Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 20 September 2026

S. Bellis
Unheaded

19 March 2026

Unheaded: Protocol Foundation — A Mapped Data Bus over IPv6 Hop-by-Hop Options

draft-bellis-unheaded-protocol-foundation-00

Abstract

The Unheaded Protocol defines a mapped data bus model that transforms IPv6 packets into addressable memory by encoding a small register file directly in the IPv6 Hop-by-Hop Options extension header.

We introduce a 20-byte Monad (register file) that carries program state through the network. At each hop, a BPF program (the Shim) performs computation on the Monad. The packet itself becomes the working memory, using exponent-encoded fields to pack rich metadata into the IPv6 option while remaining fully backward-compatible with existing networks.

To support programs larger than what fits in a single Monad, we introduce Wotan, a memory and I/O bus that bridges Monad computation to per-flow ring-buffer storage and external topics. This decouples the Monad (pure, 20-byte compute) from memory (Wotan’s configurable data planes).

This memo extends the packet format with two additional capabilities: (1) Kingdom Mode Address Reclamation, which recovers up to 224 bits of deterministic address space from IPv6 addresses within a controlled L2 fabric for use as extended computational and cryptographic registers; and (2) Post-Quantum Cryptographic Identity Binding, which cryptographically binds each service identifier in the Monad to a post-quantum keypair via the Sophia dictionary system, providing quantum-resistant authentication of per-packet metadata without increasing wire overhead.

This memo defines the normative packet format, exponent-encoding scheme, per-hop processing semantics, address reclamation model, post-quantum identity binding, optional chaos injection for resilience testing, and the complete computational model (Turing-complete with memory paging).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Statement

Classical networking separates computation from data. Computation happens in applications. Data flows through the network as opaque byte streams. This creates an expensive impedance mismatch of serialization, deserialization, protocol translation, middleware, sidecars, and proxies.

The Unheaded Protocol inverts this model: the packet carries computational state. A 20-byte register file (the Monad) is read and written by BPF programs at each hop. The packet functions as working storage of a distributed computation that executes at each operator-controlled node in the path.

Within a Limited Domain [RFC8799] — a single AS, corporation, or container fleet where every hop is operator-controlled — the protocol provides:

* Inline state: Application state is carried in the packet without intermediate serialization.

* Per-hop compute: Each hop reads the Monad state directly from the packet without requiring separate queries or network round-trips.

* Causal ordering: Anamnesis events are ordered by packet arrival at each hop, enabling causal reconstruction independent of wall-clock synchronization.

* Quantum-resistant identity: Service identifiers are cryptographically bound to post-quantum keypairs.

* Address reclamation: Deterministic address prefix bits within the Limited Domain are reclaimed as extended computational registers.

1.2. Scope and Applicability

This specification defines the complete protocol for deploying a mapped data bus over IPv6 Hop-by-Hop Options. The protocol is applicable to Limited Domains (RFC 8799) where all intermediate nodes are operator-controlled and IPv6 Hop-by-Hop option processing is enabled.

The protocol is NOT applicable to the public Internet, where intermediate routers may drop Hop-by-Hop options (RFC 9098, RFC 9673), nor to paths containing routers that do not process such options.

1.3. Cross-References to Companion Specifications

This document is part of the Unheaded Protocol specification family:

* *Sophia Dictionary Format* [SOPHIA]: Defines the serialization, storage, and distribution mechanism for semantic metadata including sub-dictionary type systems and QPACK compression headers for dictionary entries.

* *Wotan Memory Protocol* [WOTAN]: Defines the memory and I/O bus including error code taxonomy, helper return codes, and error recovery procedures.

The three specifications are designed to be read together. The Foundation specification defines the wire format and per-hop processing model. Sophia defines the semantic layer that gives meaning to exponent-encoded fields. Wotan defines the memory model that provides per-flow state beyond the 20-byte Monad.

2. Requirements Language

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

3. Terminology

This document uses the following terms:

Monad: The 20-byte register file embedded in the IPv6 Hop-by-Hop option header. It travels with the packet and is read and modified at each hop.

Shim: A BPF program that executes at each hop, reading the Monad from the packet, performing computation, and writing back to the packet.

Shield: The packet boundary logic at ingress (first hop) that stamps the packet into existence by adding the Hop-by-Hop option, and at egress (last hop) that strips the option and commits the packet to the external network.

Wotan: The memory and I/O bus that provides per-flow ring buffers, persistent storage (WAL), and bridging to external topics.

Anamnesis: The non-blocking event log implemented as a BPF ring buffer, recording packet events for observability and historical replay.

Sophia: The exponent dictionary system. BPF hash maps in kernel space, structured tables in userspace. Hot-swappable vocabulary that assigns meaning to exponent-encoded field values.

Exponent Encoding: A compositional scheme for packing metadata: each field stores a signed exponent, and the actual value is reconstructed as base^exponent * multiplier, where base and multiplier are defined per-field in Sophia.

Limited Domain: A network boundary (typically a single AS, corporation, or container fleet) where the Unheaded Protocol is deployed end-to-end. All hops within the domain are operator-controlled. Defined in [RFC8799].

Kingdom Mode: An operational mode within a Limited Domain where the

IPv6 ULA prefix is known a priori, enabling deterministic address prefix bits to be reclaimed as extended computational space.

4. Protocol Overview

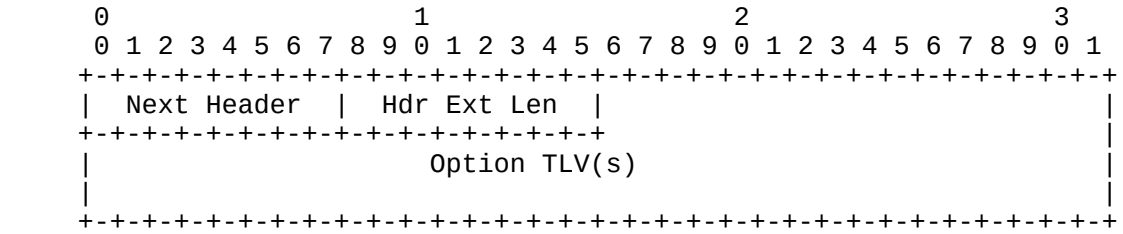
The Unheaded Protocol adds an IPv6 Hop-by-Hop Options extension header to packets at the Limited Domain ingress. The option contains a 20-byte register file (Monad) that carries computational state. At each operator-controlled hop within the domain, a BPF program (Shim) reads the Monad, performs computation, and writes the updated Monad back to the packet before forwarding it to the next hop. At egress, Shield removes the option and forwards a clean IPv6 packet to the external network.

The Monad's fields are exponent-encoded, allowing rich metadata to be packed into 8-bit signed values. Field semantics are defined by Sophia [SOPHIA], a dictionary system stored as BPF hash maps. Ring buffers (Anamnesis) record packet events at each hop for observability. Per-flow state beyond the 20-byte Monad is managed by Wotan [WOTAN].

5. Packet Format

5.1. IPv6 Hop-by-Hop Extension Header

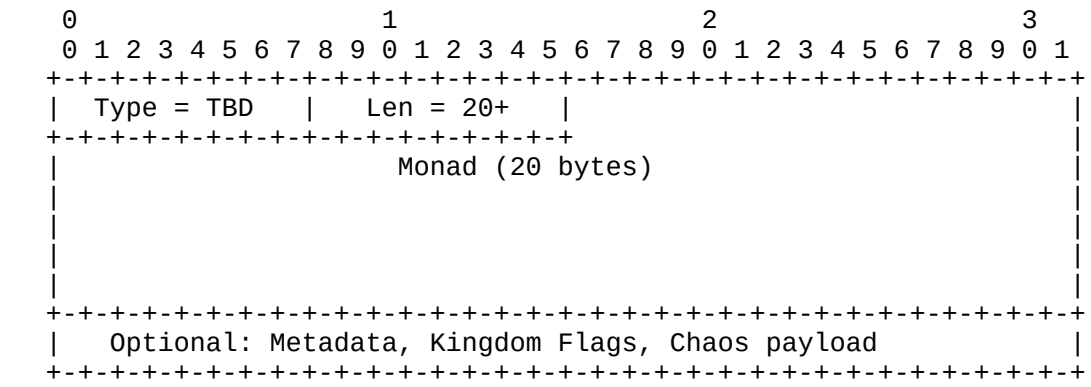
The Monad is carried in a single IPv6 Hop-by-Hop Options extension header option, formatted per RFC 8200 Section 4.



The Hop-by-Hop extension header MUST precede all other extension headers. It MUST be processed at each hop per RFC 8200 Section 4.3 and RFC 9673.

5.2. Option Type

The Monad is encoded as a single option TLV within the Hop-by-Hop extension header:



Type: To be assigned by IANA. The high-order two bits MUST be 00 (skip on unrecognized). The third bit MUST be 1 (option data may change en-route), as the Monad is modified by per-hop processing. The resulting format is 001xxxxx. Suggested value: 0x3E.

Len: The length of the option payload in octets (not including the Type and Len octets). Minimum value is 20 (Monad only); maximum is 255.

5.3. Monad Register File

The Monad is a 20-byte register file with the following layout. All multi-byte fields MUST be encoded in network byte order (big-endian).

Offset	Size	Field	Type	Description
0x00	1	version	raw uint8	Protocol version (current: 0x01)
0x01	1	src_service_id	exponent	Source service (Sophia lookup)
0x02	1	dst_service_id	exponent	Destination service (Sophia)
0x03	1	hop_count	raw uint8	Decremented at each hop (TTL-like)
0x04	1	qos_class	exponent	QoS classification
0x05	1	flow_action	exponent	Action directive
0x06	1	circuit_state	exponent	Circuit breaker state
0x07	1	flags	raw uint8	Bitfield (see Flags section)
0x08	2	latency_hint	raw uint16	Latency hint in microseconds
0x0A	1	deploy_ring	exponent	Deployment ring
0x0B	1	mesh_flags	exponent	Mesh-level flags
0x0C	1	src_prefix_lo	raw uint8	Source routing prefix low octet
0x0D	1	dst_prefix_lo	raw uint8	Destination routing prefix low octet
0x0E	4	scratch[0-3]	raw uint8	Scratch registers (4 bytes)
0x12	2	checksum	raw uint16	CRC-16/CCITT over bytes 0x00-0x13 (with checksum zeroed)
Total: 20 bytes (0x14)				

WIRE FORMAT FROZEN: The Monad register file layout defined above is FROZEN at version 0x01. No changes to byte offsets, field sizes, field ordering, or total size (20 bytes) are permitted in this or future drafts of version 0x01. Any modification to the wire format constitutes a new protocol version (0x02 or later) and requires a new IANA version registry entry.

version: The protocol version, an unsigned 8-bit integer. This document specifies version 0x01 (current). Version 0 is reserved and MUST NOT be used. Future versions (2+) are currently undefined.

Version Checking (NORMATIVE): Receivers MUST drop packets with unknown version fields immediately with no error code generation or fallback processing. Specifically, if the version field != 0x01, the packet MUST be dropped immediately (silent drop). No version negotiation, no fallback, no compatibility shim. This eliminates parser divergence attacks (X4) by ensuring all implementations reject unknown versions identically.

src_service_id: An exponent-encoded field identifying the source service. Semantics are defined by Sophia dictionary lookup [SOPHIA]. Implementations MUST NOT assume fixed semantics; the meaning is program-defined per service.

dst_service_id: An exponent-encoded field identifying the destination service. Semantics are defined by Sophia dictionary lookup [SOPHIA].

hop_count: An unsigned 8-bit counter, initially set to a deployment-defined hop limit (default 64) at Shield ingress. Each hop MUST check whether hop_count equals 0; if so, the packet MUST be dropped and an EVENT_ANOMALY MUST be emitted. Otherwise, the hop MUST decrement hop_count by 1 before forwarding.

trace_id: Flow trace correlation is derived from the IPv6 Flow Label (RFC 6437). The 20-bit Flow Label set by Shield at ingress serves as the trace correlation identifier. Shim programs MUST NOT modify the Flow Label.

qos_class: An exponent-encoded field specifying the Quality of Service class. Semantics are defined by Sophia [SOPHIA].

flow_action: An exponent-encoded field specifying the action directive for this packet. Examples include trace, sample, mirror, rate-limit, or drop. Semantics are defined by Sophia [SOPHIA].

circuit_state: An exponent-encoded field specifying the circuit breaker state (open, closed, half-open). Semantics are defined by Sophia [SOPHIA].

flags: An 8-bit bitfield controlling protocol behavior. See Section 5.2.

latency_hint: A 16-bit hint encoding the per-hop latency target, in network byte order. Interpretation is deployment-defined.

deploy_ring: An exponent-encoded field specifying the deployment ring (canary, staging, production). Semantics are defined by Sophia [SOPHIA].

mesh_flags: An exponent-encoded field specifying mesh-level flags (NAT type, direction, encryption). Semantics are defined by Sophia [SOPHIA].

src_prefix_lo: The low-order octet of the source routing prefix, used by Shim programs for routing optimization. Set by Shield at ingress from Sophia policy.

dst_prefix_lo: The low-order octet of the destination routing prefix, used by Shim programs for routing optimization. Set by Shield at ingress from Sophia policy.

scratch: Four bytes of per-hop scratch storage (scratch[0]-scratch[3]). Used by Shim programs for temporary computation. Scratch bytes form two 16-bit registers: scratch_r0 (bytes 0x0E-0x0F) and scratch_r1 (bytes 0x10-0x11). When CUSTOM flag is set, scratch_r0 and scratch_r1 carry exponent-encoded values whose semantics are deployment-defined. Shield MUST zero scratch bytes at ingress unless CUSTOM is set.

checksum: A 16-bit CRC-16/CCITT checksum computed over all 20 bytes (0x00-0x13) of the Monad with the checksum field (0x12-0x13) zeroed during computation. See Section 5.4.

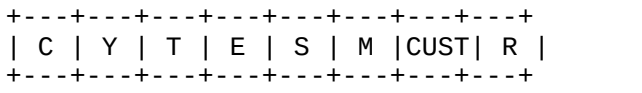
5.3.1. Extended Register Option

The 20-byte primary register file defined above MAY be complemented by an optional Extended Register Option carried as a second IPv6 HbH option within the same extension header. The extended option provides additional registers (r16-r31) using the complement space principle: the bitwise inverse of the primary register map identifies available compute capacity, formalized as a separate option with its own type code. This approach derives from the inverted subnet mask (wildcard mask) formalism established in [RFC0950].

The Extended Register Option is defined in [MONAD-EXT-REG]. Nodes that do not recognize the extended option MUST skip it per [RFC8200] option type processing rules (act=00). The primary Monad option format is unchanged; the extended option is purely additive.

5.4. Flags Bitfield

The flags field (offset 0x0B) is an 8-bit field controlling per-packet behavior:



C (0x80): CHAOS — Chaos injection active (Yaldabaoth, the chaos injection subsystem)
Y (0x40): CANARY — Canary deployment path
T (0x20): TRACED — Full trace active (all hops emit to Anamnesis)
E (0x10): ENCRYPT — Payload encrypted (intra-Kingdom TLS)
S (0x08): SAMPLED — Statistically sampled
M (0x04): MIRROR — Mirror copy (not original)
CUSTOM (0x02): Scratch and checksum fields carry exponent-encoded values
RSVD (0x01): Reserved, MUST be zero

Each bit MUST be set or cleared by Shim programs as needed. Shield MUST ensure that the C, Y, T, E, and S bits are set to consistent values at ingress based on policy.

CUSTOM (0x02): When set, the scratch fields (0x0E-0x11) and the checksum field (0x12-0x13) carry exponent-encoded values whose interpretation is defined by the active Sophia dictionary [SOPHIA]. Shield MUST NOT set CUSTOM unless configured by policy.

RSVD (0x01): Reserved. Senders MUST set to zero. Receivers MUST ignore.

5.5. Checksum Field

The checksum field (offset 0x12) holds a 16-bit CRC-16/CCITT value computed over all 20 bytes of the Monad header (offsets 0x00-0x13, inclusive), with the checksum field itself (offsets 0x12-0x13) zeroed during computation.

CRC-16/CCITT-FALSE Parameters:

Polynomial: x^16 + x^12 + x^5 + 1 (0x1021)

Initial Value: 0xFFFF

Input Reflection: false

Output Reflection: false

Final XOR: 0x0000

Computation Procedure:

The checksum is computed as follows:

1. Create a working copy of the 20-byte Monad header.

2. Set bytes 0x12-0x13 (the checksum field) to 0x0000.

3. Compute CRC-16/CCITT over all 20 bytes of this modified header.

4. Store the resulting 16-bit value at offset 0x12.

This ensures integrity protection over all Monad fields, including version, flags, flow_label, latency_hint, and reserved fields.

Shield MUST compute the checksum when creating a packet at ingress. Each hop MUST verify the checksum before processing. Each hop MUST recompute the checksum after modifying any field in offsets 0x00-0x13. The checksum field itself (offset 0x12-0x13) MUST NOT be included in the checksum computation (set to zero during computation).

If a hop detects a checksum failure, the implementation MUST:

(a) Increment a per-interface error counter.

(b) Emit an EVENT_ANOMALY event to Anamnesis if tracing is enabled.

(c) Either drop the packet (RECOMMENDED) or forward it with an anomaly flag set, depending on deployment policy.

The CRC-16 checksum provides error detection against accidental bit corruption only. It does NOT provide integrity protection against malicious modification. See Security Considerations for guidance on integrity protection mechanisms.

6. Exponent Encoding

6.1. Overview

Exponent encoding is a compositional scheme for representing rich metadata in compact form. An exponent-encoded field is a single octet interpreted as a signed 8-bit integer (two's complement, range -128 to +127).

The decoded value is computed as:

decoded = base ^ exponent

Where:

* base is the base value defined by the Sophia dictionary entry [SOPHIA] for this field position. If no Sophia entry exists, the default base is 2.

* exponent is the signed 8-bit value stored in the field.

* The result is an unsigned integer value.

An optional multiplier (unit scaling factor) may be applied:

decoded = (base ^ exponent) * multiplier

The multiplier is also defined per-field in the Sophia dictionary [SOPHIA]. If no entry exists, the multiplier is 1.

Encoders MUST NOT produce exponent values that would cause the decoded value to exceed 2^64 - 1. Decoders that encounter such values MUST treat them as errors and emit an anomaly event.

6.2. Sophia Dictionary System

Sophia dictionaries are hierarchical structures stored as BPF hash maps in kernel space and as structured tables in userspace. The full Sophia dictionary format, including sub-dictionary type systems and QPACK compression headers, is specified in [SOPHIA]. Each exponent field in the Monad has a corresponding Sophia entry that defines:

* Field name and purpose.

* Base value (typically 2, but may be 10 or other values).

* Multiplier for unit scaling.

* Semantic interpretation (lookup table from exponent to meaning).

6.2.1. Concrete Example

For src_service_id (offset 0x01):

```
Sophia Entry (service_identity):
  exponent = 0x03  (signed byte, value 3)
  base = 2
  multiplier = 1
  decoded = 2^3 * 1 = 8

Lookup: service ID 8 -> "architect" (service name)
        service ID 8 -> "fd00::8" (endpoint address)
        service ID 8 -> 0x0A03 (algorithm ID: ML-KEM-768)
```

For qos_class (offset 0x08):

```
Sophia Entry (qos_class):
  exponent = 0x02  (signed byte, value 2)
  base = 2
  multiplier = 8  (microseconds per hop)
  decoded = 2^2 * 8 = 32 microseconds
```

Interpretation: QoS class with 32-microsecond target latency.

6.2.2. Sophia Wire Format and Distribution

Sophia dictionaries MUST be distributed to all Kingdom nodes via Wotan topics [WOTAN]. Each dictionary entry MUST be serialized in CBOR format (RFC 8949) and identified by a (service_id, field_type) tuple.

A minimal Sophia dictionary that all implementations MUST support includes:

```
Root Dictionary:
  0x01 -> service_identity (sub_dict_1)
  0x02 -> flow_action (sub_dict_2)
  0x03 -> qos_class (sub_dict_3)

Sub-Dictionary: service_identity (min 16 entries)
  0x01 -> "shield" (ingress gateway)
  0x02 -> "shim" (internal hop)
  0x03 -> "architect" (application)
  ... (implementation-specific services)

Sub-Dictionary: flow_action (min 8 entries)
  0x00 -> forward (normal)
  0x01 -> trace (full event logging)
  0x02 -> sample (probabilistic logging)
  0x03 -> drop (discard packet)

Sub-Dictionary: qos_class (min 4 entries)
  0x00 -> best-effort
  0x01 -> interactive
  0x02 -> realtime
  0x03 -> bulk
```

Implementations MAY extend these dictionaries with additional entries. Dictionary version negotiation is performed through the Monad's reserved field and Anamnesis event correlation.

7. Sophia Dictionary System (Extended)

The Sophia dictionary system is the semantic layer that transforms raw exponent bytes into meaningful values. It operates at multiple levels. The full specification of the Sophia dictionary format, including sub-dictionary type systems, QPACK compression headers for dictionary entries, and hierarchical knowledge structures, is defined in [SOPHIA].

7.1. Dictionary Architecture

Sophia dictionaries are trees, not flat tables. Each byte narrows the context for the next:

```
byte_0 = 0x01 -> sophia_root -> dict_id = 1 (service_identity)
byte_1 = 0x03 -> dict_1      -> "architect"

The SAME byte_1 value in DIFFERENT contexts:
[0x01, 0x03] -> service = "architect"
[0x02, 0x03] -> action  = "sample"
[0x03, 0x03] -> qos     = "realtime"
```

With K key positions, the expressible meaning space is:

M = 256^K

```
K=1:  256 meanings           (8 bits)
K=2:  65,536 meanings        (16 bits)
K=3:  16,777,216 meanings    (24 bits)
K=8:  1.844 x 10^19 meanings (64 bits)
```

7.2. BPF Map Implementation

Sophia dictionaries are stored as BPF hash maps in kernel space per RFC 9669. Each lookup is O(1) complexity. A two-level lookup (root map to sub-dictionary) costs two hash table hits — still under 100 nanoseconds on modern hardware.

Dictionary updates propagate cluster-wide in under 10 milliseconds via atomic BPF map replacement through Wotan [WOTAN]. This allows hot-swapping of service identifiers, QoS policies, and Shim behavior without restarting any kernel components.

7.3. Minimum Required Dictionary

All implementations MUST support the following minimum Sophia dictionary:

```
Root entries (1 byte key):
  0x01 = service_identity    (sub_dict_1)
  0x02 = flow_action         (sub_dict_2)
  0x03 = qos_class           (sub_dict_3)
  0x04 = deploy_ring         (sub_dict_4)
  0x05 = circuit_state       (sub_dict_5)
  0x06 = mesh_flags          (sub_dict_6)

service_identity (sub_dict_1):
  Must include entries for all active service IDs in the Kingdom.
  Each entry maps to (service_name, endpoint_address, metadata).

flow_action (sub_dict_2):
  0x00 = FORWARD      (default: pass packet to next hop)
  0x01 = TRACE        (emit full event to Anamnesis)
  0x02 = SAMPLE       (emit with probabilistic probability)
  0x03 = DROP         (discard packet)
  0x04 = MIRROR       (clone to monitoring interface)
  (0x10-0x14 reserved for PQC key lifecycle events)

qos_class (sub_dict_3):
  0x00 = BULK         (low priority, best effort)
  0x01 = INTERACTIVE  (medium priority, <100ms latency)
  0x02 = REALTIME     (high priority, <10ms latency)

deploy_ring (sub_dict_4):
  0x00 = CANARY       (test deployment)
  0x01 = STAGING      (pre-production)
  0x02 = PRODUCTION   (user-facing)

circuit_state (sub_dict_5):
  0x00 = CLOSED       (normal operation)
  0x01 = OPEN         (circuit breaker triggered)
  0x02 = HALF_OPEN    (recovery attempt)

mesh_flags (sub_dict_6):
  0x00 = DEFAULT      (standard routing)
  0x01 = NAT_INGRESS  (behind NAT)
  0x02 = NAT_EGRESS   (acting as NAT)
  (implementation-specific flags beyond 0x02)
```

8. IANA Registration Procedures

This section provides a step-by-step guide for registering new metric types, extension headers, and protocol parameters within the Unheaded Protocol Parameters registry group.

8.1. Overview

The Unheaded Protocol defines 12 IANA registries (see Section 11). New allocations within these registries follow the registration procedures specified for each registry (Standards Action, Specification Required, Expert Review, or First Come First Served per RFC 8126).

8.2. Step-by-Step Registration Guide

To register a new metric type or protocol extension:

8.2.1. Step 1: Identify the Target Registry

Determine which of the 12 registries your allocation falls under:

Registry	Policy
Monad Protocol Version Numbers	Standards Action
Monad Flags	Specification Required
Monad Flow Actions	Expert Review
Kingdom Mode Values	Standards Action
IPv6 Hop-by-Hop Option Type	IETF Review
IPv6 Next Header Type	Standards Action
Sophia Dictionary Namespace	First Come First Served
Anamnesis Event Types	Specification Required
Error Codes	Specification Required
TLV Type Allocations	Specification Required
PQC Algorithm Identifiers	Specification Required
Wotan Topic Namespace	Expert Review

8.2.2. Step 2: Prepare the Registration Request

A registration request MUST include:

1. *Name*: A human-readable identifier for the allocation.

2. *Code Point*: The requested numeric value (or "TBD" for IANA assignment).

3. *Description*: A clear, concise description of the semantic meaning.

4. *Specification Reference*: A reference to the document that fully specifies the allocation’s behavior.

5. *Wire Format Impact*: A statement confirming whether the allocation changes the Monad wire format (MUST NOT for v0x01 allocations) or uses extension headers only.

8.2.3. Step 3: Submit the Registration

For registries requiring Expert Review or Specification Required:

1. Write an Internet-Draft describing the allocation.
2. Include the allocation in the IANA Considerations section.
3. Submit the draft to the IETF for review.
4. The designated expert(s) will evaluate per the registry policy.

For First Come First Served registries (Sophia Dictionary Namespace):

1. Submit the registration template directly to IANA.
2. IANA assigns the allocation without expert review.

8.2.4. Step 4: Verify Non-Conflict

Before submitting, verify that:

1. The requested code point is not already allocated.
2. The allocation does not conflict with existing semantics.
3. The allocation does not require changes to the frozen wire format.

8.3. Example Registration: UNHEADED_METRIC_V1

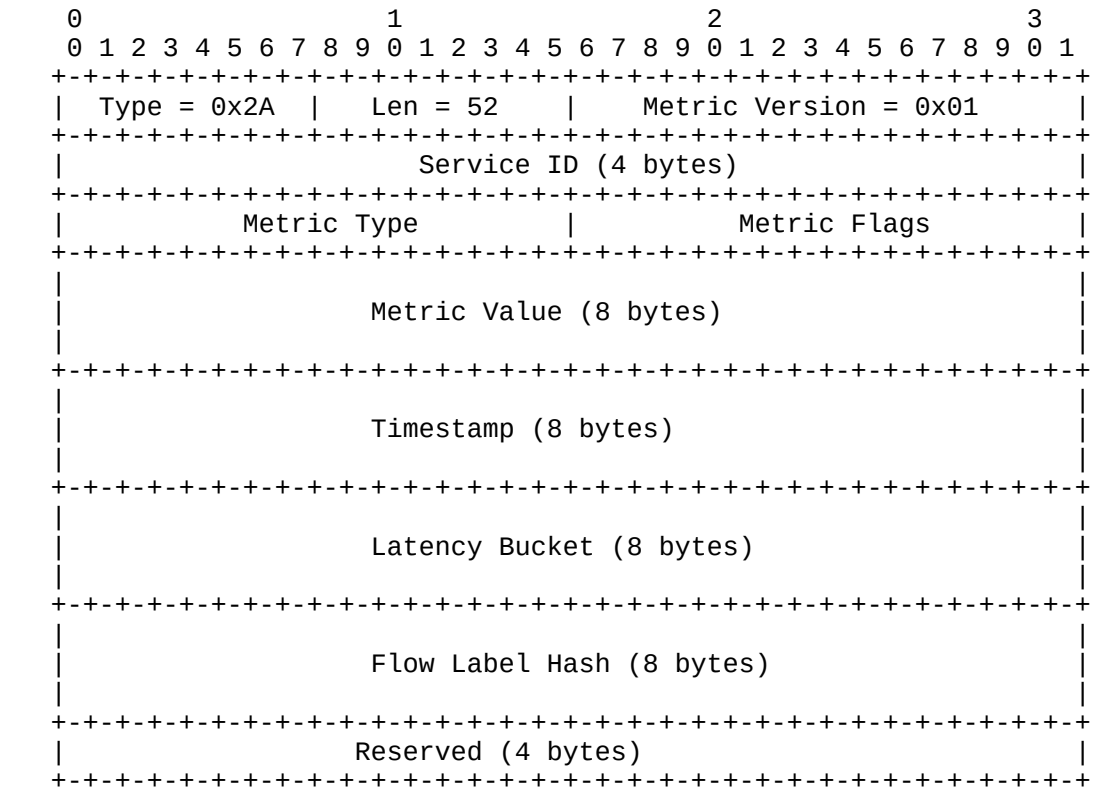
The following is an example registration for a new metric type carried in an IPv6 Hop-by-Hop extension header.

8.3.1. UNHEADED_METRIC_V1 Definition

Name:	UNHEADED_METRIC_V1
Type:	0x2A (42)
Location:	IPv6 Hop-by-Hop extension header (separate option from the Monad option)
Size:	52 bytes
Critical:	No (receivers that do not understand this option MUST skip it per RFC 8200 option type processing)

8.3.2. Wire Format

UNHEADED_METRIC_V1 is carried as a separate option TLV within the same Hop-by-Hop extension header that contains the Monad option. It does NOT modify the 20-byte Monad wire format.



Fields:

Metric Version: Unsigned 16-bit integer. Current version: 0x01.

Service ID: 32-bit identifier of the service emitting the metric. Maps to Sophia service_identity entries [SOPHIA].

Metric Type: 16-bit metric classification per the Sophia observability entry schema (0=counter, 1=gauge, 2=histogram, 3=summary).

Metric Flags: 16-bit bitfield. Bit 0: aggregatable. Bit 1: per-hop. Bit 2: cumulative. Bits 3-15: reserved (MUST be zero).

Metric Value: 64-bit unsigned integer representing the metric sample value.

Timestamp: 64-bit unsigned integer representing nanoseconds since Unix epoch.

Latency Bucket: 64-bit unsigned integer encoding the latency histogram bucket index (Prometheus-compatible bucketing).

Flow Label Hash: 64-bit hash of the IPv6 Flow Label for correlation with Anamnesis events and Wotan per-flow state [WOTAN].

Reserved: 32 bits. MUST be zero on send. MUST be ignored on receive.

8.3.3. Registration Template

Registry:	Unheaded TLV Type Allocations
Type:	0x2A
Name:	UNHEADED_METRIC_V1
Critical:	No (C=0)
Length:	52 bytes (fixed)
Reference:	[this document]

8.3.4. Deployment Notes

UNHEADED_METRIC_V1 is an OPTIONAL extension. Implementations that do not recognize Type 0x2A MUST skip the option per RFC 8200 processing rules (high-order bits 00 = skip on unrecognized). The 20-byte Monad wire format is NOT modified by this extension.

9. Security Considerations

9.1. Wire Format Immutability Threat Model

The Monad wire format is frozen at version 0x01 (20 bytes). This immutability is a security property, not merely a versioning decision.

9.1.1. Threat: Parser Divergence Attacks

If implementations parse different wire formats under the same version number, an attacker can craft packets that are interpreted differently by different hops (parser divergence). This enables:

* Policy bypass: One hop reads a DROP action where another reads FORWARD.

* State corruption: Monad field values are misaligned between hops.

* Fingerprint evasion: Packets evade IDS/IPS by exploiting parser differences.

Mitigation: All implementations of version 0x01 MUST parse exactly 20 bytes at the offsets specified in Section 5.1. Any deviation from the frozen wire format constitutes a protocol violation. The version field (offset 0x00) MUST be checked first; unknown versions MUST cause immediate silent drop.

9.1.2. Threat: Wire Format Modification via Extension

An attacker or misconfigured implementation could attempt to redefine the meaning of existing Monad fields by registering a new extension that changes their interpretation.

Mitigation: Extensions (TLV types, Kingdom Mode Extended Registers, UNHEADED_METRIC_V1) MUST be carried in separate option TLVs within the Hop-by-Hop extension header. They MUST NOT redefine the meaning of any field at offsets 0x00-0x13 of the Monad.

9.1.3. Verification Procedures

Implementations SHOULD perform the following verification checks to ensure wire format integrity:

1. *Version Check*: Offset 0x00 MUST equal 0x01. Silent drop on mismatch.

2. *Size Check*: Monad option Len field MUST be >= 20. Drop on mismatch.

3. *Checksum Check*: CRC-16/CCITT over all 20 bytes (0x00-0x13 with checksum field zeroed) MUST match bytes 0x12-0x13. Drop or flag on mismatch.

4. *Flags Check*: Bit 0 (RSVD) MUST be zero. Flag anomaly if set.

5. *HbH Count Check*: Exactly zero or one Hop-by-Hop extension headers per packet. Drop on multiple.

9.2. Extension Header Sanitization

Packets entering the Limited Domain from external sources MUST have any existing Hop-by-Hop options replaced or removed. External packets MUST NOT carry Unheaded Monad options. Packets exiting the Limited Domain MUST have their Hop-by-Hop options stripped before reaching the external network.

These requirements apply to both standard Monad fields and Kingdom Mode Extended Registers.

Shield ingress MUST validate that ingress packets do not already carry the Unheaded Monad option type. If found, the packet MUST be dropped and an anomaly event MUST be logged.

Shield egress MUST validate that the ULA prefix in any Kingdom Mode packet matches the configured Kingdom prefix. If mismatch is detected, the packet MUST be dropped.

9.3. BPF Containment

Shim programs are verified by the kernel BPF verifier per RFC 9669, which ensures:

* No out-of-bounds memory access.

* Bounded loop execution (no infinite loops).

* No unauthorized kernel function calls.

* Stack safety and register constraints.

Shim program loading REQUIRES CAP_BPF and CAP_NET_ADMIN capabilities (or equivalent). Implementations MUST use Linux kernel version 5.17 or later.

9.4. Integrity

The CRC-16 checksum detects accidental bit corruption only. It does NOT provide integrity protection against malicious modification.

Deployments requiring integrity protection against compromised intermediate nodes MUST use one of:

(a) IPsec ESP (RFC 4303) to protect the entire packet including extension headers.

(b) The optional HMAC field (not defined in this document) appended to the Monad, using a pre-shared key distributed via Sophia [SOPHIA].

(c) ML-DSA-65 (FIPS 204) signatures at Shield boundaries for post-quantum integrity.

The choice of integrity mechanism is a deployment decision outside the scope of this specification.

9.5. Trust Boundary

The trust boundary is the Limited Domain boundary. Within the domain, all Shim programs and Wotan instances are operator-controlled. Cross-domain routing is NOT supported. Inter-domain traffic MUST be encapsulated (IP-in-IP or VPN tunnel) with the Hop-by-Hop option stripped at the egress boundary and reconstructed at the ingress boundary of the next domain.

9.6. Post-Quantum Threat Model

PQC identity binding (Section 9) protects against:

* Harvest-now-decrypt-later attacks on metadata correlation.
* Service identity spoofing by a quantum-capable adversary within the Limited Domain.
* Key compromise via classical cryptanalysis of traditional key agreement algorithms.

The hybrid PQ/T mode (RFC 9370) ensures security during transition while CRQCs are not yet available.

9.7. Kingdom Mode Threat Model

Kingdom Mode Extended Registers (Section 8) are only valid within the Limited Domain. An external attacker who can inject packets with forged ULA prefixes and K flags set could:

* Forge PQC fingerprints to bypass per-hop verification.
* Inject false Extended Register values to manipulate Shim program behavior.

Shield ingress MUST validate ULA prefix provenance and MUST reject Kingdom Mode packets from outside the domain.

10. Backwards Compatibility Statement (draft-05 to draft-06)

This section formally documents that the transition from draft-05 to draft-06 is non-breaking.

10.1. Wire Format

The Monad wire format (20 bytes, version 0x01) is UNCHANGED between draft-05 and draft-06. All field offsets, sizes, types, and the total size remain identical. Implementations conforming to draft-05 will correctly parse and process packets conforming to draft-06, and vice versa.

10.2. Registry Additions

Draft-06 adds the UNHEADED_METRIC_V1 example registration (Type 0x2A, 52 bytes in a separate HbH option TLV). This is a purely additive extension. Implementations that do not recognize Type 0x2A will skip the option per RFC 8200 processing rules. No existing registry entries are modified or removed.

10.3. New Sections

The following sections are new in draft-06 and do not modify any existing protocol behavior:

1. IANA Registration Procedures (Section 9): Informational guidance for new registrations.

2. Security Considerations — Wire Format Immutability Threat Model (Section 10.1): Documents threats and mitigations for the frozen wire format.

3. Cross-References to Sophia draft-03 [SOPHIA] and Wotan draft-03 [WOTAN].

4. This backwards compatibility statement.

10.4. Interoperability

A deployment running a mix of draft-05 and draft-06 implementations will operate correctly. Draft-06 implementations produce identical wire-format packets to draft-05 implementations. The only difference is that draft-06 implementations MAY additionally produce UNHEADED_METRIC_V1 options, which draft-05 implementations will skip per standard HbH option processing.

11. IANA Considerations

All registries defined in this document are placed in a new "Unheaded Protocol Parameters" registry group, consistent with IANA practice for protocol families (cf. "QUIC Transport Parameters" for [RFC9000]).

11.1. IPv6 Hop-by-Hop Option Type

A new IPv6 Hop-by-Hop option type is requested:

Type:	TBD (suggested: 0x3E)
Name:	Unheaded Monad Option
Change Controller:	IESG
Reference:	This document

The high-order two bits MUST be 00 (skip on unrecognized) and the third bit MUST be 1 (option data may change en-route). This yields the format 001xxxxx.

11.2. Monad Protocol Version Registry

IANA is requested to create a new registry entitled "Monad Protocol Version Numbers" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Standards Action [RFC8126]

Value	Description	Reference
0x00	Reserved (MUST NOT be used)	[this document]
0x01	Unheaded Protocol v1	[this document]
0x02-0xEF	Unassigned	
0xF0-0xFE	Reserved for Experimental Use	[this document]
0xFF	Reserved (MUST NOT be used)	[this document]

The version field occupies offset 0x00 of the Monad register file (Section 5.3). Receivers MUST drop packets with unknown version values as specified in Section 5.3.

11.3. Monad Flags Bitfield Registry

IANA is requested to create a new registry entitled "Monad Flags" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Specification Required [RFC8126]

The flags field occupies offset 0x07 of the Monad register file (Section 5.5).

Bit	Name	Description	Reference
7	CHAOS	Chaos injection active (Yaldabaoth, the chaos injection subsystem)	[this document]
6	CANARY	Canary deployment path marker	[this document]
5	TRACED	Full trace active (all hops emit to Anamnesis)	[this document]
4	ENCRYPT	Payload encrypted (intra-Kingdom TLS)	[this document]
3	SAMPLED	Statistical sampling active	[this document]
2	MIRROR	Mirror copy (not original packet)	[this document]
1	CUSTOM	Scratch fields carry exponent-encoded values	[this document]
0	RSVD	Reserved. Senders MUST set to zero.	[this document]

11.4. Monad Flow Action Registry

IANA is requested to create a new registry entitled "Monad Flow Actions" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Expert Review [RFC8126]

Value	Name	Description	Reference
0x00	FORWARD	Normal forwarding (default)	[this document]
0x01	TRACE	Full event logging to Anamnesis	[this document]
0x02	SAMPLE	Probabilistic event logging	[this document]
0x03	DROP	Discard packet	[this document]
0x04	MIRROR	Clone to monitoring interface	[this document]
0x05	RATE_LIMIT	Apply rate limiting policy	[this document]
0x06	REDIRECT	Redirect to alternate dest	[this document]
0x07	INJECT	Inject synthetic packet	[this document]
0x08-0x0F	Unassigned		
0x10	KEY_ROTATE	PQC key rotation event	[this document]
0x11	KEY_REVOKE	PQC key revocation event	[this document]
0x12	KEY_DISTRIBUTE	PQC key distribution event	[this document]
0x13	KEY_VERIFY	PQC key verification event	[this document]
0x14	KEY_CHALLENGE	PQC challenge-response	[this document]
0x15-0xEF	Unassigned		
0xF0-0xFE	Experimental Use		[this document]
0xFF	Reserved	MUST NOT be used	[this document]

11.5. Kingdom Mode Registry

IANA is requested to create a new registry entitled "Kingdom Mode Values" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Standards Action [RFC8126]

Value	Name	Description	Reference
0b00	NORMAL	Standard processing	[this document]
0b01	PRIORITY	Priority processing with address-aware	[this document]
0b10	EXPERIMENTAL	Development/testing mode	[this document]
0b11	RESERVED	Reserved. MUST NOT be used.	[this document]

11.6. IPv6 Next Header Type Allocation

Next Header: 0xFE (254)
Name: Unheaded Monad Protocol
Change Controller: IESG
Reference: This document

Note: This allocation is reserved for future use and is not required by the normative protocol path, which uses IPv6 Hop-by-Hop options.

11.7. IPv6 Flow Label Per RFC 6437

Monad packets MUST use the IPv6 Flow Label field as a packet trace identifier per RFC 6437.

11.8. Sophia Dictionary Namespace Registry

Registry Name: Unheaded Sophia Dictionary Namespace
Template: Dictionary Name, Program Name, Version, Organization, Contact Email
Policy: First Come First Served

11.9. Anamnesis Event Type Registry

Registry Name: Unheaded Anamnesis Event Types
Policy: Specification Required

Initial entries:

EVENT_BORN (0x00)	Packet created at Shield (birth)
EVENT_COMPUTED (0x01)	Shim executed, Monad updated
EVENT_WOTAN_RD (0x02)	Wotan memory read
EVENT_WOTAN_WR (0x03)	Wotan memory write
EVENT_CHAOS (0x04)	Chaos mode applied
EVENT_ROLLBACK (0x05)	Monad rolled back
EVENT_DIED (0x06)	Packet reached Shield (death)
EVENT_KEY_OP (0x07)	PQC key lifecycle event
EVENT_ANOMALY (0x08)	Integrity or version error

11.10. Error Code Registry

```
  Registry Name:  Unheaded Protocol Error Codes
  Policy:         Specification Required

  Initial entries:
    NO_ERROR (0x00)                    No error (data packet)
    CRC_VALIDATION_FAILED (0x01)       CRC-16 checksum mismatch
    VERSION_NOT_SUPPORTED (0x02)       Unknown Monad version
    FLOW_LABEL_INVALID (0x03)          IPv6 flow label validation failed
    ARITHMETIC_OVERFLOW (0x04)         Exponent decoding overflow
    WOTAN_BOUNDS_CHECK_FAILED (0x05)   Wotan helper offset out of bounds
    MULTIPLE_HBH_HEADERS (0x06)        Multiple HbH options present
    UNKNOWN_CRITICAL_TLV (0x07)        Unknown critical TLV type
    WAL_SEQNO_DISCONTINUITY (0x08)     WAL sequence number gap detected
    INSUFFICIENT_BUFFER_SPACE (0x09)   No space for Monad in packet
    FLOW_STATE_CORRUPTION (0x0A)       Wotan flow state invalid
    TLS_HANDSHAKE_FAILURE (0x0B)       TLS/QUIC handshake error
    QUIC_VERSION_MISMATCH (0x0C)       QUIC version negotiation failed
    RESERVED (0x0D)                    Reserved for future use
    0x0E-0x1E                          Unallocated (future use)
    0x1F-0xFF                          Private use (testing/greasing)
```

11.11. TLV Type Registry (Extended)

```
   Registry Name:  Unheaded TLV Type Allocations
   Policy:         Specification Required

   Reserved Range: 0x00-0x1F (Monad Foundation)
   Reserved Range: 0x20-0x3F (Sophia Dictionary)
   Reserved Range: 0x40-0x5F (Wotan Memory)
   Reserved Range: 0x60-0x7F (Future Extensions)

   With critical bit:
     0x80-0x9F (Monad Foundation, critical)
     0xA0-0xBF (Sophia Dictionary, critical)
     0xC0-0xDF (Wotan Memory, critical)
     0xE0-0xFF (Future Extensions, critical)

   Initial entries:
     0x01 (Ring Path Counter, optional, 4 bytes, this document)
     0x2A (UNHEADED_METRIC_V1, optional, 52 bytes, this document)
```

11.12. PQC Algorithm Registry

Registry Name: Unheaded PQC Algorithm Identifiers
Policy: Specification Required

Initial entries:
ML-KEM-768 (0x01, 1184, FIPS 203)
ML-KEM-1024 (0x02, 1568, FIPS 203)
ML-DSA-65 (0x03, 1952, FIPS 204)
ML-DSA-87 (0x04, 2592, FIPS 204)
SLH-DSA-SHA2-128s (0x05, 32, FIPS 205)

11.13. MBC Opcode Numbers (NEW in draft-06 update)

IANA is requested to create a new registry entitled "UPC MBC Opcode Numbers" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Specification Required [RFC8126]

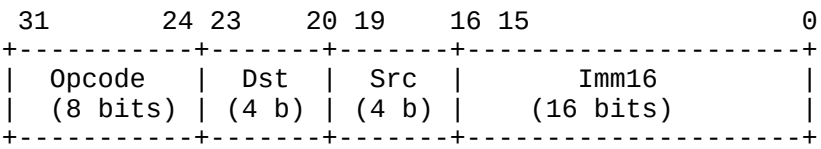
The MBC (Monad Bytecode) ISA defines the instruction set for the Unheaded Protocol Computer (UPC), a Turing-complete computational model operating within BPF programs. Each instruction is a 32-bit word with the opcode in bits [31:24].

Value	Mnemonic	Description	Reference
0x00	NOP	No operation	[this document]
0x01	ADD	dst = dst + src	[this document]
0x02	SUB	dst = dst - src	[this document]
0x03	MUL	dst = dst * src	[this document]
0x04	DIV	dst = dst / src (unsigned)	[this document]
0x05	MOD	dst = dst % src (unsigned)	[this document]
0x06	NEG	dst = -dst	[this document]
0x07	AND	dst = dst & src	[this document]
0x08	OR	dst = dst | src	[this document]
0x09	XOR	dst = dst ^ src	[this document]
0x0A	NOT	dst = ~dst	[this document]
0x0B	SHL	dst = dst << imm16	[this document]
0x0C	SHR	dst = dst >> imm16 (logical)	[this document]
0x0D	SAR	dst = dst >> imm16 (arithmetic)	[this document]
0x0E	MOV	dst = src	[this document]
0x0F	MOVI	dst = zero_extend(imm16)	[this document]
0x10	CMP	flags = cmp(dst, src)	[this document]
0x17	INT	Software interrupt (push flags+PC, jmp IVT)	[this document]
0x18	IRET	Return from interrupt (pop PC+flags)	[this document]
0x1A	PUSH	SP -= 1; ram[SP] = regs[dst]	[this document]
0x1B	POP	regs[dst] = ram[SP]; SP += 1	[this document]
0x1C	LOAD_IMM32	regs[dst][31:16] = imm16	[this document]
0x1D	ADDI	dst = dst + sign_extend(imm16)	[this document]
0x20	JMP	pc += imm16_signed (unconditional)	[this document]

0x21	JZ	if Z: pc += imm16_signed	[this document]
0x22	JNZ	if !Z: pc += imm16_signed	[this document]
0x23	JN	if N: pc += imm16_signed	[this document]
0x24	JP	if !N: pc += imm16_signed	[this document]
0x25	JC	if C: pc += imm16_signed	[this document]
0x26	JNC	if !C: pc += imm16_signed	[this document]
0x27	CALL	push(PC+1); pc = imm16	[this document]
0x28	RET	Pop PC from stack	[this document]
0x29	JMPR	pc = regs[dst] (indirect jump)	[this document]
0x2A	CALLR	push(PC+1); pc = regs[dst] (indirect call)	[this document]
0x30	LD	dst = ram[src + imm16] (32-bit load)	[this document]
0x31	ST	ram[dst + imm16] = src (32-bit store)	[this document]
0x32	LDB	dst = ram[src + imm16] (byte load)	[this document]
0x33	STB	ram[dst + imm16] = src & 0xFF (byte store)	[this document]
0x34	LDH	dst = ram[src + imm16] (16-bit load)	[this document]
0x35	STH	ram[dst + imm16] = src & 0xFFFF (16-bit store)	[this document]
0x36	SHLR	dst = dst << (src & 31) (register shift)	[this document]
0x37	SHRR	dst = dst >> (src & 31) (logical, register)	[this document]
0x38	SARR	dst = dst >> (src & 31) (arithmetic, register)	[this document]
0x39	MULH	dst = (i64(dst)*i64(src))>>32 (signed high)	[this document]
0x3A	MULHU	dst = (u64(dst)*u64(src))>>32 (unsigned high)	[this document]
0x3B	CLI	Clear interrupt flag (disable interrupts)	[this document]
0x3C	STI	Set interrupt flag (enable interrupts)	[this document]
0x3D	XCHG	Atomic exchange: dst <-> ram[src+imm]	[this document]
0x3E	CAS	Compare-and-swap: if ram==r0 then ram=r2	[this document]
0x40	SYSCALL	Invoke I/O callback; imm16 = syscall number	[this document]
0x41-0xFE		Unassigned	
0xFF	HALT	Halt execution; sets halted flag	[this document]

11.13.1. MBC Instruction Encoding

All MBC instructions use a fixed 32-bit encoding:



Dst and Src index into the 16-entry register file (r0-r15). Register r15 is the stack pointer (SP). CPU status flags: bit 0 = Zero (Z), bit 1 = Negative (N), bit 2 = Carry (C).

11.14. MBC Syscall Numbers (NEW in draft-06 update)

IANA is requested to create a new registry entitled "UPC MBC Syscall Numbers" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Expert Review [RFC8126]

MBC syscalls are invoked via the SYSCALL instruction (opcode 0x40). The syscall number is carried in imm16. Two syscall classes exist: native MBC syscalls (0x01-0x0F) and Linux-compatible syscalls (dispatched via INT 0x80 with syscall number in r0).

11.14.1. Native MBC Syscalls

Value	Name	Description	Reference
0x01	SYS_DRAW_FRAME	Copy pixel buffer to screen_map	[this document]
0x02	SYS_GET_KEY	Read keyboard state from kbd_map	[this document]
0x03	SYS_GET_TICKS	Return bpf_ktime_get_ns() / 1000000	[this document]
0x04	SYS_SLEEP	Set sleep_until = now + r0 * 1000000	[this document]
0x05-0x0F		Unassigned (native syscalls)	

11.14.2. Linux-Compatible Syscalls (INT 0x80)

Value	Name	Description	Reference
1	SYS_EXIT	Halt CPU with exit code	[this document]
2	SYS_FORK	Create child process (simplified)	[this document]
3	SYS_READ	Read from file descriptor	[this document]
4	SYS_WRITE	Write to file descriptor	[this document]
5	SYS_OPEN	Open a file	[this document]
6	SYS_CLOSE	Close a file descriptor	[this document]
7	SYS_WAITPID	Wait for child process	[this document]
11	SYS_EXECVE	Execute a program	[this document]
20	SYS_GETPID	Get process (instance) ID	[this document]
45	SYS_BRK	Set/query program break (heap end)	[this document]
54	SYS_IOCTL	Device control	[this document]
90	SYS_MMAP	Map memory	[this document]
91	SYS_MUNMAP	Unmap memory	[this document]
120	SYS_CLONE	Create child process/thread	[this document]
122	SYS_UNAME	Get system information	[this document]
146	SYS_WRITEV	Write scatter/gather	[this document]
158	SYS_SCHED_YIELD	Yield the processor	[this document]
162	SYS_NANOSLEEP	High-resolution sleep	[this document]
190	SYS_VFORK	Create child, suspend parent	[this document]
265	SYS_CLOCK_GETTIME	Get clock time	[this document]

11.15. UPC Memory Region Types (NEW in draft-06 update)

IANA is requested to create a new registry entitled "UPC Memory Region Types" in the "Unheaded Protocol Parameters" registry group.

Registration Policy: Specification Required [RFC8126]

The UPC memory model defines a 32-bit flat address space with memory-mapped I/O regions. Each region has a defined base address, size, and access semantics.

Base Address	Size	Name	Description	Reference
0x0000_0000	448 KiB	RAM	General-purpose RAM	[this document]
0x0006_8000	4 bytes	KBD_IO	Keyboard I/O word	[this document]
0x0007_0000	64 000 B	SCREEN	Screen I/O (320x200 8bpp)	[this document]
0x000F_0000	128 KiB	DEBUG	Debug region	[this document]
0x0011_0000	4 MiB	WAD	WAD data region	[this document]
0x0052_0000	16 MiB	HEAP	Heap (bump allocator)	[this document]
0x03F0_0000	(grows down)	STACK	Stack (grows downward)	[this document]
0x0200_0000*	4 MiB	RAMDISK	Block device (512B blocks)	[this document]

*RAMDISK base is in word-addressed space (0x200000 words = 32 MiB byte offset).

11.15.1. UPC Memory Hierarchy

Level	Name	Size	Latency	Type
L0	Monad Register File	20 bytes	~ns	In-packet
L1	BPF Map Cache	variable	~100-200 ns	Per-hop
L2	Wotan Ring Buffer	configurable	~1-10 us	Per-flow
L3	Write-Ahead Log	disk-backed	~100 us-1 ms	Persistent
L4	Sophia Dictionaries	BPF maps	~100-200 ns	Instruction decode

11.15.2. UPC Interrupt Vector Table

Vector	Name	Description	Reference
0x20	VECTOR_TIMER	Timer interrupt (~12 Hz)	[this document]
0x21	VECTOR_KEYBOARD	Keyboard interrupt	[this document]
0x80	VECTOR_SYSCALL	Software syscall interrupt	[this document]

Timer ticks at every 3rd XDP hop (~35 Hz XDP / 3 = ~12 Hz timer). Maximum concurrent processes: 4 (Level 4c scheduler).

11.16. UPC Event Types (NEW in draft-06 update)

IANA is requested to create a new registry entitled "UPC Event Types" in the "Unheaded Protocol Parameters" registry group.

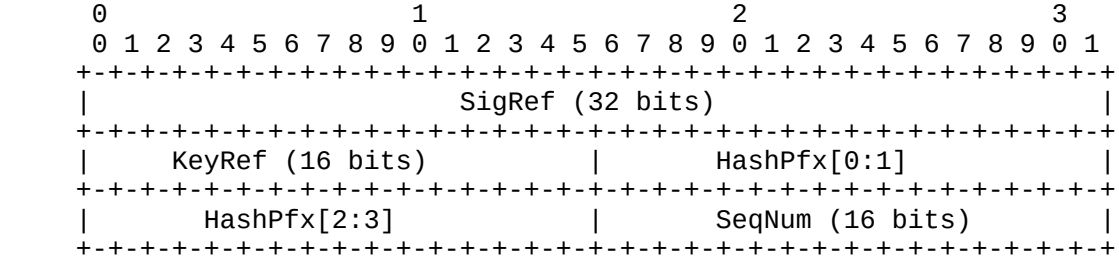
Registration Policy: Specification Required [RFC8126]

UPC events are emitted to Anamnesis ring buffers by the compute engine.

Value	Name	Description	Reference
0x10	EVENT_COMPUTE_HOP	One MBC instruction executed	[this document]
0x11	EVENT_CACHE_MISS	L1 cache miss, Wotan staging needed	[this document]
0x12	EVENT_MEM_WRITE	Dirty page written, Wotan flush	[this document]
0x13	EVENT_MEM_STAGED	Wotan staged page into L1	[this document]
0x14	EVENT_SCREEN_WRITE	SYSCALL DG_DrawFrame emitted	[this document]
0x15	EVENT_KEY_READ	SYSCALL DG_GetKey executed	[this document]
0x16	EVENT_COMPUTE_HALT	HALT syscall executed	[this document]
0x17	EVENT_COMPUTE_STALL	Stall (cache miss, sleep)	[this document]
0x18	EVENT_TTY_WRITE	TTY write (SYS_WRITE to fd 1/2)	[this document]
0x19	EVENT_CONTEXT_SWITCH	Scheduler context switch	[this document]
0x1A-0x1F		Unassigned (compute events)	

11.17. PQC Authentication Value Format

The PQC authentication value is a 12-byte structure carried in the Monad scratch bytes (offsets 0x0E-0x13) when both the SAMPLED (bit 3) and CUSTOM (bit 1) flags are set in the Monad flags field.



Fields:

SigRef: 32-bit index into the Sophia PQC_SIG_MAP BPF map. References the full post-quantum signature (which may be kilobytes in size).

KeyRef: 16-bit index into the Sophia PQC_KEY_MAP BPF map. References the corresponding public key.

HashPfx: 4-byte prefix of SHA-256(signature). Enables fast signature identification without full map lookup.

SeqNum: 16-bit per-flow sequence number for replay protection. MUST be monotonically increasing within each flow.

11.17.1. PQC Algorithm Identifiers

Value	Algorithm	Key Size	Standard	Reference
0x01	SLH-DSA	variable	FIPS 205	[this document]
0x02	ML-DSA	variable	FIPS 204	[this document]
0x03	FN-DSA	variable	FIPS 206	[this document]
0x04	ML-KEM	variable	FIPS 203	[this document]
0x05	HQC	variable	FIPS 207	[this document]

11.17.2. PQC Signature Verification Status

Value	Status	Description	Reference
0x00	Pending	Not yet verified	[this document]
0x01	Valid	Verification passed	[this document]
0x02	Invalid	Verification failed	[this document]
0x03	Expired	Key expired or revoked	[this document]
0x04	Unsupported	Algorithm not supported	[this document]

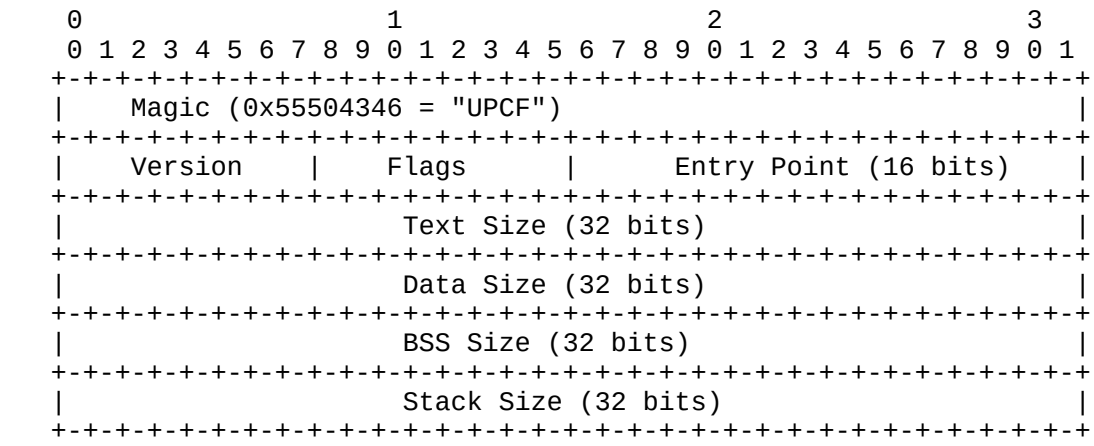
11.17.3. Compliance Tiers (Kingdom Mode K1|K0)

Value	Tier	Description	Reference
0x00	None	No PQC required	[this document]
0x01	Standard	At least one PQC signature	[this document]
0x02	Enhanced	PQC + specific algorithm reqs	[this document]
0x03	Sovereign	2-of-3 cross-algorithm multi-sig	[this document]

11.18. UPCFlat Binary Format Specification

The UPCFlat binary format is the executable format for UPC programs. A UPCFlat binary is a flat memory image loaded directly into the ROM_MAP BPF hash map at program start.

11.18.1. File Header



Magic: 0x55504346 ("UPCF" in ASCII).

Version: Binary format version (currently 0x01).

Flags: Bit 0 = requires MMU. Bit 1 = uses interrupts. Bit 2 = uses block device. Bits 3-7 reserved (MUST be zero).

Entry Point: Initial PC value (index into text section).

Text Size: Size of the text (code) section in bytes.

Data Size: Size of the initialized data section in bytes.

BSS Size: Size of the zero-initialized data section in bytes.

Stack Size: Requested stack size in bytes. Default: 0xFFFF_0000.

11.18.2. Section Layout

Offset	Content
0x00	UPCFlat header (24 bytes)
0x18	Text section (.text, MBC instructions)
0x18 + text_size	Data section (.data, initialized globals)

BSS is not stored in the binary; it is zero-filled at load time. The stack grows downward from the address specified by REG_SP_DEFAULT (0xFFFF_0000).

11.19. UNFS Filesystem Specification

The Unheaded Network Filesystem (UNFS) is a minimal filesystem for UPC programs operating on the ramdisk block device. UNFS uses fixed-size directory entries and contiguous file allocation.

11.19.1. Superblock (Block 0)

Offset	Size	Field	Description
0x00	4	magic	0x554E4653 ("UNFS")
0x04	4	total_blocks	Total blocks in filesystem
0x08	4	free_blocks	Number of free blocks
0x0C	4	root_inode	Block number of root directory
0x10	2	block_size	Block size in bytes (512)
0x12	2	version	Filesystem version (0x01)
0x14	4	flags	Mount flags (bit 0 = read-only)
0x18	8	reserved	Reserved (MUST be zero)

11.19.2. Directory Entry (32 bytes)

Offset	Size	Field	Description
0x00	16	name	Null-terminated filename (15 chars max)
0x10	4	start_block	First block of file data
0x14	4	size_bytes	File size in bytes
0x18	2	flags	Entry flags (bit 0 = directory)
0x1A	2	permissions	Unix-style permissions (rwxrwxrwx)
0x1C	4	reserved	Reserved (MUST be zero)

11.19.3. Design Constraints

* Maximum filename length: 15 characters (null-terminated in 16 bytes)
* Maximum file size: limited by contiguous block allocation
* Block size: 512 bytes (matching mbc_block::BLOCK_SIZE)
* Total blocks: 8192 (matching mbc_block::TOTAL_BLOCKS, 4 MiB ramdisk)
* No journaling (WAL provides crash recovery at the Wotan level)
* Contiguous allocation only (no fragmentation handling)

12. Author’s Address

Stevie Bellis Unheaded Email: stevie@bellis.tech

13. References

13.1. Normative References

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

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/rfc/rfc6437>.

[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>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[RFC9669] Thaler, D., Ed., "BPF Instruction Set Architecture (ISA)", RFC 9669, DOI 10.17487/RFC9669, October 2024, <https://www.rfc-editor.org/rfc/rfc9669>.

[RFC9673] Hinden, R. and G. Fairhurst, "IPv6 Hop-by-Hop Options Processing Procedures", RFC 9673, DOI 10.17487/RFC9673, October 2024, <https://www.rfc-editor.org/rfc/rfc9673>.

13.2. Informative References

[FIPS203] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard", August 2024, <https://csrc.nist.gov/pubs/fips/203/final>.

[FIPS204] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard", August 2024, <https://csrc.nist.gov/pubs/fips/204/final>.

[FIPS205] National Institute of Standards and Technology, "Stateless Hash-Based Digital Signature Standard", August 2024, <https://csrc.nist.gov/pubs/fips/205/final>.

[MONAD-EXT-REG] Bellis, S., "Monad Extended Register Option for the Unheaded Protocol", Work in Progress, Internet-Draft, draft-bellis-unheaded-monad-extended-register-00, February 2026, <draft-bellis-unheaded-monad-extended-register-00>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <https://www.rfc-editor.org/rfc/rfc792>.

[RFC0793] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

[RFC0950] Mogul, J. and J. Postel, "Internet Standard Subnetting Procedure", STD 5, RFC 950, DOI 10.17487/RFC0950, August 1985, <https://www.rfc-editor.org/rfc/rfc950>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/rfc/rfc1918>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/rfc/rfc4193>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", STD 41, RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/rfc/rfc4303>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <https://www.rfc-editor.org/rfc/rfc8300>.

[RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <https://www.rfc-editor.org/rfc/rfc8754>.

[RFC8799] Carpenter, B. and B. Liu, "Limited Domains and Internet Protocols", RFC 8799, DOI 10.17487/RFC8799, July 2020, <https://www.rfc-editor.org/rfc/rfc8799>.

[RFC9098] Gont, F., Hilliard, N., Doering, G., Kumari, W., Huston, G., and W. Liu, "Operational Implications of IPv6 Packets with Extension Headers", RFC 9098, DOI 10.17487/RFC9098, September 2021, <https://www.rfc-editor.org/rfc/rfc9098>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[RFC9197] Brockners, F., Ed., Bhandari, S., Ed., and T. Mizrahi, Ed., "Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)", RFC 9197, DOI 10.17487/RFC9197, May 2022, <https://www.rfc-editor.org/rfc/rfc9197>.

[RFC9288] Gont, F. and W. Liu, "Recommendations on the Filtering of IPv6 Packets Containing IPv6 Extension Headers at Transit Routers", RFC 9288, DOI 10.17487/RFC9288, August 2022, <https://www.rfc-editor.org/rfc/rfc9288>.

[RFC9370] Tjhai, CJ., Tomlinson, M., Bartlett, G., Fluhrer, S., Van Geest, D., Garcia-Morchon, O., and V. Smyslov, "Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 9370, DOI 10.17487/RFC9370, May 2023, <https://www.rfc-editor.org/rfc/rfc9370>.

[RFC9486] Bhandari, S., Ed. and F. Brockners, Ed., "IPv6 Options for In Situ Operations, Administration, and Maintenance (IOAM)", RFC 9486, DOI 10.17487/RFC9486, September 2023, <https://www.rfc-editor.org/rfc/rfc9486>.

[RFC9631] Bonica, R., Kamite, Y., Alston, A., Henriques, D., and L. Jalil, "The IPv6 Compact Routing Header (CRH)", RFC 9631, DOI 10.17487/RFC9631, August 2024, <https://www.rfc-editor.org/rfc/rfc9631>.

[SOPHIA] Bellis, S., "Sophia Dictionary Format for the Unheaded Protocol", Work in Progress, Internet-Draft, draft-bellis-unheaded-sophia-dictionary-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-bellis-unheaded-sophia-dictionary-00>.

[WOTAN] Bellis, S., "Wotan Memory Protocol for the Unheaded Protocol", Work in Progress, Internet-Draft, draft-bellis-unheaded-wotan-memory-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-bellis-unheaded-wotan-memory-00>.

Appendix A. Changes from draft-bellis-unheaded-protocol-foundation-05

The following changes are made in draft-06:

1. *IANA Registration Procedures (NEW)*: Added Section 9 with a step-by-step guide for registering new metric types and protocol extensions within the Unheaded Protocol Parameters registry group. Includes guidance on identifying the target registry, preparing the registration request, and verifying non-conflict with existing allocations.

2. *UNHEADED_METRIC_V1 Example Registration (NEW)*: Added a complete example registration for UNHEADED_METRIC_V1 (Type 0x2A, 52 bytes) as a separate HbH extension header option. This serves as a reference implementation for future IANA registrations. The 20-byte Monad wire format is NOT modified.

3. *Cross-References to Companion Specifications (NEW)*: Added explicit cross-references to Sophia draft-03 [SOPHIA] and Wotan draft-03 [WOTAN] throughout the document. Section 1.2 introduces the specification family structure. Sophia and Wotan references are updated from draft-02 to draft-03.

4. *Security Considerations — Wire Format Immutability (NEW)*: Added Section 10.1 documenting the threat model for wire format immutability, including parser divergence attacks, wire format modification via extension, and verification procedures.

5. *Backwards Compatibility Statement (NEW)*: Added Section 11 formally documenting that the transition from draft-05 to draft-06 is non-breaking. Wire format, registry entries, and interoperability are all preserved.

6. *Wire Format FROZEN Statement (ENHANCED)*: Added explicit "WIRE FORMAT FROZEN" statement in Section 5.1 clarifying that no changes to byte offsets, field sizes, field ordering, or total size are permitted in version 0x01.

7. *TLV Type Registry Updated*: Added UNHEADED_METRIC_V1 (0x2A) to the TLV Type Registry initial entries.

8. *UPC MBC ISA Opcode Registry (NEW)*: Added IANA registry for MBC opcode numbers (0x00-0xFF) with 48 initial entries covering arithmetic, logical, stack, branch, memory, atomic, interrupt, and system operations. Includes 32-bit instruction encoding format.

9. *UPC MBC Syscall Number Registry (NEW)*: Added IANA registry for MBC syscall numbers with two classes: native MBC syscalls (4 entries, 0x01-0x04) and Linux-compatible syscalls via INT 0x80 (20 entries, following i386 ABI numbering).

10. *UPC Memory Region Types Registry (NEW)*: Added IANA registry defining the UPC 32-bit flat address space layout with 8 memory regions (RAM, KBD_IO, SCREEN, DEBUG, WAD, HEAP, STACK, RAMDISK). Includes memory hierarchy (L0-L4) and interrupt vector table.

11. *UPC Event Types Registry (NEW)*: Added IANA registry for UPC compute engine event types (0x10-0x19) emitted to Anamnesis ring buffers, covering instruction execution, cache operations, screen I/O, TTY writes, and scheduler context switches.

12. *PQC Authentication Value Format (NEW)*: Added specification for the 12-byte PQC value carried in Monad scratch bytes (SigRef u32, KeyRef u16, HashPfx [u8;4], SeqNum u16). Includes algorithm IDs, verification status codes, and compliance tiers.

13. *UPCFlat Binary Format (NEW)*: Added specification for the UPC executable format with 24-byte header (magic 0x55504346, entry point, text/data/BSS/stack sizes) and flat memory image layout.

14. *UNFS Filesystem Specification (NEW)*: Added specification for the Unheaded Network Filesystem with superblock format, 32-byte directory entries, contiguous allocation, and 512-byte blocks on 4 MiB ramdisk.

15. *Updated Date*: Changed date from 2026-03-05 to 2026-03-15.

All changes in draft-06 are purely additive. No existing wire format, registry entry, processing rule, or normative requirement from draft-05 is modified or removed.

Appendix B. Acknowledgments

The Linux kernel BPF community (Alexei Starovoitov, Daniel Borkmann, Song Liu) for the infrastructure enabling per-packet computation in the kernel datapath.

The authors of RFC 9669 (BPF ISA), RFC 8799 (Limited Domains), and RFC 9673 (Hop-by-Hop Processing Rehabilitation) for the foundational protocols that make this design possible.

This document was co-authored with assistance from Claude (Anthropic).

Author’s Address

Stevie Bellis
Unheaded
United States of America
Email: stevie@bellis.tech