TLDNS Relay

1.0

Generated by Doxygen 1.11.0

Chapter 1

TLDNS Relay

1.1 System Function Design

1.1.1 Basic Tasks

Design a DNS relay server program that reads a ”Domain Name - IP Address” mapping table. When a client queries the IP address corresponding to a domain name, the domain name is searched in the table, resulting in three possible outcomes:

- If the result is an IP address 0.0.0.0, return an error message ”Domain name does not exist” to the client instead of returning the IP address 0.0.0.0, implementing a malicious website blocking function.
- If the result is a regular IP address, return this address to the client, implementing DNS server functionality.
- If the domain name is not found in the table, send the query to an Internet DNS server and return the result to the client, implementing DNS relay functionality.

The implementation must adhere to the DNS protocol specifications to ensure interoperability with Windows and other systems.

Notes:

1. Concurrent Clients: Allow concurrent queries from multiple clients (which may be on different computers). This means processing another client's query request even if the first query has not been answered yet (the role of the ID field in the DNS protocol header), requiring message ID translation.
2. Timeout Handling: Consider the unreliability of UDP, and handle situations where the external DNS server (relay) does not respond or responds late.

1.1.2 Additional Functions

- Implement LRU mechanism for Cache.
- Optimize the dictionary lookup algorithm.
- Ensure consistent performance across Windows/Linux source code.

1.1.3 Extra Features

- Support for IPv6.
- Cross-platform support for Windows/Linux/MacOS.
- Implement high-performance querying using an event-driven, non-blocking asynchronous I/O model.
- Implement query pools and index pools to support concurrent queries.
- Support multiple message types, including A, CNAME, SOA, MX, and AAAA.
- Provide command-line argument parsing and help documentation.

1.2 Quick Start

1.2.1 Quick Start

1. Clone the repository locally and navigate into it:
   ```
   git clone https://github.com/Word2VecT/TLDNS-Relay.git
   cd TLDNS-Relay
   ```
2. Download and install libuv.
3. Import the project folder in CLion, compile, and run.
4. Set your DNS to 127.0.0.1.
5. Enjoy!

1.2.2 Program Help

Use the -h parameter to view the program help documentation.

```
Usage:
[-a] Use the specified name server
[-d] Debug level mask, a 4-bit binary number, DEBUG, INFO, ERROR, FATAL in order
[-f] Use the specified DNS hosts file
[-l] Log information storage location
[-p] Custom listening ports
[-h] Helpful Information

Example:
-d 1111 -a 192.168.0.1 -f c:\dns-table.txt
Output all debugging information
Use the specified name server 192.168.0.1
Use the specified configuration file c:\dns-table.txt

-d 1101 -l /Users/Code -p 53
Output DEBUG, INFO, and FATAL information
Output debugging information to /Users/Code as a file
```

1.3 Reference

- Domain names - concepts and facilities. RFC 1034, RFC Editor, November 1987, DOI: 10.↩17487/RFC1034. 55 pages. Abstract: This RFC is the revised basic definition of The Domain Name System. It obsoletes RFC-882. This memo describes the domain style names and their use for host address look up and electronic mail forwarding. It discusses the clients and servers in the domain name system and the protocol used between them.
- Domain names - implementation and specification. RFC 1035, RFC Editor, November 1987, DOI: 10.17487/RFC1035. 55 pages. Abstract: This RFC is the revised specification of the protocol and format used in the implementation of the Domain Name System. It obsoletes RFC-883. This memo documents the details of the domain name client-server communication.
- Stroustrup, Bjarne. The C++ Programming Language. Pearson Education, 2013.
- Wikipedia. 红黑树 — Wikipedia, The Free Encyclopedia. [Online; accessed 01-July-2024].

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

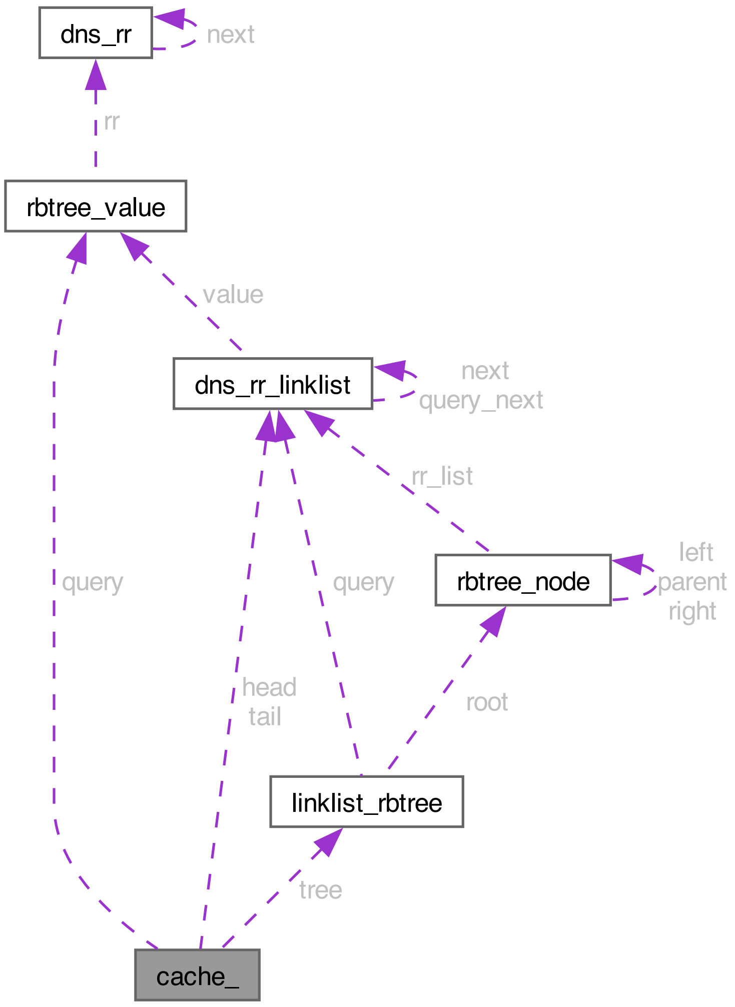
Data Structure Documentation

4.1 cache_ Struct Reference

Cash struct.

#include <cache.h>

Collaboration diagram for cache_:



Data Fields

- Dns_RR_LinkList ∗ head

  LRU header node.
- Dns_RR_LinkList ∗ tail

  LRU tail node.
- int size

  LRU size.
- Rbtree ∗ tree

  Red−black tree.
- void(∗ insert)(struct cache_ ∗cache, const Dns_Msg ∗msg)

Insert a DNS message into the cache.

- Rbtree_Value ∗(∗ query)(struct cache_ ∗cache, const Dns_Que ∗que)

  Query the cache for a DNS question.

4.1.1 Detailed Description

Cash struct.

4.1.2 Field Documentation

4.1.2.1 head

Dns_RR_LinkList∗ cache_::head

LRU header node.

4.1.2.2 insert

void(∗ cache_::insert) (struct cache_ ∗cache, const Dns_Msg ∗msg)

Insert a DNS message into the cache.

Parameters

cache	The cache where the message will be inserted.
msg	The DNS message to be inserted.

4.1.2.3 query

Rbtree_Value ∗(∗ cache_::query) (struct cache_ ∗cache, const Dns_Que ∗que)

Query the cache for a DNS question.

Parameters

cache	The cache to query.
que	The DNS question.

Returns

The value found in the cache or NULL if not found.

4.1.2.4 size

int cache_::size

LRU size.

4.1.2.5 tail

Dns_RR_LinkList∗ cache_::tail

LRU tail node.

4.1.2.6 tree

Rbtree∗ cache_::tree

Red−black tree.

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

- include/cache.h

4.2 dns_header Struct Reference

Header Section structure of DNS message.

#include <dns.h>

Data Fields

- uint16_t id
- uint8_t qr: 1
- uint8_t opcode: 4
- uint8_t aa: 1
- uint8_t tc: 1
- uint8_t rd: 1
- uint8_t ra: 1
- uint8_t z: 3
- uint8_t rcode: 4
- uint16_t qdcount
- uint16_t ancount
- uint16_t nscount
- uint16_t arcount

4.2.1 Detailed Description

Header Section structure of DNS message.

4.2.2 Field Documentation

4.2.2.1 aa

uint8_t dns_header::aa

4.2.2.2 ancount

uint16_t dns_header::ancount

4.2.2.3 arcount

uint16_t dns_header::arcount

4.2.2.4 id

uint16_t dns_header::id

4.2.2.5 nscount

uint16_t dns_header::nscount

4.2.2.6 opcode

uint8_t dns_header::opcode

4.2.2.7 qdcount

uint16_t dns_header::qdcount

4.2.2.8 qr

uint8_t dns_header::qr

4.2.2.9 ra

uint8_t dns_header::ra

4.2.2.10 rcode

uint8_t dns_header::rcode

4.2.2.11 rd

uint8_t dns_header::rd

4.2.2.12 tc

uint8_t dns_header::tc

4.2.2.13 z

uint8_t dns_header::z

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

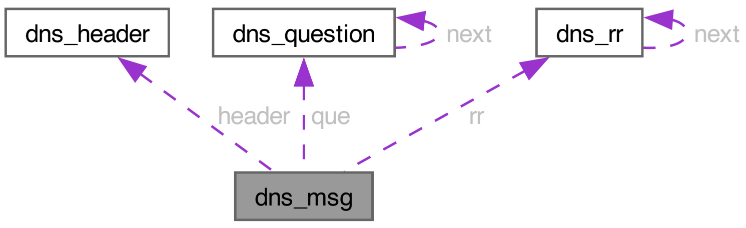
- include/dns.h

4.3 dns_msg Struct Reference

DNS message structure.

#include <dns.h>

Collaboration diagram for dns_msg:



Data Fields

- Dns_Header ∗ header

  Pointer to the Header Section.
- Dns_Que ∗ que

  Pointer to the head node of the Question Section linked list.
- Dns_RR ∗ rr

  Pointer to the head node of the Resource Record linked list.

4.3.1 Detailed Description

DNS message structure.

4.3.2 Field Documentation

4.3.2.1 header

Dns_Header∗ dns_msg::header

Pointer to the Header Section.

4.3.2.2 que

Dns_Que∗ dns_msg::que

Pointer to the head node of the Question Section linked list.

4.3.2.3 rr

Dns_RR∗ dns_msg::rr

Pointer to the head node of the Resource Record linked list.

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

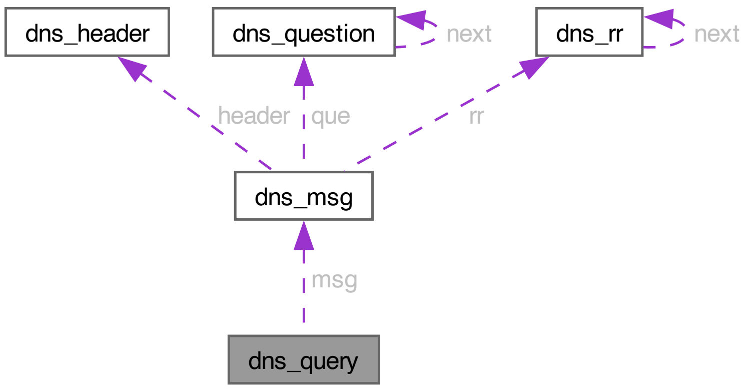
- include/dns.h

4.4 dns_query Struct Reference

DNS query structure.

#include <query_pool.h>

Collaboration diagram for dns_query:



Data Fields

- uint16_t id

  Query ID.
- uint16_t prev_id

  Original DNS query message ID.
- struct sockaddr addr

  Address of the requester.
- Dns_Msg ∗ msg

  DNS query message.
- uv_timer_t timer

  Timer.

4.4.1 Detailed Description

DNS query structure.

4.4.2 Field Documentation

4.4.2.1 addr

struct sockaddr dns_query::addr

Address of the requester.

4.4.2.2 id

uint16_t dns_query::id

Query ID.

4.4.2.3 msg

Dns_Msg* dns_query::msg

DNS query message.

4.4.2.4 prev_id

uint16_t dns_query::prev_id

Original DNS query message ID.

4.4.2.5 timer

uv_timer_t dns_query::timer

Timer.

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

- include/query_pool.h

4.5 dns_question Struct Reference

Question Section structure of DNS message, represented as a linked list.

#include <dns.h>

Collaboration diagram for dns_question:

dns_question
next

Data Fields

- uint8_t ∗ qname
- uint16_t qtype
- uint16_t qclass
- struct dns_question ∗ next

4.5.1 Detailed Description

Question Section structure of DNS message, represented as a linked list.

4.5.2 Field Documentation

4.5.2.1 next

struct dns_question∗ dns_question::next

4.5.2.2 qclass

uint16_t dns_question::qclass

4.5.2.3 qname

uint8_t∗ dns_question::qname

4.5.2.4 qtype

uint16_t dns_question::qtype

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

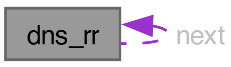
- include/dns.h

4.6 dns_rr Struct Reference

Resource Record structure of DNS message, represented as a linked list.

#include <dns.h>

Collaboration diagram for dns_rr:



Data Fields

- uint8_t ∗ name
- uint16_t type
- uint16_t class
- uint32_t ttl
- uint16_t rdlength
- uint8_t ∗ rdata
- struct dns_rr ∗ next

4.6.1 Detailed Description

Resource Record structure of DNS message, represented as a linked list.

4.6.2 Field Documentation

4.6.2.1 class

uint16_t dns_rr::class

4.6.2.2 name

uint8_t∗ dns_rr::name

4.6.2.3 next

struct dns_rr∗ dns_rr::next

4.6.2.4 rdata

uint8_t∗ dns_rr::rdata

4.6.2.5 rdlength

uint16_t dns_rr::rdlength

4.6.2.6 ttl

uint32_t dns_rr::ttl

4.6.2.7 type

uint16_t dns_rr::type

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

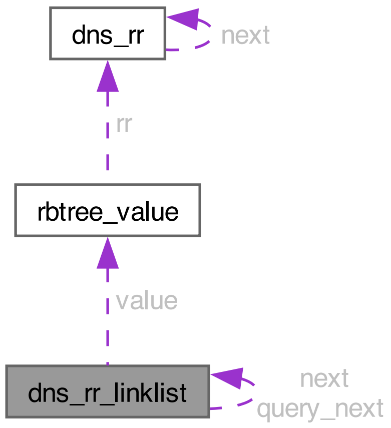
- include/dns.h

4.7 dns_rr_linklist Struct Reference

Linked list of Red-Black Tree nodes.

#include <linklist_rbtree.h>

Collaboration diagram for dns_rr_linklist:



Data Fields

- Rbtree_Value ∗ value

  Pointer to the value of the current linked list node.
- time_t expire_time

  Expiration time.
- struct dns_rr_linklist ∗ next

  Pointer to the next node in the linked list.
- void(∗ insert)(struct dns_rr_linklist ∗list, struct dns_rr_linklist ∗new_list_node)

  Insert a key-value pair into the red-black tree.
- void(∗ delete_next)(struct dns_rr_linklist ∗list)

  Delete the next element in the linked list.
- struct dns_rr_linklist ∗(∗ query_next)(struct dns_rr_linklist ∗list, const uint8_t ∗qname, const uint16_t qtype)

  Query the next element in the linked list.

4.7.1 Detailed Description

Linked list of Red-Black Tree nodes.

4.7.2 Field Documentation

4.7.2.1 delete_next

void(∗ dns_rr_linklist::delete_next) (struct dns_rr_linklist ∗list)

Delete the next element in the linked list.

Parameters

list	The linked list

4.7.2.2 expire_time

time_t dns_rr_linklist::expire_time

Expiration time.

4.7.2.3 insert

void(∗ dns_rr_linklist::insert) (struct dns_rr_linklist ∗list, struct dns_rr_linklist ∗new_list_node)

Insert a key-value pair into the red-black tree.

Parameters

tree	The red-black tree
key	The key
list	The value

4.7.2.4 next

struct dns_rr_linklist∗ dns_rr_linklist::next

Pointer to the next node in the linked list.

4.7.2.5 query_next

struct dns_rr_linklist ∗(∗ dns_rr_linklist::query_next) (struct dns_rr_linklist ∗list, const uint8_t ∗qname, const uint16_t qtype)

Query the next element in the linked list.

Parameters

list	The linked list
qname	The query name
qtype	The query type

Returns

The queried element if found, otherwise NULL

4.7.2.6 value

Rbtree_Value∗ dns_rr_linklist::value

Pointer to the value of the current linked list node.

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

- include/linklist_rbtree.h

4.8 index_ Struct Reference

Index structure.

#include <index_pool.h>

Data Fields

- uint16_t id

  The ID of the sent DNS query message.
- uint16_t prev_id

  The corresponding query ID.

4.8.1 Detailed Description

Index structure.

4.8.2 Field Documentation

4.8.2.1 id

uint16_t index_::id

The ID of the sent DNS query message.

4.8.2.2 prev_id

uint16_t index_::prev_id

The corresponding query ID.

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

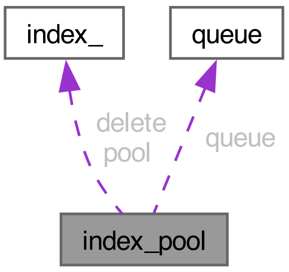
- include/index_pool.h

4.9 index_pool Struct Reference

Index pool.

#include <index_pool.h>

Collaboration diagram for index_pool:



Data Fields

- Index ∗ pool [INDEX_POOL_MAX_SIZE]

    Index pool.
- unsigned short count

    Number of indices in the pool.
- Queue ∗ queue

    Queue of unallocated indices.
- bool(∗ full)(struct index_pool ∗ipool)

    Check if the index pool is full.
- uint16_t(∗ insert)(struct index_pool ∗ipool, Index ∗req)

    Insert an index into the pool.
- bool(∗ query)(struct index_pool ∗ipool, uint16_t index)

    Query if an index exists in the pool.
- Index ∗(∗ delete)(struct index_pool ∗ipool, uint16_t index)

    Delete an index from the pool.
- void(∗ destroy)(struct index_pool ∗ipool)

    Destroy the index pool.

4.9.1 Detailed Description

Index pool.

4.9.2 Field Documentation

4.9.2.1 count

unsigned short index_pool::count

Number of indices in the pool.

4.9.2.2 delete

Index ∗(∗ index_pool::delete) (struct index_pool ∗ipool, uint16_t index)

Delete an index from the pool.

Parameters

ipool	The index pool
index	The index to delete

Returns

The deleted index

4.9.2.3 destroy

void(∗ index_pool::destroy) (struct index_pool ∗ipool)

Destroy the index pool.

Parameters

ipool	The index pool to destroy

4.9.2.4 full

bool(∗ index_pool::full) (struct index_pool ∗ipool)

Check if the index pool is full.

Parameters

ipool	The index pool

Returns

True if the index pool is full, false otherwise

4.9.2.5 insert

uint16_t(∗ index_pool::insert) (struct index_pool ∗ipool, Index ∗req)

Insert an index into the pool.

Parameters

ipool	The index pool
req	The index to insert

Returns

The ID of the inserted index

4.9.2.6 pool

Index∗ index_pool::pool[INDEX_POOL_MAX_SIZE]

Index pool.

4.9.2.7 query

bool(∗ index_pool::query) (struct index_pool ∗ipool, uint16_t index)

Query if an index exists in the pool.

Parameters

ipool	The index pool
index	The index to query

Returns

True if the index exists, false otherwise

4.9.2.8 queue

Queue* index_pool::queue

Queue of unallocated indices.

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

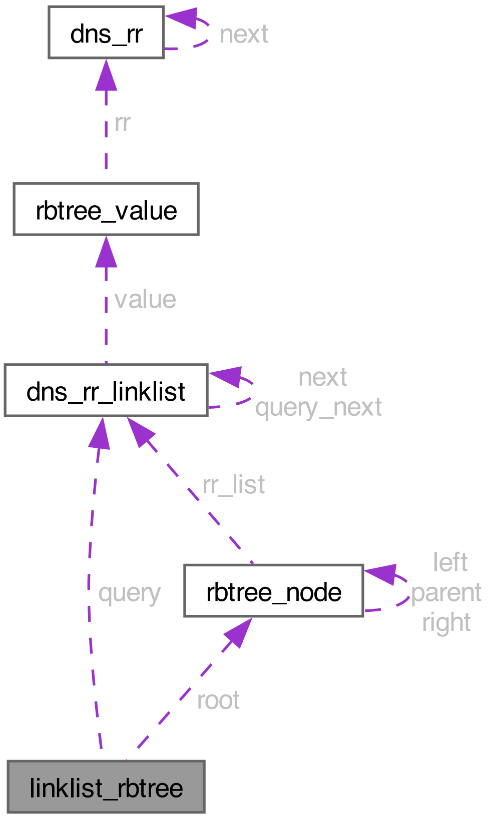
- include/index_pool.h

4.10 linklist_rbtree Struct Reference

Red-Black Tree.

#include <linklist_rbtree.h>

Collaboration diagram for linklist_rbtree:



Data Fields

- Rbtree_Node * root

  Pointer to the root node of the Red-Black Tree.
- void(* insert)(struct linklist_rbtree *tree, unsigned int key, Dns_RR_LinkList *list)

  Insert a key-value pair into the red-black tree.
- Dns_RR_LinkList *(* query)(struct linklist_rbtree *tree, unsigned int data)

  Query the red-black tree for a key.

4.10.1 Detailed Description

Red-Black Tree.

4.10.2 Field Documentation

4.10.2.1 insert

void(∗ linklist_rbtree::insert) (struct linklist_rbtree ∗tree, unsigned int key, Dns_RR_LinkList ∗list)

Insert a key-value pair into the red-black tree.

Parameters

tree	The red-black tree
key	The key
list	The value

4.10.2.2 query

Dns_RR_LinkList ∗(∗ linklist_rbtree::query) (struct linklist_rbtree ∗tree, unsigned int data)

Query the red-black tree for a key.

Parameters

tree	The red-black tree
key	The key to query

Returns

The linked list of the value if found, otherwise NULL

4.10.2.3 root

Rbtree_Node∗ linklist_rbtree::root

Pointer to the root node of the Red-Black Tree.

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

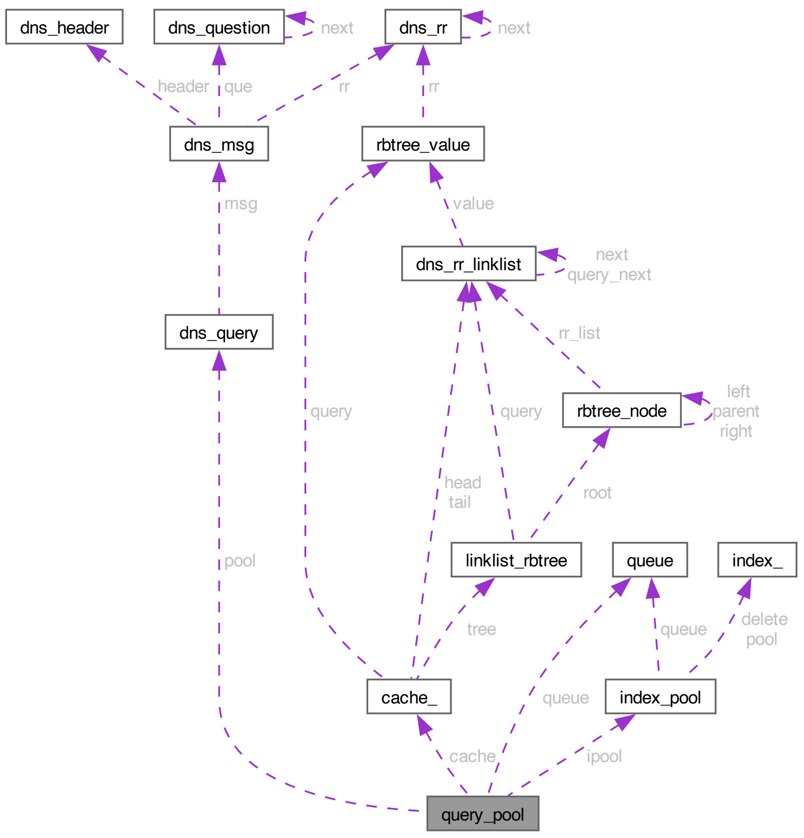
- include/linklist_rbtree.h

4.11 query_pool Struct Reference

DNS query pool.

#include <query_pool.h>

Collaboration diagram for query_pool:



Data Fields

- Dns_Query * pool [QUERY_POOL_MAX_SIZE]

  Query pool.
- unsigned short count

  Number of queries in the pool.
- Queue * queue

  Queue of unassigned query IDs.
- Index_Pool * ipool

  Index pool.
- uv_loop_t * loop

Event loop.

- Cache ∗ cache

  Cache.
- bool(∗ full)(struct query_pool ∗qpool)

  Check if the query pool is full.
- void(∗ insert)(struct query_pool ∗qpool, const struct sockaddr ∗addr, const Dns_Msg ∗msg)

  Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.
- void(∗ finish)(struct query_pool ∗qpool, const Dns_Msg ∗msg)

  Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.
- void(∗ delete)(struct query_pool ∗qpool, uint16_t id)

  Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.

4.11.1 Detailed Description

DNS query pool.

4.11.2 Field Documentation

4.11.2.1 cache

Cache∗ query_pool::cache

Cache.

4.11.2.2 count

unsigned short query_pool::count

Number of queries in the pool.

4.11.2.3 delete

void(∗ query_pool::delete) (struct query_pool ∗qpool, uint16_t id)

Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.

Parameters

qpool	The query pool
id	The ID of the query to be deleted

4.11.2.4 finish

void(∗ query_pool::finish) (struct query_pool ∗qpool, const Dns_Msg ∗msg)

Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.

Parameters

qpool	The query pool
msg	The DNS message containing the response

4.11.2.5 full

bool(∗ query_pool::full) (struct query_pool ∗qpool)

Check if the query pool is full.

Parameters

this	The query pool

Returns

true if the query pool is full, false otherwise

4.11.2.6 insert

void(∗ query_pool::insert) (struct query_pool ∗qpool, const struct sockaddr ∗addr, const Dns_Msg ∗msg)

Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.

Parameters

qpool	The query pool
addr	The address of the client
msg	The DNS message containing the query

4.11.2.7 ipool

Index_Pool* query_pool::ipool

Index pool.

4.11.2.8 loop

uv_loop_t* query_pool::loop

Event loop.

4.11.2.9 pool

Dns_Query* query_pool::pool[QUERY_POOL_MAX_SIZE]

Query pool.

4.11.2.10 queue

Queue* query_pool::queue

Queue of unassigned query IDs.

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

- include/query_pool.h

4.12 queue Struct Reference

Circular queue.

#include <queue.h>

Data Fields

- uint16_t q [QUEUE_MAX_SIZE]

    The queue.
- unsigned short head

    The head of the queue.
- unsigned short tail

    The tail of the queue.
- void(∗ push)(struct queue ∗queue, uint16_t num)

    Push a number onto the queue.
- uint16_t(∗ pop)(struct queue ∗queue)

    Pop a number from the queue.
- void(∗ destroy)(struct queue ∗queue)

    Destroy the queue.

4.12.1 Detailed Description

Circular queue.

4.12.2 Field Documentation

4.12.2.1 destroy

void(∗ queue::destroy) (struct queue ∗queue)

Destroy the queue.

Parameters

queue	The queue to destroy

4.12.2.2 head

unsigned short queue::head

The head of the queue.

4.12.2.3 pop

uint16_t(∗ queue::pop) (struct queue ∗queue)

Pop a number from the queue.

Parameters

queue	The queue

Returns

The number popped from the queue

4.12.2.4 push

void(∗ queue::push) (struct queue ∗queue, uint16_t num)

Push a number onto the queue.

Parameters

queue	The queue
num	The number to push

4.12.2.5 q

uint16_t queue::q[QUEUE_MAX_SIZE]

The queue.

4.12.2.6 tail

unsigned short queue::tail

The tail of the queue.

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

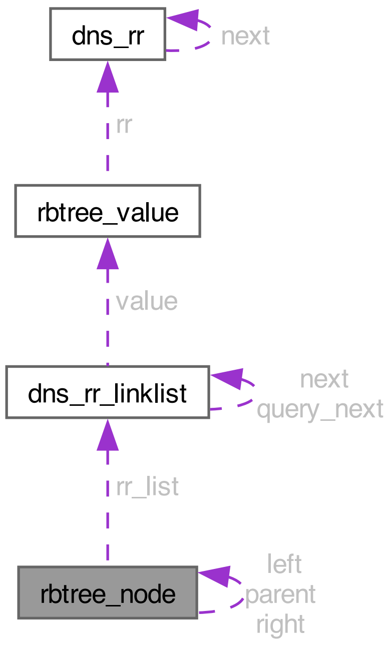
- include/queue.h

4.13 rbtree_node Struct Reference

Node of the Red-Black Tree.

#include <linklist_rbtree.h>

Collaboration diagram for rbtree_node:



Data Fields

- unsigned int key

  Key of the Red-Black Tree node.
- Dns_RR_LinkList ∗ rr_list

  Pointer to the linked list corresponding to the current node.
- Color color

  Color of the current node.
- struct rbtree_node ∗ left

  Pointer to the left child of the current node.
- struct rbtree_node ∗ right

  Pointer to the right child of the current node.
- struct rbtree_node ∗ parent

  Pointer to the parent of the current node.

4.13.1 Detailed Description

Node of the Red-Black Tree.

4.13.2 Field Documentation

4.13.2.1 color

Color rbtree_node::color

Color of the current node.

4.13.2.2 key

unsigned int rbtree_node::key

Key of the Red-Black Tree node.

4.13.2.3 left

struct rbtree_node∗ rbtree_node::left

Pointer to the left child of the current node.

4.13.2.4 parent

struct rbtree_node∗ rbtree_node::parent

Pointer to the parent of the current node.

4.13.2.5 right

struct rbtree_node∗ rbtree_node::right

Pointer to the right child of the current node.

4.13.2.6 rr_list

Dns_RR_LinkList∗ rbtree_node::rr_list

Pointer to the linked list corresponding to the current node.

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

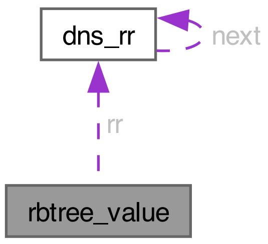
- include/linklist_rbtree.h

4.14 rbtree_value Struct Reference

Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.

#include <linklist_rbtree.h>

Collaboration diagram for rbtree_value:



Data Fields

- Dns_RR ∗ rr

  Pointer to a linked list of Dns_RR.
- uint16_t ancount

  Number of RRs in the Answer Section.
- uint16_t nscount

  Number of RRs in the Authority Section.
- uint16_t arcount

  Number of RRs in the Additional Section.
- uint8_t type

  Type of the Question corresponding to the RR.

4.14.1 Detailed Description

Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.

4.14.2 Field Documentation

4.14.2.1 ancount

uint16_t rbtree_value::ancount

Number of RRs in the Answer Section.

4.14.2.2 arcount

uint16_t rbtree_value::arcount

Number of RRs in the Additional Section.

4.14.2.3 nscount

uint16_t rbtree_value::nscount

Number of RRs in the Authority Section.

4.14.2.4 rr

Dns_RR* rbtree_value::rr

Pointer to a linked list of Dns_RR.

4.14.2.5 type

uint8_t rbtree_value::type

Type of the Question corresponding to the RR.

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

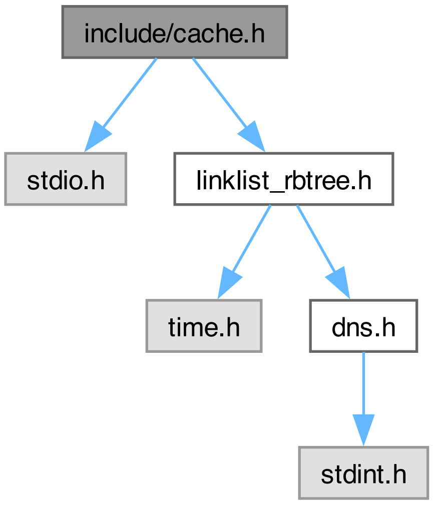
- include/linklist_rbtree.h

Chapter 5

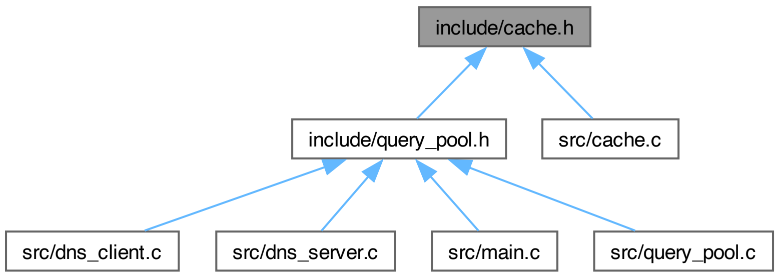
File Documentation

5.1 include/cache.h File Reference

#include <stdio.h>
#include "linklist_rbtree.h"
Include dependency graph for cache.h:



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



Data Structures

- struct cache_

  Cash struct.

Macros

- #define CACHE_SIZE 30

Typedefs

- typedef struct cache_ Cache

  Cash struct.

Functions

- Cache ∗ new_cache (FILE ∗hosts_file)

  Create a new cache and initialize it with data from the hosts file.

5.1.1 Macro Definition Documentation

5.1.1.1 CACHE_SIZE

```
#define CACHE_SIZE 30
```

5.1.2 Typedef Documentation

5.1.2.1 Cache

typedef struct cache_ Cache

Cash struct.

5.1.3 Function Documentation

5.1.3.1 new_cache()

Cache ∗ new_cache (

FILE ∗ hosts_file)

Create a new cache and initialize it with data from the hosts file.

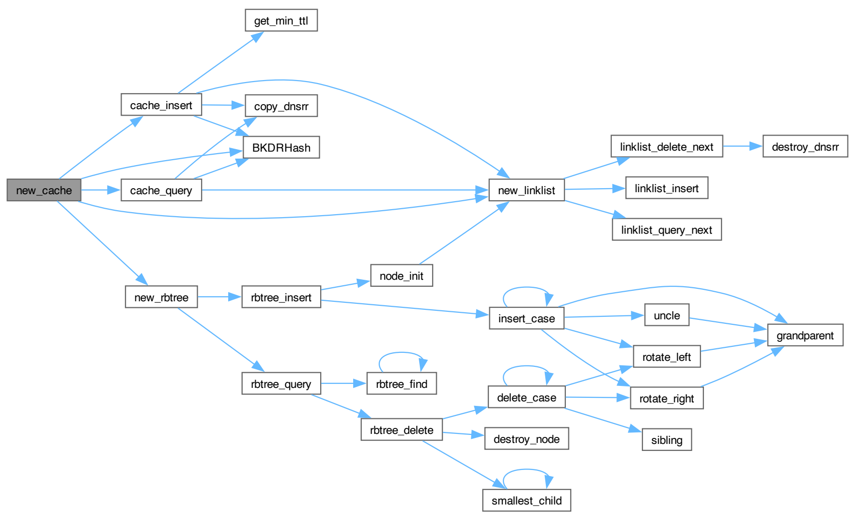
Parameters

hosts_file	The file containing hosts data.

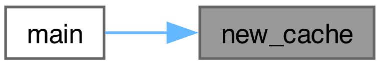
Returns

The newly created cache.

Here is the call graph for this function:



Here is the caller graph for this function:



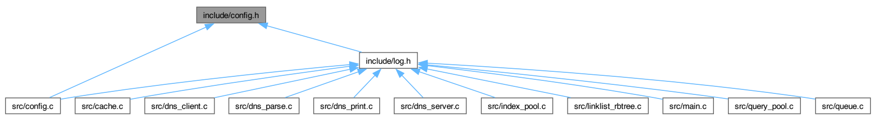
5.2 cache.h

Go to the documentation of this file.

```
#ifndef DNSR_CACHE_H
#define DNSR_CACHE_H

#include <stdio.h>

#include "linklist_rbtree.h"

#define CACHE_SIZE 30

typedef struct cache_ {
    Dns_RR_LinkList * head;
    Dns_RR_LinkList * tail;
    int size;
    Rbtree * tree;

    void (* insert)(struct cache_ * cache, const Dns_Msg * msg);

    Rbtree_Value * (* query)(struct cache_ * cache, const Dns_Que * que);
} Cache;

Cache * new_cache(FILE * hosts_file);

#endif //DNSR_CACHE_H
```

5.3 include/config.h File Reference

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



Functions

- void init_config (int argc, char ∗const ∗argv)

  Parse command line arguments.

Variables

- char ∗ REMOTE_HOST

    Remote DNS server address.
- int LOG_MASK

    Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.
- int CLIENT_PORT

    Local DNS client port.
- char ∗ HOSTS_PATH

    Hosts file path.
- char ∗ LOG_PATH

    Log file path.

5.3.1 Function Documentation

5.3.1.1 init_config()

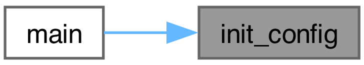
```
void init_config (
                    int argc,
                    char ∗const ∗ argv)
```

Parse command line arguments.

Parameters

argc	Number of arguments
argv	Array of argument strings

Here is the caller graph for this function:



5.3.2 Variable Documentation

5.3.2.1 CLIENT_PORT

```
int CLIENT_PORT [extern]
```

Local DNS client port.

5.3.2.2 HOSTS_PATH

char∗ HOSTS_PATH [extern]

Hosts file path.

5.3.2.3 LOG_MASK

int LOG_MASK [extern]

Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.

5.3.2.4 LOG_PATH

char∗ LOG_PATH [extern]

Log file path.

5.3.2.5 REMOTE_HOST

char∗ REMOTE_HOST [extern]

Remote DNS server address.

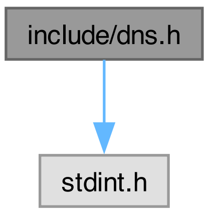
5.4 config.h

Go to the documentation of this file.

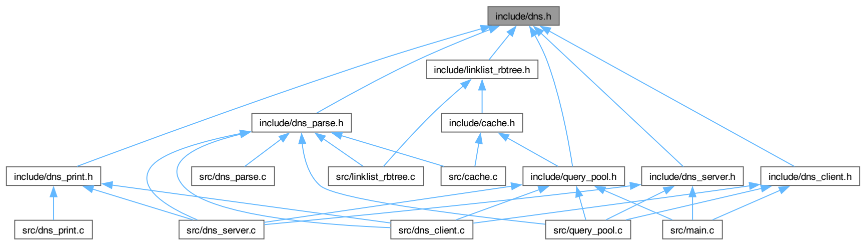
```
#ifndef DNSR_CONFIG_H
#define DNSR_CONFIG_H

extern char * REMOTE_HOST;
extern int LOG_MASK;
extern int CLIENT_PORT;
extern char * HOSTS_PATH;
extern char * LOG_PATH;

void init_config(int argc, char * const * argv);

#endif //DNSR_CONFIG_H
```

5.5 include/dns.h File Reference

#include <stdint.h>
Include dependency graph for dns.h:



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



Data Structures

- struct dns_header

  Header Section structure of DNS message.
- struct dns_question

  Question Section structure of DNS message, represented as a linked list.
- struct dns_rr

  Resource Record structure of DNS message, represented as a linked list.
- struct dns_msg

  DNS message structure.

Macros

- #define DNS_STRING_MAX_SIZE 8192
- #define DNS_RR_NAME_MAX_SIZE 512
- #define DNS_QR_QUERY 0
- #define DNS_QR_ANSWER 1
- #define DNS_OPCODE_QUERY 0
- #define DNS_OPCODE_IQUERY 1
- #define DNS_OPCODE_STATUS 2
- #define DNS_TYPE_A 1
- #define DNS_TYPE_NS 2
- #define DNS_TYPE_CNAME 5
- #define DNS_TYPE_SOA 6
- #define DNS_TYPE_PTR 12
- #define DNS_TYPE_HINFO 13
- #define DNS_TYPE_MINFO 15
- #define DNS_TYPE_MX 15
- #define DNS_TYPE_TXT 16
- #define DNS_TYPE_AAAA 28
- #define DNS_CLASS_IN 1
- #define DNS_RCODE_OK 0
- #define DNS_RCODE_NXDOMAIN 3
- #define DNS_RCODE_SERVFAIL 2

Typedefs

- typedef struct dns_header Dns_Header

  Header Section structure of DNS message.
- typedef struct dns_question Dns_Que

  Question Section structure of DNS message, represented as a linked list.
- typedef struct dns_rr Dns_RR

  Resource Record structure of DNS message, represented as a linked list.
- typedef struct dns_msg Dns_Msg

  DNS message structure.

5.5.1 Macro Definition Documentation

5.5.1.1 DNS_CLASS_IN

#define DNS_CLASS_IN 1

5.5.1.2 DNS_OPCODE_IQUERY

#define DNS_OPCODE_IQUERY 1

5.5.1.3 DNS_OPCODE_QUERY

#define DNS_OPCODE_QUERY 0

5.5.1.4 DNS_OPCODE_STATUS

#define DNS_OPCODE_STATUS 2

5.5.1.5 DNS_QR_ANSWER

#define DNS_QR_ANSWER 1

5.5.1.6 DNS_QR_QUERY

#define DNS_QR_QUERY 0

5.5.1.7 DNS_RCODE_NXDOMAIN

#define DNS_RCODE_NXDOMAIN 3

5.5.1.8 DNS_RCODE_OK

#define DNS_RCODE_OK 0

5.5.1.9 DNS_RCODE_SERVFAIL

#define DNS_RCODE_SERVFAIL 2

5.5.1.10 DNS_RR_NAME_MAX_SIZE

#define DNS_RR_NAME_MAX_SIZE 512

5.5.1.11 DNS_STRING_MAX_SIZE

#define DNS_STRING_MAX_SIZE 8192

5.5.1.12 DNS_TYPE_A

#define DNS_TYPE_A 1

5.5.1.13 DNS_TYPE_AAAA

#define DNS_TYPE_AAAA 28

5.5.1.14 DNS_TYPE_CNAME

#define DNS_TYPE_CNAME 5

5.5.1.15 DNS_TYPE_HINFO

#define DNS_TYPE_HINFO 13

5.5.1.16 DNS_TYPE_MINFO

#define DNS_TYPE_MINFO 15

5.5.1.17 DNS_TYPE_MX

#define DNS_TYPE_MX 15

5.5.1.18 DNS_TYPE_NS

#define DNS_TYPE_NS 2

5.5.1.19 DNS_TYPE_PTR

#define DNS_TYPE_PTR 12

5.5.1.20 DNS_TYPE_SOA

#define DNS_TYPE_SOA 6

5.5.1.21 DNS_TYPE_TXT

#define DNS_TYPE_TXT 16

5.5.2 Typedef Documentation

5.5.2.1 Dns_Header

typedef struct dns_header Dns_Header

Header Section structure of DNS message.

5.5.2.2 Dns_Msg

typedef struct dns_msg Dns_Msg

DNS message structure.

5.5.2.3 Dns_Que

typedef struct dns_question Dns_Que

Question Section structure of DNS message, represented as a linked list.

5.5.2.4 Dns_RR

typedef struct dns_rr Dns_RR

Resource Record structure of DNS message, represented as a linked list.

5.6 dns.h

Go to the documentation of this file.

```
#ifndef DNSR_DNS_H
#define DNSR_DNS_H

#include <stdint.h>

#define DNS_STRING_MAX_SIZE 8192
#define DNS_RR_NAME_MAX_SIZE 512

#define DNS_QR_QUERY 0
#define DNS_QR_ANSWER 1

#define DNS_OPCODE_QUERY 0
#define DNS_OPCODE_IQUERY 1
#define DNS_OPCODE_STATUS 2

#define DNS_TYPE_A 1
#define DNS_TYPE_NS 2
#define DNS_TYPE_CNAME 5
#define DNS_TYPE_SOA 6
#define DNS_TYPE_PTR 12
#define DNS_TYPE_HINFO 13
#define DNS_TYPE_MINFO 15
#define DNS_TYPE_MX 15
#define DNS_TYPE_TXT 16
#define DNS_TYPE_AAAA 28

#define DNS_CLASS_IN 1

#define DNS_RCODE_OK 0
#define DNS_RCODE_NXDOMAIN 3
#define DNS_RCODE_SERVFAIL 2

typedef struct dns_header {
    uint16_t id;
    uint8_t qr: 1;
    uint8_t opcode: 4;
    uint8_t aa: 1;
    uint8_t tc: 1;
    uint8_t rd: 1;
    uint8_t ra: 1;
    uint8_t z: 3;
    uint8_t rcode: 4;
    uint16_t qdcount;
    uint16_t ancount;
    uint16_t nscount;
    uint16_t arcount;
} Dns_Header;

typedef struct dns_question {
    uint8_t * qname;
    uint16_t qtype;
    uint16_t qclass;
    struct dns_question * next;
} Dns_Que;

typedef struct dns_rr {
    uint8_t * name;
    uint16_t type;
    uint16_t class;
    uint32_t ttl;
    uint16_t rdlength;
    uint8_t * rdata;
    struct dns_rr * next;
} Dns_RR;
```

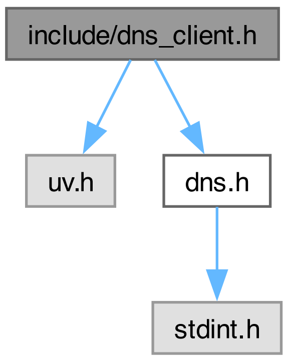
```
00068
00070 typedef struct dns_msg {
00071     Dns_Header * header;
00072     Dns_Que * que;
00073     Dns_RR * rr;
00074 } Dns_Msg;
00075
00076 #endif //DNSR_DNS_H
```

5.7 include/dns_client.h File Reference

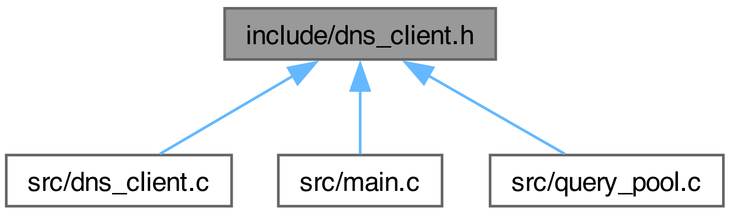
#include <uv.h>

#include ”dns.h”

Include dependency graph for dns_client.h:



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



Functions

- void init_client (uv_loop_t *loop)

  Initialize the DNS client.
- void send_to_remote (const Dns_Msg *msg)

  Send a DNS query message to the remote server.

5.7.1 Function Documentation

5.7.1.1 init_client()

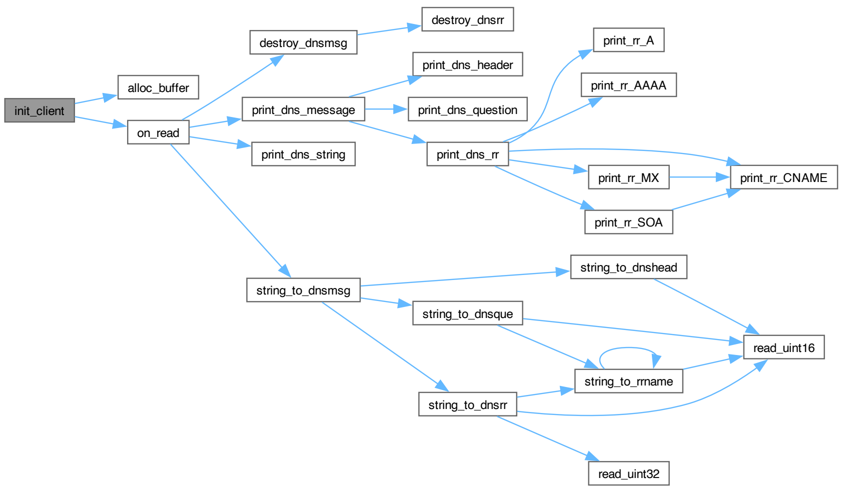
```
void init_client (
                    uv_loop_t * loop)
```

Initialize the DNS client.

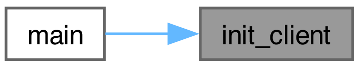
Parameters

loop	The libuv event loop

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.1.2 send_to_remote()

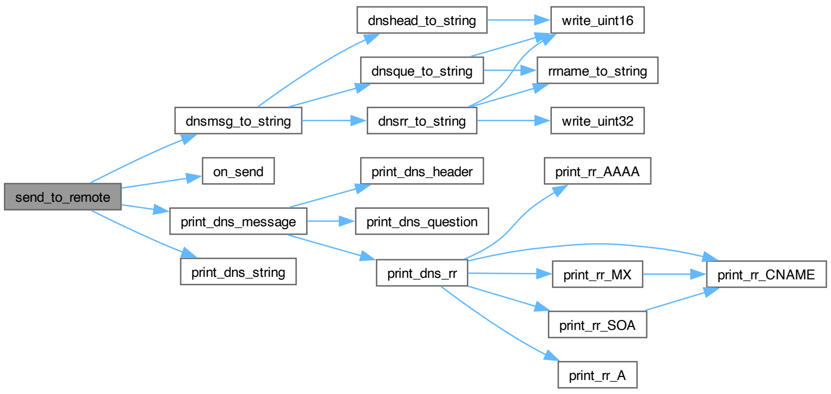
void send_to_remote (
const Dns_Msg ∗ msg)

Send a DNS query message to the remote server.

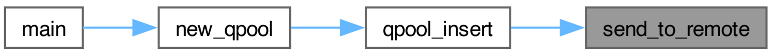
Parameters

msg	The DNS message to be sent

Here is the call graph for this function:



Here is the caller graph for this function:



5.8 dns_client.h

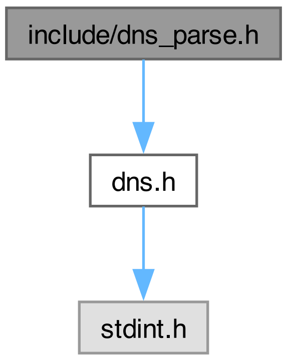
Go to the documentation of this file.

```
00001 #ifndef DNSR_DNS_CLIENT_H
00002 #define DNSR_DNS_CLIENT_H
00003
00004 #include <uv.h>
00005
00006 #include "dns.h"
00007
00012 void init_client(uv_loop_t * loop);
00013
00018 void send_to_remote(const Dns_Msg * msg);
00019
00020 #endif //DNSR_DNS_CLIENT_H
```

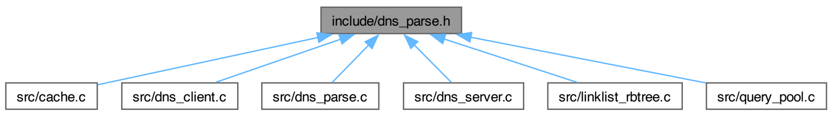
5.9 include/dns_parse.h File Reference

#include "dns.h"

Include dependency graph for dns_parse.h:



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



Functions

- void string_to_dnsmsg (Dns_Msg ∗pmsg, const char ∗pstring)

  Convert a byte stream to a DNS message structure.
- unsigned dnsmsg_to_string (const Dns_Msg ∗pmsg, char ∗pstring)

  Write a NAME field to a byte stream.
- void destroy_dnsrr (Dns_RR ∗prr)

  Release memory allocated for a Resource Record.
- void destroy_dnsmsg (Dns_Msg ∗pmsg)

  Release memory allocated for a DNS message.
- Dns_RR ∗ copy_dnsrr (const Dns_RR ∗src)

  Copy a Resource Record.
- Dns_Msg ∗ copy_dnsmsg (const Dns_Msg ∗src)

  Copy a DNS message.

5.9.1 Function Documentation

5.9.1.1 copy_dnsmsg()

Dns_Msg ∗ copy_dnsmsg (
const Dns_Msg ∗ src)

Copy a DNS message.

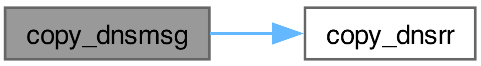
Parameters

src	The DNS message to copy

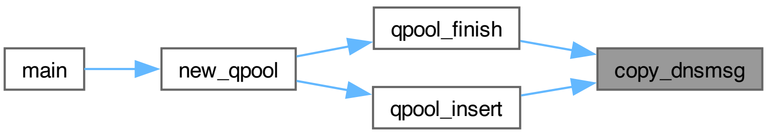
Returns

A copy of the DNS message

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.1.2 copy_dnsrr()

Dns_RR ∗ copy_dnsrr (
const Dns_RR ∗ src)

Copy a Resource Record.

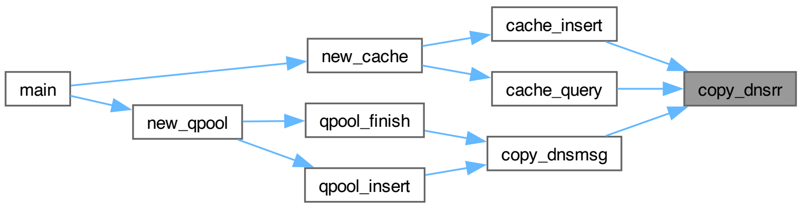
Parameters

src	The Resource Record to copy

Returns

A copy of the Resource Record

Here is the caller graph for this function:



5.9.1.3 destroy_dnsmsg()

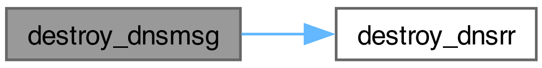
void destroy_dnsmsg (
Dns_Msg ∗ pmsg)

Release memory allocated for a DNS message.

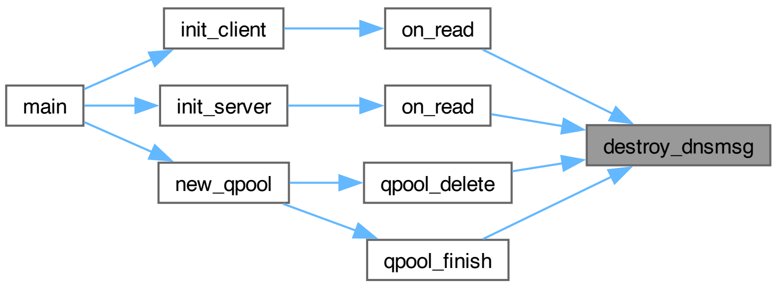
Parameters

pmsg	The DNS message to release

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.1.4 destroy_dnsrr()

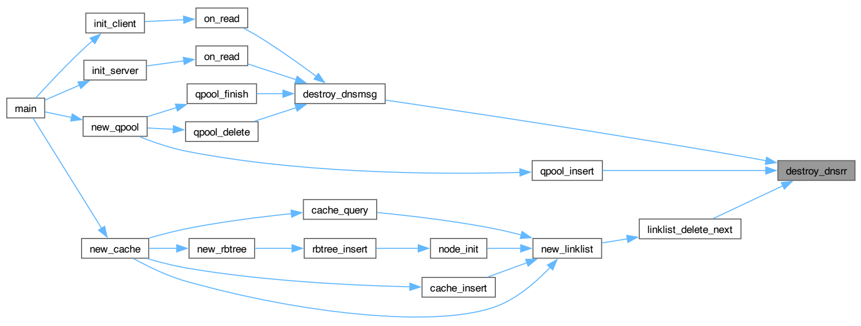
void destroy_dnsrr (
    Dns_RR ∗ prr)

Release memory allocated for a Resource Record.

Parameters

prr	The Resource Record to release

Here is the caller graph for this function:



5.9.1.5 dnsmsg_to_string()

unsigned dnsmsg_to_string (
    const Dns_Msg ∗ pmsg,
    char ∗ pstring)

Write a NAME field to a byte stream.

Parameters

pname	The NAME field
pstring	The start of the byte stream
offset	The offset in the byte stream

Note

After writing, the offset increases to the position after the NAME field

Write a NAME field to a byte stream.

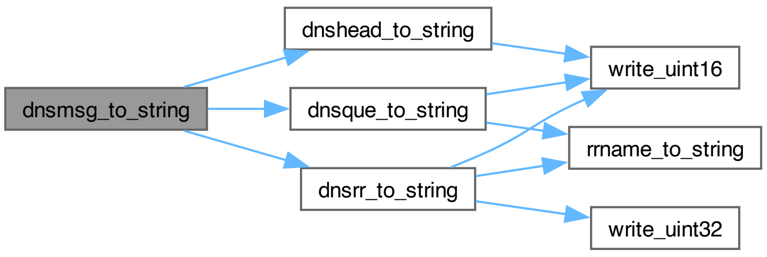
Parameters

pmsg	The DNS message structure to convert
pstring	The byte stream to write to

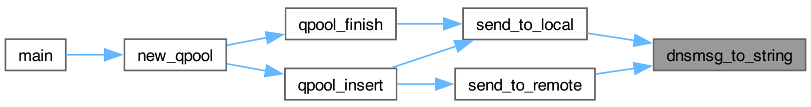
Returns

The total length of the byte stream

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.1.6 string_to_dnsmsg()

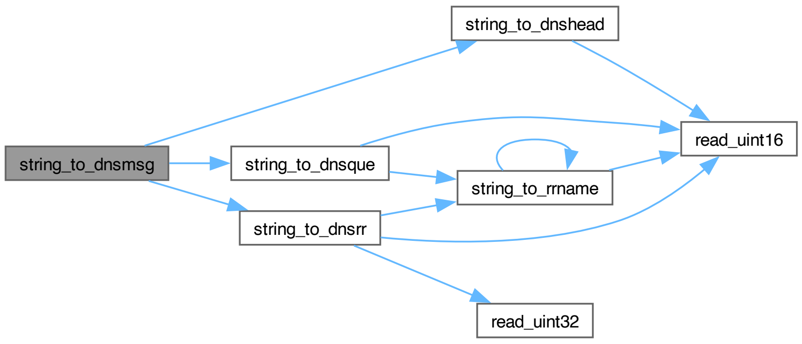
void string_to_dnsmsg (
    Dns_Msg ∗ pmsg,
    const char ∗ pstring)

Convert a byte stream to a DNS message structure.

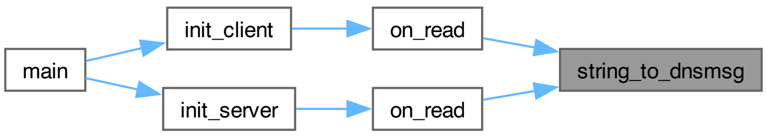
Parameters

pmsg	The DNS message structure to populate
pstring	The byte stream to read from

Here is the call graph for this function:



Here is the caller graph for this function:



5.10 dns_parse.h

Go to the documentation of this file.

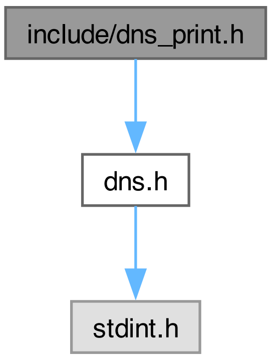
```
00001 #ifndef DNSR_DNS_PARSE_H
```

```
#define DNSR_DNS_PARSE_H

#include "dns.h"

void string_to_dnsmsg(Dns_Msg * pmsg, const char * pstring);

unsigned dnsmsg_to_string(const Dns_Msg * pmsg, char * pstring);

void destroy_dnsrr(Dns_RR * prr);

void destroy_dnsmsg(Dns_Msg * pmsg);

Dns_RR * copy_dnsrr(const Dns_RR * src);

Dns_Msg * copy_dnsmsg(const Dns_Msg * src);

#endif //DNSR_DNS_PARSE_H
```

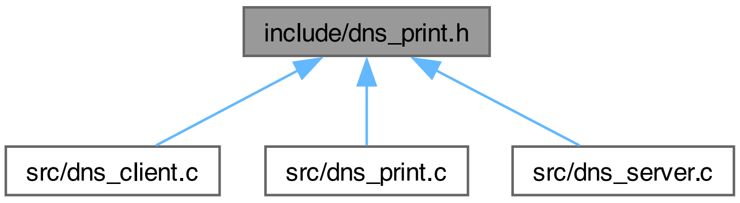
5.11 include/dns_print.h File Reference

#include "dns.h"

Include dependency graph for dns_print.h:



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



Functions

- void print_dns_string (const char ∗pstring, unsigned int len)

  Print DNS message byte stream.
- void print_dns_message (const Dns_Msg ∗pmsg)

  Print the entire DNS message.

5.11.1 Function Documentation

5.11.1.1 print_dns_message()

void print_dns_message (

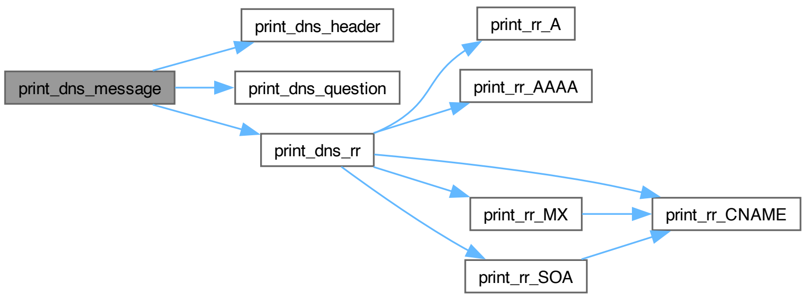
const Dns_Msg ∗ pmsg)

Print the entire DNS message.

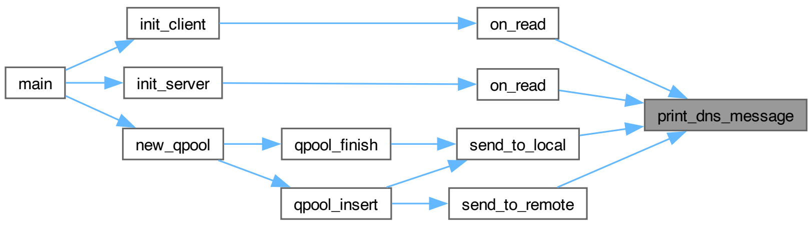
Parameters

pmsg	The DNS message

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.1.2 print_dns_string()

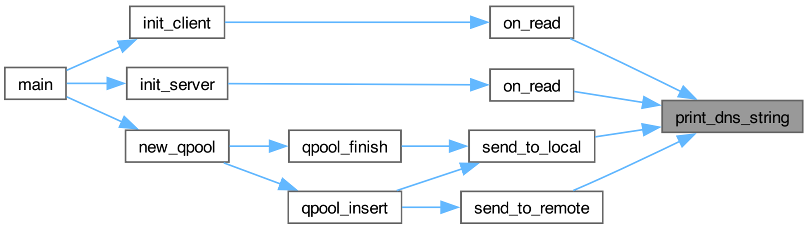
```
void print_dns_string (
                    const char ∗ pstring,
                    unsigned int len)
```

Print DNS message byte stream.

Parameters

pstring	The byte stream
len	The length of the byte stream

Here is the caller graph for this function:



5.12 dns_print.h

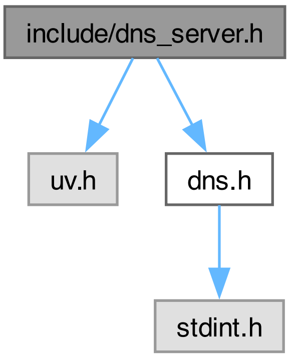
Go to the documentation of this file.

```
00001 #ifndef DNSR_DNS_PRINT_H
00002 #define DNSR_DNS_PRINT_H
00003
00004 #include "dns.h"
00005
00011 void print_dns_string(const char * pstring, unsigned int len);
00012
00017 void print_dns_message(const Dns_Msg * pmsg);
00018
00019 #endif //DNSR_DNS_PRINT_H
```

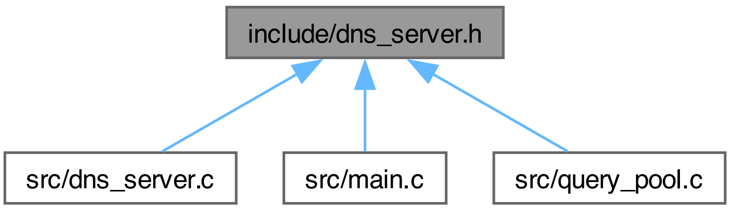
5.13 include/dns_server.h File Reference

#include <uv.h>
#include "dns.h"

Include dependency graph for dns_server.h:



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



Functions

- void init_server (uv_loop_t ∗loop)

  Initialize the DNS server.
- void send_to_local (const struct sockaddr ∗addr, const Dns_Msg ∗msg)

  Send a DNS response message to local clients.

5.13.1 Function Documentation

5.13.1.1 init_server()

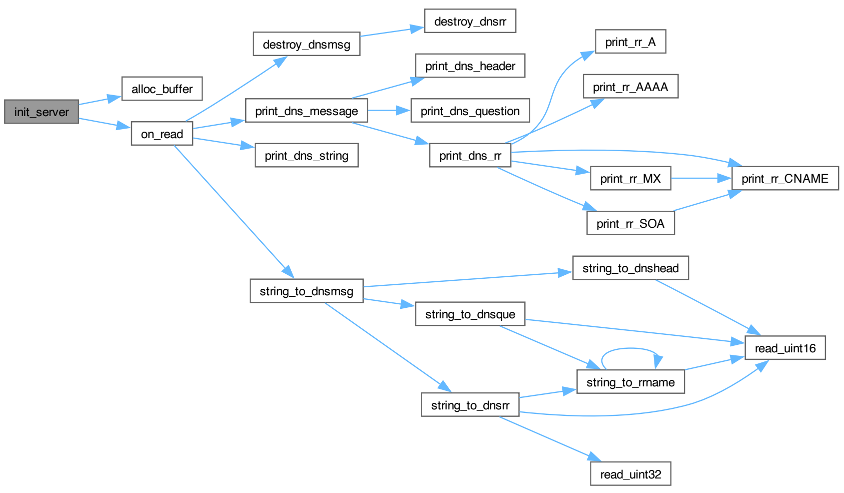
```
void init_server (
            uv_loop_t ∗ loop)
```

Initialize the DNS server.

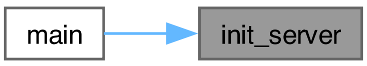
Parameters

loop	The libuv event loop

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.1.2 send_to_local()

```
void send_to_local (
                const struct sockaddr * addr,
                const Dns_Msg * msg)
```

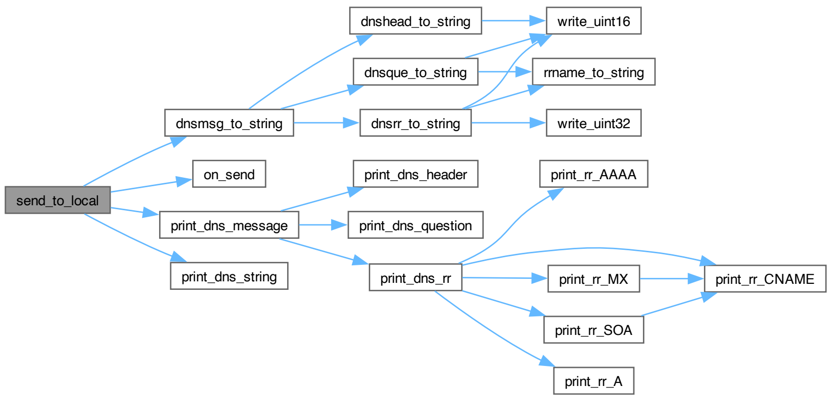
Send a DNS response message to local clients.

Parameters

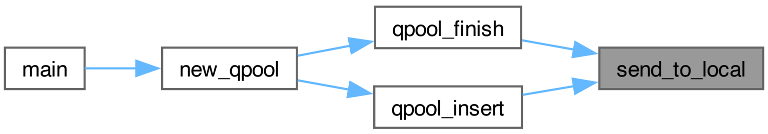
addr	The address of the local client

msg	The DNS message to be sent

Here is the call graph for this function:



Here is the caller graph for this function:



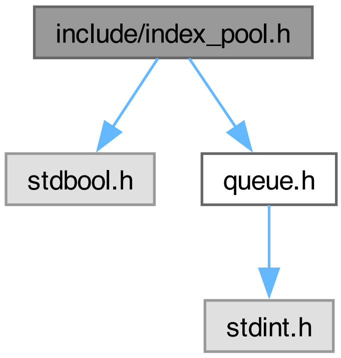
5.14 dns_server.h

Go to the documentation of this file.

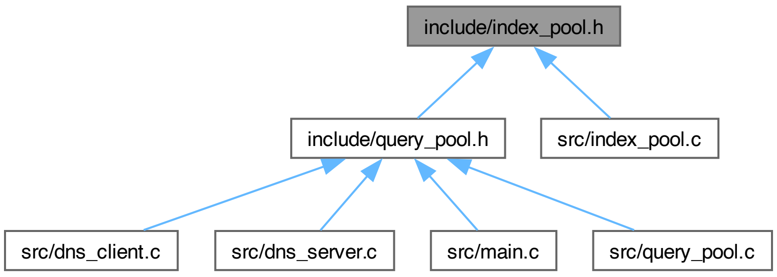
```
#ifndef DNSR_DNS_SERVER_H
#define DNSR_DNS_SERVER_H

#include <uv.h>

#include "dns.h"

void init_server(uv_loop_t * loop);

void send_to_local(const struct sockaddr * addr, const Dns_Msg * msg);

#endif //DNSR_DNS_SERVER_H
```

5.15 include/index_pool.h File Reference

#include <stdbool.h>
#include ”queue.h”
Include dependency graph for index_pool.h:



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



Data Structures

- struct index_

  Index structure.
- struct index_pool

  Index pool.

Macros

- #define INDEX_POOL_MAX_SIZE 65535

Typedefs

- typedef struct index_ Index

  Index structure.
- typedef struct index_pool Index_Pool

  Index pool.

Functions

- Index_Pool ∗ new_ipool ()

  Create a new index pool.

5.15.1 Macro Definition Documentation

5.15.1.1 INDEX_POOL_MAX_SIZE

#define INDEX_POOL_MAX_SIZE 65535

5.15.2 Typedef Documentation

5.15.2.1 Index

typedef struct index_ Index

Index structure.

5.15.2.2 Index_Pool

typedef struct index_pool Index_Pool

Index pool.

5.15.3 Function Documentation

5.15.3.1 new_ipool()

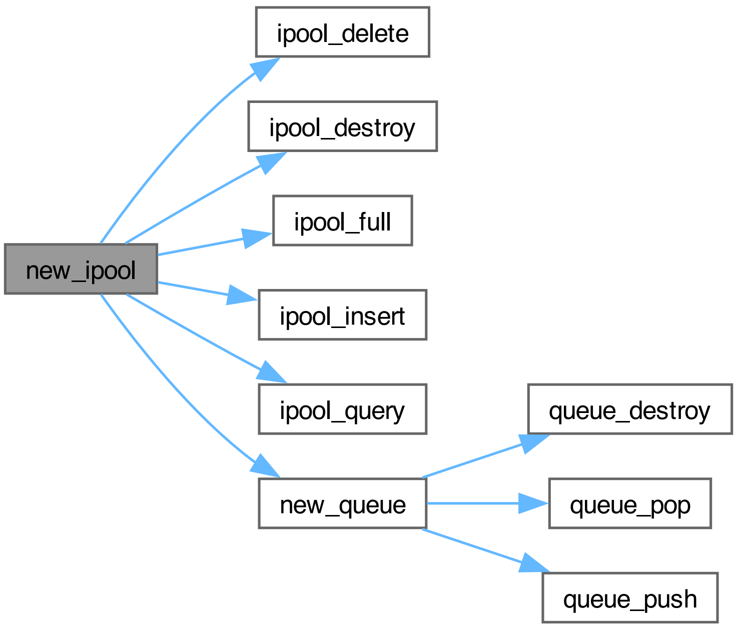
Index_Pool * new_ipool ()

Create a new index pool.

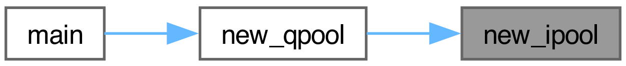
Returns

The new index pool

Here is the call graph for this function:



Here is the caller graph for this function:



5.16 index_pool.h

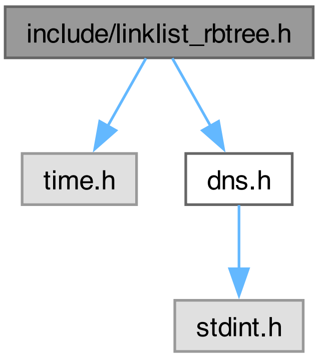
Go to the documentation of this file.

```
#ifndef DNSR_INDEX_POOL_H
#define DNSR_INDEX_POOL_H

#include <stdbool.h>

#include "queue.h"

#define INDEX_POOL_MAX_SIZE 65535

typedef struct index_
{
    uint16_t id;
    uint16_t prev_id;
} Index;

typedef struct index_pool
{
    Index * pool[INDEX_POOL_MAX_SIZE];
    unsigned short count;
    Queue * queue;

    bool (* full)(struct index_pool * ipool);

    uint16_t (* insert)(struct index_pool * ipool, Index * req);

    bool (* query)(struct index_pool * ipool, uint16_t index);

    Index * (* delete)(struct index_pool * ipool, uint16_t index);

    void (* destroy)(struct index_pool * ipool);
} Index_Pool;

Index_Pool * new_ipool();

#endif //DNSR_INDEX_POOL_H
```

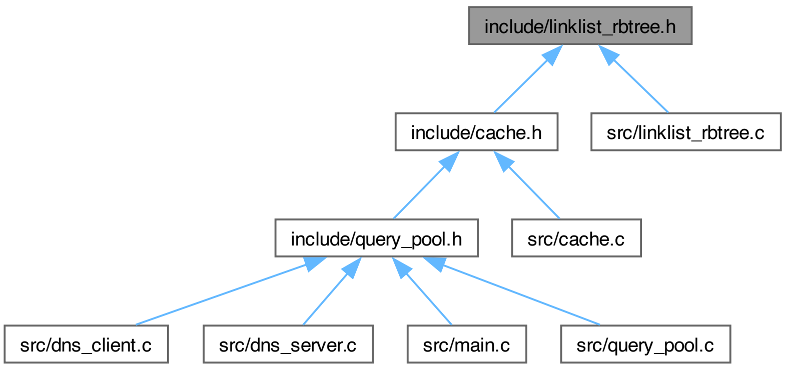
5.17 include/linklist_rbtree.h File Reference

#include <time.h>
#include "dns.h"

Include dependency graph for linklist_rbtree.h:



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



Data Structures

- struct rbtree_value

  Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.
- struct dns_rr_linklist

  Linked list of Red-Black Tree nodes.
- struct rbtree_node

  Node of the Red-Black Tree.
- struct linklist_rbtree

  Red-Black Tree.

Typedefs

- typedef struct rbtree_value Rbtree_Value

  Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.
- typedef struct dns_rr_linklist Dns_RR_LinkList

  Linked list of Red-Black Tree nodes.
- typedef struct rbtree_node Rbtree_Node

  Node of the Red-Black Tree.
- typedef struct linklist_rbtree Rbtree

  Red-Black Tree.

Enumerations

- enum Color { BLACK , RED }

  Red-Black Tree color.

Functions

- Dns_RR_LinkList ∗ new_linklist ()

  Create a new linked list.
- Rbtree ∗ new_rbtree ()

  Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

5.17.1 Typedef Documentation

5.17.1.1 Dns_RR_LinkList

typedef struct dns_rr_linklist Dns_RR_LinkList

Linked list of Red-Black Tree nodes.

5.17.1.2 Rbtree

typedef struct linklist_rbtree Rbtree

Red-Black Tree.

5.17.1.3 Rbtree_Node

typedef struct rbtree_node Rbtree_Node

Node of the Red-Black Tree.

5.17.1.4 Rbtree_Value

typedef struct rbtree_value Rbtree_Value

Value of a Red-Black Tree node's linked list, corresponding to an answer for a specific query.

5.17.2 Enumeration Type Documentation

5.17.2.1 Color

enum Color

Red-Black Tree color.

Enumerator

BLACK	
RED	

5.17.3 Function Documentation

5.17.3.1 new_linklist()

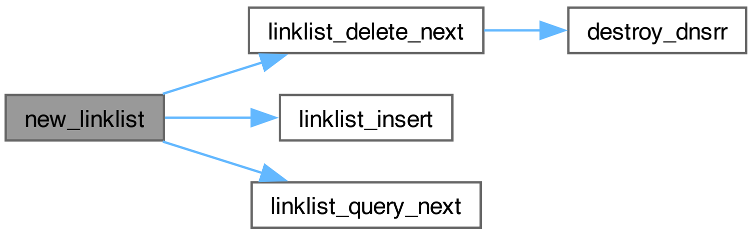
Dns_RR_LinkList ∗ new_linklist ()

Create a new linked list.

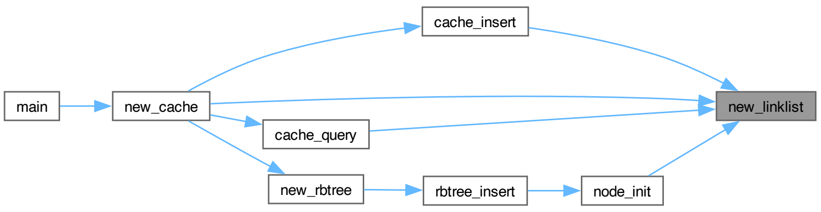
Returns

The new linked list

Here is the call graph for this function:



Here is the caller graph for this function:



5.17.3.2 new__rbtree()

Rbtree ∗ new__rbtree ()

Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

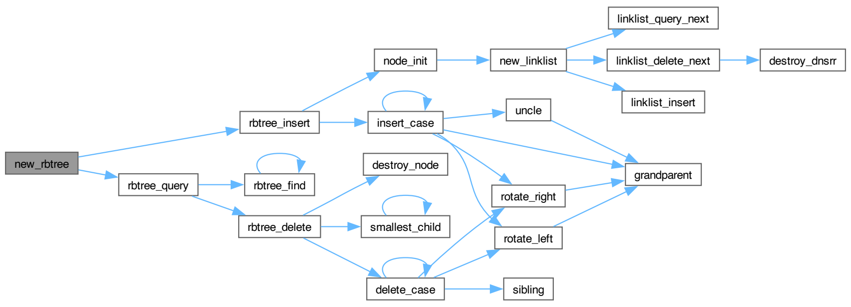
Returns

A pointer to the newly created red-black tree

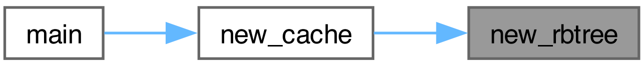
Note

If memory allocation fails, the function will log a fatal error and terminate the program.

Here is the call graph for this function:



Here is the caller graph for this function:



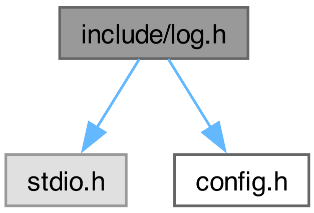
5.18 linklist_rbtree.h

Go to the documentation of this file.

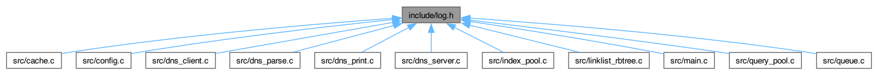
```
#ifndef DNSR_LINKLIST_RBTREE_H
#define DNSR_LINKLIST_RBTREE_H

#include <time.h>

#include "dns.h"

typedef enum {
    BLACK, RED
} Color;

typedef struct rbtree_value {
    Dns_RR *rr;
    uint16_t ancount;
    uint16_t nscount;
    uint16_t arcount;
    uint8_t type;
} Rbtree_Value;

typedef struct dns_rr_linklist {
    Rbtree_Value *value;
    time_t expire_time;
    struct dns_rr_linklist *next;

    void (*insert)(struct dns_rr_linklist *list, struct dns_rr_linklist *new_list_node);

    void (*delete_next)(struct dns_rr_linklist *list);

    struct dns_rr_linklist *(*query_next)(struct dns_rr_linklist *list, const uint8_t *qname, const uint16_t qtype);
} Dns_RR_LinkList;

typedef struct rbtree_node {
    unsigned int key;
    Dns_RR_LinkList *rr_list;
    Color color;
    struct rbtree_node *left;
    struct rbtree_node *right;
    struct rbtree_node *parent;
} Rbtree_Node;

typedef struct linklist_rbtree {
    Rbtree_Node *root;

    void (*insert)(struct linklist_rbtree *tree, unsigned int key, Dns_RR_LinkList *list);

    Dns_RR_LinkList *(*query)(struct linklist_rbtree *tree, unsigned int data);
} Rbtree;

Dns_RR_LinkList *new_linklist();

Rbtree *new_rbtree();

#endif //DNSR_LINKLIST_RBTREE_H
```

5.19 include/log.h File Reference

#include <stdio.h>
#include "config.h"
Include dependency graph for log.h:



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



Macros

- #define log_debug(args...)
- #define log_info(args...)
- #define log_error(args...)
- #define log_fatal(args...)

Variables

- FILE ∗ log_file

5.19.1 Macro Definition Documentation

5.19.1.1 log_debug

```
#define log_debug(
                args...)
```

Value:

```
if (LOG_MASK & 1) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[DEBUG] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[37m[DEBUG]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
}
```

5.19.1.2 log_error

```
#define log_error(
                args...)
```

Value:

```
if (LOG_MASK & 4) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[ERROR] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[33m[ERROR]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
}
```

5.19.1.3 log_fatal

```
#define log_fatal(
                    args...)
```

Value:

```
if (LOG_MASK & 8) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[FATAL] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[31m[FATAL]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
    exit(EXIT_FAILURE); \
}
```

5.19.1.4 log_info

```
#define log_info(
                    args...)
```

Value:

```
if (LOG_MASK & 2) \
{ \
    if (log_file != stderr) \
        fprintf(log_file, "[INFO ] %s:%d ", __FILE__, __LINE__); \
    else \
        fprintf(log_file, "\x1b[34m[INFO ]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
    fprintf(log_file, args); \
    fprintf(log_file, "\n"); \
}
```

5.19.2 Variable Documentation

5.19.2.1 log_file

FILE∗ log_file [extern]

5.20 log.h

Go to the documentation of this file.

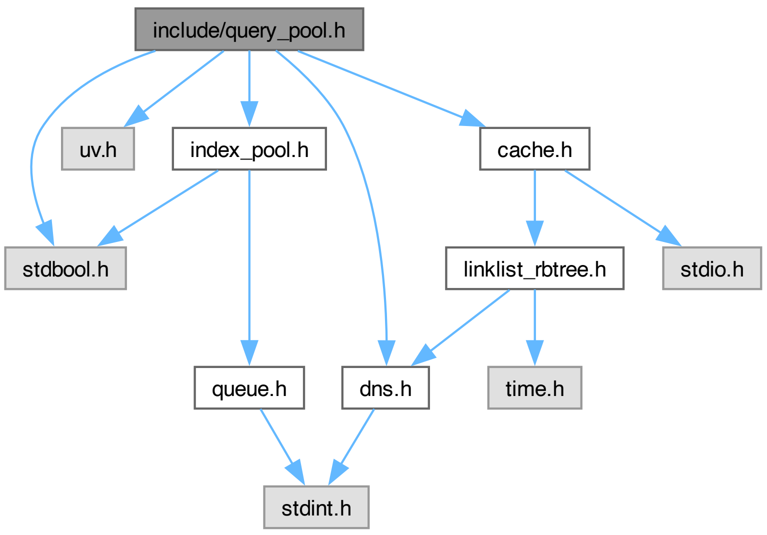
```
#ifndef DNSR_LOG_H
#define DNSR_LOG_H

#include <stdio.h>

#include "config.h"

extern FILE * log_file;

```

```
#define log_debug(args...) \
    if (LOG_MASK & 1) \
    { \
        if (log_file != stderr) \
            fprintf(log_file, "[DEBUG] %s:%d ", __FILE__, __LINE__); \
        else \
            fprintf(log_file, "\x1b[37m[DEBUG]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
        fprintf(log_file, args); \
        fprintf(log_file, "\n"); \
    }

#define log_info(args...) \
    if (LOG_MASK & 2) \
    { \
        if (log_file != stderr) \
            fprintf(log_file, "[INFO ] %s:%d ", __FILE__, __LINE__); \
        else \
            fprintf(log_file, "\x1b[34m[INFO ]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
        fprintf(log_file, args); \
        fprintf(log_file, "\n"); \
    }

#define log_error(args...) \
    if (LOG_MASK & 4) \
    { \
        if (log_file != stderr) \
            fprintf(log_file, "[ERROR] %s:%d ", __FILE__, __LINE__); \
        else \
            fprintf(log_file, "\x1b[33m[ERROR]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
        fprintf(log_file, args); \
        fprintf(log_file, "\n"); \
    }

#define log_fatal(args...) \
    if (LOG_MASK & 8) \
    { \
        if (log_file != stderr) \
            fprintf(log_file, "[FATAL] %s:%d ", __FILE__, __LINE__); \
        else \
            fprintf(log_file, "\x1b[31m[FATAL]\x1b[36m %s:%d \x1b[0m", __FILE__, __LINE__); \
        fprintf(log_file, args); \
        fprintf(log_file, "\n"); \
        exit(EXIT_FAILURE); \
    }

#endif //DNSR_LOG_H
```

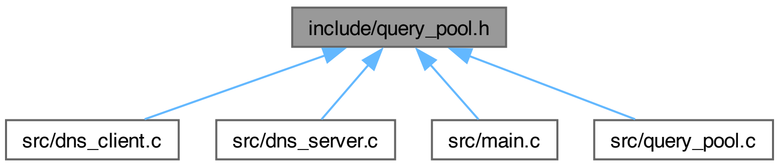
5.21 include/query_pool.h File Reference

```
#include <stdbool.h>
#include <uv.h>
#include "dns.h"
#include "index_pool.h"
#include "cache.h"
```

Include dependency graph for query_pool.h:



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



Data Structures

- struct dns_query

  DNS query structure.
- struct query_pool

  DNS query pool.

Macros

- #define QUERY_POOL_MAX_SIZE 256

Typedefs

- typedef struct dns_query Dns_Query

  DNS query structure.
- typedef struct query_pool Query_Pool

  DNS query pool.

Functions

- Query_Pool ∗ new_qpool (uv_loop_t ∗loop, Cache ∗cache)

  Create a new query pool This function initializes a new query pool and returns a pointer to it.

5.21.1 Macro Definition Documentation

5.21.1.1 QUERY_POOL_MAX_SIZE

```
#define QUERY_POOL_MAX_SIZE 256
```

5.21.2 Typedef Documentation

5.21.2.1 Dns_Query

```
typedef struct dns_query Dns_Query
```

DNS query structure.

5.21.2.2 Query_Pool

```
typedef struct query_pool Query_Pool
```

DNS query pool.

5.21.3 Function Documentation

5.21.3.1 new_qpool()

```
Query_Pool ∗ new_qpool (
                uv_loop_t ∗ loop,
                Cache ∗ cache)
```

Create a new query pool This function initializes a new query pool and returns a pointer to it.

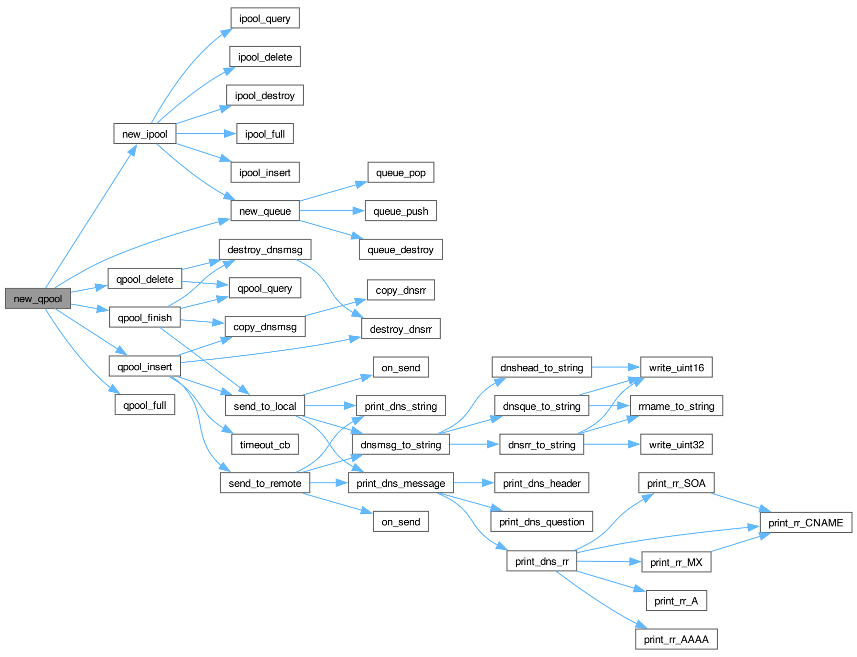
Parameters

loop	The libuv event loop
cache	The cache used for storing DNS responses

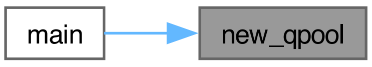
Returns

A pointer to the newly created query pool

Here is the call graph for this function:



Here is the caller graph for this function:



5.22 query_pool.h

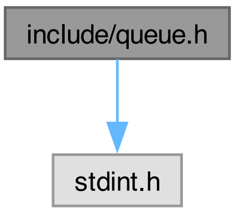
Go to the documentation of this file.

```c
#ifndef DNSR_QUERY_POOL_H
#define DNSR_QUERY_POOL_H

#include <stdbool.h>
#include <uv.h>

#include "dns.h"
#include "index_pool.h"
#include "cache.h"

#define QUERY_POOL_MAX_SIZE 256

typedef struct dns_query {
    uint16_t id;
    uint16_t prev_id;
    struct sockaddr addr;
    Dns_Msg * msg;
    uv_timer_t timer;
} Dns_Query;

typedef struct query_pool {
    Dns_Query * pool[QUERY_POOL_MAX_SIZE];
    unsigned short count;
    Queue * queue;
    Index_Pool * ipool;
    uv_loop_t * loop;
    Cache * cache;

    bool (* full)(struct query_pool * qpool);

    void (* insert)(struct query_pool * qpool, const struct sockaddr * addr, const Dns_Msg * msg);

    void (* finish)(struct query_pool * qpool, const Dns_Msg * msg);

    void (* delete)(struct query_pool * qpool, uint16_t id);
} Query_Pool;

Query_Pool *new_qpool(uv_loop_t * loop, Cache * cache);

#endif //DNSR_QUERY_POOL_H
```

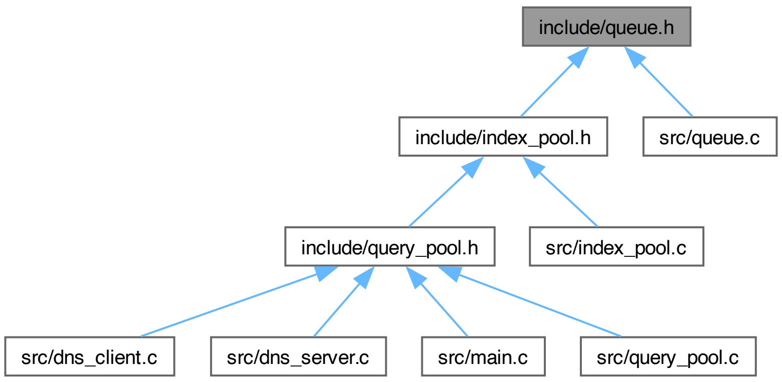
5.23 include/queue.h File Reference

#include <stdint.h>

Include dependency graph for queue.h:



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



Data Structures

- struct queue

  Circular queue.

Macros

- #define QUEUE_MAX_SIZE 65536

Typedefs

- typedef struct queue Queue

  Circular queue.

Functions

- Queue ∗ new_queue ()

  Create a new queue.

5.23.1 Macro Definition Documentation

5.23.1.1 QUEUE_MAX_SIZE

```
#define QUEUE_MAX_SIZE 65536
```

5.23.2 Typedef Documentation

5.23.2.1 Queue

typedef struct queue Queue

Circular queue.

5.23.3 Function Documentation

5.23.3.1 new_queue()

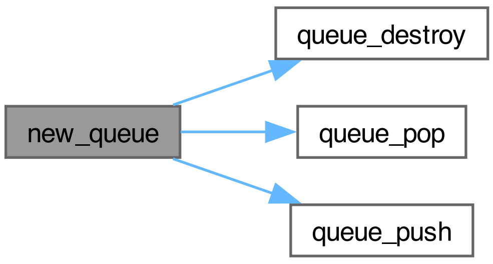
Queue ∗ new_queue ()

Create a new queue.

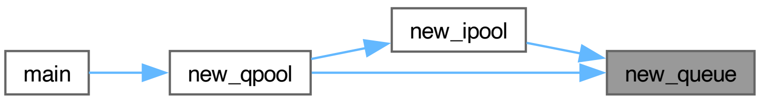
Returns

The new queue

Here is the call graph for this function:



Here is the caller graph for this function:



5.24 queue.h

Go to the documentation of this file.

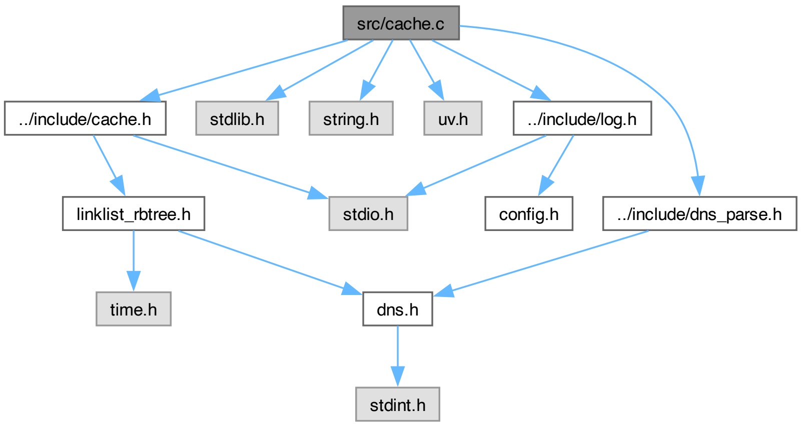
```
#include <stdint.h>

#ifndef DNSR_QUEUE_H
#define DNSR_QUEUE_H

#define QUEUE_MAX_SIZE 65536

typedef struct queue
{
    uint16_t q[QUEUE_MAX_SIZE];
    unsigned short head;
    unsigned short tail;

    void (* push)(struct queue * queue, uint16_t num);

    uint16_t (* pop)(struct queue * queue);

    void (* destroy)(struct queue * queue);
} Queue;

Queue * new_queue();

#endif //DNSR_QUEUE_H
```

5.25 README.md File Reference

5.26 src/cache.c File Reference

#include "../include/cache.h"
#include <stdlib.h>
#include <string.h>
#include <uv.h>
#include "../include/log.h"
#include "../include/dns_parse.h"

Include dependency graph for cache.c:



Functions

- static unsigned int BKDRHash (const uint8_t ∗str)

  Compute the hash of a string using the BKDR hash algorithm.
- static uint32_t get_min_ttl (const Dns_RR ∗prr)

  Get the smallest TTL (Time-To-Live) value in a list of Resource Records (RRs).
- static void cache_insert (Cache ∗cache, const Dns_Msg ∗msg)

  Insert a DNS message into the cache.
- static Rbtree_Value ∗ cache_query (Cache ∗cache, const Dns_Que ∗que)

  Query the cache for a DNS question.
- Cache ∗ new_cache (FILE ∗hosts_file)

  Create a new cache and initialize it with data from the hosts file.

5.26.1 Function Documentation

5.26.1.1 BKDRHash()

static unsigned int BKDRHash (

const uint8_t ∗ str) [static]

Compute the hash of a string using the BKDR hash algorithm.

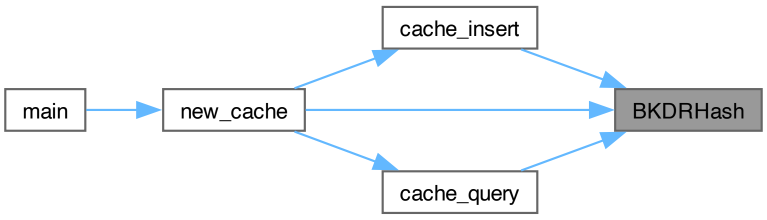
Parameters

str	The input string.

Returns

The computed hash value.

Here is the caller graph for this function:



5.26.1.2 cache_insert()

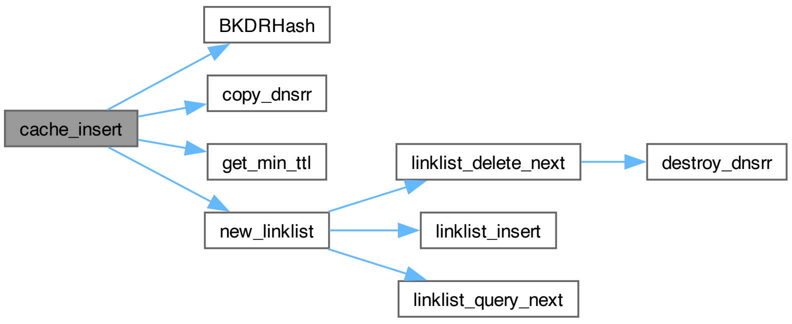
static void cache_insert (
Cache ∗ cache,
const Dns_Msg ∗ msg) [static]

Insert a DNS message into the cache.

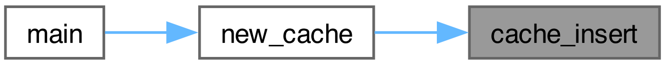
Parameters

cache	The cache where the message will be inserted.
msg	The DNS message to be inserted.

Here is the call graph for this function:



Here is the caller graph for this function:



5.26.1.3 cache_query()

static Rbtree_Value ∗ cache_query (
        Cache ∗ cache,
        const Dns_Que ∗ que) [static]

Query the cache for a DNS question.

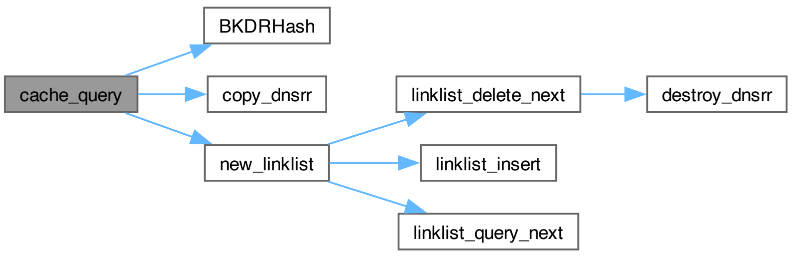
Parameters

cache	The cache to query.
que	The DNS question.

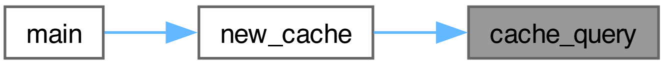
Returns

The value found in the cache or NULL if not found.

Here is the call graph for this function:



Here is the caller graph for this function:



5.26.1.4 get_min_ttl()

static uint32_t get_min_ttl (
    const Dns_RR ∗ prr) [static]

Get the smallest TTL (Time-To-Live) value in a list of Resource Records (RRs).

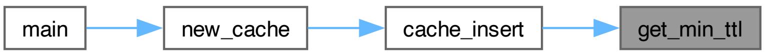
Parameters

prr	The head node of the RR linked list.

Returns

The minimum TTL value.

Here is the caller graph for this function:



5.26.1.5 new_cache()

Cache ∗ new_cache (
    FILE ∗ hosts_file)

Create a new cache and initialize it with data from the hosts file.

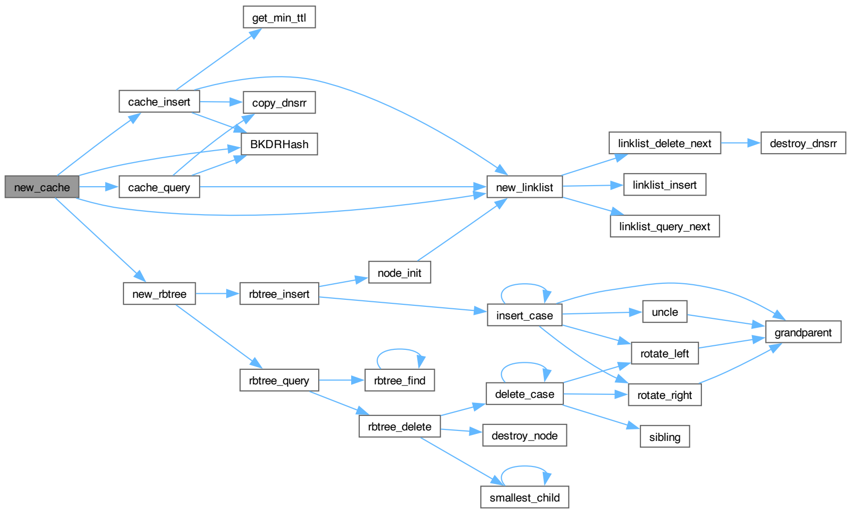
Parameters

hosts_file	The file containing hosts data.

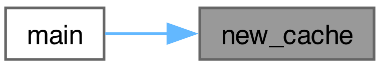
Returns

The newly created cache.

Here is the call graph for this function:



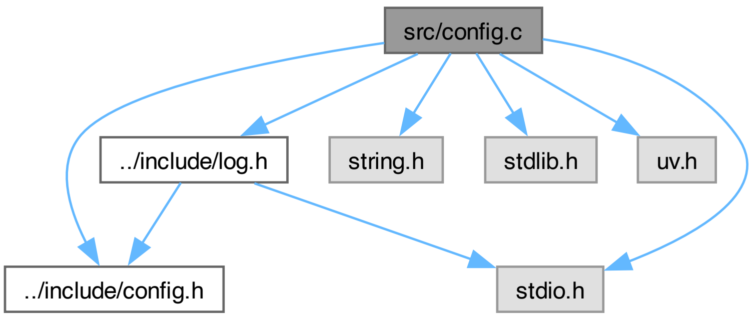
Here is the caller graph for this function:



5.27 src/config.c File Reference

#include "../include/config.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <uv.h>
#include "../include/log.h"
Include dependency graph for config.c:



Functions

- void init_config (int argc, char ∗const ∗argv)

  Parse command line arguments.

Variables

- char ∗ REMOTE_HOST = "8.8.8.8"

  Remote DNS server address.
- int LOG_MASK = 15

  Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.
- int CLIENT_PORT = 0

  Local DNS client port.
- char ∗ HOSTS_PATH = "../dnsrelay.txt"

  Hosts file path.
- char ∗ LOG_PATH = NULL

  Log file path.

5.27.1 Function Documentation

5.27.1.1 init_config()

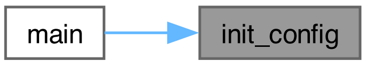
```
void init_config (
                    int argc,
                    char ∗const ∗ argv)
```

Parse command line arguments.

Parameters

argc	Number of arguments
argv	Array of argument strings

Here is the caller graph for this function:



5.27.2 Variable Documentation

5.27.2.1 CLIENT_PORT

int CLIENT_PORT = 0

Local DNS client port.

5.27.2.2 HOSTS_PATH

char∗ HOSTS_PATH = "../dnsrelay.txt"

Hosts file path.

5.27.2.3 LOG_MASK

int LOG_MASK = 15

Log print level, a four-bit binary number where the lowest to highest bits represent FATAL, ERROR, INFO and DEBUG.

5.27.2.4 LOG_PATH

char∗ LOG_PATH = NULL

Log file path.

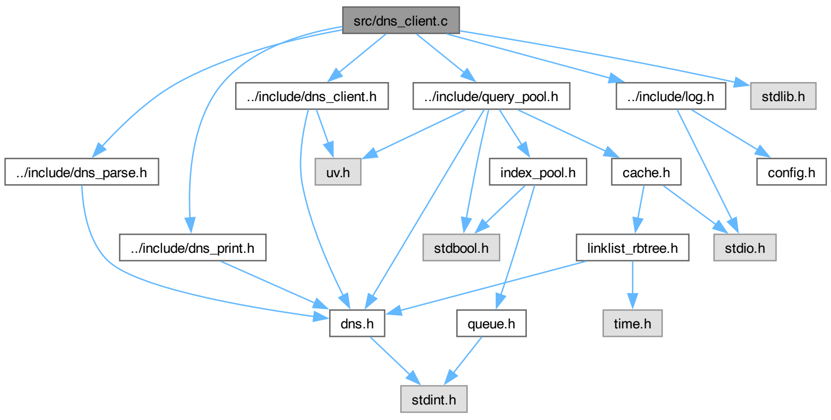
5.27.2.5 REMOTE_HOST

char∗ REMOTE_HOST = "8.8.8.8"

Remote DNS server address.

5.28 src/dns_client.c File Reference

#include "../include/dns_client.h"
#include <stdlib.h>
#include "../include/log.h"
#include "../include/dns_parse.h"
#include "../include/dns_print.h"
#include "../include/query_pool.h"
Include dependency graph for dns_client.c:



Functions

- static void alloc_buffer (uv_handle_t ∗handle, size_t suggested_size, uv_buf_t ∗buf)

  Allocate space for the buffer.
- static void on_read (uv_udp_t ∗handle, ssize_t nread, const uv_buf_t ∗buf, const struct sockaddr ∗addr, unsigned flags)

  Callback function for receiving response messages from the remote server.
- static void on_send (uv_udp_send_t ∗req, int status)

  Callback function for sending query messages to the remote server.
- void init_client (uv_loop_t ∗loop)

  Initialize the DNS client.
- void send_to_remote (const Dns_Msg ∗msg)

  Send a DNS query message to the remote server.

Variables

- static uv_udp_t client_socket

  Socket for client communication with the remote server.
- static struct sockaddr_in local_addr

  Local address.
- static struct sockaddr send_addr

  Remote server address.
- Query_Pool ∗ qpool

  Query pool.

5.28.1 Function Documentation

5.28.1.1 alloc_buffer()

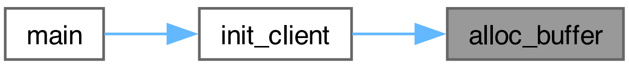
```
static void alloc_buffer (
                uv_handle_t ∗ handle,
                size_t suggested_size,
                uv_buf_t ∗ buf) [static]
```

Allocate space for the buffer.

Parameters

handle	Allocation handle
suggested_size	Suggested buffer size
buf	Buffer to be allocated

Here is the caller graph for this function:



5.28.1.2 init_client()

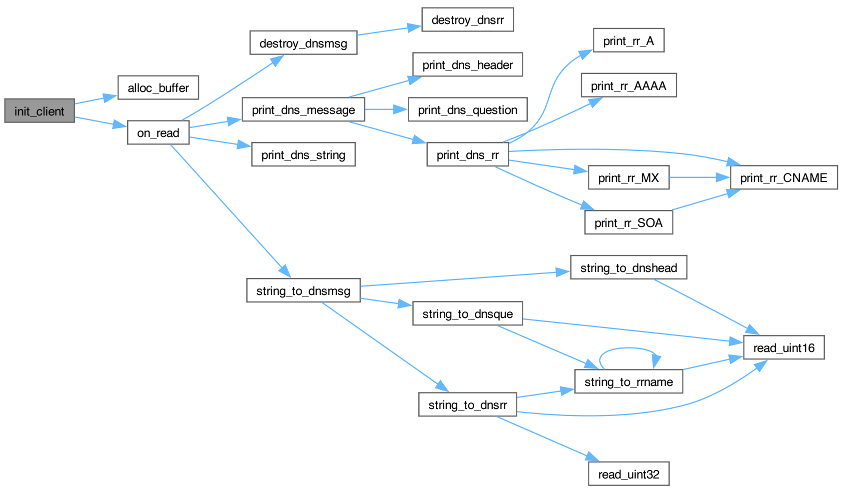
```
void init_client (
                uv_loop_t ∗ loop)
```

Initialize the DNS client.

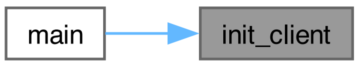
Parameters

loop	The libuv event loop

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.3 on_read()

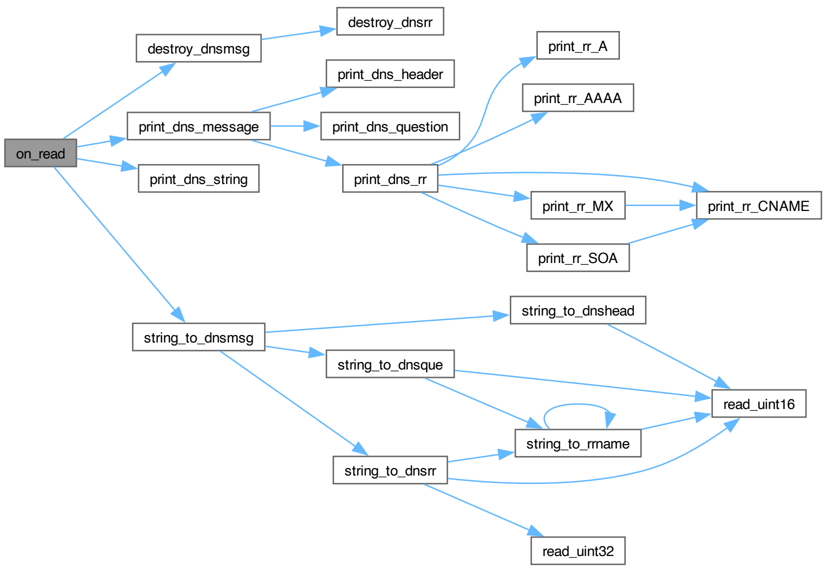
```
static void on_read (
        uv_udp_t * handle,
        ssize_t nread,
        const uv_buf_t * buf,
        const struct sockaddr * addr,
        unsigned flags) [static]
```

Callback function for receiving response messages from the remote server.

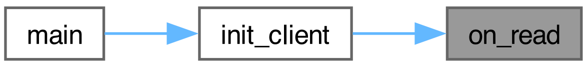
Parameters

handle	Query handle
nread	Number of bytes received
buf	Buffer containing the received message
addr	Address of the sender
flags	Flags indicating special conditions for the received data

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.4 on_send()

static void on_send (

uv_udp_send_t ∗ req,

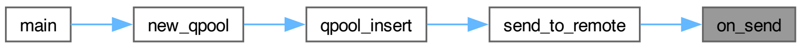
int status) [static]

Callback function for sending query messages to the remote server.

Parameters

req	Send handle
status	Send status, indicating whether the send was successful

Here is the caller graph for this function:



5.28.1.5 send_to_remote()

void send_to_remote (

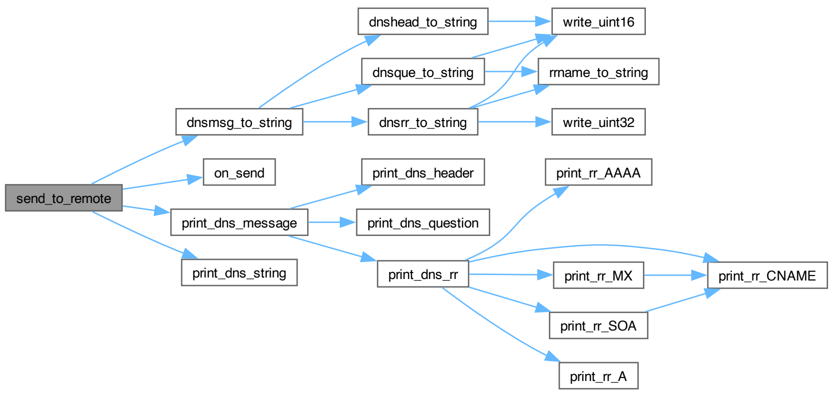
const Dns_Msg * msg)

Send a DNS query message to the remote server.

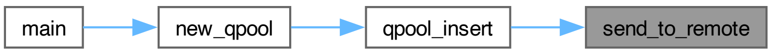
Parameters

msg	The DNS message to be sent

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.2 Variable Documentation

5.28.2.1 client_socket

uv_udp_t client_socket [static]

Socket for client communication with the remote server.

5.28.2.2 local_addr

struct sockaddr_in local_addr [static]

Local address.

5.28.2.3 qpool

Query_Pool∗ qpool [extern]

Query pool.

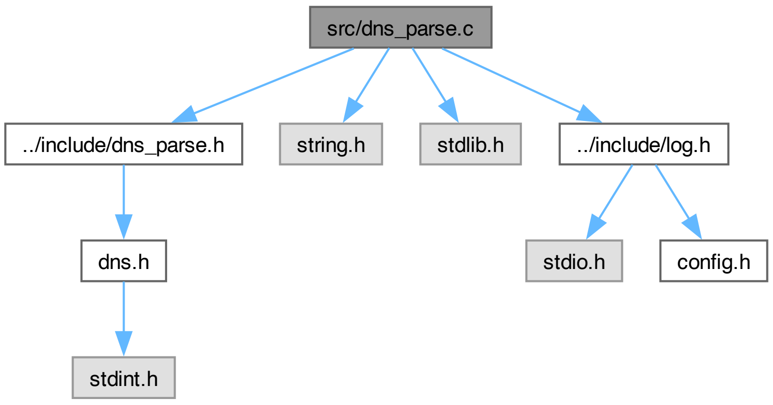
5.28.2.4 send_addr

struct sockaddr send_addr [static]

Remote server address.

5.29 src/dns_parse.c File Reference

#include "../include/dns_parse.h"
#include <string.h>
#include <stdlib.h>
#include "../include/log.h"
Include dependency graph for dns_parse.c:



Functions

- static uint16_t read_uint16 (const char ∗pstring, unsigned ∗offset)

  Read a 16-bit number in big-endian format from a byte stream.
- static uint32_t read_uint32 (const char ∗pstring, unsigned ∗offset)

  Read a 32-bit number in big-endian format from a byte stream.
- static unsigned string_to_rrname (uint8_t ∗pname, const char ∗pstring, unsigned ∗offset)

  Read a NAME field from a byte stream.
- static void string_to_dnshead (Dns_Header ∗phead, const char ∗pstring, unsigned ∗offset)

  Read a Header Section from a byte stream.
- static void string_to_dnsque (Dns_Que ∗pque, const char ∗pstring, unsigned ∗offset)

  Read a Question Section from a byte stream.
- static void string_to_dnsrr (Dns_RR ∗prr, const char ∗pstring, unsigned ∗offset)

  Read a Resource Record from a byte stream.
- void string_to_dnsmsg (Dns_Msg ∗pmsg, const char ∗pstring)

  Convert a byte stream to a DNS message structure.
- static void write_uint16 (const char ∗pstring, unsigned ∗offset, uint16_t num)

  Write a 16-bit number in little-endian format to a byte stream.

- static void write_uint32 (const char ∗pstring, unsigned ∗offset, uint32_t num)

  Write a 32-bit number in little-endian format to a byte stream.
- static void rrname_to_string (const uint8_t ∗pname, char ∗pstring, unsigned ∗offset)

  Write a NAME field to a byte stream.
- static void dnshead_to_string (const Dns_Header ∗phead, char ∗pstring, unsigned ∗offset)

  Write a Header Section to a byte stream.
- static void dnsque_to_string (const Dns_Que ∗pque, char ∗pstring, unsigned ∗offset)

  Write a Question Section to a byte stream.
- static void dnsrr_to_string (const Dns_RR ∗prr, char ∗pstring, unsigned ∗offset)

  Write a Resource Record to a byte stream.
- unsigned dnsmsg_to_string (const Dns_Msg ∗pmsg, char ∗pstring)

  Convert a DNS message structure to a byte stream.
- void destroy_dnsrr (Dns_RR ∗prr)

  Release memory allocated for a Resource Record.
- void destroy_dnsmsg (Dns_Msg ∗pmsg)

  Release memory allocated for a DNS message.
- Dns_RR ∗ copy_dnsrr (const Dns_RR ∗src)

  Copy a Resource Record.
- Dns_Msg ∗ copy_dnsmsg (const Dns_Msg ∗src)

  Copy a DNS message.

5.29.1 Function Documentation

5.29.1.1 copy_dnsmsg()

```
Dns_Msg ∗ copy_dnsmsg (
            const Dns_Msg ∗ src)
```

Copy a DNS message.

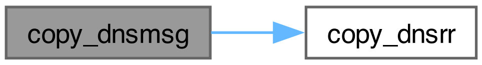
Parameters

src	The DNS message to copy

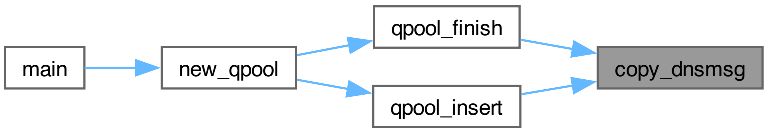
Returns

A copy of the DNS message

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.2 copy_dnsrr()

Dns_RR ∗ copy_dnsrr (
    const Dns_RR ∗ src)

Copy a Resource Record.

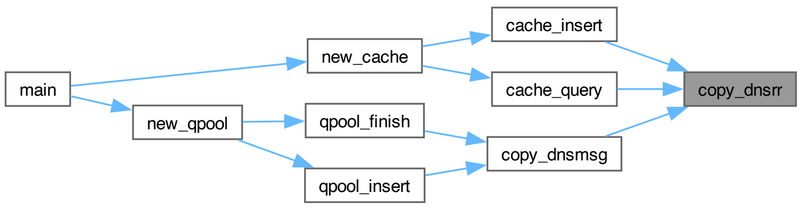
Parameters

src	The Resource Record to copy

Returns

A copy of the Resource Record

Here is the caller graph for this function:



5.29.1.3 destroy_dnsmsg()

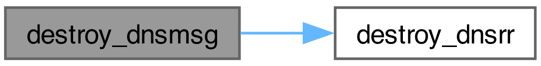
void destroy_dnsmsg (
    Dns_Msg ∗ pmsg)

Release memory allocated for a DNS message.

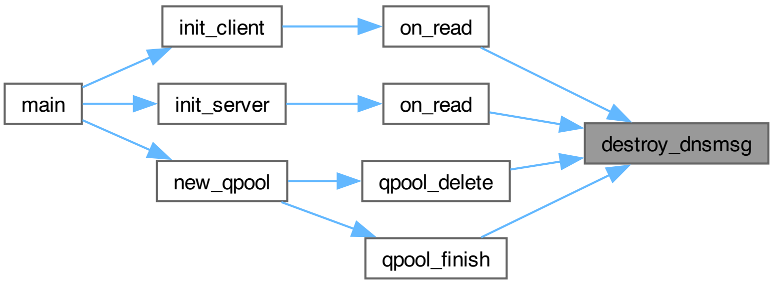
Parameters

pmsg	The DNS message to release

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.4 destroy_dnsrr()

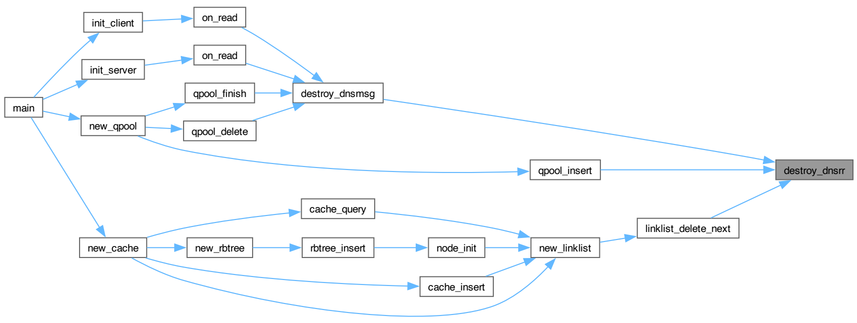
```
void destroy_dnsrr (
        Dns_RR * prr)
```

Release memory allocated for a Resource Record.

Parameters

prr	The Resource Record to release

Here is the caller graph for this function:



5.29.1.5 dnshead_to_string()

static void dnshead_to_string (
const Dns_Header ∗ phead,
char ∗ pstring,
unsigned ∗ offset) [static]

Write a Header Section to a byte stream.

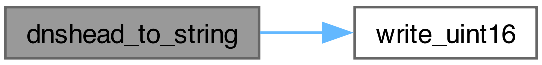
Parameters

phead	The Header Section
pstring	The start of the byte stream
offset	The offset in the byte stream

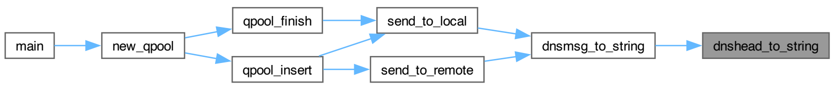
Note

After writing, the offset increases to the position after the Header Section

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.6 dnsmsg_to_string()

unsigned dnsmsg_to_string (
const Dns_Msg ∗ pmsg,
char ∗ pstring)

Convert a DNS message structure to a byte stream.

Write a NAME field to a byte stream.

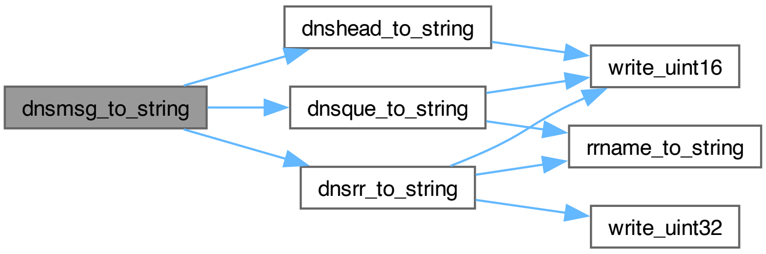
Parameters

pmsg	The DNS message structure to convert
pstring	The byte stream to write to

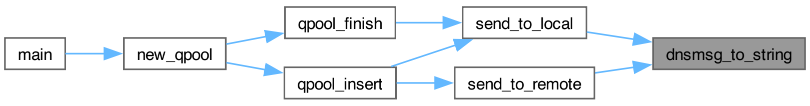
Returns

The total length of the byte stream

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.7 dnsque_to_string()

static void dnsque_to_string (
const Dns_Que ∗ pque,
char ∗ pstring,
unsigned ∗ offset) [static]

Write a Question Section to a byte stream.

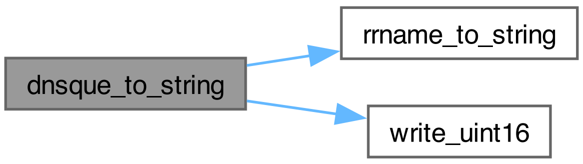
Parameters

pque	The Question Section
pstring	The start of the byte stream
offset	The offset in the byte stream

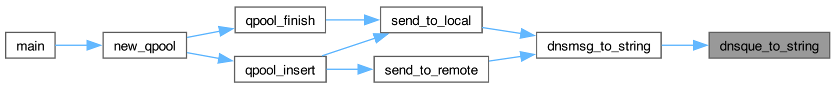
Note

After writing, the offset increases to the position after the Question Section

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.8 dnsrr_to_string()

static void dnsrr_to_string (
const Dns_RR ∗ prr,
char ∗ pstring,
unsigned ∗ offset) [static]

Write a Resource Record to a byte stream.

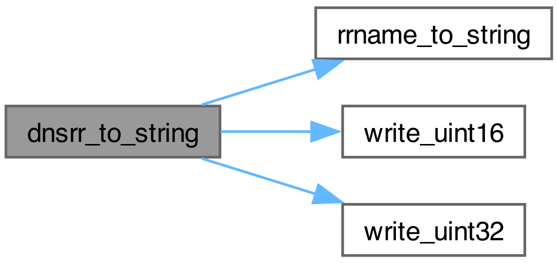
Parameters

prr	The Resource Record
pstring	The start of the byte stream
offset	The offset in the byte stream

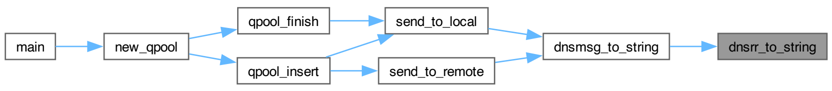
Note

After writing, the offset increases to the position after the Resource Record

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.9 read_uint16()

```
static uint16_t read_uint16 (
        const char * pstring,
        unsigned * offset) [static]
```

Read a 16-bit number in big-endian format from a byte stream.

Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream

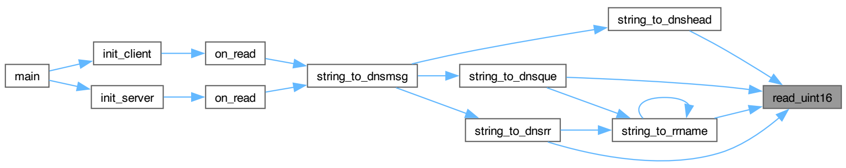
Returns

The 16-bit number starting from (pstring + *offset)

Note

After reading, the offset increases by 2

Here is the caller graph for this function:



5.29.1.10 read_uint32()

static uint32_t read_uint32 (
const char ∗ pstring,
unsigned ∗ offset) [static]

Read a 32-bit number in big-endian format from a byte stream.

Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream

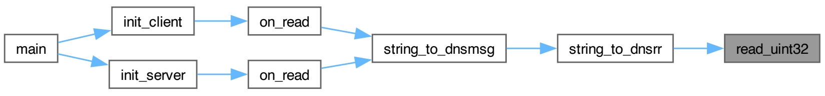
Returns

The 32-bit number starting from (pstring + ∗offset)

Note

After reading, the offset increases by 4

Here is the caller graph for this function:



5.29.1.11 rrname_to_string()

static void rrname_to_string (
    const uint8_t ∗ pname,
    char ∗ pstring,
    unsigned ∗ offset) [static]

Write a NAME field to a byte stream.

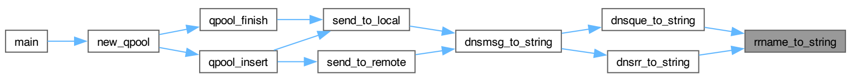
Parameters

pname	The NAME field
pstring	The start of the byte stream
offset	The offset in the byte stream

Note

After writing, the offset increases to the position after the NAME field

Here is the caller graph for this function:



5.29.1.12 string_to_dnshead()

static void string_to_dnshead (
    Dns_Header ∗ phead,
    const char ∗ pstring,
    unsigned ∗ offset) [static]

Read a Header Section from a byte stream.

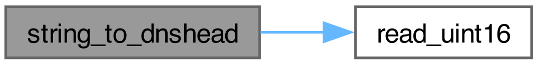
Parameters

phead	The Header Section
pstring	The start of the byte stream
offset	The offset in the byte stream

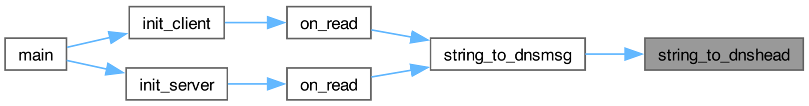
Note

After reading, the offset increases to the position after the Header Section

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.13 string_to_dnsmsg()

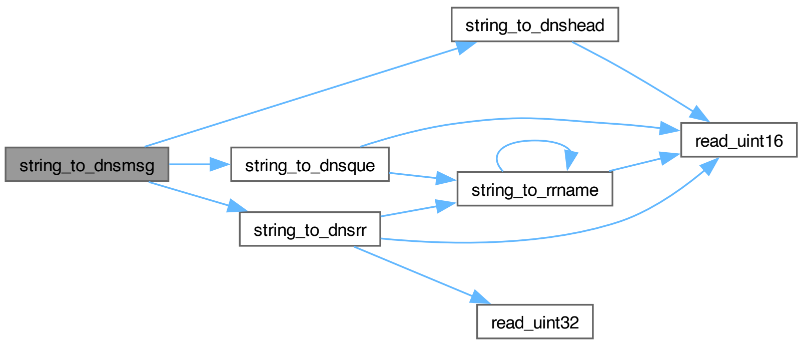
void string_to_dnsmsg (
Dns_Msg * pmsg,
const char * pstring)

Convert a byte stream to a DNS message structure.

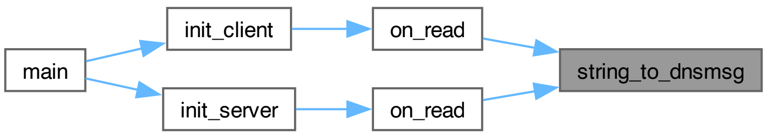
Parameters

pmsg	The DNS message structure to populate
pstring	The byte stream to read from

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.14 string_to_dnsque()

static void string_to_dnsque (
    Dns_Que ∗ pque,
    const char ∗ pstring,
    unsigned ∗ offset) [static]

Read a Question Section from a byte stream.

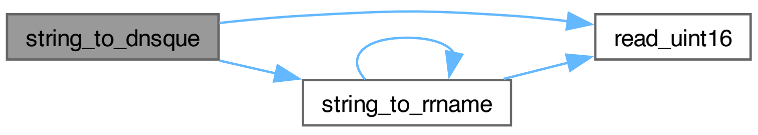
Parameters

pque	The Question Section
pstring	The start of the byte stream
offset	The offset in the byte stream

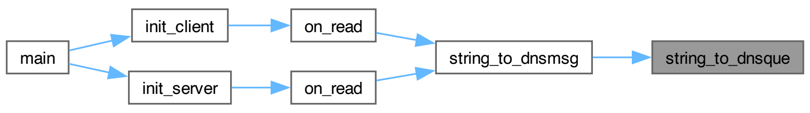
Note

After reading, the offset increases to the position after the Question Section; space is allocated for the NAME field

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.15 string_to_dnsrr()

```
static void string_to_dnsrr (
                    Dns_RR * prr,
                    const char * pstring,
                    unsigned * offset)   [static]
```

Read a Resource Record from a byte stream.

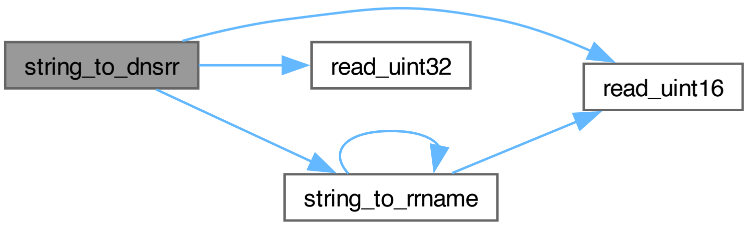
Parameters

prr	The Resource Record
pstring	The start of the byte stream
offset	The offset in the byte stream

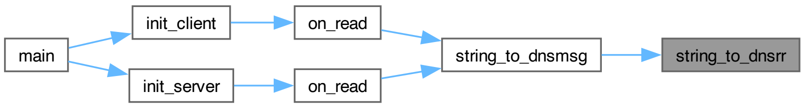
Note

After reading, the offset increases to the position after the Resource Record; space is allocated for the NAME and RDATA fields

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.16 string_to_rrname()

static unsigned string_to_rrname (
uint8_t * pname,
const char * pstring,
unsigned * offset) [static]

Read a NAME field from a byte stream.

Parameters

pname	The NAME field
pstring	The start of the byte stream
offset	The offset in the byte stream

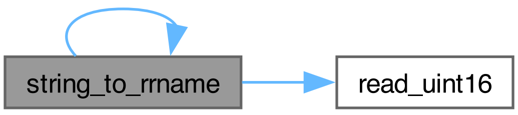
Returns

The total length of the NAME field

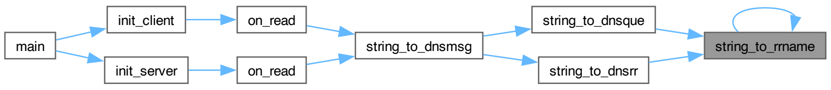
Note

After reading, the offset increases to the position after the NAME field

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.1.17 write_uint16()

static void write_uint16 (
const char ∗ pstring,
unsigned ∗ offset,
uint16_t num) [static]

Write a 16-bit number in little-endian format to a byte stream.

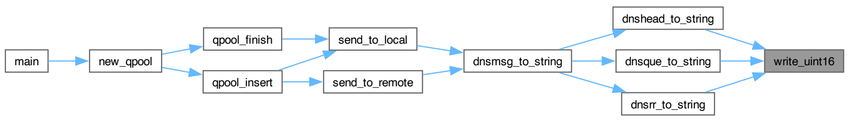
Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream
num	The number to write

Note

After writing, the offset increases by 2

Here is the caller graph for this function:



5.29.1.18 write_uint32()

static void write_uint32 (

const char ∗ pstring,

unsigned ∗ offset,

uint32_t num) [static]

Write a 32-bit number in little-endian format to a byte stream.

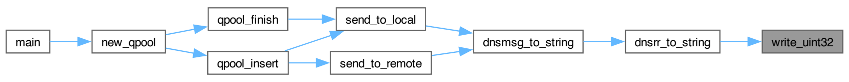
Parameters

pstring	The start of the byte stream
offset	The offset in the byte stream
num	The number to write

Note

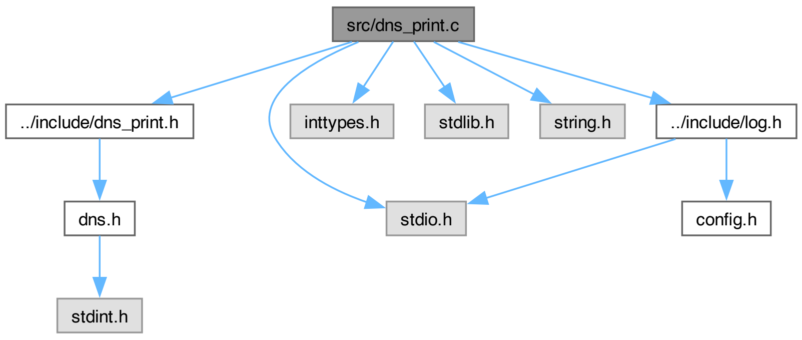
After writing, the offset increases by 4

Here is the caller graph for this function:



5.30 src/dns_print.c File Reference

#include "../include/dns_print.h"
#include <stdio.h>
#include <inttypes.h>
#include <stdlib.h>
#include <string.h>
#include "../include/log.h"
Include dependency graph for dns_print.c:



Functions

- void print_dns_string (const char ∗pstring, unsigned int len)

  Print DNS message byte stream.
- static void print_rr_A (const uint8_t ∗rdata)

  Print the rdata field of an A type RR.
- static void print_rr_AAAA (const uint8_t ∗rdata)

  Print the rdata field of an AAAA type RR.
- static void print_rr_CNAME (const uint8_t ∗rdata)

  Print the rdata field of a CNAME type RR.
- static void print_rr_SOA (uint16_t rdlength, const uint8_t ∗rdata)

  Print the rdata field of an SOA type RR.
- static void print_rr_MX (const uint8_t ∗rdata)

  Print the rdata field of an MX type RR.
- static void print_dns_header (const Dns_Header ∗phead)

  Print the Header Section.
- static void print_dns_question (const Dns_Que ∗pque)

  Print the Question Section.
- static void print_dns_rr (const Dns_RR ∗prr)

Print the Resource Record.

- void print_dns_message (const Dns_Msg *pmsg)

  Print the entire DNS message.

5.30.1 Function Documentation

5.30.1.1 print_dns_header()

static void print_dns_header (

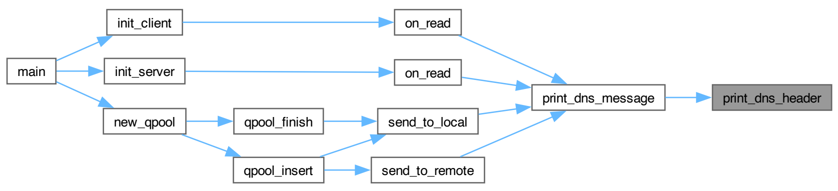
const Dns_Header * phead) [static]

Print the Header Section.

Parameters

phead	The Header Section

Here is the caller graph for this function:



5.30.1.2 print_dns_message()

void print_dns_message (

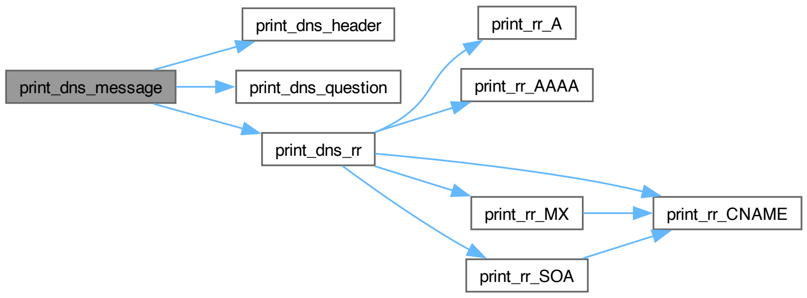
const Dns_Msg * pmsg)

Print the entire DNS message.

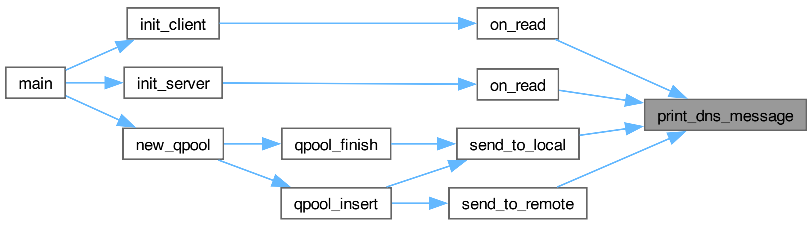
Parameters

pmsg	The DNS message

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.3 print_dns_question()

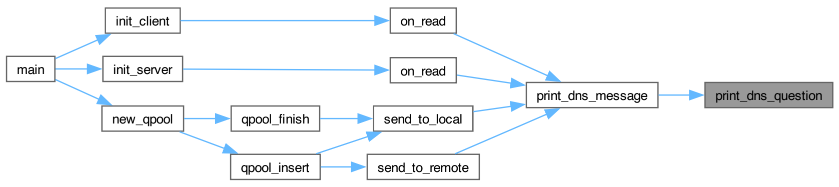
static void print_dns_question (
    const Dns_Que * pque) [static]

Print the Question Section.

Parameters

pque	The Question Section

Here is the caller graph for this function:



5.30.1.4 print_dns_rr()

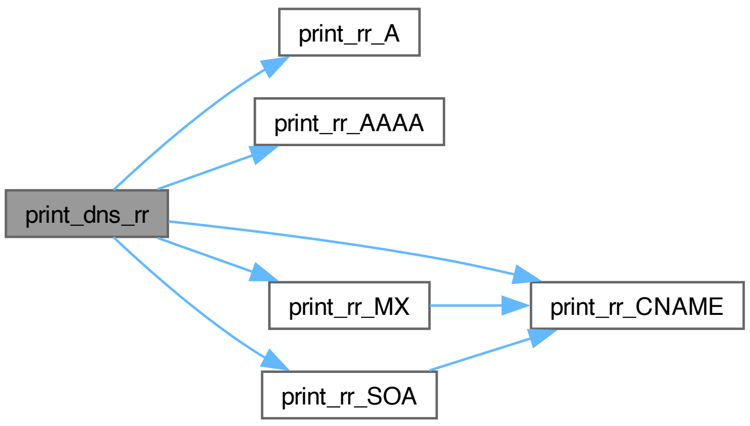
static void print_dns_rr (
const Dns_RR * prr) [static]

Print the Resource Record.

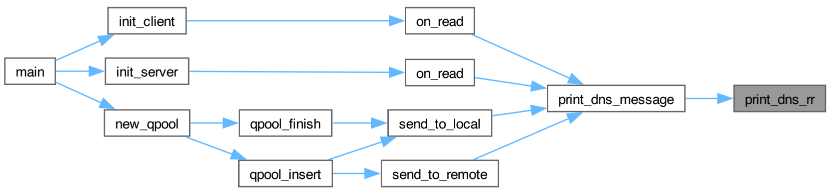
Parameters

prr	The Resource Record

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.5 print_dns_string()

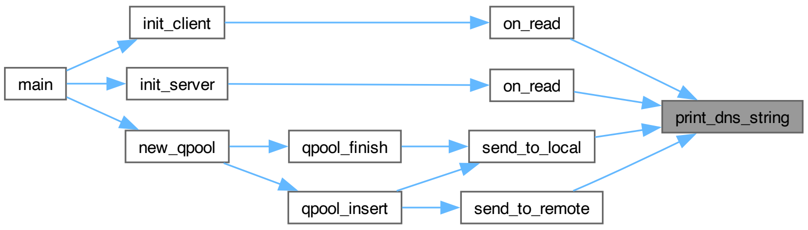
void print_dns_string (
    const char * pstring,
    unsigned int len)

Print DNS message byte stream.

Parameters

pstring	The byte stream
len	The length of the byte stream

Here is the caller graph for this function:



5.30.1.6 print_rr_A()

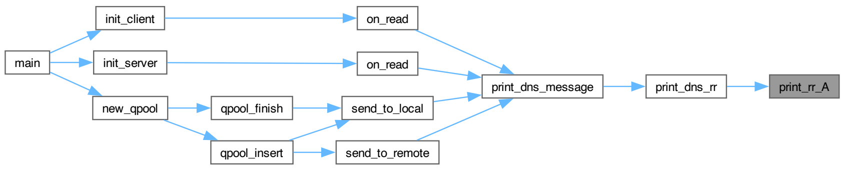
static void print_rr_A (
    const uint8_t * rdata) [static]

Print the rdata field of an A type RR.

Parameters

rdata	The rdata field

Here is the caller graph for this function:



5.30.1.7 print__rr__AAAA()

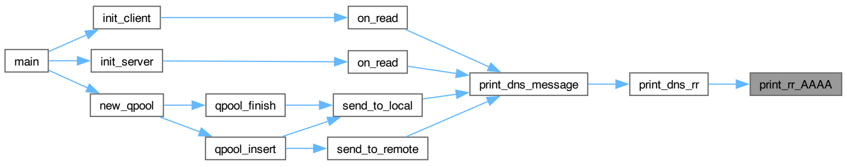
static void print__rr__AAAA (
    const uint8__t ∗ rdata) [static]

Print the rdata field of an AAAA type RR.

Parameters

rdata	The rdata field

Here is the caller graph for this function:



5.30.1.8 print__rr__CNAME()

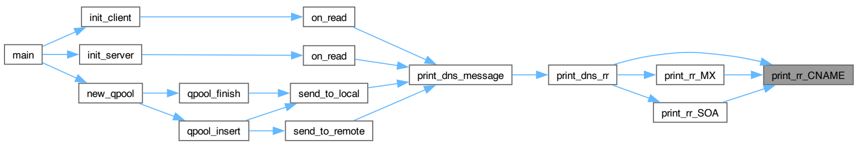
static void print__rr__CNAME (
    const uint8__t ∗ rdata) [static]

Print the rdata field of a CNAME type RR.

Parameters

rdata	The rdata field

Here is the caller graph for this function:



5.30.1.9 print_rr_MX()

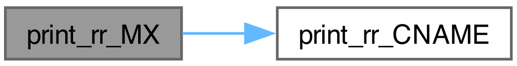
static void print_rr_MX (
const uint8_t * rdata) [static]

Print the rdata field of an MX type RR.

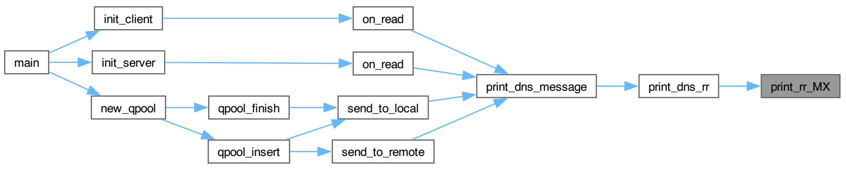
Parameters

rdata	The rdata field

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.10 print_rr_SOA()

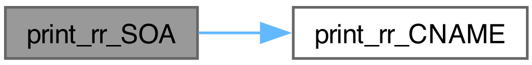
```
static void print_rr_SOA (
                    uint16_t rdlength,
                    const uint8_t * rdata) [static]
```

Print the rdata field of an SOA type RR.

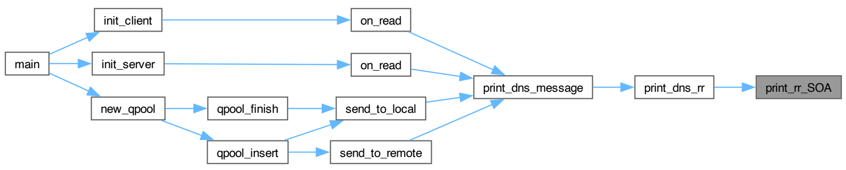
Parameters

rdlength	The rdlength field
rdata	The rdata field

Here is the call graph for this function:



Here is the caller graph for this function:

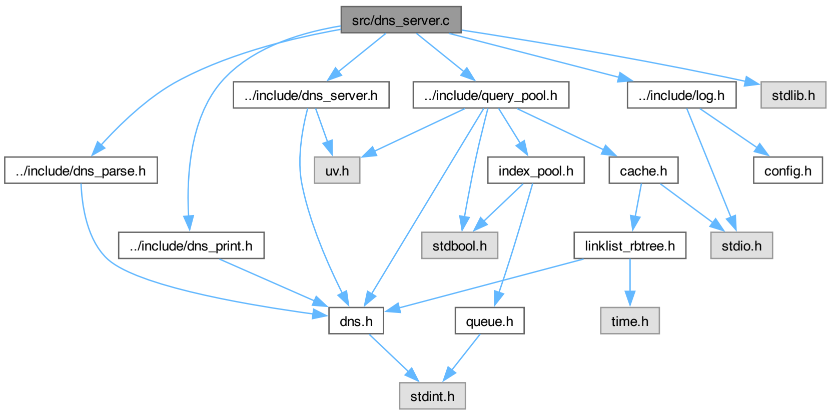


5.31 src/dns_server.c File Reference

#include "../include/dns_server.h"
#include <stdlib.h>
#include "../include/log.h"
#include "../include/dns_parse.h"
#include "../include/dns_print.h"

#include "../include/query_pool.h"

Include dependency graph for dns_server.c:



Functions

- static void alloc_buffer (uv_handle_t ∗handle, size_t suggested_size, uv_buf_t ∗buf)

  Allocate space for the buffer.
- static void on_send (uv_udp_send_t ∗req, int status)

  Callback function for sending response messages to local clients.
- static void on_read (uv_udp_t ∗handle, ssize_t nread, const uv_buf_t ∗buf, const struct sockaddr ∗addr, unsigned flags)

  Callback function for receiving query messages from local clients.
- void init_server (uv_loop_t ∗loop)

  Initialize the DNS server.
- void send_to_local (const struct sockaddr ∗addr, const Dns_Msg ∗msg)

  Send a DNS response message to local clients.

Variables

- static uv_udp_t server_socket

  Socket for server communication with local clients.
- static struct sockaddr_in recv_addr

  Address for receiving DNS query messages.
- Query_Pool ∗ qpool

  Query pool.

5.31.1 Function Documentation

5.31.1.1 alloc_buffer()

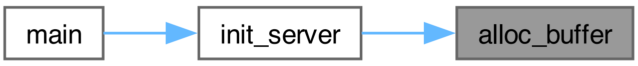
```
static void alloc_buffer (
                    uv_handle_t * handle,
                    size_t suggested_size,
                    uv_buf_t * buf)   [static]
```

Allocate space for the buffer.

Parameters

handle	Allocation handle
suggested_size	Suggested buffer size
buf	Buffer to be allocated

Allocates a buffer of fixed size DNS_STRING_MAX_SIZE for receiving DNS query messages from local clients. Here is the caller graph for this function:



5.31.1.2 init_server()

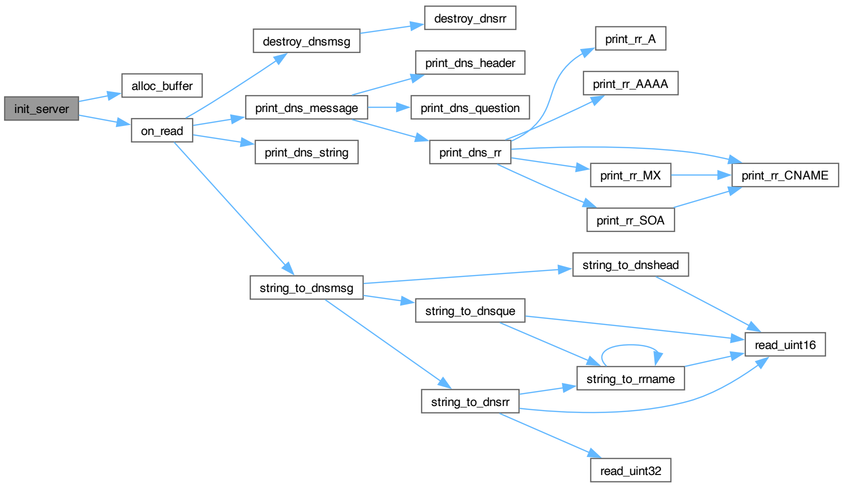
```
void init_server (
                    uv_loop_t * loop)
```

Initialize the DNS server.

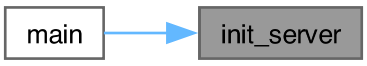
Parameters

loop	The libuv event loop

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.3 on_read()

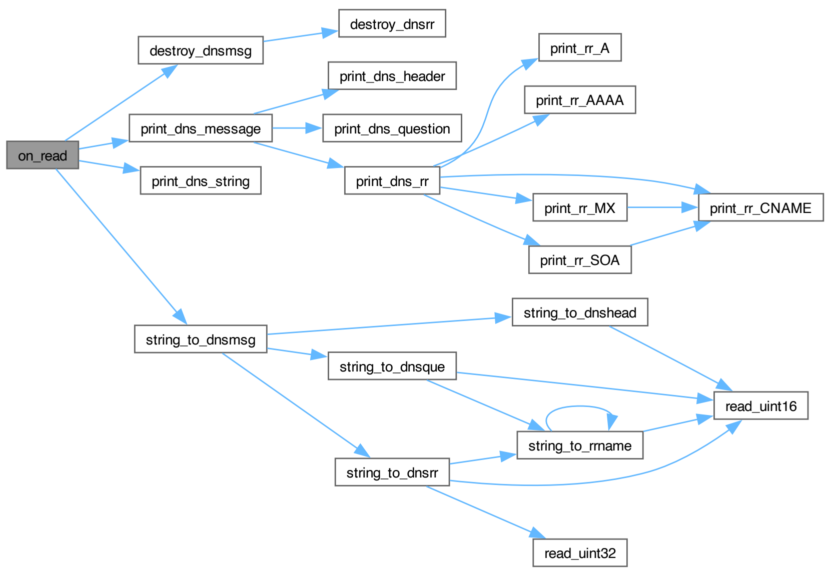
static void on_read (
uv_udp_t ∗ handle,
ssize_t nread,
const uv_buf_t ∗ buf,
const struct sockaddr ∗ addr,
unsigned flags) [static]

Callback function for receiving query messages from local clients.

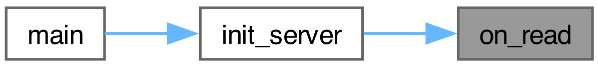
Parameters

handle	Query handle
nread	Number of bytes received
buf	Buffer containing the received message
addr	Address of the local sender
flags	Flags indicating special conditions for the received data

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.1.4 on_send()

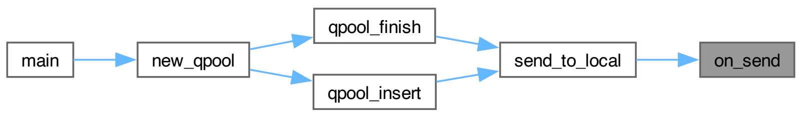
```
static void on_send (
                    uv_udp_send_t * req,
                    int status) [static]
```

Callback function for sending response messages to local clients.

Parameters

req	Send handle
status	Send status, indicating whether the send was successful

Here is the caller graph for this function:



5.31.1.5 send_to_local()

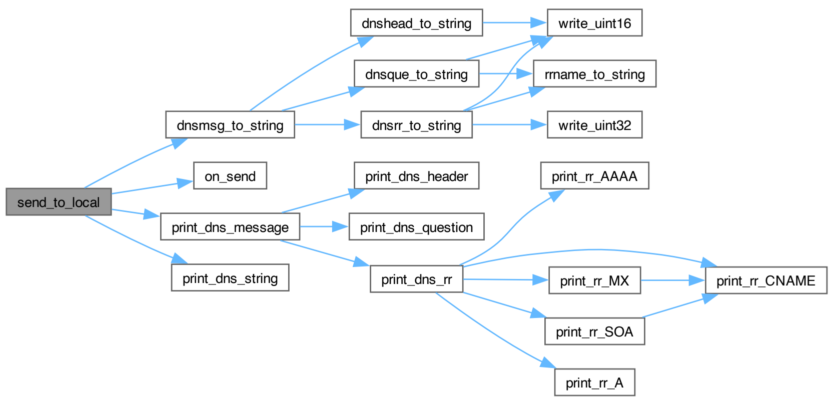
void send_to_local (
const struct sockaddr ∗ addr,
const Dns_Msg ∗ msg)

Send a DNS response message to local clients.

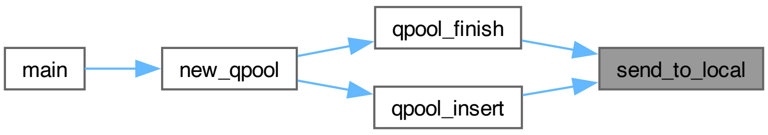
Parameters

addr	The address of the local client
msg	The DNS message to be sent

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.2 Variable Documentation

5.31.2.1 qpool

Query_Pool∗ qpool [extern]

Query pool.

5.31.2.2 recv_addr

struct sockaddr_in recv_addr [static]

Address for receiving DNS query messages.

5.31.2.3 server_socket

uv_udp_t server_socket [static]

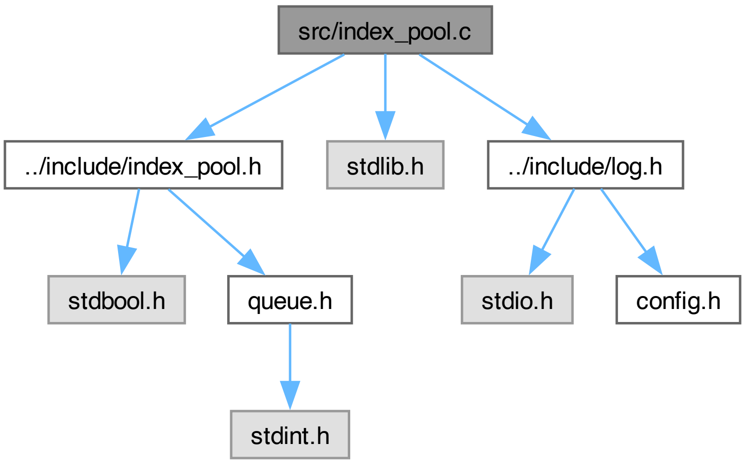
Socket for server communication with local clients.

5.32 src/index_pool.c File Reference

#include "../include/index_pool.h"
#include <stdlib.h>

#include "../include/log.h"

Include dependency graph for index_pool.c:



Functions

- static bool ipool_full (Index_Pool *ipool)

  Check if the index pool is full.
- static uint16_t ipool_insert (Index_Pool *ipool, Index *req)

  Insert an index into the pool.
- static bool ipool_query (Index_Pool *ipool, uint16_t index)

  Query if an index exists in the pool.
- static Index * ipool_delete (Index_Pool *ipool, uint16_t index)

  Delete an index from the pool.
- static void ipool_destroy (Index_Pool *ipool)

  Destroy the index pool.
- Index_Pool * new_ipool ()

  Create a new index pool.

5.32.1 Function Documentation

5.32.1.1 ipool_delete()

```
static Index * ipool_delete (
                    Index_Pool * ipool,
                    uint16_t index)    [static]
```

Delete an index from the pool.

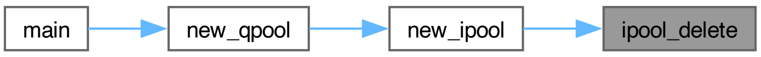
Parameters

ipool	The index pool
index	The index to delete

Returns

The deleted index

Here is the caller graph for this function:



5.32.1.2 ipool_destroy()

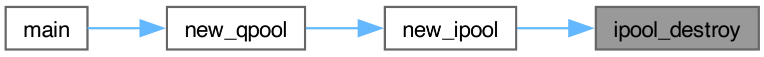
static void ipool_destroy (
Index_Pool ∗ ipool) [static]

Destroy the index pool.

Parameters

ipool	The index pool to destroy

Here is the caller graph for this function:



5.32.1.3 ipool_full()

static bool ipool_full (
Index_Pool ∗ ipool) [static]

Check if the index pool is full.

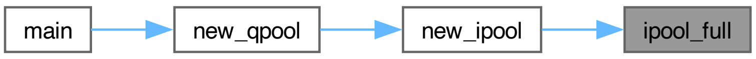
Parameters

ipool	The index pool

Returns

True if the index pool is full, false otherwise

Here is the caller graph for this function:



5.32.1.4 ipool_insert()

```
static uint16_t ipool_insert (
            Index_Pool * ipool,
            Index * req)   [static]
```

Insert an index into the pool.

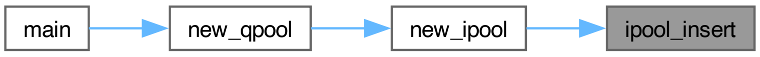
Parameters

ipool	The index pool
req	The index to insert

Returns

The ID of the inserted index

Here is the caller graph for this function:



5.32.1.5 ipool_query()

static bool ipool_query (
    Index_Pool ∗ ipool,
    uint16_t index) [static]

Query if an index exists in the pool.

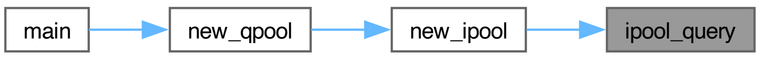
Parameters

ipool	The index pool
index	The index to query

Returns

True if the index exists, false otherwise

Here is the caller graph for this function:



5.32.1.6 new_ipool()

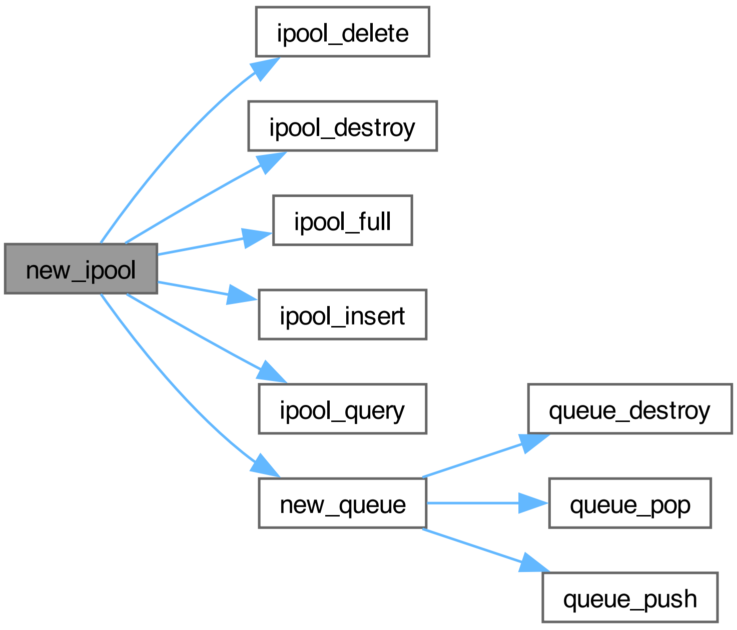
Index_Pool ∗ new_ipool ()

Create a new index pool.

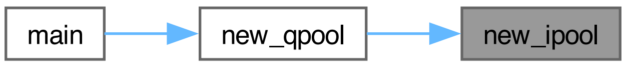
Returns

The new index pool

Here is the call graph for this function:



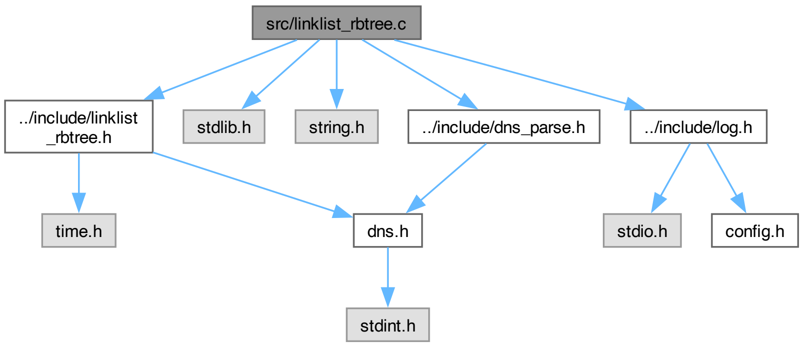
Here is the caller graph for this function:



5.33 src/linklist_rbtree.c File Reference

#include "../include/linklist_rbtree.h"
#include <stdlib.h>
#include <string.h>

#include "../include/dns_parse.h"
#include "../include/log.h"
Include dependency graph for linklist_rbtree.c:



Functions

- static void linklist_insert (Dns_RR_LinkList ∗list, Dns_RR_LinkList ∗new_list_node)

  Insert a new element into the linked list.
- static void linklist_delete_next (Dns_RR_LinkList ∗list)

  Delete the next element in the linked list.
- static Dns_RR_LinkList ∗ linklist_query_next (Dns_RR_LinkList ∗list, const uint8_↩ t ∗qname, uint16_t qtype)

  Query the next element in the linked list.
- Dns_RR_LinkList ∗ new_linklist ()

  Create a new linked list.
- static Rbtree_Node ∗ grandparent (Rbtree_Node ∗node)

  Get the grandparent of a node.
- static Rbtree_Node ∗ uncle (Rbtree_Node ∗node)

  Get the uncle of a node.
- static Rbtree_Node ∗ sibling (Rbtree_Node ∗node)

  Get the sibling of a node.
- static Rbtree_Node ∗ smallest_child (Rbtree_Node ∗node)

  Get the smallest child node in a subtree.
- static void rotate_right (Rbtree ∗tree, Rbtree_Node ∗node)

  Rotate a node to the right.
- static void rotate_left (Rbtree ∗tree, Rbtree_Node ∗node)

  Rotate a node to the left.
- static void insert_case (Rbtree ∗tree, Rbtree_Node ∗node)

  Adjust the shape of the red-black tree to keep it balanced.

- static Rbtree_Node ∗ node_init (unsigned int key, Dns_RR_LinkList ∗list, Rbtree_Node ∗fa)

  Initialize a node and allocate memory.
- void rbtree_insert (Rbtree ∗tree, unsigned int key, Dns_RR_LinkList ∗list)

  Insert a key-value pair into the red-black tree.
- static Rbtree_Node ∗ rbtree_find (Rbtree_Node ∗node, unsigned int key)

  Recursively search for a node with a given key starting from a given node.
- static void destroy_node (Rbtree_Node ∗node)

  Destroy a node in the red-black tree.
- static void delete_case (Rbtree ∗tree, Rbtree_Node ∗node)

  Adjust the shape of the red-black tree to keep it balanced.
- static void rbtree_delete (Rbtree ∗tree, Rbtree_Node ∗node)

  Delete a node from the red-black tree.
- Dns_RR_LinkList ∗ rbtree_query (Rbtree ∗tree, unsigned int key)

  Query the red-black tree for a key.
- Rbtree ∗ new_rbtree ()

  Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

Variables

- static Rbtree_Node ∗ NIL

  Leaf node.

5.33.1 Function Documentation

5.33.1.1 delete_case()

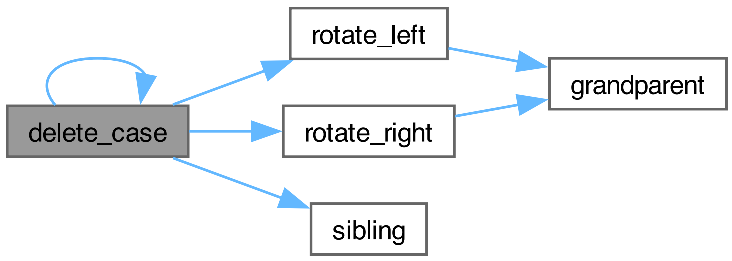
```
static void delete_case (
        Rbtree ∗ tree,
        Rbtree_Node ∗ node) [static]
```

Adjust the shape of the red-black tree to keep it balanced.

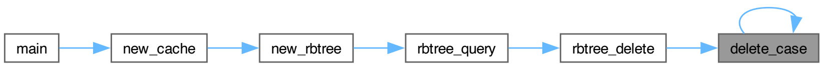
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.2 destroy_node()

static void destroy_node (
Rbtree_Node ∗ node) [static]

Destroy a node in the red-black tree.

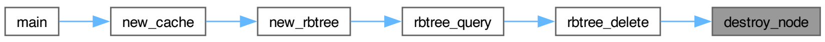
Parameters

node	The node to destroy

Note

The linked list of the node is assumed to be empty (i.e., only the head node)

Here is the caller graph for this function:



5.33.1.3 grandparent()

static Rbtree_Node ∗ grandparent (
Rbtree_Node ∗ node) [inline], [static]

Get the grandparent of a node.

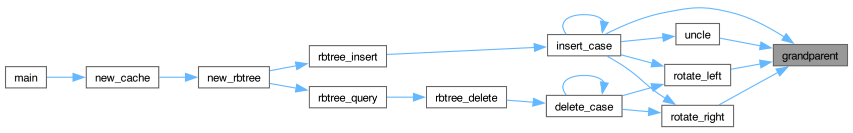
Parameters

node	The current node

Returns

The grandparent node if exists, otherwise NULL

Here is the caller graph for this function:



5.33.1.4 insert_case()

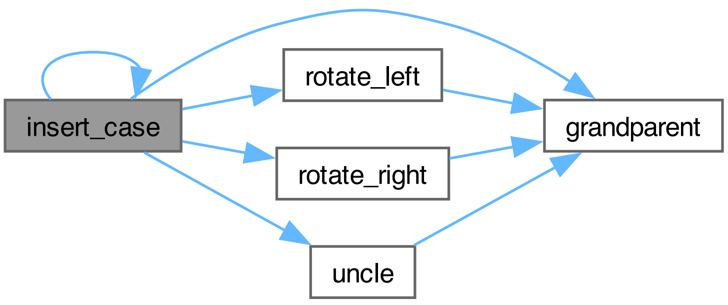
static void insert_case (
Rbtree ∗ tree,
Rbtree_Node ∗ node) [static]

Adjust the shape of the red-black tree to keep it balanced.

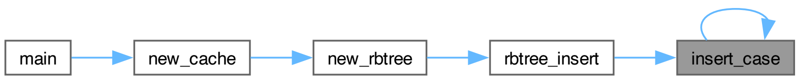
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.5 linklist_delete_next()

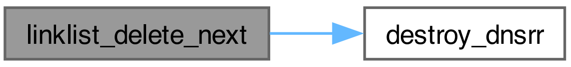
static void linklist_delete_next (
    Dns_RR_LinkList ∗ list) [static]

Delete the next element in the linked list.

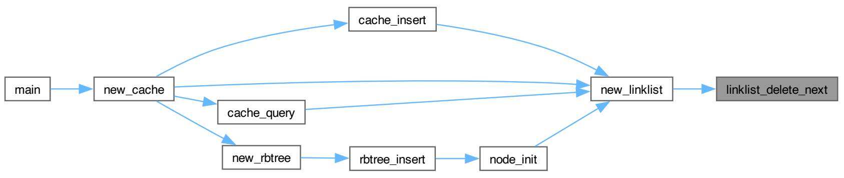
Parameters

list	The linked list

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.6 linklist_insert()

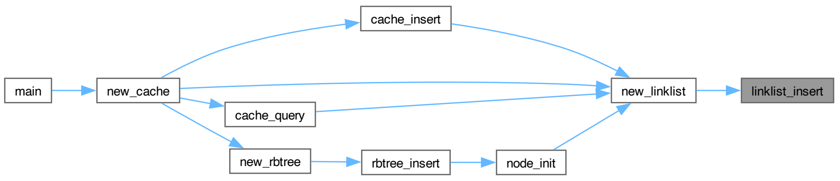
static void linklist_insert (
Dns_RR_LinkList ∗ list,
Dns_RR_LinkList ∗ new_list_node) [static]

Insert a new element into the linked list.

Parameters

list	The linked list
new_list_node	The new element to insert

Here is the caller graph for this function:



5.33.1.7 linklist_query_next()

static Dns_RR_LinkList ∗ linklist_query_next (
Dns_RR_LinkList ∗ list,
const uint8_t ∗ qname,
uint16_t qtype) [static]

Query the next element in the linked list.

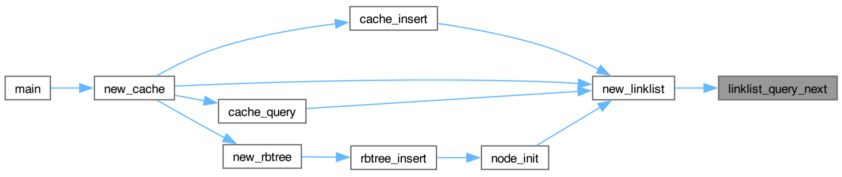
Parameters

list	The linked list
qname	The query name
qtype	The query type

Returns

The queried element if found, otherwise NULL

Here is the caller graph for this function:



5.33.1.8 new_linklist()

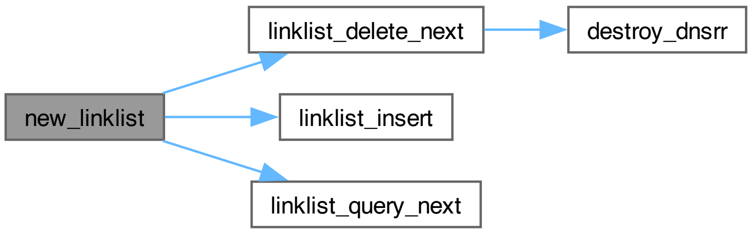
Dns_RR_LinkList * new_linklist ()

Create a new linked list.

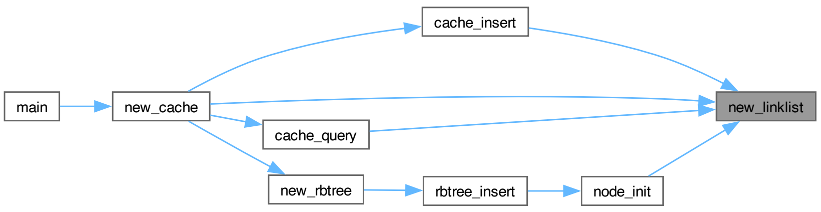
Returns

The new linked list

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.9 new_rbtree()

Rbtree ∗ new_rbtree ()

Initialize a new red-black tree This function allocates memory for a new red-black tree and its nil node, and sets up the tree's function pointers for insertion and querying.

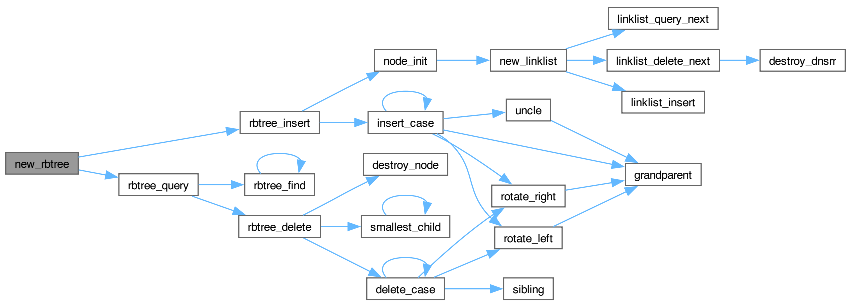
Returns

A pointer to the newly created red-black tree

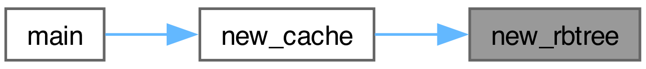
Note

If memory allocation fails, the function will log a fatal error and terminate the program.

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.10 node_init()

static Rbtree_Node ∗ node_init (
    unsigned int key,
    Dns_RR_LinkList ∗ list,
    Rbtree_Node ∗ fa) [static]

Initialize a node and allocate memory.

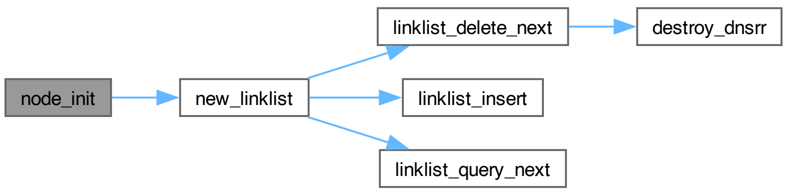
Parameters

key	The key of the node
list	The value of the node
fa	The parent node

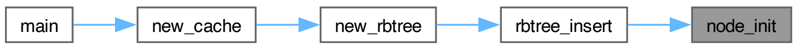
Returns

A pointer to the new node

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.11 rbtree_delete()

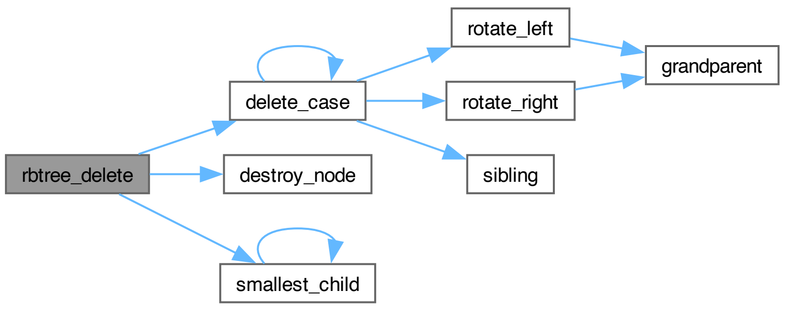
static void rbtree_delete (
    Rbtree ∗ tree,
    Rbtree_Node ∗ node) [static]

Delete a node from the red-black tree.

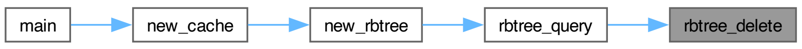
Parameters

tree	The tree that the node belongs to
node	The node to delete

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.12 rbtree_find()

static Rbtree_Node ∗ rbtree_find (
    Rbtree_Node ∗ node,
    unsigned int key) [static]

Recursively search for a node with a given key starting from a given node.

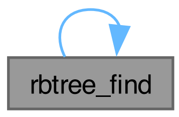
Parameters

node	The current node
key	The key to search for

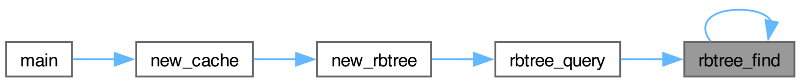
Returns

A pointer to the node if found, otherwise NULL

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.13 rbtree_insert()

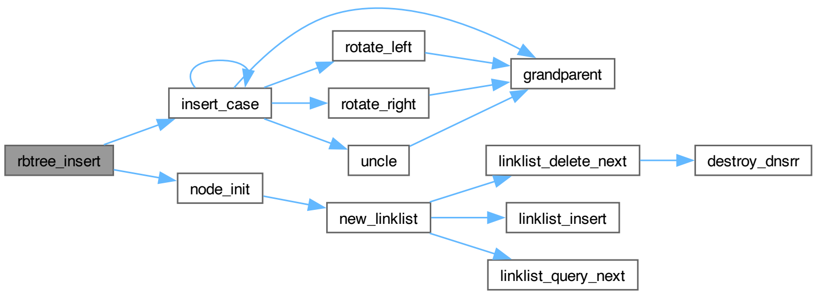
void rbtree_insert (
    Rbtree * tree,
    unsigned int key,
    Dns_RR_LinkList * list)

Insert a key-value pair into the red-black tree.

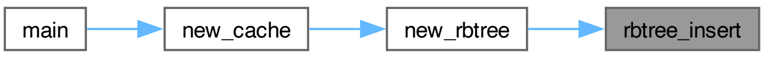
Parameters

tree	The red-black tree
key	The key
list	The value

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.14 rbtree_query()

```
Dns_RR_LinkList * rbtree_query (
        Rbtree * tree,
        unsigned int key)
```

Query the red-black tree for a key.

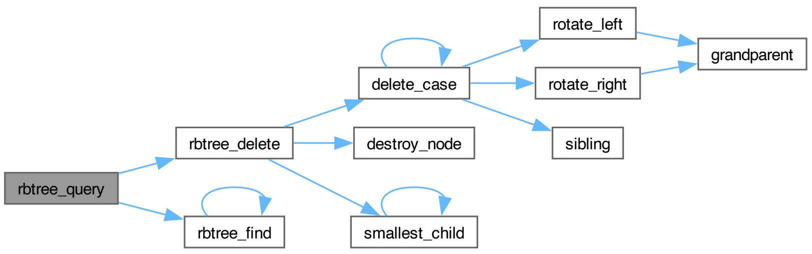
Parameters

tree	The red-black tree
key	The key to query

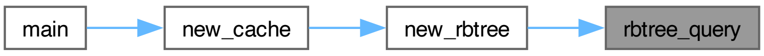
Returns

The linked list of the value if found, otherwise NULL

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.15 rotate_left()

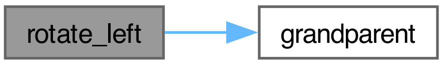
static void rotate_left (
    Rbtree ∗ tree,
    Rbtree_Node ∗ node) [static]

Rotate a node to the left.

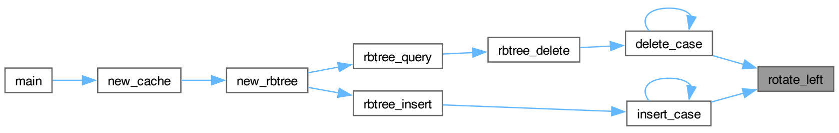
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.16 rotate_right()

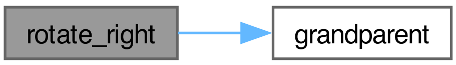
static void rotate_right (
Rbtree ∗ tree,
Rbtree_Node ∗ node) [static]

Rotate a node to the right.

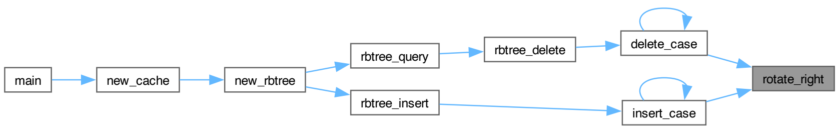
Parameters

tree	The tree that the node belongs to
node	The current node

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.17 sibling()

static Rbtree_Node ∗ sibling (
Rbtree_Node ∗ node) [inline], [static]

Get the sibling of a node.

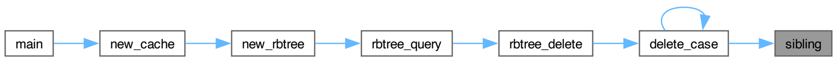
Parameters

node	The current node

Returns

The sibling node if exists, otherwise NULL

Here is the caller graph for this function:



5.33.1.18 smallest_child()

static Rbtree_Node ∗ smallest_child (
Rbtree_Node ∗ node) [static]

Get the smallest child node in a subtree.

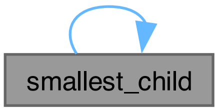
Parameters

node	The root of the subtree

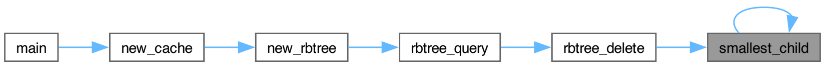
Returns

The smallest child node

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.1.19 uncle()

static Rbtree_Node ∗ uncle (
Rbtree_Node ∗ node) [inline], [static]

Get the uncle of a node.

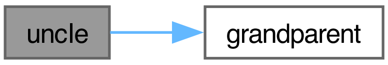
Parameters

node	The current node

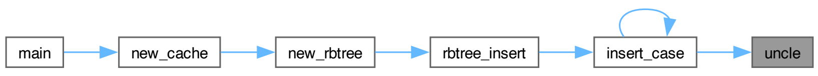
Returns

The uncle node if exists, otherwise NULL

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2 Variable Documentation

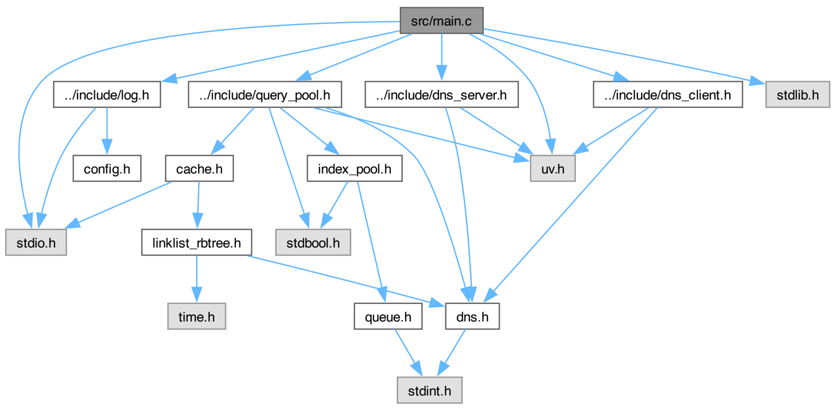
5.33.2.1 NIL

Rbtree_Node∗ NIL [static]

Leaf node.

5.34 src/main.c File Reference

#include <stdio.h>
#include <stdlib.h>
#include <uv.h>
#include "../include/log.h"
#include "../include/dns_client.h"
#include "../include/dns_server.h"
#include "../include/query_pool.h"
Include dependency graph for main.c:



Functions

- int main (int argc, char ∗argv[ ])

Variables

- uv_loop_t ∗ loop
- Cache ∗ cache
- Query_Pool ∗ qpool

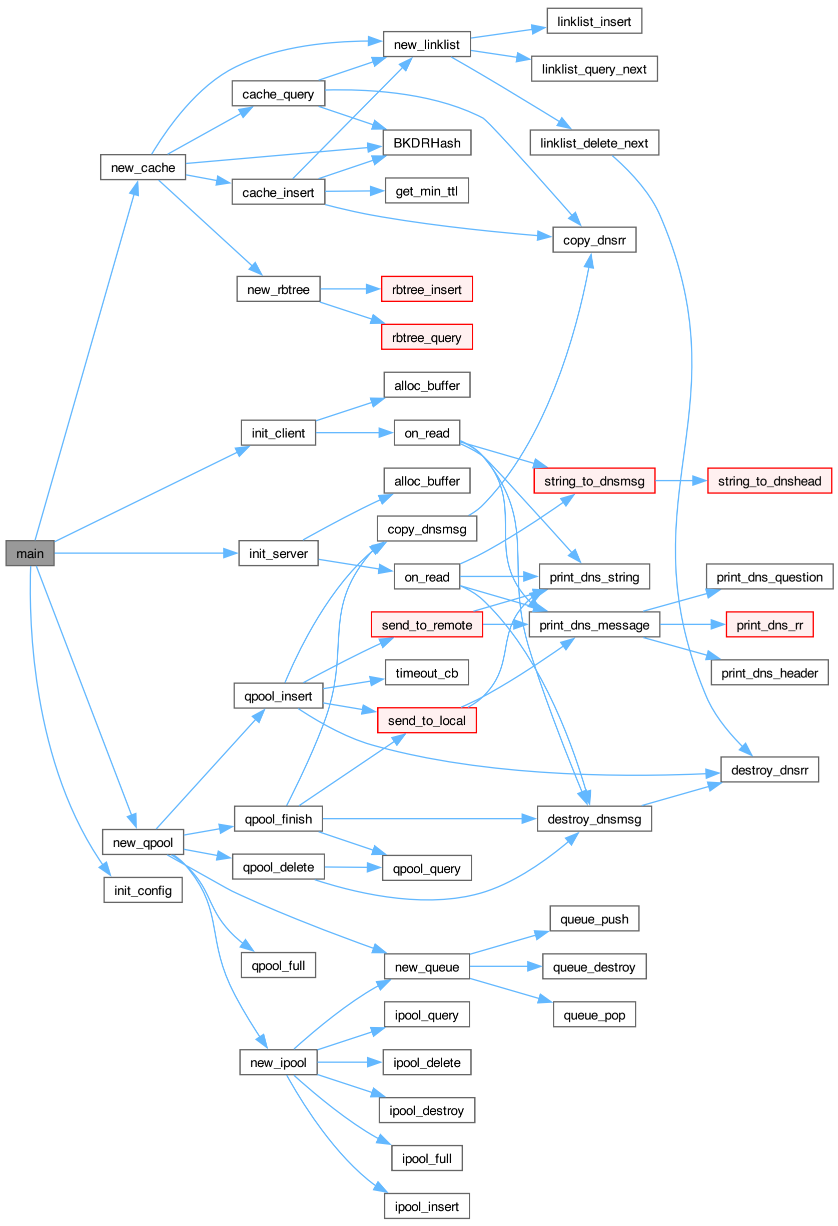
    Query pool.
- FILE ∗ log_file

5.34.1 Function Documentation

5.34.1.1 main()

```
int main (
            int argc,
            char ∗ argv[ ])
```

Here is the call graph for this function:



5.34.2 Variable Documentation

5.34.2.1 cache

Cache∗ cache

5.34.2.2 log_file

FILE∗ log_file

5.34.2.3 loop

uv_loop_t∗ loop

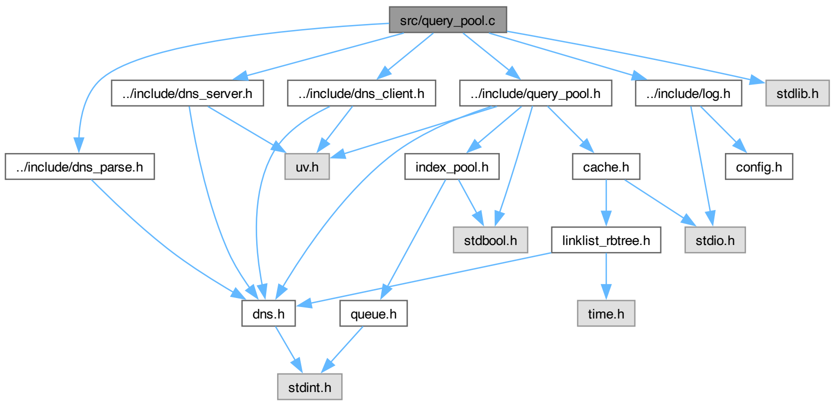
5.34.2.4 qpool

Query_Pool∗ qpool

Query pool.

5.35 src/query_pool.c File Reference

#include "../include/query_pool.h"
#include <stdlib.h>
#include "../include/log.h"
#include "../include/dns_parse.h"
#include "../include/dns_client.h"
#include "../include/dns_server.h"
Include dependency graph for query_pool.c:



Functions

- static void timeout_cb (uv_timer_t ∗timer)

  Timeout callback function This function is called when a query times out. It stops the timer and deletes the query from the query pool.
- static bool qpool_full (Query_Pool ∗this)

  Check if the query pool is full.

- static void qpool_insert (Query_Pool ∗qpool, const struct sockaddr ∗addr, const Dns_Msg ∗msg)

  Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.
- static bool qpool_query (Query_Pool ∗qpool, uint16_t id)

  Check if a query exists in the query pool.
- static void qpool_finish (Query_Pool ∗qpool, const Dns_Msg ∗msg)

  Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.
- static void qpool_delete (Query_Pool ∗qpool, uint16_t id)

  Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.
- Query_Pool ∗ new_qpool (uv_loop_t ∗loop, Cache ∗cache)

  Create a new query pool This function initializes a new query pool and returns a pointer to it.

5.35.1 Function Documentation

5.35.1.1 new_qpool()

```
Query_Pool ∗ new_qpool (
            uv_loop_t ∗ loop,
            Cache ∗ cache)
```

Create a new query pool This function initializes a new query pool and returns a pointer to it.

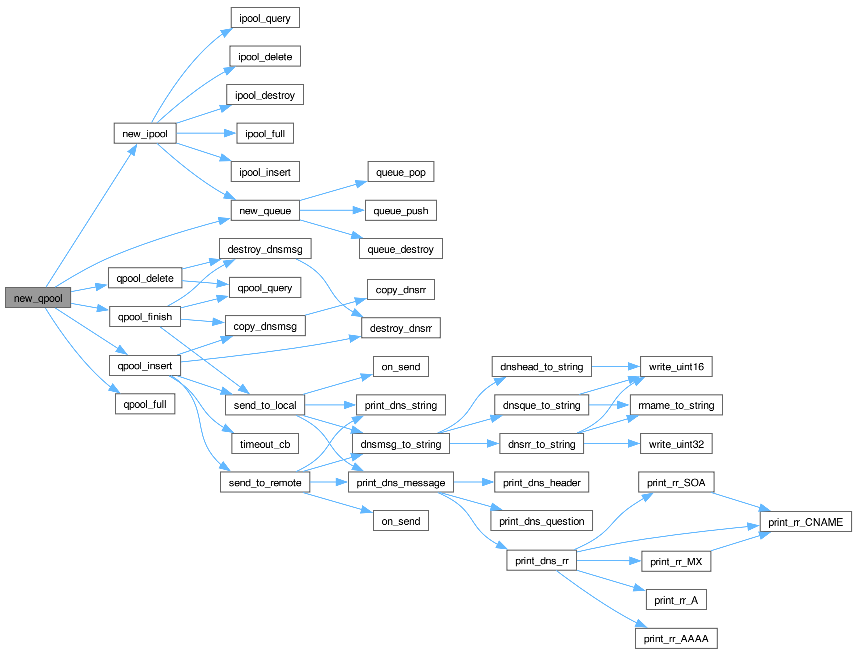
Parameters

loop	The libuv event loop
cache	The cache used for storing DNS responses

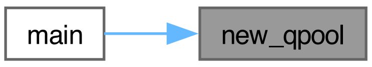
Returns

A pointer to the newly created query pool

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.2 qpool_delete()

static void qpool_delete (

Query_Pool * qpool,

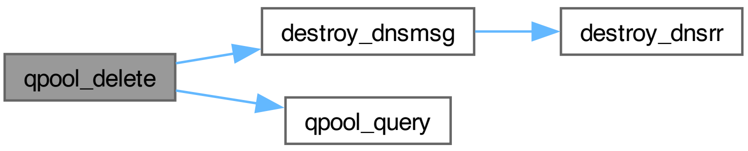
uint16_t id) [static]

Delete a query from the query pool This function deletes a query from the query pool and frees the associated resources.

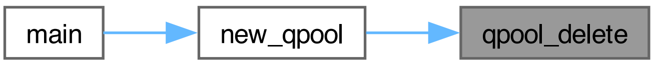
Parameters

qpool	The query pool
id	The ID of the query to be deleted

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.3 qpool_finish()

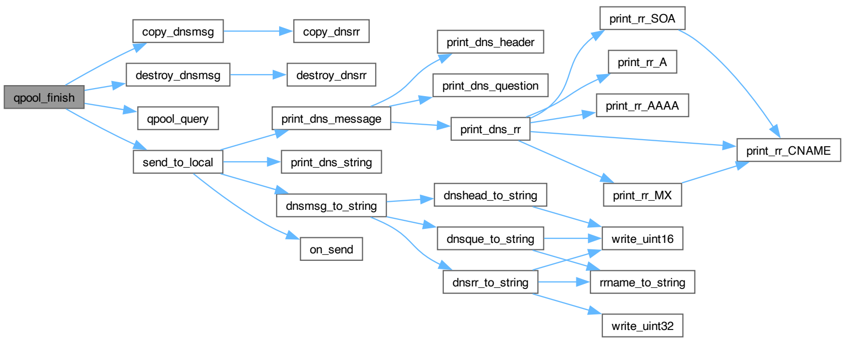
static void qpool_finish (
    Query_Pool ∗ qpool,
    const Dns_Msg ∗ msg) [static]

Finish processing a query This function is called when a response is received for a query. It processes the response, updates the cache if necessary, and sends the response to the local client.

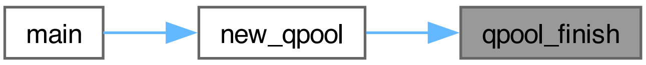
Parameters

qpool	The query pool
msg	The DNS message containing the response

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.4 qpool_full()

static bool qpool_full (
Query_Pool * this) [static]

Check if the query pool is full.

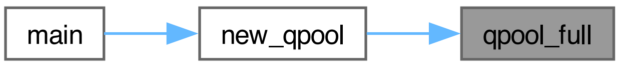
Parameters

this	The query pool

Returns

true if the query pool is full, false otherwise

Here is the caller graph for this function:



5.35.1.5 qpool_insert()

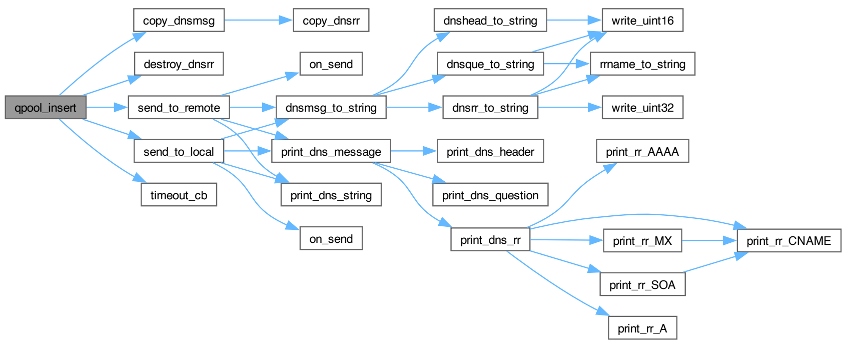
static void qpool_insert (
    Query_Pool ∗ qpool,
    const struct sockaddr ∗ addr,
    const Dns_Msg ∗ msg) [static]

Insert a new query into the query pool This function creates a new query and inserts it into the query pool. If the query is found in the cache, it is immediately processed and sent to the local client. Otherwise, it is sent to the remote DNS server and a timeout timer is started.

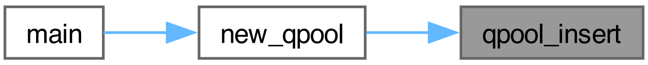
Parameters

qpool	The query pool
addr	The address of the client
msg	The DNS message containing the query

Here is the call graph for this function:



Here is the caller graph for this function:



5.35.1.6 qpool_query()

static bool qpool_query (
    Query_Pool * qpool,
    uint16_t id) [static]

Check if a query exists in the query pool.

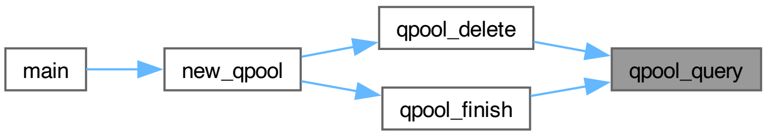
Parameters

qpool	The query pool
id	The ID of the query

Returns

true if the query exists in the query pool, false otherwise

Here is the caller graph for this function:



5.35.1.7 timeout_cb()

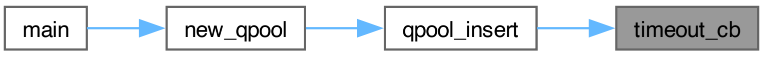
static void timeout_cb (
    uv_timer_t * timer) [static]

Timeout callback function This function is called when a query times out. It stops the timer and deletes the query from the query pool.

Parameters

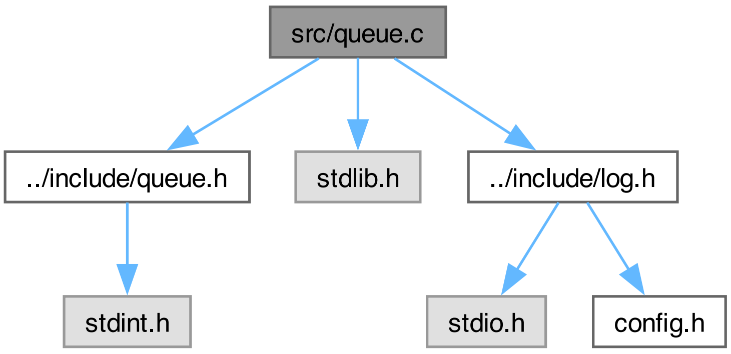
timer	The timer that timed out

Here is the caller graph for this function:



5.36 src/queue.c File Reference

#include "../include/queue.h"
#include <stdlib.h>
#include "../include/log.h"
Include dependency graph for queue.c:



Functions

- static void queue_push (Queue *queue, uint16_t num)

  Push a number onto the queue.
- static uint16_t queue_pop (Queue *queue)

  Pop a number from the queue.
- static void queue_destroy (Queue *queue)

Destroy the queue.

- Queue ∗ new_queue ()

  Create a new queue.

5.36.1 Function Documentation

5.36.1.1 new_queue()

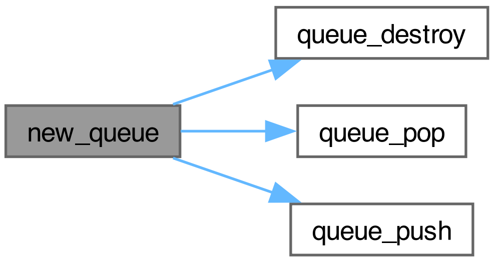
Queue ∗ new_queue ()

Create a new queue.

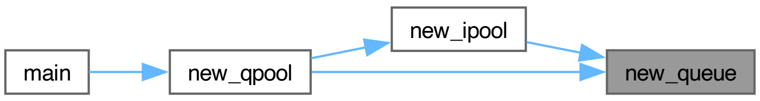
Returns

The new queue

Here is the call graph for this function:



Here is the caller graph for this function:



5.36.1.2 queue_destroy()

static void queue_destroy (

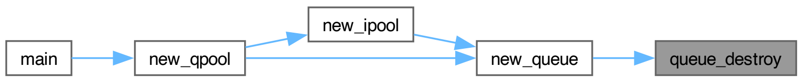
Queue ∗ queue) [static]

Destroy the queue.

Parameters

queue	The queue to destroy

Here is the caller graph for this function:



5.36.1.3 queue_pop()

static uint16_t queue_pop (
Queue ∗ queue) [static]

Pop a number from the queue.

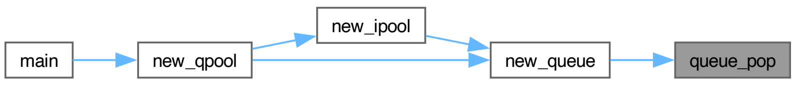
Parameters

queue	The queue

Returns

The number popped from the queue

Here is the caller graph for this function:



5.36.1.4 queue_push()

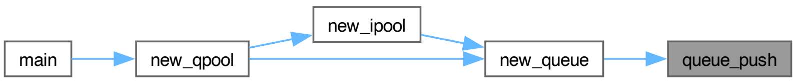
static void queue_push (
Queue ∗ queue,
uint16_t num) [static]

Push a number onto the queue.

Parameters

queue	The queue
num	The number to push

Here is the caller graph for this function:



索引