GROW S. Romijn

Internet-Draft Reliably Coded

Intended status: Standards Track J. Snijders

Expires: 15 November 2025 Fastly

E. Shryane

RIPE NCC

S. Konstantaras

AMS-IX

14 May 2025

Near Real Time Mirroring (NRTM) version 4

draft-ietf-grow-nrtm-v4-07

Abstract

This document specifies a one-way synchronization protocol for

Internet Routing Registry (IRR) records, called Near Real Time Mirroring version 4 (NRTMv4). The protocol allows

instances of IRR database servers to mirror IRR records, specified in

the Routing Policy Specification Language (RPSL), between each other.

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.

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 15 November 2025.

Copyright Notice

Copyright (c) 2025 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. Informal overview . . . . . . . . . . . . . . . . . . . . . . 3

3. Mirror server use . . . . . . . . . . . . . . . . . . . . . . 5

3.1. Key Configuration . . . . . . . . . . . . . . . . . . . . 5

3.2. Snapshot Initialization . . . . . . . . . . . . . . . . . 5

3.3. Publishing updates . . . . . . . . . . . . . . . . . . . 6

3.3.1. Delta Files . . . . . . . . . . . . . . . . . . . . . 6

3.3.2. Snapshot Files . . . . . . . . . . . . . . . . . . . 7

3.3.3. Update Notification File . . . . . . . . . . . . . . 7

3.3.4. Publication Policy Restrictions . . . . . . . . . . . 8

4. Mirror client use . . . . . . . . . . . . . . . . . . . . . . 8

4.1. Client Configuration . . . . . . . . . . . . . . . . . . 8

4.2. Initialization from snapshot . . . . . . . . . . . . . . 8

4.3. Processing Delta Files . . . . . . . . . . . . . . . . . 9

4.4. Signature and Staleness Verification . . . . . . . . . . 10

4.5. Policy Restrictions . . . . . . . . . . . . . . . . . . . 10

5. Update Notification File . . . . . . . . . . . . . . . . . . 11

5.1. Purpose . . . . . . . . . . . . . . . . . . . . . . . . . 11

5.2. Cache concerns . . . . . . . . . . . . . . . . . . . . . 11

5.3. Payload format and validation . . . . . . . . . . . . . . 11

5.4. Encoding and signature . . . . . . . . . . . . . . . . . 14

6. Snapshot File . . . . . . . . . . . . . . . . . . . . . . . . 14

6.1. Purpose . . . . . . . . . . . . . . . . . . . . . . . . . 14

6.2. Cache Concerns . . . . . . . . . . . . . . . . . . . . . 14

6.3. File format and validation . . . . . . . . . . . . . . . 14

7. Delta File . . . . . . . . . . . . . . . . . . . . . . . . . 15

7.1. Purpose . . . . . . . . . . . . . . . . . . . . . . . . . 15

7.2. Cache Concerns . . . . . . . . . . . . . . . . . . . . . 16

7.3. File format and validation . . . . . . . . . . . . . . . 16

8. Operational Considerations . . . . . . . . . . . . . . . . . 17

8.1. IRR object Validation . . . . . . . . . . . . . . . . . . 17

8.2. Intermediate mirror instances . . . . . . . . . . . . . . 18

8.3. Reading from local files . . . . . . . . . . . . . . . . 19

8.4. Public key rotation . . . . . . . . . . . . . . . . . . . 19

9. Security Considerations . . . . . . . . . . . . . . . . . . . 20

10. Acknowledgments . . . . . . . . . . . . . . . . . . . . . . . 20

11. Normative References . . . . . . . . . . . . . . . . . . . . 20

12. Informative References . . . . . . . . . . . . . . . . . . . 22

Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . 22

1. Introduction

The Internet Routing Registry (IRR) consists of several IRR

Databases, each storing objects in the Routing Policy Specification

Language (RPSL). About a dozen larger IRR Databases are well known

and widely used, operated by different organizations such as RIRs 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, designed to address

issues in existing IRR Database mirroring protocols. In NRTMv4, IRR

Databases publish their records on an HTTPS 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 [RFC8182], as there are overlaps between the two

protocols.

Of earlier NRTM versions, NRTMv3 [NRTMv3] was widely

deployed, although there is no formal specification. Some

comparisons are provided in Section 9.

2. Overview

In NRTMv4, a mirror server is an instance of IRR Database software

that has a database of IRR objects and publishes them to allow

mirroring by others. This can be retrieved by mirror clients, which

then load the IRR objects into their local storage.

Publication consists of three different files:

\* A single Update Notification File: 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: Contains all published IRR

objects at a particular version. The mirror server periodically

generates a new snapshot.

\* Zero or more Delta Files: 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 the JavaScript Object

Notation (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. All files MUST

use UTF-8 encoding [STD63] and 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 to determine whether there are any

changes, and then retrieve only the relevant Delta Files (if any), if any.

This minimizes data transfer. Deltas have sequential versions.

Mirror clients are configured with the URL of an Update Notification

File, 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 3.2). This allows long term caching and 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 identifier 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 identifier 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 may not.

3. Mirror Server Use

3.1. Key Configuration

When enabling NRTMv4 publication for an IRR Database, the operator

MUST generate and configure a private Elliptic Curve JSON Web Key

[RFC7517]. The operator then provides this 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

encoded in PEM. 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 8.4.

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. It is also RECOMMENDED that all configuration

options are clearly named to indicate that they are private

keys.

3.2. 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 cannot 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 must

reinitialize, which will force clients to reinitialize as well.

Initialization consists of these actions:

\* The mirror server MUST generate a new session identifier. 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 identifier is unique to the IRR Database, so an

instance that serves multiple IRR Databases, will create a

separate session identifier 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 identifier, a reference to the new snapshot, and no deltas.

Note that a publication, and its associated session identifiers s and versions,

always relates to a single specific IRR Database, even if multiple

databases are published from one instance. For example, a mirror

server publishing NRTMv4 for RIPE and RIPE-NONAUTH, will generate two

Update Notification Files, referring two Snapshot Files, and two sets

of Delta Files each with contiguous version numbers - all completely

independent to each other, with different session identifiers, potentially at

different times. This applies even if the same IRR server instance

produces both.

3.3. Publishing Updates

After creating the initialization files, the mirror server processes

updates by publishing Delta Files and, periodically, a new Snapshot

File.

3.3.1. 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 to allow it to be indefinitely cached. It

MUST also contain a random value that cannot be predicted before

publication, to counter negative caching issues.

\* After generating a new Delta File, a mirror server SHOULD remove

all Delta Files older than 24 hours.

\* The Update Notification File MUST be updated to include the new

Delta File and update the database version.

\* Note that, as Delta Files always contain changes compared to a

previous state, there can never be a Delta File with version 1.

3.3.2. Snapshot Files

Snapshot Files after initialization are generated as follows:

\* The mirror server MUST generate a new Snapshot File between once

per hour and once per day, if there have been changes to the IRR

objects.

\* The version number of the new snapshot MUST be equal to the last

Delta File version.

\* If there have been no changes to the IRR objects since the last

snapshot, the mirror server MUST NOT generate a new snapshot.

\* The URL where the Snapshot File is published MUST contain the

session ID and version number to allow it to be indefinitely

cached. It MUST also contain a random value that cannot 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 SHOULD continue to produce Delta Files during

this window, which 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.

3.3.3. Update Notification File

The Update Notification File MUST be updated when a new Delta or

Snapshot File is published and, even if there have been no changes,

at least every 24 hours.

3.3.4. 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. It is RECOMMENDED to modify objects in such a way that

this change is evident to humans reading the object text, 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.

4. Mirror Client Use

4.1. 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 8.4.

4.2. 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 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. In case of a mismatch of this hash,

the file MUST be rejected.

\* The client MUST record the session identifier and version of the loaded

Snapshot File.

4.3. 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 Identifier matches the previously

known session identifier. 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, 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 previous entries seen

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 can do this

by recording the files referenced by the previous valid Update

Notification File and comparing the overlapping entries with the

retrieved 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 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. 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 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.

4.4. 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 8.4.

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.

4.5. 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.

5. Update Notification File

5.1. 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, even if there have been

no changes, at least every 24 hours.

5.2. 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.

5.3. Payload Format and Validation

An example payload of an Update Notification File is provided in Figure X.

{

"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.2.047595d0fae972fbed0c51b4a41c7a349e0c47bb.json.gz",

"hash": "9a..86"

},

"deltas": [

{

"version": 2,

"url": "ca128382-78d9-41d1-8927-1ecef15275be/nrtm-delta.1.784a2a65aba22e001fd25a1b9e8544e058fbc703.json",

"hash": "62..a2"

},

{

"version": 3,

"url": "ca128382-78d9-41d1-8927-1ecef15275be/nrtm-delta.2.0f681f07cfab5611f3681bf030ec9f6fa3442fb0.json",

"hash": "25..9a"

},

{

"version": 4,

"url": "ca128382-78d9-41d1-8927-1ecef15275be/nrtm-delta.3.d9c194acbb2cb0d4088c9d8a25d5871cdd802c79.json",

"hash": "b4..13"

}

],

"metadata": {}

}

Figure X: PLEASE ADD TITLE

Note: hash and key values in this example are shortened because of

formatting.

The following validation rules MUST be observed when creating or

parsing Update Notification Files:

\* The nrtm\_version MUST be 4.

\* The timestamp MUST be an [RFC3339] timestamp 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 an Elliptic Curve 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 8.4. 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 [RFC4122] 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.json”, the full URL for the referred snapshot is

https://example.com/nrtm/ca128382-78d9-41d1-8927-1ecef15275be/

nrtm-snapshot.2.047595d0fae972fbed0c51b4a41c7a349e0c47bb.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 [SHS] of the referenced

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, 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.

5.4. Encoding and signature

\* The actual Update Notification File contents MUST be a JSON Web

Signature [RFC7515] and MUST use JWS Compact Serialization.

\* The JWS Payload MUST be the JavaScript Object Notation (JSON)

[RFC8259] serialization of the structure described in the previous

section.

\* The filename of the serialized data MUST be "update-notification-

file.jose".

\* The algorithm MUST NOT be Deprecated, and it is RECOMMENDED to use

Recommended or Recommended+ algorithms, as defined in JSON Web

Algorithms [RFC7518]

6. Snapshot File

6.1. 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.

6.2. 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 MUST be published at a URL that is unique to the

specific session and version. The URL MUST also contain a random

value that cannot be predicted before publication, to counter

negative caching issues.

Because these files never change, they MAY be cached indefinitely.

However, as snapshots are large and old snapshots will no longer be

referred by newer Update Notification Files, it is RECOMMENDED that a

limited interval is used in the order of hours or days.

To avoid race conditions where a mirror client retrieves an Update

Notification File moments before it's updated, mirror servers SHOULD

retain old Snapshot Files for at least 5 minutes after a new Update

Notification File is published.

6.3. File format and validation

Figure X depicts an example Snapshot File.

␞{

"nrtm\_version": 4,

"type": "snapshot",

"source": "EXAMPLE",

"session\_id": "ca128382-78d9-41d1-8927-1ecef15275be",

"version": 3

}

␞{"object": "route: 192.0.2.0/24\norigin: AS65530\nsource: EXAMPLE"}

␞{"object": "route: 2001:db8::/32\norigin: AS65530\nsource: EXAMPLE"}

Figure X: PLEASE ADD TITLE.

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.

7. Delta File

7.1. 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.

7.2. 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 MUST be published at a URL that is unique to the

specific session and version. The URL MUST also contain 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 is 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.

7.3. File format and validation

Figure X depicts an example Delta File.

␞{

"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": "192.0.2.0/24AS65530"

}

␞{

"action": "add\_modify",

"object": "route: 2001:db8::/32\norigin: AS65530\nsource: EXAMPLE"

}

Figure X: ADD TITLE.

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

representing a change, which MUST meet the following rules:

\* An action attribute, which is either "delete" for object

deletions, or "add\_modify" for additions or modifications.

\* If action is "delete": 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 and therefore

mirror clients MUST use case insensitive matching against their

local database.

\* If action is "add\_modify": an object attribute with the RPSL text

of the new version of the object.

8. Operational Considerations

xx

8.1. 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 which 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.

8.2. 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 function 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.

8.3. 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.

8.4. Public Key Rotation

It is RECOMMENDED that IRR Database operators rotate the signing key

on their mirror server about once per year. 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

Notification Update File to be refreshed earlier, with the

new\_signing\_key included, and thus start the propagation earlier.

\* When mirror clients next retrieve the Update Notification File,

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 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.

X

9. 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 Delta and Snapshot Files

are verified using the SHA-256 hash in the Update Notification File,

and the Update Notification File is verified using its signature.

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 [RFC3912]. This scales poorly, and there

are no standard protections against denial-of-service available.

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.

X. IANA Considerations

TBC.

10. Acknowledgments

The authors would like to thank George Michaelson, Shon Huang, Tim

Bruijnzeels, Mahesh Aggarwal, Fedor Vompe, and Paul Etchells for

their helpful review of this document and/or work on implementations.

11. Normative References

[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>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally

Unique IDentifier (UUID) URN Namespace", RFC 4122,

DOI 10.17487/RFC4122, July 2005,

<https://www.rfc-editor.org/info/rfc4122>.

[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>.

[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>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518,

DOI 10.17487/RFC7518, May 2015,

<https://www.rfc-editor.org/info/rfc7518>.

[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>.

[RFC9325] 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>.

[SHS] National Institute of Standards and Technology, "Secure

Hash Standard", March 2012,

<https://csrc.nist.gov/publications/fips/fips180-4/fips-

180-4.pdf>.

12. Informative References

[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>.

[NRTMv3] RIPE NCC, "Access to NRTM(v3)", July 2024,

<https://docs.db.ripe.net/RIPE-Database-Mirror/Access-to-

NRTM/>.

Authors' Addresses

Sasha Romijn

Reliably Coded

Amsterdam

Netherlands

Email: sasha@reliablycoded.nl

Job Snijders

Fastly

Amsterdam

Netherlands

Email: job@fastly.com

Edward Shryane

RIPE NCC

Amsterdam

Netherlands

Email: eshryane@ripe.net

Stavros Konstantaras

AMS-IX

Amsterdam

Netherlands

Email: stavros.konstantaras@ams-ix.net