@push.rocks/smartdns

DNS client and server implementation, supporting both https and udp.

- readme.md for @push.rocks/smartdns
- changelog.md for @push.rocks/smartdns

readme.md for @push.rocks/smartdns

A TypeScript-first DNS toolkit powered by high-performance Rust binaries — covering everything from simple record lookups to running a full authoritative DNS server with DNSSEC, DNS-over-HTTPS, and automatic Let's Encrypt certificates.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
pnpm install @push.rocks/smartdns
```

Architecture at a Glance □□

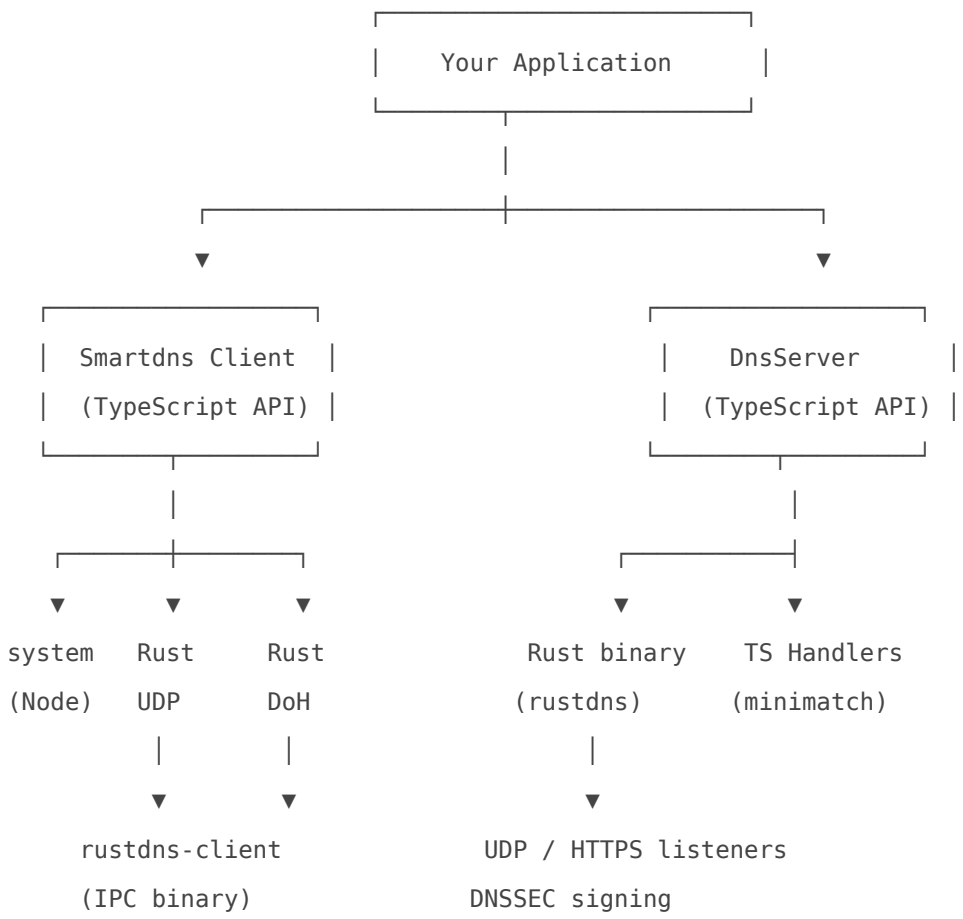
smartdns ships as **three entry points** that you can import independently:

Entry point	What it does
`@push.rocks/smartdns/client`	DNS resolution & record queries (UDP, DoH, system resolver)
`@push.rocks/smartdns/server`	Full DNS server — UDP, DoH, DNSSEC, ACME
`@push.rocks/smartdns`	Convenience re-export of both modules

Both the **client** and the **server** delegate performance-critical work to compiled **Rust binaries** that ship with the package:

- `rustdns` — The server binary: network I/O, packet parsing, DNSSEC signing
- `rustdns-client` — The client binary: UDP wire-format queries, RFC 8484 DoH resolution

TypeScript retains the public API, handler registration, ACME orchestration, and strategy routing. Communication between TypeScript and Rust happens over stdin/stdout JSON IPC via `@push.rocks/smartrust`.



Usage

Quick Start

```
// DNS client — resolve records
import { Smartdns } from '@push.rocks/smartdns/client';

const dns = new Smartdns({});
const records = await dns.getRecordsA('example.com');
console.log(records);
```

```
// DNS server — serve records
import { DnsServer } from '@push.rocks/smartdns/server';

const server = new DnsServer({
  udpPort: 5333,
  httpsPort: 8443,
  httpsKey: '...pem...',
  httpsCert: '...pem...',
  dnssecZone: 'example.com',
});

server.registerHandler('*.example.com', ['A'], (question) => ({
  name: question.name,
  type: 'A',
  class: 'IN',
  ttl: 300,
  data: '192.168.1.100',
}));

await server.start();
```

Or import from the unified entry point:

```
import { dnsClientMod, dnsServerMod } from '@push.rocks/smartdns';

const client = new dnsClientMod.Smartdns({});
const server = new dnsServerMod.DnsServer({ /* ... */ });
```

□□ DNS Client

The `Smartdns` class resolves DNS records using a configurable strategy that combines the system resolver, raw UDP queries, and DNS-over-HTTPS — all backed by a Rust binary for the wire-format transports.

Constructor Options

```
interface ISmartDnsConstructorOptions {
  strategy?: 'doh' | 'udp' | 'system' | 'prefer-system' | 'prefer-udp'; // default: 'prefer-system'
  allowDohFallback?: boolean; // fallback to DoH when system fails (default: true)
  timeoutMs?: number; // per-query timeout in milliseconds
}
```

Resolution Strategies

Strategy	Behavior
`prefer-system`	◫ Try the OS resolver first, fall back to Rust DoH. Honors `/etc/hosts`.
`system`	◫ Use only the Node.js system resolver. No Rust binary needed.
`doh`	◫ Use only DNS-over-HTTPS (RFC 8484 wire format via Cloudflare). Rust-powered.
`udp`	⚡ Use only raw UDP queries to upstream resolver (Cloudflare 1.1.1.1). Rust-powered.
`prefer-udp`	⚡ Try Rust UDP first, fall back to Rust DoH if UDP fails.

The Rust binary (`rustdns-client`) is spawned **lazily** — only on the first query that needs it. This means `system`-only usage incurs zero Rust overhead.

Querying Records

```
const dns = new Smartdns({ strategy: 'prefer-udp' });

// Type-specific helpers
const aRecords    = await dns.getRecordsA('example.com');
const aaaaRecords = await dns.getRecordsAAAA('example.com');
const txtRecords  = await dns.getRecordsTxt('example.com');

// Generic query — supports A, AAAA, CNAME, MX, TXT, NS, SOA, PTR, SRV
const mxRecords = await dns.getRecords('example.com', 'MX');

// Nameserver lookup
const nameservers = await dns.getNameServers('example.com');
```

Every query returns an array of `IDnsRecord`:

```
interface IDnsRecord {
  name: string;
  type: string;          // 'A', 'AAAA', 'TXT', 'MX', etc.
  dnsSecEnabled: boolean; // true if upstream AD flag was set
  value: string;
}
```

DNSSEC Detection □□

When using `doh`, `udp`, or `prefer-udp` strategies, the Rust binary sends queries with the EDNS0 DO (DNSSEC OK) bit set and reports the AD (Authenticated Data) flag from the upstream response:

```
const dns = new Smartdns({ strategy: 'udp' });
const records = await dns.getRecordsA('cloudflare.com');
console.log(records[0].dnsSecEnabled); // true — upstream validated DNSSEC
```

Checking DNS Propagation

Wait for a specific record to appear — essential after making DNS changes:

```
const propagated = await dns.checkUntilAvailable(
  'example.com',
  'TXT',
  'verification=abc123',
  50,   // max check cycles (default: 50)
  500   // interval in ms (default: 500)
);

if (propagated) {
  console.log('Record is live!');
}
```

The method alternates between system resolver and the configured strategy on each cycle for maximum coverage.

Configuring the System DNS Provider

Override the global Node.js DNS resolver for all subsequent lookups:

```
import { makeNodeProcessUseDnsProvider } from '@push.rocks/smartdns/client';

makeNodeProcessUseDnsProvider('cloudflare'); // 1.1.1.1 / 1.0.0.1
makeNodeProcessUseDnsProvider('google');     // 8.8.8.8 / 8.8.4.4
```

Cleanup

When you're done with a `Smartdns` instance (especially one using Rust strategies), call `destroy()` to kill the Rust child process:

```
const dns = new Smartdns({ strategy: 'udp' });
// ... do queries ...
dns.destroy(); // kills rustdns-client process
```

□□ DNS Server

The `DnsServer` class runs a production-capable authoritative DNS server backed by a Rust binary. It supports standard UDP DNS (port 53), DNS-over-HTTPS, DNSSEC signing, and automated Let's Encrypt certificates.

Server Options

```
interface IDnsServerOptions {
  udpPort: number;               // Port for UDP DNS queries
  httpsPort: number;             // Port for DNS-over-HTTPS
  httpsKey: string;              // PEM private key (path or content)
  httpsCert: string;             // PEM certificate (path or content)
  dnssecZone: string;            // Zone for DNSSEC signing
  primaryNameserver?: string;    // SOA mname field (default: 'ns1.{dnssecZone}')
  udpBindInterface?: string;     // IP to bind UDP (default: '0.0.0.0')
  httpsBindInterface?: string;   // IP to bind HTTPS (default: '0.0.0.0')
  manualUdpMode?: boolean;       // Don't auto-bind UDP socket
  manualHttpsMode?: boolean;     // Don't auto-bind HTTPS server
  enableLocalhostHandling?: boolean; // RFC 6761 localhost (default: true)
}
```

Basic Server

```
import { DnsServer } from '@push.rocks/smartdns/server';

const server = new DnsServer({
  udpPort: 5333,
  httpsPort: 8443,
  httpsKey: '...pem...',
  httpsCert: '...pem...',
  dnssecZone: 'example.com',
});

// Register handlers
server.registerHandler('example.com', ['A'], (question) => ({
  name: question.name,
  type: 'A',
  class: 'IN',
  ttl: 300,
  data: '93.184.215.14',
}));

server.registerHandler('example.com', ['TXT'], (question) => ({
  name: question.name,
  type: 'TXT',
  class: 'IN',
  ttl: 300,
  data: 'v=spf1 include:_spf.example.com ~all',
}));

await server.start();
// DNS Server started (UDP: 0.0.0.0:5333, HTTPS: 0.0.0.0:8443)
```

Handler System □□

Handlers use **glob patterns** (via `minimatch`) to match incoming query names. Multiple handlers can contribute records to the same response.

```
// Exact domain
server.registerHandler('example.com', ['A'], handler);

// All subdomains
server.registerHandler('*.example.com', ['A'], handler);

// Specific pattern
server.registerHandler('db-*.internal.example.com', ['A'], (question) => {
  const id = question.name.match(/db-(\d+)/)?.[1];
  return {
    name: question.name,
    type: 'A',
    class: 'IN',
    ttl: 60,
    data: `10.0.1.${id}`,
  };
});

// Catch-all
server.registerHandler('*', ['A'], (question) => ({
  name: question.name,
  type: 'A',
  class: 'IN',
  ttl: 300,
  data: '127.0.0.1',
}));

// Multiple record types
server.registerHandler('example.com', ['MX'], (question) => ({
  name: question.name,
  type: 'MX',
  class: 'IN',
  ttl: 300,
  data: { preference: 10, exchange: 'mail.example.com' },
}));

// Unregister a handler
server.unregisterHandler('example.com', ['A']);
```

When no handler matches, the server automatically returns an **SOA record** for the zone.

DNSSEC

DNSSEC is enabled automatically when you set the `dnssecZone` option. The Rust backend handles:

- **Key generation** — ECDSA P-256 (algorithm 13) by default
- **DNSKEY / DS record** generation
- **RRSIG signing** for all responses
- **NSEC records** for authenticated denial of existence

```
const server = new DnsServer({
  udpPort: 53,
  httpsPort: 443,
  httpsKey: '...',
  httpsCert: '...',
  dnssecZone: 'secure.example.com',
});

// Just register handlers as usual — signing is automatic
server.registerHandler('secure.example.com', ['A'], (q) => ({
  name: q.name,
  type: 'A',
  class: 'IN',
  ttl: 300,
  data: '10.0.0.1',
}));

await server.start();
```

Supported algorithms: **ECDSAP256SHA256** (13), **ED25519** (15), **RSASHA256** (8).

SOA Records

The server auto-generates SOA records for zones when no specific handler matches. Customize the primary nameserver:

```
const server = new DnsServer({
  // ...
  dnssecZone: 'example.com',
  primaryNameserver: 'ns1.example.com', // defaults to 'ns1.{dnssecZone}'
});
```

```
// Generated SOA includes:
// mname:   ns1.example.com
// rname:   hostmaster.example.com
// serial:  Unix timestamp
// refresh: 3600, retry: 600, expire: 604800, minimum: 86400
```

Let's Encrypt Integration □□

Built-in ACME DNS-01 challenge support for automatic SSL certificates:

```
const server = new DnsServer({
  udpPort: 53,
  httpsPort: 443,
  httpsKey: '/path/to/key.pem',
  httpsCert: '/path/to/cert.pem',
  dnssecZone: 'example.com',
});

await server.start();

const result = await server.retrieveSslCertificate(
  ['example.com', 'www.example.com'],
  {
    email: 'admin@example.com',
    staging: false,
    certDir: './certs',
  }
);

if (result.success) {
  console.log('Certificate installed!');
  // The server automatically:
  // 1. Registers temporary _acme-challenge TXT handlers
  // 2. Completes DNS-01 validation
  // 3. Updates the HTTPS server with the new cert
  // 4. Cleans up challenge handlers
}
```

Interface Binding

Restrict the server to specific network interfaces:

```
// Localhost only — great for development
const server = new DnsServer({
  // ...
  udpBindInterface: '127.0.0.1',
  httpsBindInterface: '127.0.0.1',
});

// Different interfaces per protocol
const server = new DnsServer({
  // ...
  udpBindInterface: '192.168.1.100',
  httpsBindInterface: '10.0.0.50',
});
```

Manual Socket Handling □□

For clustering, load balancing, or custom transports, take control of socket management:

```
import { DnsServer } from '@push.rocks/smartdns/server';
import * as dgram from 'dgram';

// Manual UDP mode — you control the socket
const server = new DnsServer({
  // ...
  manualUdpMode: true,
});

await server.start(); // HTTPS auto-binds, UDP does not

const socket = dgram.createSocket('udp4');
socket.on('message', (msg, rinfo) => {
  server.handleUdpMessage(msg, rinfo, (response, responseRinfo) => {
    socket.send(response, responseRinfo.port, responseRinfo.address);
  });
});
```

```
socket.bind(5353);
```

Full manual mode (both protocols):

```
const server = new DnsServer({
  // ...
  manualUdpMode: true,
  manualHttpsMode: true,
});

await server.start(); // Neither protocol binds automatically
```

Process individual DNS packets directly:

```
// Synchronous (TypeScript fallback)
const response = server.processRawDnsPacket(packetBuffer);

// Asynchronous (via Rust bridge — includes DNSSEC signing)
const response = await server.processRawDnsPacketAsync(packetBuffer);
```

Load Balancing Example

```
import * as dgram from 'dgram';
import * as os from 'os';

const numCPUs = os.cpus().length;

for (let i = 0; i < numCPUs; i++) {
  const socket = dgram.createSocket({ type: 'udp4', reuseAddr: true });

  socket.on('message', (msg, rinfo) => {
    server.handleUdpMessage(msg, rinfo, (response, rinfo) => {
      socket.send(response, rinfo.port, rinfo.address);
    });
  });

  socket.bind(53);
}
```

Stopping the Server

```
await server.stop();
```

This gracefully shuts down the Rust process and releases all bound sockets.

□□ Rust Crate Structure

The Rust workspace (`rust/crates/`) contains five crates:

Crate	Purpose
`rustdns`	Server binary — IPC management loop, handler callback routing
`rustdns-client`	Client binary — stateless UDP/DoH query proxy
`rustdns-protocol`	DNS wire format parsing, encoding, and RDATA decode/encode
`rustdns-server`	Async UDP + HTTPS servers (tokio, hyper, rustls)
`rustdns-dnssec`	ECDSA/ED25519 key generation and RRset signing

Pre-compiled binaries for `linux_amd64` and `linux_arm64` are included in `dist_rust/`. Cross-compilation is handled by `@git.zone/tsrust`.

□□ Testing

```
# Run all tests
pnpm test

# Run specific test file
tstest test/test.client.ts --verbose
tstest test/test.server.ts --verbose
```

Example test:

```
import { expect, tap } from '@git.zone/tstest/tapbundle';
import { Smartdns } from '@push.rocks/smartdns/client';

tap.test('resolve A records via UDP', async () => {
  const dns = new Smartdns({ strategy: 'udp' });
  const records = await dns.getRecordsA('google.com');
  expect(records).toBeArray();
  expect(records[0]).toHaveProperty('type', 'A');
  expect(records[0]).toHaveProperty('value');
  dns.destroy();
});

tap.test('detect DNSSEC via DoH', async () => {
  const dns = new Smartdns({ strategy: 'doh' });
  const records = await dns.getRecordsA('cloudflare.com');
  expect(records[0].dnsSecEnabled).toBeTrue();
  dns.destroy();
});

export default tap.start();
```

License and Legal Information

Trademarks

Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

changelog.md for @push.rocks/smartdns

2026-02-20 - 7.9.0 - feat(server)

emit query events with questions, answered status, response time and timestamp

- Added IDnsQueryCompletedEvent interface with questions, answered, responseTimeMs and timestamp fields
- DnsServer now extends EventEmitter and calls super() in constructor
- DnsServer emits a 'query' event on incoming dnsQuery from Rust bridge, providing answers and timing
- Imported IIpcDnsQuestion and used TypeScript 'satisfies' for the emitted event object

2026-02-12 - 7.8.1 - fix(server)

Require Rust bridge for DNS packet processing; remove synchronous TypeScript fallback; change handler API to accept IDnsQuestion and adjust query API

- Breaking API change: handler signature changed from dns-packet.Question to IDnsQuestion — update registered handlers accordingly.
- Synchronous TypeScript fallback (processDnsRequest/processRawDnsPacket) removed; callers must start the server/bridge and use the async bridge path (processRawDnsPacketAsync) or the new resolveQuery API.
- processRawDnsPacketAsync now throws if the Rust bridge is not started — call start() before processing packets.
- Public/test API rename/adjustments: processDnsRequest usages were replaced with resolveQuery and tests updated to use tapbundle_serverside.
- Dependency changes: moved dns-packet to devDependencies, bumped @push.rocks/smartenv to ^6.0.0, updated @git.zone/* build/test tools and @types/node; removed @push.rocks/smartrequest from client plugin exports.
- Plugins: dns-packet removed from exported plugins and minimatch kept; ts_client no longer exports smartrequest.

2026-02-11 - 7.8.0 - feat(rustdns-client)

add Rust DNS client binary and TypeScript IPC bridge to enable UDP and DoH resolution, RDATA decoding, and DNSSEC AD/rcode support

- Add new rust crate rustdns-client with IPC management, DoH and UDP resolvers (resolver_doh.rs, resolver_udp.rs) and ipc types
- Integrate Rust client via a new TypeScript RustDnsClientBridge that spawns rustdns-client and communicates over JSON IPC
- Expose Rust-based resolution from Smartdns (new strategies: 'udp', 'prefer-udp'; DoH routed through Rust) and add destroy() to clean up the bridge
- Extend rustdns-protocol with RDATA decoders (A, AAAA, TXT, MX, NS/CNAME/PTR name decoding, SOA, SRV), AD flag detection and rcode() helper
- Update tests to cover Rust/UDP/DoH paths, DNSSEC AD flag, SOA round-trip and performance assertions
- Update packaging/readmes and build metadata (npmextra.json, ts_client/readme, ts_server/readme) and Cargo manifests/lock for the new crate

2026-02-11 - 7.7.1 - fix(tests)

prune flaky SOA integration and performance tests that rely on external tools and long-running signing/serialization checks

- Removed 'Test raw SOA serialization' from test/test.soa.debug.ts
- Removed dig-based 'Test SOA timeout with real dig command' from test/test.soa.timeout.ts
- Removed 'Check DNSSEC signing performance for SOA' and related serialization/signing performance checks
- Removed unused imports (plugins, execSync) and testPort constant; minor whitespace/cleanup in stopServer

2026-02-11 - 7.7.0 - feat(rust)

add Rust-based DNS server backend with IPC management and TypeScript bridge

- Adds a new rust/ workspace with crates: rustdns, rustdns-protocol, rustdns-server, rustdns-dnssec (DNS packet parsing/encoding, UDP/HTTPS servers, DNSSEC signing).
- Implements an IPC management loop and command/event protocol (stdin/stdout) for communication between Rust and TypeScript (ipc_types, management).
- Introduces DnsResolver and DNSSEC key/signing logic in Rust (keys, signing, keytag), plus UDP and DoH HTTPS server implementations.
- Adds a TypeScript Rust bridge (ts_server/classes.rustdnsbridge.ts) using @push.rocks/smartrust to spawn and talk to the Rust binary; exposes spawn/start/stop/processPacket/ping APIs.
- Removes JS-based DNSSEC implementation and updates ts_server plugins to use smartrust; adds tsrust integration and tsrust devDependency and build step in package.json.
- Documentation and tooling: README updated with Rust backend architecture, .gitignore updated for rust/target, Cargo config for cross-compile linker added.

2025-09-12 - 7.6.1 - fix(classes.dnsclient)

Remove redundant DOH response parsing in getRecords to avoid duplicate processing and clean up client code

- Removed a duplicated/extra iteration that parsed DNS-over-HTTPS (DoH) answers in ts_client/classes.dnsclient.ts.
- Prevents double-processing or incorrect return behavior from Smartdns.getRecords when using DoH providers.
- Changes affect the Smartdns client implementation (ts_client/classes.dnsclient.ts).

2025-09-12 - 7.6.0 - feat(dnsserver)

Return multiple matching records, improve DNSSEC RRset signing, add client resolution strategy and localhost handling, update tests

- Server: process all matching handlers for a question so multiple records (NS, A, TXT, etc.) are returned instead of stopping after the first match
- DNSSEC: sign entire RRsets together (single RRSIG per RRset) and ensure DNSKEY/DS generation and key-tag computation are handled correctly

- Server: built-in localhost handling (RFC 6761) with an enableLocalhostHandling option and synthetic answers for localhost/127.0.0.1 reverse lookups
- Server: improved SOA generation (primary nameserver handling), name serialization (trim trailing dot), and safer start/stop behavior
- Client: added resolution strategy options (doh | system | prefer-system), allowDohFallback and per-query timeout support; improved DoH and system lookup handling (proper TXT quoting and name trimming)
- Tests: updated expectations and test descriptions to reflect correct multi-record behavior and other fixes

2025-09-12 - 7.5.1 - fix(dependencies)

Bump dependency versions and add pnpm workspace onlyBuiltDependencies

- Bumped @push.rocks/smartenv from ^5.0.5 to ^5.0.13
- Bumped @git.zone/tsbuild from ^2.6.4 to ^2.6.8
- Bumped @git.zone/tstest from ^2.3.1 to ^2.3.7
- Added pnpm-workspace.yaml with onlyBuiltDependencies: [esbuild, mongodb-memory-server, puppeteer]

2025-06-01 - 7.5.0 - feat(dnssec)

Add MX record DNSSEC support for proper serialization and authentication of mail exchange records

- Serialize MX records by combining a 16-bit preference with the exchange domain name
- Enable DNSSEC signature generation for MX records to authenticate mail exchange data
- Update documentation to include the new MX record DNSSEC support in version v7.4.8

2025-05-30 - 7.4.7 - fix(dnsserver)

Update documentation to clarify the primaryNameserver option and SOA record behavior in the DNS server. The changes detail how the primaryNameserver configuration customizes the SOA mname, ensures proper DNSSEC signing for RRsets, and updates the configuration interface examples.

- Documented the primaryNameserver option in IDnsServerOptions with default behavior (ns1.{dnssecZone})
- Clarified SOA record generation including mname, rname, serial, and TTL fields
- Updated readme examples to demonstrate binding interfaces and proper DNS server configuration

2025-05-30 - 7.4.6 - docs(readme)

Document the primaryNameserver option and SOA record behavior in the DNS server documentation.

- Added comprehensive documentation for the primaryNameserver option in IDnsServerOptions
- Explained SOA record automatic generation and the role of the primary nameserver
- Clarified that only one nameserver is designated as primary in SOA records
- Updated the configuration options interface documentation with all available options

2025-05-30 - 7.4.3 - fix(dnsserver)

Fix DNSSEC RRset signing, SOA record timeout issues, and add configurable primary nameserver support.

- Fixed DNSSEC to sign entire RRsets together instead of individual records (one RRSIG per record type)
- Fixed SOA record serialization by implementing proper wire format encoding in serializeRData method
- Fixed RRSIG generation by using correct field names (signersName) and types (string typeCovered)
- Added configurable primary nameserver via primaryNameserver option in IDnsServerOptions
- Enhanced test coverage with comprehensive SOA and DNSSEC test scenarios

2025-05-30 - 7.4.2 - fix(dnsserver)

Enable multiple DNS record support by removing the premature break in processDnsRequest. Now the DNS server aggregates answers from all matching handlers for NS, A, and TXT records, and improves NS record serialization for DNSSEC.

- Removed the break statement in processDnsRequest to allow all matching handlers to contribute responses.
- Updated NS record serialization to properly handle domain names in DNSSEC context.
- Enhanced tests for round-robin A records and multiple TXT records scenarios.

2025-05-28 - 7.4.1 - fix(test/server)

Fix force cleanup in DNS server tests by casting server properties before closing sockets

- Cast server to any to safely invoke close() on httpsServer and udpServer in test cleanup
- Ensures proper emergency cleanup of server sockets without direct access to private properties

2025-05-28 - 7.4.0 - feat(manual socket handling)

Add comprehensive manual socket handling documentation for advanced DNS server use cases

- Introduced detailed examples for configuring manual UDP and HTTPS socket handling
- Provided sample code for load balancing, clustering, custom transport protocols, and multi-interface binding
- Updated performance and best practices sections to reflect manual socket handling benefits

2025-05-28 - 7.3.0 - feat(dnsserver)

Add manual socket mode support to enable external socket control for the DNS server.

- Introduced new manualUdpMode and manualHttpsMode options in the server options interface.

- Added initializeServers, initializeUdpServer, and initializeHttpsServer methods for manual socket initialization.
- Updated start() and stop() methods to handle both automatic and manual socket binding modes.
- Enhanced UDP and HTTPS socket error handling and IP address validations.
- Removed obsolete internal documentation file (readme.plan2.md).

2025-05-28 - 7.2.0 - feat(dns-server)

Improve DNS server interface binding by adding explicit IP validation, configurable UDP/HTTPS binding, and enhanced logging.

- Added udpBindInterface and httpsBindInterface options to IDnsServerOptions
- Implemented IP address validation for both IPv4 and IPv6 in the start() method
- Configured UDP and HTTPS servers to bind to specified interfaces with detailed logging
- Updated documentation to include interface binding examples (localhost and specific interfaces)
- Enhanced tests to cover valid and invalid interface binding scenarios

2025-05-27 - 7.1.0 - feat(docs)

Improve documentation for advanced DNS features and update usage examples for both DNS client and server.

- Revamped readme.hints with expanded architecture overview and detailed explanations of DNSSEC, Let's Encrypt integration, and advanced handler patterns.
- Updated readme.md with clearer instructions and code examples for A, AAAA, TXT, MX record queries, DNS propagation checks, and usage of UDP and DNS-over-HTTPS.
- Enhanced TAP tests documentation demonstrating both client and server flows.
- Bumped version from 7.0.2 to 7.1.0 in preparation for the next release.

2025-05-27 - 7.1.0 - feat(docs)

Improve documentation for advanced DNS features by updating usage examples for DNS client and server, and enhancing instructions for DNSSEC and Let's Encrypt integration.

- Revamped readme.hints with an expanded architecture overview and detailed client/server feature explanations.
- Updated readme.md to include clearer instructions and code examples for A, AAAA, TXT, MX record queries and DNS propagation checks.
- Enhanced examples for using DNSSEC, including detailed examples for DNSKEY, DS, and RRSIG records.
- Added new instructions for setting up DNS-over-HTTPS (DoH), UDP-based resolution, and pattern-based routing for handlers.
- Improved testing documentation with updated TAP tests demonstrating both DNS client and server flows.

2025-05-27 - 7.0.2 - fix(dns-client)

Improve test assertions for DNS record queries and correct counter increment logic in DNS client

- Updated test cases in test/test.client.ts to use dynamic assertions with 'google.com' instead of fixed values
- Adjusted checkUntilAvailable tests to verify proper behavior when DNS TXT record is missing
- Fixed counter increment in ts_client/classes.dnsclient.ts to avoid post-increment issues, ensuring proper retry delays

2025-05-27 - 7.0.1 - fix(test & plugins)

Rename test client variable and export smartrequest in client plugins

- Renamed variable 'testDnsly' to 'testDnsClient' in test/test.client.ts for better clarity.
- Added @push.rocks/smartrequest dependency in package.json and updated ts_client/plugins.ts to export it.

2025-05-27 - 7.0.0 - BREAKING CHANGE(core)

Refactor module entry point and update plugin imports; remove deprecated dnsly.plugins, update dependency versions, and adjust test imports

- Changed module export in package.json from './dist_ts_server/index.js' to './dist_ts/index.js'
- Updated dependency versions, notably upgrading '@tsclass/tsclass' from 5.0.0 to 9.2.0 and updating tap bundles to '@git.zone/tstest' packages
- Removed the redundant ts_client/dnsly.plugins.ts file and replaced its usage with the updated ts_client/plugins.ts
- Adjusted test files to use export default tap.start() and updated import paths for tap bundles
- Added tspublish.json files in ts, ts_client, and ts_server directories to control publish order

2025-03-21 - 6.3.0 - feat(dns-server)

Enhance DNS server functionality with advanced DNSSEC signing (supporting both ECDSA and ED25519), improved SSL certificate retrieval using Let's Encrypt, and refined handler management for cleaner shutdowns.

- Updated package metadata with expanded keywords and revised dependency versions
- Improved DNSSEC signing logic to support both ECDSA and ED25519 algorithms
- Added unregisterHandler method for cleaner handler lifecycle management
- Enhanced SSL certificate retrieval workflow with better DNS challenge handling
- Refined test utilities for more robust DNS operations

2025-03-21 - 6.3.0 - feat(dns-server)

Enhance DNS server functionality with advanced DNSSEC signing (including ED25519 support), improved certificate retrieval using Let's Encrypt, updated package metadata, and refined test utilities for more robust DNS operations.

- Updated package.json and npmextra.json with expanded keywords and revised dependency versions
- Improved DNSSEC signing logic to support both ECDSA and ED25519 algorithms
- Added unregisterHandler method and enhanced server stop logic for cleaner shutdowns
- Enhanced SSL certificate retrieval workflow with better challenge handling and test support

2024-09-21 - 6.2.1 - fix(core)

Fixing issues with keywords and readme formatting.

- Synchronized keywords field between npmextra.json and package.json.
- Updated readme.md to fix formatting issues and added new sections.

2024-09-19 - 6.2.0 - feat(dnssec)

Introduced DNSSEC support with ECDSA algorithm

- Added `DnsSec` class for handling DNSSEC operations.
- Updated `DnsServer` to support DNSSEC with ECDSA.
- Shifted DNS-related helper functions to `DnsServer` class.
- Integrated parsing and handling of DNSKEY and RRSIG records in `DnsServer`.

2024-09-19 - 6.1.1 - fix(ts_server)

Update DnsSec class to fully implement key generation and DNSKEY record creation.

- Added complete support for ECDSA and ED25519 algorithms in the DnsSec class.
- Implemented DNSKEY generation and KeyTag computation methods.
- Improved error handling and initialized the appropriate cryptographic instances based on the algorithm.

2024-09-18 - 6.1.0 - feat(smartdns)

Add DNS Server and DNSSEC tools with comprehensive unit tests

- Updated package dependencies to the latest versions
- Introduced DnsServer class for handling DNS requests over both HTTPS and UDP with support for custom handlers
- Added DnsSec class for generating and managing DNSSEC keys and DS records
- Implemented unit tests for DnsServer and Smartdns classes

2024-06-02 - 6.0.0 - server/client

Main description here

- **Breaking Change:** Move from client only to server + client exports.

2024-03-30 - 5.0.4 - maintenance

Range contains relevant changes

- Switch to new org scheme

2023-04-08 - 5.0.4 - core

Main description here

- Core update
- Fixes applied to the system

2022-07-27 - 5.0.0 - core

Update contains relevant changes

- **Breaking Change:** Major update and core changes
- Fixes and updates applied

2022-07-27 - 4.0.11 - core

Range contains relevant changes

- **Breaking Change:** Core update and changes applied

2021-08-24 - 4.0.10 - core

Range contains relevant changes

- Fixes applied to the core functionalities

2021-01-23 - 4.0.8 - core

Range contains relevant changes

- Updates and fixes to the core components

2020-08-05 - 4.0.4 - core

Range contains relevant changes

- Multiple core fixes applied

2020-02-15 - 4.0.0 - core

Main description here

- Core updates
- Fixes applied across the system

2020-02-15 - 3.0.8 - core

Core updates with major changes

- **Breaking Change:** Now uses Google DNS HTTPS API and handles DNSSEC validation

2019-01-07 - 3.0.6 - core

Range contains relevant changes

- Fixes and updates applied to the core

2018-05-13 - 3.0.4 - core

Range contains relevant changes

- Fixes applied, including `fix .checkUntilAvailable` error

2018-05-13 - 3.0.0 - ci

Main description here

- CI changes and updates to the access level and global packages

2017-07-31 - 2.0.10 - package

Update to new package name and improved record retrieval

- **Breaking Change:** Package name update and record retrieval improvements

2017-01-27 - 2.0.1 - maintenance

Multiple fixes and merges

2017-01-27 - 2.0.0 - core

Fix typings and update to better API

2016-11-15 - 1.0.7 - initial

Initial setup and improvements

- Initial deployment
- README improvements