DNS Performance Testing Toolkit

Project Overview:

The DNS Performance Testing Toolkit is a comprehensive, open-source software designed to measure and analyze DNS server performance across various network protocols and configurations. It provides software development engineers, network administrators, and performance researchers with advanced tools to evaluate DNS infrastructure efficiency, reliability, and scalability.

Date of creation: 2025-03-23

Author: Automatically generated by QuantalQ

Project Version: d7c4d37

Link to Project: https://github.com/DNS-OARC/dnsperf

Contents

DNS Performance Testing Toolkit

Introduction

Overview

The DNS Performance Testing Toolkit is a comprehensive, open-source software designed to measure and analyze DNS server performance across various network protocols and configurations. It provides software development engineers, network administrators, and performance researchers with advanced tools to evaluate DNS infrastructure efficiency, reliability, and scalability.

Content of the Software

The toolkit consists of several specialized utilities:

- **dnsperf**: Measures DNS server query performance by sending large volumes of queries
- **resperf**: Evaluates DNS server resolution performance under increasing load
- **dnsperf-ecs-gen.py**: Generates EDNS Client Subnet options for geolocation testing
- **queryparse**: Analyzes DNS queries from packet captures for pattern recognition

These tools support multiple transport protocols (UDP, TCP, DoT, DoH) and provide detailed performance metrics to help identify bottlenecks, optimize configurations, and ensure DNS infrastructure can handle expected workloads.

Purpose of the Software

The DNS Performance Testing Toolkit addresses the critical need for robust DNS performance evaluation tools in modern network environments. Its primary purposes include:

1. Measuring DNS server performance metrics such as queries per second (QPS) and response latency
2. Evaluating DNS server behavior under varying load conditions
3. Testing DNS server performance across different transport protocols
4. Analyzing DNS query patterns from real-world traffic
5. Simulating geographically distributed client queries using EDNS Client Subnet options

The toolkit helps network administrators and DNS operators ensure their DNS infrastructure can handle expected traffic loads, identify performance bottlenecks, and validate configuration changes before deployment.

Scope

The DNS Performance Testing Toolkit focuses on:

- Performance testing of DNS servers using various protocols and configurations
- Analysis of DNS query patterns and server responses
- Generation of detailed performance metrics and statistics
- Support for modern DNS protocols including DNS over TLS (DoT) and DNS over HTTPS (DoH)

The toolkit does not include:

- DNS server implementation
- DNS zone management tools
- DNS record editing capabilities
- DNS security auditing features (beyond basic TSIG authentication testing)

Target Audience

The primary audience for this documentation includes:

- **Software Development Engineers**: Developers working on DNS server implementations or DNS-related applications
- **Network Administrators**: IT professionals responsible for maintaining DNS infrastructure
- **DNS Performance Researchers**: Individuals studying DNS performance characteristics

- **System Architects**: Professionals designing DNS-intensive infrastructures
- **Product Owners/Managers**: Stakeholders seeking to understand the technical aspects of DNS performance testing

Glossary

Term/Acronym	Full Form	Description
ALPN	Application-Layer Protocol Negotiation	A TLS extension that allows application layer protocol negotiation
DNS	Domain Name System	A hierarchical and decentralized naming system for computers, services, or resources connected to the Internet
DNSSEC	Domain Name System Security Extensions	A set of extensions to DNS that provide authentication and integrity
DoH	DNS over HTTPS	A method of performing remote DNS resolution via the HTTPS protocol
DoT	DNS over TLS	A security protocol for encrypting DNS communications
ECS	EDNS Client Subnet	An extension allowing specification of the network subnet of the client making a DNS query
EDNS	Extension Mechanisms for DNS	A protocol extension to carry additional information in DNS packets
HMAC	Hash-based Message Authentication Code	A specific type of message authentication code involving a cryptographic hash function and a secret key
HTTP/2	Hypertext Transfer Protocol version 2	A major revision of the HTTP network protocol used in DoH
LDNS	Library DNS	A DNS library used for DNS-related operations
NXDOMAIN	Non-Existent Domain	A DNS response indicating the queried domain does not exist
OPT	Option	A DNS resource record type used for extending DNS capabilities
pcap	Packet Capture	A file format for storing network traffic data
POSIX	Portable Operating System Interface	A family of standards for maintaining compatibility between operating systems
pthread	POSIX Threads	A threading library for creating and managing threads
QPS	Queries Per Second	A measure of the number of DNS queries a server can handle in one second
RD	Recursion Desired	A DNS query flag indicating the client wants recursive resolution
RDATA	Resource Data	The data payload of a DNS resource record
RR	Resource Record	The basic data unit in the Domain Name System

Term/Acronym	Full Form	Description
TCP	Transmission Control Protocol	A network communication protocol that ensures reliable, ordered, and error-checked delivery of data
TSIG	Transaction SIGnature	A method of authenticating DNS messages
UDP	User Datagram Protocol	A connectionless transport layer protocol
URI	Uniform Resource Identifier	A string of characters that unambiguously identifies a particular resource
Wire Format	Network Protocol Data Representation	The binary representation of data as it is transmitted over a network

For a complete glossary with additional terms, see the Consolidated Glossary.

Content of the overall Documentation

Getting Started

Installation Guide

Prerequisites Before installing the DNS Performance Testing Toolkit, ensure your system meets the following requirements:

- C compiler (GCC or Clang)
- Make and autotools
- OpenSSL development libraries (for TLS support)
- nghttp2 development libraries (for HTTP/2 support)
- POSIX-compliant operating system

- Python 3.x (for contrib utilities)

For the contrib utilities, additional Python dependencies are required:

- dnspython
- pcapy (for queryparse)
- libpcap development libraries (for pcapy)

Installation Steps

1. **Clone or download the repository**:

   ```
   git clone https://github.com/DNS-OARC/dnsperf.git
   cd dnsperf
   ```

2. **Generate the configure script**:

   ```
   ./autogen.sh
   ```

3. **Configure the build**:

   ```
   ./configure
   ```

4. **Build the software**:

   ```
   make
   ```

5. **Install the software**:

   ```
   make install
   ```

Package-based Installation For Debian/Ubuntu:

```
apt-get install dnsperf
```

For Red Hat/CentOS/Fedora:

```
yum install dnsperf
```

For macOS (using Homebrew):

```
brew install dnsperf
```

System Requirements

Minimum Requirements

- CPU: 1 GHz dual-core processor
- Memory: 512 MB RAM
- Disk Space: 50 MB
- Network: Basic network connectivity to the DNS server being tested
- Operating System: Linux, FreeBSD, macOS, or other POSIX-compliant OS

Recommended Requirements

- CPU: 2 GHz quad-core processor or better
- Memory: 2 GB RAM or more
- Disk Space: 100 MB
- Network: High-bandwidth, low-latency connection to the DNS server being tested
- Operating System: Recent Linux distribution (Ubuntu 18.04+, CentOS 7+, etc.)

For high-volume testing (>10,000 QPS), additional resources may be required:

- Multi-core CPU (8+ cores recommended)
- 4+ GB RAM
- Network interface capable of handling the desired traffic volume

Quick Start / First Run

Basic dnsperf Test

1. **Create a simple query file** (queries.txt):

   ```
   example.com A
   www.example.com A
   mail.example.com MX
   ```

2. **Run a basic test against a local DNS server**:

   ```
   dnsperf -s 127.0.0.1 -d queries.txt
   ```

3. **View the results**:

   ```
   Statistics:

     Queries sent:         3
     Queries completed:    3 (100.00%)
     Queries lost:         0 (0.00%)

     Response codes:       NOERROR 3 (100.00%)
     Average packet size:  request 31, response 75
     Run time (s):         0.002
     Queries per second:   1500.00

     Average Latency (s):  0.000726 (min 0.000123, max 0.001223)
     Latency StdDev (s):   0.000458
   ```

Basic resperf Test

1. **Run a basic resolution performance test**:

   ```
   resperf -s 127.0.0.1 -d queries.txt
   ```

2. **View the results and generated plot data**:

   ```
   Statistics:

     Queries sent:         3
     Queries completed:    3
     Queries lost:         0
     Response codes:       NOERROR 3 (100.00%)

     Run time (s):         60.001
     Maximum throughput:   100.00 qps
     Lost at that point:   0.00%
   ```

Using Different Transport Protocols Test with TCP:

```
dnsperf -s 127.0.0.1 -d queries.txt -M tcp
```

Test with DNS over TLS:

```
dnsperf -s 127.0.0.1 -p 853 -d queries.txt -M dot
```

Test with DNS over HTTPS:

```
dnsperf -s 127.0.0.1 -p 443 -d queries.txt -M doh --doh-uri /dns-query
```

Prerequisites

The DNS Performance Testing Toolkit has several dependencies that must be installed before building or using the software:

Required Dependencies

- **C Compiler and Build Tools**:
  - GCC or Clang
  - Make
  - Autoconf
  - Automake
  - Libtool
- **Library Dependencies**:
  - OpenSSL (for TLS support in DoT and DoH)
  - nghttp2 (for HTTP/2 support in DoH)
  - POSIX Threads (pthread)

Optional Dependencies

- **For Code Coverage**:
  - gcov
  - lcov
- **For Contrib Utilities**:
  - Python 3.x
  - dnspython
  - pcapy
  - libpcap

Environment Setup Ensure your environment has the necessary development tools:

For Debian/Ubuntu:

```
apt-get install build-essential autoconf automake libtool libssl-dev libnghttp2-dev python3 python3-pip
pip3 install dnspython pcapy
```

For Red Hat/CentOS/Fedora:

```
yum install gcc make autoconf automake libtool openssl-devel nghttp2-devel python3 python3-pip
pip3 install dnspython pcapy
```

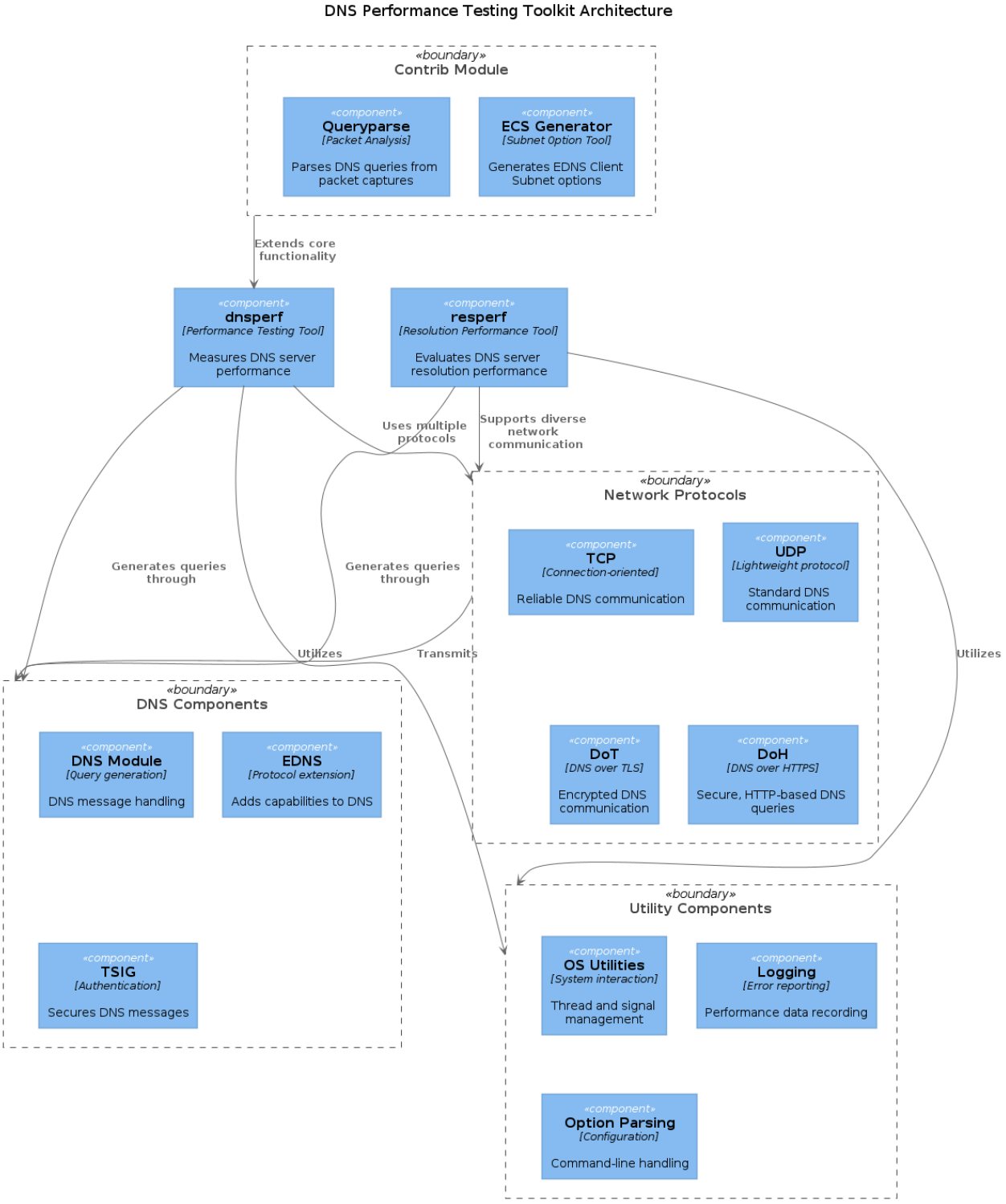
For macOS:

```
brew install autoconf automake libtool openssl nghttp2 python3
pip3 install dnspython pcapy
```

Architecture and Design Overview

System Architecture

The DNS Performance Testing Toolkit implements a sophisticated hybrid architecture that combines multiple architectural patterns to achieve high-performance DNS testing capabilities across various protocols.



The architecture is based on several key patterns:

1. **Multi-threaded Architecture**: Separate sender and receiver threads for efficient concurrent operations
2. **Event-Driven Architecture**: Non-blocking I/O multiplexing for efficient network communication
3. **State Machine Pattern**: Connection management with explicit state tracking
4. **Layered Architecture**: Clear separation of concerns across distinct layers
5. **Strategy Pattern**: Common interface for different network protocol implementations
6. **Factory Method Pattern**: Encapsulated object creation and configuration
7. **Callback Pattern**: Event handling through function pointers

For more detailed information about the architecture, refer to the High-Level Documentation.

User Guide

Features Overview

The DNS Performance Testing Toolkit provides a comprehensive set of features for DNS performance testing and analysis:

Core Features

1. **DNS Query Performance Testing (dnsperf)**
   - Send large volumes of DNS queries to measure server performance
   - Support for multiple transport protocols (UDP, TCP, DoT, DoH)
   - Detailed performance metrics (QPS, latency, packet sizes)
   - Configurable concurrency through multi-threading
   - Support for EDNS extensions and TSIG authentication
   - Latency histogram generation for detailed analysis
2. **DNS Resolution Performance Testing (resperf)**
   - Evaluate DNS server performance under increasing load
   - Determine maximum sustainable query rate
   - Generate plot data for performance visualization
   - Configurable test phases (ramp-up, constant traffic, wait)
   - Support for multiple transport protocols
   - Detailed performance metrics and statistics
3. **EDNS Client Subnet Generation (dnsperf-ecs-gen.py)**
   - Generate EDNS Client Subnet options for geolocation testing
   - Simulate queries from different network locations
   - Easy integration with dnsperf and resperf
4. **DNS Query Analysis (queryparse)**
   - Extract and analyze DNS queries from packet captures
   - Generate query type statistics
   - Filter queries based on recursion flags
   - Process both queries and responses

Protocol Support

- **UDP**: Standard connectionless DNS protocol (port 53)
- **TCP**: Connection-oriented DNS protocol (port 53)
- **DNS over TLS (DoT)**: Encrypted DNS over TLS (port 853)
- **DNS over HTTPS (DoH)**: Encrypted DNS over HTTP/2 (port 443)

Authentication Support

- **TSIG Authentication**: Support for multiple HMAC algorithms
  - MD5
  - SHA1
  - SHA224
  - SHA256
  - SHA384
  - SHA512

Performance Measurement

- **Queries Per Second (QPS)**: Measure sustained query rate
- **Latency Statistics**: Average, minimum, maximum, and standard deviation
- **Connection Statistics**: Connection attempts, successes, and latency
- **Response Code Distribution**: Analysis of server response codes

- **Packet Size Analysis**: Average request and response sizes

User Interface Guide

The DNS Performance Testing Toolkit provides command-line interfaces for all its components:

dnsperf Command-Line Interface

```
dnsperf [-f family] [-M mode] [-s server_addr] [-p port] [-a local_addr]
        [-x local_port] [-d datafile] [-c clients] [-T threads] [-n maxruns]
        [-l timelimit] [-b buffer_size] [-t timeout] [-e] [-E code:value]
        [-D] [-y [alg:]name:secret] [-q num_queries] [-Q max_qps]
        [-S stats_interval] [-u] [-B] [-v] [-W]
        [--doh-uri uri] [--doh-method method] [--suppress message-types]
        [--num-queries-per-conn queries] [--verbose-interval-stats]
        [--latency-histogram] [--qps-threshold-wait microseconds]
        [--tls-sni name]
```

Key options:

- `-s server_addr`: DNS server address
- `-p port`: DNS server port
- `-d datafile`: Input data file containing queries
- `-M mode`: Transport mode (udp, tcp, dot, doh)
- `-c clients`: Number of clients (sockets)
- `-T threads`: Number of threads
- `-Q max_qps`: Maximum queries per second
- `-S stats_interval`: Statistics interval

resperf Command-Line Interface

```
resperf [-f family] [-M mode] [-s server_addr] [-p port] [-a local_addr]
        [-x local_port] [-d datafile] [-t timeout] [-b buffer_size] [-e]
        [-E code:value] [-D] [-y [alg:]name:secret] [-i plot_interval]
        [-m max_qps] [-P plotfile] [-r ramp_time] [-c constant_traffic_time]
        [-L max_query_loss] [-C clients] [-q num_outstanding] [-v] [-W] [-R]
        [-F fall_behind] [--doh-uri uri] [--doh-method method]
        [--tls-sni name] [--suppress message-types]
        [--num-queries-per-conn queries]
```

Key options:

- `-s server_addr`: DNS server address
- `-p port`: DNS server port
- `-d datafile`: Input data file containing queries
- `-M mode`: Transport mode (udp, tcp, dot, doh)
- `-m max_qps`: Maximum queries per second
- `-r ramp_time`: Ramp-up time in seconds
- `-c constant_traffic_time`: Constant traffic time in seconds
- `-P plotfile`: Plot data file name

dnsperf-ecs-gen.py Command-Line Interface

`dnsperf-ecs-gen.py <address/srclen[/scopelen]>`

Example:

`dnsperf-ecs-gen.py 192.0.2.1/24`

queryparse Command-Line Interface

`queryparse [-i FILE] [-o FILE] [-r] [-R]`

Key options:

- `-i FILE`: Input packet capture file
- `-o FILE`: Output file for parsed queries
- `-r`: Keep queries with RD flag set to 0
- `-R`: Parse query responses instead of queries

Step-by-Step Tutorials

Tutorial 1: Basic DNS Performance Testing

1. **Create a query file**: Create a file named `queries.txt` with the following content:

   ```
   example.com A
   www.example.com A
   mail.example.com MX
   ```

2. **Run a basic performance test**:

   ```
   dnsperf -s 8.8.8.8 -d queries.txt
   ```

3. **Analyze the results**: Look at the statistics output to determine queries per second, latency, and other metrics.

Tutorial 2: Testing with Different Transport Protocols

1. **Test with UDP (default)**:

   ```
   dnsperf -s 8.8.8.8 -d queries.txt
   ```

2. **Test with TCP**:

   ```
   dnsperf -s 8.8.8.8 -d queries.txt -M tcp
   ```

3. **Test with DNS over TLS**:

   ```
   dnsperf -s 8.8.8.8 -p 853 -d queries.txt -M dot
   ```

4. **Test with DNS over HTTPS**:

   ```
   dnsperf -s 8.8.8.8 -p 443 -d queries.txt -M doh --doh-uri /dns-query
   ```

5. **Compare the results** to see performance differences between protocols.

Tutorial 3: Finding Maximum Sustainable Query Rate

1. **Create a query file** with a variety of domain names.
2. **Run resperf with default settings**:

   ```
   resperf -s 8.8.8.8 -d queries.txt
   ```

3. **Adjust the ramp-up time** for more detailed results:

   ```
   resperf -s 8.8.8.8 -d queries.txt -r 120
   ```

4. **Add a constant traffic phase** to verify stability:

   ```
   resperf -s 8.8.8.8 -d queries.txt -r 60 -c 30
   ```

5. **Analyze the plot data file** (resperf.gnuplot) to visualize the performance curve.

Tutorial 4: Testing with EDNS Client Subnet

1. **Generate an ECS option**:

   ```
   dnsperf-ecs-gen.py 192.0.2.1/24
   ```

2. **Use the ECS option with dnsperf**:

   ```
   dnsperf -s 8.8.8.8 -d queries.txt -e -E 8:00010000200001c0000201
   ```

3. **Run tests with different subnet values** to simulate queries from different locations.

Tutorial 5: Analyzing DNS Queries from Packet Captures

1. **Capture DNS traffic** using tcpdump or Wireshark:

   ```
   tcpdump -i eth0 -w dns_traffic.pcap port 53
   ```

2. **Analyze the captured queries**:

   ```
   queryparse -i dns_traffic.pcap -o extracted_queries.txt
   ```

3. **View query statistics** which are output to stderr.

4. **Use the extracted queries for performance testing**:

   ```
   dnsperf -s 8.8.8.8 -d extracted_queries.txt
   ```

Usage Scenarios / Use Cases

Scenario 1: DNS Server Capacity Planning **Goal**: Determine if a DNS server can handle expected traffic volume.

Steps:

1. Create a query file that represents typical DNS queries for your environment.
2. Run dnsperf with increasing QPS values:

   ```
   dnsperf -s your.dns.server -d queries.txt -Q 1000
   dnsperf -s your.dns.server -d queries.txt -Q 2000
   dnsperf -s your.dns.server -d queries.txt -Q 5000
   ```

3. Monitor server resources during testing.
4. Determine the maximum sustainable QPS before errors or timeouts increase.
5. Plan server capacity based on the results, ensuring adequate headroom for peak traffic.

Scenario 2: Protocol Performance Comparison **Goal**: Compare performance of different DNS transport protocols.

Steps:

1. Create a consistent query file for testing.
2. Test with each protocol:

   ```
   dnsperf -s your.dns.server -d queries.txt -M udp
   dnsperf -s your.dns.server -d queries.txt -M tcp
   dnsperf -s your.dns.server -d queries.txt -M dot -p 853
   dnsperf -s your.dns.server -d queries.txt -M doh -p 443 --doh-uri /dns-query
   ```

3. Compare QPS and latency metrics across protocols.
4. Evaluate the trade-offs between performance and security/privacy.

Scenario 3: DNS Server Configuration Optimization **Goal**: Optimize DNS server configuration for maximum performance.

Steps:

1. Create a baseline performance measurement with current configuration.
2. Modify one server parameter at a time (cache size, thread count, etc.).
3. Run resperf after each change:

   ```
   resperf -s your.dns.server -d queries.txt -r 60
   ```

4. Compare maximum sustainable QPS across different configurations.
5. Implement the configuration that provides the best performance.

Scenario 4: Geolocation-Based DNS Testing **Goal**: Test DNS server responses for clients from different geographic locations.

Steps:

1. Generate ECS options for different geographic regions:

   ```
   dnsperf-ecs-gen.py 192.0.2.1/24    # Region 1
   dnsperf-ecs-gen.py 198.51.100.1/24 # Region 2
   dnsperf-ecs-gen.py 203.0.113.1/24  # Region 3
   ```

2. Run dnsperf with each ECS option:

   ```
   dnsperf -s your.dns.server -d queries.txt -e -E 8:00010000200001c0000201
   ```

3. Compare responses to verify correct geolocation-based behavior.

Scenario 5: DNS Traffic Analysis and Replay **Goal**: Analyze real-world DNS traffic and replay it for performance testing.

Steps:

1. Capture DNS traffic from a production environment.
2. Extract queries using queryparse:

   ```
   queryparse -i captured_traffic.pcap -o extracted_queries.txt
   ```

3. Analyze query patterns and distribution.
4. Replay the extracted queries against test servers:

   ```
   dnsperf -s test.dns.server -d extracted_queries.txt
   ```

5. Compare performance with production traffic patterns.

Troubleshooting Common Issues

Connection Failures **Issue**: dnsperf or resperf cannot connect to the DNS server.

Solutions:

- Verify the server address and port are correct.
- Check network connectivity (ping, traceroute).
- Ensure firewall rules allow the traffic.
- For DoT/DoH, verify the server supports these protocols.
- Try using the `-v` (verbose) option for more detailed error messages.

Low Performance **Issue**: Performance is significantly lower than expected.

Solutions:

- Increase the number of clients (`-c`) and threads (`-T`).
- Check for network bandwidth limitations.
- Verify the server is not resource-constrained.
- For TCP/DoT/DoH, connection establishment overhead may limit performance.
- Try adjusting socket buffer size (`-b`).
- Ensure the query file is properly formatted.

Query Timeouts **Issue**: Many queries are timing out.

Solutions:

- Increase the timeout value (`-t`).
- Reduce the query rate (`-Q`).
- Check server load and responsiveness.
- Verify network conditions between client and server.
- For high-volume testing, ensure adequate resources on both client and server.

Input File Errors **Issue**: Errors reading the query file.

Solutions:

- Verify the file exists and is readable.
- Check file format (one query per line: "name type").
- For binary format (`-B`), ensure proper wire format.
- Try using a simpler query file to isolate the issue.

Protocol-Specific Issues **Issue**: Problems with specific transport protocols.

Solutions:

- **UDP**: Check for packet size limitations (consider enabling EDNS).
- **TCP**: Verify server accepts TCP connections on port 53.
- **DoT**: Ensure server certificate is valid and trusted.
- **DoH**: Verify correct URI path and HTTP method.

Memory Usage Issues **Issue**: High memory usage or out-of-memory errors.

Solutions:

- Reduce the number of outstanding queries (`-q`).
- Decrease the number of clients and threads.
- For resperf, adjust the maximum outstanding queries parameter.
- Ensure adequate system resources for the test configuration.

Developer Guide

Codebase Overview

The DNS Performance Testing Toolkit consists of several key components organized in a modular structure:

Core Components

1. **dnsperf**: Main performance testing tool for measuring DNS server query performance
2. **resperf**: Specialized tool for evaluating DNS resolution performance under increasing load

Network Protocol Implementations

1. **UDP**: Basic connectionless DNS protocol implementation
2. **TCP**: Connection-oriented DNS protocol with state management
3. **DoT (DNS over TLS)**: Encrypted DNS using TLS
4. **DoH (DNS over HTTPS)**: DNS over HTTP/2 with TLS

DNS Protocol Components

1. **DNS Module**: Core DNS message handling
2. **EDNS Module**: Extension mechanisms for DNS
3. **TSIG Module**: Transaction signature authentication
4. **Query Type Module**: DNS record type handling

Utility Components

1. **OS Module**: Operating system interaction and thread management
2. **Logging Module**: Error reporting and logging
3. **Option Parsing Module**: Command-line argument handling
4. **Datafile Module**: Input data file processing

Contrib Utilities

1. **ECS Generator**: Python utility for generating EDNS Client Subnet options
2. **Queryparse**: Python utility for analyzing DNS queries from packet captures

The codebase follows a hybrid architecture combining multi-threading, event-driven I/O, state machines, and layered design to achieve high performance and flexibility.

Folder Structure & Key Components

```
/
 src/                  # Core source code
    dnsperf.c         # Main dnsperf implementation
    resperf.c         # Main resperf implementation
    net.c             # Network abstraction layer
    net_udp.c         # UDP protocol implementation
    net_tcp.c         # TCP protocol implementation
    net_dot.c         # DNS over TLS implementation
    net_doh.c         # DNS over HTTPS implementation
    dns.c             # DNS message handling
    edns.c            # EDNS extension support
    tsig.c            # TSIG authentication
    datafile.c        # Input file processing
    os.c              # OS interaction utilities
    log.c             # Logging and error reporting
    opt.c             # Option parsing
    util.h            # Utility functions and macros
    test/             # Test files and scripts

 contrib/              # Contributed utilities
    ecs-gen/          # EDNS Client Subnet generator
       dnsperf-ecs-gen.py  # ECS generation script
    queryparse/       # DNS query analysis tool
        queryparse    # Query parsing script

 doc/                  # Documentation
```

```
m4/                  # Autoconf macros

scripts/             # Build and utility scripts
```

Installation for Development

To set up a development environment for the DNS Performance Testing Toolkit:

1. **Clone the repository**:

   ```
   git clone https://github.com/DNS-OARC/dnsperf.git
   cd dnsperf
   ```

2. **Install development dependencies**: For Debian/Ubuntu:

   ```
   apt-get install build-essential autoconf automake libtool libssl-dev libnghttp2-dev python3 python3
   pip3 install dnspython pcapy
   ```

3. **Generate the configure script**:

   ```
   ./autogen.sh
   ```

4. **Configure for development**:

   ```
   ./configure --enable-gcov
   ```

5. **Build the software**:

   ```
   make
   ```

6. **Run tests**:

   ```
   make check
   ```

Build and Deployment Process

Build Process The DNS Performance Testing Toolkit uses the GNU Autotools build system:

1. **Generate build files**:

   ```
   ./autogen.sh
   ```

2. **Configure the build**:

   ```
   ./configure [options]
   ```

3. **Build the software**:

   ```
   make
   ```

4. **Run tests**:

   ```
   make check
   ```

5. **Install**:

   ```
   make install
   ```

Deployment Options

1. **Direct Installation**: Install directly from source as described above.
2. **Package Creation**: The project includes support for creating distribution packages:
   - **Debian/Ubuntu**:

     ```
     dpkg-buildpackage -us -uc
     ```

   - **RPM-based (Red Hat/CentOS/Fedora)**:

```
rpmbuild -ba rpm/dnsperf.spec
```

3. **Docker Deployment**: A Dockerfile is provided for containerized deployment:

```
docker build -t dnsperf .
docker run -it dnsperf
```

Coding Standards and Conventions

The DNS Performance Testing Toolkit follows these coding standards:

1. **C Code Style**:
   - Indentation: 4 spaces (no tabs)
   - Line length: 80 characters preferred, 100 maximum
   - Function naming: lowercase with underscores (e.g., `perf_net_open`)
   - Variable naming: lowercase with underscores
   - Constants and macros: uppercase with underscores
2. **Code Organization**:
   - One primary functionality per file
   - Related functions grouped together
   - Clear separation between interface and implementation
3. **Documentation**:
   - Function headers with description, parameters, and return values
   - File headers with copyright and license information
   - Inline comments for complex logic
4. **Error Handling**:
   - Consistent error reporting through the logging module
   - Return value checking for all function calls
   - Proper resource cleanup in error paths
5. **Memory Management**:
   - Explicit memory allocation and deallocation
   - Avoid memory leaks by tracking all allocations
   - Use appropriate data structures for the task
6. **Thread Safety**:
   - Explicitly document thread-safety of functions
   - Use appropriate synchronization primitives
   - Avoid global state when possible

API Documentation

Information not available: Detailed API documentation for individual functions and modules. The source code contains function headers, but a comprehensive API reference is not provided in the available documentation.

Database Schema and Interaction

Information not available: The DNS Performance Testing Toolkit does not appear to use a database for operation.

Testing

Testing Strategy Overview

The DNS Performance Testing Toolkit employs a comprehensive testing strategy that focuses on functional validation, error handling, and performance verification. The testing approach is primarily based on shell script-driven integration tests rather than traditional unit testing frameworks.

Testing Approach The project uses a shell script-based testing approach implemented through seven test scripts that validate different aspects of the software:

1. **Command-Line Interface Testing (test1.sh)**

- Validates command-line options and help functionality
- Tests error handling for invalid options

2. **Network Protocol Testing (test2.sh)**
   - Tests DNS queries over UDP, TCP, DoT, and DoH
   - Validates TSIG authentication with various hash algorithms
   - Tests IPv4 and IPv6 server communication
3. **Error Handling Testing (test3.sh)**
   - Validates error handling for invalid inputs and configurations
   - Tests behavior with non-existent files and invalid parameters
4. **Domain Name Processing (test4.sh)**
   - Tests domain name parsing and validation
   - Verifies warning generation for malformed inputs
5. **Server Interaction Testing (test5.sh)**
   - Tests interaction with dummy DNS servers
   - Validates SSL/TLS certificate handling for secure protocols
6. **Response Validation (test6.sh)**
   - Tests handling of specific DNS response types
   - Validates NXDOMAIN and other response scenarios
7. **Binary Data Handling (test7.sh)**
   - Tests handling of binary blob files
   - Validates edge cases in query transmission

Types of Tests

Functional Tests The primary testing approach uses shell scripts to validate the functionality of the tools:

- **Integration Tests**: Test the complete functionality of dnsperf and resperf
- **Protocol Tests**: Verify support for different transport protocols
- **Error Handling Tests**: Ensure proper behavior with invalid inputs
- **Edge Case Tests**: Test boundary conditions and special cases

Code Coverage The project supports code coverage analysis using gcov, which can be enabled with the `--enable-gcov` configuration option. This generates coverage reports for all source files when tests are run.

Performance Testing As performance testing tools themselves, dnsperf and resperf include self-validation mechanisms:

- **Consistency Checks**: Verify consistent behavior across runs
- **Resource Monitoring**: Track memory and CPU usage during tests
- **Error Rate Analysis**: Monitor query timeouts and failures

Running Tests Locally

To run the test suite locally:

1. **Build the software with tests enabled**:

   ```
   ./configure
   make
   ```

2. **Run the test suite**:

   ```
   make check
   ```

3. **Run individual test scripts**:

   ```
   cd src/test
   ./test1.sh
   ./test2.sh
   # etc.
   ```

4. **Enable code coverage reporting**:

```
./configure --enable-gcov
make check
# Coverage data will be generated in .gcda and .gcno files
```

Continuous Integration & Testing

The project uses Travis CI for continuous integration, which runs the complete test suite on each commit. The CI pipeline:

1. Builds the project with different compilers (gcc and clang)
2. Runs all test scripts
3. Captures and reports test results

The CI configuration is defined in the `.travis.yml` file in the project root.

Known Issues and Test Results

Information not available: Specific known issues and recent test results are not documented in the available materials.

Configuration and Deployment

Configuration Management

The DNS Performance Testing Toolkit is primarily configured through command-line options rather than configuration files. This approach provides flexibility for different testing scenarios without requiring persistent configuration.

dnsperf Configuration Options Key configuration options for dnsperf include:

Option	Description	Default
`-s server_addr`	DNS server address	127.0.0.1
`-p port`	DNS server port	Protocol-dependent (53, 853, or 443)
`-M mode`	Transport mode (udp, tcp, dot, doh)	udp
`-c clients`	Number of clients (sockets)	1
`-T threads`	Number of threads	1
`-q num_queries`	Maximum number of outstanding queries	100
`-Q max_qps`	Maximum queries per second	0 (unlimited)
`-t timeout`	Timeout for query completion (seconds)	5

resperf Configuration Options Key configuration options for resperf include:

Option	Description	Default
`-s server_addr`	DNS server address	127.0.0.1
`-p port`	DNS server port	Protocol-dependent (53, 853, or 443)
`-M mode`	Transport mode (udp, tcp, dot, doh)	udp
`-m max_qps`	Maximum queries per second	100000
`-r ramp_time`	Ramp-up time (seconds)	60
`-c constant_traffic_time`	Constant traffic time (seconds)	0
`-L max_query_loss`	Maximum acceptable query loss (percent)	100

Environment Variables

While the toolkit primarily uses command-line options, some environment variables can affect its behavior:

Variable	Description	Default
`OPENSSL_CONF`	Path to OpenSSL configuration file	System default
`LD_LIBRARY_PATH`	Path to search for shared libraries	System default
`DNSPERF_DATAFILE`	Default data file path	None
`DNSPERF_SERVER`	Default server address	None

Deployment Guide

Standalone Deployment

1. **Install dependencies**:

   ```
   # Debian/Ubuntu
   apt-get install libssl-dev libnghttp2-dev

   # Red Hat/CentOS/Fedora
   yum install openssl-devel nghttp2-devel
   ```

2. **Build and install**:

   ```
   ./configure
   make
   make install
   ```

3. **Verify installation**:

   ```
   dnsperf -h
   resperf -h
   ```

Docker Deployment

1. **Build the Docker image**:

   ```
   docker build -t dnsperf .
   ```

2. **Run dnsperf in a container**:

   ```
   docker run -it dnsperf dnsperf -s 8.8.8.8 -d /path/to/queries.txt
   ```

3. **Mount a local query file**:

   ```
   docker run -it -v $(pwd)/queries.txt:/queries.txt dnsperf dnsperf -s 8.8.8.8 -d /queries.txt
   ```

Package-based Deployment

1. **Debian/Ubuntu**:

   ```
   # Install from package repository
   apt-get install dnsperf

   # Or build and install a local package
   dpkg-buildpackage -us -uc
   dpkg -i ../dnsperf_*.deb
   ```

2. **Red Hat/CentOS/Fedora**:

   ```
   # Install from package repository
   yum install dnsperf
   ```

```
# Or build and install a local package
rpmbuild -ba rpm/dnsperf.spec
rpm -i ~/rpmbuild/RPMS/*/dnsperf-*.rpm
```

Scaling Considerations

When scaling DNS performance testing to high query volumes, consider the following:

1. **System Resources**:
   - Ensure sufficient CPU cores (increase `-T` threads accordingly)
   - Provide adequate memory (at least 2GB for high-volume testing)
   - Check network interface capacity (1Gbps+ recommended for >100k QPS)
2. **Socket Limits**:
   - Increase system socket limits:
     `ulimit -n 65536`
   - Adjust kernel parameters:
     `sysctl -w net.ipv4.ip_local_port_range="1024 65535"`
3. **Multiple Instances**:
   - For extremely high loads, run multiple instances from different machines
   - Aggregate results for comprehensive analysis
4. **Network Considerations**:
   - Ensure low-latency network path to the DNS server
   - Consider testing from multiple network locations
   - Be aware of potential rate limiting by intermediate firewalls
5. **Query File Optimization**:
   - Use large, diverse query files for realistic testing
   - Consider using the `-R` option with resperf to reuse query files

Backup and Restore Procedures

Information not available: The DNS Performance Testing Toolkit does not appear to require specific backup and restore procedures as it does not maintain persistent state or databases.

Integration and APIs

API Endpoints Documentation

Information not available: The DNS Performance Testing Toolkit does not expose API endpoints for external integration. It operates as a command-line tool rather than a service with API endpoints.

External Integrations and Dependencies

The DNS Performance Testing Toolkit integrates with several external libraries and services:

Core Dependencies

1. **OpenSSL**
   - **Purpose**: Provides cryptographic operations and TLS support
   - **Integration Method**: Linked library
   - **Configuration**: Configured during build process
   - **Version Requirements**: Minimum TLS 1.2 support
2. **nghttp2**
   - **Purpose**: Provides HTTP/2 protocol support for DoH
   - **Integration Method**: Linked library
   - **Configuration**: Configured during build process
   - **Version Requirements**: Latest stable version recommended
3. **POSIX Threads (pthread)**
   - **Purpose**: Provides threading and synchronization primitives

- **Integration Method**: Linked library
- **Configuration**: Standard system library
- **Version Requirements**: POSIX-compliant implementation

Python Dependencies (for contrib utilities)

1. **Python 3.x**
   - **Purpose**: Runtime for contrib utilities
   - **Integration Method**: System requirement
   - **Version Requirements**: Python 3.x
2. **dnspython**
   - **Purpose**: DNS query parsing and manipulation
   - **Integration Method**: Python package
   - **Version Requirements**: 2.x
3. **pcapy**
   - **Purpose**: Packet capture processing
   - **Integration Method**: Python package
   - **Version Requirements**: 0.11.x
4. **libpcap**
   - **Purpose**: Underlying packet capture library
   - **Integration Method**: Linked library
   - **Version Requirements**: 1.9.x

Authentication and Authorization

The DNS Performance Testing Toolkit does not implement its own authentication or authorization mechanisms for tool usage. However, it does support TSIG authentication for DNS messages:

TSIG Authentication

- **Purpose**: Authenticates DNS messages sent to servers
- **Implementation**: Uses HMAC algorithms through OpenSSL
- **Configuration**: Provided via the `-y [alg:]name:secret` option
- **Supported Algorithms**: MD5, SHA1, SHA224, SHA256, SHA384, SHA512

Webhooks and Callback Interfaces

Information not available: The DNS Performance Testing Toolkit does not appear to implement webhooks or callback interfaces.

Data Exchange Formats

The DNS Performance Testing Toolkit uses several data formats for input and output:

Input Formats

1. **Query Input Format**
   - **Format**: Text file with one query per line
   - **Example**: `example.com A`
   - **Usage**: Default input format for dnsperf and resperf
2. **Dynamic Update Format**
   - **Format**: Text file with DNS record specifications
   - **Example**: `example.com. 300 IN A 192.0.2.1`
   - **Usage**: Used with dnsperf's `-u` option
3. **TCP-stream Binary Format**
   - **Format**: Binary DNS messages with 2-byte length prefix
   - **Usage**: Used with dnsperf's `-B` option
4. **Packet Capture Format**

- **Format**: Standard pcap file format
- **Usage**: Input for queryparse utility

Output Formats

1. **Performance Statistics**
   - **Format**: Human-readable text output
   - **Example**: Query counts, QPS, latency statistics
   - **Usage**: Standard output from dnsperf and resperf
2. **Plot Data**
   - **Format**: Tab-separated values
   - **Columns**: Time, target QPS, actual QPS, responses/sec, failures/sec, latency
   - **Usage**: Generated by resperf for visualization
3. **Query Extraction**
   - **Format**: Text file with one query per line
   - **Example**: `example.com A`
   - **Usage**: Output from queryparse utility

Security

Assets in the Software

The DNS Performance Testing Toolkit handles several types of assets that have security implications:

1. **Authentication Credentials**
   - **TSIG Keys**: Shared secrets for DNS message authentication
   - **Security Measures**: Keys are parsed from user input and stored in memory only during runtime
   - **Criticality**: Low - Keys are provided by users at runtime and not persistently stored
2. **Network Connections**
   - **TLS Connections**: Encrypted DNS communication
   - **Security Measures**: Minimum TLS version set to 1.2
   - **Criticality**: Medium - Proper TLS configuration is important for secure communication
3. **Query Data**
   - **DNS Queries**: Domain names and query types
   - **Security Measures**: Input validation and proper handling
   - **Criticality**: Low - Query data is typically not sensitive
4. **Server Certificates**
   - **TLS Server Authentication**: Certificates for DoT and DoH
   - **Security Measures**: Standard OpenSSL certificate validation
   - **Criticality**: Medium - Important for preventing man-in-the-middle attacks

Security Guidelines

When using the DNS Performance Testing Toolkit, consider the following security guidelines:

1. **TSIG Authentication**
   - Use strong hash algorithms (SHA256 or above) rather than MD5 or SHA1
   - Protect TSIG keys from unauthorized access
   - Do not use production TSIG keys for testing
2. **TLS Configuration**
   - Ensure proper TLS configuration on both client and server sides
   - Verify server certificates are valid and trusted
   - Use the `--tls-sni` option to specify the correct server name
3. **Input Validation**
   - Validate all input files and parameters
   - Be cautious with user-supplied query files
   - Sanitize data extracted from packet captures
4. **Network Security**

- Be aware of the network impact of high-volume testing
- Obtain permission before testing against production servers
- Consider network security implications of DNS traffic

5. **Dependency Management**
   - Keep OpenSSL and nghttp2 libraries updated
   - Apply security patches promptly
   - Use the latest stable versions of all dependencies

Data Privacy Considerations

The DNS Performance Testing Toolkit itself does not store or process personal data. However, DNS queries may contain domain names that could be considered sensitive. Consider the following privacy considerations:

1. **Query Data Privacy**
   - DNS queries may reveal browsing habits or sensitive information
   - Anonymize or sanitize real-world query data used for testing
   - Be cautious when sharing query files or test results
2. **Network Traffic**
   - DNS traffic generated during testing may be visible on the network
   - Consider using encrypted protocols (DoT, DoH) for sensitive testing
   - Be aware of network monitoring and data retention policies
3. **EDNS Client Subnet**
   - ECS options include IP address information
   - Use fictional IP addresses for testing when possible
   - Be aware of potential privacy implications when using real subnet information

Vulnerability Management

Information not available: Specific vulnerability management processes for the DNS Performance Testing Toolkit are not documented in the available materials.

Authentication and Authorization Mechanisms

The DNS Performance Testing Toolkit includes the following authentication mechanisms:

1. **TSIG Authentication**
   - **Purpose**: Authenticates DNS messages sent to servers
   - **Implementation**: Uses HMAC algorithms through OpenSSL
   - **Supported Algorithms**: MD5, SHA1, SHA224, SHA256, SHA384, SHA512
   - **Usage**: Provided via the `-y [alg:]name:secret` option
2. **TLS Client Authentication**
   - **Purpose**: Authenticates the client to the server in DoT and DoH
   - **Implementation**: Standard TLS client authentication
   - **Configuration**: Uses OpenSSL for TLS operations
   - **Note**: Client certificate authentication is not explicitly documented

The toolkit does not implement authorization mechanisms as it operates as a client-side testing tool.

Change Log and Release Notes

Versioning Scheme

The DNS Performance Testing Toolkit follows semantic versioning (SemVer):

- **Major Version**: Incremented for incompatible API changes
- **Minor Version**: Incremented for new functionality in a backward-compatible manner
- **Patch Version**: Incremented for backward-compatible bug fixes

Version numbers are in the format `MAJOR.MINOR.PATCH` (e.g., 2.14.0).

Release History

The DNS Performance Testing Toolkit has a long development history, with regular releases providing new features and improvements. Below is a chronological summary of major releases:

Recent Releases

Version	Release Date	Maintainer	Key Focus
2.14.0	2024-01-18	Jerry Lundström	Thread naming and TSIG improvements
2.13.1	2023-08-23	Jerry Lundström	In-progress queries and TCP transport fixes
2.13.0	2023-06-15	Jerry Lundström	TLS SNI support
2.12.0	2023-05-21	Jerry Lundström	DoH fixes and QPS handling improvements
2.11.2	2023-03-16	Jerry Lundström	Long option argument handling fixes
2.11.1	2023-03-10	Jerry Lundström	Long option argument handling fixes
2.11.0	2023-02-08	Jerry Lundström	Latency histograms and verbose statistics
2.10.0	2022-11-11	Jerry Lundström	Binary datafile format
2.9.0	2021-12-08	Jerry Lundström	Message suppression and connection control
2.8.0	2021-11-02	Jerry Lundström	DoH response handling improvements
2.7.1	2021-09-17	Jerry Lundström	DNS wire format construction fixes
2.7.0	2021-08-09	Jerry Lundström	DNS-over-HTTPS support
2.6.0	2021-05-31	Jerry Lundström	EDNS options for resperf
2.5.2	2021-03-25	Jerry Lundström	TCP and DoT reconnect improvements
2.5.1	2021-03-22	Jerry Lundström	TYPEnnn and ANY support
2.5.0	2021-03-12	Jerry Lundström	TCP/DoT reconnection support
2.4.2	2021-02-23	Jerry Lundström	Datafile reading fixes
2.4.1	2021-02-09	Jerry Lundström	Socket readiness function fixes
2.4.0	2020-12-09	Jerry Lundström	Removal of BIND dependency
2.3.4	2020-05-15	Jerry Lundström	BIND 9.16 compatibility
2.3.3	2020-05-06	Jerry Lundström	TCP/TLS connection handling improvements
2.3.2	2019-08-23	Jerry Lundström	TSIG buffer overflow fix
2.3.1	2019-07-24	Jerry Lundström	Network receive code optimization
2.3.0	2019-07-17	Jerry Lundström	TCP and TLS transport support
2.2.1	2019-01-28	Jerry Lundström	Minor formatting fix
2.2.0	2019-01-25	Jerry Lundström	First DNS-OARC release with autotools

Historical Releases

Version	Release Date	Key Focus
2.1.0.0	2015-12-15	Multi-client simulation capabilities
2.0.0.0	2012-03-01	Major overhaul with enhanced output and performance
1.0.2.0	2011-12-22	RHEL6-64 and Solaris 10 x86-64 support
1.0.1.0	2008-01-10	Binary builds and sample query files
1.0.0.1	2006-12-21	Timeout handling fixes

Development Patterns

Analysis of the project's development history reveals several notable patterns:

1. **Regular Release Cadence**: The project has maintained a steady release schedule, with multiple releases per year since being maintained by DNS-OARC.
2. **Consistent Maintainership**: Since 2019, Jerry Lundström has been the primary maintainer, providing stability in the development process.
3. **Incremental Feature Development**: New features are typically introduced in minor version updates, following semantic versioning principles.

4. **Protocol Evolution**: The project has evolved from supporting only UDP to adding TCP (2.3.0), TLS (2.3.0), and HTTP/2 (2.7.0) as DNS transport protocols.
5. **Community Contributions**: Several releases acknowledge contributions from community members, particularly from ISC (Internet Systems Consortium).
6. **Dependency Management**: A significant milestone was the removal of BIND dependencies in version 2.4.0, making the toolkit easier to build and deploy.
7. **Focus on Performance**: Many updates focus specifically on performance optimization, connection handling, and accurate measurement capabilities.

Notable Changes

Major Feature Additions

1. **Transport Protocol Support**:
   - **TCP and TLS Support** (2.3.0): Added support for DNS over TCP and TLS
   - **DNS-over-HTTPS Support** (2.7.0): Added support for DoH with HTTP/2
   - **TLS SNI Support** (2.13.0): Added Server Name Indication for TLS connections
2. **Performance Measurement Improvements**:
   - **Latency Histograms** (2.11.0): Detailed latency distribution analysis
   - **Verbose Interval Statistics** (2.11.0): Enhanced real-time performance reporting
   - **Connection Statistics** (2.5.0): Added metrics for connection attempts and latency
3. **Input/Output Enhancements**:
   - **Binary Datafile Format** (2.10.0): Added support for pre-compiled DNS wire format, improving performance up to 150x for certain use cases
   - **EDNS Client Subnet Support** (2.6.0): Added support for geolocation testing
4. **Operational Improvements**:
   - **Reconnection Support** (2.5.0): Automatic reconnection for TCP and DoT protocols
   - **Message Suppression** (2.9.0): Ability to suppress verbose messages for cleaner output
   - **Thread Naming** (2.14.0): Named threads for better debugging and monitoring

Significant Architectural Changes

1. **Removal of BIND Dependency** (2.4.0):
   - Eliminated dependency on BIND's internal development libraries
   - Simplified building and packaging
   - Reduced external dependencies to OpenSSL and optional LDNS
2. **Multi-threaded Architecture** (2.1.0.0):
   - Enhanced thread management for better performance
   - Added options to control thread count independently from client count
3. **Connection Management** (2.5.0):
   - Improved state tracking for connections
   - Added reconnection capabilities for stateful protocols
   - Enhanced socket readiness detection

Security and Stability Improvements

1. **TSIG Improvements**:
   - Fixed buffer overflow with large digest algorithms (2.3.2)
   - Added per-thread TSIG contexts to prevent crashes (2.14.0)
2. **TLS Enhancements**:
   - Enforced minimum TLS version 1.2 (2.4.0)
   - Added SNI support for better server compatibility (2.13.0)
3. **Error Handling**:
   - Improved socket error detection and recovery (2.11.0)
   - Enhanced handling of connection resets (2.3.3)
   - Fixed query timeout tracking (2.13.1)

This development history demonstrates the project's commitment to maintaining a robust, high-performance DNS testing toolkit that evolves with DNS protocol developments and community needs.

Versioning Scheme

The DNS Performance Testing Toolkit follows semantic versioning (SemVer):

- **Major Version**: Incremented for incompatible API changes
- **Minor Version**: Incremented for new functionality in a backward-compatible manner
- **Patch Version**: Incremented for backward-compatible bug fixes

Version numbers are in the format `MAJOR.MINOR.PATCH` (e.g., 2.3.4).

Release History

Information not available: A detailed release history with version numbers and dates is not provided in the available documentation. The project maintains a CHANGES file that would contain this information.

Notable Changes

Information not available: Specific details about major features, enhancements, and fixes in each release are not provided in the available documentation. The project maintains a CHANGES file that would contain this information.

License and Legal Information

Software Licensing

The DNS Performance Testing Toolkit is open-source software licensed under the Apache License, Version 2.0.

License Details:

- **License**: Apache License, Version 2.0
- **License Text**: Available in the LICENSE file in the project repository
- **License URL**: https://www.apache.org/licenses/LICENSE-2.0

The Apache License 2.0 allows users to:

- Use the software for any purpose
- Distribute the software
- Modify the software
- Distribute modified versions of the software

Under the conditions that they:

- Include the original copyright notice
- Include the license text
- State significant changes made to the software
- Include any existing NOTICE file

Mixed Licensing for Components

While the main project is licensed under Apache License 2.0, some components have different licenses:

1. **src/ext/parse_uri.*** files are licensed under the Tsujikawa License (MIT-style license) with copyright by Tatsuhiro Tsujikawa.
2. **src/ext/hg64.*** files are licensed under the Mozilla Public License 2.0 (MPL-2.0) with copyright by Internet Systems Consortium, Inc. ("ISC").

Copyright Notices

The copyright for the DNS Performance Testing Toolkit is held by multiple organizations over time:

- Copyright © 2019-2024 OARC, Inc.
- Copyright © 2017-2018 Akamai Technologies
- Copyright © 2006-2016 Nominum, Inc.

This reflects the project's development history, having been initially developed by Nominum, then maintained by Akamai Technologies, and now by DNS-OARC since 2019.

Contribution Guidelines

No formal contribution guidelines (such as a CONTRIBUTING.md file or Code of Conduct) are provided in the repository. The project accepts contributions through its GitHub repository at https://github.com/DNS-OARC/dnsperf.

Issues and bug reports can be submitted through the GitHub issue tracker: - https://github.com/DNS-OARC/dnsperf/issues

General support and discussion are available through: - Mattermost: https://chat.dns-oarc.net/community/channels/oarc-software

The project has a history of accepting community contributions, particularly from ISC (Internet Systems Consortium), with proper attribution in the release notes.

Software Licensing

The DNS Performance Testing Toolkit is open-source software licensed under the Apache License, Version 2.0.

License Details:

- **License**: Apache License, Version 2.0
- **License Text**: Available in the LICENSE file in the project repository
- **License URL**: https://www.apache.org/licenses/LICENSE-2.0

The Apache License 2.0 allows users to:

- Use the software for any purpose
- Distribute the software
- Modify the software
- Distribute modified versions of the software

Under the conditions that they:

- Include the original copyright notice
- Include the license text
- State significant changes made to the software
- Include any existing NOTICE file

Contribution Guidelines

Information not available: Specific contribution guidelines for the DNS Performance Testing Toolkit are not provided in the available documentation.

Copyright Notices

Copyright © DNS-OARC, Inc.

The DNS Performance Testing Toolkit includes components with various copyright holders, as specified in the source code files and LICENSE document.

Appendix

References

- IETF DNS Standards
- DNS Performance Testing Guidelines
- EDNS Client Subnet Specification
- DNS-OARC GitHub Repository

Additional Resources and Further Reading

- DNS Performance Measurement Techniques
- Open-Source DNS Tools
- DNS-OARC Workshop Presentations
- DNS Protocol Specifications