An Implementation of DHCPv6 by Dhawal Kumar

Introduction

The Dynamic Host Configuration Protocol (DHCPv6 [1]) enables DHCP servers to pass configuration information, via extensions [2], to IPv6 nodes. It offers the capability of automatic allocation of reusable network addresses and additional configuration flexibility. This protocol is a stateful counterpart to the IPv6 Stateless Address Autoconfiguration protocol [3], and can be used separately or together with the latter to obtain configuration information.

The Architecture

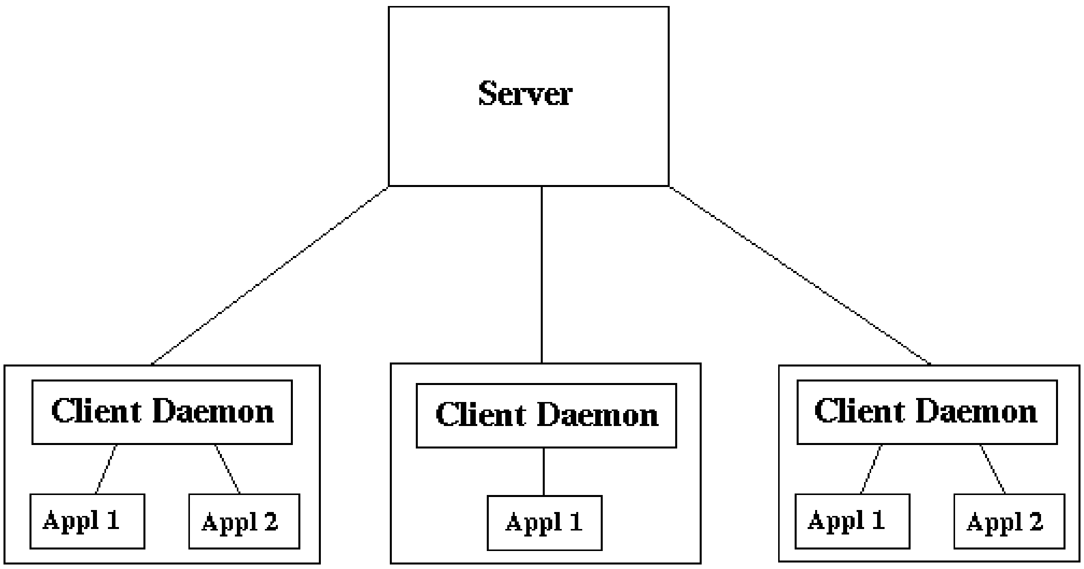

Fig. 1: The Architecture

The above figure shows the architecture for which DHCPv6 has been designed.

The server listens to requests from various clients (which are the client daemons running on different machines) on Agent port. The DHCPv6 server is a member of two multicast groups viz. FF02::1:2 and FF05::1:3 used in case the server address is not known.

The client daemon listens to requests from the various applications on CLD_TCP_PORT (for the definition of this port number see project/code/common/address_ports.h. You can, if you like, change this port number). Note that only one client daemon per machine is allowed.

All applications need to send an init to the client daemon before they can obtain any configuration information from the server. Applications generally use loopback address to talk to client daemon. Note that the requesting application must be running on the same machine as the client daemon.

Working of Client daemon

When the client daemon starts, it discovers all interfaces and opens a socket per address per interface (including the link local address) for listening to replies from server. The clientd daemon listens on a tcp socket for requests from applications. This socket is bound to CLD_TCP_PORT. When a connect() request arrives the client daemon does an accept() and puts the application and the new descriptor thus obtained into accept list. When the init from the application comes the application is removed from the accept list and put into init list (After the init the application can request any configuration information from the client daemon and the daemon would convey the message to the server and return the reply from the server back to the client). Once the application has entered the init list it remains there. The client daemon keeps looping to check if there is a request pending for any application and in such a case retransmits its message (The daemon also checks if there is reply from the server and in that case forwards it to the appropriate application). After retransmitting the message the daemon sleeps for min(next retransmission time of message from all applications whose requests are pending). In case there are no requests pending the client daemon sleeps for SLEEP_TIME * 100.

Working of Server

When the server starts it discovers all the interfaces and opens one socket for listening to messages from clients and one for listening to messages from administrator. The administrator can make the server initiate a reconfigure event through this socket. At present the server takes no actions if there is a message on such a socket. Since I haven’t implemented the reconfiguration part of the protocol, the server doesn’t need to retransmit any message. The responsibility of retransmission lies with the client daemon. The server simply loops to check if there is a request from a client and takes necessary actions. After this the server sleeps for min(lease expiry time of addresses leased). If there are no leases, the server sleeps for SLEEP_TIME * 100. SLEEP_TIME is defined in defines.h. Therefore, you can change SLEEP_TIME for server and client daemon independently.

Earlier Work

I had to begin work from scratch. My work is mostly based on [1] and [2].

Implementation Features

The client daemon can handle multiple clients. The daemon sleeps for the amount of time which is just sufficient and does not waste resources by doing a poll. In addition it makes several checks on the data send to it by the application.

Features not implemented/supported

- Relay hasn't been implemented. But there is support for offlink clients. Therefore, to support offlink clients, you just need to build a relay and the server and client daemon code need not be changed.
- The r bit and the associated transaction id cache.
- Handling of reconfigure messages.

References

[1] J. Bound, C. Perkins, Dynamic Host Configuration Protocol for IPv6 (DHCPv6), draft-ietf-dhc-dhcpv6-15.txt, Internet Draft, 5 May 2000
[2] C. Perkins, J. Bound, Extensions for the Dynamic Host Configuration Protocol for IPv6, draft-ietf-dhc-v6exts-12.txt, Internet Draft, 5 May 2000
[3] S. Thomson, T. Narten, IPv6 Stateless Address Autoconfiguration, Request For Comments 2462, Draft Standard, December 1998
[4] J. Bound, L. Toutain, H. Afifi, Dual Stack Transition Mechanism (DSTM), draft-ietf-ngtrans-dstm-00.txt, Internet Draft, October 1999