

DNS Operations  
Internet-Draft  
Intended status: Standards Track  
Expires: 21 July 2026

A. Ferro  
ApertoDNS  
17 January 2026

ApertoDNS Protocol: A Modern Dynamic DNS Update Protocol  
draft-ferro-dnsop-apertodns-protocol-01

## Abstract

This document specifies the ApertoDNS Protocol, a modern RESTful protocol for dynamic DNS (DDNS) updates. It provides a secure, provider-agnostic alternative to legacy protocols, with native support for IPv4, IPv6, bulk updates, automatic IP detection, TXT record management for ACME DNS-01 challenges, and standardized authentication mechanisms.

The protocol uses well-known URIs (RFC 8615), JSON payloads (RFC 8259), and bearer token authentication (RFC 6750) to enable interoperable dynamic DNS services across different providers.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 July 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights

and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Protocol Versioning . . . . .	4
1.2. Requirements Language . . . . .	4
1.3. Goals . . . . .	4
2. Terminology . . . . .	5
3. Protocol Overview . . . . .	5
3.1. Base URL . . . . .	6
3.2. Content Type . . . . .	6
3.3. Response Format . . . . .	6
4. Conformance Requirements . . . . .	6
4.1. Conformance Levels . . . . .	6
4.2. Capability Advertisement . . . . .	7
4.3. Interoperability . . . . .	7
5. Authentication . . . . .	7
5.1. Supported Methods . . . . .	7
5.2. Token Format . . . . .	7
5.3. Token Transmission . . . . .	8
6. Endpoints . . . . .	8
6.1. Discovery Endpoint (/info) . . . . .	8
6.1.1. Response Fields . . . . .	8
6.1.2. Capability Fields . . . . .	9
6.1.3. Example Response . . . . .	10
6.2. Health Endpoint (/health) . . . . .	11
6.2.1. Example Response . . . . .	11
6.3. Update Endpoint (/update) . . . . .	11
6.3.1. Request Fields . . . . .	11
6.3.2. Auto-Detection Limitations . . . . .	12
6.3.3. Example Request . . . . .	12
6.3.4. Example Response . . . . .	12
6.3.5. Auto-Detection Failure Response . . . . .	13
6.4. Bulk Update Endpoint (/bulk-update) . . . . .	13
6.4.1. Example Request . . . . .	13
6.4.2. Example Response . . . . .	13
6.5. Status Endpoint (/status/{hostname}) . . . . .	14
6.5.1. Example Response . . . . .	14
6.6. Domains Endpoint (/domains) . . . . .	14
6.6.1. Example Response . . . . .	15
6.7. TXT Record Endpoint (/txt) . . . . .	15
6.7.1. Design Rationale . . . . .	15
6.7.2. Set TXT Record (POST) . . . . .	15
6.7.3. Delete TXT Record (DELETE) . . . . .	16

6.7.4. Get TXT Records (GET)	17
7. Error Handling	18
7.1. HTTP Status Codes	18
7.2. Error Response Format	18
7.3. Standard Error Codes	19
7.4. Rate Limiting Headers	20
8. Legacy Compatibility	20
8.1. Legacy Response Codes	20
9. Comparison with RFC 2136	21
10. Security Considerations	21
10.1. Transport Security	21
10.2. Token Security	21
10.3. Hostname Validation	22
10.4. Rate Limiting	22
10.5. DNS Rebinding Prevention	22
10.6. Input Validation	22
10.7. TXT Record Abuse Prevention	22
10.8. Internationalized Domain Names	23
11. Privacy Considerations	23
11.1. Data Minimization	24
11.2. User Control	24
11.3. Traffic Analysis	24
11.4. Encryption	24
12. IANA Considerations	24
12.1. Well-Known URI Registration	24
13. References	25
13.1. Normative References	25
13.2. Informative References	26
Appendix A. Acknowledgments	26
Appendix B. Implementation Status	26
B.1. ApertoDNS	26
Appendix C. OpenAPI Specification	27
Appendix D. Example Update Flow	27
Appendix E. Changes from Legacy DDNS Protocols	27
Appendix F. Changes from -00	28
F.1. Version 1.2.3 Changes	28
F.2. Version 1.3.0 Changes (TXT Records)	28
Author's Address	29

## 1. Introduction

Dynamic DNS (DDNS) services allow users with dynamically assigned IP addresses to maintain a consistent hostname that automatically updates when their IP address changes. This capability is essential for home users, small businesses, and IoT devices that need to be reachable despite lacking static IP addresses.

While RFC 2136 [RFC2136] defines DNS UPDATE for programmatic DNS modifications, most consumer-facing DDNS services use simpler HTTP-based protocols. The de facto standard for consumer DDNS emerged organically without formal specification.

This lack of standardization has led to:

- \* Inconsistent implementations across providers
- \* Security vulnerabilities from ad-hoc designs
- \* Limited feature sets (e.g., no native IPv6 support)
- \* Vendor lock-in due to proprietary extensions
- \* No formal capability negotiation

This document specifies the ApertoDNS Protocol as a modern, secure, and fully interoperable alternative designed for the current Internet landscape.

### 1.1. Protocol Versioning

The protocol version specified in discovery responses (e.g., "1.3.0") refers to the semantic version of the protocol specification itself. This document represents the first IETF standardization of a protocol that has been in production use since 2024. The version number in the discovery endpoint reflects the feature set available, while the Internet-Draft version (e.g., "-01") tracks the IETF document revision process separately.

### 1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 1.3. Goals

The ApertoDNS Protocol is designed with the following goals:

- \* **\*Provider-agnostic\***: Any DDNS provider can implement this protocol using their own domain and branding
- \* **\*Secure by default\***: HTTPS required, bearer token authentication

- \* **\*Modern\***: JSON responses, proper HTTP semantics, native IPv6
- \* **\*Discoverable\***: Self-describing via discovery endpoint
- \* **\*Extensible\***: Capability negotiation allows future enhancements
- \* **\*Backward compatible\***: Optional legacy endpoint for existing clients

## 2. Terminology

This document uses the following terms:

**DDNS**: Dynamic DNS. A service that automatically updates DNS records when a client's IP address changes.

**Provider**: An organization or service implementing this protocol to offer DDNS services to users.

**Hostname**: A fully qualified domain name (FQDN) managed by the provider and associated with a user account.

**Token**: An authentication credential issued by the provider, used to authorize API requests.

**Auto-detection**: Server-side determination of the client's IP address from the incoming HTTP request, used when the client specifies "auto" as the IP value.

**Client**: Software or device that makes requests to a DDNS provider to update DNS records.

**A-label**: The ASCII-Compatible Encoding (ACE) form of an Internationalized Domain Name label, as defined in [RFC5891].

**TXT Record**: A DNS resource record type used to store arbitrary text data, commonly used for domain verification and ACME DNS-01 challenges.

**ACME DNS-01**: A challenge type defined in [RFC8555] where domain ownership is proven by provisioning a DNS TXT record.

## 3. Protocol Overview

The ApertoDNS Protocol is a RESTful API using JSON over HTTPS. All protocol endpoints are located under the well-known URI path `/.well-known/apertodns/v1/`.

### 3.1. Base URL

Conforming implementations MUST serve all endpoints under:

`https://{provider-domain}/.well-known/apertodns/v1/`

The use of well-known URIs [RFC8615] ensures consistent endpoint discovery across providers.

### 3.2. Content Type

All request and response bodies MUST use the `application/json` media type [RFC8259] unless otherwise specified.

### 3.3. Response Format

All responses MUST include a boolean success field at the top level:

```
{
  "success": true,
  "data": { ... }
}
```

Or for errors:

```
{
  "success": false,
  "error": {
    "code": "error_code",
    "message": "Human-readable description"
  }
}
```

## 4. Conformance Requirements

This section defines the requirements for conforming implementations.

### 4.1. Conformance Levels

This protocol defines two conformance levels:

**Core Conformance:** A conforming implementation MUST implement the following endpoints: `/info`, `/health`, and `/update`. A conforming implementation MUST support bearer token authentication. A conforming implementation MUST serve all endpoints over HTTPS.

**Full Conformance:** In addition to core conformance requirements, a

fully conforming implementation MUST implement: /bulk-update, /status/{hostname}, and /domains endpoints.

#### 4.2. Capability Advertisement

Implementations MUST accurately advertise their capabilities in the /info endpoint response. Implementations MUST NOT advertise capabilities they do not support.

#### 4.3. Interoperability

Implementations SHOULD accept requests from any conforming client. Implementations MUST NOT require proprietary extensions for basic DDNS functionality.

### 5. Authentication

#### 5.1. Supported Methods

Protected endpoints require authentication via one of the following methods:

1. \*Bearer Token\* (RECOMMENDED) [RFC6750]: Authorization: Bearer {token}
2. \*API Key Header\*: X-API-Key: {token}
3. \*HTTP Basic\* (legacy only): Authorization: Basic {credentials}

Implementations MUST support bearer token authentication. Implementations MAY support additional methods.

#### 5.2. Token Format

Tokens SHOULD follow the format:

{provider}\_{environment}\_{random}

Where:

- \* {provider}: Provider identifier (e.g., "apertodns", "example")
- \* {environment}: Token environment ("live", "test", "sandbox")
- \* {random}: Cryptographically secure random string (minimum 32 characters recommended)

Example: apertodns\_live\_Kj8mP2xL9nQ4wR7vY1zA3bC6dE0fG5hI

This format enables:

- \* Easy identification of token source during debugging
- \* Environment separation (production vs. testing)
- \* Consistent token handling across providers

### 5.3. Token Transmission

Tokens MUST be transmitted only in HTTP headers. Tokens MUST NOT appear in URLs, query parameters, or request bodies where they might be logged.

## 6. Endpoints

### 6.1. Discovery Endpoint (/info)

GET /.well-known/apertodns/v1/info

The discovery endpoint returns provider information, capabilities, and configuration. This endpoint MUST NOT require authentication.

#### 6.1.1. Response Fields

Field	Type	Required	Description
protocol	string	YES	MUST be "apertodns"
protocol_version	string	YES	Semantic version (e.g., "1.3.0")
provider	object	YES	Provider information
capabilities	object	YES	Supported features
authentication	object	YES	Supported auth methods
endpoints	object	YES	Available endpoint paths
rate_limits	object	NO	Rate limiting configuration
server_time	string	NO	Current server time



			(ISO 8601)	
+-----+	+-----+	+-----+	+-----+	+-----+

Table 1

### 6.1.2. Capability Fields

The capabilities object MUST include the following fields:

Field	Type	Description
ipv4	boolean	IPv4 address updates supported
ipv6	boolean	IPv6 address updates supported
auto_ip_detection	boolean	Automatic IP detection supported
bulk_update	boolean	Bulk update endpoint available
max_bulk_size	integer	Maximum hostnames per bulk request

Table 2

The capabilities object MAY include the following OPTIONAL fields:

Field	Type	Description
webhooks	boolean	Provider-specific webhook support available
txt_records	boolean	TXT record management supported
txt_max_records	integer	Maximum TXT records per hostname (default: 5)

Table 3

When webhooks is true, the provider offers webhook notifications for DNS update events such as IP address changes. The webhook API is implementation-specific and not standardized by this protocol version. Providers offering webhooks SHOULD document their webhook API separately.

When `txt_records` is true, the provider supports TXT record management via the `/txt` endpoint. This capability enables ACME DNS-01 challenges [RFC8555] for automated certificate issuance.

The capabilities object MAY include additional fields for future extensions. Unknown capability fields SHOULD be ignored by clients.

#### 6.1.3. Example Response

```
{
  "success": true,
  "data": {
    "protocol": "apertodns",
    "protocol_version": "1.3.0",
    "provider": {
      "name": "Example DDNS",
      "website": "https://example.com",
      "documentation": "https://example.com/docs",
      "support_email": "support@example.com"
    },
    "capabilities": {
      "ipv4": true,
      "ipv6": true,
      "auto_ip_detection": true,
      "bulk_update": true,
      "webhooks": true,
      "max_bulk_size": 100
    },
    "authentication": {
      "methods": ["bearer_token", "api_key_header"],
      "token_format": "{provider}_{environment}_{random}"
    },
    "endpoints": {
      "info": "/.well-known/apertodns/v1/info",
      "health": "/.well-known/apertodns/v1/health",
      "update": "/.well-known/apertodns/v1/update",
      "bulk_update": "/.well-known/apertodns/v1/bulk-update",
      "status": "/.well-known/apertodns/v1/status/{hostname}",
      "domains": "/.well-known/apertodns/v1/domains"
    },
    "rate_limits": {
      "update": {"requests": 60, "window_seconds": 60},
      "bulk_update": {"requests": 10, "window_seconds": 60}
    },
    "server_time": "2025-01-01T12:00:00.000Z"
  }
}
```

## 6.2. Health Endpoint (/health)

GET /.well-known/apertodns/v1/health

Returns service health status. This endpoint MUST NOT require authentication and SHOULD be used for monitoring.

### 6.2.1. Example Response

```
{
  "success": true,
  "data": {
    "status": "healthy",
    "timestamp": "2025-01-01T12:00:00.000Z"
  }
}
```

The status field MUST be one of: "healthy", "degraded", or "unhealthy".

## 6.3. Update Endpoint (/update)

POST /.well-known/apertodns/v1/update

Authorization: Bearer {token}

Content-Type: application/json

Updates DNS records for a single hostname. This endpoint MUST require authentication.

### 6.3.1. Request Fields

Field	Type	Required	Description
hostname	string	YES	Fully qualified domain name
ipv4	string	NO	IPv4 address or "auto"
ipv6	string	NO	IPv6 address or "auto"
ttl	integer	NO	Time to live in seconds (60-86400)

Table 4

At least one of ipv4 or ipv6 SHOULD be provided. If neither is provided, implementations SHOULD use auto-detection for IPv4.

The special value "auto" instructs the server to detect the client's IP address from the incoming request.

#### 6.3.2. Auto-Detection Limitations

Auto-detection is constrained by HTTP connection semantics: the server can only detect the address family used by the client's TCP connection. This is a fundamental characteristic of HTTP-based protocols, not a limitation specific to this specification.

The following constraints apply:

- \* If the client connects via IPv4: only IPv4 can be auto-detected
- \* If the client connects via IPv6: only IPv6 can be auto-detected
- \* Cross-family detection is not possible (e.g., detecting IPv4 address when connected via IPv6)

When a client requests auto-detection for an address family that does not match the connection's address family, the server **MUST** return an error with the appropriate error code (ipv4\_auto\_failed or ipv6\_auto\_failed).

Clients requiring dual-stack updates (both IPv4 and IPv6) **SHOULD** use one of the following approaches:

1. Provide explicit IP addresses for both fields
2. Make separate update requests over each address family
3. Use auto-detection for the matching address family and provide an explicit address for the other

#### 6.3.3. Example Request

```
{
  "hostname": "home.example.com",
  "ipv4": "auto",
  "ttl": 300
}
```

#### 6.3.4. Example Response

```
{
  "success": true,
  "data": {
    "hostname": "home.example.com",
    "ipv4": "203.0.113.50",
    "previous_ipv4": "203.0.113.49",
    "ttl": 300,
    "changed": true,
    "timestamp": "2025-01-01T12:00:00.000Z"
  }
}
```

The changed field indicates whether the IP address was actually modified (false if the new IP matches the existing record).

#### 6.3.5. Auto-Detection Failure Response

When auto-detection fails due to address family mismatch:

```
{
  "success": false,
  "error": {
    "code": "ipv4_auto_failed",
    "message": "Cannot auto-detect IPv4 from IPv6 connection"
  }
}
```

#### 6.4. Bulk Update Endpoint (/bulk-update)

```
POST /.well-known/apertodns/v1/bulk-update
Authorization: Bearer {token}
Content-Type: application/json
```

Updates multiple hostnames in a single request. Providers advertising `bulk_update: true` in capabilities MUST implement this endpoint.

##### 6.4.1. Example Request

```
{
  "updates": [
    {"hostname": "home.example.com", "ipv4": "auto"},
    {"hostname": "office.example.com", "ipv4": "203.0.113.51"}
  ]
}
```

##### 6.4.2. Example Response

```
{
  "success": true,
  "data": {
    "summary": {
      "total": 2,
      "successful": 2,
      "failed": 0
    },
    "results": [
      {
        "hostname": "home.example.com",
        "success": true,
        "ipv4": "203.0.113.50",
        "changed": true
      },
      {
        "hostname": "office.example.com",
        "success": true,
        "ipv4": "203.0.113.51",
        "changed": true
      }
    ]
  }
}
```

#### 6.5. Status Endpoint (/status/{hostname})

```
GET /.well-known/apertodns/v1/status/{hostname}
Authorization: Bearer {token}
```

Returns current DNS record status for a hostname.

##### 6.5.1. Example Response

```
{
  "success": true,
  "data": {
    "hostname": "home.example.com",
    "ipv4": "203.0.113.50",
    "ipv6": "2001:db8::1",
    "ttl": 300,
    "last_updated": "2025-01-01T12:00:00.000Z"
  }
}
```

#### 6.6. Domains Endpoint (/domains)

```
GET /.well-known/apertodns/v1/domains
Authorization: Bearer {token}
```

Returns list of domains and hostnames available to the authenticated user.

#### 6.6.1. Example Response

```
{
  "success": true,
  "data": {
    "domains": [
      {
        "domain": "example.com",
        "hostnames": ["home.example.com", "office.example.com"]
      }
    ]
  }
}
```

#### 6.7. TXT Record Endpoint (/txt)

The TXT record endpoint enables management of DNS TXT records, primarily for ACME DNS-01 challenges [RFC8555] used in automated certificate issuance. Providers advertising `txt_records: true` in capabilities MUST implement this endpoint.

##### 6.7.1. Design Rationale

TXT record management is designed to support wildcard certificate issuance, which requires multiple simultaneous TXT records during the ACME validation process. The protocol supports:

- \* **\*Accumulation\***: Multiple TXT values can coexist for the same hostname prefix (e.g., `_acme-challenge.example.com`)
- \* **\*Selective deletion\***: Individual TXT values can be removed without affecting others
- \* **\*Automatic cleanup\***: Providers MAY implement TTL-based expiration for TXT records

##### 6.7.2. Set TXT Record (POST)

```
POST /.well-known/apertodns/v1/txt
Authorization: Bearer {token}
Content-Type: application/json
```

Creates or adds a TXT record value for the specified hostname.  
 Multiple calls with different values MUST accumulate (not replace)  
 TXT records for the same hostname, up to the provider's limit.

#### 6.7.2.1. Request Fields

Field	Type	Required	Description
hostname	string	YES	FQDN for the TXT record
value	string	YES	TXT record value (max 255 chars)
ttl	integer	NO	Time to live in seconds (default: 60)

Table 5

#### 6.7.2.2. Example Request

```
{
  "hostname": "_acme-challenge.home.example.com",
  "value": "gfj9Xq...Rg85nM",
  "ttl": 60
}
```

#### 6.7.2.3. Example Response

```
{
  "success": true,
  "data": {
    "hostname": "_acme-challenge.home.example.com",
    "value": "gfj9Xq...Rg85nM",
    "ttl": 60,
    "record_count": 1,
    "timestamp": "2026-01-01T12:00:00.000Z"
  }
}
```

The `record_count` field indicates the total number of TXT values currently stored for this hostname after the operation.

#### 6.7.3. Delete TXT Record (DELETE)

```
DELETE /.well-known/apertodns/v1/txt
Authorization: Bearer {token}
Content-Type: application/json
```



Removes a specific TXT record value. If value is omitted, ALL TXT records for the hostname are removed.

#### 6.7.3.1. Request Fields

Field	Type	Required	Description
hostname	string	YES	FQDN of the TXT record
value	string	NO	Specific value to delete (omit to delete all)

Table 6

#### 6.7.3.2. Example Request (selective deletion)

```
{
  "hostname": "_acme-challenge.home.example.com",
  "value": "gfj9Xq...Rg85nM"
}
```

#### 6.7.3.3. Example Response

```
{
  "success": true,
  "data": {
    "hostname": "_acme-challenge.home.example.com",
    "deleted": true,
    "values_removed": 1,
    "remaining_count": 1,
    "timestamp": "2026-01-01T12:00:00.000Z"
  }
}
```

#### 6.7.4. Get TXT Records (GET)

```
GET /.well-known/apertodns/v1/txt/{hostname}
Authorization: Bearer {token}
```

Returns all TXT record values for a hostname.

#### 6.7.4.1. Example Response

```

{
  "success": true,
  "data": {
    "hostname": "_acme-challenge.home.example.com",
    "values": [
      "gfj9Xq...Rg85nM",
      "hK7pLm...Yt42xQ"
    ],
    "ttl": 60,
    "record_count": 2
  }
}

```

## 7. Error Handling

### 7.1. HTTP Status Codes

Implementations MUST use appropriate HTTP status codes as defined in [RFC9110]:

Status	Usage
200	Successful request
400	Invalid request (bad hostname, invalid IP)
401	Missing or invalid authentication
403	Not authorized for requested resource
404	Resource not found
429	Rate limit exceeded
500	Server error

Table 7

### 7.2. Error Response Format

```

{
  "success": false,
  "error": {
    "code": "error_code",
    "message": "Human-readable description"
  }
}

```

### 7.3. Standard Error Codes

Code	HTTP Status	Description
unauthorized	401	Missing authentication
invalid_token	401	Invalid or expired token
forbidden	403	Not authorized for resource
not_found	404	Hostname not found
rate_limited	429	Too many requests
invalid_hostname	400	Invalid hostname format
invalid_ip	400	Invalid IP address format
hostname_not_owned	403	User does not own hostname
ipv4_auto_failed	400	Cannot auto-detect IPv4 from IPv6 connection
ipv6_auto_failed	400	Cannot auto-detect IPv6 from IPv4 connection
validation_error	400	Request validation failed
invalid_ttl	400	TTL value out of acceptable range
txt_not_supported	400	Provider does not support TXT records
txt_limit_exceeded	400	Maximum TXT records per hostname exceeded
txt_invalid_name	400	TXT hostname must use allowed prefix

txt_value_too_long	400	TXT value exceeds 255 characters
--------------------	-----	----------------------------------

Table 8

#### 7.4. Rate Limiting Headers

When rate limiting is applied, responses SHOULD include:

- \* Retry-After: Seconds until rate limit resets
- \* X-RateLimit-Limit: Maximum requests per window
- \* X-RateLimit-Remaining: Remaining requests in window
- \* X-RateLimit-Reset: Unix timestamp when window resets

#### 8. Legacy Compatibility

For backward compatibility with existing DDNS clients, providers MAY implement:

```
GET /nic/update?hostname={hostname}&myip={ip}
Authorization: Basic {credentials}
```

##### 8.1. Legacy Response Codes

Responses MUST be plain text (not JSON):

Response	Meaning
good {ip}	Update successful
nochg {ip}	No change needed
badauth	Authentication failed
notfqdn	Invalid hostname
nohost	Hostname not found
abuse	Account blocked

Table 9

This endpoint is provided for compatibility only. New implementations SHOULD use the modern JSON endpoints.

## 9. Comparison with RFC 2136

RFC 2136 [RFC2136] defines DNS UPDATE, a protocol for dynamic updates to DNS zones. The ApertoDNS Protocol differs in several key aspects:

Aspect	RFC 2136	ApertoDNS Protocol
Transport	DNS (UDP/TCP)	HTTPS
Format	DNS wire format	JSON
Auth	TSIG/SIG(0)	Bearer tokens
Discovery	None	/info endpoint
IPv6	Supported	Native support
Bulk ops	Per-message	Dedicated endpoint

Table 10

The ApertoDNS Protocol is designed for consumer DDNS services where simplicity and HTTP integration are priorities, while RFC 2136 is suited for direct DNS zone manipulation.

## 10. Security Considerations

### 10.1. Transport Security

All endpoints MUST be served over HTTPS using TLS 1.2 or higher. Implementations MUST NOT support plaintext HTTP for any protocol endpoint.

Implementations SHOULD support TLS 1.3 and SHOULD disable older cipher suites known to be weak.

### 10.2. Token Security

- \* Tokens MUST be generated using cryptographically secure random number generators (CSPRNG)
- \* Tokens SHOULD have configurable expiration

- \* Providers SHOULD support token revocation
- \* Tokens MUST NOT be logged in server access logs
- \* Tokens MUST NOT appear in URLs or error messages

### 10.3. Hostname Validation

Before processing any update request, implementations MUST verify that the authenticated user owns or has permission to modify the requested hostname. Failure to validate ownership could allow unauthorized DNS modifications.

### 10.4. Rate Limiting

Providers SHOULD implement rate limiting to prevent:

- \* Brute-force token guessing
- \* Denial of service attacks
- \* Excessive DNS propagation load

Rate limits SHOULD be advertised in the discovery endpoint and communicated via response headers.

### 10.5. DNS Rebinding Prevention

Implementations MUST validate that IP addresses in update requests are not private, loopback, or link-local addresses unless explicitly configured to allow such addresses.

### 10.6. Input Validation

All user input MUST be validated:

- \* Hostnames MUST conform to DNS naming rules
- \* IP addresses MUST be valid IPv4 or IPv6 format
- \* TTL values MUST be within acceptable ranges
- \* TXT values MUST NOT exceed 255 characters

### 10.7. TXT Record Abuse Prevention

TXT records can potentially be abused for:

- \* **\*DNS tunneling\***: Encoding data in TXT records for covert communication
- \* **\*Data exfiltration\***: Using DNS queries to leak sensitive data
- \* **\*Resource exhaustion\***: Creating excessive TXT records

Implementations MUST implement safeguards:

- \* Limit TXT value length to 255 characters (DNS standard)
- \* Limit number of TXT records per hostname (default: 5)
- \* Restrict TXT hostnames to approved prefixes (e.g., `_acme-challenge.`)
- \* Implement rate limiting on TXT operations
- \* Consider automatic expiration of TXT records (recommended: 24 hours)

#### 10.8. Internationalized Domain Names

When handling Internationalized Domain Names (IDNs), the following requirements apply as specified in [RFC5891]:

- \* Clients SHOULD convert IDN hostnames to their A-label (ASCII Compatible Encoding) form before sending requests
- \* Servers MUST accept hostnames in A-label form
- \* Servers MAY accept hostnames in U-label (Unicode) form and convert them to A-labels internally
- \* Servers MUST store and return hostnames in a consistent form

For example, a client wishing to update the hostname "例え.example.com" SHOULD send the request with the A-label form "xn--r8jz45g.example.com".

Implementations that accept U-label input MUST perform IDNA2008 validation as specified in [RFC5891] before processing the request.

#### 11. Privacy Considerations

This section addresses privacy considerations as recommended by [RFC6973].

### 11.1. Data Minimization

Providers SHOULD minimize the collection and retention of personal data. Specifically:

- \* IP address history SHOULD have configurable retention periods
- \* Update timestamps MAY be retained for operational purposes
- \* Providers SHOULD document their data retention policies

### 11.2. User Control

Users SHOULD have mechanisms to:

- \* View their stored data
- \* Delete their accounts and associated data
- \* Export their data in a portable format

### 11.3. Traffic Analysis

DDNS updates inherently reveal:

- \* That a user's IP address has changed
- \* The timing of IP address changes
- \* The association between a hostname and IP address

Providers should be aware that this information could be used to track user behavior or network changes.

### 11.4. Encryption

All communications MUST be encrypted via HTTPS, preventing passive observation of update requests and tokens.

## 12. IANA Considerations

### 12.1. Well-Known URI Registration

This document requests registration of the following well-known URI suffix:

URI Suffix: apertodns



Change Controller: IETF

Specification Document: This document

Related Information: None

The well-known URI `/.well-known/apertodns/` is used as the base path for all protocol endpoints.

## 13. References

### 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <<https://www.rfc-editor.org/rfc/rfc5891>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<https://www.rfc-editor.org/rfc/rfc6750>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/rfc/rfc8259>>.
- [RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <<https://www.rfc-editor.org/rfc/rfc8555>>.
- [RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/rfc/rfc8615>>.

- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/rfc/rfc9110>>.

### 13.2. Informative References

- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<https://www.rfc-editor.org/rfc/rfc2136>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/rfc/rfc6973>>.

### Appendix A. Acknowledgments

Thanks to the dynamic DNS community for decades of service enabling home users and small businesses to maintain stable hostnames with dynamic IP addresses.

Thanks to the IETF DNSOP working group participants for their review and feedback on this specification.

### Appendix B. Implementation Status

Note to RFC Editor: Please remove this appendix before publication.

This section records the status of known implementations of the protocol defined by this specification.

#### B.1. ApertoDNS

Organization: ApertoDNS

Implementation: Reference implementation

Description: Full protocol support including all endpoints, bulk updates, webhooks, and legacy compatibility

Level of Maturity: Production

Coverage: Complete

Licensing: Proprietary service, open protocol

Contact: support@apertodns.com

URL: https://apertodns.com

## Appendix C. OpenAPI Specification

A complete OpenAPI 3.0.3 specification for this protocol is available at:

<https://github.com/apertodns/apertodns-protocol/blob/main/openapi.yaml>

This specification can be used to:

- \* Generate client libraries in various programming languages
- \* Create interactive API documentation
- \* Validate implementations for conformance

## Appendix D. Example Update Flow

The following illustrates a typical update flow:

1. Client discovers provider capabilities: ~~~ GET /.well-known/apertodns/v1/info ~~~
2. Client authenticates and requests update: ~~~ POST /.well-known/apertodns/v1/update Authorization: Bearer example\_live\_abcl23 Content-Type: application/json  
  
{"hostname": "home.example.com", "ipv4": "auto"} ~~~
3. Provider validates token and hostname ownership
4. Provider updates DNS record
5. Provider returns result: ~~~json { "success": true, "data": { "hostname": "home.example.com", "ipv4": "203.0.113.50", "changed": true } } ~~~
6. DNS propagates the new record

## Appendix E. Changes from Legacy DDNS Protocols

For implementers familiar with legacy HTTP-based DDNS protocols (commonly referred to as "dyndns2" in client implementations such as ddclient), key differences include:

- \* JSON responses instead of plain text
- \* Bearer token authentication instead of HTTP Basic
- \* Explicit capability negotiation via /info endpoint
- \* Dedicated endpoints for different operations
- \* Standardized error codes and response formats
- \* Native IPv6 support with separate fields
- \* Bulk update support for multiple hostnames
- \* Well-known URI path for consistent discovery

#### Appendix F. Changes from -00

This section summarizes changes from draft-ferro-dnsop-apertodns-protocol-00:

##### F.1. Version 1.2.3 Changes

- \* Added `ipv4_auto_failed` and `ipv6_auto_failed` error codes to Section 8.3 (Standard Error Codes)
- \* Added "Auto-Detection Limitations" subsection to Section 7.3 (Update Endpoint) documenting HTTP connection semantics constraints
- \* Added `validation_error` and `invalid_ttl` error codes
- \* Added example response for auto-detection failure
- \* Updated protocol version in examples from "1.2.0" to "1.3.0"
- \* Added acknowledgment of IETF DNSOP working group

##### F.2. Version 1.3.0 Changes (TXT Records)

- \* Added TXT record management endpoint (`/txt`) for ACME DNS-01 challenges
- \* Added `txt_records` and `txt_max_records` capability fields
- \* Added TXT-related error codes: `txt_not_supported`, `txt_limit_exceeded`, `txt_invalid_name`, `txt_value_too_long`

- \* Added "TXT Record Abuse Prevention" section to Security Considerations
- \* Added RFC 8555 (ACME) to normative references
- \* Added TXT Record and ACME DNS-01 terminology definitions
- \* Updated protocol version to 1.3.0

Author's Address

Andrea Ferro  
ApertoDNS  
Italy  
Email: support@apertodns.com