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

S. Bellis
Unheaded

19 March 2026

Wotan Memory Protocol for the Unheaded Protocol
draft-bellis-unheaded-wotan-memory-00

Abstract

Wotan is the memory and I/O bus for the Unheaded Protocol, providing addressable per-flow storage for BPF programs executing within the Limited Domain.

The Wotan protocol specifies the BPF helper interface for memory access, the address space layout for per-flow data structures, a five-level cache hierarchy (L0 through L4), and the topic-based I/O model for interaction with userspace services.

This memo defines the memory model, helper functions, address space, cache miss protocol, gRPC streaming contracts, triple-role architecture, reliability guarantees, and I/O topic naming conventions for systems implementing the Unheaded Protocol’s computational layer.

Draft-03 introduces a structured error code taxonomy with severity levels, helper return codes for common operations, and error recovery procedures. Draft-02 security patches W1-W8 are retained.

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 20 September 2026.

Copyright Notice

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

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

Table of Contents

1. Introduction

The Unheaded Protocol [UNHEADED-FOUNDATION] specifies a 20-byte register file (the Monad) that travels with every packet through a Limited Domain. BPF programs at each hop read and write the Monad, performing stateless per-packet computation.

Many use cases require state beyond the 20-byte Monad: buffering input, accumulating results, maintaining per-flow state machines, or storing scratch memory for complex algorithms.

Wotan provides this state via a hierarchical memory model:

* L0: Monad (20 bytes, in packet, per-hop latency ~320 ns)

* L1: Per-hop BPF map cache (64-byte cache lines, ~100-200 ns latency)

* L2: Per-flow ring buffer RAM (configurable size, ~1-10 us latency)

* L3: Write-Ahead Log (persistent storage, ~100 us-1 ms latency)

* L4: Sophia dictionaries [UNHEADED-SOPHIA] (instruction decode, ~100-200 ns latency)

This memo defines the Wotan memory protocol: the BPF helper interface, address space layout, cache coherency model, and userspace I/O interaction.

1.1. Cross-References

This document is part of the Unheaded Protocol specification family:

* *Protocol Foundation* [UNHEADED-FOUNDATION]: Defines the Monad wire format (20 bytes, FROZEN at v0x01), per-hop processing, IANA registries, IANA registration procedures, and the wire format immutability threat model.

* *Sophia Dictionary Format* [UNHEADED-SOPHIA]: Defines the semantic layer including sub-dictionary type systems for hierarchical knowledge and QPACK compression headers for dictionary entries.

2. Terminology and Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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.

The following terms are used:

Flow Label: The IPv6 Flow Label field (20 bits) or derived hash used to key per-flow state in Wotan. Maps packets to unique per-flow ring buffers.

Ring Buffer: A BPF ring buffer (BPF_MAP_TYPE_RINGBUF) allocated per flow, used as L2 memory (general-purpose RAM) with configurable size via --ring-size.

Cache Line: A 64-byte unit of L1 cache (per-hop BPF map), with tag, valid, dirty, and LRU tracking.

Write-Back: Transfer of dirty cache lines from L1 (per-hop map) to L2 (ring buffer) for persistence or hand-off.

Memory-Mapped I/O: Designated address ranges that publish to or read from Wotan topics (e.g., write to address 0x0000C000 publishes to compute.screen).

3. Error Code Taxonomy (NEW in draft-03)

3.1. Overview

Draft-02 defined error handling for BPF helpers using standard errno codes (-ENOENT, -EFAULT, -ENOMEM, -EACCES, -EINVAL, -EAGAIN). Draft-03 introduces a structured error code taxonomy that classifies errors by severity, origin, and recommended recovery action.

This taxonomy applies to all Wotan operations: BPF helper functions, gRPC streaming, WAL operations, and control frame exchanges.

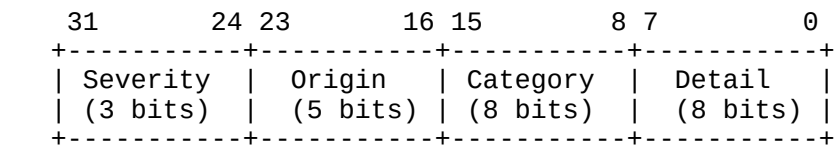
3.2. Error Severity Levels

Each error code is assigned a severity level:

Severity	Code	Description	Action
INFO	0	Informational event	Log, continue processing
WARNING	1	Degraded but functional	Log, emit metric, continue
ERROR	2	Operation failed	Log, retry or degrade
CRITICAL	3	Subsystem failure	Log, alert, isolate
FATAL	4	Unrecoverable failure	Log, halt, require restart

3.3. Structured Error Code Format

Error codes in draft-03 use a 32-bit structured format:



Severity: 3-bit severity level (0-4, see above). Bits 31-29.

Origin: 5-bit origin identifier. Bits 28-24.

Origin Code	Name	Description
0x00	WOTAN_CORE	Core Wotan daemon
0x01	WOTAN_L1	L1 cache subsystem
0x02	WOTAN_L2	L2 ring buffer subsystem
0x03	WOTAN_L3	L3 WAL subsystem
0x04	WOTAN_GRPC	gRPC streaming subsystem
0x05	WOTAN_TOPIC	Topic routing subsystem
0x06	WOTAN_SETTINGS	SETTINGS exchange
0x07	WOTAN_GOAWAY	GOAWAY frame processing
0x08	SOPHIA_LOOKUP	Sophia dictionary lookup
0x09	SOPHIA_UPDATE	Sophia dictionary update
0x0A	SHIELD_INGRESS	Shield ingress processing
0x0B	SHIELD_EGRESS	Shield egress processing
0x0C	SHIM_EXEC	Shim program execution
0x0D-0x1E	Reserved	
0x1F	VENDOR_SPECIFIC	Vendor-specific origin

Category: 8-bit error category. Bits 23-16.

Category	Name	Description
0x00	NONE	No error
0x01	ACCESS_CONTROL	Authorization / permission error
0x02	BOUNDS_CHECK	Address / offset out of range
0x03	RESOURCE	Memory / buffer / disk exhausted
0x04	INTEGRITY	Checksum / HMAC / CRC failure
0x05	PROTOCOL	Wire format / version mismatch
0x06	TIMEOUT	Operation timed out
0x07	CONCURRENCY	Lock contention / CAS failure
0x08	CONFIGURATION	Settings mismatch / invalid param
0x09	DEPENDENCY	External dependency unavailable
0x0A	DATA_CORRUPTION	Data integrity violation
0x0B	RATE_LIMIT	Rate limit exceeded
0x0C-0xFE	Reserved	
0xFF	VENDOR_SPECIFIC	Vendor-specific category

Detail: 8-bit error detail code. Bits 7-0. Interpretation depends on the category.

3.4. Helper Return Code Mapping

BPF helper functions continue to return standard errno codes for backward compatibility. The structured error code is available via an auxiliary error detail mechanism.

3.4.1. BPF Helper Return Codes

errno	Structured Code	Severity
-ENOENT (-2)	[ERROR, L2, RESOURCE, 0x01]	ERROR
-EFAULT (-14)	[ERROR, L1, BOUNDS, 0x01]	ERROR
-ENOMEM (-12)	[WARN, L1, RESOURCE, 0x01]	WARNING
-EACCES (-13)	[ERROR, CORE, ACCESS, 0x01]	ERROR
-EINVAL (-22)	[ERROR, CORE, PROTOCOL, 0x01]	ERROR
-EAGAIN (-11)	[INFO, L1, CONCURRENCY, 0x01]	INFO
-EBUSY (-16)	[WARN, L3, CONCURRENCY, 0x01]	WARNING

3.4.2. Auxiliary Error Detail

When a BPF helper returns a negative errno, implementations SHOULD write the full 32-bit structured error code to a per-CPU BPF array map:

```
struct {
    __uint(type, BPF_MAP_TYPE_PERCPU_ARRAY);
    __uint(max_entries, 1);
    __type(key, u32);
    __type(value, u32);  // 32-bit structured error code
} wotan_last_error SEC(".maps");
```

Shim programs MAY read this map after a helper returns an error to obtain detailed error information:

```
ret = bpf_wotan_read(flow_label, addr, buf, len);
if (ret < 0) {
    u32 key = 0;
    u32 *err = bpf_map_lookup_elem(&wotan_last_error, &key);
    if (err) {
        u8 severity = (*err >> 29) & 0x7;
        u8 origin = (*err >> 24) & 0x1F;
        u8 category = (*err >> 16) & 0xFF;
        u8 detail = *err & 0xFF;
        // Handle based on severity/category
    }
}
```

3.5. Common Error Codes

3.5.1. L1 Cache Errors

Code	Errno	Description
[WARN, L1, RESOURCE, 0x01]	-ENOMEM	Cache miss (line not in L1)
[ERROR, L1, BOUNDS, 0x01]	-EFAULT	Offset > 63 (read)
[ERROR, L1, BOUNDS, 0x02]	-EFAULT	Offset > 56 (write, 8-byte)
[INFO, L1, CONCURRENCY, 0x01]	-EAGAIN	CAS comparison failed
[ERROR, L1, BOUNDS, 0x03]	-EFAULT	CAS alignment error (not 8-byte)
[WARN, L1, RATE_LIMIT, 0x01]	-EBUSY	Cache-miss rate exceeded (W6)

3.5.2. L2 Ring Buffer Errors

Code	Errno	Description
[ERROR, L2, RESOURCE, 0x01]	-ENOENT	Flow not found (no ring buf)
[ERROR, L2, RESOURCE, 0x02]	-ENOMEM	Ring buffer full (overflow)
[ERROR, L2, INTEGRITY, 0x01]	N/A	CRC-32 mismatch on entry
[ERROR, L2, INTEGRITY, 0x02]	N/A	Seqno discontinuity (W1)
[ERROR, L2, DATA_CORRUPTION, 0x01]	N/A	Valid flag inconsistency

3.5.3. L3 WAL Errors

Code	Errno	Description
[ERROR, L3, RESOURCE, 0x01]	-ENOSPC	WAL disk full
[ERROR, L3, INTEGRITY, 0x01]	N/A	HMAC-SHA256 mismatch (W4)
[ERROR, L3, INTEGRITY, 0x02]	N/A	CRC-32 mismatch on WAL entry
[ERROR, L3, INTEGRITY, 0x03]	N/A	Seqno gap in WAL (W1)
[WARN, L3, CONCURRENCY, 0x01]	N/A	Compaction lock contention (W5)
[CRITICAL, L3, DATA_CORRUPTION, 0x01]	N/A	WAL file corrupted (unrecoverable)

3.5.4. gRPC Streaming Errors

Code	Errno	Description
[ERROR, GRPC, ACCESS, 0x01]	N/A	Unauthorized topic publish
[WARN, GRPC, RESOURCE, 0x01]	N/A	Subscriber buffer overflow
[ERROR, GRPC, TIMEOUT, 0x01]	N/A	Subscription idle timeout (5m)
[ERROR, GRPC, PROTOCOL, 0x01]	N/A	Invalid SETTINGS frame (W7)
[ERROR, GRPC, PROTOCOL, 0x02]	N/A	Invalid GOAWAY frame (W8)
[WARN, GRPC, RATE_LIMIT, 0x01]	N/A	GOAWAY rate limit exceeded

3.5.5. Sophia Lookup Errors

Code	Errno	Description
[ERROR, SOPHIA, RESOURCE, 0x01]	N/A	Dictionary not initialized
[ERROR, SOPHIA, BOUNDS, 0x01]	N/A	Nesting depth exceeded (8 levels)
[ERROR, SOPHIA, DATA_CORRUPTION, 0x01]	N/A	Circular reference detected
[ERROR, SOPHIA, INTEGRITY, 0x01]	N/A	ML-DSA-65 signature invalid
[ERROR, SOPHIA, RESOURCE, 0x02]	N/A	Dictionary full (128 entries)
[WARN, SOPHIA, INTEGRITY, 0x02]	N/A	QPACK decompression failure

3.6. Error Recovery Procedures (NEW in draft-03)

3.6.1. Recovery by Severity Level

Severity	Recovery Procedure
INFO	Log event. No recovery action needed. Continue processing immediately.
WARNING	Log event. Emit metric (increment warning counter). Continue processing with degraded behavior: - Cache miss: retry via BPF_TAIL_CALL (3 attempts max) - Rate limit: back off (100ms delay) - Buffer overflow: drain to L3 before retry
ERROR	Log event. Emit metric. Skip current operation. Apply fallback: - Access denied: drop packet or use default value - Bounds check: skip memory access, use zero-fill - Resource exhaustion: degrade to stateless mode - Integrity failure: reject data, emit EVENT_ANOMALY
CRITICAL	Log event. Emit alert (Wotan alerts.* topic). Isolate affected subsystem: - L3 corruption: disable WAL writes, switch to L2-only - gRPC failure: disconnect and reconnect (exponential backoff) - Sophia failure: use cached dictionary version
FATAL	Log event. Emit emergency alert. Halt affected flow: - WAL unrecoverable: drop flow state, reallocate - Total memory exhaustion: enter Emergency Mode (per [UNHEADED-FOUNDATION] Section 10.3) - Require operator intervention for restart

3.6.2. Automatic Recovery State Machine

Each Wotan subsystem (L1, L2, L3, gRPC, Sophia) maintains an independent recovery state machine:

```
States:
  HEALTHY     -> All operations succeeding
  DEGRADED    -> Some operations failing, recovery in progress
  RECOVERING  -> Active recovery procedure executing
  FAILED      -> Recovery exhausted, subsystem disabled

Transitions:
  HEALTHY -> DEGRADED:
    Trigger: error_count > threshold (configurable, default 10)
    Action: Enable recovery procedures

  DEGRADED -> RECOVERING:
    Trigger: Recovery procedure initiated
    Action: Execute recovery steps per severity level

  RECOVERING -> HEALTHY:
    Trigger: 3 consecutive successful operations
    Action: Reset error counters, resume normal operation

  RECOVERING -> FAILED:
    Trigger: Recovery attempts exhausted (max 5)
    Action: Disable subsystem, alert operator

  FAILED -> RECOVERING:
    Trigger: Operator intervention (manual restart command)
    Action: Re-execute recovery from clean state

  DEGRADED -> HEALTHY:
    Trigger: error_count drops below threshold / 2
    Action: Resume normal operation
```

3.6.3. Recovery Metrics

Implementations MUST export the following recovery metrics:

```
wotan_error_total{severity, origin, category}     (counter)
wotan_recovery_attempts_total{subsystem}           (counter)
wotan_recovery_success_total{subsystem}            (counter)
wotan_recovery_failure_total{subsystem}            (counter)
wotan_subsystem_state{subsystem}                   (gauge: 0-3)
wotan_time_in_degraded_seconds{subsystem}          (histogram)
```

3.6.4. Cross-Subsystem Recovery

When one subsystem enters FAILED state, dependent subsystems MUST be notified:

```
   Dependency Graph:
     L1 cache -> L2 ring buffer -> L3 WAL
     gRPC streaming -> L2 ring buffer
     Sophia lookup -> L1 cache

   Example: L3 WAL enters FAILED state:
     1. L2 ring buffer: Switch to overflow-drop mode (no L3 drain)
     2. L1 cache: Continue operating (L1 hits unaffected)
     3. gRPC: Continue streaming (events may lack persistence guarantee)
     4. Alert: Emit CRITICAL alert on Wotan alerts.* topic
     5. Dashboard: Display L3 subsystem as red (FAILED)
```

4. Architecture Overview

4.1. Role in the Unheaded Protocol

Wotan bridges Monad computation [UNHEADED-FOUNDATION] to memory and I/O:

* Shim programs (BPF) running at each hop read/write Wotan memory via BPF helpers (bpf_wotan_read, bpf_wotan_write, bpf_wotan_cas).

* Wotan maintains per-flow state keyed by IPv6 Flow Label.

* Wotan interfaces with userspace via ring buffer events and pub/sub topics.

* Wotan implements cache miss handling, prefetching, and Write-Ahead Log management.

4.2. Memory Hierarchy

Level	Name	Size	Latency	Backing
L0	Monad (packet)	20 bytes	~ns	wire
L1	Cache (BPF map)	variable	~100-200ns	per-hop
L2	Ring Buffer (RAM)	configurable	~1-10us	per-flow
L3	Write-Ahead Log	disk	~100us-1ms	persistent
L4	Sophia dictionaries	BPF maps	~100-200ns	instruction decode

Wotan implements transparent L1->L2 promotion on cache miss, L2->L3 flush on overflow, and L3->L2 recovery on process restart.

4.3. Separation of Compute and Memory

The Monad is transient compute state (stateless by design). Wotan is persistent state machine storage. This separation allows:

* Shim programs to remain stateless with respect to the packet format.

* External state to be accessed in a controlled, measurable manner.

* Cache miss latency to be handled without blocking per-hop logic.

* Memory updates to be tracked in Anamnesis for observability.

5. BPF Helper Interface

BPF Shim programs access Wotan memory via three helper functions. All helpers operate on a 32-bit address space keyed by IPv6 Flow Label. Error codes follow the taxonomy defined in Section 3.

5.1. bpf_wotan_read

Read from Wotan memory.

long bpf_wotan_read(u32 flow_label, u32 addr, void *buf, u32 len);

Arguments: - flow_label: 20-bit IPv6 Flow Label (zero-extended to u32) - addr: 32-bit address within the flow’s address space - buf: pointer to destination buffer - len: number of bytes to read (MUST be 1, 2, or 4)

Returns: - On success: number of bytes read (len) - -ENOENT (-2): flow_label not found [ERROR, L2, RESOURCE, 0x01] - -EFAULT (-14): addr out of bounds [ERROR, L1, BOUNDS, 0x01] - -ENOMEM (-12): L1 cache miss [WARN, L1, RESOURCE, 0x01] - -EACCES (-13): not authorized [ERROR, CORE, ACCESS, 0x01] - -EINVAL (-22): len not in {1, 2, 4} [ERROR, CORE, PROTOCOL, 0x01]

5.2. bpf_wotan_write

Write to Wotan memory.

long bpf_wotan_write(u32 flow_label, u32 addr, const void *buf, u32 len);

Arguments: - flow_label: 20-bit IPv6 Flow Label - addr: 32-bit address within the flow’s address space - buf: pointer to source buffer - len: number of bytes to write (MUST be 1, 2, or 4)

Returns: - On success: number of bytes written (len) - -ENOENT (-2): flow_label not found [ERROR, L2, RESOURCE, 0x01] - -EFAULT (-14): addr out of bounds [ERROR, L1, BOUNDS, 0x02] - -ENOMEM (-12): L1 cache miss [WARN, L1, RESOURCE, 0x01] - -EACCES (-13): not authorized [ERROR, CORE, ACCESS, 0x01] - -EINVAL (-22): len not in {1, 2, 4} [ERROR, CORE, PROTOCOL, 0x01]

5.3. bpf_wotan_cas

Atomic compare-and-swap on Wotan memory.

```
long bpf_wotan_cas(u32 flow_label, u32 addr, u32 expected, u32 desired);
```

Arguments: - flow_label: 20-bit IPv6 Flow Label - addr: 32-bit address (MUST be 8-byte aligned per PATCH W3) - expected: expected current value (u32) - desired: value to write if current == expected

Returns: - 0: swap successful [INFO, L1, NONE, 0x00] - -EAGAIN (-11): current != expected [INFO, L1, CONCURRENCY, 0x01] - -ENOENT (-2): flow_label not found [ERROR, L2, RESOURCE, 0x01] - -EFAULT (-14): addr out of bounds or not 8-byte aligned [ERROR, L1, BOUNDS, 0x03] - -EACCES (-13): not authorized [ERROR, CORE, ACCESS, 0x01]

5.4. Error Handling (Enhanced in draft-03)

Implementations MUST handle all specified error codes. The structured error taxonomy (Section 3) provides additional detail beyond errno:

For each helper return:

1. Check errno (standard error handling, backward compatible)
2. Optionally read wotan_last_error map for structured code
3. Apply recovery procedure per severity level (Section 3.5.1)
4. Emit metric: wotan_helper_error_total{errno, helper_name}

RECOMMENDED error handling by severity:

* INFO (-EAGAIN): Retry immediately (CAS retry loop, max 10 iterations)

* WARNING (-ENOMEM): Retry via BPF_TAIL_CALL (max 3 attempts)

* ERROR (-ENOENT, -EFAULT, -EACCES, -EINVAL): Skip operation, use default value, emit EVENT_ANOMALY

Programs MUST NOT crash on negative returns; they MUST check return values and branch accordingly.

6. UPC Memory Model Extensions (NEW in draft-03 update)

6.1. Overview

The Unheaded Protocol Computer (UPC) extends the Wotan memory model with dedicated BPF map regions for program code, screen I/O, keyboard input, and block device emulation. These extensions enable general-purpose computation within the BPF datapath.

6.2. ROM_MAP (Program Code Storage)

The ROM_MAP is a BPF hash map that stores MBC program instructions. The program counter (PC) indexes into this map.

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 262144);   // 256K instruction slots
    __type(key, u32);              // Word-aligned instruction address
    __type(value, u32);            // 32-bit MBC instruction word
} rom_map SEC(".maps");
```

ROM_MAP is loaded from a UPCFlat binary (see [UNHEADED-FOUNDATION] UPCFlat Binary Format) before the first packet is processed. ROM_MAP MUST NOT be modified during program execution.

6.3. RAM_MAP (Read-Write Memory)

The RAM_MAP is a BPF hash map that provides the UPC's general-purpose read-write memory. All LD/ST family instructions access this map.

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 1048576);  // 1M word slots (4 MiB)
    __type(key, u32);              // Word-aligned byte address
    __type(value, u32);            // 32-bit word value
} ram_map SEC(".maps");
```

6.3.1. Address Space Layout

The RAM_MAP address space follows the layout defined in [UNHEADED-FOUNDATION] UPC Memory Region Types:

```
Address Range        Size        Region        Description
-----------          --------    ----------    -------------------------
0x0000_0000-0x0006_FFFF  448 KiB  RAM         .rodata + .data + .bss
0x0006_8000          4 bytes     KBD_IO        Keyboard I/O (scancode<<1|pressed)
0x0007_0000-0x0007_FA00  64 000 B SCREEN      Screen pixels (320x200 8bpp)
0x000F_0000-0x0010_FFFF  128 KiB  DEBUG       Debug output region
0x0011_0000-0x0050_FFFF  ~4 MiB   WAD         WAD data (doom1.wad)
0x0052_0000-0x0152_0000  16 MiB   HEAP        Bump allocator heap
0x03F0_0000              (grows down) STACK    Stack (r15 = SP)
```

6.4. SCREEN_MAP (Video Output)

The SCREEN_MAP stores the pixel framebuffer and is separate from RAM_MAP for performance isolation.

```
struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __uint(max_entries, 16000);    // 320x200/4 = 16000 words
    __type(key, u32);              // Word index
    __type(value, u32);            // 4 packed pixels (8-bit palette)
} screen_map SEC(".maps");
```

Writes to RAM_MAP addresses in the SCREEN range (0x0007_0000 to 0x0007_FA00) are intercepted and redirected to SCREEN_MAP. A SYS_DRAW_FRAME syscall (0x01) copies the current pixel buffer from RAM_MAP to SCREEN_MAP and emits an EVENT_SCREEN_WRITE (0x14) to the Anamnesis ring buffer.

6.5. KBD_MAP (Keyboard Input)

```
struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __uint(max_entries, 1);
    __type(key, u32);              // Always 0
    __type(value, struct kbd_state);
} kbd_map SEC(".maps");

struct kbd_state {
    u32 key;       // Keycode
    u32 pressed;   // 1 = pressed, 0 = released
    u64 sequence;  // Monotonically increasing event counter
};
```

The SYS_GET_KEY syscall (0x02) reads from KBD_MAP. The BPF program checks the sequence counter to determine if a new key event has occurred since the last read. Wotan userspace writes key events to KBD_MAP via bpf_map_update_elem.

6.6. CPU_MAP (Processor State)

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 256);      // Max 256 concurrent flows
    __type(key, u32);              // IPv6 Flow Label
    __type(value, struct cpu_state);
} cpu_map SEC(".maps");

struct cpu_state {
    u32 regs[16];           // General-purpose registers r0-r15
    u32 pc;                 // Program counter (ROM_MAP index)
    u8  flags;              // CPU flags: Z(0), N(1), C(2)
    u8  stalled;            // 1 if waiting for cache miss
    u8  halted;             // 1 if HALT executed
    u8  _pad;
    u64 sleep_until;        // bpf_ktime_get_ns() wakeup time
    u64 insn_count;         // Total instructions executed
    u64 cache_hits;         // L1 cache hits
    u64 cache_misses;       // L1 cache misses
    u8  interrupt_pending;  // Non-zero if interrupt waiting
    u8  interrupt_vector;   // Pending interrupt vector
    u8  interrupts_enabled; // Non-zero if accepting interrupts
    u8  _pad2;
    u32 tick_counter;       // Timer interrupt tick counter
};
```

CPU state persists across packet hops. Each IPv6 flow label maps to an independent CPU instance. The MBC CPU processes one instruction per hop (per-packet computation model).

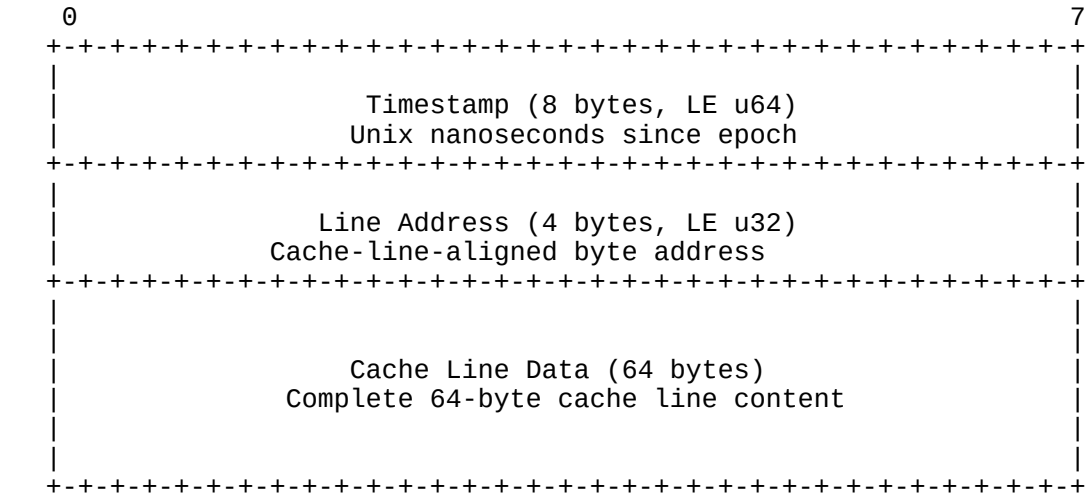
7. WAL Specification (NEW in draft-03 update)

7.1. Overview

The Write-Ahead Log (WAL) provides L3 persistence for Wotan memory. It is an append-only log of dirty cache line writes, enabling crash recovery by replaying WAL records to reconstruct L2 state.

7.2. WAL Record Format

Each WAL record is exactly 76 bytes:



Total: 8 + 4 + 64 = 76 bytes per record.

7.3. WAL Operation

7.3.1. Append

When a dirty cache line is evicted from L1 or during periodic writeback (default: every 1 ms), the WAL appends a record:

1. Open WAL file in append mode (O_APPEND | O_WRONLY)
2. Write 76-byte record (timestamp + address + data)
3. Call fsync() to ensure durability
4. Update WAL sequence counter

Implementations MUST call fsync() after each write to guarantee that the record is durable before acknowledging the writeback.

7.3.2. Recovery

On process restart, WAL records are replayed in order:

1. Open WAL file in read mode
2. Read 76-byte records sequentially
3. For each record:
   a. Validate record integrity (HMAC-SHA256 per PATCH W4)
   b. Write cache line data to L2 ring buffer at line_addr
   c. Mark page as clean
4. Resume normal operation

7.3.3. Compaction

WAL compaction reduces the file size by removing superseded records (records where a later record writes to the same address):

1. Acquire exclusive compaction lock (PATCH W5)
2. Read all records, building address -> latest_record map
3. Write retained records to new WAL file
4. Atomically rename new file over old
5. Release compaction lock

Compaction MUST NOT run concurrently with append operations.

7.4. WAL Configuration

Parameter	Default	Description
wal_enabled	false	Enable L3 persistence
wal_path	(none)	WAL file path on disk
writeback_period	1 ms	Dirty cache flush interval
wal_max_size	256 MiB	Maximum WAL file size before compaction
wal_fsync	true	fsync after each write

8. TTY Subsystem (NEW in draft-03 update)

8.1. Overview

The TTY subsystem provides console I/O for UPC programs that use standard Unix-style write(2) and read(2) syscalls on file descriptors 0 (stdin), 1 (stdout), and 2 (stderr). TTY output is captured by the BPF compute engine and forwarded to Wotan as EVENT_TTY_WRITE (0x18) events.

8.2. Circular Buffer

Each UPC flow maintains a TTY circular buffer in L2 memory for console output accumulation:

```
struct tty_buffer {
    u8   data[4096];     // 4 KiB circular buffer
    u16  head;           // Write position (next byte to write)
    u16  tail;           // Read position (next byte to read)
    u32  total_written;  // Total bytes written (monotonic counter)
    u32  overflow_count; // Number of bytes dropped due to full buffer
};
```

8.2.1. Write Operation (SYS_WRITE to fd 1 or 2)

When a UPC program issues SYS_WRITE with fd=1 or fd=2:

1. Copy bytes from RAM_MAP[buf..buf+count] to tty_buffer.data
   at the current head position
2. Advance head = (head + count) % 4096
3. If head would overtake tail (buffer full):
   a. Drop oldest bytes by advancing tail
   b. Increment overflow_count
4. Increment total_written by count
5. Emit EVENT_TTY_WRITE (0x18) to Anamnesis ring buffer
6. Publish tty output to Wotan topic compute.tty.{flow_label}

8.2.2. Read Operation (SYS_READ from fd 0)

When a UPC program issues SYS_READ with fd=0:

1. Check if tail != head (data available)
2. If data available: copy min(count, available) bytes to
   RAM_MAP[buf..buf+n]
3. Advance tail = (tail + n) % 4096
4. Return number of bytes read in r0
5. If no data: set r0 = 0 (non-blocking) or stall CPU

8.3. Event Emission

TTY events are published on the Wotan topic:

```
Topic: compute.tty.{flow_label}
Payload:
  - timestamp (u64, Unix nanoseconds)
  - flow_label (u32)
  - fd (u8, 0=stdin, 1=stdout, 2=stderr)
  - length (u16, bytes written)
  - data (variable, TTY output bytes)
```

Dashboard subscribers MAY render TTY output in a terminal emulator widget for real-time program output monitoring.

8.4. TTY Configuration

Parameter	Default	Description
tty_buffer_size	4096	Circular buffer size (bytes)
tty_emit_events	true	Emit EVENT_TTY_WRITE to Anamnesis
tty_publish_topic	true	Publish to Wotan compute.tty.*
tty_max_line_len	256	Max bytes per single write event

9. Security Considerations

9.1. Topic Injection Attacks

A malicious application could publish events to unauthorized topics.

Mitigation (MANDATORY): - Topic access control via Sophia [UNHEADED-SOPHIA] (per-program topic whitelist) - Verify publisher identity (program ID from BPF context) - Enforce least-privilege (default: deny all topics) - Log unauthorized publish attempts - Emit ANOMALY event to Anamnesis

9.2. Ring Buffer Memory Exhaustion (PATCH W6)

Per-program cache-miss rate limiting (10K misses/sec budget). See draft-02 Section 8.2 for complete specification.

9.3. Cross-Flow Memory Access (PATCH W2)

64-bit composite L1 cache keys prevent birthday attack collisions. See draft-02 Section 4.1 for complete specification.

9.4. CAS Alignment Violations (PATCH W3)

8-byte alignment enforced by BPF verifier at load time. See draft-02 Section 3.2.1 for complete specification.

9.5. WAL Tampering Detection (PATCH W4)

HMAC-SHA256 authentication on WAL entries. See draft-02 Section 3.1 for specification.

9.6. WAL Compaction Race Conditions (PATCH W5)

Exclusive mutex during compaction. See draft-02 Section 3.3.

9.7. GOAWAY Frame DoS (PATCH W8)

Frame validation and rate limiting. See draft-02 Section 10.2.

9.8. Normative Error Code Cross-Reference (13 codes)

The following 13 normative error codes are defined across the Unheaded Protocol specification family. This cross-reference table provides a single point of lookup:

Code	Name	Spec	Section	Level
0x0000	UNHD_NO_ERROR	Foundation	18.11	Flow
0x0001	UNHD_PROTOCOL_ERROR	Foundation	18.11	System
0x0002	UNHD_INVALID_FRAME	Foundation	18.11	Domain
0x0003	UNHD_FLOW_CONTROL_ERROR	Foundation	18.11	Domain
0x0004	UNHD_SETTINGS_TIMEOUT	Foundation	18.11	System
0x0005	UNHD_STREAM_CLOSED	Foundation	18.11	Flow
0x0006	UNHD_FRAME_SIZE_ERROR	Foundation	18.11	Domain
0x0007	UNHD_REFUSED_STREAM	Foundation	18.11	Flow
0x0008	UNHD_CANCEL	Foundation	18.11	Flow
0x0009	UNHD_COMPRESSION_ERROR	Foundation	18.11	Domain
0x000A	UNHD_CONNECT_ERROR	Foundation	18.11	System
0x000B	UNHD_ENHANCE_YOUR_CALM	Foundation	18.11	System
0x000C	UNHD_INTERNAL_ERROR	Foundation	18.11	System

Error codes 0x0000-0x003F are in the Standards Range and can only be registered during initialization (Specification Required policy per RFC 8126). Extension codes 0x0040-0x00FF are available for protocol extensions. Codes 0x1F00+ are reserved for testing (greasing per the 0x1F*N+0x21 pattern).

9.8.1. Error Level Definitions

Level	Description	Scope
Flow	Error specific to a single flow	Affects one stream only
Domain	Error affecting a domain	Affects one connection
System	Error affecting the system	Affects all connections

9.9. Error Code Information Leakage

Structured error codes (Section 3) may reveal internal architecture details to an attacker observing BPF program behavior or Wotan metrics.

Mitigation: - Error detail codes MUST NOT contain sensitive data (keys, addresses) - Metrics SHOULD aggregate error counts, not individual error details - The wotan_last_error map is per-CPU and ephemeral (not persisted) - External-facing error messages SHOULD use generic descriptions

9.10. Cross-Reference with Foundation and Sophia

Security considerations in this memo are aligned with:

1. *[UNHEADED-FOUNDATION] Section 10 - Security Considerations*: Wire format immutability threat model, parser divergence attacks, BPF containment, and integrity protection mechanisms.

2. *[UNHEADED-SOPHIA] Section 9 - Security Considerations*: Dictionary poisoning, nested dictionary security, QPACK decompression security, and BPF map access control.

10. IANA Considerations

This memo does not request IANA registration of option types or protocol numbers; those are handled by [UNHEADED-FOUNDATION].

Wotan topic naming uses informal convention (compute._, sophia._, anamnesis.*). If standardization is needed, IANA may create a registry:

Registry Name: Wotan Topic Namespace
Policy: Expert Review
Template: Topic Name, Component, Description, Reference

10.1. Wotan Error Origin Registry (NEW in draft-03)

Registry Name: Unheaded Wotan Error Origin Codes
Template: Origin Code (0x00-0x1F), Origin Name, Description, Specification Reference
Policy: Specification Required

Initial entries: See Section 3.2 (16 entries, codes 0x00-0x0C)

10.2. Wotan Error Category Registry (NEW in draft-03)

Registry Name: Unheaded Wotan Error Category Codes
Template: Category Code (0x00-0xFF), Category Name, Description, Specification Reference
Policy: Specification Required

Initial entries: See Section 3.2 (12 entries, codes 0x00-0x0B)

11. Author’s Address

Stevie Bellis Unheaded Email: stevie@bellis.tech

12. References

12.1. 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>.

[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>.

[UNHEADED-FOUNDATION] Bellis, S., "Unheaded: Protocol Foundation", Work in Progress, Internet-Draft, draft-bellis-unheaded-protocol-foundation-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-bellis-unheaded-protocol-foundation-00>.

[UNHEADED-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>.

12.2. Informative References

Appendix A. Changes from draft-bellis-unheaded-wotan-memory-02

The following changes are made in draft-03:

1. *Error Code Taxonomy (NEW)*: Added Section 3 defining a structured 32-bit error code format with severity levels (INFO, WARNING, ERROR, CRITICAL, FATAL), origin codes (16 subsystems), and category codes (12 categories). Provides fine-grained error classification beyond standard errno codes.

2. *Helper Return Codes (ENHANCED)*: Added structured error code annotations to all BPF helper return values (Section 5). Each errno return now maps to a full 32-bit structured code. Added auxiliary error detail mechanism via per-CPU BPF array map (wotan_last_error).

3. *Error Recovery Procedures (NEW)*: Added Section 3.5 defining recovery procedures by severity level, automatic recovery state machine (HEALTHY -> DEGRADED -> RECOVERING -> FAILED), recovery metrics, and cross-subsystem recovery dependency graph.

4. *Cross-References to Foundation draft-06 and Sophia draft-03 (UPDATED)*: Updated UNHEADED-FOUNDATION reference from draft-04 to draft-06. Updated UNHEADED-SOPHIA reference from draft-01 to draft-03. Added Section 1.1 documenting the specification family structure.

5. *Wotan Error Origin Registry (NEW IANA)*: Added IANA registry for error origin codes (0x00-0x1F).

6. *Wotan Error Category Registry (NEW IANA)*: Added IANA registry for error category codes (0x00-0xFF).

7. *Error Code Information Leakage (NEW Security)*: Added security consideration for structured error codes potentially revealing internal architecture details.

8. *UPC Memory Model Extensions (NEW)*: Added ROM_MAP, RAM_MAP, SCREEN_MAP, KBD_MAP, and CPU_MAP BPF map definitions with complete structure specifications. Defines the UPC address space layout, screen I/O redirection, keyboard input protocol, and per-flow CPU state persistence model.

9. *WAL Specification (NEW)*: Added complete Write-Ahead Log specification with 76-byte record format (8-byte timestamp + 4-byte address + 64-byte cache line data), append/fsync semantics, crash recovery replay procedure, and compaction protocol with exclusive locking.

10. *TTY Subsystem (NEW)*: Added console I/O subsystem with 4 KiB circular buffer, write/read operations on fd 0/1/2, overflow handling, EVENT_TTY_WRITE emission, and Wotan topic publication on compute.tty.{flow_label}.

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

All draft-02 content is retained, including security patches W1-W8. No existing wire format, processing rule, or normative requirement from draft-02 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