Stream:	Internet Engineering Task Force (IETF)			
RFC:	10063			
Category:	Standards Track			
Published:	November 2026			
ISSN:	2070-1721			
Authors:	S. Romijn	J. Snijders	E. Shryane	S. Konstantaras
	Reliably Coded	*BSD*	*RIPE NCC*	*AMS-IX*

RFC 10063
Near Real Time Mirroring (NRTM) Version 4

Abstract

This document specifies Near Real Time Mirroring version 4 (NRTMv4): a one-way synchronization protocol for Internet Routing Registry (IRR) records. NRTMv4 is based on files that are distributed over HTTPS. The protocol allows instances of IRR Database servers to mirror IRR records, specified in the Routing Policy Specification Language (RPSL), between each other.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc10063.

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
2. Overview 4
3. Terminology 5
   3.1. Requirements Language 6
4. Mirror Server Use 6
   4.1. Key Configuration 6
   4.2. Snapshot Initialization 7
   4.3. Publishing Updates 7
      4.3.1. Delta Files 7
      4.3.2. Snapshot Files 8
      4.3.3. Update Notification File 9
      4.3.4. Publication Policy Restrictions 9
5. Mirror Client Use 9
   5.1. Client Configuration 9
   5.2. Update Notification File Retrieval 9
   5.3. Initialization from Snapshot 9
   5.4. Processing Delta Files 10
   5.5. Error Recovery 11
   5.6. Signature and Staleness Verification 11
   5.7. Policy Restrictions 11
6. Update Notification File 12
   6.1. Purpose 12
   6.2. Cache Concerns 12
   6.3. Payload Format and Validation 12
   6.4. Encoding and Signature 14
7. Snapshot File 15
   7.1. Purpose 15
   7.2. Cache Concerns 15

7.3. File Format and Validation 15
8. Delta File 16
8.1. Purpose 16
8.2. Cache Concerns 16
8.3. File Format and Validation 16
9. Operational Considerations 18
9.1. Time 18
9.2. IRR Object Validation 18
9.3. Intermediate Mirror Instances 18
9.4. Reading from Local Files 19
9.5. Storage Considerations 19
9.6. Public Key Rotation 19
9.7. Scalability 20
9.8. NRTMv3 Compatibility 20
10. IANA Considerations 20
11. Security Considerations 20
12. References 21
12.1. Normative References 21
12.2. Informative References 22
Acknowledgments 23
Authors' Addresses 23

1. Introduction

The Internet Routing Registry (IRR) consists of several IRR Databases, each representing objects in the Routing Policy Specification Language (RPSL) [RFC2622]. About a dozen larger IRR Databases are well known and widely used, operated by different organizations such as Regional Internet Registries and some large network operators. IRR objects serve many purposes ranging from manual research by operators to automated network configuration and filtering.

Most of these well-known IRR Databases mirror IRR objects from some others so that queries run against these instances provide a comprehensive view. Some parties also mirror IRR Databases to private IRR server instances to reduce latency when replying to queries, analyze IRR objects, or other purposes.

NRTM version 4 (NRTMv4) is a protocol for IRR mirroring and is designed to address issues in existing IRR Database mirroring protocols. In NRTMv4, IRR Databases publish their records on an HTTPS [STD97] endpoint with periodic Snapshot Files and regular Delta Files. Signing allows integrity checks. By only generating files once and publishing them over HTTPS, scalability is improved. NRTMv4 borrows some concepts from the Resource Public Key Infrastructure (RPKI) Repository Delta Protocol [RFC8182], as there are overlaps between the two protocols.

Reliable and secure distribution of IRR data is important for global routing operations since network operators depend on IRR objects for route filtering and network configuration. Prior mirroring protocols lack integrity verification, have no formal specification, and scale poorly. NRTMv4 addresses these shortcomings by providing a formally specified, scalable, and integrity-protected mirroring protocol. Of earlier NRTM versions, none were formally specified, but NRTMv3 [NRTMv3] was widely deployed. Some comparisons are made in Section 11.

2. Overview

In NRTMv4, a mirror server is an instance of IRR Database software that has a database of IRR objects and makes them available on an HTTPS endpoint to allow mirroring by others. These objects can be retrieved by mirror clients, which then load them into their local storage.

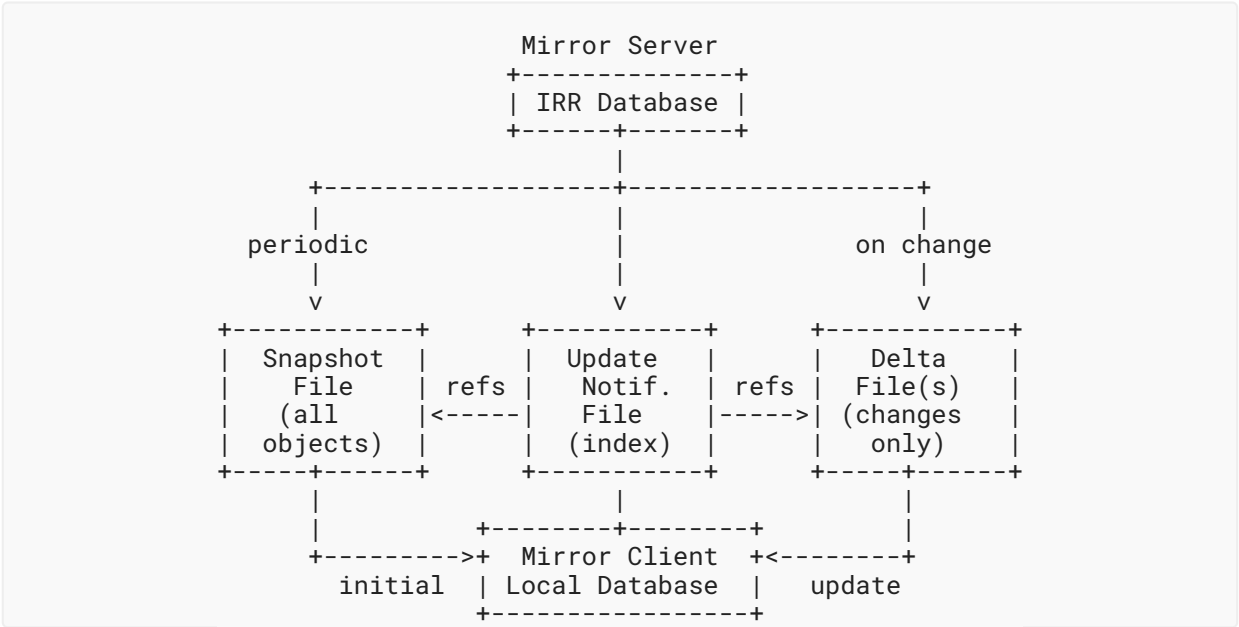


Figure 1: NRTMv4 Protocol Overview

Publication consists of three different files:

- A single Update Notification File. This specifies the current IRR Database version and locations of the Snapshot File and Delta Files. It is signed to allow verification of the authenticity of the Update Notification File.
- A single active Snapshot File. This contains all published IRR objects at a particular version. The mirror server periodically generates a new snapshot.
- Zero or more Delta Files. These contain the changes between two Database versions.

The Update Notification File **MUST** be in the JSON Web Signature [RFC7515] format, where the payload is in JSON format [RFC8259]. The Snapshot File and Delta Files **MUST** be in the JSON Text Sequences [RFC7464] format so that each object in large files can be parsed independently. Files **MAY** be compressed with GZIP [RFC1952].

Mirror clients initially retrieve the small Update Notification File and a Snapshot File from which they initialize their local copy of the Database. After that, mirror clients only retrieve the Update Notification File periodically, typically every minute, to determine whether there are any changes, and then retrieve only the relevant Delta Files, if any. This minimizes data transfer. Deltas have sequential versions.

Mirror clients are configured with the URL of an Update Notification File, the name of the IRR Database, and a public signing key. This public key is used to verify the Update Notification File, which, in turn, contains hashes of all the Snapshot and Delta Files.

Upon initialization, the mirror server generates a session identifier (session_id) for the Database (Section 4.2). This allows long-term caching and is used by the client to determine that the Delta Files continue to form a full set of changes allowing an update to the latest version. If the mirror server loses partial history or the mirror client starts mirroring from a different server, the session_id change will force a full reload from the latest Snapshot File, ensuring there are no accidental mirroring gaps.

Mirror servers can use caching to reduce their load, particularly because snapshots and deltas are immutable for a given session_id and version number. These are also the largest files. Update Notification Files may not be cached for longer than one minute, but are fairly small.

Note that in NRTMv4, contiguous version numbers are used for the Database version and Delta Files. This is different and unrelated to the serial in NRTMv3. NRTMv3 serials refer to a single change to a single object, whereas an NRTMv4 version refers to one delta, possibly containing multiple changes to multiple objects. NRTMv3 serials can also contain gaps, NRTMv4 versions do not.

3. Terminology

IRR Database: A logical collection of IRR objects represented in RPSL, grouped by the value in their "source" attribute. That value is the name of the IRR Database. No particular storage technology is implied.

Snapshot File: A file that reflects the complete contents of the IRR objects in an IRR Database at a particular point in time.

Delta File: A file that contains all changes for exactly one incremental update of the IRR Database.

Update Notification File: A file generated by the mirror server and used by mirror clients to discover which changes exist between the state of the IRR mirror server and of the mirror client.

Mirror Server: An instance of IRR Database software that has a database of IRR objects and makes them available for retrieval on an HTTPS endpoint, to allow mirroring by others.

Mirror Client: An instance of IRR Database software that retrieves IRR objects from a mirror server and loads them into its local storage.

Version: A monotonically increasing positive integer, scoped to a session_id, that identifies a particular state of the IRR Database.

3.1. 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.

4. Mirror Server Use

4.1. Key Configuration

When enabling NRTMv4 publication for an IRR Database, the operator **MUST** generate a JSON Web Key [RFC7517] pair and configure the private key in the mirror server. The operator then provides the corresponding public key, the name of the IRR Database, and publication URL of the Update Notification File to any operators of mirror clients. The published public key **MUST** be encapsulated following Textual Encoding of Subject Public Key Info Section 13 of [RFC7468], sometimes referred to as "PEM encoding". The process for providing this is not in the scope of this protocol, but a typical case is publication on the operator's known website. Key rotation is described in Section 9.6.

It is **RECOMMENDED** that implementations provide easily accessible tools for operators to generate new signing keys to enter into their configuration and assist with key rotation. All configuration options **SHOULD** be clearly named to indicate that they are private keys. Clear naming reduces the risk of operator error causing accidental exposure.

4.2. Snapshot Initialization

A mirror server **MUST** follow the initialization steps upon the first export for an IRR Database by that mirror server, or if the server lost history and can not reliably produce a continuous set of deltas from a previous state.

In other words, either the mirror server guarantees that clients following the deltas have a correct and complete view, or the mirror server must reinitialize, which will force clients to reinitialize as well.

Initialization consists of these actions:

- The mirror server **MUST** generate a new session_id. This **MUST** be a Universally Unique Identifier version 4 (UUIDv4) (Section 5.4 of [RFC9562]) and **MUST** be the same across all client sessions. The session_id is unique to the IRR Database, so an instance that serves multiple IRR Databases will create a separate session_id for each.
- The server **MUST** generate a snapshot for version number one. This may contain an empty array of objects if the IRR Database is currently empty.
- The server **MUST** generate a new Update Notification File with the new session_id, a reference to the new snapshot, and no deltas.

Note that a publication, and its associated session_ids and versions, always relates to a single specific IRR Database, even if multiple databases are published from one instance. For example, a mirror server that serves two IRR Databases named EXAMPLE and EXAMPLE-NONAUTH will generate two Update Notification Files, referring to two Snapshot Files and two sets of Delta Files. Each set of Delta Files has contiguous version numbers. The two publications are completely independent of each other, with different session IDs, and may be published at different times. This applies even if the same IRR server instance produces both.

4.3. Publishing Updates

After creating the initialization files, the mirror server processes updates by generating Delta Files and, periodically, a new Snapshot File, and placing them on the HTTPS endpoint.

4.3.1. Delta Files

Changes to IRR objects **MUST** be recorded in Delta Files. One Delta File can contain multiple changes.

Updates are generated as follows:

- A mirror server **MUST** publish a Delta File approximately every minute if there have been changes to IRR objects in that time frame.
- If a mirror server is lagging in production of Delta Files, such as after an initialization or server downtime, it **MUST** generate one larger "catch up" Delta File rather than individual Delta Files for every one-minute window.

- A new Delta File **MUST** be generated with a new version, one greater than the last Delta File version, or one greater than the last Snapshot File version if there were no prior deltas at all.
- The Delta File **MUST** include all changes that happened during the time frame in the order in which they occurred. If multiple changes have occurred within the time frame that would cancel each other out, like an addition and immediate deletion of the same object, the mirror server **MUST** still include all these changes.
- The URL where the Delta File is published **MUST** contain the session ID and version number so that the content at a given URL is immutable and can be cached indefinitely by HTTP intermediaries. It **MUST** also contain a random value that can not be predicted before publication to counter negative caching issues.
- The Update Notification File **MUST** be updated to include the new Delta File.
- References to Delta Files older than 24 hours **SHOULD** be removed from the Update Notification File. This limits the potential length of the Delta chain that clients must process, where a Snapshot refresh may be more efficient, while also preventing overly frequent Snapshot reloads. However, a mirror server **MUST NOT** remove references to Delta Files with a version higher than the current Snapshot File version, as this would leave clients with no path from the snapshot to the current state. Refer also to Section 9.5.
- Note that, as Delta Files always contain changes compared to a previous state, there can never be a Delta File with version 1.

Mirror servers with high object churn are **RECOMMENDED** to generate new Snapshot Files more frequently than the required minimum to limit the length of the Delta chain that clients must traverse on initialization or after a gap. Long Delta chains increase initialization time and resource consumption for mirror clients.

4.3.2. Snapshot Files

Snapshot Files after initialization are generated as follows:

- If there have been no changes to the IRR objects since the last snapshot, the mirror server **MUST NOT** generate a new snapshot.
- The mirror server **MUST** generate a new Snapshot File at least once per day if there have been changes to the IRR objects. Mirror servers should not generate new Snapshot Files more than once per hour, as snapshot generation can be resource-intensive.
- The version number of the new snapshot **MUST** be equal to the last Delta File version at the time the server begins generating the snapshot.
- The URL where the Snapshot File is published **MUST** contain the session ID and version number so that the content at a given URL is immutable and can be cached indefinitely by HTTP intermediaries. It **MUST** also contain a random value that can not be predicted before publication to counter negative caching issues.
- The Update Notification File **MUST** be updated to include the new snapshot if one was generated.
- Snapshot generation may take some time, and in that time, newer changes may occur that are not part of the snapshot in progress. The mirror server **MUST** continue to produce Delta

Files during this window. This means that the server may publish a Snapshot File with a version number older than the most recent Delta File at the time of publication.

4.3.3. Update Notification File

The Update Notification File **MUST** be updated when a new Delta or Snapshot File is published and at least every 24 hours, even if there have been no changes. The full format and validation rules are specified in Section 6.

4.3.4. Publication Policy Restrictions

A mirror server **MAY** have a policy that restricts the publication of certain IRR objects or attributes, or modifies these before publication. Typical scenarios for this include preventing the distribution of certain personal data or password hashes, or excluding objects which do not meet validation rules like RPKI consistency. If objects are modified, it is **RECOMMENDED** that the mirror server makes it evident to humans reading the object text that the object was modified, for example, by adding remark lines or comments.

Mirror servers are **RECOMMENDED** to remove password hashes from the auth lines in mntner objects, as they have little use beyond the authoritative server and their publication may be a security risk.

If a mirror server has a policy that restricts or modifies object publication, this **MUST** be applied consistently to Snapshot Files and Delta Files from the moment the policy is enacted or modified.

5. Mirror Client Use

5.1. Client Configuration

Mirror clients are configured with the name of the IRR Database, the URL of the Update Notification File, and the public key currently used for signing the Update Notification File. Key rotation is described in Section 9.6.

5.2. Update Notification File Retrieval

Mirror clients may retrieve the Update Notification File to check for changes every minute. Mirror clients **MUST NOT** check more than once per minute.

5.3. Initialization from Snapshot

Clients **MUST** initialize from a Snapshot File when initially configured, or if they are not able to update their local data from the provided Delta Files:

- The client **MUST** retrieve the Update Notification File.
- The client **MUST** verify that the source attribute in the Update Notification File matches the configured IRR Database name.
- The client **MUST** retrieve the Snapshot File and load the objects into its local storage.

- The mirror client **MUST** verify that the calculated SHA-256 [SHS] hash of the Snapshot File matches the hash in the Update Notification File that referenced it. If the Snapshot File was compressed with GZIP, the hash **MUST** match the compressed data. As the Update Notification File is signed, this hash verification proves the integrity and authenticity for the Snapshot File. In case of a mismatch of this hash, the file **MUST** be rejected.
- The client **MUST** record the session_id and version of the loaded Snapshot File.
- The file **MUST** be rejected if an Update Notification File or a Snapshot File has an invalid syntax (e.g., missing mandatory parameters). For validating the IRR objects contained in the files, see Section 9.2.

5.4. Processing Delta Files

If a mirror client has previously initialized from a snapshot:

- The client **MUST** retrieve the Update Notification File.
- The client **MUST** verify that the source attribute in the Update Notification File matches the configured IRR Database name.
- The client **MUST** verify that the session_id matches the previously known session_id. If this does not match, the client **MUST** reinitialize from the snapshot.
- The client **MUST** verify that the Update Notification File version is the same or higher than the client's current most recent version. If not, the Update Notification File **MUST** be rejected. It is **RECOMMENDED** for the client to distinguish between an Update Notification File that is a single version older, and a much older version, in any status messages. The former can occur from time to time in synchronization issues and the latter is more likely a faulty implementation.
- The client **MUST** verify that the Update Notification File contains one contiguous set of Delta File versions after the client's current most recent version up to the latest version in the Update Notification File. If the Delta File versions are not contiguous, the Update Notification File **MUST** be rejected. If the available Delta File versions do not range from the client's most recent version plus one, the client **MUST** reinitialize from the snapshot.
- The mirror client **MUST** verify that the hashes of each Delta and Snapshot File have not changed compared to the previous valid Update Notification File for the same file type and version. If a newer Update Notification File contains a different hash for a specific file, this indicates a misconfiguration in the server and the client **MUST** reject the Update Notification File.
- The client **MUST** retrieve all Delta Files for versions since the client's last known version, if there are any.
- The mirror client **MUST** verify that the calculated SHA-256 hash of each newly downloaded Delta File matches the hash in the Update Notification File that referenced it. If the Delta File was compressed with GZIP, the hash **MUST** match the compressed file. As the Update Notification File is signed, this hash verification proves the integrity and authenticity for the Delta File. In case of a mismatch of this hash, the Delta File **MUST** be rejected.
- The client **MUST** process all changes in the Delta Files in order: lowest Delta File version number first, and in the order of the changes list in the Delta File.

- The client **MUST** update its records of the most recent version to the version of the Update Notification File.
- If an Update Notification File or a Delta File has an invalid syntax (including missing mandatory parameters), the file **MUST** be rejected. For validating the IRR objects contained in the files, see Section 9.2.

If the Update Notification File or one of the Delta Files is rejected, the mirror client **MUST NOT** process any newer Deltas than those that are valid and have been successfully verified. If some Delta Files are rejected, it **MAY** process the valid Delta Files, but **MUST NOT** skip over any rejected Delta Files while doing so. Additionally, the changes in a specific Delta File **MUST** be processed either completely, or not at all, i.e., a Delta File must never be partially processed.

5.5. Error Recovery

When a mirror client fails to retrieve any file due to a transient error, such as a network failure or an HTTP 5xx response, the client **SHOULD** retry before concluding the retrieval has failed. It is **RECOMMENDED** that implementations use a bounded exponential backoff strategy, starting from a short initial interval (e.g., a few seconds) up to a capped maximum interval (e.g., several minutes), with a bounded total retry duration. This prevents unnecessary Snapshot imports on clients and can prevent clients overwhelming a server on transient failures. Mirror clients **SHOULD** log retry and recovery events, including the reason, to assist operators in diagnosing mirroring failures.

If a file is rejected, the mirror client **MUST NOT** load any data from it. If the error may be transient, the mirror client **SHOULD** retry. After exhausting retries, if the required Delta Files remain unavailable or are still rejected, the mirror client **SHOULD** reinitialize from the Snapshot File to prevent using stale data. If the Snapshot File itself cannot be retrieved or is rejected after retries, as there is no further fallback, the client **SHOULD** alert the operator and stop until the issue is resolved.

5.6. Signature and Staleness Verification

Every time a mirror client retrieves a new version of the Update Notification File, it **MUST** verify the included signature. The signature **MUST** be valid for the configured public key for the contents of the Update Notification File. If the signature does not match, the mirror client **MUST** reject the Update Notification File unless a key rotation is in progress as described in Section 9.6.

A mirror client can use the generation timestamp in the Update Notification File to check whether the file is stale, as the mirror server must update this file at least every 24 hours. If the generation timestamp is more than 24 hours ago, the file is stale and the mirror client **SHOULD** warn the operator in log messages or other alerting, but **MAY** continue to process it otherwise.

5.7. Policy Restrictions

A mirror client **MAY** have a policy that restricts the processing of objects to certain object classes, or other limitations on which objects it processes.

If a mirror client has a policy that restricts object processing, this **MUST** be applied consistently to Snapshot Files and Delta Files from the moment the policy is enacted or modified.

6. Update Notification File

6.1. Purpose

The Update Notification File is generated by the mirror server and used by mirror clients to discover whether any changes exist between the state of the IRR mirror server and of the mirror client. It also describes the location of the Snapshot File and incremental Delta Files. Finally, the generation timestamp can be used to detect whether the file is stale.

The mirror server **MUST** generate a new Update Notification File every time there are new deltas or snapshots and at least every 24 hours, even if there have been no changes.

6.2. Cache Concerns

A mirror server may use caching infrastructure to cache the Update Notification File and reduce the load of HTTPS requests.

However, since this file is used by mirror clients to determine whether any updates are available, the mirror server **SHOULD** ensure that this file is not cached for longer than one minute. An exception to this rule is that it is better to serve a stale Update Notification File rather than no Update Notification File.

Mirror servers **SHOULD** set the HTTP `Cache-Control` response header on the Update Notification File to `no-cache` or `max-age=60`, consistent with the one-minute caching limit. Mirror servers **SHOULD** set `Cache-Control: immutable` on responses for Snapshot Files and Delta Files, as these are immutable for a given session_id and version. Otherwise, clients and intermediaries may unnecessarily refetch large files. Mirror clients **MAY** use HTTP conditional requests (e.g., using `If-None-Match` with ETags or `If-Modified-Since`) when polling the Update Notification File to reduce bandwidth consumption when no changes have occurred.

6.3. Payload Format and Validation

An example payload of an Update Notification File is provided below:

```
{
  "nrtm_version": 4,
  "timestamp": "2025-12-01T15:00:00Z",
  "type": "notification",
  "next_signing_key": "bnJ0..bXY0",
  "source": "EXAMPLE",
  "session_id": "ca128382-78d9-41d1-8927-1ecef15275be",
  "version": 4,
  "snapshot": {
    "version": 3,
    "url":
"ca128382-78d9-41d1-8927-1ecef15275be/nrtm-snapshot.3.04759.json.gz",
    "hash": "9a..86"
  },
  "deltas": [
    {
      "version": 2,
      "url":
"ca128382-78d9-41d1-8927-1ecef15275be/nrtm-delta.2.784a2a6.json",
      "hash": "62..a2"
    },
    {
      "version": 3,
      "url":
"ca128382-78d9-41d1-8927-1ecef15275be/nrtm-delta.3.0f681f0.json",
      "hash": "25..9a"
    },
    {
      "version": 4,
      "url":
"ca128382-78d9-41d1-8927-1ecef15275be/nrtm-delta.4.d9c194a.json",
      "hash": "b4..13"
    }
  ],
  "metadata": {}
}
```

> Note: Hash and key values in this example are shortened because of formatting.

In this example, the overall version is 4, the snapshot is at version 3, and deltas are listed for versions 2, 3, and 4. The deltas at versions 2 and 3 predate the snapshot, but are still within the 24-hour retention window. A new client would initialize from the snapshot at version 3 and then apply only delta 4. An existing client already at version 1 can instead apply deltas 2, 3, and 4 without reinitializing from the snapshot.

The following validation rules **MUST** be observed when creating or parsing Update Notification Files:

- The nrtm_version **MUST** be 4.
- The timestamp **MUST** be in the format specified in [RFC3339] with the time-offset set to "Z".
- The type **MUST** be "notification".

- The optional field next_signing_key is used for in-band key rotation. If present, it **MUST** be a JWK [RFC7517] public key encoded in PEM, which matches the private key the mirror server will start using to sign the Update Notification File in the near future. Key rotation is described in Section 9.6. If there is no next signing key, this key **MUST** be omitted.
- The source **MUST** be a valid IRR object name [RFC2622].
- The session_id attribute **MUST** be a UUIDv4 (Section 5.4 of [RFC9562]) unique to this session for this source.
- The version **MUST** be an unsigned positive integer and be equal to the highest version of the deltas and snapshot.
- The file **MUST** contain exactly one snapshot.
- The file **MAY** contain one or more deltas.
- The deltas **MUST** have a sequential contiguous set of version numbers.
- Each snapshot and delta element **MUST** have a version, URL, and hash attribute. The URL must be relative to the path of the Update Notification File. For example, if the Update Notification File example above is published on https://example.com/nrtm/update-notification-file.jose, the full URL for the referred snapshot is https://example.com/nrtm/ca128382-78d9-41d1-8927-1ecef15275be/nrtm-snapshot.3.04759.json.gz. If the snapshot or delta file was compressed with GZIP, the filename **MUST** end in ".gz", and the hash **MUST** match the compressed data.
- The hash attribute in snapshot and delta elements **MUST** be the hexadecimal encoding of the SHA-256 hash of the referenced file. This hash algorithm, used to verify the integrity of the Snapshot and Delta Files, is fixed and independent of the JWS signing algorithm used for the Update Notification File. The mirror client **MUST** verify this hash when the file is retrieved and reject the file if the hash does not match.
- The metadata key **MAY** be present and is used for metadata produced by the server to aid in tracing and debugging. This can contain information like the name of the host on which the file was generated or the name and version of the software used. Each mirror server may choose which fields to include or choose to not include any metadata. The mirror server **SHOULD NOT** cause excessive size increases by adding extensive metadata in the Update Notification File, as it is the most frequently retrieved file.

6.4. Encoding and Signature

- The actual Update Notification File content is a JSON Web Signature [RFC7515] using JWS Compact Serialization.
- The JWS Payload is the JSON [RFC8259] serialization of the structure described in the previous section.
- The filename of the serialized data **MUST** be "update-notification-file.jose".
- Mirror clients **MUST** implement ES256. Mirror servers **MAY** use ES256 or other digital signature algorithms from the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA_jose]; the algorithm **MUST NOT** be "none", a MAC algorithm, or `Deprecated` or `Prohibited`. It is **RECOMMENDED** to use ES256 or other Recommended or Recommended+ algorithms. If a server uses an algorithm other than ES256, it is the server operator's responsibility to ensure their mirror clients support that algorithm.

7. Snapshot File

7.1. Purpose

The Snapshot File reflects the complete and current contents of all IRR objects in an IRR Database. Mirror clients **MUST** use this to initialize their local copy of the IRR Database. Snapshot Files are not individually signed. Their integrity is verified by comparing the calculated SHA-256 hash with the hash listed in the signed Update Notification File (see Section 6.4).

7.2. Cache Concerns

A snapshot reflects the content of the IRR Database at a specific point in time; for that reason, it can be considered immutable data. Snapshot Files are published at a URL that is unique to the specific session and version. The URL also contains a random value that can not be predicted before publication to counter negative caching issues.

Because these files never change, they can be cached. However, as snapshots are large and old snapshots will no longer be referred by newer Update Notification Files, it is **RECOMMENDED** that a limited cache interval is used.

7.3. File Format and Validation

Example Snapshot File, where ␞ (U+241E) represents the record separator control character (U+001E) as specified in [RFC7464]:

```
␞{
  "nrtm_version": 4,
  "type": "snapshot",
  "source": "EXAMPLE",
  "session_id": "ca128382-78d9-41d1-8927-1ecef15275be",
  "version": 3
}
␞{"object": "route: 192.0.2.0/24\norigin: AS64500\nsource: EXAMPLE"}
␞{"object": "route6: 2001:db8::/32\norigin: AS64500\nsource: EXAMPLE"}
```

> Note: IRR object texts in this example are shortened because of formatting.

The file is in JSON Text Sequences [RFC7464] format and **MUST** contain one or more records (it must contain at least the header). The first record is the file header and the following validation rules **MUST** be observed when creating or parsing a Snapshot File header:

- The nrtm_version **MUST** be 4.
- The type **MUST** be "snapshot".
- The source **MUST** match the source in the Update Notification File.
- The session_id attribute **MUST** match the session_id in the Update Notification File.

- The version **MUST** be an unsigned positive integer, matching the Update Notification File entry for this snapshot.

The remaining records (zero or more) **MUST** each contain a string representation of an IRR object. The source attribute in the IRR object texts **MUST** match the source attribute of the Snapshot File.

8. Delta File

8.1. Purpose

A Delta File contains all changes for exactly one incremental update of the IRR Database. It may include new, modified, and deleted objects. Delta Files can contain multiple alterations to multiple objects. Delta Files are not individually signed. Their integrity is verified by comparing the calculated SHA-256 hash with the hash listed in the signed Update Notification File (see Section 6.4).

8.2. Cache Concerns

Deltas reflect the difference in content of the IRR Database from one version to another; for that reason, it can be considered immutable data. Delta Files are published at a URL that is unique to the specific session and version. The URL also contains a random value that can not be predicted before publication to counter negative caching issues.

To avoid race conditions where a mirror client retrieves an Update Notification File moments before it's updated, mirror servers **SHOULD** retain old Delta Files for at least 5 minutes after a new Update Notification File is published that no longer contains these Delta Files.

8.3. File Format and Validation

Example Delta File, where ␞ (U+241E) represents the record separator control character (U+001E) as specified in [RFC7464]:

```
␞{
  "nrtm_version": 4,
  "type": "delta",
  "source": "EXAMPLE",
  "session_id": "ca128382-78d9-41d1-8927-1ecef15275be",
  "version": 3
}
␞{
  "action": "delete",
  "object_class": "person",
  "primary_key": "PRSN1-EXAMPLE"
}
␞{
  "action": "delete",
  "object_class": "route",
  "primary_key": "192.0.2.0/24AS64500"
}
␞{
  "action": "add_modify",
  "object": "route6: 2001:db8::/32\norigin: AS64500\nsource: EXAMPLE"
}
```

> Note: IRR object texts in this example are shortened because of formatting.

The file is in JSON Text Sequences [RFC7464] format and **MUST** contain two or more records (at least the header and one change). The first record is the file header and the following validation rules **MUST** be observed when creating or parsing a Delta File header:

- The nrtm_version **MUST** be 4.
- The type **MUST** be "delta".
- The source **MUST** match the source in the Update Notification File.
- The session_id attribute **MUST** match the session_id in the Update Notification File.
- The version **MUST** be an unsigned positive integer, matching the Update Notification File entry for this delta.

The remaining records (one or more) **MUST** each contain a JSON object that represents a change. The JSON object **MUST** meet the following rules:

- It has an "action" attribute, which is either "delete" for object deletions or "add_modify" for additions or modifications.
- If action is "delete": It has an "object_class" attribute with the RPSL object class name, and a "primary_key" attribute with the primary key of the deleted object. For objects that are listed in [RFC2622] and [RFC4012], the primary key is the value of the RPSL field defined as "class key". For object classes that define a pair of attributes as class key, e.g. route, the values of the individual attributes are appended together without separators. For any other objects, the primary key is the value of the RPSL field with the same name as the object class name. The primary key and object class name are not case-sensitive. Therefore, mirror clients **MUST** use case-insensitive matching against their local database.

- If action is "add_modify": It has an "object" attribute with the RPSL text of the new version of the object.

9. Operational Considerations

9.1. Time

IRR clients and servers are **RECOMMENDED** to use NTP [RFC5905] synchronization to avoid timing discrepancies.

9.2. IRR Object Validation

Throughout the years, various implementations of IRR servers have taken liberties with the various RFCs regarding RPSL. Implementations have introduced different new object classes, attributes, and validation rules. Current IRR Databases also contain legacy objects that were created under different validation rules. In practice, there is no uniformly implemented standard for RPSL, but merely rough outlines partially documented in different places.

This has the potential to create interoperability issues. Some are addressed by NRTMv4, like having a consistent character set when mirroring data between implementations. However, some issues can not be addressed in this way, such as one implementation introducing a new object class that is entirely unknown to another implementation.

A mirror client **SHOULD** be able to handle unknown object classes and objects that are invalid according to its own validation rules, which may mean simply discarding them without rejecting remaining objects or preventing future updates.

It is **RECOMMENDED** for mirror clients to log these cases, particularly those where an object was discarded due to violating validation rules. These cases create an inconsistency between the IRR objects of the server and client, and logs facilitate later analysis.

It is recommended for mirror clients to be flexible where possible and reasonable when applying their own validation rules to IRR objects retrieved from mirror servers. For example, a route object with an origin attribute that is not a valid AS number can't be usefully interpreted. There is no way for an IRR server to correctly parse and index such an object. However, a route-set object whose name does not start with "RS-" [RFC2622], or an inetnum with an unknown extra "org" attribute, still allows the mirror client to interpret it unambiguously even if it does not meet the mirror client's own validation rules for authoritative records.

9.3. Intermediate Mirror Instances

An IRR Database generally has a single authoritative source. In some cases, an instance run by a third party will function as a kind of intermediate: both being a mirror client, mirroring IRR objects from the authoritative source, and simultaneously functioning as a mirror server to yet another mirror client.

There are various operational reasons for such a setup, such as the intermediate filtering certain records. Regardless of the reason, the mirror client and server function of an IRR server must be treated as separate processes. In particular, this means they **MUST** have separate session IDs. The intermediate server **MUST NOT** republish the same files it retrieved from the authoritative source with the same session ID.

9.4. Reading from Local Files

In the typical use case for NRTMv4, a mirror client retrieves files from an HTTPS endpoint. However, implementations **MAY** also support reading from files on the local filesystem instead for when operators want to use a different method to retrieve or distribute the files. When reading from local files, mirror clients **SHOULD** still follow all validation rules, including the validation of the signature and hashes, to prevent corrupted or tampered data from being loaded.

9.5. Storage Considerations

After publishing a new Update Notification File and absent local policy, a mirror server **SHOULD** remove any Snapshot and Delta Files that are no longer referenced, as unreferenced files consume storage unnecessarily. To avoid race conditions where a mirror client retrieves an Update Notification File moments before it is updated, mirror servers **SHOULD** retain unreferenced files for at least 5 minutes after a new Update Notification File is published.

9.6. Public Key Rotation

It is recommended that the IRR Database operators rotate the signing key on their mirror server with a frequency that is not disruptive to operations but preserves the familiarity of the practices to accomplish key rotation. Many organizations have settled on annual cycles. The next_signing_key field in the Update Notification File supports in-band key rotation using the following process:

- The server operator generates a new key and configures this in the mirror server implementation as the upcoming new signing key.
- The mirror server **MUST** include this key in the next_signing_key field in any Update Notification File generated while the new signing key is configured. Hence, the new signing key will start being propagated to the mirror clients with the next publication of the Notification File, which will take at most 24 hours. Mirror server implementations **MAY** offer a method to cause the Update Notification File to be refreshed earlier, with the next_signing_key included, and thus start the propagation earlier.
- When mirror clients retrieve the Update Notification File next, they **MUST** detect the next_signing_key field and store the key in their configuration.
- After allowing mirror clients time to have seen the new Update Notification File with the next_signing_key field, the mirror server operator configures the new key as the currently active key and removes the old key. Any Update Notification File generated after this point **MUST** be signed with this new key and will not contain a next_signing_key field.

- The **RECOMMENDED** period between publication of the upcoming key in the next_signing_key field and removal of the old key is one week. This offers all active clients a reasonable chance to follow the rotation process.
- When mirror clients retrieve an Update Notification File and find that the signature does not match, they **MUST** attempt to verify against a next_signing_key encountered in a previous (valid) file. If the signature matches for this new key, the client **MUST** update its configuration to use the new key for validation. After this, the client **MUST NOT** use the old key for validation at any time; a mirror server cannot switch back to an old key.

If a mirror client never retrieves an Update Notification file at any point during the rotation process, it will no longer be able to verify the signature. In that scenario manual recovery is required, similar to a first time configuration of a new mirror client.

9.7. Scalability

As NRTMv4 uses HTTPS for all file retrieval, mirror servers can use standard HTTP caching infrastructure, horizontal scaling, and CDN integration to handle load. Caching considerations for each file type are discussed in the respective Cache Concerns sections.

Delta Files are typically small, as they only contain changes since the previous version. Snapshot Files can be larger, but all files may be compressed with GZIP to reduce storage and bandwidth requirements.

9.8. NRTMv3 Compatibility

NRTMv4 is a new protocol that is not backwards compatible with NRTMv3. There is no migration path between the two protocols; operators wishing to support both can run them in parallel.

10. IANA Considerations

This document has no IANA actions.

11. Security Considerations

IRR objects serve many purposes, including automated network configuration and filtering. Manipulation of IRR objects can therefore have a significant security impact. However, security in existing protocols is mostly absent.

Before NRTMv4, the most common protocols for IRR Database mirroring are FTP for retrieving full snapshots and NRTM version 3 for retrieving later changes. There are no provisions for integrity or authenticity and there are various scenarios where mirroring may not be reliable.

NRTMv4 requires integrity verification. The Update Notification File is signed and clients must verify its signature. The signed Update Notification File contains SHA-256 hashes of all Delta and Snapshot Files, which clients must verify through calculating and comparing hashes on retrieval. Together, the signature and hash verification form a chain of trust from the signing key to each individual file. Additionally, the channel security offered by HTTPS further limits security risks.

By allowing publication on any HTTPS endpoint, NRTMv4 allows for extensive scaling and there are many existing techniques and services to protect against denial-of-service attacks. In contrast, NRTMv3 required mirror clients to directly query the IRR server instance with special WHOIS queries. This scales poorly and there are no standard protections against denial-of-service available.

Since Snapshot and Delta Files may be compressed with GZIP, a maliciously crafted compressed file could expand to a much larger size than expected, potentially exhausting memory or disk resources. Implementations **SHOULD** enforce limits on the decompressed size of files relative to the compressed size or the expected data volume for the IRR Database. See also the security considerations of [RFC6713].

The HTTPS endpoint used for NRTMv4 **MUST** be configured according to the best practices in [BCP195]. Mirror clients **MUST NOT** use other protocols than HTTPS, such as HTTP or FTP.

12. References

12.1. Normative References

[BCP195] Best Current Practice 195, <https://www.rfc-editor.org/info/bcp195>.
At the time of writing, this BCP comprises the following:

Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/info/rfc8996>.

Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

Salz, R. and N. Aviram, "New Protocols Using TLS Must Require TLS 1.3", BCP 195, RFC 9852, DOI 10.17487/RFC9852, July 2026, <https://www.rfc-editor.org/info/rfc9852>.

[IANA_jose] IANA, "JSON Object Signing and Encryption (JOSE)", <https://www.iana.org/assignments/jose>.

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, DOI 10.17487/RFC1952, May 1996, <https://www.rfc-editor.org/info/rfc1952>.

[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/info/rfc2119>.

[RFC2622] Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D., and M. Terpstra, "Routing Policy Specification Language (RPSL)", RFC 2622, DOI 10.17487/RFC2622, June 1999, <https://www.rfc-editor.org/info/rfc2622>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC4012] Blunk, L., Damas, J., Parent, F., and A. Robachevsky, "Routing Policy Specification Language next generation (RPSLng)", RFC 4012, DOI 10.17487/RFC4012, March 2005, <https://www.rfc-editor.org/info/rfc4012>.

[RFC7464] Williams, N., "JavaScript Object Notation (JSON) Text Sequences", RFC 7464, DOI 10.17487/RFC7464, February 2015, <https://www.rfc-editor.org/info/rfc7464>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/info/rfc7468>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[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/info/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/info/rfc8259>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[SHS] NIST, "Secure Hash Standard (SHS)", NIST FIPS 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[STD97] Internet Standard 97, <https://www.rfc-editor.org/info/std97>.
At the time of writing, this STD comprises the following:

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/info/rfc9110>.

12.2. Informative References

[NRTMv3] RIPE NCC, "Access to NRTM", March 2026, <https://docs.db.ripe.net/RIPE-Database-Mirror/Access-to-NRTM/>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6713] Levine, J., "The 'application/zlib' and 'application/gzip' Media Types", RFC 6713, DOI 10.17487/RFC6713, August 2012, <https://www.rfc-editor.org/info/rfc6713>.

[RFC8182] Bruijnzeels, T., Muravskiy, O., Weber, B., and R. Austein, "The RPKI Repository Delta Protocol (RRDP)", RFC 8182, DOI 10.17487/RFC8182, July 2017, <https://www.rfc-editor.org/info/rfc8182>.

Acknowledgments

The authors would like to thank George Michaelson, Shon Huang, Tim Bruijnzeels, Mahesh Aggarwal, Fedor Vompe, Paul Etchells, and Mohamed Boucadair for their helpful review of this document and/or work on implementations. The authors would also like to thank Daniele Ceccarelli, Claudio Allocchio, Menachem Dodge, Julian Reschke, Watson Ladd, and Paul Kyzivat for their reviews during IETF Last Call. The authors thank Gorry Fairhurst, Andy Newton, Éric Vyncke, Mike Bishop, Deb Cooley, Roman Danyliw, Mahesh Jethanandani, Ketan Talaulikar, Gunter Van de Velde, Christopher Inacio, Jim Guichard, and Tommy Jensen for the reviews and feedback.

Authors' Addresses

Sasha Romijn
Reliably Coded
Amsterdam
Netherlands
Email: sasha@reliablycoded.nl
URI: https://www.reliablycoded.nl/

Job Snijders
BSD Software Development
Amsterdam
Netherlands
Email: job@bsd.nl
URI: https://www.bsd.nl/

Edward Shryane
RIPE NCC
Amsterdam
Netherlands
Email: eshryane@ripe.net

Stavros Konstantaras
AMS-IX
Amsterdam
Netherlands
Email: stavros.konstantaras@ams-ix.net