Dynamo Reference Manual

Generated by Doxygen 1.5.1

Tue Nov 27 13:54:32 2007

Contents

Chapter 1

Dynamo Class Index

1.1 Dynamo Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Dynamo File Index

2.1 Dynamo File List

Here is a list of all files with brief descriptions:

Chapter 3

Dynamo Class Documentation

3.1 arpMsg Struct Reference

`#include <arpping.h>`

Public Attributes

- **ethhdr ethhdr**
- u_short **htype**
- u_short **ptype**
- u_char **hlen**
- u_char **plen**
- u_short **operation**
- u_char **sHaddr** [6]
- u_char **sInaddr** [4]
- u_char **tHaddr** [6]
- u_char **tInaddr** [4]
- u_char **pad** [18]

3.1.1 Detailed Description

Definition at line 13 of file arpping.h.

3.1.2 Member Data Documentation

3.1.2.1 struct ethhdr arpMsg::ethhdr

Definition at line 14 of file arpping.h.

3.1.2.2 u_short arpMsg::htype

Definition at line 15 of file arpping.h.

3.1.2.3 u_short arpMsg::ptype

Definition at line 16 of file arpping.h.

3.1.2.4 u_char arpMsg::hlen

Definition at line 17 of file arpping.h.

3.1.2.5 u_char arpMsg::plen

Definition at line 18 of file arpping.h.

3.1.2.6 u_short arpMsg::operation

Definition at line 19 of file arpping.h.

3.1.2.7 u_char arpMsg::sHaddr[6]

Definition at line 20 of file arpping.h.

3.1.2.8 u_char arpMsg::sInaddr[4]

Definition at line 21 of file arpping.h.

3.1.2.9 u_char arpMsg::tHaddr[6]

Definition at line 22 of file arpping.h.

3.1.2.10 u_char arpMsg::tInaddr[4]

Definition at line 23 of file arpping.h.

3.1.2.11 u_char arpMsg::pad[18]

Definition at line 24 of file arpping.h.

The documentation for this struct was generated from the following file:

- **arpping.h**

3.2 authorized_entry Struct Reference

`#include <ha_config.h>`

Public Attributes

- **node node**
- int **spi_low**
- int **spi_high**
- in_addr **network**
- in_addr **netmask**

3.2.1 Detailed Description

Definition at line 110 of file ha_config.h.

3.2.2 Member Data Documentation

3.2.2.1 struct node authorized_entry::node

Definition at line 111 of file ha_config.h.

3.2.2.2 int authorized_entry::spi_low

Definition at line 112 of file ha_config.h.

3.2.2.3 int authorized_entry::spi_high

Definition at line 113 of file ha_config.h.

3.2.2.4 struct in_addr authorized_entry::network

Definition at line 114 of file ha_config.h.

3.2.2.5 struct in_addr authorized_entry::netmask

Definition at line 115 of file ha_config.h.

The documentation for this struct was generated from the following file:

- **ha_config.h**

3.3 client_config_t Struct Reference

`#include <ha.h>`

Public Attributes

- char **foreground**
- char **quit_after_lease**
- char **abort_if_no_lease**
- char **background_if_no_lease**
- char * **interface**
- char * **pidfile**
- char * **script**
- unsigned char * **clientid**
- unsigned char * **hostname**
- int **ifindex**
- unsigned char **arp** [6]

3.3.1 Detailed Description

Definition at line 158 of file ha.h.

3.3.2 Member Data Documentation

3.3.2.1 char client_config_t::foreground

Definition at line 159 of file ha.h.

Referenced by udhcp().

3.3.2.2 char client_config_t::quit_after_lease

Definition at line 160 of file ha.h.

Referenced by udhcp().

3.3.2.3 char client_config_t::abort_if_no_lease

Definition at line 161 of file ha.h.

Referenced by udhcp().

3.3.2.4 char client_config_t::background_if_no_lease

Definition at line 162 of file ha.h.

Referenced by udhcp().

3.3.2.5 char* client_config_t::interface

Definition at line 163 of file ha.h.

Referenced by udhcp().

3.3.2.6 char* client_config_t::pidfile

Definition at line 164 of file ha.h.

Referenced by udhcp().

3.3.2.7 char* client_config_t::script

Definition at line 165 of file ha.h.

Referenced by run_script().

3.3.2.8 unsigned char* client_config_t::clientid

Definition at line 166 of file ha.h.

Referenced by udhcp().

3.3.2.9 unsigned char* client_config_t::hostname

Definition at line 167 of file ha.h.

3.3.2.10 int client_config_t::ifindex

Definition at line 168 of file ha.h.

Referenced by send_discover(), send_renew(), send_selecting(), and udhcp().

3.3.2.11 unsigned char client_config_t::arp[6]

Definition at line 169 of file ha.h.

Referenced by udhcp().

The documentation for this struct was generated from the following file:

- **ha.h**

3.4 config_keyword Struct Reference

`#include <files.h>`

Public Attributes

- char **keyword** [14]
- int(∗ **handler**)(char ∗line, void ∗**var**)
- void ∗ **var**
- char **def** [30]

3.4.1 Detailed Description

Definition at line 5 of file files.h.

3.4.2 Member Data Documentation

3.4.2.1 char config_keyword::keyword[14]

Definition at line 6 of file files.h.

Referenced by read_config().

3.4.2.2 int(∗ config_keyword::handler)(char ∗line, void ∗var)

Referenced by read_config().

3.4.2.3 void∗ config_keyword::var

Definition at line 8 of file files.h.

Referenced by read_config().

3.4.2.4 char config_keyword::def[30]

Definition at line 9 of file files.h.

Referenced by read_config().

The documentation for this struct was generated from the following file:

- **files.h**

3.5 dhcp_mobile Struct Reference

Public Attributes

- unsigned char * **nai**
- int **state**
- unsigned long **requested_ip**
- unsigned long **server_addr**
- unsigned long **timeout**
- int **packet_num**
- int **fd**
- int **signal_pipe** [2]
- int **nai_length**
- int **lease**
- int **spi**

3.5.1 Detailed Description

Definition at line 115 of file ha.c.

3.5.2 Member Data Documentation

3.5.2.1 unsigned char* dhcp_mobile::nai

Definition at line 116 of file ha.c.

Referenced by add_dhcp_mobile(), and udhcp().

3.5.2.2 int dhcp_mobile::state

Definition at line 117 of file ha.c.

Referenced by add_dhcp_mobile(), del_dhcp_mobile(), and udhcp().

3.5.2.3 unsigned long dhcp_mobile::requested_ip

Definition at line 118 of file ha.c.

Referenced by add_dhcp_mobile(), del_dhcp_mobile(), and udhcp().

3.5.2.4 unsigned long dhcp_mobile::server_addr

Definition at line 119 of file ha.c.

Referenced by del_dhcp_mobile().

3.5.2.5 unsigned long dhcp_mobile::timeout

Definition at line 120 of file ha.c.

Referenced by add_dhcp_mobile(), del_dhcp_mobile(), and udhcp().

3.5.2.6 int dhcp_mobile::packet_num

Definition at line 121 of file ha.c.

Referenced by del_dhcp_mobile(), and udhcp().

3.5.2.7 int dhcp_mobile::fd

Definition at line 122 of file ha.c.

Referenced by del_dhcp_mobile(), and udhcp().

3.5.2.8 int dhcp_mobile::signal_pipe[2]

Definition at line 123 of file ha.c.

Referenced by del_dhcp_mobile(), and udhcp().

3.5.2.9 int dhcp_mobile::nai_length

Definition at line 124 of file ha.c.

Referenced by add_dhcp_mobile(), del_dhcp_mobile(), and udhcp().

3.5.2.10 int dhcp_mobile::lease

Definition at line 125 of file ha.c.

Referenced by del_dhcp_mobile().

3.5.2.11 int dhcp_mobile::spi

Definition at line 126 of file ha.c.

Referenced by del_dhcp_mobile().

The documentation for this struct was generated from the following file:

- **ha.c**

3.6 dhcp_option Struct Reference

`#include <options.h>`

Public Attributes

- char **name** [10]
- char **flags**
- unsigned char **code**

3.6.1 Detailed Description

Definition at line 24 of file options.h.

3.6.2 Member Data Documentation

3.6.2.1 char dhcp_option::name[10]

Definition at line 25 of file options.h.

Referenced by attach_option().

3.6.2.2 char dhcp_option::flags

Definition at line 26 of file options.h.

Referenced by add_simple_option(), and attach_option().

3.6.2.3 unsigned char dhcp_option::code

Definition at line 27 of file options.h.

Referenced by add_simple_option(), and attach_option().

The documentation for this struct was generated from the following file:

- **options.h**

3.7 dhcpMessage Struct Reference

`#include <packet.h>`

Public Attributes

- u_int8_t **op**
- u_int8_t **htype**
- u_int8_t **hlen**
- u_int8_t **hops**
- u_int32_t **xid**
- u_int16_t **secs**
- u_int16_t **flags**
- u_int32_t **ciaddr**
- u_int32_t **yiaddr**
- u_int32_t **siaddr**
- u_int32_t **giaddr**
- u_int8_t **chaddr** [16]
- u_int8_t **sname** [64]
- u_int8_t **file** [128]
- u_int32_t **cookie**
- u_int8_t **options** [308]

3.7.1 Detailed Description

Definition at line 7 of file packet.h.

3.7.2 Member Data Documentation

3.7.2.1 u_int8_t dhcpMessage::op

Definition at line 8 of file packet.h.

Referenced by get_packet(), and init_header().

3.7.2.2 u_int8_t dhcpMessage::htype

Definition at line 9 of file packet.h.

Referenced by init_header().

3.7.2.3 u_int8_t dhcpMessage::hlen

Definition at line 10 of file packet.h.

Referenced by init_header().

3.7.2.4 u_int8_t dhcpMessage::hops

Definition at line 11 of file packet.h.

3.7.2.5 u_int32_t dhcpMessage::xid

Definition at line 12 of file packet.h.

Referenced by send_discover(), send_release(), send_renew(), send_selecting(), and udhcp().

3.7.2.6 u_int16_t dhcpMessage::secs

Definition at line 13 of file packet.h.

3.7.2.7 u_int16_t dhcpMessage::flags

Definition at line 14 of file packet.h.

Referenced by get_packet().

3.7.2.8 u_int32_t dhcpMessage::ciaddr

Definition at line 15 of file packet.h.

Referenced by send_release(), and send_renew().

3.7.2.9 u_int32_t dhcpMessage::yiaddr

Definition at line 16 of file packet.h.

Referenced by sendACK(), sendOffer(), and udhcp().

3.7.2.10 u_int32_t dhcpMessage::siaddr

Definition at line 17 of file packet.h.

3.7.2.11 u_int32_t dhcpMessage::giaddr

Definition at line 18 of file packet.h.

3.7.2.12 u_int8_t dhcpMessage::chaddr[16]

Definition at line 19 of file packet.h.

Referenced by sendACK(), and sendOffer().

3.7.2.13 u_int8_t dhcpMessage::sname[64]

Definition at line 20 of file packet.h.

Referenced by get_option().

3.7.2.14 u_int8_t dhcpMessage::file[128]

Definition at line 21 of file packet.h.

Referenced by get_option().

3.7.2.15 u_int32_t dhcpMessage::cookie

Definition at line 22 of file packet.h.

Referenced by get_packet(), get_raw_packet(), and init_header().

3.7.2.16 u_int8_t dhcpMessage::options[308]

Definition at line 23 of file packet.h.

Referenced by get_option(), init_header(), send_discover(), send_inform(), send_release(), send_selecting(), sendACK(), and sendOffer().

The documentation for this struct was generated from the following file:

- **packet.h**

3.8 dhcpOfferedAddr Struct Reference

`#include <leases.h>`

Public Attributes

- u_int8_t **chaddr** [16]
- u_int32_t **yiaddr**
- u_int32_t **expires**

3.8.1 Detailed Description

Definition at line 6 of file leases.h.

3.8.2 Member Data Documentation

3.8.2.1 u_int8_t dhcpOfferedAddr::chaddr[16]

Definition at line 7 of file leases.h.

Referenced by add_lease().

3.8.2.2 u_int32_t dhcpOfferedAddr::yiaddr

Definition at line 8 of file leases.h.

Referenced by add_lease(), and sendOffer().

3.8.2.3 u_int32_t dhcpOfferedAddr::expires

Definition at line 9 of file leases.h.

Referenced by add_lease(), lease_expired(), oldest_expired_lease(), and sendOffer().

The documentation for this struct was generated from the following file:

- **leases.h**

3.9 fa_spi_entry Struct Reference

`#include <ha_config.h>`

Public Attributes

- **node node**
- int **spi**
- in_addr **addr**
- int **alg**
- unsigned char **shared_secret** [MAXSHAREDSECRETLEN]
- int **shared_secret_len**

3.9.1 Detailed Description

Definition at line 118 of file ha_config.h.

3.9.2 Member Data Documentation

3.9.2.1 struct node fa_spi_entry::node

Definition at line 119 of file ha_config.h.

3.9.2.2 int fa_spi_entry::spi

Definition at line 120 of file ha_config.h.

3.9.2.3 struct in_addr fa_spi_entry::addr

Definition at line 121 of file ha_config.h.

3.9.2.4 int fa_spi_entry::alg

Definition at line 122 of file ha_config.h.

3.9.2.5 unsigned char fa_spi_entry::shared_secret[MAXSHAREDSECRETLEN]

Definition at line 123 of file ha_config.h.

3.9.2.6 int fa_spi_entry::shared_secret_len

Definition at line 124 of file ha_config.h.

The documentation for this struct was generated from the following file:

- **ha_config.h**

3.10 ha_config Struct Reference

`#include <ha_config.h>`

Public Attributes

- int **max_bindings**
- int **ha_default_tunnel_lifetime**
- int **reg_error_reply_interval**
- list **spi_list**
- list **authorized_list**
- list **fa_spi_list**
- list **interfaces**
- int **syslog_facility**
- char **ha_api_read_socket_path** [MAXFILENAMELEN+1]
- char **ha_api_read_socket_group** [MAXGROUPNAMELEN+1]
- char **ha_api_read_socket_owner** [MAXOWNERNAMELEN+1]
- int **ha_api_read_socket_permissions**
- char **ha_api_admin_socket_path** [MAXFILENAMELEN+1]
- char **ha_api_admin_socket_group** [MAXGROUPNAMELEN+1]
- char **ha_api_admin_socket_owner** [MAXOWNERNAMELEN+1]
- int **ha_api_admin_socket_permissions**
- int **udpport**
- int **socket_priority**
- int **enable_triangle_tunneling**
- int **enable_reverse_tunneling**
- int **pubkey_hash_method**
- char **dhcp_if** [MAXFILENAMELEN+1]
- char **ha_nai** [MAX_NAI_LEN+1]
- int **ha_nai_len**
- in_addr **sha_addr**
- __u32 **priv_ha**

3.10.1 Detailed Description

Definition at line 70 of file ha_config.h.

3.10.2 Member Data Documentation

3.10.2.1 int ha_config::max_bindings

Definition at line 71 of file ha_config.h.

Referenced by load_config().

3.10.2.2 int ha_config::ha_default_tunnel_lifetime

Definition at line 72 of file ha_config.h.

Referenced by load_config().

3.10.2.3 int ha_config::reg_error_reply_interval

Definition at line 73 of file ha_config.h.

Referenced by load_config().

3.10.2.4 struct list ha_config::spi_list

Definition at line 74 of file ha_config.h.

Referenced by cleanup_config(), and load_config().

3.10.2.5 struct list ha_config::authorized_list

Definition at line 75 of file ha_config.h.

Referenced by cleanup_config(), and load_config().

3.10.2.6 struct list ha_config::fa_spi_list

Definition at line 76 of file ha_config.h.

Referenced by cleanup_config(), and load_config().

3.10.2.7 struct list ha_config::interfaces

Definition at line 77 of file ha_config.h.

Referenced by cleanup_config(), and load_config().

3.10.2.8 int ha_config::syslog_facility

Definition at line 78 of file ha_config.h.

Referenced by load_config().

3.10.2.9 char ha_config::ha_api_read_socket_path[MAXFILENAMELEN+1]

Definition at line 79 of file ha_config.h.

3.10.2.10 char ha_config::ha_api_read_socket_group[MAXGROUPNAMELEN+1]

Definition at line 80 of file ha_config.h.

3.10.2.11 char ha_config::ha_api_read_socket_owner[MAXOWNERNAMELEN+1]

Definition at line 81 of file ha_config.h.

3.10.2.12 int ha_config::ha_api_read_socket_permissions

Definition at line 82 of file ha_config.h.

3.10.2.13 char ha_config::ha_api_admin_socket_path[MAXFILENAMELEN+1]

Definition at line 83 of file ha_config.h.

3.10.2.14 char ha_config::ha_api_admin_socket_group[MAXGROUPNAMELEN+1]

Definition at line 84 of file ha_config.h.

3.10.2.15 char ha_config::ha_api_admin_socket_owner[MAXOWNERNAMELEN+1]

Definition at line 85 of file ha_config.h.

3.10.2.16 int ha_config::ha_api_admin_socket_permissions

Definition at line 86 of file ha_config.h.

3.10.2.17 int ha_config::udpport

Definition at line 87 of file ha_config.h.

Referenced by load_config().

3.10.2.18 int ha_config::socket_priority

Definition at line 88 of file ha_config.h.

Referenced by load_config().

3.10.2.19 int ha_config::enable_triangle_tunneling

Definition at line 89 of file ha_config.h.

Referenced by load_config().

3.10.2.20 int ha_config::enable_reverse_tunneling

Definition at line 90 of file ha_config.h.

Referenced by load_config().

3.10.2.21 int ha_config::pubkey_hash_method

Definition at line 91 of file ha_config.h.

Referenced by load_config().

3.10.2.22 char ha_config::dhcp_if[MAXFILENAMELEN+1]

Definition at line 92 of file ha_config.h.

3.10.2.23 char ha_config::ha_nai[MAX_NAI_LEN+1]

Definition at line 93 of file ha_config.h.

3.10.2.24 int ha_config::ha_nai_len

Definition at line 94 of file ha_config.h.

3.10.2.25 struct in_addr ha_config::sha_addr

Definition at line 95 of file ha_config.h.

3.10.2.26 __u32 ha_config::priv_ha

Definition at line 96 of file ha_config.h.

The documentation for this struct was generated from the following file:

- **ha_config.h**

3.11 ha_tunnel_data Struct Reference

`#include <ha.h>`

Public Attributes

- in_addr **lower_saddr**
- __u32 **nonce**
- unsigned char **auth_type**
- int **reverse_tunnel**
- int **encapsulation**
- char **arp_if** [IFNAMSIZ]
- time_t **last_failure_time**

3.11.1 Detailed Description

Definition at line 139 of file ha.h.

3.11.2 Member Data Documentation

3.11.2.1 struct in_addr ha_tunnel_data::lower_saddr

Definition at line 140 of file ha.h.

3.11.2.2 __u32 ha_tunnel_data::nonce

Definition at line 141 of file ha.h.

3.11.2.3 unsigned char ha_tunnel_data::auth_type

Definition at line 142 of file ha.h.

3.11.2.4 int ha_tunnel_data::reverse_tunnel

Definition at line 146 of file ha.h.

3.11.2.5 int ha_tunnel_data::encapsulation

Definition at line 147 of file ha.h.

3.11.2.6 char ha_tunnel_data::arp_if[IFNAMSIZ]

Definition at line 148 of file ha.h.

3.11.2.7 time_t ha_tunnel_data::last_failure_time

Definition at line 149 of file ha.h.

The documentation for this struct was generated from the following file:

- **ha.h**

3.12 interface_entry Struct Reference

`#include <ha_config.h>`

Public Attributes

- **node node**
- char **dev** [IFNAMSIZ+1]
- int **ha_disc**
- int **agentadv**
- int **interval**
- in_addr **force_addr**
- int **if_index**
- in_addr **addr**
- in_addr **bcaddr**
- timeval **last_adv**
- int **icmp_sock**
- int **udp_sock**
- int **udp_bc_sock**
- int **udp_bc_sock2**

3.12.1 Detailed Description

Definition at line 50 of file ha_config.h.

3.12.2 Member Data Documentation

3.12.2.1 struct node interface_entry::node

Definition at line 51 of file ha_config.h.

3.12.2.2 char interface_entry::dev[IFNAMSIZ+1]

Definition at line 54 of file ha_config.h.

3.12.2.3 int interface_entry::ha_disc

Definition at line 55 of file ha_config.h.

3.12.2.4 int interface_entry::agentadv

Definition at line 56 of file ha_config.h.

3.12.2.5 int interface_entry::interval

Definition at line 57 of file ha_config.h.

3.12.2.6 struct in_addr interface_entry::force_addr

Definition at line 58 of file ha_config.h.

3.12.2.7 int interface_entry::if_index

Definition at line 62 of file ha_config.h.

3.12.2.8 struct in_addr interface_entry::addr

Definition at line 63 of file ha_config.h.

3.12.2.9 struct in_addr interface_entry::bcaddr

Definition at line 64 of file ha_config.h.

3.12.2.10 struct timeval interface_entry::last_adv

Definition at line 66 of file ha_config.h.

3.12.2.11 int interface_entry::icmp_sock

Definition at line 67 of file ha_config.h.
Referenced by cleanup_config().

3.12.2.12 int interface_entry::udp_sock

Definition at line 67 of file ha_config.h.
Referenced by cleanup_config().

3.12.2.13 int interface_entry::udp_bc_sock

Definition at line 67 of file ha_config.h.
Referenced by cleanup_config().

3.12.2.14 int interface_entry::udp_bc_sock2

Definition at line 67 of file ha_config.h.
Referenced by cleanup_config().
The documentation for this struct was generated from the following file:

- **ha_config.h**

3.13 lease_t Struct Reference

Public Attributes

- unsigned char **chaddr** [16]
- u_int32_t **yiaddr**
- u_int32_t **expires**

3.13.1 Detailed Description

Definition at line 25 of file dumpleases.c.

3.13.2 Member Data Documentation

3.13.2.1 unsigned char lease_t::chaddr[16]

Definition at line 26 of file dumpleases.c.

3.13.2.2 u_int32_t lease_t::yiaddr

Definition at line 27 of file dumpleases.c.

3.13.2.3 u_int32_t lease_t::expires

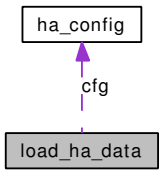
Definition at line 28 of file dumpleases.c.

The documentation for this struct was generated from the following file:

- **dumpleases.c**

3.14 load_ha_data Struct Reference

Collaboration diagram for load_ha_data:



Public Attributes

- **ha_config** * **cfg**
- int **process_spi_list**
- int **process_authorized_list**
- int **process_fa_spi_list**
- int **process_interfaces**

3.14.1 Detailed Description

Definition at line 28 of file ha_config.c.

3.14.2 Member Data Documentation

3.14.2.1 struct ha_config* load_ha_data::cfg

Definition at line 29 of file ha_config.c.

Referenced by load_config().

3.14.2.2 int load_ha_data::process_spi_list

Definition at line 30 of file ha_config.c.

Referenced by load_config().

3.14.2.3 int load_ha_data::process_authorized_list

Definition at line 31 of file ha_config.c.

Referenced by load_config().

3.14.2.4 int load_ha_data::process_fa_spi_list

Definition at line 32 of file ha_config.c.

Referenced by load_config().

3.14.2.5 int load_ha_data::process_interfaces

Definition at line 33 of file ha_config.c.

Referenced by load_config().

The documentation for this struct was generated from the following file:

- **ha_config.c**

3.15 spi_entry Struct Reference

`#include <ha_config.h>`

Public Attributes

- **node node**
- int **spi**
- int **auth_alg**
- int **replay_method**
- int **timestamp_tolerance**
- int **max_lifetime**
- unsigned char **shared_secret** [MAXSHAREDSECRETLEN]
- int **shared_secret_len**

3.15.1 Detailed Description

Definition at line 99 of file ha_config.h.

3.15.2 Member Data Documentation

3.15.2.1 struct node spi_entry::node

Definition at line 100 of file ha_config.h.

3.15.2.2 int spi_entry::spi

Definition at line 101 of file ha_config.h.

Referenced by cleanup_config().

3.15.2.3 int spi_entry::auth_alg

Definition at line 102 of file ha_config.h.

3.15.2.4 int spi_entry::replay_method

Definition at line 103 of file ha_config.h.

3.15.2.5 int spi_entry::timestamp_tolerance

Definition at line 104 of file ha_config.h.

3.15.2.6 int spi_entry::max_lifetime

Definition at line 105 of file ha_config.h.

3.15.2.7 unsigned char spi_entry::shared_secret[MAXSHAREDSECRETLEN]

Definition at line 106 of file ha_config.h.

3.15.2.8 int spi_entry::shared_secret_len

Definition at line 107 of file ha_config.h.

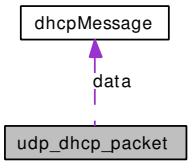
The documentation for this struct was generated from the following file:

- **ha_config.h**

3.16 udp_dhcp_packet Struct Reference

`#include <packet.h>`

Collaboration diagram for udp_dhcp_packet:



Public Attributes

- iphdr **ip**
- udphdr **udp**
- **dhcpMessage data**

3.16.1 Detailed Description

Definition at line 26 of file packet.h.

3.16.2 Member Data Documentation

3.16.2.1 struct iphdr udp_dhcp_packet::ip

Definition at line 27 of file packet.h.

Referenced by get_raw_packet().

3.16.2.2 struct udphdr udp_dhcp_packet::udp

Definition at line 28 of file packet.h.

Referenced by get_raw_packet().

3.16.2.3 struct dhcpMessage udp_dhcp_packet::data

Definition at line 29 of file packet.h.

Referenced by get_raw_packet().

The documentation for this struct was generated from the following file:

- **packet.h**

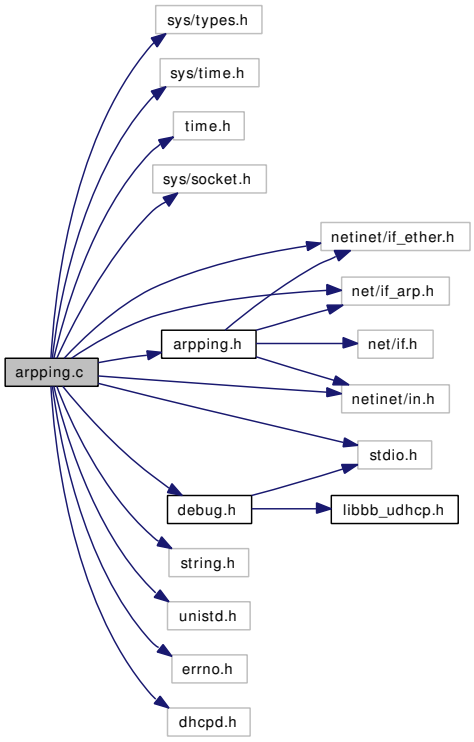
Chapter 4

Dynamo File Documentation

4.1 arpping.c File Reference

```
#include <sys/types.h>
#include <sys/time.h>
#include <time.h>
#include <sys/socket.h>
#include <netinet/if_ether.h>
#include <net/if_arp.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include "dhcpd.h"
#include "debug.h"
#include "arpping.h"
```

Include dependency graph for arpping.c:



Functions

- int **arpping** (u_int32_t yiaddr, u_int32_t ip, unsigned char *mac, char *interface)

4.1.1 Function Documentation

4.1.1.1 int arpping (u_int32_t *yiaddr*, u_int32_t *ip*, unsigned char * *mac*, char * *interface*)

Definition at line 35 of file arpping.c.

References DEBUG, LOG, LOG_ERR, LOG_INFO, and MAC_BCAST_ADDR.

Referenced by check_ip().

```
36 {
37
38         int     timeout = 2;
39         int     optval = 1;
40         int     s;                      /* socket */
41         int     rv = 1;                 /* return value */
42         struct sockaddr addr;           /* for interface name */
43         struct arpMsg   arp;
44         fd_set          fdset;
45         struct timeval  tm;
46         time_t          prevTime;
47
48
49         if ((s = socket (PF_PACKET, SOCK_PACKET, htons(ETH_P_ARP))) == -1) {
50                 LOG(LOG_ERR, "Could not open raw socket");
51                 return -1;
```

```
52          }
53
54          if (setsockopt(s, SOL_SOCKET, SO_BROADCAST, &optval, sizeof(optval)) == -1) {
55                  LOG(LOG_ERR, "Could not setsocketopt on raw socket");
56                  close(s);
57                  return -1;
58          }
59
60          /* send arp request */
61          memset(&arp, 0, sizeof(arp));
62          memcpy(arp.ethhdr.h_dest, MAC_BCAST_ADDR, 6);   /* MAC DA */
63          memcpy(arp.ethhdr.h_source, mac, 6);            /* MAC SA */
64          arp.ethhdr.h_proto = htons(ETH_P_ARP);          /* protocol type (Ethernet) */
65          arp.htype = htons(ARPHRD_ETHER);                /* hardware type */
66          arp.ptype = htons(ETH_P_IP);                    /* protocol type (ARP message) */
67          arp.hlen = 6;                                   /* hardware address length */
68          arp.plen = 4;                                   /* protocol address length */
69          arp.operation = htons(ARPOP_REQUEST);           /* ARP op code */
70          *((u_int *) arp.sInaddr) = ip;                  /* source IP address */
71          memcpy(arp.sHaddr, mac, 6);                     /* source hardware address */
72          *((u_int *) arp.tInaddr) = yiaddr;              /* target IP address */
73
74          memset(&addr, 0, sizeof(addr));
75          strcpy(addr.sa_data, interface);
76          if (sendto(s, &arp, sizeof(arp), 0, &addr, sizeof(addr)) < 0)
77                  rv = 0;
78
79          /* wait arp reply, and check it */
80          tm.tv_usec = 0;
81          time(&prevTime);
82          while (timeout > 0) {
83                  FD_ZERO(&fdset);
84                  FD_SET(s, &fdset);
85                  tm.tv_sec = timeout;
86                  if (select(s + 1, &fdset, (fd_set *) NULL, (fd_set *) NULL, &tm) < 0) {
87                          DEBUG(LOG_ERR, "Error on ARPING request: %s", strerror(errno));
88                          if (errno != EINTR) rv = 0;
89                  } else if (FD_ISSET(s, &fdset)) {
90                          if (recv(s, &arp, sizeof(arp), 0) < 0 ) rv = 0;
91                          if (arp.operation == htons(ARPOP_REPLY) &&
92                              bcmp(arp.tHaddr, mac, 6) == 0 &&
93                              *((u_int *) arp.sInaddr) == yiaddr) {
94                                  DEBUG(LOG_INFO, "Valid arp reply receved for this address");
95                                  rv = 0;
96                                  break;
97                          }
98                  }
99                  timeout -= time(NULL) - prevTime;
100                  time(&prevTime);
101         }
102         close(s);
103         DEBUG(LOG_INFO, "%salid arp replies for this address", rv ? "No v" : "V");
104         return rv;
105 }
```

4.2 arpping.h File Reference

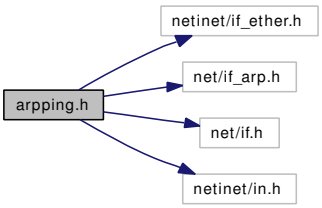
`#include <netinet/if_ether.h>`

`#include <net/if_arp.h>`

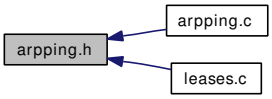
`#include <net/if.h>`

`#include <netinet/in.h>`

Include dependency graph for arpping.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct **arpMsg**

Functions

- int **arpping** (u_int32_t yiaddr, u_int32_t ip, unsigned char *arp, char *interface)

4.2.1 Function Documentation

4.2.1.1 int arpping (u_int32_t *yiaddr*, u_int32_t *ip*, unsigned char * *arp*, char * *interface*)

Definition at line 35 of file arpping.c.

References DEBUG, LOG, LOG_ERR, LOG_INFO, and MAC_BCAST_ADDR.

Referenced by check_ip().

```
36 {
37
38         int     timeout = 2;
39         int     optval = 1;
40         int     s;                      /* socket */
41         int     rv = 1;                 /* return value */
42         struct sockaddr addr;           /* for interface name */
43         struct arpMsg   arp;
44         fd_set          fdset;
```

```
45         struct timeval  tm;
46         time_t          prevTime;
47
48
49         if ((s = socket (PF_PACKET, SOCK_PACKET, htons(ETH_P_ARP))) == -1) {
50                 LOG(LOG_ERR, "Could not open raw socket");
51                 return -1;
52         }
53
54         if (setsockopt(s, SOL_SOCKET, SO_BROADCAST, &optval, sizeof(optval)) == -1) {
55                 LOG(LOG_ERR, "Could not setsocketopt on raw socket");
56                 close(s);
57                 return -1;
58         }
59
60         /* send arp request */
61         memset(&arp, 0, sizeof(arp));
62         memcpy(arp.ethhdr.h_dest, MAC_BCAST_ADDR, 6);   /* MAC DA */
63         memcpy(arp.ethhdr.h_source, mac, 6);            /* MAC SA */
64         arp.ethhdr.h_proto = htons(ETH_P_ARP);          /* protocol type (Ethernet) */
65         arp.htype = htons(ARPHRD_ETHER);                /* hardware type */
66         arp.ptype = htons(ETH_P_IP);                    /* protocol type (ARP message) */
67         arp.hlen = 6;                                   /* hardware address length */
68         arp.plen = 4;                                   /* protocol address length */
69         arp.operation = htons(ARPOP_REQUEST);           /* ARP op code */
70         *((u_int *) arp.sInaddr) = ip;                  /* source IP address */
71         memcpy(arp.sHaddr, mac, 6);                     /* source hardware address */
72         *((u_int *) arp.tInaddr) = yiaddr;              /* target IP address */
73
74         memset(&addr, 0, sizeof(addr));
75         strcpy(addr.sa_data, interface);
76         if (sendto(s, &arp, sizeof(arp), 0, &addr, sizeof(addr)) < 0)
77                 rv = 0;
78
79         /* wait arp reply, and check it */
80         tm.tv_usec = 0;
81         time(&prevTime);
82         while (timeout > 0) {
83                 FD_ZERO(&fdset);
84                 FD_SET(s, &fdset);
85                 tm.tv_sec = timeout;
86                 if (select(s + 1, &fdset, (fd_set *) NULL, (fd_set *) NULL, &tm) < 0) {
87                         DEBUG(LOG_ERR, "Error on ARPING request: %s", strerror(errno));
88                         if (errno != EINTR) rv = 0;
89                 } else if (FD_ISSET(s, &fdset)) {
90                         if (recv(s, &arp, sizeof(arp), 0) < 0 ) rv = 0;
91                         if (arp.operation == htons(ARPOP_REPLY) &&
92                             bcmp(arp.tHaddr, mac, 6) == 0 &&
93                             *((u_int *) arp.sInaddr) == yiaddr) {
94                                 DEBUG(LOG_INFO, "Valid arp reply receved for this address");
95                                 rv = 0;
96                                 break;
97                         }
98                 }
99                 timeout -= time(NULL) - prevTime;
100                 time(&prevTime);
101         }
102         close(s);
103         DEBUG(LOG_INFO, "%salid arp replies for this address", rv ? "No v" : "V");
104         return rv;
105 }
```

4.3 clientpacket.c File Reference

```
#include <string.h>
#include <sys/socket.h>
#include <features.h>
#include <asm/types.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "dhcpd.h"
#include "packet.h"
#include "options.h"
#include "dhcpc.h"
#include "debug.h"
```

Include dependency graph for clientpacket.c:

Functions

- unsigned long **random_xid** (void)
- int **send_discover** (unsigned long xid, unsigned long requested)
- int **send_selecting** (unsigned long xid, unsigned long server, unsigned long requested)
- int **send_renew** (unsigned long xid, unsigned long server, unsigned long ciaddr)
- int **send_release** (unsigned long server, unsigned long ciaddr)
- int **get_raw_packet** (struct **dhcpMessage** *payload, int fd)

4.3.1 Function Documentation

4.3.1.1 int get_raw_packet (struct dhcpMessage * *payload*, int *fd*)

Definition at line 182 of file clientpacket.c.

References checksum(), CLIENT_PORT, dhcpMessage::cookie, udp_dhcp_packet::data, DEBUG, DHCP_MAGIC, udp_dhcp_packet::ip, LOG, LOG_ERR, LOG_INFO, and udp_dhcp_packet::udp.

Referenced by udhcp().

```
183 {
184         int bytes;
185         struct udp_dhcp_packet packet;
186         u_int32_t source, dest;
187         u_int16_t check;
188 
189         memset(&packet, 0, sizeof(struct udp_dhcp_packet));
190         bytes = read(fd, &packet, sizeof(struct udp_dhcp_packet));
191         if (bytes < 0) {
192                 DEBUG(LOG_INFO, "couldn't read on raw listening socket -- ignoring");
193                 usleep(500000); /* possible down interface, looping condition */
194                 return -1;
195         }
196 
197         if (bytes < (int) (sizeof(struct iphdr) + sizeof(struct udphdr))) {
198                 DEBUG(LOG_INFO, "message too short, ignoring");
199                 return -2;
200         }
201 
202         if (bytes < ntohs(packet.ip.tot_len)) {
203                 DEBUG(LOG_INFO, "Truncated packet");
204                 return -2;
205         }
206 
207         /* ignore any extra garbage bytes */
208         bytes = ntohs(packet.ip.tot_len);
209 
210         /* Make sure its the right packet for us, and that it passes sanity checks */
211         if (packet.ip.protocol != IPPROTO_UDP || packet.ip.version != IPVERSION ||
212             packet.ip.ihl != sizeof(packet.ip) >> 2 || packet.udp.dest != htons(CLIENT_PORT) ||
213             bytes > (int) sizeof(struct udp_dhcp_packet) ||
214             ntohs(packet.udp.len) != (short) (bytes - sizeof(packet.ip))) {
215                 DEBUG(LOG_INFO, "unrelated/bogus packet");
216                 return -2;
217         }
218 
219         /* check IP checksum */
220         check = packet.ip.check;
221         packet.ip.check = 0;
222         if (check != checksum(&(packet.ip), sizeof(packet.ip))) {
223                 DEBUG(LOG_INFO, "bad IP header checksum, ignoring");
224                 return -1;
225         }
226 
227         /* verify the UDP checksum by replacing the header with a psuedo header */
228         source = packet.ip.saddr;
229         dest = packet.ip.daddr;
230         check = packet.udp.check;
231         packet.udp.check = 0;
232         memset(&packet.ip, 0, sizeof(packet.ip));
233 
234         packet.ip.protocol = IPPROTO_UDP;
235         packet.ip.saddr = source;
236         packet.ip.daddr = dest;
237         packet.ip.tot_len = packet.udp.len; /* cheat on the psuedo-header */
238         if (check && check != checksum(&packet, bytes)) {
239                 DEBUG(LOG_ERR, "packet with bad UDP checksum received, ignoring");
240                 return -2;
241         }
242 
243         memcpy(payload, &(packet.data), bytes - (sizeof(packet.ip) + sizeof(packet.udp)));
244 
245         if (ntohl(payload->cookie) != DHCP_MAGIC) {
246                 LOG(LOG_ERR, "received bogus message (bad magic) -- ignoring");
247                 return -2;
```

```
248         }
249         DEBUG(LOG_INFO, "oooooh!!! got some!");
250         return bytes - (sizeof(packet.ip) + sizeof(packet.udp));
251
252 }
```

Here is the call graph for this function:

4.3.1.2 unsigned long random_xid (void)

Definition at line 52 of file clientpacket.c.

References LOG, and LOG_WARNING.

Referenced by send_release(), and udhcp().

```
53 {
54         static int initialized;
55         if (!initialized) {
56                 int fd;
57                 unsigned long seed;
58
59                 fd = open("/dev/urandom", 0);
60                 if (fd < 0 || read(fd, &seed, sizeof(seed)) < 0) {
61                         LOG(LOG_WARNING, "Could not load seed from /dev/urandom: %s",
62                                 strerror(errno));
63                         seed = time(0);
64                 }
65                 if (fd >= 0) close(fd);
66                 srand(seed);
67                 initialized++;
68         }
69         return rand();
70 }
```

4.3.1.3 int send_discover (unsigned long *xid*, unsigned long *requested*)

Definition at line 108 of file clientpacket.c.

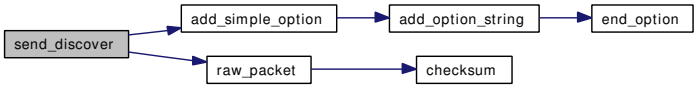
References add_simple_option(), client_config, CLIENT_PORT, DHCP_REQUESTED_IP, DHCPDISCOVER, client_config_t::ifindex, LOG, LOG_DEBUG, MAC_BCAST_ADDR, dhcpMessage::options, raw_packet(), SERVER_PORT, and dhcpMessage::xid.

Referenced by udhcp().

```
109 {
110         struct dhcpMessage packet;
111
112         init_packet(&packet, DHCPDISCOVER);
113         packet.xid = xid;
114         if (requested)
115                 add_simple_option(packet.options, DHCP_REQUESTED_IP, requested);
116
117         add_requests(&packet);
118         LOG(LOG_DEBUG, "Sending discover...");
```

```
119         return raw_packet(&packet, INADDR_ANY, CLIENT_PORT, INADDR_BROADCAST,
120                                 SERVER_PORT, MAC_BCAST_ADDR, client_config.ifindex);
121 }
```

Here is the call graph for this function:



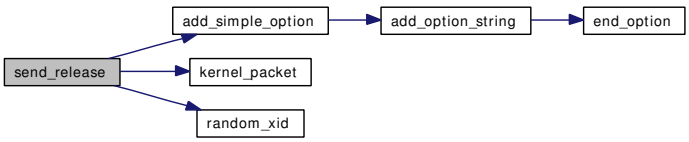
4.3.1.4 int send_release (unsigned long *server*, unsigned long *ciaddr*)

Definition at line 165 of file clientpacket.c.

References add_simple_option(), dhcpMessage::ciaddr, CLIENT_PORT, DHCP_REQUESTED_IP, DHCP_SERVER_ID, DHCPRELEASE, kernel_packet(), LOG, LOG_DEBUG, dhcpMessage::options, random_xid(), SERVER_PORT, and dhcpMessage::xid.

```
166 {
167         struct dhcpMessage packet;
168 
169         init_packet(&packet, DHCPRELEASE);
170         packet.xid = random_xid();
171         packet.ciaddr = ciaddr;
172 
173         add_simple_option(packet.options, DHCP_REQUESTED_IP, ciaddr);
174         add_simple_option(packet.options, DHCP_SERVER_ID, server);
175 
176         LOG(LOG_DEBUG, "Sending release...");
177         return kernel_packet(&packet, ciaddr, CLIENT_PORT, server, SERVER_PORT);
178 }
```

Here is the call graph for this function:



4.3.1.5 int send_renew (unsigned long *xid*, unsigned long *server*, unsigned long *ciaddr*)

Definition at line 145 of file clientpacket.c.

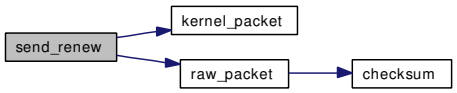
References dhcpMessage::ciaddr, client_config, CLIENT_PORT, DHCPREQUEST, client_config_t::ifindex, kernel_packet(), LOG, LOG_DEBUG, MAC_BCAST_ADDR, raw_packet(), SERVER_PORT, and dhcpMessage::xid.

Referenced by udhcp().

```
146 {
147         struct dhcpMessage packet;
```

```
148         int ret = 0;
149
150         init_packet(&packet, DHCPREQUEST);
151         packet.xid = xid;
152         packet.ciaddr = ciaddr;
153
154         add_requests(&packet);
155         LOG(LOG_DEBUG, "Sending renew...");
156         if (server)
157                 ret = kernel_packet(&packet, ciaddr, CLIENT_PORT, server, SERVER_PORT);
158         else ret = raw_packet(&packet, INADDR_ANY, CLIENT_PORT, INADDR_BROADCAST,
159                               SERVER_PORT, MAC_BCAST_ADDR, client_config.ifindex);
160         return ret;
161 }
```

Here is the call graph for this function:



4.3.1.6 int send_selecting (unsigned long *xid*, unsigned long *server*, unsigned long *requested*)

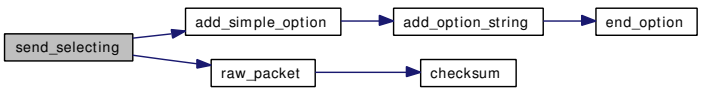
Definition at line 125 of file clientpacket.c.

References add_simple_option(), client_config, CLIENT_PORT, DHCP_REQUESTED_IP, DHCP_SERVER_ID, DHCPREQUEST, client_config_t::ifindex, LOG, LOG_DEBUG, MAC_BCAST_ADDR, dhcpMessage::options, raw_packet(), SERVER_PORT, and dhcpMessage::xid.

Referenced by udhcp().

```
126 {
127         struct dhcpMessage packet;
128         struct in_addr addr;
129
130         init_packet(&packet, DHCPREQUEST);
131         packet.xid = xid;
132
133         add_simple_option(packet.options, DHCP_REQUESTED_IP, requested);
134         add_simple_option(packet.options, DHCP_SERVER_ID, server);
135
136         add_requests(&packet);
137         addr.s_addr = requested;
138         LOG(LOG_DEBUG, "Sending select for %s...", inet_ntoa(addr));
139         return raw_packet(&packet, INADDR_ANY, CLIENT_PORT, INADDR_BROADCAST,
140                               SERVER_PORT, MAC_BCAST_ADDR, client_config.ifindex);
141 }
```

Here is the call graph for this function:



4.4 clientpacket.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- unsigned long **random_xid** (void)
- int **send_discover** (unsigned long xid, unsigned long requested)
- int **send_selecting** (unsigned long xid, unsigned long server, unsigned long requested)
- int **send_renew** (unsigned long xid, unsigned long server, unsigned long ciaddr)
- int **send_release** (unsigned long server, unsigned long ciaddr)
- int **get_raw_packet** (struct **dhcpMessage** *payload, int fd)

4.4.1 Function Documentation

4.4.1.1 int get_raw_packet (struct dhcpMessage * *payload*, int *fd*)

Definition at line 182 of file clientpacket.c.

References checksum(), CLIENT_PORT, dhcpMessage::cookie, udp_dhcp_packet::data, DEBUG, DHCP_MAGIC, udp_dhcp_packet::ip, LOG, LOG_ERR, LOG_INFO, and udp_dhcp_packet::udp.

Referenced by udhcp().

```
183 {
184         int bytes;
185         struct udp_dhcp_packet packet;
186         u_int32_t source, dest;
187         u_int16_t check;
188
189         memset(&packet, 0, sizeof(struct udp_dhcp_packet));
190         bytes = read(fd, &packet, sizeof(struct udp_dhcp_packet));
191         if (bytes < 0) {
192                 DEBUG(LOG_INFO, "couldn't read on raw listening socket -- ignoring");
193                 usleep(500000); /* possible down interface, looping condition */
194                 return -1;
195         }
196
197         if (bytes < (int) (sizeof(struct iphdr) + sizeof(struct udphdr))) {
198                 DEBUG(LOG_INFO, "message too short, ignoring");
199                 return -2;
200         }
201
202         if (bytes < ntohs(packet.ip.tot_len)) {
203                 DEBUG(LOG_INFO, "Truncated packet");
204                 return -2;
205         }
206
207         /* ignore any extra garbage bytes */
208         bytes = ntohs(packet.ip.tot_len);
209
210         /* Make sure its the right packet for us, and that it passes sanity checks */
211         if (packet.ip.protocol != IPPROTO_UDP || packet.ip.version != IPVERSION ||
212             packet.ip.ihl != sizeof(packet.ip) >> 2 || packet.udp.dest != htons(CLIENT_PORT) ||
```

```
213             bytes > (int) sizeof(struct udp_dhcp_packet) ||
214             ntohs(packet.udp.len) != (short) (bytes - sizeof(packet.ip))) {
215                 DEBUG(LOG_INFO, "unrelated/bogus packet");
216                 return -2;
217         }
218 
219         /* check IP checksum */
220         check = packet.ip.check;
221         packet.ip.check = 0;
222         if (check != checksum(&(packet.ip), sizeof(packet.ip))) {
223                 DEBUG(LOG_INFO, "bad IP header checksum, ignoring");
224                 return -1;
225         }
226 
227         /* verify the UDP checksum by replacing the header with a psuedo header */
228         source = packet.ip.saddr;
229         dest = packet.ip.daddr;
230         check = packet.udp.check;
231         packet.udp.check = 0;
232         memset(&packet.ip, 0, sizeof(packet.ip));
233 
234         packet.ip.protocol = IPPROTO_UDP;
235         packet.ip.saddr = source;
236         packet.ip.daddr = dest;
237         packet.ip.tot_len = packet.udp.len; /* cheat on the psuedo-header */
238         if (check && check != checksum(&packet, bytes)) {
239                 DEBUG(LOG_ERR, "packet with bad UDP checksum received, ignoring");
240                 return -2;
241         }
242 
243         memcpy(payload, &(packet.data), bytes - (sizeof(packet.ip) + sizeof(packet.udp)));
244 
245         if (ntohl(payload->cookie) != DHCP_MAGIC) {
246                 LOG(LOG_ERR, "received bogus message (bad magic) -- ignoring");
247                 return -2;
248         }
249         DEBUG(LOG_INFO, "oooooh!!! got some!");
250         return bytes - (sizeof(packet.ip) + sizeof(packet.udp));
251 
252 }
```

Here is the call graph for this function:

4.4.1.2 unsigned long random_xid (void)

Definition at line 52 of file clientpacket.c.

References LOG, and LOG_WARNING.

Referenced by send_release(), and udhcp().

```
53 {
54         static int initialized;
55         if (!initialized) {
56                 int fd;
57                 unsigned long seed;
58 
59                 fd = open("/dev/urandom", 0);
60                 if (fd < 0 || read(fd, &seed, sizeof(seed)) < 0) {
```

```
61                          LOG(LOG_WARNING, "Could not load seed from /dev/urandom: %s",
62                                  strerror(errno));
63                          seed = time(0);
64                  }
65                  if (fd >= 0) close(fd);
66                  srand(seed);
67                  initialized++;
68          }
69          return rand();
70 }
```

4.4.1.3 int send_discover (unsigned long *xid*, unsigned long *requested*)

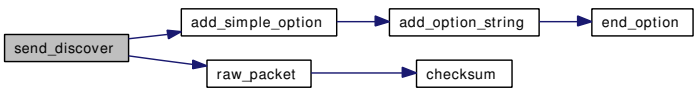
Definition at line 108 of file clientpacket.c.

References add_simple_option(), client_config, CLIENT_PORT, DHCP_REQUESTED_IP, DHCPDISCOVER, client_config_t::ifindex, LOG, LOG_DEBUG, MAC_BCAST_ADDR, dhcpMessage::options, raw_packet(), SERVER_PORT, and dhcpMessage::xid.

Referenced by udhcp().

```
109 {
110         struct dhcpMessage packet;
111
112         init_packet(&packet, DHCPDISCOVER);
113         packet.xid = xid;
114         if (requested)
115                 add_simple_option(packet.options, DHCP_REQUESTED_IP, requested);
116
117         add_requests(&packet);
118         LOG(LOG_DEBUG, "Sending discover...");
119         return raw_packet(&packet, INADDR_ANY, CLIENT_PORT, INADDR_BROADCAST,
120                                 SERVER_PORT, MAC_BCAST_ADDR, client_config.ifindex);
121 }
```

Here is the call graph for this function:



4.4.1.4 int send_release (unsigned long *server*, unsigned long *ciaddr*)

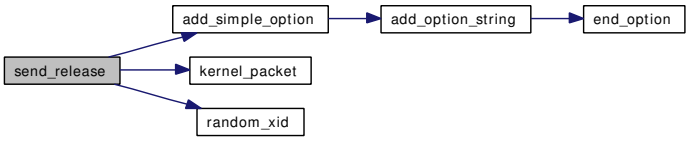
Definition at line 165 of file clientpacket.c.

References add_simple_option(), dhcpMessage::ciaddr, CLIENT_PORT, DHCP_REQUESTED_IP, DHCP_SERVER_ID, DHCPRELEASE, kernel_packet(), LOG, LOG_DEBUG, dhcpMessage::options, random_xid(), SERVER_PORT, and dhcpMessage::xid.

```
166 {
167         struct dhcpMessage packet;
168
169         init_packet(&packet, DHCPRELEASE);
170         packet.xid = random_xid();
171         packet.ciaddr = ciaddr;
172
173         add_simple_option(packet.options, DHCP_REQUESTED_IP, ciaddr);
```

```
174         add_simple_option(packet.options, DHCP_SERVER_ID, server);
175
176         LOG(LOG_DEBUG, "Sending release...");
177         return kernel_packet(&packet, ciaddr, CLIENT_PORT, server, SERVER_PORT);
178 }
```

Here is the call graph for this function:



4.4.1.5 int send_renew (unsigned long *xid*, unsigned long *server*, unsigned long *ciaddr*)

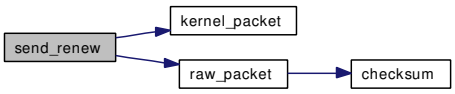
Definition at line 145 of file clientpacket.c.

References dhcpMessage::ciaddr, client_config, CLIENT_PORT, DHCPREQUEST, client_config_t::ifindex, kernel_packet(), LOG, LOG_DEBUG, MAC_BCAST_ADDR, raw_packet(), SERVER_PORT, and dhcpMessage::xid.

Referenced by udhcp().

```
146 {
147         struct dhcpMessage packet;
148         int ret = 0;
149
150         init_packet(&packet, DHCPREQUEST);
151         packet.xid = xid;
152         packet.ciaddr = ciaddr;
153
154         add_requests(&packet);
155         LOG(LOG_DEBUG, "Sending renew...");
156         if (server)
157                 ret = kernel_packet(&packet, ciaddr, CLIENT_PORT, server, SERVER_PORT);
158         else ret = raw_packet(&packet, INADDR_ANY, CLIENT_PORT, INADDR_BROADCAST,
159                               SERVER_PORT, MAC_BCAST_ADDR, client_config.ifindex);
160         return ret;
161 }
```

Here is the call graph for this function:



4.4.1.6 int send_selecting (unsigned long *xid*, unsigned long *server*, unsigned long *requested*)

Definition at line 125 of file clientpacket.c.

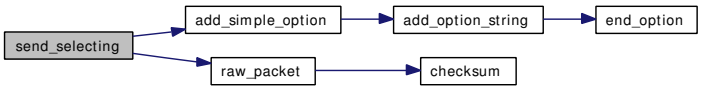
References add_simple_option(), client_config, CLIENT_PORT, DHCP_REQUESTED_IP, DHCP_SERVER_ID, DHCPREQUEST, client_config_t::ifindex, LOG, LOG_DEBUG,

MAC_BCAST_ADDR, dhcpMessage::options, raw_packet(), SERVER_PORT, and dhcpMessage::xid.

Referenced by udhcp().

```
126 {
127         struct dhcpMessage packet;
128         struct in_addr addr;
129 
130         init_packet(&packet, DHCPREQUEST);
131         packet.xid = xid;
132 
133         add_simple_option(packet.options, DHCP_REQUESTED_IP, requested);
134         add_simple_option(packet.options, DHCP_SERVER_ID, server);
135 
136         add_requests(&packet);
137         addr.s_addr = requested;
138         LOG(LOG_DEBUG, "Sending select for %s...", inet_ntoa(addr));
139         return raw_packet(&packet, INADDR_ANY, CLIENT_PORT, INADDR_BROADCAST,
140                                 SERVER_PORT, MAC_BCAST_ADDR, client_config.ifindex);
141 }
```

Here is the call graph for this function:

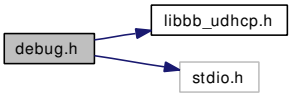


4.5 debug.h File Reference

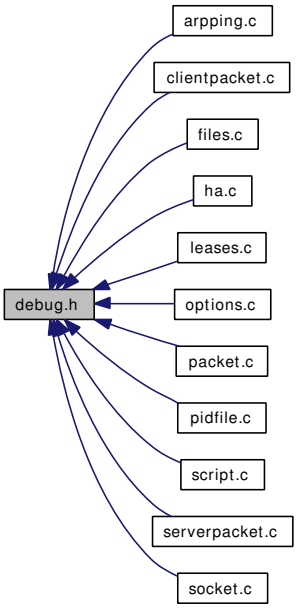
```
#include "libbb_udhcp.h"
```

```
#include <stdio.h>
```

Include dependency graph for debug.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **LOG_EMERG** "EMERGENCY!"
- #define **LOG_ALERT** "ALERT!"
- #define **LOG_CRIT** "critical!"
- #define **LOG_WARNING** "warning"
- #define **LOG_ERR** "error"
- #define **LOG_INFO** "info"
- #define **LOG_DEBUG** "debug"
- #define **LOG**(level, str, args...)
- #define **OPEN_LOG**(name) do {;} while(0)
- #define **CLOSE_LOG**() do {;} while(0)
- #define **DEBUG**(level, str, args...) do {;} while(0)

4.5.1 Define Documentation

4.5.1.1 #define CLOSE_LOG() do {;} while(0)

Definition at line 30 of file debug.h.

4.5.1.2 #define DEBUG(level, str, args...) do {;} while(0)

Definition at line 38 of file debug.h.

Referenced by add_option_string(), add_simple_option(), arpping(), attach_option(), get_packet(), get_raw_packet(), listen_socket(), raw_packet(), raw_socket(), read_interface(), read_leases(), run_script(), sendNAK(), and udhcp().

4.5.1.3 #define LOG(level, str, args...)

Value:

```
do { printf("%s, ", level); \
                                printf(str, ## args); \
                                printf("\n"); } while(0)
```

Definition at line 26 of file debug.h.

Referenced by add_option_string(), arpping(), check_ip(), get_option(), get_packet(), get_raw_packet(), pidfile_acquire(), random_xid(), read_config(), read_interface(), read_leases(), run_script(), send_discover(), send_release(), send_renew(), send_selecting(), sendACK(), sendOffer(), udhcp(), and write_leases().

4.5.1.4 #define LOG_ALERT "ALERT!"

Definition at line 20 of file debug.h.

4.5.1.5 #define LOG_CRIT "critical!"

Definition at line 21 of file debug.h.

4.5.1.6 #define LOG_DEBUG "debug"

Definition at line 25 of file debug.h.

Referenced by send_discover(), send_release(), send_renew(), and send_selecting().

4.5.1.7 #define LOG_EMERG "EMERGENCY!"

Definition at line 19 of file debug.h.

4.5.1.8 #define LOG_ERR "error"

Definition at line 23 of file debug.h.

Referenced by add_option_string(), add_simple_option(), arpping(), get_packet(), get_raw_packet(), listen_socket(), pidfile_acquire(), raw_packet(), raw_socket(), read_config(), read_interface(), read_leases(), run_script(), udhcp(), and write_leases().

4.5.1.9 #define LOG_INFO "info"

Definition at line 24 of file debug.h.

Referenced by add_option_string(), arpping(), attach_option(), check_ip(), get_packet(), get_raw_packet(), listen_socket(), raw_socket(), read_interface(), read_leases(), run_script(), sendACK(), sendNAK(), sendOffer(), and udhcp().

4.5.1.10 #define LOG_WARNING "warning"

Definition at line 22 of file debug.h.

Referenced by get_option(), random_xid(), read_leases(), and sendOffer().

4.5.1.11 #define OPEN_LOG(name) do {;} while(0)

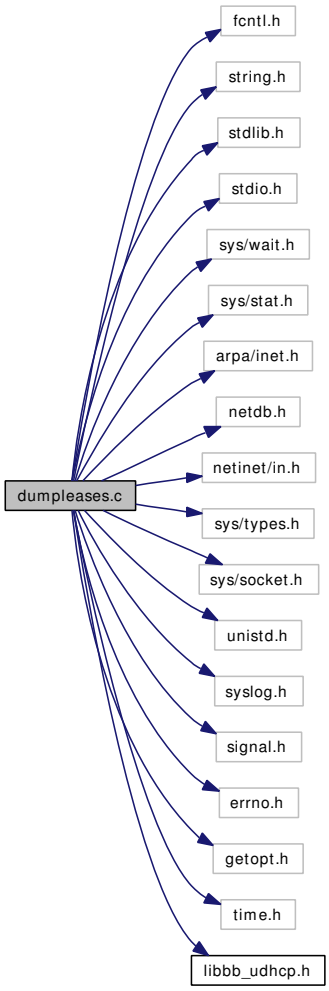
Definition at line 29 of file debug.h.

Referenced by udhcp().

4.6 dumpleases.c File Reference

```
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <syslog.h>
#include <signal.h>
#include <errno.h>
#include <getopt.h>
#include <time.h>
#include "libbb_udhcp.h"
```

Include dependency graph for dumpleases.c:



Classes

- struct **lease_t**

Defines

- #define **REMAINING** 0
- #define **ABSOLUTE** 1

Functions

- int **main** (int argc, char *argv[])

4.6.1 Define Documentation

4.6.1.1 #define ABSOLUTE 1

Definition at line 23 of file dumpleases.c.

Referenced by main().

4.6.1.2 #define REMAINING 0

Definition at line 22 of file dumpleases.c.

Referenced by main().

4.6.2 Function Documentation

4.6.2.1 int main (int *argc*, char * *argv*[])

Definition at line 34 of file dumpleases.c.

References ABSOLUTE, and REMAINING.

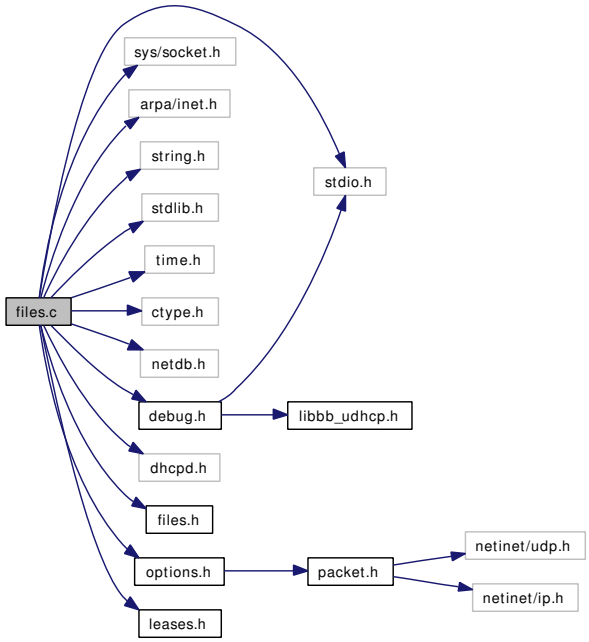
```
36 {
37         FILE *fp;
38         int i, c, mode = REMAINING;
39         long expires;
40         char file[255] = "/var/lib/misc/udhcpd.leases";
41         struct lease_t lease;
42         struct in_addr addr;
43 
44         static struct option options[] = {
45                 {"absolute", 0, 0, 'a'},
46                 {"remaining", 0, 0, 'r'},
47                 {"file", 1, 0, 'f'},
48                 {"help", 0, 0, 'h'},
49                 {0, 0, 0, 0}
50         };
51 
52         while (1) {
53                 int option_index = 0;
54                 c = getopt_long(argc, argv, "arf:h", options, &option_index);
55                 if (c == -1) break;
56 
57                 switch (c) {
58                 case 'a': mode = ABSOLUTE; break;
59                 case 'r': mode = REMAINING; break;
60                 case 'f':
61                         strncpy(file, optarg, 255);
62                         file[254] = '\0';
63                         break;
64                 case 'h':
65                         printf("Usage: dumpleases -f <file> -[r|a]\n\n");
66                         printf("  -f, --file=FILENAME             Leases file to load\n");
67                         printf("  -r, --remaining                 Interepret lease times as time remaing\n");
68                         printf("  -a, --absolute                  Interepret lease times as expire time\n");
69                         break;
70                 }
71         }
72 
73         if (!(fp = fopen(file, "r"))) {
74                 perror("could not open input file");
75                 return 0;
76         }
77 
78         printf("Mac Address       IP-Address      Expires %s\n", mode == REMAINING ? "in" : "at");
79         /*     "00:00:00:00:00:00 255.255.255.255 Wed Jun 30 21:49:08 1993" */
80         while (fread(&lease, sizeof(lease), 1, fp)) {
81 
82                 for (i = 0; i < 6; i++) {
```

```
83                      printf("%02x", lease.chaddr[i]);
84                      if (i != 5) printf(":");
85              }
86              addr.s_addr = lease.yiaddr;
87              printf(" %-15s", inet_ntoa(addr));
88              expires = ntohl(lease.expires);
89              printf(" ");
90              if (mode == REMAINING) {
91                      if (!expires) printf("expired\n");
92                      else {
93                              if (expires > 60*60*24) {
94                                      printf("%ld days, ", expires / (60*60*24));
95                                      expires %= 60*60*24;
96                              }
97                              if (expires > 60*60) {
98                                      printf("%ld hours, ", expires / (60*60));
99                                      expires %= 60*60;
100                             }
101                             if (expires > 60) {
102                                     printf("%ld minutes, ", expires / 60);
103                                     expires %= 60;
104                             }
105                             printf("%ld seconds\n", expires);
106                     }
107             } else printf("%s", ctime(&expires));
108     }
109     fclose(fp);
110
111     return 0;
112 }
```

4.7 files.c File Reference

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <netdb.h>
#include "debug.h"
#include "dhcpd.h"
#include "files.h"
#include "options.h"
#include "leases.h"
```

Include dependency graph for files.c:



Functions

- int **read_config** (char *file)
- void **write_leases** (void)
- void **read_leases** (char *file)

4.7.1 Function Documentation

4.7.1.1 int read_config (char * *file*)

Definition at line 177 of file files.c.

References config_keyword::def, config_keyword::handler, config_keyword::keyword, LOG, LOG_ERR, and config_keyword::var.

```
178 {
179         FILE *in;
180         char buffer[80], orig[80], *token, *line;
181         int i;
182 
183         for (i = 0; strlen(keywords[i].keyword); i++)
184                 if (strlen(keywords[i].def))
185                         keywords[i].handler(keywords[i].def, keywords[i].var);
186 
187         if (!(in = fopen(file, "r"))) {
188                 LOG(LOG_ERR, "unable to open config file: %s", file);
189                 return 0;
190         }
191 
192         while (fgets(buffer, 80, in)) {
193                 if (strchr(buffer, '\n')) *(strchr(buffer, '\n')) = '\0';
194                 strncpy(orig, buffer, 80);
195                 if (strchr(buffer, '#')) *(strchr(buffer, '#')) = '\0';
196                 token = buffer + strspn(buffer, " \t");
197                 if (*token == '\0') continue;
198                 line = token + strcspn(token, " \t=");
199                 if (*line == '\0') continue;
200                 *line = '\0';
201                 line++;
202 
203                 /* eat leading whitespace */
204                 line = line + strspn(line, " \t=");
205                 /* eat trailing whitespace */
206                 for (i = strlen(line); i > 0 && isspace(line[i - 1]); i--);
207                 line[i] = '\0';
208 
209                 for (i = 0; strlen(keywords[i].keyword); i++)
210                         if (!strcasecmp(token, keywords[i].keyword))
211                                 if (!keywords[i].handler(line, keywords[i].var)) {
212                                         LOG(LOG_ERR, "unable to parse '%s'", orig);
213                                         /* reset back to the default value */
214                                         keywords[i].handler(keywords[i].def, keywords[i].var);
215                                 }
216         }
217         fclose(in);
218         return 1;
219 }
```

4.7.1.2 void read_leases (char * *file*)

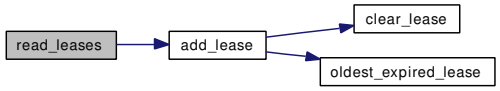
Definition at line 257 of file files.c.

References add_lease(), DEBUG, LOG, LOG_ERR, LOG_INFO, and LOG_WARNING.

```
258 {
259         FILE *fp;
260         unsigned int i = 0;
261         struct dhcpOfferedAddr lease;
262 
```

```
263         if (!(fp = fopen(file, "r"))) {
264                 LOG(LOG_ERR, "Unable to open %s for reading", file);
265                 return;
266         }
267
268         while (i < server_config.max_leases && (fread(&lease, sizeof lease, 1, fp) == 1)) {
269                 /* ADDME: is it a static lease */
270                 if (lease.yiaddr >= server_config.start && lease.yiaddr <= server_config.end) {
271                         lease.expires = ntohl(lease.expires);
272                         if (!server_config.remaining) lease.expires -= time(0);
273                         if (!(add_lease(lease.chaddr, lease.yiaddr, lease.expires))) {
274                                 LOG(LOG_WARNING, "Too many leases while loading %s\n", file);
275                                 break;
276                         }
277                         i++;
278                 }
279         }
280         DEBUG(LOG_INFO, "Read %d leases", i);
281         fclose(fp);
282 }
```

Here is the call graph for this function:



4.7.1.3 void write_leases (void)

Definition at line 222 of file files.c.

References lease_expired(), LOG, and LOG_ERR.

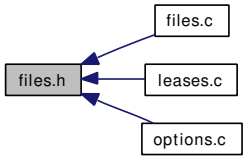
```
223 {
224         FILE *fp;
225         unsigned int i;
226         char buf[255];
227         time_t curr = time(0);
228         unsigned long lease_time;
229
230         if (!(fp = fopen(server_config.lease_file, "w"))) {
231                 LOG(LOG_ERR, "Unable to open %s for writing", server_config.lease_file);
232                 return;
233         }
234
235         for (i = 0; i < server_config.max_leases; i++) {
236                 if (leases[i].yiaddr != 0) {
237                         if (server_config.remaining) {
238                                 if (lease_expired(&(leases[i])))
239                                         lease_time = 0;
240                                 else lease_time = leases[i].expires - curr;
241                         } else lease_time = leases[i].expires;
242                         lease_time = htonl(lease_time);
243                         fwrite(leases[i].chaddr, 16, 1, fp);
244                         fwrite(&(leases[i].yiaddr), 4, 1, fp);
245                         fwrite(&lease_time, 4, 1, fp);
246                 }
247         }
248         fclose(fp);
249
250         if (server_config.notify_file) {
```

```
251                 sprintf(buf, "%s %s", server_config.notify_file, server_config.lease_file);
252                 system(buf);
253         }
254 }
```

Here is the call graph for this function:

4.8 files.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct **config_keyword**

Functions

- int **read_config** (char *file)
- void **write_leases** (void)
- void **read_leases** (char *file)

4.8.1 Function Documentation

4.8.1.1 int read_config (char * *file*)

Definition at line 177 of file files.c.

References config_keyword::def, config_keyword::handler, config_keyword::keyword, LOG, LOG_ERR, and config_keyword::var.

```
178 {
179         FILE *in;
180         char buffer[80], orig[80], *token, *line;
181         int i;
182 
183         for (i = 0; strlen(keywords[i].keyword); i++)
184                 if (strlen(keywords[i].def))
185                         keywords[i].handler(keywords[i].def, keywords[i].var);
186 
187         if (!(in = fopen(file, "r"))) {
188                 LOG(LOG_ERR, "unable to open config file: %s", file);
189                 return 0;
190         }
191 
192         while (fgets(buffer, 80, in)) {
193                 if (strchr(buffer, '\n')) *(strchr(buffer, '\n')) = '\0';
194                 strncpy(orig, buffer, 80);
195                 if (strchr(buffer, '#')) *(strchr(buffer, '#')) = '\0';
196                 token = buffer + strspn(buffer, " \t");
197                 if (*token == '\0') continue;
198                 line = token + strcspn(token, " \t=");
199                 if (*line == '\0') continue;
200                 *line = '\0';
201                 line++;
202 
203                 /* eat leading whitespace */
204                 line = line + strspn(line, " \t=");
```

```
205                 /* eat trailing whitespace */
206                 for (i = strlen(line); i > 0 && isspace(line[i - 1]); i--);
207                 line[i] = '\0';
208
209                 for (i = 0; strlen(keywords[i].keyword); i++)
210                         if (!strcasecmp(token, keywords[i].keyword))
211                                 if (!keywords[i].handler(line, keywords[i].var)) {
212                                         LOG(LOG_ERR, "unable to parse '%s'", orig);
213                                         /* reset back to the default value */
214                                         keywords[i].handler(keywords[i].def, keywords[i].var);
215                                 }
216         }
217         fclose(in);
218         return 1;
219 }
```

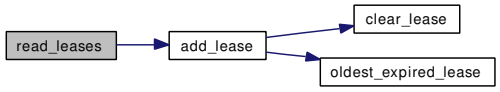
4.8.1.2 void read_leases (char * *file*)

Definition at line 257 of file files.c.

References add_lease(), DEBUG, LOG, LOG_ERR, LOG_INFO, and LOG_WARNING.

```
258 {
259         FILE *fp;
260         unsigned int i = 0;
261         struct dhcpOfferedAddr lease;
262
263         if (!(fp = fopen(file, "r"))) {
264                 LOG(LOG_ERR, "Unable to open %s for reading", file);
265                 return;
266         }
267
268         while (i < server_config.max_leases && (fread(&lease, sizeof lease, 1, fp) == 1)) {
269                 /* ADDME: is it a static lease */
270                 if (lease.yiaddr >= server_config.start && lease.yiaddr <= server_config.end) {
271                         lease.expires = ntohl(lease.expires);
272                         if (!server_config.remaining) lease.expires -= time(0);
273                         if (!(add_lease(lease.chaddr, lease.yiaddr, lease.expires))) {
274                                 LOG(LOG_WARNING, "Too many leases while loading %s\n", file);
275                                 break;
276                         }
277                         i++;
278                 }
279         }
280         DEBUG(LOG_INFO, "Read %d leases", i);
281         fclose(fp);
282 }
```

Here is the call graph for this function:



4.8.1.3 void write_leases (void)

Definition at line 222 of file files.c.

References lease_expired(), LOG, and LOG_ERR.

```
223 {
224         FILE *fp;
225         unsigned int i;
226         char buf[255];
227         time_t curr = time(0);
228         unsigned long lease_time;
229 
230         if (!(fp = fopen(server_config.lease_file, "w"))) {
231                 LOG(LOG_ERR, "Unable to open %s for writing", server_config.lease_file);
232                 return;
233         }
234 
235         for (i = 0; i < server_config.max_leases; i++) {
236                 if (leases[i].yiaddr != 0) {
237                         if (server_config.remaining) {
238                                 if (lease_expired(&(leases[i])))
239                                         lease_time = 0;
240                                 else lease_time = leases[i].expires - curr;
241                         } else lease_time = leases[i].expires;
242                         lease_time = htonl(lease_time);
243                         fwrite(leases[i].chaddr, 16, 1, fp);
244                         fwrite(&(leases[i].yiaddr), 4, 1, fp);
245                         fwrite(&lease_time, 4, 1, fp);
246                 }
247         }
248         fclose(fp);
249 
250         if (server_config.notify_file) {
251                 sprintf(buf, "%s %s", server_config.notify_file, server_config.lease_file);
252                 system(buf);
253         }
254 }
```

Here is the call graph for this function:

4.9 frontend.c File Reference

```
#include <string.h>
```

Include dependency graph for frontend.c:

Functions

- int **udhcpd_main** (int argc, char *argv[])
- int **udhcpc_main** (int argc, char *argv[])
- int **main** (int argc, char *argv[])

4.9.1 Function Documentation

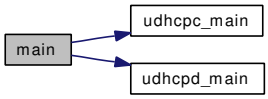
4.9.1.1 int main (int *argc*, char * *argv*[])

Definition at line 6 of file frontend.c.

References udhcpc_main(), and udhcpd_main().

```
7 {
8         int ret = 0;
9         char *base = strrchr(argv[0], '/');
10
11         if (strstr(base ? (base + 1) : argv[0], "dhcpd"))
12                 ret = udhcpd_main(argc, argv);
13         else ret = udhcpc_main(argc, argv);
14
15         return ret;
16 }
```

Here is the call graph for this function:



4.9.1.2 int udhcpc_main (int *argc*, char * *argv*[])

Referenced by main().

4.9.1.3 int udhcpd_main (int *argc*, char * *argv*[])

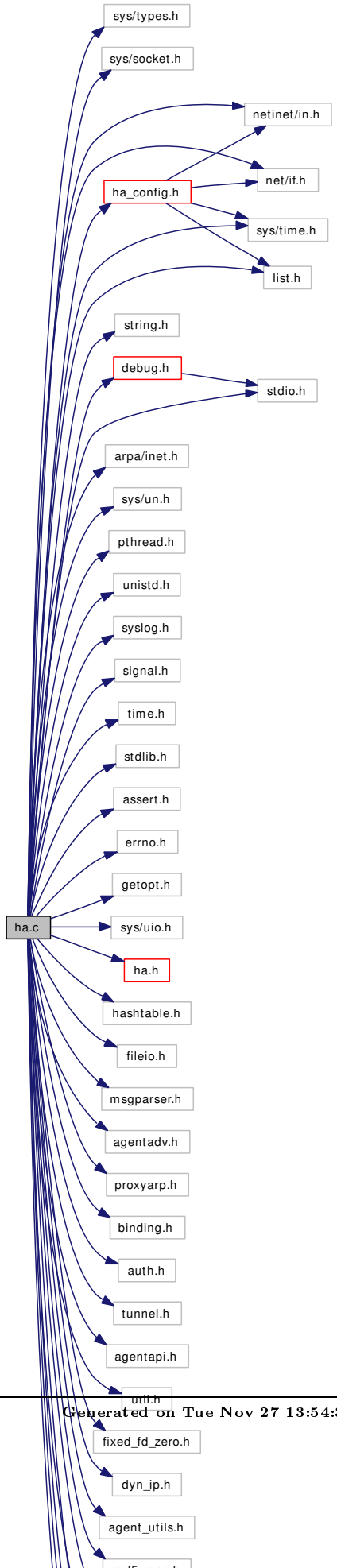
Referenced by main().

4.10 ha.c File Reference

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <net/if.h>
#include <pthread.h>
#include <sys/time.h>
#include <unistd.h>
#include <syslog.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <assert.h>
#include <errno.h>
#include <getopt.h>
#include <sys/uio.h>
#include "ha.h"
#include "list.h"
#include "hashtable.h"
#include "fileio.h"
#include "msgparser.h"
#include "agentadv.h"
#include "proxyarp.h"
#include "binding.h"
#include "auth.h"
#include "tunnel.h"
#include "debug.h"
#include "agentapi.h"
#include "util.h"
#include "fixed_fd_zero.h"
#include "dyn_ip.h"
#include "ha_config.h"
```

```
#include "agent_utils.h"
#include "md5_mac.h"
#include "options.h"
#include "clientpacket.h"
#include "packet.h"
#include "script.h"
#include "socket.h"
#include "pidfile.h"
```

Include dependency graph for ha.c:


sys/types.h
sys/socket.h
netinet/in.h
net/if.h
ha_config.h
sys/time.h
list.h
string.h
debug.h
stdio.h
arpa/inet.h
sys/un.h
pthread.h
unistd.h
syslog.h
signal.h
time.h
stdlib.h
assert.h
errno.h
getopt.h
ha.c
sys/uio.h
ha.h
hashtable.h
fileio.h
msgparser.h
agentadv.h
proxyarp.h
binding.h
auth.h
tunnel.h
agentapi.h
util.h
fixed_fd_zero.h
dyn_ip.h
agent_utils.h

Classes

- struct **dhcp_mobile**

Defines

- #define **DEBUG_FLAG** 'H'
- #define **__USE_BSD**
- #define **DEFAULT_SCRIPT** "/usr/share/udhcpc/default.script"
- #define **LISTEN_NONE** 0
- #define **LISTEN_KERNEL** 1
- #define **LISTEN_RAW** 2
- #define **ASSERT** assert
- #define **MIN**(x, y) ((x) < (y) ? (x) : (y))
- #define **LOG2**(lev, fmt, args...)
- #define **MAX_ADV_DELAY** 200000.0

Functions

- int **find_dhcp_mobile** (unsigned char *nai, int nai_length)
- int **add_dhcp_mobile** (unsigned long ip, unsigned char *nai, int nai_length)
- int **del_dhcp_mobile** (int i)
- unsigned long **udhcp** (int selected_mobile, int flag)

Variables

- **client_config_t client_config**
- int **opt_foreground**
- char * **opt_config**
- **dhcp_mobile dhcp_mobile_array** [HA_DEFAULT_MAX_BINDINGS]
- int **cur_mobiles** = 0
- int **maxmobiles** = HA_DEFAULT_MAX_BINDINGS
- fa_nai_ext * **ha_nai** = NULL

4.10.1 Define Documentation

4.10.1.1 #define __USE_BSD

Definition at line 51 of file ha.c.

4.10.1.2 #define ASSERT assert

Definition at line 138 of file ha.c.

Referenced by load_config().

4.10.1.3 #define DEBUG_FLAG 'H'

Definition at line 34 of file ha.c.

4.10.1.4 #define DEFAULT_SCRIPT "/usr/share/udhcpc/default.script"

Definition at line 88 of file ha.c.

4.10.1.5 #define LISTEN_KERNEL 1

Definition at line 106 of file ha.c.

Referenced by udhcp().

4.10.1.6 #define LISTEN_NONE 0

Definition at line 105 of file ha.c.

Referenced by udhcp().

4.10.1.7 #define LISTEN_RAW 2

Definition at line 107 of file ha.c.

Referenced by udhcp().

4.10.1.8 #define LOG2(lev, fmt, args...)

Value:

```
{ DEBUG(DEBUG_FLAG, fmt, ## args); \
  syslog(lev, fmt, ## args); }
```

Definition at line 141 of file ha.c.

4.10.1.9 #define MAX_ADV_DELAY 200000.0

Definition at line 144 of file ha.c.

4.10.1.10 #define MIN(x, y) ((x) < (y) ? (x) : (y))

Definition at line 139 of file ha.c.

4.10.2 Function Documentation

4.10.2.1 int add_dhcp_mobile (unsigned long *ip*, unsigned char * *nai*, int *nai_length*)

DYNAMO add_dhcp_mobile: : Mobile nodes home address : pointer to NAI : NAIs length

Adds mobile node to the array (, ,).

Returns: 1 on success or 0 on failure

Definition at line 229 of file ha.c.

References cur_mobiles, dhcp_mobile_array, DHCPDISCOVER, maxmobiles, dhcp_mobile::nai, dhcp_mobile::nai_length, dhcp_mobile::requested_ip, dhcp_mobile::state, dhcp_mobile::timeout, and xmalloc.

```
230 {
231         if (cur_mobiles==maxmobiles) return 0;
232         dhcp_mobile_array[cur_mobiles].requested_ip=ip;
233         dhcp_mobile_array[cur_mobiles].timeout=0; //just to be sure
234         dhcp_mobile_array[cur_mobiles].state=DHCPDISCOVER;
235         dhcp_mobile_array[cur_mobiles].nai= xmalloc(nai_length);
236         memcpy(dhcp_mobile_array[cur_mobiles].nai, nai, nai_length);
237         dhcp_mobile_array[cur_mobiles].nai_length=nai_length;
238         cur_mobiles++;
239         return 1;
240 }
```

4.10.2.2 int del_dhcp_mobile (int *i*)

DYNAMO del_dhcp_mobile: : Mobile nodes index on array

Deletes mobile node to the array ().

Returns: 1 on success or 0 on failure

Definition at line 253 of file ha.c.

References cur_mobiles, dhcp_mobile_array, dhcp_mobile::fd, dhcp_mobile::lease, dhcp_mobile::nai_length, dhcp_mobile::packet_num, dhcp_mobile::requested_ip, dhcp_mobile::server_addr, dhcp_mobile::signal_pipe, dhcp_mobile::spi, dhcp_mobile::state, and dhcp_mobile::timeout.

```
254 {
255 if(cur_mobiles <= i || i<0) return 0; //index out of array
256         free(dhcp_mobile_array[i].nai);
257         /*Lets set all to default*/
258   dhcp_mobile_array[i].state=0;
259         dhcp_mobile_array[i].requested_ip=0; /* = 0 */
260         dhcp_mobile_array[i].server_addr=0;
261         dhcp_mobile_array[i].timeout=0;
262         dhcp_mobile_array[i].packet_num=0; /* = 0 */
263         dhcp_mobile_array[i].fd=0;
264         dhcp_mobile_array[i].signal_pipe[0]=0;
265   dhcp_mobile_array[i].signal_pipe[1]=0;
266         dhcp_mobile_array[i].nai_length=0;
267         dhcp_mobile_array[i].lease=0;
268         dhcp_mobile_array[i].spi=0;
269         dhcp_mobile_array[i]=dhcp_mobile_array[cur_mobiles-1];
270         cur_mobiles--;
271   return 1;
272
273 }
```

4.10.2.3 int find_dhcp_mobile (unsigned char * *nai*, int *nai_length*)

DYNAMO find_dhcp_mobile: : pointer to NAI : NAIs length

Finds mobile node from the array (,).

Returns: Array index on success or -1 on failure

Definition at line 194 of file ha.c.

References cur_mobiles, and dhcp_mobile_array.

```
195 {
196 int i, n, found;
197         found=-1;
198         for(i=0;i<cur_mobiles;i++)
199         {
200                 if(nai_length==dhcp_mobile_array[i].nai_length)
201                 {
202                         n=memcmp(nai, dhcp_mobile_array[i].nai,nai_length);
203                         if( n==0 ) //or (memcmp(nai, dhcp_mobile_array[i].nai,nai_length)==0) return i;
204                         {
205                          found=i;
206                          break;
207                         }
208                 }
209
210         }
211
212 return found;
213
214 }
```

4.10.2.4 unsigned long udhcp (int *selected_mobile*, int *flag*)

DYNAMO udhcp: : Selected mobile node's index on array : Flag that tells what to do with selected mobile node

Perfoms wanted DHCP-call for selected mobile node (,).

Returns: IP-address on success or 0 on release or -1 on error.

Definition at line 388 of file ha.c.

References client_config_t::abort_if_no_lease, client_config_t::arp, client_config_t::background_if_no_lease, BOUND, client_config, CLIENT_PORT, client_config_t::clientid, DEBUG, DHCP_CLIENT_ID, DHCP_LEASE_TIME, DHCP_MESSAGE_TYPE, dhcp_mobile_array, DHCP_SERVER_ID, DHCPACK, DHCPNAK, DHCPOFFER, dhcp_mobile::fd, client_config_t::foreground, get_option(), get_packet(), get_raw_packet(), client_config_t::ifindex, INIT_SELECTING, client_config_t::interface, LISTEN_KERNEL, LISTEN_NONE, LISTEN_RAW, listen_socket(), LOG, LOG_ERR, LOG_INFO, dhcp_mobile::nai, dhcp_mobile::nai_length, OPEN_LOG, OPT_CODE, OPT_DATA, OPT_LEN, dhcp_mobile::packet_num, client_config_t::pidfile, pidfile_acquire(), pidfile_write_release(), client_config_t::quit_after_lease, random_xid(), raw_socket(), read_interface(), REBINDING, RELEASED, RELEASEIP, RENEW_REQUESTED, RENEWING, RENEWIP, dhcp_mobile::requested_ip, REQUESTING, REQUESTIP, run_script(), send_discover(), send_renew(), send_selecting(), dhcp_mobile::signal_pipe, dhcp_mobile::state, dhcp_mobile::timeout, dhcpMessage::xid, xmalloc, and dhcpMessage::yiaddr.

```
389 {
390
391         unsigned char *temp, *message;
392         unsigned long t1 = 0, t2 = 0, xid = 0;
393         unsigned long start = 0, lease;
394         fd_set rfds;
395         int retval;
396         struct timeval tv;
397         int c, len;
398         struct dhcpMessage packet;
399         struct in_addr temp_addr;
400         int pid_fd;
```

```
401         time_t now;
402         int max_fd;
403         int sig;
404 
405 
406         if (flag == REQUESTIP)
407         {
408          listen_mode = 0;
409          dhcp_mobile_array[selected_mobile].state=1;
410         }
411 
412         /*TODO Check that really sends DHCPRELEASE*/
413         if (flag == RELEASEIP)
414         {
415         perform_release(selected_mobile);
416         return 0;
417         }
418 
419         if(flag == RENEWIP)
420         {
421         perform_renew(selected_mobile);
422         return dhcp_mobile_array[selected_mobile].requested_ip;
423         }
424 
425         OPEN_LOG("udhcpc");
426 
427         pid_fd = pidfile_acquire(client_config.pidfile);
428         pidfile_write_release(pid_fd);
429         if (read_interface(client_config.interface, &client_config.ifindex,
430                            NULL, client_config.arp) < 0)
431                 exit_client(1); //should never exit in this version!
432 
433         /*
434         DYNAMO:Set NAI into client_id field
435         */
436 
437         int mal_big = dhcp_mobile_array[selected_mobile].nai_length+2;
438         client_config.clientid = xmalloc(mal_big);
439         client_config.clientid[OPT_CODE] = DHCP_CLIENT_ID;
440         client_config.clientid[OPT_LEN] = dhcp_mobile_array[selected_mobile].nai_length;
441         client_config.clientid[OPT_DATA] = 1;
442         memcpy(client_config.clientid + 2, dhcp_mobile_array[selected_mobile].nai, dhcp_mobile_array[selected_mobile].na
443 
444         /* setup signal handlers */
445         socketpair(AF_UNIX, SOCK_STREAM, 0, dhcp_mobile_array[selected_mobile].signal_pipe);
446         signal(SIGUSR1, signal_handler);
447         signal(SIGUSR2, signal_handler);
448         signal(SIGTERM, signal_handler);
449 
450         dhcp_mobile_array[selected_mobile].state = INIT_SELECTING;
451 
452         change_mode(LISTEN_RAW, selected_mobile); //mitÃď tÃďmÃď tekee?
453 
454         for (;;) {
455 
456                 tv.tv_sec = dhcp_mobile_array[selected_mobile].timeout - time(0);
457                 tv.tv_usec = 0;
458                 FD_ZERO(&rfds);
459 
460                 if (listen_mode != LISTEN_NONE && dhcp_mobile_array[selected_mobile].fd < 0) {
461                         if (listen_mode == LISTEN_KERNEL)
462                                 dhcp_mobile_array[selected_mobile].fd = listen_socket(INADDR_ANY, CLIENT_PORT, client_co
463                         else
464                                 dhcp_mobile_array[selected_mobile].fd = raw_socket(client_config.ifindex);
465                         if (dhcp_mobile_array[selected_mobile].fd < 0) {
466                                 LOG(LOG_ERR, "FATAL: couldn't listen on socket, %s", strerror(errno));
467                                 exit_client(0);
```

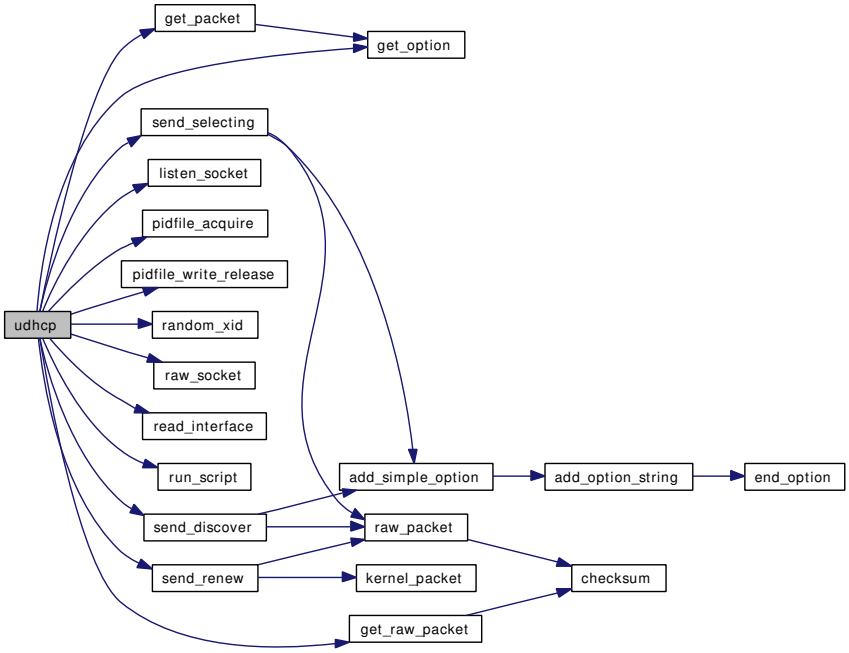
```
468                     }
469             }
470             if (dhcp_mobile_array[selected_mobile].fd >= 0) FD_SET(dhcp_mobile_array[selected_mobile].fd, &rfds);
471             FD_SET(dhcp_mobile_array[selected_mobile].signal_pipe[0], &rfds);
472
473             if (tv.tv_sec > 0) {
474                     DEBUG(LOG_INFO, "Waiting on select...\n");
475                     max_fd = dhcp_mobile_array[selected_mobile].signal_pipe[0] > dhcp_mobile_array[selected_mobile].
476                     retval = select(max_fd + 1, &rfds, NULL, NULL, &tv);
477             } else retval = 0; /* If we already timed out, fall through */
478
479             now = time(0);
480             if (retval == 0) {
481                     /* timeout dropped to zero */
482                     switch (dhcp_mobile_array[selected_mobile].state) {
483                     case INIT_SELECTING:
484                             if (dhcp_mobile_array[selected_mobile].packet_num < 3) {
485                                     if (dhcp_mobile_array[selected_mobile].packet_num == 0)
486                                             xid = random_xid();
487
488                                     /* send discover packet */
489
490                                     send_discover(xid, dhcp_mobile_array[selected_mobile].requested_ip); /* broadcas
491
492                                     dhcp_mobile_array[selected_mobile].timeout = now + ((dhcp_mobile_array[selected_
493                                     dhcp_mobile_array[selected_mobile].packet_num++;
494                             } else {
495                                     if (client_config.background_if_no_lease) {
496                                             LOG(LOG_INFO, "No lease, forking to background.");
497                                             background();
498                                     } else if (client_config.abort_if_no_lease) {
499                                             LOG(LOG_INFO, "No lease, failing.");
500                                             exit_client(1);
501                                     }
502                                     /* wait to try again */
503                                     dhcp_mobile_array[selected_mobile].packet_num = 0;
504                                     dhcp_mobile_array[selected_mobile].timeout = now + 60;
505                             }
506
507                             break;
508                     case RENEW_REQUESTED:
509                     case REQUESTING:
510                             if (dhcp_mobile_array[selected_mobile].packet_num < 3) {
511                                     /* send request packet */
512
513                                     if (dhcp_mobile_array[selected_mobile].state == RENEW_REQUESTED)
514                                             send_renew(xid, dhcp_mobile_array[selected_mobile].server_addr, dhcp_mob
515                                     else send_selecting(xid, dhcp_mobile_array[selected_mobile].server_addr, dhcp_mo
516
517                                     dhcp_mobile_array[selected_mobile].timeout = now + ((dhcp_mobile_array[selected_
518                                     dhcp_mobile_array[selected_mobile].packet_num++;
519                             } else {
520
521                                     /* timed out, go back to init state */
522                                     if (dhcp_mobile_array[selected_mobile].state == RENEW_REQUESTED) run_script(NULL
523                                     dhcp_mobile_array[selected_mobile].state = INIT_SELECTING;
524                                     dhcp_mobile_array[selected_mobile].timeout = now;
525                                     dhcp_mobile_array[selected_mobile].packet_num = 0;
526                                     change_mode(LISTEN_RAW, selected_mobile);
527                             }
528                             break;
529                     case BOUND:
530                             /* Lease is starting to run out, time to enter renewing state */
531                             dhcp_mobile_array[selected_mobile].state = RENEWING; //tï£¡hï£¡n muutoksia??!
532                             change_mode(LISTEN_KERNEL, selected_mobile);
533
534                             DEBUG(LOG_INFO, "Entering renew state");
```

```
535                             /* fall right through */
536                     case RENEWING:
537                             /* Either set a new T1, or enter REBINDING state */
538                             if ((t2 - t1) <= (lease / 14400 + 1)) {
539                                     /* timed out, enter rebinding state */
540                                     dhcp_mobile_array[selected_mobile].state = REBINDING;
541                                     dhcp_mobile_array[selected_mobile].timeout = now + (t2 - t1);
542                                     DEBUG(LOG_INFO, "Entering rebinding state");
543                             } else {
544                                     /* send a request packet */
545                                     send_renew(xid, dhcp_mobile_array[selected_mobile].server_addr, dhcp_mobile_arra
546 
547                                     t1 = (t2 - t1) / 2 + t1;
548                                     dhcp_mobile_array[selected_mobile].timeout = t1 + start;
549                             }
550                             break;
551                     case REBINDING:
552                             /* Either set a new T2, or enter INIT state */
553                             if ((lease - t2) <= (lease / 14400 + 1)) {
554                                     /* timed out, enter init state */
555                                     dhcp_mobile_array[selected_mobile].state = INIT_SELECTING;
556                                     LOG(LOG_INFO, "Lease lost, entering init state");
557                                     run_script(NULL, "deconfig");
558                                     dhcp_mobile_array[selected_mobile].timeout = now;
559                                     dhcp_mobile_array[selected_mobile].packet_num = 0;
560                                     change_mode(LISTEN_RAW, selected_mobile);
561                             } else {
562                                     /* send a request packet */
563                                     send_renew(xid, 0, dhcp_mobile_array[selected_mobile].requested_ip); /* broadcas
564 
565                                     t2 = (lease - t2) / 2 + t2;
566                                     dhcp_mobile_array[selected_mobile].timeout = t2 + start;
567                             }
568                             break;
569                     case RELEASED:
570                             /* yah, I know, *you* say it would never happen */
571                             dhcp_mobile_array[selected_mobile].timeout = 0;
572                             break;
573                     }
574 
575 
576             } else if (retval > 0 && listen_mode != LISTEN_NONE && FD_ISSET(dhcp_mobile_array[selected_mobile].fd, &
577                     /* a packet is ready, read it */
578 
579                     if (listen_mode == LISTEN_KERNEL)
580                             len = get_packet(&packet, dhcp_mobile_array[selected_mobile].fd);
581                     else len = get_raw_packet(&packet, dhcp_mobile_array[selected_mobile].fd);
582 
583                     if (len == -1 && errno != EINTR) {
584                             DEBUG(LOG_INFO, "error on read, %s, reopening socket", strerror(errno));
585                             change_mode(listen_mode, selected_mobile); /* just close and reopen */
586                     }
587                     if (len < 0) continue;
588 
589 
590                     if (packet.xid != xid) {
591                             DEBUG(LOG_INFO, "Ignoring XID %lx (our xid is %lx)",
592                                     (unsigned long) packet.xid, xid);
593                             continue;
594                     }
595 
596                     if ((message = get_option(&packet, DHCP_MESSAGE_TYPE)) == NULL) {
597                             DEBUG(LOG_ERR, "couldnt get option from packet -- ignoring");
598                             continue;
599                     }
600 
601                     switch (dhcp_mobile_array[selected_mobile].state) {
```

```
602                         case INIT_SELECTING:
603                                 /* Must be a DHCPOFFER to one of our xid's */
604                                 if (*message == DHCPOFFER) {
605                                         if ((temp = get_option(&packet, DHCP_SERVER_ID))) {
606                                                 memcpy(&dhcp_mobile_array[selected_mobile].server_addr, temp, 4);
607                                                 xid = packet.xid;
608                                                 dhcp_mobile_array[selected_mobile].requested_ip = packet.yiaddr; //set g
609 
610                                                 /* enter requesting state */
611                                                 dhcp_mobile_array[selected_mobile].state = REQUESTING;
612                                                 dhcp_mobile_array[selected_mobile].timeout = now;
613                                                 dhcp_mobile_array[selected_mobile].packet_num = 0;
614                                         } else {
615                                                 DEBUG(LOG_ERR, "No server ID in message");
616                                         }
617                                 }
618                                 break;
619                         case RENEW_REQUESTED:
620                         case REQUESTING:
621                         case RENEWING:
622                         case REBINDING:
623                                 if (*message == DHCPACK) {
624                                         if (!(temp = get_option(&packet, DHCP_LEASE_TIME))) {
625                                                 LOG(LOG_ERR, "No lease time with ACK, using 1 hour lease");
626                                                 lease = 60 * 60;
627                                         } else {
628                                                 memcpy(&lease, temp, 4);
629                                                 lease = ntohl(lease);
630                                         }
631 
632                                         /* enter bound state */
633                                         t1 = lease / 2;
634 
635                                         /* little fixed point for n * .875 */
636                                         t2 = (lease * 0x7) >> 3;
637                                         temp_addr.s_addr = packet.yiaddr;
638                                         LOG(LOG_INFO, "Lease of %s obtained, lease time %ld",
639                                                 inet_ntoa(temp_addr), lease);
640                                         start = now;
641                                         dhcp_mobile_array[selected_mobile].timeout = t1 + start;
642                                         dhcp_mobile_array[selected_mobile].requested_ip = packet.yiaddr;
643                                         run_script(&packet,
644                                             ((dhcp_mobile_array[selected_mobile].state == RENEWING || dhcp_mobile
645 
646                                         dhcp_mobile_array[selected_mobile].state = BOUND;
647                                         change_mode(LISTEN_NONE, selected_mobile);
648                                         if (client_config.quit_after_lease){
649                                                 return dhcp_mobile_array[selected_mobile].requested_ip;
650                                                 }
651                                         if (!client_config.foreground){
652 
653                                                 return dhcp_mobile_array[selected_mobile].requested_ip;
654                                                 }
655 
656                                 } else if (*message == DHCPNAK) {
657                                         /* return to init state */
658                                         LOG(LOG_INFO, "Received DHCP NAK");
659                                         run_script(&packet, "nak");
660                                         if (dhcp_mobile_array[selected_mobile].state != REQUESTING)
661                                                 run_script(NULL, "deconfig");
662                                         dhcp_mobile_array[selected_mobile].state = INIT_SELECTING;
663                                         dhcp_mobile_array[selected_mobile].timeout = now;
664                                         dhcp_mobile_array[selected_mobile].requested_ip = 0;
665                                         dhcp_mobile_array[selected_mobile].packet_num = 0;
666                                         change_mode(LISTEN_RAW, selected_mobile);
667                                         sleep(3); /* avoid excessive network traffic */
668                                 }
```

```
669                             break;
670                     /* case BOUND, RELEASED: - ignore all packets */
671                     }
672             } else if (retval > 0 && FD_ISSET(dhcp_mobile_array[selected_mobile].signal_pipe[0], &rfds)) {
673                     if (read(dhcp_mobile_array[selected_mobile].signal_pipe[0], &sig, sizeof(signal)) < 0) {
674                             DEBUG(LOG_ERR, "Could not read signal: %s",
675                                     strerror(errno));
676                             continue; /* probably just EINTR */
677                     }
678                     switch (sig) {
679                     case SIGUSR1:
680                             perform_renew(selected_mobile);
681                             break;
682                     case SIGUSR2:
683                             perform_release(selected_mobile);
684                             break;
685                     case SIGTERM:
686                             LOG(LOG_INFO, "Received SIGTERM");
687                             exit_client(0);
688                     }
689             } else if (retval == -1 && errno == EINTR) {
690                     /* a signal was caught */
691             } else {
692                     /* An error occured */
693                     DEBUG(LOG_ERR, "Error on select");
694             }
695
696     }
697
698     return 0;
699 }
```

Here is the call graph for this function:



4.10.3 Variable Documentation

4.10.3.1 struct client_config_t client_config

Initial value:

```
 {

        abort_if_no_lease: 0,
        foreground: 0,
        quit_after_lease: 0,
        background_if_no_lease: 0,
        interface: "eth2",
        pidfile: NULL,
        script: DEFAULT_SCRIPT,
        clientid: NULL,
        hostname: NULL,
        ifindex: 0,
        arp: "\0\0\0\0\0\0",
}
```

Definition at line 90 of file ha.c.

Referenced by run_script(), send_discover(), send_renew(), send_selecting(), and udhcp().

4.10.3.2 int cur_mobiles = 0

Definition at line 131 of file ha.c.

Referenced by add_dhcp_mobile(), del_dhcp_mobile(), and find_dhcp_mobile().

4.10.3.3 struct dhcp_mobile dhcp_mobile_array[HA_DEFAULT_MAX_BINDINGS]

Definition at line 129 of file ha.c.

Referenced by add_dhcp_mobile(), del_dhcp_mobile(), find_dhcp_mobile(), and udhcp().

4.10.3.4 struct fa_nai_ext* ha_nai = NULL

Definition at line 166 of file ha.c.

4.10.3.5 int maxmobiles = HA_DEFAULT_MAX_BINDINGS

Definition at line 132 of file ha.c.

Referenced by add_dhcp_mobile().

4.10.3.6 char* opt_config

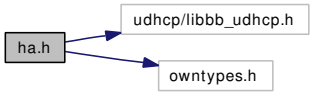
4.10.3.7 int opt_foreground

4.11 ha.h File Reference

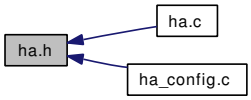
```
#include "udhcp/libbb_udhcp.h"
```

```
#include "owntypes.h"
```

Include dependency graph for ha.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct **ha_tunnel_data**
- struct **client_config_t**

Defines

- #define **INIT_SELECTING** 0
- #define **REQUESTING** 1
- #define **BOUND** 2
- #define **RENEWING** 3
- #define **REBINDING** 4
- #define **INIT_REBOOT** 5
- #define **RENEW_REQUESTED** 6
- #define **RELEASED** 7
- #define **REQUESTIP** 0
- #define **RENEWIP** 1
- #define **RELEASEIP** 2
- #define **LEASE_TIME** (60*60*24*10)
- #define **DHCPD_CONF_FILE** "/etc/udhcpd.conf"
- #define **SERVER_PORT** 67
- #define **CLIENT_PORT** 68
- #define **DHCP_MAGIC** 0x63825363
- #define **DHCP_PADDING** 0x00
- #define **DHCP_SUBNET** 0x01
- #define **DHCP_TIME_OFFSET** 0x02
- #define **DHCP_ROUTER** 0x03
- #define **DHCP_TIME_SERVER** 0x04
- #define **DHCP_NAME_SERVER** 0x05
- #define **DHCP_DNS_SERVER** 0x06
- #define **DHCP_LOG_SERVER** 0x07

- #define **DHCP_COOKIE_SERVER** 0x08
- #define **DHCP_LPR_SERVER** 0x09
- #define **DHCP_HOST_NAME** 0x0c
- #define **DHCP_BOOT_SIZE** 0x0d
- #define **DHCP_DOMAIN_NAME** 0x0f
- #define **DHCP_SWAP_SERVER** 0x10
- #define **DHCP_ROOT_PATH** 0x11
- #define **DHCP_IP_TTL** 0x17
- #define **DHCP_MTU** 0x1a
- #define **DHCP_BROADCAST** 0x1c
- #define **DHCP_NTP_SERVER** 0x2a
- #define **DHCP_WINS_SERVER** 0x2c
- #define **DHCP_REQUESTED_IP** 0x32
- #define **DHCP_LEASE_TIME** 0x33
- #define **DHCP_OPTION_OVER** 0x34
- #define **DHCP_MESSAGE_TYPE** 0x35
- #define **DHCP_SERVER_ID** 0x36
- #define **DHCP_PARAM_REQ** 0x37
- #define **DHCP_MESSAGE** 0x38
- #define **DHCP_MAX_SIZE** 0x39
- #define **DHCP_T1** 0x3a
- #define **DHCP_T2** 0x3b
- #define **DHCP_VENDOR** 0x3c
- #define **DHCP_CLIENT_ID** 0x3d
- #define **DHCP_END** 0xFF
- #define **BOOTREQUEST** 1
- #define **BOOTREPLY** 2
- #define **ETH_10MB** 1
- #define **ETH_10MB_LEN** 6
- #define **DHCPDISCOVER** 1
- #define **DHCPOFFER** 2
- #define **DHCPREQUEST** 3
- #define **DHCPDECLINE** 4
- #define **DHCPACK** 5
- #define **DHCPNAK** 6
- #define **DHCPRELEASE** 7
- #define **DHCPINFORM** 8
- #define **BROADCAST_FLAG** 0x8000
- #define **OPTION_FIELD** 0
- #define **FILE_FIELD** 1
- #define **SNAME_FIELD** 2
- #define **MAC_BCAST_ADDR** (unsigned char *) "\xff\xff\xff\xff\xff\xff"
- #define **OPT_CODE** 0
- #define **OPT_LEN** 1
- #define **OPT_DATA** 2
- #define **HA_CONF_FILE** "dynhad.conf"
- #define **HA_GLOBAL_CONF_FILE** SYSCONFDIR "/" HA_CONF_FILE
- #define **HA_PID_FILE** PIDDIR "/dynhad.pid"

Enumerations

- enum { **ENCAPS_IPIP**, **ENCAPS_MINIMAL**, **ENCAPS_GRE** }
- enum { **AUTH_RFC2002**, **AUTH_RFC2002BIS** }

Variables

- **client_config_t client_config**

4.11.1 Define Documentation

4.11.1.1 #define BOOTREPLY 2

Definition at line 100 of file ha.h.

Referenced by init_header().

4.11.1.2 #define BOOTREQUEST 1

Definition at line 99 of file ha.h.

Referenced by get_packet(), and init_header().

4.11.1.3 #define BOUND 2

Definition at line 28 of file ha.h.

Referenced by udhcp().

4.11.1.4 #define BROADCAST_FLAG 0x8000

Definition at line 114 of file ha.h.

Referenced by get_packet().

4.11.1.5 #define CLIENT_PORT 68

Definition at line 58 of file ha.h.

Referenced by get_raw_packet(), send_discover(), send_release(), send_renew(), send_selecting(), and udhcp().

4.11.1.6 #define DHCP_BOOT_SIZE 0x0d

Definition at line 74 of file ha.h.

4.11.1.7 #define DHCP_BROADCAST 0x1c

Definition at line 80 of file ha.h.

4.11.1.8 #define DHCP_CLIENT_ID 0x3d

Definition at line 94 of file ha.h.

Referenced by udhcp().

4.11.1.9 #define DHCP_COOKIE_SERVER 0x08

Definition at line 71 of file ha.h.

4.11.1.10 #define DHCP_DNS_SERVER 0x06

Definition at line 69 of file ha.h.

4.11.1.11 #define DHCP_DOMAIN_NAME 0x0f

Definition at line 75 of file ha.h.

4.11.1.12 #define DHCP_END 0xFF

Definition at line 96 of file ha.h.

Referenced by add_option_string(), end_option(), get_option(), and init_header().

4.11.1.13 #define DHCP_HOST_NAME 0x0c

Definition at line 73 of file ha.h.

4.11.1.14 #define DHCP_IP_TTL 0x17

Definition at line 78 of file ha.h.

4.11.1.15 #define DHCP_LEASE_TIME 0x33

Definition at line 84 of file ha.h.

Referenced by send_inform(), sendACK(), sendOffer(), and udhcp().

4.11.1.16 #define DHCP_LOG_SERVER 0x07

Definition at line 70 of file ha.h.

4.11.1.17 #define DHCP_LPR_SERVER 0x09

Definition at line 72 of file ha.h.

4.11.1.18 #define DHCP_MAGIC 0x63825363

Definition at line 60 of file ha.h.

Referenced by get_packet(), get_raw_packet(), and init_header().

4.11.1.19 #define DHCP_MAX_SIZE 0x39

Definition at line 90 of file ha.h.

4.11.1.20 #define DHCP_MESSAGE 0x38

Definition at line 89 of file ha.h.

4.11.1.21 #define DHCP_MESSAGE_TYPE 0x35

Definition at line 86 of file ha.h.

Referenced by init_header(), and udhcp().

4.11.1.22 #define DHCP_MTU 0x1a

Definition at line 79 of file ha.h.

4.11.1.23 #define DHCP_NAME_SERVER 0x05

Definition at line 68 of file ha.h.

4.11.1.24 #define DHCP_NTP_SERVER 0x2a

Definition at line 81 of file ha.h.

4.11.1.25 #define DHCP_OPTION_OVER 0x34

Definition at line 85 of file ha.h.

Referenced by get_option().

4.11.1.26 #define DHCP_PADDING 0x00

Definition at line 63 of file ha.h.

Referenced by end_option(), and get_option().

4.11.1.27 #define DHCP_PARAM_REQ 0x37

Definition at line 88 of file ha.h.

4.11.1.28 #define DHCP_REQUESTED_IP 0x32

Definition at line 83 of file ha.h.

Referenced by send_discover(), send_release(), send_selecting(), and sendOffer().

4.11.1.29 #define DHCP_ROOT_PATH 0x11

Definition at line 77 of file ha.h.

4.11.1.30 #define DHCP_ROUTER 0x03

Definition at line 66 of file ha.h.

4.11.1.31 #define DHCP_SERVER_ID 0x36

Definition at line 87 of file ha.h.

Referenced by send_release(), send_selecting(), and udhcp().

4.11.1.32 #define DHCP_SUBNET 0x01

Definition at line 64 of file ha.h.

4.11.1.33 #define DHCP_SWAP_SERVER 0x10

Definition at line 76 of file ha.h.

4.11.1.34 #define DHCP_T1 0x3a

Definition at line 91 of file ha.h.

4.11.1.35 #define DHCP_T2 0x3b

Definition at line 92 of file ha.h.

4.11.1.36 #define DHCP_TIME_OFFSET 0x02

Definition at line 65 of file ha.h.

4.11.1.37 #define DHCP_TIME_SERVER 0x04

Definition at line 67 of file ha.h.

4.11.1.38 #define DHCP_VENDOR 0x3c

Definition at line 93 of file ha.h.

Referenced by get_packet().

4.11.1.39 #define DHCP_WINS_SERVER 0x2c

Definition at line 82 of file ha.h.

4.11.1.40 #define DHCPACK 5

Definition at line 109 of file ha.h.

Referenced by init_header(), send_inform(), sendACK(), and udhcp().

4.11.1.41 #define DHCPD_CONF_FILE "/etc/udhcpd.conf"

Definition at line 50 of file ha.h.

4.11.1.42 #define DHCPDECLINE 4

Definition at line 108 of file ha.h.

4.11.1.43 #define DHCPDISCOVER 1

Definition at line 105 of file ha.h.

Referenced by add_dhcp_mobile(), init_header(), and send_discover().

4.11.1.44 #define DHCPINFORM 8

Definition at line 112 of file ha.h.

Referenced by init_header().

4.11.1.45 #define DHCPNAK 6

Definition at line 110 of file ha.h.

Referenced by init_header(), sendNAK(), and udhcp().

4.11.1.46 #define DHCPOFFER 2

Definition at line 106 of file ha.h.

Referenced by init_header(), sendOffer(), and udhcp().

4.11.1.47 #define DHCPRELEASE 7

Definition at line 111 of file ha.h.

Referenced by init_header(), and send_release().

4.11.1.48 #define DHCPREQUEST 3

Definition at line 107 of file ha.h.

Referenced by init_header(), send_renew(), and send_selecting().

4.11.1.49 #define ETH_10MB 1

Definition at line 102 of file ha.h.

Referenced by init_header().

4.11.1.50 #define ETH_10MB_LEN 6

Definition at line 103 of file ha.h.

Referenced by init_header().

4.11.1.51 #define FILE_FIELD 1

Definition at line 117 of file ha.h.

Referenced by get_option().

4.11.1.52 #define HA_CONF_FILE "dynhad.conf"

Definition at line 129 of file ha.h.

4.11.1.53 #define HA_GLOBAL_CONF_FILE SYSCONFDIR "/" HA_CONF_FILE

Definition at line 130 of file ha.h.

4.11.1.54 #define HA_PID_FILE PIDDIR "/dynhad.pid"

Definition at line 134 of file ha.h.

4.11.1.55 #define INIT_REBOOT 5

Definition at line 31 of file ha.h.

4.11.1.56 #define INIT_SELECTING 0

Definition at line 26 of file ha.h.

Referenced by udhcp().

4.11.1.57 #define LEASE_TIME (60*60*24*10)

Definition at line 47 of file ha.h.

4.11.1.58 #define MAC_BCAST_ADDR (unsigned char *) "\xff\xff\xff\xff\xff\xff"

Definition at line 121 of file ha.h.

Referenced by arpping(), send_discover(), send_renew(), and send_selecting().

4.11.1.59 #define OPT_CODE 0

Definition at line 122 of file ha.h.

Referenced by add_option_string(), add_simple_option(), attach_option(), find_option(), get_option(), send_inform(), sendACK(), sendOffer(), and udhcp().

4.11.1.60 #define OPT_DATA 2

Definition at line 124 of file ha.h.

Referenced by udhcp().

4.11.1.61 #define OPT_LEN 1

Definition at line 123 of file ha.h.

Referenced by add_option_string(), add_simple_option(), attach_option(), end_option(), get_option(), get_packet(), and udhcp().

4.11.1.62 #define OPTION_FIELD 0

Definition at line 116 of file ha.h.

Referenced by get_option().

4.11.1.63 #define REBINDING 4

Definition at line 30 of file ha.h.

Referenced by udhcp().

4.11.1.64 #define RELEASED 7

Definition at line 33 of file ha.h.

Referenced by udhcp().

4.11.1.65 #define RELEASEIP 2

Definition at line 37 of file ha.h.

Referenced by udhcp().

4.11.1.66 #define RENEW_REQUESTED 6

Definition at line 32 of file ha.h.

Referenced by udhcp().

4.11.1.67 #define RENEWING 3

Definition at line 29 of file ha.h.

Referenced by udhcp().

4.11.1.68 #define RENEWIP 1

Definition at line 36 of file ha.h.

Referenced by udhcp().

4.11.1.69 #define REQUESTING 1

Definition at line 27 of file ha.h.

Referenced by udhcp().

4.11.1.70 #define REQUESTIP 0

Definition at line 35 of file ha.h.

Referenced by udhcp().

4.11.1.71 #define SERVER_PORT 67

Definition at line 57 of file ha.h.

Referenced by send_discover(), send_release(), send_renew(), and send_selecting().

4.11.1.72 #define SNAME_FIELD 2

Definition at line 118 of file ha.h.

Referenced by get_option().

4.11.2 Enumeration Type Documentation

4.11.2.1 anonymous enum

Enumerator:

ENCAPS_IPIP
ENCAPS_MINIMAL
ENCAPS_GRE

Definition at line 136 of file ha.h.

```
136 { ENCAPS_IPIP, ENCAPS_MINIMAL, ENCAPS_GRE };
```

4.11.2.2 anonymous enum

Enumerator:

AUTH_RFC2002
AUTH_RFC2002BIS

Definition at line 137 of file ha.h.

```
137 { AUTH_RFC2002, AUTH_RFC2002BIS };
```

4.11.3 Variable Documentation

4.11.3.1 struct client_config_t client_config

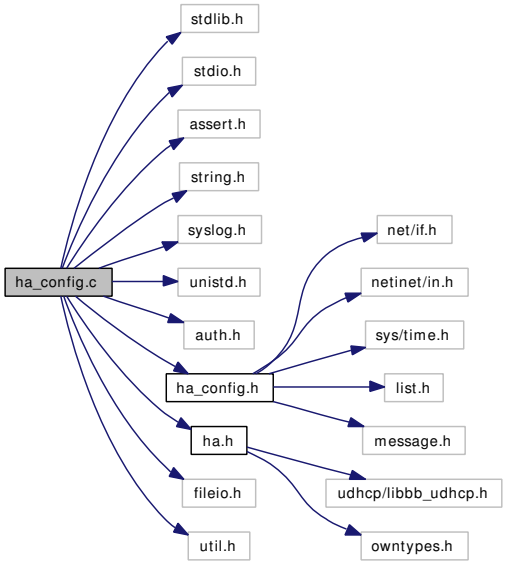
Definition at line 90 of file ha.c.

Referenced by run_script(), send_discover(), send_renew(), send_selecting(), and udhcp().

4.12 ha_config.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <syslog.h>
#include <unistd.h>
#include "auth.h"
#include "ha_config.h"
#include "ha.h"
#include "fileio.h"
#include "util.h"
```

Include dependency graph for ha_config.c:



Classes

- struct **load_ha_data**

Defines

- #define **ASSERT** assert

Functions

- void **cleanup_config** (struct **ha_config** *cfg)

- int **load_config** (struct **ha_config** *cfg, char *program_name, char *config_file)

4.12.1 Define Documentation

4.12.1.1 #define ASSERT assert

Definition at line 26 of file ha_config.c.

4.12.2 Function Documentation

4.12.2.1 void cleanup_config (struct ha_config * *cfg*)

Definition at line 47 of file ha_config.c.

References ha_config::authorized_list, ha_config::fa_spi_list, interface_entry::icmp_sock, ha_config::interfaces, spi_entry::spi, ha_config::spi_list, interface_entry::udp_bc_sock, interface_entry::udp_bc_sock2, and interface_entry::udp_sock.

Referenced by load_config().

```
48 {
49         struct spi_entry *spi;
50         struct fa_spi_entry *fa_spi;
51         struct authorized_entry *auth;
52         struct interface_entry *iface;
53 
54         if (cfg == NULL) return;
55 
56         spi = (struct spi_entry *) list_remove_first(&cfg->spi_list);
57         while (spi != NULL) {
58                 free(spi);
59                 spi = (struct spi_entry *) list_remove_first(&cfg->spi_list);
60         }
61 
62         /* remove all from authorized list */
63         auth = (struct authorized_entry *)
64                 list_remove_first(&cfg->authorized_list);
65         while (auth != NULL) {
66                 free(auth);
67                 auth = (struct authorized_entry *)
68                         list_remove_first(&cfg->authorized_list);
69         }
70 
71         fa_spi = (struct fa_spi_entry *) list_remove_first(&cfg->fa_spi_list);
72         while (fa_spi != NULL) {
73                 free(fa_spi);
74                 fa_spi = (struct fa_spi_entry *)
75                         list_remove_first(&cfg->fa_spi_list);
76         }
77 
78         iface = (struct interface_entry *)
79                 list_remove_first(&cfg->interfaces);
80         while (iface != NULL) {
81                 if (iface->icmp_sock >= 0)
82                         close(iface->icmp_sock);
83                 if (iface->udp_sock >= 0)
84                         close(iface->udp_sock);
85                 if (iface->udp_bc_sock >= 0)
86                         close(iface->udp_bc_sock);
87                 if (iface->udp_bc_sock2 >= 0)
88                         close(iface->udp_bc_sock2);
89                 free(iface);
```

```
90                 iface = (struct interface_entry *)
91                         list_remove_first(&cfg->interfaces);
92         }
93 }
```

4.12.2.2 int load_config (struct ha_config * *cfg*, char * *program_name*, char * *config_file*)

Definition at line 97 of file ha_config.c.

References ASSERT, ha_config::authorized_list, load_ha_data::cfg, cleanup_config(), ha_config::enable_reverse_tunneling, ha_config::enable_triangle_tunneling, ha_config::fa_spi_list, FALSE, HA_DEFAULT_MAX_BINDINGS, HA_DEFAULT_REG_ERROR_REPLY_INTERVAL, HA_DEFAULT_REG_PORT, HA_DEFAULT_SYSLOG_FACILITY, ha_config::ha_default_tunnel_lifetime, HA_DEFAULT_TUNNEL_LIFETIME, HASH_METHOD_CHECK, ha_config::interfaces, ha_config::max_bindings, load_ha_data::process_authorized_list, load_ha_data::process_fa_spi_list, load_ha_data::process_interfaces, load_ha_data::process_spi_list, ha_config::pubkey_hash_method, ha_config::reg_error_reply_interval, ha_config::socket_priority, ha_config::spi_list, ha_config::syslog_facility, TRUE, and ha_config::udpport.

```
98 {
99         FILE *file;
100         struct load_ha_data ha;
101
102         ASSERT(cfg);
103         ha.cfg = cfg;
104         memset(cfg, 0, sizeof(struct ha_config));
105         ha.process_spi_list = FALSE;
106         list_init(&cfg->spi_list);
107         ha.process_authorized_list = FALSE;
108         list_init(&cfg->authorized_list);
109         ha.process_fa_spi_list = FALSE;
110         list_init(&cfg->fa_spi_list);
111         ha.process_interfaces = FALSE;
112         list_init(&cfg->interfaces);
113
114         /* set default values */
115         cfg->max_bindings = HA_DEFAULT_MAX_BINDINGS;
116         cfg->ha_default_tunnel_lifetime = HA_DEFAULT_TUNNEL_LIFETIME;
117         cfg->reg_error_reply_interval = HA_DEFAULT_REG_ERROR_REPLY_INTERVAL;
118         cfg->syslog_facility = HA_DEFAULT_SYSLOG_FACILITY;
119         cfg->udpport = HA_DEFAULT_REG_PORT;
120         cfg->socket_priority = -1;
121         cfg->enable_triangle_tunneling = TRUE;
122         cfg->enable_reverse_tunneling = TRUE;
123         cfg->pubkey_hash_method = HASH_METHOD_CHECK;
124
125         file = fopen(config_file, "r");
126         if (file == NULL) {
127                 fprintf(stderr,
128                         "%s: Could not open configuration file '%s'.\n",
129                         program_name, config_file);
130                 return FALSE;
131         }
132         if (load_data(&ha, file, process_load_ha) == FALSE) {
133                 fprintf(stderr,
134                         "%s: Error while interpreting file '%s'!\n",
135                         program_name, config_file);
136                 fclose(file);
137                 cleanup_config(cfg);
138                 return FALSE;
```

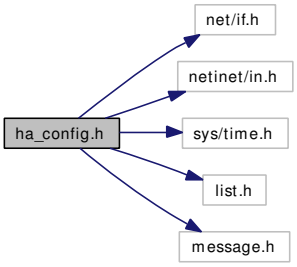
```
139         }
140         fclose(file);
141
142         openlog("home agent", LOG_PID | LOG_CONS, cfg->syslog_facility);
143
144         return TRUE;
145 }
```

Here is the call graph for this function:

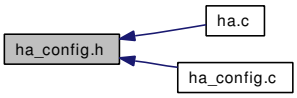
4.13 ha_config.h File Reference

```
#include <net/if.h>
#include <netinet/in.h>
#include <sys/time.h>
#include "list.h"
#include "message.h"
```

Include dependency graph for ha_config.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct **interface_entry**
- struct **ha_config**
- struct **spi_entry**
- struct **authorized_entry**
- struct **fa_spi_entry**

Defines

- #define **MAXFILENAMELEN** 256
- #define **MAXSHAREDSECRETLEN** 32
- #define **MAXOWNERNAMELEN** 8
- #define **MAXGROUPNAMELEN** 8
- #define **HA_DEFAULT_MAX_BINDINGS** 20
- #define **HA_DEFAULT_TUNNEL_LIFETIME** 500
- #define **HA_DEFAULT_REG_ERROR_REPLY_INTERVAL** 10
- #define **HA_DEFAULT_SYSLOG_FACILITY** LOG_LOCAL0
- #define **HA_DEFAULT_REG_PORT** 434
- #define **HA_MOBILE_HASHTABLE_SIZE** 256
- #define **MAXMSG** 2048

- #define **HASH_METHOD_NONE** 0
- #define **HASH_METHOD_CHECK** 1
- #define **HASH_METHOD_REQUIRE** 2

Enumerations

- enum { **INTERFACE_AGENTADV_ONLY_SOLICITED** = 0, **INTERFACE_AGENTADV_ALL** = 1, **INTERFACE_AGENTADV_NONE** = -1 }

Functions

- int **load_config** (struct **ha_config** *ha, char *program_name, char *config_file)
- void **cleanup_config** (struct **ha_config** *cfg)

4.13.1 Define Documentation

4.13.1.1 #define HA_DEFAULT_MAX_BINDINGS 20

Definition at line 27 of file ha_config.h.

Referenced by load_config().

4.13.1.2 #define HA_DEFAULT_REG_ERROR_REPLY_INTERVAL 10

Definition at line 29 of file ha_config.h.

Referenced by load_config().

4.13.1.3 #define HA_DEFAULT_REG_PORT 434

Definition at line 33 of file ha_config.h.

Referenced by load_config().

4.13.1.4 #define HA_DEFAULT_SYSLOG_FACILITY LOG_LOCAL0

Definition at line 30 of file ha_config.h.

Referenced by load_config().

4.13.1.5 #define HA_DEFAULT_TUNNEL_LIFETIME 500

Definition at line 28 of file ha_config.h.

Referenced by load_config().

4.13.1.6 #define HA_MOBILE_HASHTABLE_SIZE 256

Definition at line 35 of file ha_config.h.

4.13.1.7 #define HASH_METHOD_CHECK 1

Definition at line 41 of file ha_config.h.

Referenced by load_config().

4.13.1.8 #define HASH_METHOD_NONE 0

Definition at line 40 of file ha_config.h.

4.13.1.9 #define HASH_METHOD_REQUIRE 2

Definition at line 42 of file ha_config.h.

4.13.1.10 #define MAXFILENAMELEN 256

Definition at line 22 of file ha_config.h.

4.13.1.11 #define MAXGROUPNAMELEN 8

Definition at line 25 of file ha_config.h.

4.13.1.12 #define MAXMSG 2048

Definition at line 37 of file ha_config.h.

4.13.1.13 #define MAXOWNERNAMELEN 8

Definition at line 24 of file ha_config.h.

4.13.1.14 #define MAXSHAREDSECRETLEN 32

Definition at line 23 of file ha_config.h.

4.13.2 Enumeration Type Documentation

4.13.2.1 anonymous enum

Enumerator:

INTERFACE_AGENTADV_ONLY_SOLICITED
INTERFACE_AGENTADV_ALL
INTERFACE_AGENTADV_NONE

Definition at line 44 of file ha_config.h.

```
44     {
45        INTERFACE_AGENTADV_ONLY_SOLICITED = 0,
46        INTERFACE_AGENTADV_ALL = 1,
47        INTERFACE_AGENTADV_NONE = -1
48 };
```

4.13.3 Function Documentation

4.13.3.1 void cleanup_config (struct ha_config * *cfg*)

Definition at line 47 of file ha_config.c.

References ha_config::authorized_list, ha_config::fa_spi_list, interface_entry::icmp_sock, ha_config::interfaces, spi_entry::spi, ha_config::spi_list, interface_entry::udp_bc_sock, interface_entry::udp_bc_sock2, and interface_entry::udp_sock.

Referenced by load_config().

```
48 {
49         struct spi_entry *spi;
50         struct fa_spi_entry *fa_spi;
51         struct authorized_entry *auth;
52         struct interface_entry *iface;
53
54         if (cfg == NULL) return;
55
56         spi = (struct spi_entry *) list_remove_first(&cfg->spi_list);
57         while (spi != NULL) {
58                 free(spi);
59                 spi = (struct spi_entry *) list_remove_first(&cfg->spi_list);
60         }
61
62         /* remove all from authorized list */
63         auth = (struct authorized_entry *)
64                 list_remove_first(&cfg->authorized_list);
65         while (auth != NULL) {
66                 free(auth);
67                 auth = (struct authorized_entry *)
68                         list_remove_first(&cfg->authorized_list);
69         }
70
71         fa_spi = (struct fa_spi_entry *) list_remove_first(&cfg->fa_spi_list);
72         while (fa_spi != NULL) {
73                 free(fa_spi);
74                 fa_spi = (struct fa_spi_entry *)
75                         list_remove_first(&cfg->fa_spi_list);
76         }
77
78         iface = (struct interface_entry *)
79                 list_remove_first(&cfg->interfaces);
80         while (iface != NULL) {
81                 if (iface->icmp_sock >= 0)
82                         close(iface->icmp_sock);
83                 if (iface->udp_sock >= 0)
84                         close(iface->udp_sock);
85                 if (iface->udp_bc_sock >= 0)
86                         close(iface->udp_bc_sock);
87                 if (iface->udp_bc_sock2 >= 0)
88                         close(iface->udp_bc_sock2);
89                 free(iface);
90                 iface = (struct interface_entry *)
91                         list_remove_first(&cfg->interfaces);
92         }
93 }
```

4.13.3.2 int load_config (struct ha_config * *ha*, char * *program_name*, char * *config_file*)

Definition at line 97 of file ha_config.c.

References ASSERT, ha_config::authorized_list, load_ha_data::cfg, cleanup_config(), ha_config::enable_reverse_tunneling, ha_config::enable_triangle_tunneling, ha_config::fa_spi_list, FALSE, HA_DEFAULT_MAX_BINDINGS, HA_DEFAULT_REG_ERROR_REPLY_INTERVAL, HA_DEFAULT_REG_PORT, HA_DEFAULT_SYSLOG_FACILITY, HA_DEFAULT_TUNNEL_LIFETIME, ha_config::ha_default_tunnel_lifetime, HASH_METHOD_CHECK, ha_config::interfaces, ha_config::max_bindings, load_ha_data::process_authorized_list, load_ha_data::process_fa_spi_list, load_ha_data::process_interfaces, load_ha_data::process_spi_list, ha_config::pubkey_hash_method, ha_config::reg_error_reply_interval, ha_config::socket_priority, ha_config::spi_list, ha_config::syslog_facility, TRUE, and ha_config::udpport.

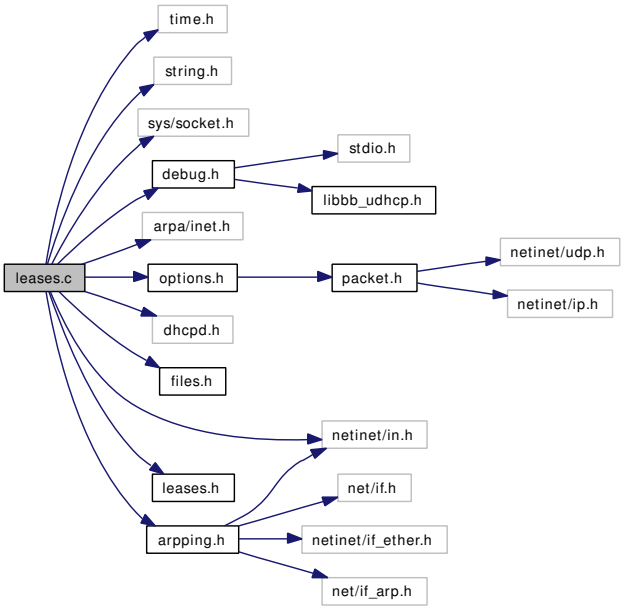
```
98 {
99         FILE *file;
100         struct load_ha_data ha;
101
102         ASSERT(cfg);
103         ha.cfg = cfg;
104         memset(cfg, 0, sizeof(struct ha_config));
105         ha.process_spi_list = FALSE;
106         list_init(&cfg->spi_list);
107         ha.process_authorized_list = FALSE;
108         list_init(&cfg->authorized_list);
109         ha.process_fa_spi_list = FALSE;
110         list_init(&cfg->fa_spi_list);
111         ha.process_interfaces = FALSE;
112         list_init(&cfg->interfaces);
113
114         /* set default values */
115         cfg->max_bindings = HA_DEFAULT_MAX_BINDINGS;
116         cfg->ha_default_tunnel_lifetime = HA_DEFAULT_TUNNEL_LIFETIME;
117         cfg->reg_error_reply_interval = HA_DEFAULT_REG_ERROR_REPLY_INTERVAL;
118         cfg->syslog_facility = HA_DEFAULT_SYSLOG_FACILITY;
119         cfg->udpport = HA_DEFAULT_REG_PORT;
120         cfg->socket_priority = -1;
121         cfg->enable_triangle_tunneling = TRUE;
122         cfg->enable_reverse_tunneling = TRUE;
123         cfg->pubkey_hash_method = HASH_METHOD_CHECK;
124
125         file = fopen(config_file, "r");
126         if (file == NULL) {
127                 fprintf(stderr,
128                         "%s: Could not open configuration file '%s'.\n",
129                         program_name, config_file);
130                 return FALSE;
131         }
132         if (load_data(&ha, file, process_load_ha) == FALSE) {
133                 fprintf(stderr,
134                         "%s: Error while interpreting file '%s'!\n",
135                         program_name, config_file);
136                 fclose(file);
137                 cleanup_config(cfg);
138                 return FALSE;
139         }
140         fclose(file);
141
142         openlog("home agent", LOG_PID | LOG_CONS, cfg->syslog_facility);
143
144         return TRUE;
145 }
```

Here is the call graph for this function:

4.14 leases.c File Reference

```
#include <time.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "debug.h"
#include "dhcpd.h"
#include "files.h"
#include "options.h"
#include "leases.h"
#include "arpping.h"
```

Include dependency graph for leases.c:



Functions

- void **clear_lease** (u_int8_t *chaddr, u_int32_t yiaddr)
- **dhcpOfferedAddr** * **add_lease** (u_int8_t *chaddr, u_int32_t yiaddr, unsigned long lease)
- int **lease_expired** (struct **dhcpOfferedAddr** *lease)
- **dhcpOfferedAddr** * **oldest_expired_lease** (void)
- **dhcpOfferedAddr** * **find_lease_by_chaddr** (u_int8_t *chaddr)
- **dhcpOfferedAddr** * **find_lease_by_yiaddr** (u_int32_t yiaddr)
- u_int32_t **find_address** (int check_expired)
- int **check_ip** (u_int32_t addr)

Variables

- unsigned char **blank_chaddr** [] = {[0 ... 15] = 0}

4.14.1 Function Documentation

4.14.1.1 struct dhcpOfferedAddr* add_lease (u_int8_t * *chaddr*, u_int32_t *yiaddr*, unsigned long *lease*)

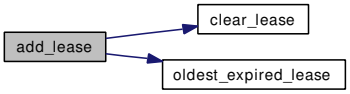
Definition at line 37 of file leases.c.

References dhcpOfferedAddr::chaddr, clear_lease(), dhcpOfferedAddr::expires, oldest_expired_lease(), and dhcpOfferedAddr::yiaddr.

Referenced by check_ip(), read_leases(), sendACK(), and sendOffer().

```
38 {
39         struct dhcpOfferedAddr *oldest;
40 
41         /* clean out any old ones */
42         clear_lease(chaddr, yiaddr);
43 
44         oldest = oldest_expired_lease();
45 
46         if (oldest) {
47                 memcpy(oldest->chaddr, chaddr, 16);
48                 oldest->yiaddr = yiaddr;
49                 oldest->expires = time(0) + lease;
50         }
51 
52         return oldest;
53 }
```

Here is the call graph for this function:



4.14.1.2 int check_ip (u_int32_t *addr*)

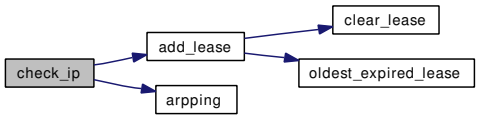
Definition at line 139 of file leases.c.

References add_lease(), arpping(), blank_chaddr, LOG, and LOG_INFO.

Referenced by find_address().

```
140 {
141         struct in_addr temp;
142 
143         if (arpping(addr, server_config.server, server_config.arp, server_config.interface) == 0) {
144                 temp.s_addr = addr;
145                 LOG(LOG_INFO, "%s belongs to someone, reserving it for %ld seconds",
146                         inet_ntoa(temp), server_config.conflict_time);
147                 add_lease(blank_chaddr, addr, server_config.conflict_time);
148                 return 1;
149         } else return 0;
150 }
```

Here is the call graph for this function:



4.14.1.3 void clear_lease (u_int8_t * *chaddr*, u_int32_t *yiaddr*)

Definition at line 22 of file leases.c.

Referenced by add_lease().

```
23 {
24         unsigned int i, j;
25
26         for (j = 0; j < 16 && !chaddr[j]; j++);
27
28         for (i = 0; i < server_config.max_leases; i++)
29                 if ((j != 16 && !memcmp(leases[i].chaddr, chaddr, 16)) ||
30                     (yiaddr && leases[i].yiaddr == yiaddr)) {
31                         memset(&(leases[i]), 0, sizeof(struct dhcpOfferedAddr));
32                 }
33 }
```

4.14.1.4 u_int32_t find_address (int *check_expired*)

Definition at line 107 of file leases.c.

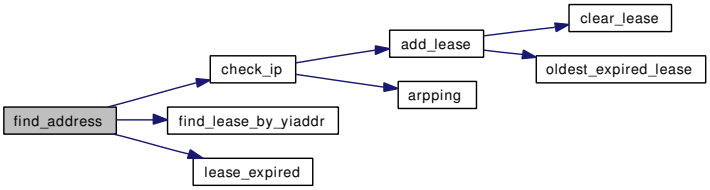
References check_ip(), find_lease_by_yiaddr(), and lease_expired().

Referenced by sendOffer().

```
108 {
109         u_int32_t addr, ret;
110         struct dhcpOfferedAddr *lease = NULL;
111
112         addr = ntohl(server_config.start); /* addr is in host order here */
113         for (;addr <= ntohl(server_config.end); addr++) {
114
115                 /* ie, 192.168.55.0 */
116                 if (!(addr & 0xFF)) continue;
117
118                 /* ie, 192.168.55.255 */
119                 if ((addr & 0xFF) == 0xFF) continue;
120
121                 /* lease is not taken */
122                 ret = htonl(addr);
123                 if ((!(lease = find_lease_by_yiaddr(ret)) ||
124
125                      /* or it expired and we are checking for expired leases */
126                      (check_expired  && lease_expired(lease))) &&
127
128                      /* and it isn't on the network */
129                      !check_ip(ret)) {
130                         return ret;
131                         break;
132                 }
133         }
```

```
134         return 0;
135 }
```

Here is the call graph for this function:



4.14.1.5 struct dhcpOfferedAddr* find_lease_by_chaddr (u_int8_t * *chaddr*)

Definition at line 82 of file leases.c.

Referenced by sendOffer().

```
83 {
84         unsigned int i;
85
86         for (i = 0; i < server_config.max_leases; i++)
87                 if (!memcmp(leases[i].chaddr, chaddr, 16)) return &(leases[i]);
88
89         return NULL;
90 }
```

4.14.1.6 struct dhcpOfferedAddr* find_lease_by_yiaddr (u_int32_t *yiaddr*)

Definition at line 94 of file leases.c.

Referenced by find_address(), and sendOffer().

```
95 {
96         unsigned int i;
97
98         for (i = 0; i < server_config.max_leases; i++)
99                 if (leases[i].yiaddr == yiaddr) return &(leases[i]);
100
101         return NULL;
102 }
```

4.14.1.7 int lease_expired (struct dhcpOfferedAddr * *lease*)

Definition at line 57 of file leases.c.

References dhcpOfferedAddr::expires.

Referenced by find_address(), sendOffer(), and write_leases().

```
58 {
59         return (lease->expires < (unsigned long) time(0));
60 }
```

4.14.1.8 struct dhcpOfferedAddr* oldest_expired_lease (void)

Definition at line 64 of file leases.c.

References dhcpOfferedAddr::expires.

Referenced by add_lease().

```
65 {
66         struct dhcpOfferedAddr *oldest = NULL;
67         unsigned long oldest_lease = time(0);
68         unsigned int i;
69
70
71         for (i = 0; i < server_config.max_leases; i++)
72                 if (oldest_lease > leases[i].expires) {
73                         oldest_lease = leases[i].expires;
74                         oldest = &(leases[i]);
75                 }
76         return oldest;
77
78 }
```

4.14.2 Variable Documentation

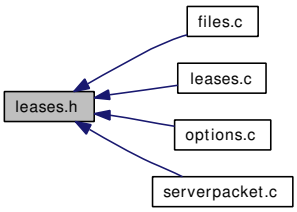
4.14.2.1 unsigned char blank_chaddr[ ] = {[0 ... 15] = 0}

Definition at line 19 of file leases.c.

Referenced by check_ip().

4.15 leases.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct **dhcpOfferedAddr**

Functions

- void **clear_lease** (u_int8_t *chaddr, u_int32_t yiaddr)
- **dhcpOfferedAddr** * **add_lease** (u_int8_t *chaddr, u_int32_t yiaddr, unsigned long lease)
- int **lease_expired** (struct **dhcpOfferedAddr** *lease)
- **dhcpOfferedAddr** * **oldest_expired_lease** (void)
- **dhcpOfferedAddr** * **find_lease_by_chaddr** (u_int8_t *chaddr)
- **dhcpOfferedAddr** * **find_lease_by_yiaddr** (u_int32_t yiaddr)
- u_int32_t **find_address** (int check_expired)
- int **check_ip** (u_int32_t addr)

Variables

- unsigned char **blank_chaddr** []

4.15.1 Function Documentation

4.15.1.1 struct dhcpOfferedAddr* add_lease (u_int8_t * *chaddr*, u_int32_t *yiaddr*, unsigned long *lease*)

Definition at line 37 of file leases.c.

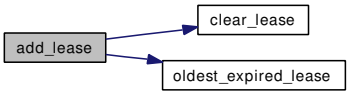
References dhcpOfferedAddr::chaddr, clear_lease(), dhcpOfferedAddr::expires, oldest_expired_lease(), and dhcpOfferedAddr::yiaddr.

Referenced by check_ip(), read_leases(), sendACK(), and sendOffer().

```
38 {
39         struct dhcpOfferedAddr *oldest;
40
41         /* clean out any old ones */
42         clear_lease(chaddr, yiaddr);
43
44         oldest = oldest_expired_lease();
45
```

```
46         if (oldest) {
47                 memcpy(oldest->chaddr, chaddr, 16);
48                 oldest->yiaddr = yiaddr;
49                 oldest->expires = time(0) + lease;
50         }
51
52         return oldest;
53 }
```

Here is the call graph for this function:



4.15.1.2 int check_ip (u_int32_t *addr*)

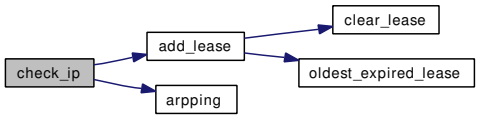
Definition at line 139 of file leases.c.

References add_lease(), arpping(), blank_chaddr, LOG, and LOG_INFO.

Referenced by find_address().

```
140 {
141         struct in_addr temp;
142
143         if (arpping(addr, server_config.server, server_config.arp, server_config.interface) == 0) {
144                 temp.s_addr = addr;
145                 LOG(LOG_INFO, "%s belongs to someone, reserving it for %ld seconds",
146                         inet_ntoa(temp), server_config.conflict_time);
147                 add_lease(blank_chaddr, addr, server_config.conflict_time);
148                 return 1;
149         } else return 0;
150 }
```

Here is the call graph for this function:



4.15.1.3 void clear_lease (u_int8_t * *chaddr*, u_int32_t *yiaddr*)

Definition at line 22 of file leases.c.

Referenced by add_lease().

```
23 {
24         unsigned int i, j;
25
26         for (j = 0; j < 16 && !chaddr[j]; j++);
27
28         for (i = 0; i < server_config.max_leases; i++)
```

```
29                 if ((j != 16 && !memcmp(leases[i].chaddr, chaddr, 16)) ||
30                     (yiaddr && leases[i].yiaddr == yiaddr)) {
31                         memset(&(leases[i]), 0, sizeof(struct dhcpOfferedAddr));
32                 }
33 }
```

4.15.1.4 u_int32_t find_address (int *check_expired*)

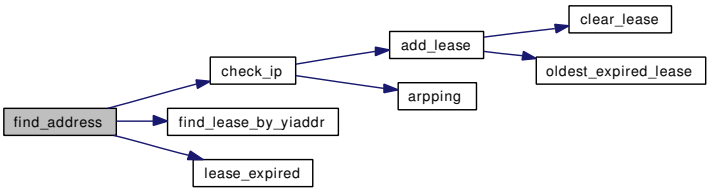
Definition at line 107 of file leases.c.

References check_ip(), find_lease_by_yiaddr(), and lease_expired().

Referenced by sendOffer().

```
108 {
109         u_int32_t addr, ret;
110         struct dhcpOfferedAddr *lease = NULL;
111
112         addr = ntohl(server_config.start); /* addr is in host order here */
113         for (;addr <= ntohl(server_config.end); addr++) {
114
115                 /* ie, 192.168.55.0 */
116                 if (!(addr & 0xFF)) continue;
117
118                 /* ie, 192.168.55.255 */
119                 if ((addr & 0xFF) == 0xFF) continue;
120
121                 /* lease is not taken */
122                 ret = htonl(addr);
123                 if ((!(lease = find_lease_by_yiaddr(ret)) ||
124
125                      /* or it expired and we are checking for expired leases */
126                      (check_expired  && lease_expired(lease))) &&
127
128                      /* and it isn't on the network */
129                      !check_ip(ret)) {
130                         return ret;
131                         break;
132                 }
133         }
134         return 0;
135 }
```

Here is the call graph for this function:



4.15.1.5 struct dhcpOfferedAddr* find_lease_by_chaddr (u_int8_t * *chaddr*)

Definition at line 82 of file leases.c.

Referenced by sendOffer().

```
83 {
84         unsigned int i;
85
86         for (i = 0; i < server_config.max_leases; i++)
87                 if (!memcmp(leases[i].chaddr, chaddr, 16)) return &(leases[i]);
88
89         return NULL;
90 }
```

4.15.1.6 struct dhcpOfferedAddr* find_lease_by_yiaddr (u_int32_t *yiaddr*)

Definition at line 94 of file leases.c.

Referenced by find_address(), and sendOffer().

```
95 {
96         unsigned int i;
97
98         for (i = 0; i < server_config.max_leases; i++)
99                 if (leases[i].yiaddr == yiaddr) return &(leases[i]);
100
101         return NULL;
102 }
```

4.15.1.7 int lease_expired (struct dhcpOfferedAddr * *lease*)

Definition at line 57 of file leases.c.

References dhcpOfferedAddr::expires.

Referenced by find_address(), sendOffer(), and write_leases().

```
58 {
59         return (lease->expires < (unsigned long) time(0));
60 }
```

4.15.1.8 struct dhcpOfferedAddr* oldest_expired_lease (void)

Definition at line 64 of file leases.c.

References dhcpOfferedAddr::expires.

Referenced by add_lease().

```
65 {
66         struct dhcpOfferedAddr *oldest = NULL;
67         unsigned long oldest_lease = time(0);
68         unsigned int i;
69
70
71         for (i = 0; i < server_config.max_leases; i++)
72                 if (oldest_lease > leases[i].expires) {
73                         oldest_lease = leases[i].expires;
74                         oldest = &(leases[i]);
75                 }
76         return oldest;
77
78 }
```

4.15.2 Variable Documentation

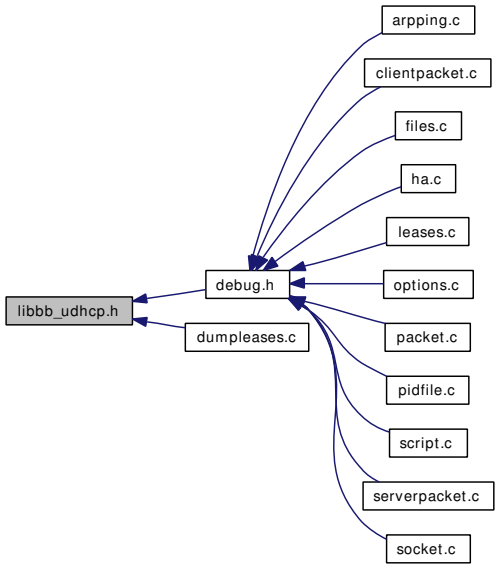
4.15.2.1 unsigned char blank_chaddr[]

Definition at line 19 of file leases.c.

Referenced by check_ip().

4.16 libbb_udhcp.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define **TRUE** 1
- #define **FALSE** 0
- #define **xmalloc** malloc

4.16.1 Define Documentation

4.16.1.1 #define FALSE 0

Definition at line 23 of file libbb_udhcp.h.

Referenced by load_config().

4.16.1.2 #define TRUE 1

Definition at line 22 of file libbb_udhcp.h.

Referenced by load_config().

4.16.1.3 #define xmalloc malloc

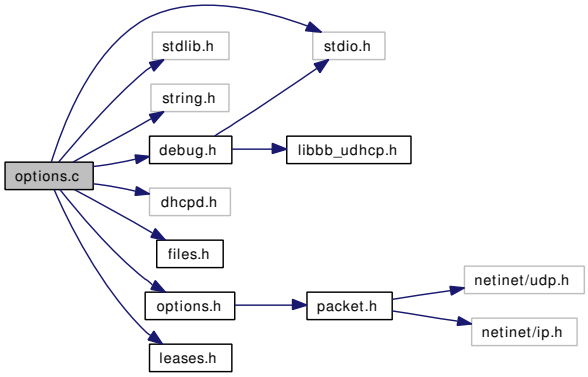
Definition at line 25 of file libbb_udhcp.h.

Referenced by add_dhcp_mobile(), and udhcp().

4.17 options.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "debug.h"
#include "dhcpd.h"
#include "files.h"
#include "options.h"
#include "leases.h"
```

Include dependency graph for options.c:



Functions

- unsigned char * **get_option** (struct **dhcpMessage** *packet, int code)
- int **end_option** (unsigned char *optionptr)
- int **add_option_string** (unsigned char *optionptr, unsigned char *string)
- int **add_simple_option** (unsigned char *optionptr, unsigned char code, u_int32_t data)
- option_set * **find_option** (struct option_set *opt_list, char code)
- void **attach_option** (struct option_set **opt_list, struct **dhcp_option** *option, char *buffer, int length)

Variables

- **dhcp_option options** []
- int **option_lengths** []

4.17.1 Function Documentation

4.17.1.1 int add_option_string (unsigned char * *optionptr*, unsigned char * *string*)

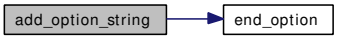
Definition at line 133 of file options.c.

References DEBUG, DHCP_END, end_option(), LOG, LOG_ERR, LOG_INFO, OPT_CODE, and OPT_LEN.

Referenced by add_simple_option(), send_inform(), sendACK(), and sendOffer().

```
134 {
135         int end = end_option(optionptr);
136
137         /* end position + string length + option code/length + end option */
138         if (end + string[OPT_LEN] + 2 + 1 >= 308) {
139                 LOG(LOG_ERR, "Option 0x%02x did not fit into the packet!", string[OPT_CODE]);
140                 return 0;
141         }
142         DEBUG(LOG_INFO, "adding option 0x%02x", string[OPT_CODE]);
143         memcpy(optionptr + end, string, string[OPT_LEN] + 2);
144         optionptr[end + string[OPT_LEN] + 2] = DHCP_END;
145         return string[OPT_LEN] + 2;
146 }
```

Here is the call graph for this function:



4.17.1.2 int add_simple_option (unsigned char * *optionptr*, unsigned char *code*, u_int32_t *data*)

Definition at line 150 of file options.c.

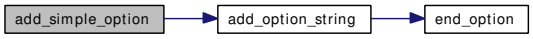
References add_option_string(), dhcp_option::code, DEBUG, dhcp_option::flags, LOG_ERR, OPT_CODE, OPT_LEN, option_lengths, options, and TYPE_MASK.

Referenced by init_header(), send_discover(), send_release(), send_selecting(), sendACK(), and sendOffer().

```
151 {
152         char length = 0;
153         int i;
154         unsigned char option[2 + 4];
155         unsigned char *u8;
156         u_int16_t *u16;
157         u_int32_t *u32;
158         u_int32_t aligned;
159         u8 = (unsigned char *) &aligned;
160         u16 = (u_int16_t *) &aligned;
161         u32 = &aligned;
162
163         for (i = 0; options[i].code; i++)
164                 if (options[i].code == code) {
165                         length = option_lengths[options[i].flags & TYPE_MASK];
166                 }
167
168         if (!length) {
169                 DEBUG(LOG_ERR, "Could not add option 0x%02x", code);
170                 return 0;
171         }
172
173         option[OPT_CODE] = code;
174         option[OPT_LEN] = length;
175
176         switch (length) {
```

```
177                 case 1: *u8 =  data; break;
178                 case 2: *u16 = data; break;
179                 case 4: *u32 = data; break;
180         }
181         memcpy(option + 2, &aligned, length);
182         return add_option_string(optionptr, option);
183 }
```

Here is the call graph for this function:



4.17.1.3 void attach_option (struct option_set ** *opt_list*, struct dhcp_option * *option*, char * *buffer*, int *length*)

Definition at line 198 of file options.c.

References dhcp_option::code, DEBUG, find_option(), dhcp_option::flags, LOG_INFO, dhcp_option::name, OPT_CODE, OPT_LEN, and OPTION_LIST.

```
199 {
200         struct option_set *existing, *new, **curr;
201
202         /* add it to an existing option */
203         if ((existing = find_option(*opt_list, option->code))) {
204                 DEBUG(LOG_INFO, "Attaching option %s to existing member of list", option->name);
205                 if (option->flags & OPTION_LIST) {
206                         if (existing->data[OPT_LEN] + length <= 255) {
207                                 existing->data = realloc(existing->data,
208                                                          existing->data[OPT_LEN] + length + 2);
209                                 memcpy(existing->data + existing->data[OPT_LEN] + 2, buffer, length);
210                                 existing->data[OPT_LEN] += length;
211                         } /* else, ignore the data, we could put this in a second option in the future */
212                 } /* else, ignore the new data */
213         } else {
214                 DEBUG(LOG_INFO, "Attaching option %s to list", option->name);
215
216                 /* make a new option */
217                 new = malloc(sizeof(struct option_set));
218                 new->data = malloc(length + 2);
219                 new->data[OPT_CODE] = option->code;
220                 new->data[OPT_LEN] = length;
221                 memcpy(new->data + 2, buffer, length);
222
223                 curr = opt_list;
224                 while (*curr && (*curr)->data[OPT_CODE] < option->code)
225                         curr = &(*curr)->next;
226
227                 new->next = *curr;
228                 *curr = new;
229         }
230 }
```

Here is the call graph for this function:

4.17.1.4 int end_option (unsigned char * *optionptr*)

Definition at line 119 of file options.c.

References DHCP_END, DHCP_PADDING, and OPT_LEN.

Referenced by add_option_string().

```
120 {
121         int i = 0;
122
123         while (optionptr[i] != DHCP_END) {
124                 if (optionptr[i] == DHCP_PADDING) i++;
125                 else i += optionptr[i + OPT_LEN] + 2;
126         }
127         return i;
128 }
```

4.17.1.5 struct option_set* find_option (struct option_set * *opt_list*, char *code*)

Definition at line 187 of file options.c.

References OPT_CODE.

Referenced by attach_option().

```
188 {
189         while (opt_list && opt_list->data[OPT_CODE] < code)
190                 opt_list = opt_list->next;
191
192         if (opt_list && opt_list->data[OPT_CODE] == code) return opt_list;
193         else return NULL;
194 }
```

4.17.1.6 unsigned char* get_option (struct dhcpMessage * *packet*, int *code*)

Definition at line 64 of file options.c.

References DHCP_END, DHCP_OPTION_OVER, DHCP_PADDING, dhcpMessage::file, FILE_FIELD, LOG, LOG_WARNING, OPT_CODE, OPT_LEN, OPTION_FIELD, dhcpMessage::options, dhcpMessage::sname, and SNAME_FIELD.

Referenced by get_packet(), sendACK(), sendOffer(), and udhcp().

```
65 {
66         int i, length;
67         unsigned char *optionptr;
68         int over = 0, done = 0, curr = OPTION_FIELD;
69
70         optionptr = packet->options;
71         i = 0;
72         length = 308;
73         while (!done) {
74                 if (i >= length) {
75                         LOG(LOG_WARNING, "bogus packet, option fields too long.");
76                         return NULL;
77                 }
78                 if (optionptr[i + OPT_CODE] == code) {
79                         if (i + 1 + optionptr[i + OPT_LEN] >= length) {
80                                 LOG(LOG_WARNING, "bogus packet, option fields too long.");
```

```
81                                  return NULL;
82                          }
83                          return optionptr + i + 2;
84                  }
85                  switch (optionptr[i + OPT_CODE]) {
86                  case DHCP_PADDING:
87                          i++;
88                          break;
89                  case DHCP_OPTION_OVER:
90                          if (i + 1 + optionptr[i + OPT_LEN] >= length) {
91                                  LOG(LOG_WARNING, "bogus packet, option fields too long.");
92                                  return NULL;
93                          }
94                          over = optionptr[i + 3];
95                          i += optionptr[OPT_LEN] + 2;
96                          break;
97                  case DHCP_END:
98                          if (curr == OPTION_FIELD && over & FILE_FIELD) {
99                                  optionptr = packet->file;
100                                  i = 0;
101                                  length = 128;
102                                  curr = FILE_FIELD;
103                          } else if (curr == FILE_FIELD && over & SNAME_FIELD) {
104                                  optionptr = packet->sname;
105                                  i = 0;
106                                  length = 64;
107                                  curr = SNAME_FIELD;
108                          } else done = 1;
109                          break;
110                  default:
111                          i += optionptr[OPT_LEN + i] + 2;
112                  }
113          }
114          return NULL;
115 }
```

4.17.2 Variable Documentation

4.17.2.1 int option_lengths[]

Initial value:

```
 {
        [OPTION_IP] =           4,
        [OPTION_IP_PAIR] =      8,
        [OPTION_BOOLEAN] =      1,
        [OPTION_STRING] =       1,
        [OPTION_U8] =           1,
        [OPTION_U16] =          2,
        [OPTION_S16] =          2,
        [OPTION_U32] =          4,
        [OPTION_S32] =          4
}
```

Definition at line 50 of file options.c.

Referenced by add_simple_option().

4.17.2.2 struct dhcp_option options[]

Initial value:

```
{
        {"subnet",      OPTION_IP | OPTION_REQ,                 0x01},
        {"timezone",    OPTION_S32,                             0x02},
        {"router",      OPTION_IP | OPTION_LIST | OPTION_REQ,   0x03},
        {"timesvr",     OPTION_IP | OPTION_LIST,                0x04},
        {"namesvr",     OPTION_IP | OPTION_LIST,                0x05},
        {"dns",         OPTION_IP | OPTION_LIST | OPTION_REQ,   0x06},
        {"logsvr",      OPTION_IP | OPTION_LIST,                0x07},
        {"cookiesvr",   OPTION_IP | OPTION_LIST,                0x08},
        {"lprsvr",      OPTION_IP | OPTION_LIST,                0x09},
        {"hostname",    OPTION_STRING | OPTION_REQ,             0x0c},
        {"bootsize",    OPTION_U16,                             0x0d},
        {"domain",      OPTION_STRING | OPTION_REQ,             0x0f},
        {"swapsvr",     OPTION_IP,                              0x10},
        {"rootpath",    OPTION_STRING,                          0x11},
        {"ipttl",       OPTION_U8,                              0x17},
        {"mtu",         OPTION_U16,                             0x1a},
        {"broadcast",   OPTION_IP | OPTION_REQ,                 0x1c},
        {"ntpsrv",      OPTION_IP | OPTION_LIST,                0x2a},
        {"wins",        OPTION_IP | OPTION_LIST,                0x2c},
        {"requestip",   OPTION_IP,                              0x32},
        {"lease",       OPTION_U32,                             0x33},
        {"dhcptype",    OPTION_U8,                              0x35},
        {"serverid",    OPTION_IP,                              0x36},
        {"message",     OPTION_STRING,                          0x38},
        {"tftp",        OPTION_STRING,                          0x42},
        {"bootfile",    OPTION_STRING,                          0x43},
        {"",            0x00,                           0x00}
}
```

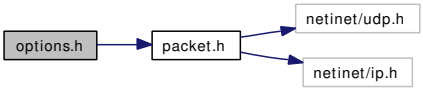
Definition at line 18 of file options.c.

Referenced by add_simple_option().

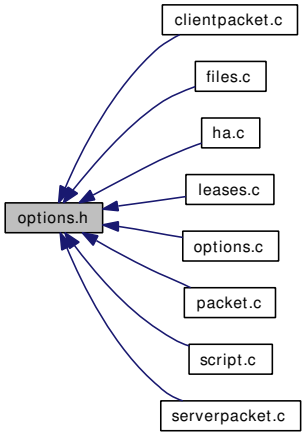
4.18 options.h File Reference

```
#include "packet.h"
```

Include dependency graph for options.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct **dhcp_option**

Defines

- #define **TYPE_MASK** 0x0F
- #define **OPTION_REQ** 0x10
- #define **OPTION_LIST** 0x20

Enumerations

- enum {
  OPTION_IP = 1, **OPTION_IP_PAIR**, **OPTION_STRING**, **OPTION_BOOLEAN**,
  OPTION_U8, **OPTION_U16**, **OPTION_S16**, **OPTION_U32**,
  OPTION_S32 }

Functions

- unsigned char * **get_option** (struct **dhcpMessage** *packet, int code)

- int **end_option** (unsigned char *optionptr)
- int **add_option_string** (unsigned char *optionptr, unsigned char *string)
- int **add_simple_option** (unsigned char *optionptr, unsigned char code, u_int32_t data)
- option_set * **find_option** (struct option_set *opt_list, char code)
- void **attach_option** (struct option_set **opt_list, struct **dhcp_option** *option, char *buffer, int length)

Variables

- **dhcp_option options** []
- int **option_lengths** []

4.18.1 Define Documentation

4.18.1.1 #define OPTION_LIST 0x20

Definition at line 22 of file options.h.

Referenced by attach_option().

4.18.1.2 #define OPTION_REQ 0x10

Definition at line 21 of file options.h.

4.18.1.3 #define TYPE_MASK 0x0F

Definition at line 7 of file options.h.

Referenced by add_simple_option().

4.18.2 Enumeration Type Documentation

4.18.2.1 anonymous enum

Enumerator:

OPTION_IP
OPTION_IP_PAIR
OPTION_STRING
OPTION_BOOLEAN
OPTION_U8
OPTION_U16
OPTION_S16
OPTION_U32
OPTION_S32

Definition at line 9 of file options.h.

```
9       {
10          OPTION_IP=1,
11          OPTION_IP_PAIR,
12          OPTION_STRING,
13          OPTION_BOOLEAN,
14          OPTION_U8,
15          OPTION_U16,
16          OPTION_S16,
17          OPTION_U32,
18          OPTION_S32
19  };
```

4.18.3 Function Documentation

4.18.3.1 int add_option_string (unsigned char * *optionptr*, unsigned char * *string*)

Definition at line 133 of file options.c.

References DEBUG, DHCP_END, end_option(), LOG, LOG_ERR, LOG_INFO, OPT_CODE, and OPT_LEN.

Referenced by add_simple_option(), send_inform(), sendACK(), and sendOffer().

```
134 {
135         int end = end_option(optionptr);
136
137         /* end position + string length + option code/length + end option */
138         if (end + string[OPT_LEN] + 2 + 1 >= 308) {
139                 LOG(LOG_ERR, "Option 0x%02x did not fit into the packet!", string[OPT_CODE]);
140                 return 0;
141         }
142         DEBUG(LOG_INFO, "adding option 0x%02x", string[OPT_CODE]);
143         memcpy(optionptr + end, string, string[OPT_LEN] + 2);
144         optionptr[end + string[OPT_LEN] + 2] = DHCP_END;
145         return string[OPT_LEN] + 2;
146 }
```

Here is the call graph for this function:

4.18.3.2 int add_simple_option (unsigned char * *optionptr*, unsigned char *code*, u_int32_t *data*)

Definition at line 150 of file options.c.

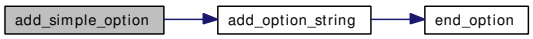
References add_option_string(), dhcp_option::code, DEBUG, dhcp_option::flags, LOG_ERR, OPT_CODE, OPT_LEN, option_lengths, options, and TYPE_MASK.

Referenced by init_header(), send_discover(), send_release(), send_selecting(), sendACK(), and sendOffer().

```
151 {
152         char length = 0;
153         int i;
154         unsigned char option[2 + 4];
155         unsigned char *u8;
```

```
156         u_int16_t *u16;
157         u_int32_t *u32;
158         u_int32_t aligned;
159         u8 = (unsigned char *) &aligned;
160         u16 = (u_int16_t *) &aligned;
161         u32 = &aligned;
162 
163         for (i = 0; options[i].code; i++)
164                 if (options[i].code == code) {
165                         length = option_lengths[options[i].flags & TYPE_MASK];
166                 }
167 
168         if (!length) {
169                 DEBUG(LOG_ERR, "Could not add option 0x%02x", code);
170                 return 0;
171         }
172 
173         option[OPT_CODE] = code;
174         option[OPT_LEN] = length;
175 
176         switch (length) {
177                 case 1: *u8 =  data; break;
178                 case 2: *u16 = data; break;
179                 case 4: *u32 = data; break;
180         }
181         memcpy(option + 2, &aligned, length);
182         return add_option_string(optionptr, option);
183 }
```

Here is the call graph for this function:



4.18.3.3 void attach_option (struct option_set ** *opt_list*, struct dhcp_option * *option*, char * *buffer*, int *length*)

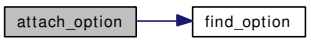
Definition at line 198 of file options.c.

References dhcp_option::code, DEBUG, find_option(), dhcp_option::flags, LOG_INFO, dhcp_option::name, OPT_CODE, OPT_LEN, and OPTION_LIST.

```
199 {
200         struct option_set *existing, *new, **curr;
201 
202         /* add it to an existing option */
203         if ((existing = find_option(*opt_list, option->code))) {
204                 DEBUG(LOG_INFO, "Attaching option %s to existing member of list", option->name);
205                 if (option->flags & OPTION_LIST) {
206                         if (existing->data[OPT_LEN] + length <= 255) {
207                                 existing->data = realloc(existing->data,
208                                                          existing->data[OPT_LEN] + length + 2);
209                                 memcpy(existing->data + existing->data[OPT_LEN] + 2, buffer, length);
210                                 existing->data[OPT_LEN] += length;
211                         } /* else, ignore the data, we could put this in a second option in the future */
212                 } /* else, ignore the new data */
213         } else {
214                 DEBUG(LOG_INFO, "Attaching option %s to list", option->name);
215 
216                 /* make a new option */
217                 new = malloc(sizeof(struct option_set));
218                 new->data = malloc(length + 2);
```

```
219                 new->data[OPT_CODE] = option->code;
220                 new->data[OPT_LEN] = length;
221                 memcpy(new->data + 2, buffer, length);
222
223                 curr = opt_list;
224                 while (*curr && (*curr)->data[OPT_CODE] < option->code)
225                         curr = &(*curr)->next;
226
227                 new->next = *curr;
228                 *curr = new;
229         }
230 }
```

Here is the call graph for this function:



4.18.3.4 int end_option (unsigned char * *optionptr*)

Definition at line 119 of file options.c.

References DHCP_END, DHCP_PADDING, and OPT_LEN.

Referenced by add_option_string().

```
120 {
121         int i = 0;
122
123         while (optionptr[i] != DHCP_END) {
124                 if (optionptr[i] == DHCP_PADDING) i++;
125                 else i += optionptr[i + OPT_LEN] + 2;
126         }
127         return i;
128 }
```

4.18.3.5 struct option_set* find_option (struct option_set * *opt_list*, char *code*)

Definition at line 187 of file options.c.

References OPT_CODE.

Referenced by attach_option().

```
188 {
189         while (opt_list && opt_list->data[OPT_CODE] < code)
190                 opt_list = opt_list->next;
191
192         if (opt_list && opt_list->data[OPT_CODE] == code) return opt_list;
193         else return NULL;
194 }
```

4.18.3.6 unsigned char* get_option (struct dhcpMessage * *packet*, int *code*)

Definition at line 64 of file options.c.

References DHCP_END, DHCP_OPTION_OVER, DHCP_PADDING, dhcpMessage::file, FILE_FIELD, LOG, LOG_WARNING, OPT_CODE, OPT_LEN, OPTION_FIELD, dhcpMessage::options, dhcpMessage::sname, and SNAME_FIELD.

Referenced by get_packet(), sendACK(), sendOffer(), and udhcp().

```
65 {
66         int i, length;
67         unsigned char *optionptr;
68         int over = 0, done = 0, curr = OPTION_FIELD;
69
70         optionptr = packet->options;
71         i = 0;
72         length = 308;
73         while (!done) {
74                 if (i >= length) {
75                         LOG(LOG_WARNING, "bogus packet, option fields too long.");
76                         return NULL;
77                 }
78                 if (optionptr[i + OPT_CODE] == code) {
79                         if (i + 1 + optionptr[i + OPT_LEN] >= length) {
80                                 LOG(LOG_WARNING, "bogus packet, option fields too long.");
81                                 return NULL;
82                         }
83                         return optionptr + i + 2;
84                 }
85                 switch (optionptr[i + OPT_CODE]) {
86                 case DHCP_PADDING:
87                         i++;
88                         break;
89                 case DHCP_OPTION_OVER:
90                         if (i + 1 + optionptr[i + OPT_LEN] >= length) {
91                                 LOG(LOG_WARNING, "bogus packet, option fields too long.");
92                                 return NULL;
93                         }
94                         over = optionptr[i + 3];
95                         i += optionptr[OPT_LEN] + 2;
96                         break;
97                 case DHCP_END:
98                         if (curr == OPTION_FIELD && over & FILE_FIELD) {
99                                 optionptr = packet->file;
100                                 i = 0;
101                                 length = 128;
102                                 curr = FILE_FIELD;
103                         } else if (curr == FILE_FIELD && over & SNAME_FIELD) {
104                                 optionptr = packet->sname;
105                                 i = 0;
106                                 length = 64;
107                                 curr = SNAME_FIELD;
108                         } else done = 1;
109                         break;
110                 default:
111                         i += optionptr[OPT_LEN + i] + 2;
112                 }
113         }
114         return NULL;
115 }
```

4.18.4 Variable Documentation

4.18.4.1 int option_lengths[]

Definition at line 50 of file options.c.

Referenced by add_simple_option().

4.18.4.2 struct dhcp_option options[]

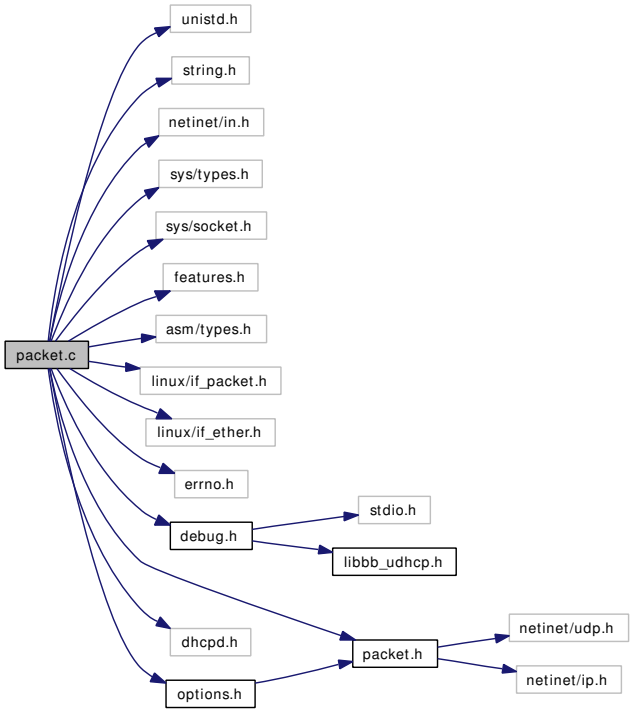
Definition at line 18 of file options.c.

Referenced by add_simple_option().

4.19 packet.c File Reference

```
#include <unistd.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <features.h>
#include <asm/types.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <errno.h>
#include "packet.h"
#include "debug.h"
#include "dhcpd.h"
#include "options.h"
```

Include dependency graph for packet.c:



Functions

- void **init_header** (struct **dhcpMessage** *packet, char type)
- int **get_packet** (struct **dhcpMessage** *packet, int fd)

- u_int16_t **checksum** (void *addr, int count)
- int **raw_packet** (struct **dhcpMessage** *payload, u_int32_t source_ip, int source_port, u_int32_t dest_ip, int dest_port, unsigned char *dest_arp, int ifindex)
- int **kernel_packet** (struct **dhcpMessage** *payload, u_int32_t source_ip, int source_port, u_int32_t dest_ip, int dest_port)

4.19.1 Function Documentation

4.19.1.1 u_int16_t checksum (void * *addr*, int *count*)

Definition at line 88 of file packet.c.

Referenced by get_raw_packet(), and raw_packet().

```
89 {
90         /* Compute Internet Checksum for "count" bytes
91          *         beginning at location "addr".
92          */
93         register int32_t sum = 0;
94         u_int16_t *source = (u_int16_t *) addr;
95 
96         while (count > 1)  {
97                 /*  This is the inner loop */
98                 sum += *source++;
99                 count -= 2;
100         }
101 
102         /*  Add left-over byte, if any */
103         if (count > 0) {
104                 /* Make sure that the left-over byte is added correctly both
105                  * with little and big endian hosts */
106                 u_int16_t tmp = 0;
107                 *(unsigned char *) (&tmp) = * (unsigned char *) source;
108                 sum += tmp;
109         }
110         /*  Fold 32-bit sum to 16 bits */
111         while (sum >> 16)
112                 sum = (sum & 0xffff) + (sum >> 16);
113 
114         return ~sum;
115 }
```

4.19.1.2 int get_packet (struct dhcpMessage * *packet*, int *fd*)

Definition at line 47 of file packet.c.

References BOOTREQUEST, BROADCAST_FLAG, dhcpMessage::cookie, DEBUG, DHCP_MAGIC, DHCP_VENDOR, dhcpMessage::flags, get_option(), LOG, LOG_ERR, LOG_INFO, dhcpMessage::op, and OPT_LEN.

Referenced by udhcp().

```
48 {
49         int bytes;
50         int i;
51         const char broken_vendors[][8] = {
52                 "MSFT 98",
53                 ""
54         };
55         char unsigned *vendor;
```

```
56
57         memset(packet, 0, sizeof(struct dhcpMessage));
58         bytes = read(fd, packet, sizeof(struct dhcpMessage));
59         if (bytes < 0) {
60
61                 DEBUG(LOG_INFO, "couldn't read on listening socket, ignoring");
62                 return -1;
63         }
64
65         if (ntohl(packet->cookie) != DHCP_MAGIC) {
66                 LOG(LOG_ERR, "received bogus message, ignoring");
67                 return -2;
68         }
69
70         DEBUG(LOG_INFO, "Received a packet");
71
72         if (packet->op == BOOTREQUEST && (vendor = get_option(packet, DHCP_VENDOR))) {
73                 for (i = 0; broken_vendors[i][0]; i++) {
74                         if (vendor[OPT_LEN - 2] == (unsigned char) strlen(broken_vendors[i]) &&
75                             !strncmp(vendor, broken_vendors[i], vendor[OPT_LEN - 2])) {
76                                 DEBUG(LOG_INFO, "broken client (%s), forcing broadcast",
77                                                 broken_vendors[i]);
78                                 packet->flags |= htons(BROADCAST_FLAG);
79                         }
80                 }
81         }
82
83
84         return bytes;
85 }
```

Here is the call graph for this function:

4.19.1.3 void init_header (struct dhcpMessage * *packet*, char *type*)

Definition at line 23 of file packet.c.

References add_simple_option(), BOOTREPLY, BOOTREQUEST, dhcpMessage::cookie, DHCP_END, DHCP_MAGIC, DHCP_MESSAGE_TYPE, DHCPACK, DHCPDISCOVER, DHCPINFORM, DHCPNAK, DHCPOFFER, DHCPRELEASE, DHCPREQUEST, ETH_10MB, ETH_10MB_LEN, dhcpMessage::hlen, dhcpMessage::htype, dhcpMessage::op, and dhcpMessage::options.

```
24 {
25         memset(packet, 0, sizeof(struct dhcpMessage));
26         switch (type) {
27         case DHCPDISCOVER:
28         case DHCPREQUEST:
29         case DHCPRELEASE:
30         case DHCPINFORM:
31                 packet->op = BOOTREQUEST;
32                 break;
33         case DHCPOFFER:
34         case DHCPACK:
35         case DHCPNAK:
36                 packet->op = BOOTREPLY;
37         }
38         packet->htype = ETH_10MB;
```

```
39         packet->hlen = ETH_10MB_LEN;
40         packet->cookie = htonl(DHCP_MAGIC);
41         packet->options[0] = DHCP_END;
42         add_simple_option(packet->options, DHCP_MESSAGE_TYPE, type);
43 }
```

Here is the call graph for this function:

4.19.1.4 int kernel_packet (struct dhcpMessage * *payload*, u_int32_t *source_ip*, int *source_port*, u_int32_t *dest_ip*, int *dest_port*)

Definition at line 174 of file packet.c.

Referenced by send_release(), and send_renew().

```
176 {
177         int n = 1;
178         int fd, result;
179         struct sockaddr_in client;
180 
181         if ((fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
182                 return -1;
183 
184         if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char *) &n, sizeof(n)) == -1)
185                 return -1;
186 
187         memset(&client, 0, sizeof(client));
188         client.sin_family = AF_INET;
189         client.sin_port = htons(source_port);
190         client.sin_addr.s_addr = source_ip;
191 
192         if (bind(fd, (struct sockaddr *)&client, sizeof(struct sockaddr)) == -1)
193                 return -1;
194 
195         memset(&client, 0, sizeof(client));
196         client.sin_family = AF_INET;
197         client.sin_port = htons(dest_port);
198         client.sin_addr.s_addr = dest_ip;
199 
200         if (connect(fd, (struct sockaddr *)&client, sizeof(struct sockaddr)) == -1)
201                 return -1;
202 
203         result = write(fd, payload, sizeof(struct dhcpMessage));
204         close(fd);
205 
206         return result;
207 }
```

4.19.1.5 int raw_packet (struct dhcpMessage * *payload*, u_int32_t *source_ip*, int *source_port*, u_int32_t *dest_ip*, int *dest_port*, unsigned char * *dest_arp*, int *ifindex*)

Definition at line 119 of file packet.c.

References checksum(), DEBUG, and LOG_ERR.

Referenced by send_discover(), send_renew(), and send_selecting().

```
121 {
122         int fd;
123         int result;
124         struct sockaddr_ll dest;
125         struct udp_dhcp_packet packet;
126 
127         if ((fd = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_IP))) < 0) {
128 
129                 DEBUG(LOG_ERR, "socket call failed: %s", strerror(errno));
130                 return -1;
131         }
132 
133         memset(&dest, 0, sizeof(dest));
134         memset(&packet, 0, sizeof(packet));
135 
136         dest.sll_family = AF_PACKET;
137         dest.sll_protocol = htons(ETH_P_IP);
138         dest.sll_ifindex = ifindex;
139         dest.sll_halen = 6;
140         memcpy(dest.sll_addr, dest_arp, 6);
141         if (bind(fd, (struct sockaddr *)&dest, sizeof(struct sockaddr_ll)) < 0) {
142                 DEBUG(LOG_ERR, "bind call failed: %s", strerror(errno));
143                 close(fd);
144                 return -1;
145         }
146 
147         packet.ip.protocol = IPPROTO_UDP;
148         packet.ip.saddr = source_ip;
149         packet.ip.daddr = dest_ip;
150         packet.udp.source = htons(source_port);
151         packet.udp.dest = htons(dest_port);
152         packet.udp.len = htons(sizeof(packet.udp) + sizeof(struct dhcpMessage)); /* cheat on the psuedo-header */
153         packet.ip.tot_len = packet.udp.len;
154         memcpy(&(packet.data), payload, sizeof(struct dhcpMessage));
155         packet.udp.check = checksum(&packet, sizeof(struct udp_dhcp_packet));
156 
157         packet.ip.tot_len = htons(sizeof(struct udp_dhcp_packet));
158         packet.ip.ihl = sizeof(packet.ip) >> 2;
159         packet.ip.version = IPVERSION;
160         packet.ip.ttl = IPDEFTTL;
161         packet.ip.check = checksum(&(packet.ip), sizeof(packet.ip));
162 
163         result = sendto(fd, &packet, sizeof(struct udp_dhcp_packet), 0, (struct sockaddr *) &dest, sizeof(dest));
164         if (result <= 0) {
165 
166                 DEBUG(LOG_ERR, "write on socket failed: %s", strerror(errno));
167         }
168         close(fd);
169         return result;
170 }
```

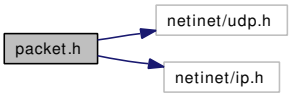
Here is the call graph for this function:

4.20 packet.h File Reference

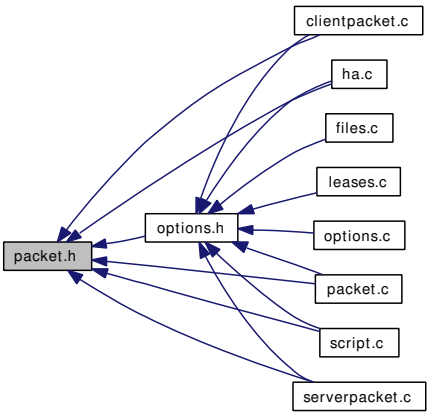
`#include <netinet/udp.h>`

`#include <netinet/ip.h>`

Include dependency graph for packet.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct **dhcpMessage**
- struct **udp_dhcp_packet**

Functions

- void **init_header** (struct **dhcpMessage** *packet, char type)
- int **get_packet** (struct **dhcpMessage** *packet, int fd)
- u_int16_t **checksum** (void *addr, int count)
- int **raw_packet** (struct **dhcpMessage** *payload, u_int32_t source_ip, int source_port, u_int32_t dest_ip, int dest_port, unsigned char *dest_arp, int ifindex)
- int **kernel_packet** (struct **dhcpMessage** *payload, u_int32_t source_ip, int source_port, u_int32_t dest_ip, int dest_port)

4.20.1 Function Documentation

4.20.1.1 u_int16_t checksum (void * *addr*, int *count*)

Definition at line 88 of file packet.c.

Referenced by get_raw_packet(), and raw_packet().

```
89 {
90         /* Compute Internet Checksum for "count" bytes
91          *         beginning at location "addr".
92          */
93         register int32_t sum = 0;
94         u_int16_t *source = (u_int16_t *) addr;
95 
96         while (count > 1)  {
97                 /*  This is the inner loop */
98                 sum += *source++;
99                 count -= 2;
100         }
101 
102         /*  Add left-over byte, if any */
103         if (count > 0) {
104                 /* Make sure that the left-over byte is added correctly both
105                  * with little and big endian hosts */
106                 u_int16_t tmp = 0;
107                 *(unsigned char *) (&tmp) = * (unsigned char *) source;
108                 sum += tmp;
109         }
110         /*  Fold 32-bit sum to 16 bits */
111         while (sum >> 16)
112                 sum = (sum & 0xffff) + (sum >> 16);
113 
114         return ~sum;
115 }
```

4.20.1.2 int get_packet (struct dhcpMessage * *packet*, int *fd*)

Definition at line 47 of file packet.c.

References BOOTREQUEST, BROADCAST_FLAG, dhcpMessage::cookie, DEBUG, DHCP_-MAGIC, DHCP_VENDOR, dhcpMessage::flags, get_option(), LOG, LOG_ERR, LOG_INFO, dhcpMessage::op, and OPT_LEN.

Referenced by udhcp().

```
48 {
49         int bytes;
50         int i;
51         const char broken_vendors[][8] = {
52                 "MSFT 98",
53                 ""
54         };
55         char unsigned *vendor;
56 
57         memset(packet, 0, sizeof(struct dhcpMessage));
58         bytes = read(fd, packet, sizeof(struct dhcpMessage));
59         if (bytes < 0) {
60 
61                 DEBUG(LOG_INFO, "couldn't read on listening socket, ignoring");
62                 return -1;
63         }
64 
65         if (ntohl(packet->cookie) != DHCP_MAGIC) {
66                 LOG(LOG_ERR, "received bogus message, ignoring");
67                 return -2;
68         }
69 
70         DEBUG(LOG_INFO, "Received a packet");
71 
72         if (packet->op == BOOTREQUEST && (vendor = get_option(packet, DHCP_VENDOR))) {
73                 for (i = 0; broken_vendors[i][0]; i++) {
```

```
74                          if (vendor[OPT_LEN - 2] == (unsigned char) strlen(broken_vendors[i]) &&
75                              !strncmp(vendor, broken_vendors[i], vendor[OPT_LEN - 2])) {
76                                  DEBUG(LOG_INFO, "broken client (%s), forcing broadcast",
77                                          broken_vendors[i]);
78                                  packet->flags |= htons(BROADCAST_FLAG);
79                          }
80                  }
81          }
82
83
84          return bytes;
85  }
```

Here is the call graph for this function:

4.20.1.3 void init_header (struct dhcpMessage * *packet*, char *type*)

Definition at line 23 of file packet.c.

References add_simple_option(), BOOTREPLY, BOOTREQUEST, dhcpMessage::cookie, DHCP_END, DHCP_MAGIC, DHCP_MESSAGE_TYPE, DHCPACK, DHCPDISCOVER, DHCPINFORM, DHCPNAK, DHCPOFFER, DHCPRELEASE, DHCPREQUEST, ETH_10MB, ETH_10MB_LEN, dhcpMessage::hlen, dhcpMessage::htype, dhcpMessage::op, and dhcpMessage::options.

```
24 {
25          memset(packet, 0, sizeof(struct dhcpMessage));
26          switch (type) {
27          case DHCPDISCOVER:
28          case DHCPREQUEST:
29          case DHCPRELEASE:
30          case DHCPINFORM:
31                  packet->op = BOOTREQUEST;
32                  break;
33          case DHCPOFFER:
34          case DHCPACK:
35          case DHCPNAK:
36                  packet->op = BOOTREPLY;
37          }
38          packet->htype = ETH_10MB;
39          packet->hlen = ETH_10MB_LEN;
40          packet->cookie = htonl(DHCP_MAGIC);
41          packet->options[0] = DHCP_END;
42          add_simple_option(packet->options, DHCP_MESSAGE_TYPE, type);
43  }
```

Here is the call graph for this function:

4.20.1.4 int kernel_packet (struct dhcpMessage * *payload*, u_int32_t *source_ip*, int *source_port*, u_int32_t *dest_ip*, int *dest_port*)

Definition at line 174 of file packet.c.

Referenced by send_release(), and send_renew().

```
176 {
177         int n = 1;
178         int fd, result;
179         struct sockaddr_in client;
180 
181         if ((fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
182                 return -1;
183 
184         if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char *) &n, sizeof(n)) == -1)
185                 return -1;
186 
187         memset(&client, 0, sizeof(client));
188         client.sin_family = AF_INET;
189         client.sin_port = htons(source_port);
190         client.sin_addr.s_addr = source_ip;
191 
192         if (bind(fd, (struct sockaddr *)&client, sizeof(struct sockaddr)) == -1)
193                 return -1;
194 
195         memset(&client, 0, sizeof(client));
196         client.sin_family = AF_INET;
197         client.sin_port = htons(dest_port);
198         client.sin_addr.s_addr = dest_ip;
199 
200         if (connect(fd, (struct sockaddr *)&client, sizeof(struct sockaddr)) == -1)
201                 return -1;
202 
203         result = write(fd, payload, sizeof(struct dhcpMessage));
204         close(fd);
205 
206         return result;
207 }
```

4.20.1.5 int raw_packet (struct dhcpMessage * *payload*, u_int32_t *source_ip*, int *source_port*, u_int32_t *dest_ip*, int *dest_port*, unsigned char * *dest_arp*, int *ifindex*)

Definition at line 119 of file packet.c.

References checksum(), DEBUG, and LOG_ERR.

Referenced by send_discover(), send_renew(), and send_selecting().

```
121 {
122         int fd;
123         int result;
124         struct sockaddr_ll dest;
125         struct udp_dhcp_packet packet;
126 
127         if ((fd = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_IP))) < 0) {
128 
129                 DEBUG(LOG_ERR, "socket call failed: %s", strerror(errno));
130                 return -1;
131         }
132 
133         memset(&dest, 0, sizeof(dest));
134         memset(&packet, 0, sizeof(packet));
135 
136         dest.sll_family = AF_PACKET;
137         dest.sll_protocol = htons(ETH_P_IP);
138         dest.sll_ifindex = ifindex;
139         dest.sll_halen = 6;
```

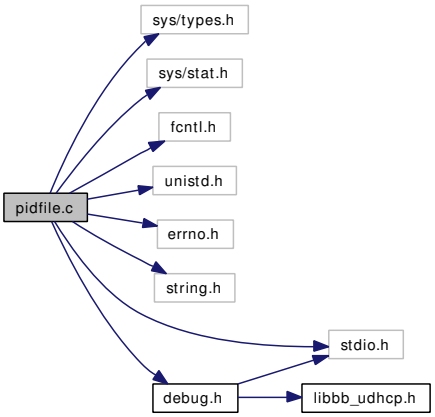
```
140         memcpy(dest.sll_addr, dest_arp, 6);
141         if (bind(fd, (struct sockaddr *)&dest, sizeof(struct sockaddr_ll)) < 0) {
142                 DEBUG(LOG_ERR, "bind call failed: %s", strerror(errno));
143                 close(fd);
144                 return -1;
145         }
146 
147         packet.ip.protocol = IPPROTO_UDP;
148         packet.ip.saddr = source_ip;
149         packet.ip.daddr = dest_ip;
150         packet.udp.source = htons(source_port);
151         packet.udp.dest = htons(dest_port);
152         packet.udp.len = htons(sizeof(packet.udp) + sizeof(struct dhcpMessage)); /* cheat on the psuedo-header */
153         packet.ip.tot_len = packet.udp.len;
154         memcpy(&(packet.data), payload, sizeof(struct dhcpMessage));
155         packet.udp.check = checksum(&packet, sizeof(struct udp_dhcp_packet));
156 
157         packet.ip.tot_len = htons(sizeof(struct udp_dhcp_packet));
158         packet.ip.ihl = sizeof(packet.ip) >> 2;
159         packet.ip.version = IPVERSION;
160         packet.ip.ttl = IPDEFTTL;
161         packet.ip.check = checksum(&(packet.ip), sizeof(packet.ip));
162 
163         result = sendto(fd, &packet, sizeof(struct udp_dhcp_packet), 0, (struct sockaddr *) &dest, sizeof(dest));
164         if (result <= 0) {
165 
166                 DEBUG(LOG_ERR, "write on socket failed: %s", strerror(errno));
167         }
168         close(fd);
169         return result;
170 }
```

Here is the call graph for this function:

4.21 pidfile.c File Reference

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include "debug.h"
```

Include dependency graph for pidfile.c:



Functions

- int **pidfile_acquire** (char *pidfile)
- void **pidfile_write_release** (int pid_fd)
- void **pidfile_delete** (char *pidfile)

4.21.1 Function Documentation

4.21.1.1 int pidfile_acquire (char * *pidfile*)

Definition at line 32 of file pidfile.c.

References LOG, and LOG_ERR.

Referenced by udhcp().

```
33 {
34         int pid_fd;
35         if (pidfile == NULL) return -1;
36
37         pid_fd = open(pidfile, O_CREAT | O_WRONLY, 0644);
38         if (pid_fd < 0) {
39                 LOG(LOG_ERR, "Unable to open pidfile %s: %s\n",
```

```
40                      pidfile, strerror(errno));
41         } else {
42                 lockf(pid_fd, F_LOCK, 0);
43         }
44
45         return pid_fd;
46 }
```

4.21.1.2 void pidfile_delete (char * *pidfile*)

Definition at line 64 of file pidfile.c.

```
65 {
66         if (pidfile) unlink(pidfile);
67 }
```

4.21.1.3 void pidfile_write_release (int *pid_fd*)

Definition at line 49 of file pidfile.c.

Referenced by udhcp().

```
50 {
51         FILE *out;
52
53         if (pid_fd < 0) return;
54
55         if ((out = fdopen(pid_fd, "w")) != NULL) {
56                 fprintf(out, "%d\n", getpid());
57                 fclose(out);
58         }
59         lockf(pid_fd, F_UNLCK, 0);
60         close(pid_fd);
61 }
```

4.22 pidfile.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int **pidfile_acquire** (char *pidfile)
- void **pidfile_write_release** (int pid_fd)
- void **pidfile_delete** (char *pidfile)

4.22.1 Function Documentation

4.22.1.1 int pidfile_acquire (char * *pidfile*)

Definition at line 32 of file pidfile.c.

References LOG, and LOG_ERR.

Referenced by udhcp().

```
33 {
34         int pid_fd;
35         if (pidfile == NULL) return -1;
36
37         pid_fd = open(pidfile, O_CREAT | O_WRONLY, 0644);
38         if (pid_fd < 0) {
39                 LOG(LOG_ERR, "Unable to open pidfile %s: %s\n",
40                     pidfile, strerror(errno));
41         } else {
42                 lockf(pid_fd, F_LOCK, 0);
43         }
44
45         return pid_fd;
46 }
```

4.22.1.2 void pidfile_delete (char * *pidfile*)

Definition at line 64 of file pidfile.c.

```
65 {
66         if (pidfile) unlink(pidfile);
67 }
```

4.22.1.3 void pidfile_write_release (int *pid_fd*)

Definition at line 49 of file pidfile.c.

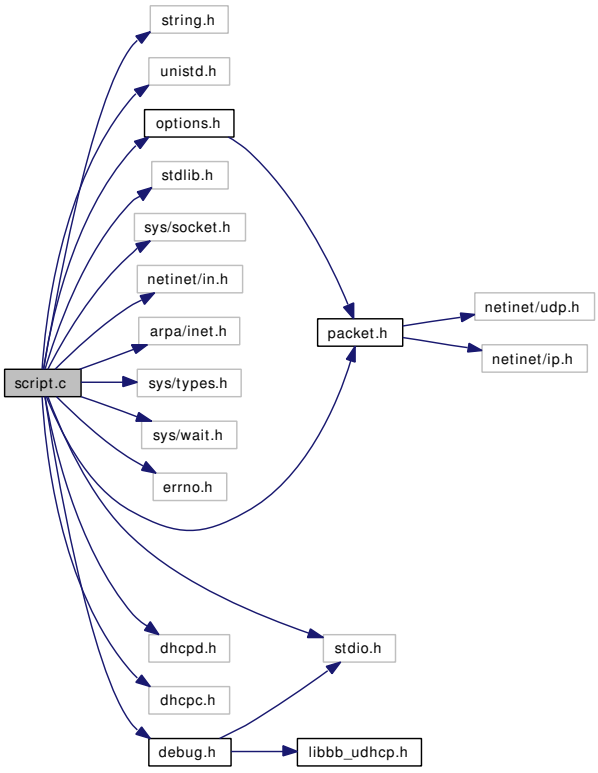
Referenced by udhcp().

```
50 {
51         FILE *out;
52
53         if (pid_fd < 0) return;
54
55         if ((out = fdopen(pid_fd, "w")) != NULL) {
56                 fprintf(out, "%d\n", getpid());
57                 fclose(out);
58         }
59         lockf(pid_fd, F_UNLCK, 0);
60         close(pid_fd);
61 }
```

4.23 script.c File Reference

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include "options.h"
#include "dhcpd.h"
#include "dhcpc.h"
#include "packet.h"
#include "debug.h"
```

Include dependency graph for script.c:



Functions

- void **run_script** (struct **dhcpMessage** *packet, const char *name)

4.23.1 Function Documentation

4.23.1.1 void run_script (struct dhcpMessage * *packet*, const char * *name*)

Definition at line 202 of file script.c.

References client_config, DEBUG, LOG, LOG_ERR, LOG_INFO, and client_config_t::script.

Referenced by udhcp().

```
203 {
204         int pid;
205         char **envp;
206 
207         if (client_config.script == NULL)
208                 return;
209 
210         /* call script */
211         pid = fork();
212         if (pid) {
213                 waitpid(pid, NULL, 0);
214                 return;
215         } else if (pid == 0) {
216                 envp = fill_envp(packet);
217 
218                 /* close fd's? */
219 
220                 /* exec script */
221                 DEBUG(LOG_INFO, "execle'ing %s", client_config.script);
222                 execle(client_config.script, client_config.script,
223                        name, NULL, envp);
224                 LOG(LOG_ERR, "script %s failed: %s",
225                     client_config.script, strerror(errno));
226                 exit(1);
227         }
228 }
```

4.24 script.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void **run_script** (struct **dhcpMessage** *packet, const char *name)

4.24.1 Function Documentation

4.24.1.1 void run_script (struct dhcpMessage * *packet*, const char * *name*)

Definition at line 202 of file script.c.

References client_config, DEBUG, LOG, LOG_ERR, LOG_INFO, and client_config_t::script.

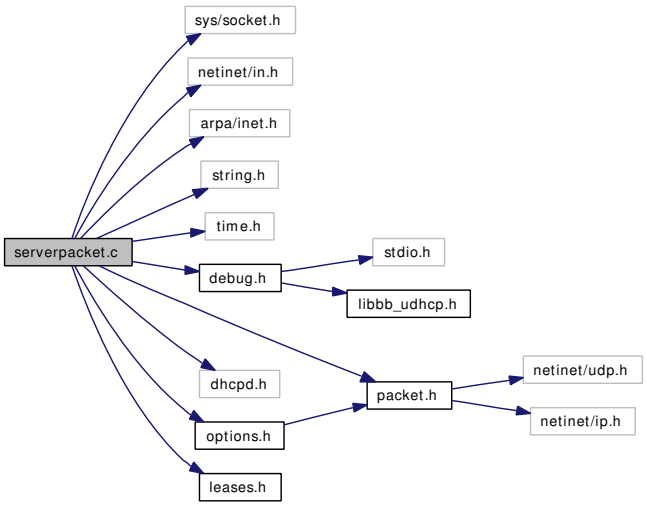
Referenced by udhcp().

```
203 {
204         int pid;
205         char **envp;
206
207         if (client_config.script == NULL)
208                 return;
209
210         /* call script */
211         pid = fork();
212         if (pid) {
213                 waitpid(pid, NULL, 0);
214                 return;
215         } else if (pid == 0) {
216                 envp = fill_envp(packet);
217
218                 /* close fd's? */
219
220                 /* exec script */
221                 DEBUG(LOG_INFO, "execle'ing %s", client_config.script);
222                 execle(client_config.script, client_config.script,
223                        name, NULL, envp);
224                 LOG(LOG_ERR, "script %s failed: %s",
225                     client_config.script, strerror(errno));
226                 exit(1);
227         }
228 }
```

4.25 serverpacket.c File Reference

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <time.h>
#include "packet.h"
#include "debug.h"
#include "dhcpd.h"
#include "options.h"
#include "leases.h"
```

Include dependency graph for serverpacket.c:



Functions

- int **sendOffer** (struct **dhcpMessage** *oldpacket)
- int **sendNAK** (struct **dhcpMessage** *oldpacket)
- int **sendACK** (struct **dhcpMessage** *oldpacket, u_int32_t yiaddr)
- int **send_inform** (struct **dhcpMessage** *oldpacket)

4.25.1 Function Documentation

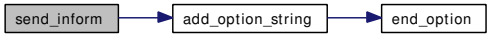
4.25.1.1 int send_inform (struct dhcpMessage * *oldpacket*)

Definition at line 243 of file serverpacket.c.

References add_option_string(), DHCP_LEASE_TIME, DHCPACK, OPT_CODE, and dhcpMessage::options.

```
244 {
245         struct dhcpMessage packet;
246         struct option_set *curr;
247 
248         init_packet(&packet, oldpacket, DHCPACK);
249 
250         curr = server_config.options;
251         while (curr) {
252                 if (curr->data[OPT_CODE] != DHCP_LEASE_TIME)
253                         add_option_string(packet.options, curr->data);
254                 curr = curr->next;
255         }
256 
257         add_bootp_options(&packet);
258 
259         return send_packet(&packet, 0);
260 }
```

Here is the call graph for this function:



4.25.1.2 int sendACK (struct dhcpMessage * *oldpacket*, u_int32_t *yiaddr*)

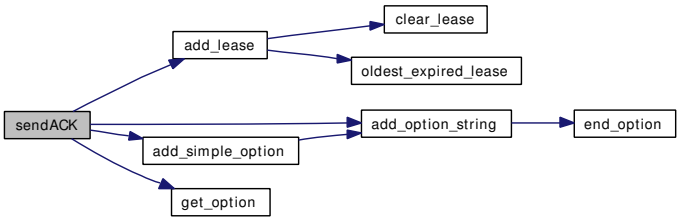
Definition at line 200 of file serverpacket.c.

References add_lease(), add_option_string(), add_simple_option(), dhcpMessage::chaddr, DHCP_LEASE_TIME, DHCPACK, get_option(), LOG, LOG_INFO, OPT_CODE, dhcpMessage::options, and dhcpMessage::yiaddr.

```
201 {
202         struct dhcpMessage packet;
203         struct option_set *curr;
204         unsigned char *lease_time;
205         u_int32_t lease_time_align = server_config.lease;
206         struct in_addr addr;
207 
208         init_packet(&packet, oldpacket, DHCPACK);
209         packet.yiaddr = yiaddr;
210 
211         if ((lease_time = get_option(oldpacket, DHCP_LEASE_TIME))) {
212                 memcpy(&lease_time_align, lease_time, 4);
213                 lease_time_align = ntohl(lease_time_align);
214                 if (lease_time_align > server_config.lease)
215                         lease_time_align = server_config.lease;
216                 else if (lease_time_align < server_config.min_lease)
217                         lease_time_align = server_config.lease;
218         }
219 
220         add_simple_option(packet.options, DHCP_LEASE_TIME, htonl(lease_time_align));
221 
222         curr = server_config.options;
223         while (curr) {
224                 if (curr->data[OPT_CODE] != DHCP_LEASE_TIME)
225                         add_option_string(packet.options, curr->data);
226                 curr = curr->next;
227         }
228 
229         add_bootp_options(&packet);
230 
231         addr.s_addr = packet.yiaddr;
```

```
232         LOG(LOG_INFO, "sending ACK to %s", inet_ntoa(addr));
233
234         if (send_packet(&packet, 0) < 0)
235                 return -1;
236
237         add_lease(packet.chaddr, packet.yiaddr, lease_time_align);
238
239         return 0;
240 }
```

Here is the call graph for this function:



4.25.1.3 int sendNAK (struct dhcpMessage * *oldpacket*)

Definition at line 189 of file serverpacket.c.

References DEBUG, DHCPNAK, and LOG_INFO.

```
190 {
191         struct dhcpMessage packet;
192
193         init_packet(&packet, oldpacket, DHCPNAK);
194
195         DEBUG(LOG_INFO, "sending NAK");
196         return send_packet(&packet, 1);
197 }
```

4.25.1.4 int sendOffer (struct dhcpMessage * *oldpacket*)

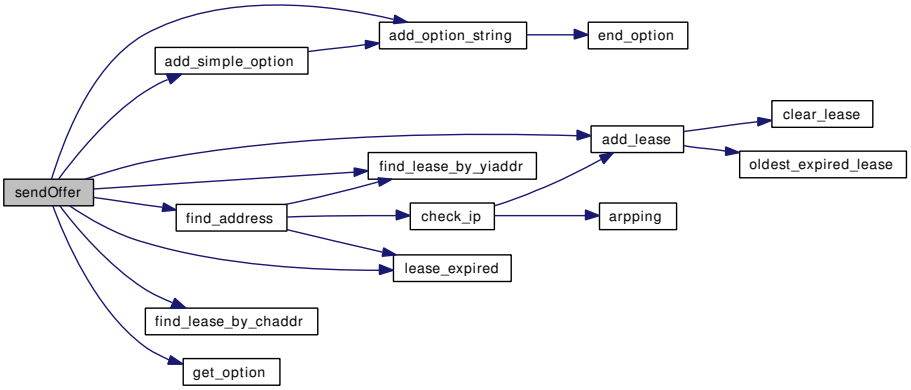
Definition at line 108 of file serverpacket.c.

References add_lease(), add_option_string(), add_simple_option(), dhcpMessage::chaddr, DHCP_LEASE_TIME, DHCP_REQUESTED_IP, DHCPOFFER, dhcpOfferedAddr::expires, find_address(), find_lease_by_chaddr(), find_lease_by_yiaddr(), get_option(), lease_expired(), LOG, LOG_INFO, LOG_WARNING, OPT_CODE, dhcpMessage::options, dhcpMessage::yiaddr, and dhcpOfferedAddr::yiaddr.

```
109 {
110         struct dhcpMessage packet;
111         struct dhcpOfferedAddr *lease = NULL;
112         u_int32_t req_align, lease_time_align = server_config.lease;
113         unsigned char *req, *lease_time;
114         struct option_set *curr;
115         struct in_addr addr;
116
117         init_packet(&packet, oldpacket, DHCPOFFER);
118
119         /* ADDME: if static, short circuit */
```

```
120         /* the client is in our lease/offered table */
121         if ((lease = find_lease_by_chaddr(oldpacket->chaddr))) {
122                 if (!lease_expired(lease))
123                         lease_time_align = lease->expires - time(0);
124                 packet.yiaddr = lease->yiaddr;
125 
126         /* Or the client has a requested ip */
127         } else if ((req = get_option(oldpacket, DHCP_REQUESTED_IP)) &&
128 
129                    /* Don't look here (ugly hackish thing to do) */
130                    memcpy(&req_align, req, 4) &&
131 
132                    /* and the ip is in the lease range */
133                    ntohl(req_align) >= ntohl(server_config.start) &&
134                    ntohl(req_align) <= ntohl(server_config.end) &&
135 
136                    /* and its not already taken/offered */ /* ADDME: check that its not a static lease */
137                    ((!(lease = find_lease_by_yiaddr(req_align)) ||
138 
139                    /* or its taken, but expired */ /* ADDME: or maybe in here */
140                    lease_expired(lease)))) {
141                                 packet.yiaddr = req_align; /* FIXME: oh my, is there a host using this IP? */
142 
143         /* otherwise, find a free IP */ /*ADDME: is it a static lease? */
144         } else {
145                 packet.yiaddr = find_address(0);
146 
147                 /* try for an expired lease */
148                 if (!packet.yiaddr) packet.yiaddr = find_address(1);
149         }
150 
151         if(!packet.yiaddr) {
152                 LOG(LOG_WARNING, "no IP addresses to give -- OFFER abandoned");
153                 return -1;
154         }
155 
156         if (!add_lease(packet.chaddr, packet.yiaddr, server_config.offer_time)) {
157                 LOG(LOG_WARNING, "lease pool is full -- OFFER abandoned");
158                 return -1;
159         }
160 
161         if ((lease_time = get_option(oldpacket, DHCP_LEASE_TIME))) {
162                 memcpy(&lease_time_align, lease_time, 4);
163                 lease_time_align = ntohl(lease_time_align);
164                 if (lease_time_align > server_config.lease)
165                         lease_time_align = server_config.lease;
166         }
167 
168         /* Make sure we aren't just using the lease time from the previous offer */
169         if (lease_time_align < server_config.min_lease)
170                 lease_time_align = server_config.lease;
171         /* ADDME: end of short circuit */
172         add_simple_option(packet.options, DHCP_LEASE_TIME, htonl(lease_time_align));
173 
174         curr = server_config.options;
175         while (curr) {
176                 if (curr->data[OPT_CODE] != DHCP_LEASE_TIME)
177                         add_option_string(packet.options, curr->data);
178                 curr = curr->next;
179         }
180 
181         add_bootp_options(&packet);
182 
183         addr.s_addr = packet.yiaddr;
184         LOG(LOG_INFO, "sending OFFER of %s", inet_ntoa(addr));
185         return send_packet(&packet, 0);
186 }
```

Here is the call graph for this function:



4.26 serverpacket.h File Reference

Functions

- int **sendOffer** (struct **dhcpMessage** *oldpacket)
- int **sendNAK** (struct **dhcpMessage** *oldpacket)
- int **sendACK** (struct **dhcpMessage** *oldpacket, u_int32_t yiaddr)
- int **send_inform** (struct **dhcpMessage** *oldpacket)

4.26.1 Function Documentation

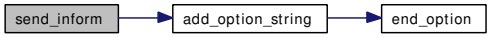
4.26.1.1 int send_inform (struct dhcpMessage * *oldpacket*)

Definition at line 243 of file serverpacket.c.

References add_option_string(), DHCP_LEASE_TIME, DHCPACK, OPT_CODE, and dhcpMessage::options.

```
244 {
245         struct dhcpMessage packet;
246         struct option_set *curr;
247 
248         init_packet(&packet, oldpacket, DHCPACK);
249 
250         curr = server_config.options;
251         while (curr) {
252                 if (curr->data[OPT_CODE] != DHCP_LEASE_TIME)
253                         add_option_string(packet.options, curr->data);
254                 curr = curr->next;
255         }
256 
257         add_bootp_options(&packet);
258 
259         return send_packet(&packet, 0);
260 }
```

Here is the call graph for this function:



4.26.1.2 int sendACK (struct dhcpMessage * *oldpacket*, u_int32_t *yiaddr*)

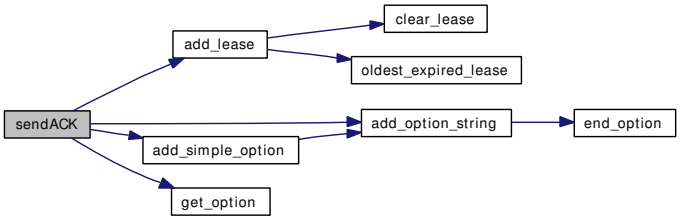
Definition at line 200 of file serverpacket.c.

References add_lease(), add_option_string(), add_simple_option(), dhcpMessage::chaddr, DHCP_LEASE_TIME, DHCPACK, get_option(), LOG, LOG_INFO, OPT_CODE, dhcpMessage::options, and dhcpMessage::yiaddr.

```
201 {
202         struct dhcpMessage packet;
203         struct option_set *curr;
204         unsigned char *lease_time;
205         u_int32_t lease_time_align = server_config.lease;
206         struct in_addr addr;
207 
```

```
208         init_packet(&packet, oldpacket, DHCPACK);
209         packet.yiaddr = yiaddr;
210 
211         if ((lease_time = get_option(oldpacket, DHCP_LEASE_TIME))) {
212                 memcpy(&lease_time_align, lease_time, 4);
213                 lease_time_align = ntohl(lease_time_align);
214                 if (lease_time_align > server_config.lease)
215                         lease_time_align = server_config.lease;
216                 else if (lease_time_align < server_config.min_lease)
217                         lease_time_align = server_config.lease;
218         }
219 
220         add_simple_option(packet.options, DHCP_LEASE_TIME, htonl(lease_time_align));
221 
222         curr = server_config.options;
223         while (curr) {
224                 if (curr->data[OPT_CODE] != DHCP_LEASE_TIME)
225                         add_option_string(packet.options, curr->data);
226                 curr = curr->next;
227         }
228 
229         add_bootp_options(&packet);
230 
231         addr.s_addr = packet.yiaddr;
232         LOG(LOG_INFO, "sending ACK to %s", inet_ntoa(addr));
233 
234         if (send_packet(&packet, 0) < 0)
235                 return -1;
236 
237         add_lease(packet.chaddr, packet.yiaddr, lease_time_align);
238 
239         return 0;
240 }
```

Here is the call graph for this function:



4.26.1.3 int sendNAK (struct dhcpMessage * *oldpacket*)

Definition at line 189 of file serverpacket.c.

References DEBUG, DHCPNAK, and LOG_INFO.

```
190 {
191         struct dhcpMessage packet;
192 
193         init_packet(&packet, oldpacket, DHCPNAK);
194 
195         DEBUG(LOG_INFO, "sending NAK");
196         return send_packet(&packet, 1);
197 }
```

4.26.1.4 int sendOffer (struct dhcpMessage * *oldpacket*)

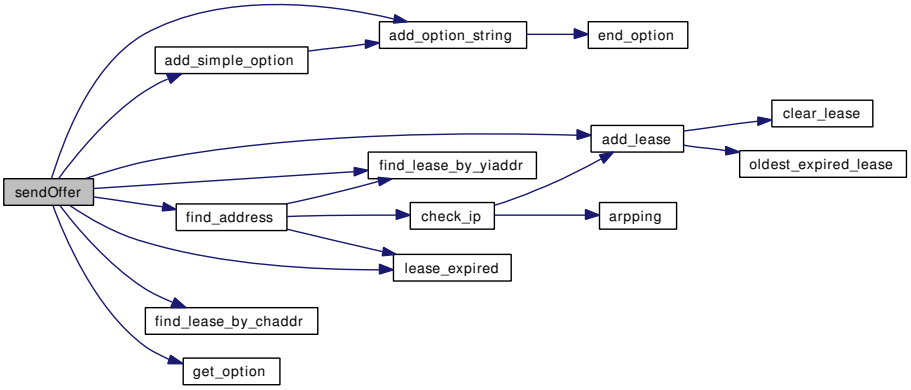
Definition at line 108 of file serverpacket.c.

References add_lease(), add_option_string(), add_simple_option(), dhcpMessage::chaddr, DHCP_LEASE_TIME, DHCP_REQUESTED_IP, DHCPOFFER, dhcpOfferedAddr::expires, find_address(), find_lease_by_chaddr(), find_lease_by_yiaddr(), get_option(), lease_expired(), LOG, LOG_INFO, LOG_WARNING, OPT_CODE, dhcpMessage::options, dhcpOfferedAddr::yiaddr, and dhcpMessage::yiaddr.

```
109 {
110         struct dhcpMessage packet;
111         struct dhcpOfferedAddr *lease = NULL;
112         u_int32_t req_align, lease_time_align = server_config.lease;
113         unsigned char *req, *lease_time;
114         struct option_set *curr;
115         struct in_addr addr;
116 
117         init_packet(&packet, oldpacket, DHCPOFFER);
118 
119         /* ADDME: if static, short circuit */
120         /* the client is in our lease/offered table */
121         if ((lease = find_lease_by_chaddr(oldpacket->chaddr))) {
122                 if (!lease_expired(lease))
123                         lease_time_align = lease->expires - time(0);
124                 packet.yiaddr = lease->yiaddr;
125 
126         /* Or the client has a requested ip */
127         } else if ((req = get_option(oldpacket, DHCP_REQUESTED_IP)) &&
128 
129                    /* Don't look here (ugly hackish thing to do) */
130                    memcpy(&req_align, req, 4) &&
131 
132                    /* and the ip is in the lease range */
133                    ntohl(req_align) >= ntohl(server_config.start) &&
134                    ntohl(req_align) <= ntohl(server_config.end) &&
135 
136                    /* and its not already taken/offered */ /* ADDME: check that its not a static lease */
137                    ((!(lease = find_lease_by_yiaddr(req_align)) ||
138 
139                    /* or its taken, but expired */ /* ADDME: or maybe in here */
140                    lease_expired(lease)))) {
141                                 packet.yiaddr = req_align; /* FIXME: oh my, is there a host using this IP? */
142 
143         /* otherwise, find a free IP */ /*ADDME: is it a static lease? */
144         } else {
145                 packet.yiaddr = find_address(0);
146 
147                 /* try for an expired lease */
148                 if (!packet.yiaddr) packet.yiaddr = find_address(1);
149         }
150 
151         if(!packet.yiaddr) {
152                 LOG(LOG_WARNING, "no IP addresses to give -- OFFER abandoned");
153                 return -1;
154         }
155 
156         if (!add_lease(packet.chaddr, packet.yiaddr, server_config.offer_time)) {
157                 LOG(LOG_WARNING, "lease pool is full -- OFFER abandoned");
158                 return -1;
159         }
160 
161         if ((lease_time = get_option(oldpacket, DHCP_LEASE_TIME))) {
162                 memcpy(&lease_time_align, lease_time, 4);
163                 lease_time_align = ntohl(lease_time_align);
164                 if (lease_time_align > server_config.lease)
```

```
165                         lease_time_align = server_config.lease;
166         }
167
168         /* Make sure we aren't just using the lease time from the previous offer */
169         if (lease_time_align < server_config.min_lease)
170                 lease_time_align = server_config.lease;
171         /* ADDME: end of short circuit */
172         add_simple_option(packet.options, DHCP_LEASE_TIME, htonl(lease_time_align));
173
174         curr = server_config.options;
175         while (curr) {
176                 if (curr->data[OPT_CODE] != DHCP_LEASE_TIME)
177                         add_option_string(packet.options, curr->data);
178                 curr = curr->next;
179         }
180
181         add_bootp_options(&packet);
182
183         addr.s_addr = packet.yiaddr;
184         LOG(LOG_INFO, "sending OFFER of %s", inet_ntoa(addr));
185         return send_packet(&packet, 0);
186 }
```

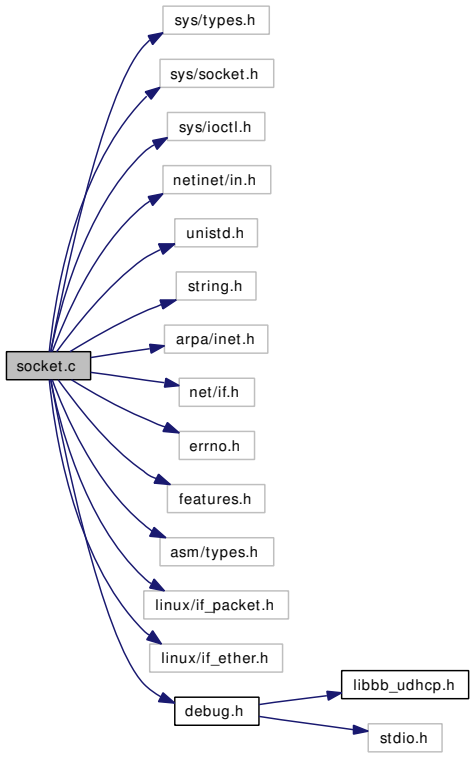
Here is the call graph for this function:



4.27 socket.c File Reference

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <errno.h>
#include <features.h>
#include <asm/types.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include "debug.h"
```

Include dependency graph for socket.c:



Functions

- int **read_interface** (char *interface, int *ifindex, u_int32_t *addr, unsigned char *arp)
- int **listen_socket** (unsigned int ip, int port, char *inf)
- int **raw_socket** (int ifindex)

4.27.1 Function Documentation

4.27.1.1 int listen_socket (unsigned int *ip*, int *port*, char * *inf*)

Definition at line 93 of file socket.c.

References DEBUG, LOG_ERR, and LOG_INFO.

Referenced by udhcp().

```
94 {
95         struct ifreq interface;
96         int fd;
97         struct sockaddr_in addr;
98         int n = 1;
99
100         DEBUG(LOG_INFO, "Opening listen socket on 0x%08x:%d %s\n", ip, port, inf);
101         if ((fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
102                 DEBUG(LOG_ERR, "socket call failed: %s", strerror(errno));
103                 return -1;
104         }
105
106         memset(&addr, 0, sizeof(addr));
107         addr.sin_family = AF_INET;
108         addr.sin_port = htons(port);
109         addr.sin_addr.s_addr = ip;
110
111         if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char *) &n, sizeof(n)) == -1) {
112                 close(fd);
113                 return -1;
114         }
115         if (setsockopt(fd, SOL_SOCKET, SO_BROADCAST, (char *) &n, sizeof(n)) == -1) {
116                 close(fd);
117                 return -1;
118         }
119
120         strncpy(interface.ifr_ifrn.ifrn_name, inf, IFNAMSIZ);
121         if (setsockopt(fd, SOL_SOCKET, SO_BINDTODEVICE,(char *)&interface, sizeof(interface)) < 0) {
122                 close(fd);
123                 return -1;
124         }
125
126         if (bind(fd, (struct sockaddr *)&addr, sizeof(struct sockaddr)) == -1) {
127                 close(fd);
128                 return -1;
129         }
130
131         return fd;
132 }
```

4.27.1.2 int raw_socket (int *ifindex*)

Definition at line 135 of file socket.c.

References DEBUG, LOG_ERR, and LOG_INFO.

Referenced by udhcp().

```
136 {
137         int fd;
138         struct sockaddr_ll sock;
139 
140         DEBUG(LOG_INFO, "Opening raw socket on ifindex %d\n", ifindex);
141         if ((fd = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_IP))) < 0) {
142                 DEBUG(LOG_ERR, "socket call failed: %s", strerror(errno));
143                 return -1;
144         }
145 
146         sock.sll_family = AF_PACKET;
147         sock.sll_protocol = htons(ETH_P_IP);
148         sock.sll_ifindex = ifindex;
149         if (bind(fd, (struct sockaddr *) &sock, sizeof(sock)) < 0) {
150                 DEBUG(LOG_ERR, "bind call failed: %s", strerror(errno));
151                 close(fd);
152                 return -1;
153         }
154 
155         return fd;
156 }
```

4.27.1.3 int read_interface (char * *interface*, int * *ifindex*, u_int32_t * *addr*, unsigned char * *arp*)

Definition at line 46 of file socket.c.

References DEBUG, LOG, LOG_ERR, and LOG_INFO.

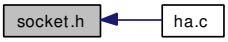
Referenced by udhcp().

```
47 {
48         int fd;
49         struct ifreq ifr;
50         struct sockaddr_in *our_ip;
51 
52         memset(&ifr, 0, sizeof(struct ifreq));
53         if((fd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) >= 0) {
54                 ifr.ifr_addr.sa_family = AF_INET;
55                 strcpy(ifr.ifr_name, interface);
56 
57                 if (addr) {
58                         if (ioctl(fd, SIOCGIFADDR, &ifr) == 0) {
59                                 our_ip = (struct sockaddr_in *) &ifr.ifr_addr;
60                                 *addr = our_ip->sin_addr.s_addr;
61                                 DEBUG(LOG_INFO, "%s (our ip) = %s", ifr.ifr_name, inet_ntoa(our_ip->sin_addr));
62                         } else {
63                                 LOG(LOG_ERR, "SIOCGIFADDR failed, is the interface up and configured?: %s",
64                                                 strerror(errno));
65                                 return -1;
66                         }
67                 }
68 
69                 if (ioctl(fd, SIOCGIFINDEX, &ifr) == 0) {
70                         DEBUG(LOG_INFO, "adapter index %d", ifr.ifr_ifindex);
71                         *ifindex = ifr.ifr_ifindex;
72                 } else {
73                         LOG(LOG_ERR, "SIOCGIFINDEX failed!: %s", strerror(errno));
74                         return -1;
75                 }
76                 if (ioctl(fd, SIOCGIFHWADDR, &ifr) == 0) {
77                         memcpy(arp, ifr.ifr_hwaddr.sa_data, 6);
```

```
78                          DEBUG(LOG_INFO, "adapter hardware address %02x:%02x:%02x:%02x:%02x:%02x",
79                                  arp[0], arp[1], arp[2], arp[3], arp[4], arp[5]);
80                  } else {
81                          LOG(LOG_ERR, "SIOCGIFHWADDR failed!: %s", strerror(errno));
82                          return -1;
83                  }
84          } else {
85                  LOG(LOG_ERR, "socket failed!: %s", strerror(errno));
86                  return -1;
87          }
88          close(fd);
89          return 0;
90  }
```

4.28 socket.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int **read_interface** (char *interface, int *ifindex, u_int32_t *addr, unsigned char *arp)
- int **listen_socket** (unsigned int ip, int port, char *inf)
- int **raw_socket** (int ifindex)

4.28.1 Function Documentation

4.28.1.1 int listen_socket (unsigned int *ip*, int *port*, char * *inf*)

Definition at line 93 of file socket.c.

References DEBUG, LOG_ERR, and LOG_INFO.

Referenced by udhcp().

```
94 {
95         struct ifreq interface;
96         int fd;
97         struct sockaddr_in addr;
98         int n = 1;
99 
100         DEBUG(LOG_INFO, "Opening listen socket on 0x%08x:%d %s\n", ip, port, inf);
101         if ((fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
102                 DEBUG(LOG_ERR, "socket call failed: %s", strerror(errno));
103                 return -1;
104         }
105 
106         memset(&addr, 0, sizeof(addr));
107         addr.sin_family = AF_INET;
108         addr.sin_port = htons(port);
109         addr.sin_addr.s_addr = ip;
110 
111         if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char *) &n, sizeof(n)) == -1) {
112                 close(fd);
113                 return -1;
114         }
115         if (setsockopt(fd, SOL_SOCKET, SO_BROADCAST, (char *) &n, sizeof(n)) == -1) {
116                 close(fd);
117                 return -1;
118         }
119 
120         strncpy(interface.ifr_ifrn.ifrn_name, inf, IFNAMSIZ);
121         if (setsockopt(fd, SOL_SOCKET, SO_BINDTODEVICE,(char *)&interface, sizeof(interface)) < 0) {
122                 close(fd);
123                 return -1;
124         }
125 
126         if (bind(fd, (struct sockaddr *)&addr, sizeof(struct sockaddr)) == -1) {
127                 close(fd);
128                 return -1;
129         }
130 
```

```
131         return fd;
132 }
```

4.28.1.2 int raw_socket (int *ifindex*)

Definition at line 135 of file socket.c.

References DEBUG, LOG_ERR, and LOG_INFO.

Referenced by udhcp().

```
136 {
137         int fd;
138         struct sockaddr_ll sock;
139 
140         DEBUG(LOG_INFO, "Opening raw socket on ifindex %d\n", ifindex);
141         if ((fd = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_IP))) < 0) {
142                 DEBUG(LOG_ERR, "socket call failed: %s", strerror(errno));
143                 return -1;
144         }
145 
146         sock.sll_family = AF_PACKET;
147         sock.sll_protocol = htons(ETH_P_IP);
148         sock.sll_ifindex = ifindex;
149         if (bind(fd, (struct sockaddr *) &sock, sizeof(sock)) < 0) {
150                 DEBUG(LOG_ERR, "bind call failed: %s", strerror(errno));
151                 close(fd);
152                 return -1;
153         }
154 
155         return fd;
156 }
```

4.28.1.3 int read_interface (char * *interface*, int * *ifindex*, u_int32_t * *addr*, unsigned char * *arp*)

Definition at line 46 of file socket.c.

References DEBUG, LOG, LOG_ERR, and LOG_INFO.

Referenced by udhcp().

```
47 {
48         int fd;
49         struct ifreq ifr;
50         struct sockaddr_in *our_ip;
51 
52         memset(&ifr, 0, sizeof(struct ifreq));
53         if((fd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) >= 0) {
54                 ifr.ifr_addr.sa_family = AF_INET;
55                 strcpy(ifr.ifr_name, interface);
56 
57                 if (addr) {
58                         if (ioctl(fd, SIOCGIFADDR, &ifr) == 0) {
59                                 our_ip = (struct sockaddr_in *) &ifr.ifr_addr;
60                                 *addr = our_ip->sin_addr.s_addr;
61                                 DEBUG(LOG_INFO, "%s (our ip) = %s", ifr.ifr_name, inet_ntoa(our_ip->sin_addr));
62                         } else {
63                                 LOG(LOG_ERR, "SIOCGIFADDR failed, is the interface up and configured?: %s",
64                                                 strerror(errno));
65                                 return -1;
```

```
66                      }
67              }
68
69              if (ioctl(fd, SIOCGIFINDEX, &ifr) == 0) {
70                      DEBUG(LOG_INFO, "adapter index %d", ifr.ifr_ifindex);
71                      *ifindex = ifr.ifr_ifindex;
72              } else {
73                      LOG(LOG_ERR, "SIOCGIFINDEX failed!: %s", strerror(errno));
74                      return -1;
75              }
76              if (ioctl(fd, SIOCGIFHWADDR, &ifr) == 0) {
77                      memcpy(arp, ifr.ifr_hwaddr.sa_data, 6);
78                      DEBUG(LOG_INFO, "adapter hardware address %02x:%02x:%02x:%02x:%02x:%02x",
79                              arp[0], arp[1], arp[2], arp[3], arp[4], arp[5]);
80              } else {
81                      LOG(LOG_ERR, "SIOCGIFHWADDR failed!: %s", strerror(errno));
82                      return -1;
83              }
84      } else {
85              LOG(LOG_ERR, "socket failed!: %s", strerror(errno));
86              return -1;
87      }
88      close(fd);
89      return 0;
90 }
```

Index