is added to the DhcpRequest messages. *DhcpReleaseAck* message is intrduced to confirm address release.

Although these modifications solves problems we have encountered during the beta-testing with the existing implementation that is introduced in this paper, the further discussions and evaluations, especially with the group of the standard efforts will be extremely important.

## Acknowledgements

We appreciate the assistance of Mr. Keisuke Uehara and Mr. Kazunori Sugiura. We also give our thanks to the members of the WIDE Project, and especially to the members of a VIP working group for a cooperation of the test operation.

## References

[1] R. Droms, "Dynamic Host Configuration Protocol," RFC1541, October 1993.

[2] R. Droms, "Dynamic Host Configuration Protocol," Internet Draft, draft-ietf-dhc-dhcp-01.txt, March 1995.

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

[4] W. Wimer, "Clarifications and Extensions for Bootstrap Protocol," RFC1542, October 1993.

[5] S. Alexander, and R. Droms, "DHCP Options and BOOTP Vendor Extensions," RFC1533, October 1993.

[6] R. Droms, "Interoperation Between DHCP and BOOTP," RFC1534, October 1993.

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

[8] F. Teraoka, K. Uehara, H. Sunahara, and J. Murai, "VIP: A Protocol Providing Host Mobility," CACM, Vol.37, No.8 pp.67-75, August 1994.

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

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

[11] R. Droms, Charter of Dynamic Host Configuration working group, Proceedings of the Thirty-First Internet Engineering Task Force, pp.153-158, San Jose, California, December 5-9, 1994.

## Author Information

Akihiro Tominaga is a graduate student of Graduate School of Media and Governance, Keio University. He may be reached as `tomy@sfc.wide.ad.jp`.

Dr. Osamu Nakamura is with Keio University. He may be reached as `osamu@wide.ad.jp`.

Dr. Fumio Teraoka is with Sony Computer Science Laboratory Incorporation. He may be reached as `tera@csl.sony.co.jp`.

Dr. Jun Murai is with Keio University. He may be reached as `jun@wide.ad.jp`.