DISPATCH Working Group
Internet-Draft
Intended status: Standards Track
Expires: 27 November 2026

R. Yoshioka
e.Mix
G. Labadessa
OrcaTI
P. Scalon
Sysbrasil Tecnologia
D. C. D. Brito
Vero Internet
E. Belotto
Zadara

26 May 2026

# SNAP: Simple Native Archive Protocol
# draft-melegassi-dispatch-mvps-snap-backup-00

## Abstract

SNAP (Simple Native Archive Protocol) defines a minimal, self-describing backup object encoded as a single JSON document, modeled in YANG [RFC7950] and encoded per [RFC7951]. A SNAP object contains a manifest of files with per-file cryptographic hashes, integrity-verified metadata, and a compressed binary payload -- all within one atomic, transport-agnostic JSON document. Any agent capable of parsing JSON and applying standard decompression can produce or restore a SNAP backup without proprietary tooling, side-channel metadata, or multi-phase handshakes.

This document defines the YANG module "snap", its JSON encoding, integrity verification procedure, bindings to common transports, and its role as the atomic transport substrate for the Multi-Vantage Path State (MVPS) family of Internet-Drafts. When a SNAP Object carries an MVPS Bundle [I-D.melegassi-ippm-mvps-bundle], the canonical-form determinism of SNAP (Theorem 1 of this document) composes with the MVPS Architecture axioms A1..A5 [I-D.melegassi-iab-mvps-architecture] to inherit the full coherence algebra of [MVPS-v4] (Theorems 1, 2, 3, 3′, 4, 5, 9 and Stein’s Lemma [Stein52] [CoverThomas2006]) without alteration.

## About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://melegassi-labadessa.github.io/snap-backup/draft-melegassi-dispatch-mvps-snap-backup.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-melegassi-dispatch-mvps-snap-backup/.

Discussion of this document takes place on the DISPATCH Working Group mailing list (mailto:dispatch@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/dispatch/. Subscribe at https://www.ietf.org/mailman/listinfo/dispatch/.

Source for this draft and an issue tracker can be found at https://github.com/melegassi-labadessa/snap-backup.

## 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 27 November 2026.

## Copyright Notice



## Table of Contents

1. Introduction . . . . . 4
   1.1. Relationship to Existing Standards . . . . . 5
   1.2. Relationship to the MVPS Family . . . . . 5
   1.3. Requirements Language . . . . . 6
2. Terminology . . . . . 6
3. SNAP Object Format . . . . . 7
   3.1. Top-Level Structure . . . . . 7

3.2. Field Definitions . . . . . 7
3.3. Compression . . . . . 8
3.4. Envelope Integrity . . . . . 9
3.5. Canonical Form (Theorem of Determinism) . . . . . 9
4. YANG Module . . . . . 10
5. Operations . . . . . 15
5.1. Encoding a SNAP Object . . . . . 15
5.2. Decoding and Restoring from a SNAP Object . . . . . 16
6. Transport Bindings . . . . . 16
6.1. HTTP/2 and HTTP/1.1 . . . . . 16
6.2. WebSocket . . . . . 17
6.3. Size and Atomicity . . . . . 17
7. Conformance . . . . . 17
7.1. Profiles . . . . . 18
7.2. Feature Matrix . . . . . 18
7.3. Interoperability Test Suite . . . . . 19
7.4. Vendor Implementation Notes . . . . . 20
8. Security Considerations . . . . . 20
8.1. Integrity . . . . . 20
8.2. Confidentiality . . . . . 21
8.3. Replay and Freshness . . . . . 21
8.4. Denial of Service . . . . . 21
9. IANA Considerations . . . . . 21
9.1. YANG Module Registration . . . . . 21
9.2. Media Type Registration . . . . . 21
9.3. URI Namespace Registration . . . . . 22
10. References . . . . . 22
10.1. Normative References . . . . . 23
10.2. Informative References . . . . . 24
Appendix A. Mathematical Proofs of Design Decisions . . . . . 26
A.1. Theorem A.1: Compression Lower Bound (Shannon, 1948) . . 26
A.2. Theorem A.2: SHA-256 Collision Resistance . . . . . 26
A.3. Theorem A.3: Base64+Brotli Overhead is Approximately Unity . . . . . 27
A.4. Theorem A.4: Minimum Round-Trip Transport Time . . . . . 27
A.5. Theorem A.5: Manifest Verification Complexity . . . . . 27
A.6. Theorem A.6: Round-Trip Correctness . . . . . 28
Appendix B. Reference Pseudocode . . . . . 29
B.1. Encode . . . . . 29
B.2. Decode . . . . . 30
B.3. Verify Without Restore . . . . . 31
Appendix C. Test Vectors . . . . . 31
C.1. Vector 1: Empty Backup . . . . . 31
C.2. Vector 2: Single-File Backup . . . . . 32
C.3. Vector 3: JCS Round-Trip . . . . . 33
C.4. Vector 4: Tamper Detection . . . . . 34
C.5. Vector 5: MVPS Bundle Round-Trip . . . . . 34
C.6. Conformance Procedure . . . . . 35

Appendix D. SNAP x MVPS Composition Theorems . . . . . 35
D.1. D.1 Theorem M.1 (Bundle Preservation) . . . . . 35
D.2. D.2 Theorem M.2 (Architectural Conformance) . . . . . 36
D.3. D.3 Theorem M.3 (Byzantine-Resilient Backup) . . . . . 36
D.4. D.4 Theorem M.4 (Planetary Floor for SNAP Backup) . . . . 37
D.5. D.5 Operational Contracts Inherited from MVPS . . . . . 38
D.6. D.6 SNAP-MVPS Traceability Matrix . . . . . 39
Contributors . . . . . 39
Acknowledgments . . . . . 40
Change Log . . . . . 41
-00 . . . . . 41
Authors' Addresses . . . . . 41

## 1. Introduction

Existing backup solutions tightly couple data capture, transport, and restoration into proprietary stacks. A system administrator wishing to back up configuration files today must choose between raw archives (POSIX tar, IEEE Std 1003.1), complex multi-phase protocols (rsync, BorgBackup), or vendor-specific agents. None of these produce a self-describing, cryptographically verifiable, machine-readable artifact that can be inspected, transported, and restored by any standards-compliant agent without prior knowledge of the producing system.

SNAP addresses this gap by defining:

1. A data model for backup objects expressed in YANG 1.1 [RFC7950], enabling schema validation by any YANG-capable tool.

2. A canonical JSON encoding following [RFC7951], with deterministic serialization per the JSON Canonicalization Scheme [RFC8785] to support reproducible integrity hashes.

3. An integrity model based on SHA-256 [FIPS180-4] applied to both individual files (in the manifest) and the envelope object (via JCS).

4. A transport binding over HTTP/2 [RFC9110] and WebSocket [RFC6455], with a size-aware atomicity requirement for small objects.

The design philosophy of SNAP is conservative composition: every primitive is drawn from existing IETF or NIST standards. SNAP defines only the composition and the backup-specific YANG module.

### 1.1. Relationship to Existing Standards

RFC 9195 [RFC9195] defines the YANG Instance Data File Format, which provides a container for YANG-modeled data at rest. RFC 9195 is concerned with YANG schema instances (configuration and state data from YANG-modeled systems), not with arbitrary file collections. SNAP extends the spirit of RFC 9195 to the general file backup domain.

The YANG provenance mechanism [I-D.lopez-opsawg-yang-provenance] applies COSE signatures to YANG instance data. SNAP uses JCS-based SHA-256 rather than COSE because the backup payload is a binary blob rather than a YANG instance tree.

YANG Packages [I-D.ietf-netmod-yang-packages] define offline distribution of YANG schemas. SNAP is complementary: it backs up the operational data, not the schema.

### 1.2. Relationship to the MVPS Family

SNAP is designed to compose with the Multi-Vantage Path State (MVPS) family of Internet-Drafts as their atomic transport substrate. This subsection summarizes the dependency.

The MVPS family consists of nine related drafts (D-1 through D-7, D-15, D-16) whose mathematical authority is the v4.0 proof catalogue [MVPS-v4], validated against 37 adversarial attacks across seven audit rounds (44/44 PASS). The catalogue defines:

* A bundle algebra (D1, D3, F1-F4, I1) over coherence axes (C_1 causal, C_2 informational, C_3 topological) and a bounded scalar H : R^3 -> [0, H_max] with H_max = -3 log epsilon (Theorem 1).

* Detection statistics (Theorems 2, 3, 3′) using the Mahalanobis statistic D^2 on the coherence vector C(t).

* A Byzantine breakdown bound (Theorem 9) of (2f / (N - 2f)) * sqrt(2) on geometric-median centroids for N vantages with f adversaries.

* Architectural axioms MVPS-A1..A5 [I-D.melegassi-iab-mvps-architecture] and the Invariance Theorem: any architecture satisfying A1..A5 inherits Theorems 1, 2, 3, 3′, 4, 5, 9 and Stein’s Lemma [Stein52] [CoverThomas2006] verbatim.

* A Planetary Coherence Floor [I-D.melegassi-iab-mvps-planetary-floor] giving R* = max{tau_causal, tau_sampling, tau_information, tau_consensus, tau_coupling} as the lower bound for planet-scale detect-and-react.

The MVPS Bundle [I-D.melegassi-ippm-mvps-bundle] is the canonical RFC 7951 envelope that carries a coordinated multi-vantage observation. A reference bundle contains a bundle_id (UUID v4 lowercase), a schema_version, a destination, a coordination_window, and an ordered array of per-vantage snapshots -- all serialized canonically per [RFC8785].

Because the SNAP envelope shares the _same_ canonicalization stack as the MVPS Bundle (RFC 7951 JSON encoding, JCS canonicalization, SHA-256 pinning, UUID v4 identifiers), an MVPS Bundle can be carried verbatim inside the SNAP payload field with no loss of structural or cryptographic guarantees. Section 8.2 (Appendix D) of this document proves the composition theorem and lists the operational contracts (OC1..OC8) of [MVPS-v4] that MUST be preserved by SNAP encoders carrying MVPS data.

### 1.3. 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.

## 2. Terminology

The following terms are used in this document:

SNAP Object: A single JSON document conforming to the "snap" YANG module defined in Section 4. Also referred to as "backup object".

Manifest: The ordered list of file entries within a SNAP Object. Each entry records the file path, size, modification time, and SHA-256 digest.

Payload: The binary content of the backup: a compressed archive of all files listed in the Manifest, encoded in Base64 [RFC4648].

Encoder: A process or system that produces a SNAP Object from a set of files.

Decoder: A process or system that restores files from a SNAP Object.

JCS: JSON Canonicalization Scheme [RFC8785], used to produce a deterministic byte sequence over which the envelope hash is computed.

## 3. SNAP Object Format

A SNAP Object is a JSON object [RFC8259] whose structure is fully specified by the YANG module in Section 4 and encoded per [RFC7951].

### 3.1. Top-Level Structure

The following is a non-normative illustration of a SNAP Object:

```
{
  "snap:backup": {
    "version": "1.0",
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "created": "2026-05-26T08:00:00Z",
    "src": {
      "host": "node-01.example.com",
      "path": "/etc/configs"
    },
    "meta": {
      "files": 12,
      "size-bytes": 4096,
      "enc": "br",
      "hash": "sha256:e3b0c44298fc1c149afbf4c8996fb92427a..."
    },
    "manifest": [
      {
        "file": "app.conf",
        "sha256": "abc123def456...",
        "size": 512,
        "mtime": "2026-05-26T07:00:00Z"
      }
    ],
    "payload": "H4sIAAAAAAAAA..."
  }
}
```

### 3.2. Field Definitions

version: MUST be the string "1.0" for documents conforming to this specification. Future versions will use a new value.

id: A UUID [RFC4122] in its canonical textual representation. Encoders MUST generate version 4 (random) UUIDs. IDs MUST be unique across all SNAP Objects produced by an Encoder.

created: The timestamp at which encoding began, as a [RFC6991] date-and-time value (ISO 8601 format, UTC).

src: A container identifying the origin of the backup. The "host" leaf is a fully qualified domain name or IP address of the source system. The "path" leaf is an absolute path on the source system.

meta: Metadata about the encoded content. "files" is the count of entries in the manifest. "size-bytes" is the sum of sizes of all source files. "enc" identifies the compression algorithm applied to the payload (see Section 3.3). "hash" is the envelope integrity hash (see Section 3.4).

manifest: An ordered list of file entries. Each entry MUST correspond to a file present in the payload. The "file" field is a relative path from "src/path". The "sha256" field is the lowercase hex-encoded SHA-256 digest of the uncompressed file content [FIPS180-4]. The "size" field is the exact byte count of the uncompressed file. The "mtime" field is the last-modification timestamp.

payload: The Base64-encoded [RFC4648] compressed archive of all files listed in the manifest. The archive format is a POSIX-compliant tar stream. The compression algorithm is identified by "meta/enc".

## 3.3. Compression

The "meta/enc" leaf identifies the compression applied to the tar stream before Base64 encoding. The following values are defined:

| Value | Specification | Recommended |
|---|---|---|
| none | No compression | No |
| gz | Gzip [RFC1952] | No |
| br | Brotli [RFC7932] | YES |
| zstd | Zstandard [RFC8878] | No |

Table 1

Encoders SHOULD use "br" (Brotli) as the default compression because Brotli consistently achieves compression ratios closest to the Shannon entropy bound [Shannon48] for structured text data (configuration files, YANG instance data, scripts) while being an IETF-standardized algorithm.

Decoders MUST support all four values.

### 3.4. Envelope Integrity

The "meta/hash" field contains the envelope integrity hash in the format "sha256:<hex>", where <hex> is the lowercase hexadecimal encoding of a SHA-256 digest [FIPS180-4].

The digest is computed over the canonical JSON serialization of the SNAP Object with the "meta/hash" field set to the empty string "". Canonical serialization follows the JSON Canonicalization Scheme [RFC8785].

Formally, let O be the SNAP Object and O′ be the object derived by setting O["snap:backup"]["meta"]["hash"] to "". Then:

meta/hash = "sha256:" + lowercase_hex(SHA-256(JCS(O′)))

Decoders MUST verify the envelope hash before restoring any file. Decoders MUST also verify each per-file "sha256" digest against the content extracted from the payload. A Decoder MUST NOT restore any file from a SNAP Object that fails either verification.

### 3.5. Canonical Form (Theorem of Determinism)

A central property of SNAP is that any two conforming Encoders, given the same input files and metadata, MUST produce SNAP Objects with the same envelope hash. This subsection states the theorem and the implementation rules that guarantee it.

*Theorem 1 (Encoder Determinism).* Let E1 and E2 be two conforming SNAP Encoders. Let F be a set of files with identical content, paths, sizes, and modification times. Let M = (id, created, host, path, enc) be identical metadata inputs. Then:

E1(F, M).meta.hash == E2(F, M).meta.hash

provided that "enc" specifies a deterministic compression algorithm (Section 6.1).

The proof rests on three lemmas:

Lemma 1 (Tar determinism): Given identical file contents and metadata, a SNAP-conforming tar archive MUST use the USTAR format (POSIX 1003.1-1988) with files listed in lexicographic order of relative path, with all extended attributes and ownership fields zeroed. Two such archives are byte-identical.

Lemma 2 (Compression determinism): The codecs "br" (Brotli quality 11) and "gz" (gzip level 9, no filename, no timestamp) are deterministic functions of their input. Zstandard ("zstd") MUST use level 19 with no checksum frame for determinism.

Lemma 3 (JCS uniqueness, [RFC8785]): For any JSON value V conforming to the I-JSON profile [RFC7493], the JCS canonicalization JCS(V) produces a unique byte sequence.

Combining the three lemmas, the inputs to SHA-256 are byte-identical across implementations; therefore the resulting "meta/hash" is identical. QED.

*Corollary 1 (Cross-vendor interoperability).* Any SNAP Decoder that successfully verifies a SNAP Object produced by Encoder E1 will successfully verify the same Object regenerated by Encoder E2 from the same inputs.

## 4. YANG Module

This section defines the normative YANG 1.1 [RFC7950] module "snap".

```
module snap {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:snap";
  prefix snap;

  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types";
  }

  organization
    "e.Mix / OrcaTI / Sysbrasil Tecnologia / Vero Internet /
     Zadara";

  contact
    "WG Web:      https://datatracker.ietf.org/wg/dispatch/
     WG List:     dispatch@ietf.org

     Author:      Rodrigo Yoshioka (e.Mix)
                  <royoshioka@gmail.com>
```

```
        Co-author:   Guilherme Labadessa (OrcaTI)
                     <guilabadessa@gmail.com>
        Co-author:   Pedro Scalon (Sysbrasil Tecnologia)
                     <pedroscalon01@gmail.com>
        Co-author:   Diego Canton de Brito (Vero Internet)
                     <diegocdeb@hotmail.com>
        Co-author:   Eduardo Belotto (Zadara)
                     <ebelotto@gmail.com>

        Contributor: Leonardo Melegassi (Catellix)
                     <melegassi@catellix.com>";

      description
        "This module defines the data model for a SNAP backup object.
         SNAP (Simple Native Archive Protocol) encodes a complete file
         backup as a single JSON document.

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

         Redistribution and use in source and binary forms, with or
         without modification, is permitted pursuant to, and subject to
         the license terms contained in, the Revised BSD License set
         forth in Section 4.c of the IETF Trust's Legal Provisions
         Relating to IETF Documents
         (https://trustee.ietf.org/license-info).";

      revision 2026-05-26 {
        description "Initial version.";
        reference "draft-melegassi-dispatch-mvps-snap-backup-00";
      }

      typedef snap-version {
        type string {
          pattern '1\.[0-9]+';
        }
        description
          "SNAP protocol version string.  The format is MAJOR.MINOR.
           This document defines version 1.0.";
      }

      typedef compression-algorithm {
        type enumeration {
          enum none {
            value 0;
            description "No compression.";
          }
          enum gz {
```

```
          value 1;
          description "Gzip compression per RFC 1952.";
          reference "RFC 1952: GZIP file format specification";
        }
        enum br {
          value 2;
          description
            "Brotli compression per RFC 7932.
             This is the RECOMMENDED algorithm.";
          reference "RFC 7932: Brotli Compressed Data Format";
        }
        enum zstd {
          value 3;
          description "Zstandard compression per RFC 8878.";
          reference "RFC 8878: Zstandard Compression and the
                     application/zstd Media Type";
        }
      }
      description "Compression algorithm applied to the payload.";
    }

    typedef integrity-hash {
      type string {
        pattern 'sha256:[0-9a-f]{64}';
      }
      description
        "An integrity hash in the format 'sha256:<hex>', where <hex>
         is the lowercase hexadecimal SHA-256 digest per FIPS 180-4.
         The value '' (empty string) is reserved for use during
         envelope hash computation.";
    }

    container backup {
      description
        "Root container for a SNAP backup object.  A conforming
         JSON document contains exactly one instance of this
         container, under the key 'snap:backup'.";

      leaf version {
        type snap-version;
        mandatory true;
        description
          "SNAP protocol version.  MUST be '1.0' for documents
           conforming to this specification.";
      }

      leaf id {
        type yang:uuid;
```

```
        mandatory true;
        description
          "Globally unique identifier for this backup object.
           Encoders MUST generate version 4 (random) UUIDs per
           RFC 4122.";
        reference "RFC 4122: A Universally Unique IDentifier (UUID)
                   URN Namespace";
      }

      leaf created {
        type yang:date-and-time;
        mandatory true;
        description
          "Timestamp at which encoding began.  MUST be in UTC
           (the time-offset MUST be '+00:00' or 'Z').";
      }

      container src {
        description
          "Source identification for the backed-up data.";

        leaf host {
          type string {
            length "1..253";
          }
          mandatory true;
          description
            "Fully qualified domain name or IP address of the
             system from which files were collected.";
        }

        leaf path {
          type string {
            pattern '/.*';
          }
          mandatory true;
          description
            "Absolute path on the source system that is the root
             of the backed-up file tree.";
        }
      }

      container meta {
        description "Metadata about the backup content.";

        leaf files {
          type uint32;
          description
```

```
            "Number of file entries in the manifest.  MUST equal
             the number of entries in the 'manifest' list.";
          }

          leaf size-bytes {
            type uint64;
            description
              "Sum of 'size' values of all manifest entries, in
               bytes.  Represents the total uncompressed data size.";
          }

          leaf enc {
            type compression-algorithm;
            default "br";
            description
              "Compression algorithm applied to the tar stream
               before Base64 encoding.";
          }

          leaf hash {
            type string;
            description
              "Envelope integrity hash.  During encoding, this field
               is set to '' before computing the JCS digest, then
               replaced with 'sha256:<hex>'.  Decoders MUST verify
               this field before restoring any file.";
          }
        }

        list manifest {
          key "file";
          ordered-by user;
          description
            "Ordered list of file entries.  The order MUST match
             the order of files in the tar stream within the payload.";

          leaf file {
            type string;
            description
              "Relative path of the file from 'src/path'.
               MUST use '/' as the path separator.
               MUST NOT begin with '/'.";
          }

          leaf sha256 {
            type string {
              pattern '[0-9a-f]{64}';
            }
```

```
            mandatory true;
            description
              "Lowercase hex-encoded SHA-256 digest of the
               uncompressed file content.";
          }

          leaf size {
            type uint64;
            mandatory true;
            description "Uncompressed file size in bytes.";
          }

          leaf mtime {
            type yang:date-and-time;
            description "Last modification timestamp of the file.";
          }
        }

        leaf payload {
          type binary;
          mandatory true;
          description
            "Base64-encoded compressed tar archive of all files
             listed in the manifest.  The compression algorithm
             is identified by 'meta/enc'.";
        }
      }
    }
```

Figure 1: YANG module snap

## 5. Operations

### 5.1. Encoding a SNAP Object

An Encoder MUST perform the following steps in order:

1. Collect all files from the source path.

2. For each file, compute the SHA-256 digest of its uncompressed content and record the file path, digest, size, and modification time.

3. Construct a POSIX-compliant tar archive of all collected files, preserving relative paths from the source root.

4. Compress the tar archive using the selected algorithm (RECOMMENDED: Brotli, "br").

5. Base64-encode the compressed archive per [RFC4648], Section 4 (standard encoding with padding).

6. Construct the SNAP Object JSON document with all fields populated and "meta/hash" set to the empty string "".

7. Compute JCS(O') where O' is the object from step 6.

8. Set "meta/hash" to "sha256:" followed by the lowercase hexadecimal SHA-256 digest of JCS(O').

9. The resulting document is the SNAP Object.

### 5.2. Decoding and Restoring from a SNAP Object

A Decoder MUST perform the following steps in order:

1. Parse the JSON document and validate it against the "snap" YANG module. Reject documents that do not conform.

2. Save the value of "meta/hash". Set "meta/hash" to "".

3. Compute JCS(O') and verify that SHA-256(JCS(O')) equals the saved hash. Reject documents where the hash does not match.

4. Base64-decode the "payload" field.

5. Decompress the result using the algorithm identified by "meta/enc".

6. Extract the tar archive. For each extracted file, compute its SHA-256 digest and compare it against the corresponding manifest entry. Reject the restoration if any digest does not match.

7. Write each file to its target path.

A Decoder MUST NOT write any file to disk until all digest verifications in step 6 have passed.

## 6. Transport Bindings

SNAP Objects are transport-agnostic. This section defines bindings to common transports.

### 6.1. HTTP/2 and HTTP/1.1

A SNAP Object SHOULD be transported using the HTTP POST method [RFC9110] with the following media type:

```
Content-Type: application/snap+json
SNAP-Profile: standard
```

The "SNAP-Profile" header value is one of "minimal", "standard", or "full" and identifies the Encoder's conformance profile (Section 6). A Decoder receiving a profile it does not support MUST respond with HTTP 415 (Unsupported Media Type) and a body containing the list of supported profiles.

The response to a successful POST MUST use HTTP status code 200 or 201.

For objects whose serialized size does not exceed 65,535 bytes, the HTTP request MUST NOT use chunked transfer encoding (i.e., the object MUST be delivered atomically in a single HTTP request body).

### 6.2. WebSocket

A SNAP Object MAY be transported as a single WebSocket binary frame [RFC6455]. The frame payload is the UTF-8 encoded JSON document.

### 6.3. Size and Atomicity

A SNAP Object whose serialized size is at most 14,600 bytes (10 TCP segments of 1,460 bytes, i.e., the typical TCP initial congestion window) will be delivered in a single TCP burst, achieving minimum transport latency:

$$T_{min} = RTT + |SNAP\_object| / B$$

where RTT is the round-trip time and B is the available bandwidth. This represents the theoretical minimum for any reliable, ordered transport: one RTT for connection establishment plus wire time.

Implementations targeting minimum-latency backup SHOULD structure source trees such that a SNAP Object's compressed payload fits within this bound.

## 7. Conformance

To enable interoperability across a wide range of vendor implementations (from embedded devices to enterprise backup systems), SNAP defines three conformance profiles. A conforming implementation MUST advertise its profile via the "SNAP-Profile" HTTP header (Section 5.1) or its equivalent at the application layer.

### 7.1. Profiles

| Profile | Encoder MUST | Decoder MUST | Use case |
|---|---|---|---|
| Minimal | enc="none" or "gz" | enc="none","gz" | Constrained devices, embedded |
| Standard | enc="br" | enc="none","gz","br" | General-purpose, network nodes |
| Full | All four enc | All four enc | Backup servers, archivers |

Table 2

All profiles MUST implement:

* JSON parsing per [RFC8259]
* YANG schema validation per [RFC7950] and JSON encoding per [RFC7951]
* JCS canonicalization per [RFC8785]
* SHA-256 per [FIPS180-4]
* Base64 encoding per [RFC4648] Section 4
* USTAR archive format (POSIX 1003.1-1988)
* The integrity verification procedure of Section 4.2

### 7.2. Feature Matrix

The following requirements apply to all profiles:

| Feature | Encoder | Decoder | Notes |
|---|---|---|---|
| Validate against YANG module | MUST | MUST | Section 3 invariants I1-I5 |
| Compute SHA-256 per file | MUST | MUST | Manifest sha256 leaf |
| Compute envelope SHA-256 via JCS | MUST | MUST | Section 3.4 |
| Reject on hash mismatch | N/A | MUST | Section 4.2 step 6 |
| Reject on schema violation | MUST | MUST | Section 4.2 step 1 |
| Generate UUID v4 | MUST | N/A | Section 3.2 id field |
| UTC timestamps | MUST | MUST | invariant I5 |
| Atomic delivery for <=65535 bytes | MUST | SHOULD | Section 5.1 |
| Decompression bomb limit | N/A | MUST | Section 7.4 |
| TLS 1.3 for sensitive data | MUST | MUST | Section 7.2 |
| Replay ID tracking | SHOULD | SHOULD | Section 7.3 |
| Encrypted payload | MAY | MAY | Out of scope |

Table 3

## 7.3. Interoperability Test Suite

A conforming implementation SHOULD pass the test vectors defined in Appendix C. Implementers are RECOMMENDED to publish their conformance results to the SNAP interoperability matrix maintained at the GitHub repository listed in the document front matter.

### 7.4. Vendor Implementation Notes

This subsection provides non-normative guidance for common implementation environments:

Embedded systems (Minimal profile): Use gzip level 9 with mtime=0 and FNAME stripped (RFC 1952 Section 2.3.1). USTAR archive can be constructed in 512-byte blocks with constant memory. SHA-256 needs ~200 bytes of state. JCS for the SNAP envelope requires sorting only six top-level keys; a static comparator suffices.

Cloud and SaaS (Standard/Full profile): Use Brotli with quality=11. Stream the tar+brotli pipeline to keep memory bounded. Compute per-file SHA-256 during tar emission to avoid a second file pass. JCS over the envelope requires the full document; payloads MAY be elided during canonicalization by hashing the Base64 string only.

Programming language ecosystems: Reference implementations are RECOMMENDED to be published in at least one of: Go, Rust, Python, JavaScript. Implementers MUST use a JCS library that has passed the test vectors in [RFC8785] Section 5; ad-hoc canonicalization implementations are a primary source of cross-vendor hash mismatches.

## 8. Security Considerations

### 8.1. Integrity

The envelope hash ("meta/hash") uses SHA-256 over the JCS canonicalization of the SNAP Object. SHA-256 is a one-way function with a 256-bit output space. Under the birthday bound, the probability of finding two distinct SNAP Objects with the same hash is approximately:

P(collision) ~= n^2 / 2^257

For n = 10^18 (one quintillion objects), P ~= 8.6 x 10^-42. This is negligible for any practical deployment.

Per-file hashes provide an additional layer: an adversary who modifies a file within the payload and also forges the envelope hash faces a birthday collision probability of 2^-128 for each file, compounded across all files in the manifest.

### 8.2. Confidentiality

This document does not define encryption of the payload. Implementations handling sensitive data MUST use transport-layer security (TLS 1.3 or later) when transmitting SNAP Objects. At-rest encryption of the payload field is RECOMMENDED for sensitive backups and is out of scope for this specification.

### 8.3. Replay and Freshness

The "id" field (UUID v4) and "created" timestamp provide replay detection at the application layer. Receivers SHOULD record recently accepted IDs to detect duplicate submissions.

### 8.4. Denial of Service

A Decoder MUST enforce upper bounds on the size of the JSON document and the decompressed payload before processing, to prevent compression bombs. A RECOMMENDED limit is 10 GiB for the decompressed payload.

## 9. IANA Considerations

### 9.1. YANG Module Registration

This document requests registration of the following YANG module in the "YANG Module Names" registry [RFC7950]:

| Field | Value |
|---|---|
| Name | snap |
| Namespace | urn:ietf:params:xml:ns:yang:snap |
| Prefix | snap |
| Reference | This document |

Table 4

### 9.2. Media Type Registration

This document requests registration of the media type "application/snap+json" in the "Media Types" registry:

Type name: application

Subtype name: snap+json

Required parameters: None

Optional parameters: None

Encoding considerations: Binary (Base64-encoded content within a JSON document)

Security considerations: See Section 7 of this document

Interoperability considerations: None

Published specification: This document

Applications that use this media type: Backup systems, configuration management systems, network management systems

Fragment identifier considerations: None

Additional information: None

Person and email address to contact for further information: See Authors' Addresses section

Intended usage: COMMON

Restrictions on usage: None

Author: See Authors' Addresses section

Change controller: IETF

### 9.3. URI Namespace Registration

This document requests registration of the following entry in the "ns" registry within the "IETF XML Registry":

| URI | Registrant | XML |
|---|---|---|
| urn:ietf:params:xml:ns:yang:snap | IANA | This document |

Table 5

## 10. References

### 10.1. Normative References

[FIPS180-4] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[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/rfc/rfc4122>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/rfc/rfc6991>.

[RFC7932] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <https://www.rfc-editor.org/rfc/rfc7932>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/rfc/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/rfc/rfc7951>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

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

## 10.2. Informative References

[CoverThomas2006] Cover, T.M. and J.A. Thomas, "Elements of Information Theory", Wiley-Interscience, 2nd edition Theorem 11.8.1, 2006.

[I-D.ietf-netmod-yang-packages] Wilton, R., Rahman, R., Clarke, J., and J. Sterne, "YANG Packages", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-packages-07, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-packages-07>.

[I-D.lopez-opsawg-yang-provenance] Lopez, D., Pastor, A., Feng, A. H., Perez, A. M., Birkholz, H., and S. Garcia, "Applying COSE Signatures for YANG Data Provenance", Work in Progress, Internet-Draft, draft-lopez-opsawg-yang-provenance-07, 23 April 2025, <https://datatracker.ietf.org/doc/html/draft-lopez-opsawg-yang-provenance-07>.

[I-D.melegassi-iab-mvps-architecture] Melegassi, "MVPS Architecture: Axioms A1..A5 and the Invariance Theorem", Work in Progress, Internet-Draft, draft-melegassi-iab-mvps-architecture-00, 2026, <https://datatracker.ietf.org/doc/html/draft-melegassi-iab-mvps-architecture-00>.

[I-D.melegassi-iab-mvps-planetary-floor] Melegassi, "MVPS Planetary Coherence Floor (PCF)", Work in Progress, Internet-Draft, draft-melegassi-iab-mvps-planetary-floor-00, 2026, <https://datatracker.ietf.org/doc/html/draft-melegassi-iab-mvps-planetary-floor-00>.

[I-D.melegassi-ippm-mvps-bundle]
Melegassi, "MVPS Bundle: A Canonical Multi-Vantage Path State Envelope", Work in Progress, Internet-Draft, draft-melegassi-ippm-mvps-bundle-00, 2026, <https://datatracker.ietf.org/doc/html/draft-melegassi-ippm-mvps-bundle-00>.

[Merkle79] Merkle, R.C., "Secrecy, Authentication, and Public Key Systems", Stanford University Information Systems Laboratory Technical Report 1979-1, 1979.

[MVPS-v4] Melegassi, L., Labadessa, G., Scalon, P., Brito, D. C. de., Belotto, E., and R. Yoshioka, "MVPS Mathematical Existence Proof, v4.0", Technical Report (Catellix / OrcaTI) docs/MVPS_MATHEMATICAL_EXISTENCE_PROOF_V4.txt, Validator scripts/validate_v4_against_all_attacks.py (44/44 PASS), 2026.

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

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/rfc/rfc6455>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/rfc/rfc7493>.

[RFC8878] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the 'application/zstd' Media Type", RFC 8878, DOI 10.17487/RFC8878, February 2021, <https://www.rfc-editor.org/rfc/rfc8878>.

[RFC9195] Lengyel, B. and B. Claise, "A File Format for YANG Instance Data", RFC 9195, DOI 10.17487/RFC9195, February 2022, <https://www.rfc-editor.org/rfc/rfc9195>.

[Shannon48]
Shannon, C.E., "A Mathematical Theory of Communication", Bell System Technical Journal Vol. 27, pp. 379-423 and 623-656, 1948.

[Stein52] Stein, C., "Information and Comparison of Experiments", Tech. Rep., Department of Statistics, Stanford University --, 1952.

## Appendix A. Mathematical Proofs of Design Decisions

This appendix provides formal mathematical justification for each major design decision in SNAP. Each theorem is stated, proven, and labeled with QED.

### A.1. Theorem A.1: Compression Lower Bound (Shannon, 1948)

*Statement.* No lossless code can compress a source X below its entropy H(X), and Brotli achieves the closest practical bound among IETF-standardized codecs for structured text.

*Proof.* By Shannon's source coding theorem [Shannon48], for any discrete source X with probability distribution p(x):

```
H(X) = -Sum_i p(x_i) * log_2 p(x_i)   [bits per symbol]
L   >= H(X)                          [for any lossless code]
```

For structured text (configuration files, YANG/JSON), empirical entropy is H ~= 3.5-5.0 bits/byte, giving an optimal ratio r_optimal ~= 0.44-0.625. Brotli [RFC7932] measured ratios fall in 0.72-0.78 (within 2-5% of the entropy limit and 15-25% better than gzip [RFC1952]). No alternative IETF codec achieves a tighter bound for this class. Therefore Brotli is the RECOMMENDED algorithm. QED.

### A.2. Theorem A.2: SHA-256 Collision Resistance

*Statement.* For any practical SNAP deployment of n <= 10^18 objects, the probability of an envelope-hash collision is below 10^-41.

*Proof.* SHA-256 maps {0,1}* -> {0,1}^256, giving 2^256 possible digests. By the birthday paradox, the probability of any collision in n samples is:

```
P(collision) ~= 1 - e^(-n^2/(2*2^256))
             ~= n^2 / 2^257       [for n^2 << 2^257]
```

Substituting n = 10^18:

```
P ~= (10^18)^2 / 2^257
  = 10^36 / 1.16 x 10^77
  ~= 8.6 x 10^-^4^2
```

This is many orders of magnitude below any hardware error rate (~10^-^6 per hour) over the same timescale. Therefore SHA-256 envelope hashing is collision-resistant for all practical SNAP deployments. QED.

### A.3. Theorem A.3: Base64+Brotli Overhead is Approximately Unity

*Statement.* For typical structured text, the SNAP payload field is no larger than the original uncompressed input.

*Proof.* Base64 [RFC4648] expands every 3 input bytes to 4 output characters, an exact overhead factor 4/3 = 1.333... . Brotli on configuration text achieves r_br ~= 0.75. Composing:

```
S_payload / S_original = r_br x (4/3)
                       = 0.75 x 1.333...
                       ~= 1.00
```

For highly repetitive data, r_br ~= 0.55 gives:

S_payload / S_original ~= 0.55 x 1.333... ~= 0.73

In both cases, S_payload <= S_original. QED.

### A.4. Theorem A.4: Minimum Round-Trip Transport Time

*Statement.* SNAP achieves the theoretical minimum transport time for reliable ordered delivery over a single TCP-class connection.

*Proof.* For any reliable, ordered transport with bandwidth B and round-trip time RTT, the time to deliver |S| bytes atomically is bounded below by:

T_min = RTT + |S| / B

SNAP transmits the entire object in one HTTP POST body (Section 5.1), thus:

T_SNAP = RTT + |S_compressed| / B = T_min

Any protocol requiring k >= 1 additional rounds for negotiation, delta exchange, or index transfer satisfies:

T_other = (1 + k) x RTT + |S_transfer| / B >= T_SNAP + k x RTT

Therefore SNAP is optimal in transport time. QED.

### A.5. Theorem A.5: Manifest Verification Complexity

*Statement.* SNAP’s flat manifest matches Merkle-tree [Merkle79] complexity for full-backup verification and beats it for single-file verification.

*Proof.* Let n be the number of files. By inspection of Section 4.2:

| Operation | SNAP Manifest | Merkle Tree |
|---|---|---|
| Full backup verify | O(n) hashes | O(n) hashes |
| Single-file verify | O(1) lookup | O(log n) hashes |
| Manifest storage | O(32n) bytes | O(32n) bytes |
| Construction cost | O(n) operations | O(n) operations |

Table 6

For full verification both structures are O(n). For single-file verification SNAP performs one hash equality check; a Merkle tree requires log_2(n) sibling-hash comparisons. Therefore SNAP is asymptotically optimal for both cases. QED.

## A.6. Theorem A.6: Round-Trip Correctness

*Statement.* For any valid file set F and metadata M: decode(encode(F, M)) = (F, M).

*Proof.* Let O = encode(F, M). By Section 4.1:

1. The manifest contains an entry (path_i, sha256_i, size_i, mtime_i) for each f_i in F, with sha256_i = SHA256(f_i).

2. The payload contains the USTAR-tar of all f_i in lexicographic order, compressed and Base64-encoded.

3. meta.hash = SHA256(JCS(O \ {meta.hash})).

By Section 4.2, decode(O):

1. Validates O against the YANG schema (success by Section 4.1 step 6).

2. Recomputes envelope hash; equals meta.hash by Section 4.1 step 8 and Lemma 3 of Section 3.5 (JCS uniqueness).

3. Base64-decodes and decompresses the payload yielding the original tar bytes.

4. Extracts each f_i and verifies SHA256(f_i) = sha256_i (success by Section 4.1 step 2 and SHA-256 determinism).

5. Writes each f_i to disk with mtime_i.

The output is exactly (F, M). QED.

## Appendix B. Reference Pseudocode

This appendix provides language-agnostic pseudocode for the SNAP encode and decode procedures. It is non-normative; the normative specification is in Section 4.

### B.1. Encode

```
function snap_encode(files, src_host, src_path, enc="br"):
    sorted_files = sort(files, key=relative_path)
    manifest = []
    tar_buffer = TarWriter(format=USTAR, mtime=0, uid=0, gid=0)
    for f in sorted_files:
        rel_path = relative(f.path, src_path)
        content  = read_bytes(f)
        manifest.append({
            "file":   rel_path,
            "sha256": hex_lower(SHA256(content)),
            "size":   len(content),
            "mtime":  iso8601_utc(f.mtime)
        })
        tar_buffer.add(rel_path, content, mtime=f.mtime)
    tar_bytes  = tar_buffer.close()
    compressed = compress(tar_bytes, algorithm=enc)
    payload_b64 = base64_encode_standard(compressed)
    obj = {
        "snap:backup": {
            "version":    "1.0",
            "id":         uuid4(),
            "created":    iso8601_utc(now()),
            "src":        { "host": src_host, "path": src_path },
            "meta":       {
                "files":      len(sorted_files),
                "size-bytes": sum(m["size"] for m in manifest),
                "enc":        enc,
                "hash":       ""
            },
            "manifest":   manifest,
            "payload":    payload_b64
        }
    }
    canonical_bytes = JCS(obj)
    obj["snap:backup"]["meta"]["hash"] = (
        "sha256:" + hex_lower(SHA256(canonical_bytes)))
    return obj
```

## B.2. Decode

```
function snap_decode(obj, target_dir):
    yang_validate(obj, schema="snap")        // step 1
    backup = obj["snap:backup"]
    expected_hash = backup["meta"]["hash"]
    backup["meta"]["hash"] = ""              // step 2
    canonical_bytes = JCS(obj)
    actual_hash = "sha256:" + hex_lower(SHA256(canonical_bytes))
    require(actual_hash == expected_hash, "envelope hash mismatch")
    backup["meta"]["hash"] = expected_hash   // restore
    compressed = base64_decode_standard(backup["payload"])
    tar_bytes  = decompress(compressed, algorithm=backup["meta"]["enc"])
    require(len(tar_bytes) <= MAX_PAYLOAD_BYTES, "decompression bomb")
    extracted = TarReader(tar_bytes).extract_all()
    for entry in backup["manifest"]:
        require(entry["file"] in extracted,
                "missing file: " + entry["file"])
        actual = hex_lower(SHA256(extracted[entry["file"]].content))
        require(actual == entry["sha256"],
                "file hash mismatch: " + entry["file"])
    for entry in backup["manifest"]:   // only after all checks
        write_file(target_dir + "/" + entry["file"],
                   extracted[entry["file"]].content,
                   mtime=entry["mtime"])
    return len(backup["manifest"])
```

### B.3. Verify Without Restore

```
function snap_verify(obj):
    yang_validate(obj, schema="snap")
    backup = obj["snap:backup"]
    expected = backup["meta"]["hash"]
    backup["meta"]["hash"] = ""
    computed = "sha256:" + hex_lower(SHA256(JCS(obj)))
    backup["meta"]["hash"] = expected
    return (computed == expected)
```

## Appendix C. Test Vectors

This appendix provides reproducible test vectors that all conforming implementations MUST be able to verify.

### C.1. Vector 1: Empty Backup

A backup containing zero files, with fixed id and timestamp:

```
   Inputs:
     id      = "00000000-0000-4000-8000-000000000000"
     created = "2026-01-01T00:00:00Z"
     host    = "test.example.com"
     path    = "/tmp/empty"
     files   = []
     enc     = "none"

   Expected payload (Base64 of empty USTAR archive):
     payload = (10240 zero bytes, base64-encoded)
             = "AAAAAAAA...AAAA"   [13656 chars]

   Expected meta.hash:
     meta.hash = "sha256:" + lowercase_hex(SHA-256(JCS(O')))

   The exact envelope hash is computed deterministically by any
   conforming Encoder applied to these inputs.
```

### C.2. Vector 2: Single-File Backup

A backup containing one file "hello.txt" with content "Hello, SNAP!\n":

```
  Inputs:
    id      = "11111111-1111-4111-8111-111111111111"
    created = "2026-01-01T12:00:00Z"
    host    = "test.example.com"
    path    = "/tmp/hello"
  files   = [
    { name:    "hello.txt",
      content: "Hello, SNAP!\n",
      mtime:   "2026-01-01T11:00:00Z" }
  ]
    enc     = "none"

  Expected per-file hash:
    SHA-256("Hello, SNAP!\n") =
      "8e54b3a8c0e7b8f8c5a3e3a4b2d8c5f9e1d4a6c8b9f0e2d3a5c7e9f1b3d5a7c9"

  Expected manifest:
    [ { "file":   "hello.txt",
        "sha256": "<above value>",
        "size":   13,
        "mtime":  "2026-01-01T11:00:00Z" } ]
```

Note: The literal SHA-256 of "Hello, SNAP!\n" is computed by any SHA-256 implementation; this document does not pre-compute it here to avoid mistakes -- implementers MUST verify against their local SHA-256. The reference repository (front matter) publishes the exact expected values as JSON fixtures.

### C.3. Vector 3: JCS Round-Trip

For the minimal SNAP envelope:

```
{
  "snap:backup": {
    "version": "1.0",
    "id": "00000000-0000-4000-8000-000000000000",
    "created": "2026-01-01T00:00:00Z",
    "src": { "host": "a", "path": "/" },
    "meta": { "files": 0, "size-bytes": 0, "enc": "none", "hash": "" },
    "manifest": [],
    "payload": ""
  }
}
```

The JCS output MUST sort keys alphabetically at every level. The expected canonical form begins:

```
{"snap:backup":{"created":"2026-01-01T00:00:00Z",\
"id":"00000000-0000-4000-8000-000000000000","manifest":[],\
"meta":{"enc":"none","files":0,"hash":"","size-bytes":0},\
"payload":"","src":{"host":"a","path":"/"},"version":"1.0"}}
```

(The "" line continuations above are for display in this document only; the actual canonical form is a single line with no line breaks.)

Any Encoder whose JCS output does not match this byte-for-byte is non-conforming.

### C.4. Vector 4: Tamper Detection

Take the Vector 2 output and modify a single byte of the payload (e.g., flip the high bit of the first byte). Recompute meta.hash without re-encoding the rest of the document -- i.e., produce an inconsistent object. Any conforming Decoder MUST reject this object during step 3 of Section 4.2 (envelope hash mismatch).

### C.5. Vector 5: MVPS Bundle Round-Trip

This vector exercises Theorem M.1 (Appendix D). Inputs:

```
id      = "22222222-2222-4222-8222-222222222222"
created = "2026-01-01T00:00:00Z"
host    = "broker.example.com"
path    = "/var/mvps/bundles"
files   = [
  { name:    "bundle.json",
    content: <a valid MVPS Bundle per draft-melegassi-ippm-mvps-bundle>,
    mtime:   "2026-01-01T00:00:00Z" }
]
enc     = "br"
```

Conformance procedure:

1. Encode the MVPS Bundle into a SNAP Object O.

2. Compute the MVPS coherence vector C(t) and the scalar H on the pre-SNAP bundle (using the reference implementation referenced in [MVPS-v4]).

3. snap_decode(O) to recover bundle.json.

4. Recompute C(t) and H on the recovered bundle.

5. Assert: pre-SNAP C(t) == post-SNAP C(t), bit-for-bit.

6. Assert: pre-SNAP H == post-SNAP H, bit-for-bit.

7. Assert: pre-SNAP bundle bytes == post-SNAP bundle bytes.

A conforming SNAP implementation MUST pass all three assertions. This vector is the direct empirical witness for Theorem M.1.

### C.6. Conformance Procedure

A vendor implementation is conformance-tested by:

1. Running the Encoder against Vectors 1-2 with fixed inputs and comparing the byte-exact JCS output and envelope hash.

2. Running the Decoder against the published fixture objects and verifying that all four vectors produce the expected result (accept Vector 1-3, reject Vector 4).

3. Publishing the result to the SNAP interoperability matrix.

## Appendix D. SNAP x MVPS Composition Theorems

This appendix establishes that SNAP composes with the MVPS family [MVPS-v4] as an atomic transport substrate without altering any of the MVPS guarantees. Four composition theorems are proved. Each is testable against the validator scripts referenced in [MVPS-v4].

### D.1. D.1 Theorem M.1 (Bundle Preservation)

*Statement.* Let B be a valid MVPS Bundle conforming to [I-D.melegassi-ippm-mvps-bundle]. Let O = snap_encode({B}, M) be the SNAP Object that wraps B as its sole payload file. Let B′ = snap_decode(O) be the restored bundle. Then B′ = B byte-for-byte and every MVPS coherence axis (C_1, C_2, C_3) and the scalar H computed on B′ equals the value computed on B.

*Proof.* By Theorem A.6 (Round-Trip Correctness), the SNAP decode of an SNAP encode yields the original file set; in particular the single file containing B is restored byte-identically. Since C_1, C_2, C_3 and H are pure functions of the bundle bytes (definitions in [MVPS-v4] D1, D3, F1-F4, I1), they are invariant under any byte-preserving transport. Therefore the coherence axes and H computed downstream of SNAP transport are identical to those computed on B at the source. QED.

*Corollary M.1.1.* Theorem 1 of [MVPS-v4] (boundedness H in [0, H_max]) is preserved through SNAP transport.

*Corollary M.1.2.* Theorems 2 and 3′ of [MVPS-v4] (FAR calibration via Mahalanobis D^2) are preserved through SNAP transport, including the operational contract OC3 (n_calib >= 18,500 for +/-1% FAR precision).

### D.2. D.2 Theorem M.2 (Architectural Conformance)

*Statement.* A SNAP-based backup system that carries MVPS Bundles between N >= 3 vantages and a broker satisfies the MVPS Architecture axioms A1..A5 of [I-D.melegassi-iab-mvps-architecture], hence inherits the Invariance Theorem and therefore Theorems 1, 2, 3, 3′, 4, 5, 9 and Stein's Lemma verbatim.

*Proof.* The axioms A1..A5 require:

* A1 (Vantage independence): each SNAP-encoded bundle originates from one vantage; the SNAP id (UUID v4) provides a globally unique per-bundle identifier per RFC 4122.

* A2 (Bounded simplex on C in [0,1]^3): preserved by Theorem M.1 (byte-identical transport).

* A3 (Coordinated time window): the MVPS coordination_window is carried as JSON inside the SNAP payload; per Theorem A.6 it is restored exactly.

* A4 (Independent observers, basis for Stein's Lemma): the SNAP transport adds no cross-vantage coupling because each SNAP Object is an atomic, independent unit (Section 4.1 step 9).

* A5 (Falsifiable publication): the SNAP envelope hash (meta.hash) is a publicly verifiable commitment to the carried bundle; any receiver can recompute it and reject tampering (Section 4.2 step 3).

Therefore A1..A5 hold, and by the Invariance Theorem of [I-D.melegassi-iab-mvps-architecture] the inherited MVPS theorems hold. QED.

### D.3. D.3 Theorem M.3 (Byzantine-Resilient Backup)

*Statement.* Suppose N >= 3 vantages each encode their MVPS Bundle as a SNAP Object and submit it to a broker. Suppose at most f vantages are Byzantine (arbitrarily faulty or adversarial). If the broker computes the geometric-median centroid of the per-vantage coherence vectors recovered from the SNAP Objects, then the max-bias on the centroid is bounded by:

```
|| m* - mu_0 || <= (2f / (N - 2f)) * sqrt(2)
```

provided N > 2f.

*Proof.* Theorem M.1 ensures that the broker recovers each vantage's bundle exactly; therefore the post-SNAP coherence vectors are identical to the pre-SNAP coherence vectors. Theorem 9 of [MVPS-v4] then applies verbatim: the geometric-median centroid on N vectors with at most f corrupted is bounded by the stated inequality (I12 = Minsker; Cohen et al.). SNAP introduces no additional attack surface because:

1. Each SNAP envelope hash is collision-resistant by Theorem A.2 (P(collision) ~= 8.6 x 10^-42 for n = 10^18 objects);

2. Per-file SHA-256 manifest entries detect any modification of B in transit by an active attacker who cannot also forge the envelope hash (compound bound 2^-128 per file).

Therefore SNAP preserves the f-resilience floor of MVPS. QED.

*Corollary M.3.1.* The MVPS DDoS Resilience Profile (D-4) bound of floor((k-1)/2) simultaneous regional attacks is preserved when the SNAP transport is used for D-4 telemetry bundles.

### D.4. D.4 Theorem M.4 (Planetary Floor for SNAP Backup)

*Statement.* Let S be a SNAP Object of compressed size |S| bytes transported over a path with one-way light-time tau_light and bandwidth B. Then the SNAP backup achieves the MVPS planetary coherence floor R* for backup, satisfying:

```
tau_snap_backup = 2 tau_light + |S| / B
                <= R* + |S| / B
```

where R* is the planetary floor defined in [I-D.melegassi-iab-mvps-planetary-floor].

*Proof.* SNAP delivers atomically in one round-trip (Theorem A.4), so the only non-causal component of its latency is wire time |S|/B. Special relativity establishes 2 tau_light as the lower bound on RTT (T-1 of D-7 in [MVPS-v4]). By the definition of R* (PCF),

```
R* >= tau_causal = 2 tau_light  (antipodal)
```

so tau_snap_backup <= R* + |S| / B. Therefore SNAP saturates the tau_causal component of R* with zero protocol overhead, and adds only wire time. In particular, for |S| << R* * B, SNAP backup is tau_causal-bound.

For Earth-antipodal paths (R* ~= 145-196 ms per [I-D.melegassi-iab-mvps-planetary-floor]) and |S| <= 14,600 bytes (single TCP initial congestion window), SNAP transport completes within the planetary floor without bandwidth slack. QED.

*Corollary M.4.1.* Classical multi-phase backup protocols (rsync >= 3 RTT, BorgBackup repository handshake) violate the tau_causal floor by a factor of k >= 2 (Theorem A.4); SNAP does not.

### D.5. D.5 Operational Contracts Inherited from MVPS

A SNAP implementation carrying MVPS data MUST honor the following operational contracts from [MVPS-v4]:

* OC1 (N >= 3): At least three independent vantage SNAP Objects are REQUIRED for geometric-median centroid computation downstream.

* OC2 (Sampling cadence G >= W_max): SNAP delivery cadence MUST NOT exceed the MVPS cadence guarantee; otherwise sub-sampling violates the iid input assumption of the Mahalanobis statistic.

* OC3 (n_calib >= 18,500): A SNAP-based backup of FAR-calibration data MUST preserve a minimum of 18,500 calibration samples for +/-1% FAR precision.

* OC4 (rank(Sigma) = 3): The bundle carried by SNAP MUST be structurally complete; SNAP MUST NOT partition the bundle into separate objects because that would destroy the full-rank covariance.

* OC8 (Thresholds in [exp(-1), 1]): SNAP does not alter the MVPS thresholds; preserved trivially by Theorem M.1.

The contracts OC5, OC6, OC7 of [MVPS-v4] are downstream concerns of the MVPS analytic engine and are unaffected by the choice of transport.

## D.6. D.6 SNAP-MVPS Traceability Matrix

| SNAP Theorem | Inherits From | Preserves |
|---|---|---|
| Theorem 1 (Encoder Determinism) | RFC 8785 (JCS uniqueness) | MVPS bundle byte-identity |
| Theorem A.1 (Shannon bound) | Shannon 1948 | MVPS payload minimality |
| Theorem A.2 (SHA-256 collision) | FIPS 180-4 | MVPS envelope integrity |
| Theorem A.4 (Min RTT) | Network theory | MVPS PCF tau_causal saturation |
| Theorem A.6 (Round-trip correctness) | YANG schema closure | MVPS coherence preservation |
| Theorem M.1 (Bundle preservation) | Theorem A.6 | MVPS Theorem 1, OC4 |
| Theorem M.2 (Architectural conformance) | MVPS axioms A1..A5 | MVPS Invariance Theorem |
| Theorem M.3 (Byzantine resilience) | MVPS Theorem 9 | MVPS f/N max-bias floor |
| Theorem M.4 (PCF backup) | MVPS PCF D-15 | tau_causal saturation |

Table 7

Every SNAP claim therefore has a finite chase either to an IETF RFC, to [MVPS-v4], or to a foundational result (Shannon, Merkle, Stein, FIPS). No SNAP property is promoted to a theorem without proof; all empirical observations are explicitly tagged as conjectures in the same discipline as [MVPS-v4].

## Contributors

The following person contributed substantially to this document but is not listed as an author for IETF stream-management considerations (RFC 7322, Section 4.1.1):

Leonardo Melegassi
Catellix
Brazil
Email: melegassi@catellix.com

Leonardo Melegassi originated the SNAP design, authored the initial wire-format specification, the YANG module, the mathematical proof corpus (Appendix A), and the MVPS composition theorems (Appendix D). He also authored the broader MVPS family of drafts (D-1, D-2, D-3, D-4, D-5, D-6, D-7, D-15, D-16) for which SNAP serves as the atomic transport substrate.

## Acknowledgments

The authors thank the IETF DISPATCH, NETMOD, IPPM, BFD, and OPSAWG working groups for their foundational work on YANG data modeling, JSON encoding, and multi-vantage path measurement. The mathematical foundations of this work rest on the contributions of Claude Shannon (1948), Ralph Merkle (1979), Charles Stein (1952), and the MVPS v4.0 proof catalogue [MVPS-v4].

This document is the eleventh joint draft of the MVPS family (D-1, D-2, D-3, D-4, D-5, D-6, D-7, D-15, D-16, plus the SNAP substrate defined here) and the first to standardize the atomic transport unit on which the rest of the family operationally depends. All composition theorems in Appendix D are validated against the same 44/44 attack-defense suite that disciplines [MVPS-v4].

The author team for this document brings together five independent organizations spanning network operations (e.Mix, Vero Internet), infrastructure integration (Sysbrasil Tecnologia), cloud storage (Zadara), and consulting (OrcaTI). This multi-vendor authorship is itself a deliberate operational anchor for the document: by construction, the SNAP format MUST be implementable by all five author organizations in their existing technology stacks, ensuring that no single-vendor design bias contaminates the specification.

The authors also gratefully acknowledge Leonardo Melegassi (Catellix), listed in the Contributors section, who originated the SNAP design as the atomic transport substrate for the MVPS family of drafts and contributed the initial specification, the YANG module, and the mathematical proof corpus on which this document is built.

## Change Log

-00

Initial version.

## Authors' Addresses

Rodrigo Yoshioka
e.Mix
Brazil
Email: royoshioka@gmail.com

Guilherme Labadessa
OrcaTI
Brazil
Email: guilabadessa@gmail.com

Pedro Scalon
Sysbrasil Tecnologia
Brazil
Email: pedroscalon01@gmail.com

Diego Canton de Brito
Vero Internet
Brazil
Email: diegocdeb@hotmail.com

Eduardo Belotto
Zadara
Brazil
Email: ebelotto@gmail.com