DHCP Server Technical Reference

Interniche Legacy Document

Version 1.00

Date: 15-May-2017 13:08

Table of Contents

1 Overview

This Technical reference is provided with the InterNiche Portable Dynamic Host Configuration Protocol (DHCP) server. The purpose of this Document is to provide enough information so that a moderately experienced "C" programmer with a reasonable understanding of TCP/IP protocols can port the InterNiche Server to a new environment.

1.1 Terms and Conventions

In this document, the term "stack", when used without other qualification, means the InterNiche TCP/IP and related code as ported to an embedded system. "System" refers to your embedded system. "Sockets" refers to the TCP API developed for UNIX at U.C. Berkeley. A "user" or "porting engineer" usually refers to the engineer who is porting the server software. An "end user" refers to the person who ultimately ends up using the "user's" product. "FCS" is an acronym for "First Customer Ship", the point in the software development cycle when the product is declared ready to ship. A "packet" is a sequence of bytes sent on network hardware, also known as a "frame" or a "datagram".

Names of files, C structures and C routines are displayed as follows: `c_routine()`.

Samples of source code from C programs are displayed in these boxes:

```
/* C source file - yet another 'hello; program */
main()
{
   printf("hello world.\n");
}
```

1.2 What DHCP Does

DHCP stands for Dynamic Host Configuration Protocol. It is designed to ease configuration management of large networks by allowing the network administrator to collect all the IP hosts "soft" configuration information into a single computer. This includes IP address, name, gateway, and default servers. There are about 50 of these information items which can be assigned with DHCP, and DHCP is designed so that "custom" configuration items can be added easily.

DHCP is a "client/server" protocol, meaning that machine with the DHCP database "serves" requests from DHCP clients. The clients typically initiate the transaction by requesting an IP address and perhaps other information from the server. The server looks up the client in its database, usually by the client's media address, and assigns the requested fields. Clients do not always need to be in the server's database. If an unknown client submits a request, the server may optionally assign the client a free IP address from a "pool" of free addresses kept for this purpose. The server may also assign the client default information of the local network, such as the default gateway, the DNS server, and routing information.

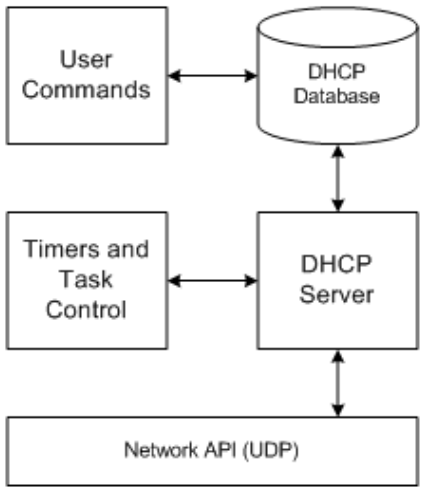
When the IP addresses is assigned, it is "leased" to the client for a finite amount of time. The DHCP client needs to keep track of this lease time, and obtain a lease extension from the server before the lease time runs out. Once the lease has elapsed, the client should not send any more IP packets (except DHCP requests) until he get another address. This approach allows computers (such as laptops or factory floor monitors) which will not be permanently attached to the network to share IP addresses and not hog them when they are not using the net.

1.3 BOOTP

In both this manual and other DHCP literature, you will find numerous cross references to BOOTP. BOOTP is the protocol which preceded DHCP in the TCP/IP world, and in fact DHCP is just s superset of BOOTP. The main differences between the two are the lease concept, which was created for DHCP, and the ability to assign addresses from a pool. The InterNiche DHCP client can work with old fashioned BOOTP servers, and the InterNiche DHCP server can serve IP addresses to old BOOTP clients.

1.4 Porting

While processing DHCP requests, the server will consult a local database. This database is generally set up by an end user and contains the IP address pools, default information for clients, and perhaps special configurations for specific clients. This database may be stored in a traditional disk file or in flash. On some embedded systems it may even be practical to ship the server with a factory configured database.



In the world of portable stacks, the stack designer does not know what tasking system, user applications, or interfaces will be supported in the target system. So a "portable" stack is one that is designed with simple, generic interfaces in these areas, and a "glue" layer is created which maps this generic interface into the specific interfaces available on the target system. Again using the example of sending a packet, the stack would be designed with a generic `send_packet()` call, and the porting engineer would code a "glue" routine to send the packet on the target system's network interface.

Making a stack portable involves minimizing the number of calls which have to go across glue routines, and keeping the glue routines simple and therefore easy to implement. The glue routines also need to be well documented. The interfaces to the InterNiche DHCP have evolved through years of porting to a variety of processors, network media, and tasking systems. Wherever possible we have used standard interfaces (e. g. Sockets, ANSI C library) or included glue routines to illustrate their use.

The bulk of the work in porting a stack is understanding and implementing these glue routines. The InterNiche DHCP server has two kinds of glue routines: database access and network access.

The calls to set and extract database items are abstracted, which means they are generic calls which will need to be provided as part of the port. Files are provided which use standard file IO calls (`fopen`, `fread`, etc) to implement a fully functional database, so if your target system has a disk-like device you will most likely be able to use this code as is. If your database will be kept in flash memory (and it does not have a file-system like API), you will need to develop you own data structures and the code to access them.

The other set of glue routines needed for a port is the network access. If you are using the DHCP server with the InterNiche IP stack, you are in luck: a file is provided which implements the required routines on the lightweight API. We also provide a Sockets based glue layer for DHCP server. Most of our customers can use one of these two layers. The rest will need to provide some simple routines to allow the server access to their UDP protocol.

1.5 Requirements

Before beginning a port, the programmer should ensure that the necessary resources are available in the target environment. Here is a brief summary of services InterNiche DHCP server needs from the system:

- Access to a UDP layer, with "listen" capabilities (e.g. Sockets)
- A timer which ticks at least once a second.
- A non-volatile read/write method for storing database items (e.g. disk or flash memory)
- Memory as described below.

Operating System Requirements

The DHCP server also requires a few basic services from the operating system. These are listed here:

clock tick	The DHCP server needs to be called once a second to free resources for addresses which have timed out.
memory access	The standard calloc() and free() library calls are ideal, however DHCP server can also be mapped to a "partition" based system with very little effort.

2 Step by Step Porting Guide

The section describes the steps needed to port the InterNiche DHCP server to a new environment. The discussions below generally assume that the stack is being ported to a small or embedded system with a Sockets API interface and that a minimal ANSI C library is available.

The recommended steps to getting the server working on your target system are as follows:

1. Copy the portable source files into your development environment.
2. Create your version of `dhcpport.h` and compile portable sources.
3. Code your glue layers (`dhcpport.c`) and compile.
4. Build a system, test, and debug.

2.1 Source Files List

Before beginning, you should be aware of which files in the InterNiche distribution are the "portable" files and which are not. The portable files are those which should be compiled and used on any target system without modification. The unportable, or "port dependent" files, are likely to need some modification for different target systems. The following is a list of DHCP server source files which should NOT need to be modified in the course of a normal port. If you feel you need to modify one of these files in the course of a routine port, please discuss it with InterNiche's technical support staff first, so we can either suggest an alternative, or modify our sources to reflect the change.

dhcpsvr.c, dhcpsvr.h, dhs_mod.c	The kernel of the DHCP server. Do not modify.

The following port-dependent files may require some modification by the porting engineer.

dhs_file.c	The database IO files: may need modification on some ports.
dhs_port.c	The network (Sockets) and other system dependent calls
dhs_port.h	Configuration defaults, definition of client structure, prototypes of port-dependent functions
dhs_nt.c	InterNiche menu system routines - can be used as is with InterNiche menu system, else may be replaced or omitted.

Standard Macros and Definitions

InterNiche modules expect the macros in the following list to be defined appropriately for your platform. Example definitions can be found in the `ipport.h` file that was provided with your delivery

- htonl
- htons
- ntohl
- ntohs
- TRUE
- FALSE
- NULL

Memory Allocation

The DHCP server code allocates and frees memory blocks dynamically as it runs. It uses the macros listed below to do this. If your target system supports standard C `calloc()` and `free()`, the macros map directly as follows:

```
/* map malloc and free to system calls */
#define DHCP_ALLOC(size)   calloc(1, size)   /* DHCP header alloc/free */
#define DHCP_FREE(ptr)     free(ptr)
#define DHPOOL_ALLOC(size) calloc(1, size)   /* DHCP free pool list alloc/free */
#define DHPOOL_FREE(ptr)   free(ptr)
#define DHENT_ALLOC(size)  calloc(1, size)   /* DHCP entry list alloc/free */
#define DHENT_FREE(ptr)    free(ptr)
```

Many RTOS systems do not use calloc due to performance issues. Generally, they use a system which supports allocations of fixed size "partitions" (blocks) instead. The macros above are designed to support this - each of the `_ALLOC` macros only allocates a single size. Thus the macros can be mapped to a call to allocate the next largest partition size.

Debugging Aids

`dtrap()` is a macro called by the DHCP code whenever it detects a situation which should not be occurring. The intention is for the `dtrap()` routine or macro to try to trap to whatever debugger may be in use by the programmer. Think of it as an embedded break point. For most Intel x86 processor debuggers, this can be done with an int 3 opcode. The macro below is one such an example:

```
#define dtrap();   _asm{ int 3 }
```

The stack code will generally continue executing after a `dtrap()`, but it usually indicates that something is wrong with the port. NO PRODUCT BASED ON THIS CODE SHOULD BE SHIPPED UNTIL THE CAUSES OF ALL CALLS TO `dtrap()` HAVE BEEN ELIMINATED! When it comes time to ship code, `dtrap()` can be redefined to a null function to slightly reduce code size.

The next few primitives have the same function and syntax as `printf()`. They have separate names so that they can have their output redirected or be completely disabled independently of each other. The first, `dprintf()`, is used throughout the stack code to print warning messages when something seems to be wrong. This should be mapped to a debugging console or log during development, and generally ifdefed away for FCS. The `gio_printf()` call is for printing statistical and debug information from the DHCP "menu" functions. These will certainly be useful during product development, and depending on the nature of the product may be needed in the end user's release.

Features and Options

The DHCP server can optionally read a standard UNIX-like bootptab file and use it to initialize entries in the DHCP address pool. This is provided for backward compatibility with BOOTP. This feature requires access to a conventional file system via the standard C calls `fopen()`, `fgets()` and fclose(). If you are designing a DHCP server for use with older UNIX workstations, you should try to include this feature. To do this, `dhcpport.h` should contain this definition:

```
#define BOOTPTAB 1 /* will try to read a BSD bootptab file */
```

2.2 dhcpport.c - The glue Layer

The reference port the files `dhs_port.c` and `dhs_file.c` contain the routines that map the generic service requests DHCP makes to specific services your target system provides. These may need to be implemented as minimal layer of C code.

UDP Hooks

Usually the most complex part of the glue layers is the network interface. DHCP needs to send and receive packets via the UDP protocol. If you are using InterNiche's lightweight API or a standard "Sockets" interface, sample `dhcpport.c` files are provided which do most of the work for you. Otherwise, you will need to implement the routines described in User Provided Functions. These are summarized below:

```
/* dhcp server's per-port utility for sending datagrams */
int dhs_udp_send(int iface, void * outbuf, int outlen);

/* portable dhcp server received packet handler */
int dhs_upcall(PACKET pkt, void *data, struct sockaddr_in *sin);
```

This callback in `dhs_port.c` calls the portable function `dhs_receeive()` in dhcpsvr.c.

Timers and Multitasking

A DHCP server only needs to get CPU time upon two events, each of which is handled by a callback routine. The events are an arriving DHCP packet, and the once-per-second timer. The arriving DHCP packets are processed in `dhcp_receive()`. The once-per-second timer is implemented by calling `dhcp_timeisup()` once a second - the code to initialize this timer may be part of dhcpport.c.

The other aspect of multitasking is to protect sensitive structures from being corrupted by code re-entry. This is accomplished by two macros which protect critical sections of code: `LOCK_NET_RESOURCE (DHS_RESID)` and `UNLOCK_NET_RESOURCE(DHS_RESID)`. These MACROs are defined in osport.h. Their implementation is highly port-specific. In the example port, they are mapped to the functions `tk_res_lock` and `tk_res_unlock` in misclib/task.c.

2.3 The DHCP Database

The DHCP Server requires a database containing the addresses and configuration parameters that will be given to clients when they make DHCP requests. The database consists of:

Address pools:	Ranges of addresses that can be assigned to clients
Interface defaults:	Configuration parameters to be assigned to client requests received on a specific interface.
Client Table:	A table of specific addresses and/or configuration parameters to be given to specific clients based on their client ID.

The DHCP data items supported by the client table and interface defaults are:

```
char name[DHS_MAX_HOSTNAME];   /* String for name */
ip_addr ipaddr;    /* client's assigned IP address */
ip_addr snmask;    /* client's assigned subnet mask */
ip_addr gwaddr;    /* client's assigned default gateway */
ip_addr dnsaddr;   /* Domain Name server */
char clientId[DHS_MAX_CLIENTID]; /* usually client's hardware address */
unshort  type;     /* type of this entry, see DHT_ defines in dhcpsvr.h */
unshort  status;   /*status of this entry, see DHT_ defines in dhcpsvr.h */
uint32_t lease;    /* dual use: lease period or when offer will timeout*/
```

Note: The *snmask* and *address* parameters are always stored in network byte order. Other parameters such as the `uint32_t` "lease" are stored in host byte order and must be converted by network/host macros when they are moved between the network and memory.

Most of these fields should be familiar to programmers with some exposure to TCP/IP networks, however the clientId field deserves some explanation. This field is the unique ID which the DHCP server will use to track each client as it asks for and receives an IP address and configuration. On old BOOTP systems this was always the Ethernet (or token ring) MAC address, since this was always unique. This tradition has generally been carried forward to DHCP, and if you are using DHCP over Ethernet or token ring, then using MAC address as client ID is by far the simplest way to go. The DHCP clients will determine the size of this field, and on these media it will always be six since both MAC addresses are six bytes in length. However since DHCP may be used over PPP and other address-less type links, there may be cases where the clientId field will not be six bytes in length. The define `DHS_MAX_CLIENTID` is used because the size my vary between interfaces on a single system. If you implement a server which will use an unusual clientId size, be sure to modify the default value to the size your media will be using.

Database Initialization

The database could be created/initialized by a number of methods: hardcoded at compile time, set at initialization via a script, or some other port-specific implementation such as being read and parsed from a configuration file.

The example port is based primarily on script files that are read at initialization time. Database initialization moves in three levels, from the more generic to the specific. At each level, the defaults are used unless they are modified by more specific commands. The top level consists of generic compile-time defaults for parameters. These are expected to be modified at run-time by per-interface default parameters and address pool configurations. Finally there is a command to set individual configurations that will be given to specific clients based on their client-id.

dhs_port.h

`dhs_port.h` contains a set of defines that limit the maximum sizes of fields and arrays, such as:

DHS_MAX_IFACES	Maximum number of DHCPSVR interfaces that can be configured
DHS_MAX_DOM_NAME	Maximum domain-name size that can be configured.

`dhs_port.h` contains the definition of `struct dhs_client`. This primarily consists of fields containing the parameters that will be given to clients with each DHCP Request. You may want to add fields to this structure or to save space by removing fields that will not be used by your DHCP Server implementation.

`dhs_port.h` also contains default values for client parameters. These will be given out with the DHCP Request, unless they are modified by the more specific run-time commands listed in the next sections.

Script commands

At the second level, script commands are used to set per-interface default parameters and pools of IP addresses that may be assigned. The default parameters that will be assigned for each interface can be set at run-time via a script or set of commands. At a minimum the system must be told to use the default parameters for the interface by the command, "`dhs_ifcfg -i X`", where 'X' represents an interface number. Any additional parameters provided with this command will over-ride any previous values for these parameters.

In the demo implementation, there are no defaults for address pools. At least one address pool must be configured at run-time via one or more "`dhs_addrpool`" commands.

Finally, when the client ID is known for clients that will be requesting DHCP configuration, specific configurations can be defined for each client ID. Clients not given a specific configuration will be given the parameters for the interface and an address from the range of addresses in the address pool[s] for the interface.

The following is an example of a script file:

```
# Assign the compile time default parameters to clients on this interface
dhs_ifcfg -i 1

# Assign a specific gateway and domain name to clients on this interface.
# (Use compile-time default parameters for the rest)
dhs_ifcfg -i 2 -g 192.168.0.1 -n mydomain

# Define a pool of address that can be assigned on any interface
dhs_addrpool -l 192.168.0.2 -h 192-168.5.254 -s 255.255.0.0

# Give the client with the specified client ID an infinite lease on the
# lowest address in the pool */
dhs_client -c 00:08:f3:2d:04: 40 -a 192.168.0.2 -l 0xFFFFFFFF
```

The script file may contain any number of commands. It is normally read at init time, but one or more individual commands may be issued at any time.

dhcpsvr_file

All run-time configuration information is stored in RAM. Each time a client configuration is added to the in-RAM client table, it is also written to the data-base file, "`dhcpsvr_file`". This is done because it is normally desirable for the DHCP server to try to give a client the same IP address and configuration each time it makes a DHCP request. The `dhcpsvr_file` is a binary file. Each client entry is made by simply by writing out the struct dhs_client for that client.

The dhcpsvr_file is read at initialization time before the script commands are called. A dhs_client command will overwrite any existing fields obtained for that client from the `dhcpsvr_file`, and the new values will be saved in the file.

2.4 Testing

Once your `dhcpport.h` file is set up and your glue layers are coded, compiled, and linked, you are ready to test your server. The steps for a basic test are simple: start your DHCP server, then reboot any DHCP client machine. The two machines should complete a 4 packet exchange as described in the DHCP RFCs. If you have replaced the `dhs_file.c` file IO code with your own, you should also test to ensure that both per-client data and host data are being set properly.

In any case you should now have a working DHCP server.

3 Troubleshooting

If your implementation of the DHCP server has problems, there are several techniques you can use to track down the problems. The problems generally fall into two categories: connecting the server to UDP and keeping the database information accurate.

3.1 UDP Transport

Although InterNiche provides code for most common UDP APIs, there are still a variety of things which can go wrong. Since the DHCP server always operates by responding to client requests, the first problem you are likely to encounter is the inability to receive packets. If you have tried rebooting a DHCP client and the server has not responded, you will want to make sure the DHCP server actually received the packet from the client. The easiest way is to use the dhs netstat command in the DHCP server menus - it provides counters for all types of packets received and sent. If these are all zero, this tells you to go back and debug your UDP "listen" and receive code.

If the menu counters indicate a discover packet was received and an offer packet was sent, but no request was received, it is possible your UDP send has problems - the server thinks it sent the packet to the UDP layer, but UDP never got it onto the network. Debugging your `dhs_udp_send()` code is then indicated.

The DHCP server, unlike many networking protocols, is quite amenable to source level debugging with breakpoints. Since each DHCP packet is sent from the server as a reply to a client packet, setting a breakpoint on `dhcp_receive()` will allow you to trace the entire DHCP transaction all the way through to the sending of the response.

In all cases, a Packet Analyzer is an invaluable tool for debugging this sort of problem. An analyzer will capture on packets on the LAN to which it is attached, and save them for later review. Most support filters, so you can set them to capture only the packets of interest - in this case BOOTP/DHCP packets.

3.2 Database Debugging

If the DHCP packets are being exchanged between client and server, but the IP configuration information is not what you expected, there are some simple techniques you can use to find the problem:

1. Double check your database files. Incorrectly entered MAC addresses are a common source of trouble in per-client setups. The clients will not be found in the database and will be assigned default values instead.
2. Make sure the files are being read into the DHCP server's internal structures correctly. The `dhs netstat` command can be used to display information even for clients which have not generated a request yet. If the IP configuration information is not correct here, it will not be correct on the net.
3. Use a packet analyzer to check the information in the reply packets coming out of the server. If the packets do not reflect the data revealed by `dhs netstat` then there is an encoding problem of some kind. The most common cause of this is "endian" issues.

4 The DHCP User Menu

4.1 dhs_addrpool

Command Name

`dhs_addrpool - configure an address pool`

Syntax

`dhs_addrpool [-i <iface>] {-l <low addr> -h <high addr> -s <subnetmask>}`

Parameters

	Command without address parameters displays the current address pools for all or the specified interface
`-i`	Command applies to the specified interface
`-l`	lowest IP address in pool
`-h`	highest IP address in pool
`-s`	Subnet mask in IP addr format (dotted notation)

Description

This command is used to configure an address pool

Notes/Status

- In order to configure an address pool, the `'-l'`, `'-h'` and `'-s'` options must all be used.
- If the `'-i'` option is used, then the addresses in the pool will be given only to clients on the specified interface. The specified interface must already have a default configuration
- If the `'-i'` option is not used, then at least one interface must already have a default configuration

Location

This command is provided by the `DHCP Server` module when `DHCP_SERVER` is defined.

4.2 dhs_client

Command Name

`dhs_client - Configure parameters to be given to a specific DHCP client`

Syntax

```
dhs_client -c <Client ID> [-a <IP addr>] [-i <iface>] [-d <DNS addr>] [-g
<gw addr>] [-h <host name>] [-l <lease>] [-s <subnetmask>]
```

Parameters

`-c`	string: usually the MAC address (include colons) of the client to be given these parameters
`-a`	IP address to be given to this client
`-i`	Interface number. Start with the default parameters for this interface. These may be modified by the other parameters
`-d`	IP address of DNS server
`-g`	IP address of gateway
`-h`	Host name to be given to this client
`-l`	Integer: lease in seconds
`-s`	Subnet mask in IP addr format (dotted notation)

Description

This command is used to configure the parameters be given to a specific DHCP client when it makes a DHCP Request. A client configuration is saved both in an in-RAM client table and in a client table saved in persistent storage

Notes/Status

- The `'-c'` parameter is required.

Location

This command is provided by the `DHCP Server` module when `DHCP_SERVER` is defined.

4.3 dhs_enable

Command Name

`dhs_enable - Enable/Disable DHCP Server`

Syntax

```
dhs_enable -d | -e
```

Parameters

`-d`	Disable the DHCP Server
`-e`	Enable the DHCP Server

Description

This command is used to enable/disable the DHCP Server.

Notes/Status

- The DHCP Server cannot be enabled until at least one interface has been give default parameters and at least one address pool has been configured. Be default, it will be enabled as soon as this condition is met

Location

This command is provided by the `DHCP Server` module when `DHCP_SERVER` is defined.

4.4 dhs_ifcfg

Command Name

`dhs_ifcfg - configure default parameters to be delivered with addresses on this interface`

Syntax

```
dhs_ifcfg {-i <iface> [-d <dns addr>] [-e <yes/no>] [-g <gwaddr>]
          [-l <lease name>] [-n <domain name>]
          [-o <mask>] [-s <subnetmask>]}
```

Parameters

	dhs_ifcfg without parameters displays the current default paramters for all interfaces
`-i`	Interface number
`-d`	IP address of DNS server
`-e`	string: enable "yes" or "no"
`-g`	IP address of gateway
`-l`	Default lease in seconds
`-n`	Domain name
`-o`	hexidecimal integer. Mask of acceptable options
`-s`	Subnet mask in IP addr format (dotted notation)

Description

This command is used to configure the default parameters that will be included with each DHCP address given out on this interface.

Notes/Status

- The file "dhs_port.h" contains system defaults for all parameters except `'-i'`.
- At least one interface must be configured before the DHCP server will be enabled. The minimum command is dhs_ifcfg -i <iface>
- At lease one address pool must also be configured before the DHCP Server will be enabled.

Location

This command is provided by the `DHCP Server` module when `DHCP_SERVER` is defined.

4.5 dhs_netstat

Command Name

`dhs_netstat - Display DHCP Server statistics and parameters`

Syntax

`dhs_netstat [-d] [-e <index>] [-l] [-p]`

Parameters

	Command without parameters displays statistics for the DHCP Server
`-d`	List default parameters for all interfaces
`-e`	Integer index of client entry. List parameters to be given to the specifed client
`-l`	List addresses that have been assigned
`-p`	Display free address pools

Description

This command is used to display DHCP Server statistics or to display the specified parameter list[s].

Notes/Status

- The index needed with the `'-e'` option can be obtained from the list provided by a previous dhs netstat command that used `'-l'` option.

Location

This command is provided by the `DHCP Server` module when `DHCP_SERVER` is defined.

5 User Provided Functions

The functions described in this section must be provided by the porting programmer as part of the porting the InterNiche DHCP server. The InterNiche provided reference port can be referenced for examples. If you are using the InterNiche IP Stack, many of these functions are already provided in it.

In the default port these functions are either mapped directly to system calls via macros in `dhcpport.h` or implemented in `dhcpport.c`.

5.1 UDP Network API Layer

These three functions are those which allow DHCP to send/receive UDP datagrams. Implementations are provided for standard Sockets and InterNiche's lightweight UDP API. The first few are calls the DHCP server code makes to the API layer code, whereas `dhcp_receive()` and `dhcp_timeisup()` are DHCP server internal functions which needs to be called from the UDP glue layer whenever a DHCP server packet is received.

Name

`dhs_udp_send()`

Syntax

```
int dh_udp_send(int iface, void * outbuf, int outlen);
```

Parameters

iface	the index for the interface the packet is to be sent on.
outbuf	the data buffer containing the UDP header
outlen	length of the outbuf, usually BOOTP or DHCP header structure size.

Description

Broadcast a UDP datagram on the interface indicated. Buffer with UDP data to send and a length are passed.

Returns

Returns 0 if OK, else non-zero error.

5.2 UDP Callback Function

Name

`dhs_upcall()`

Syntax

```
dhs_upcall(PACKET pkt, void *data, struct sockaddr_in *sin)
```

Parameters

pkt	The PACKET strucvture
data	pointer to the start of the BOOTP/DHCP header (UDP data)
sin	Pointer to the socket address structure.

Description

This is called from the per-port protocol stack hooks whenever the UDP listen to the DHCP/BOOTP server port has received a DHCP packet. Length and interface have already been checked.

Returns

This routine returns 0 if OK, -1 if packet has an error.

5.3 Timer Callback Function

Name

`dhs_timeisup()`

Syntax

```
void dhs_timeisup(void);
```

Parameters

None

Description

The DHCP clock tick. This should be called once a second by the host system. It allows the DHCP server to track lease time-outs and recycle unclaimed IP addresses.

Returns

Nothing.