mtd64-ng

A lightweight multithreaded C++11 DNS64 server.

Developer Documentation

Table of Contents

Introducion

Design goals

The main design goals for the mtd64-ng server are speed, simplicity, good code quality and expandability. Using OO decomposition and the RAII idiom (Resource Acquisition Is Initialization) ensures that raw, sensitive resources (memory, sockets) are always released, and greatly simplifies exception handling.

C++11

Using the new features of the C++11 standard enables us to write more efficient and better readable code. Move semantics helps to avoid unnecessary copies, lambda expressions and initializer lists helps achieving better code readability. The built-in thread, mutex, condition_variable and atomic classes and templates provide a standardized and cross-platform interface for multithreaded programming.

Architecture

Classes

ThreadPool, WorkerThread

ThreadPool is a generic thread pool class written in C++11 using the built-in thread classes. Its constructor starts the given number of worker threads (using the WorkerThread functors to provide the main loop for the threads). An std::deque queues the tasks and a condition_variable is used to signal avaliable tasks to sleeping worker threads.

Tasks are *moved* into the queue using an std::function template. This makes it possible to add functions, functors and lambda expressions to the queue.

DNSHeader

DNSHeader represents the header of a DNS packet. It is not constructed, but casted on the raw data stream, making it more efficient.

Polymorphic setter and getter functions perform the necessary bit masking and network byte order vs. host byte order conversions.

DNSLabel, DNSQname, DNSQuestion, DNSResource, DNSPacket

These classes represent parts of the DNS packet. DNSQname aggregates DNSLabels and DNSQuestions and DNSResources contain a DNSQname (beside the other standard fields). DNSPacket aggregates a DNSHeader, DNSQuestions and DNSResources. Setter and getter functions perform all the necessary conversions, including resizing the packet (and moving all the Questions, Resources and Labels) when the rdata field of a DNSQname or DNSResource is changed – which is an essential operation in synthesizing the IPv6 address form the IPv4 address.

DNSSource, DNSClient

The DNSSource interface provides an interface to 'send' DNS query packets and get an answer. Implementing classes can use their own strategies to do that (forwarding or recursing, caching or not).

The DNSClient implements the DNSSource interface and acts as a non-caching forwarder to resolve DNS queries.

Query

The Query class implements a DNS query. It is a functor which can be supplied to a ThreadPool as a task. It stores a pointer to the raw packet data and performs the business logic using the previous helper classes.

Server

The Server class implements the DNS64 server. It loads and stores the configuration, starts the thread pool, opens the sockets, then receives and converts DNS query packets into Query objects and stores them in a queue to be executed by the thread pool.

Flow of a query

In this section we describe the generic flow of a query through the mtd-64 server:

1. The Server receives a UDP packet on the configured port from a Client.
2. The Server creates a Query object from the packet and places it in the queue.
3. When there is an avaliable worker thread the Query starts to execute.
4. The Query objects determines whether the packet really is a DNS Query.
5. If so it forwards the query to one of the configured DNS servers using the configured selection mode. If it fails (timeout occurs at timeout-time) it tries an another DNS server, at most resend-attempts time.

6. The Query objects determines whether a synthesis action is needed. Synthesis is required if: the question in the query is for an AAAA record AND the domain exists AND there is no AAAA records in the answer section.
7. If a synthesis is not needed the answer is sent to the Client.
8. If synthesis is needed the Query class synthesizes the answer using the DNS packet manipulator classes and Server, then sends the synthesized answer to the Client.

Configuration

nameserver

Function: defines a recursor to use for name resolution. Using the default option the server uses the nameservers configured in */etc/resolv.conf*

Valid values: <ip> or “default”

Default value: none

dns64-prefix

Function: Specifies the DNS64 prefix to use.

Valid values: As described in RFC 6052. <ipv6>/{32,40,48,56,64,96}

Default value: none

debugging

Function: Turns on or off debugging mode. In debugging mode a more verbose log output is generated

Valid values: yes or no

Default value: no

timeout-time

Function: Specifies the timeout for DNS recursor queries.

Valid values: a double-precision floating-point number, greater than 0s and lower than 32 768 s

Default value: 1.0 s

resend-attempts

Function: How many times will the DNS server tries to resend a DNS query message if there is no answer.

Valid values: 0 – SHORT_INT_MAX (0 means no retries)

Default value: 2

response-maxlength

Function: Maximum length of the IPv6 response message (UDP payload).

Valid values: Any valid packet size, but 512 is **strongly** recommended, since it is the RFC standard.

Default value: 512

num-threads

Function: Specifies the number of threads to start in the thread pool.

Valid values: 1 – SHORT_INT_MAX

Default value: 10

port

Function: Specifies the port for the server to listen on.

Valid values: Any valid UDP port.

Default value: 53