KRILL-BIA Engineering Specification

Implementation Reference for Stigmergic Consensus + Pentastratic Immunity (MVP Scope)

Version 0.2 — February 2026 Companion to: KRILL Bio-Inspired Architecture (Research Paper v1.0)

ES-0. Purpose and Scope

This document bridges the gap between the KRILL-BIA research paper and a working implementation. It specifies:

- Byte-level wire formats for all messages
- State machines with explicit transitions, timeouts, and edge cases
- Concrete algorithms (not descriptions) for each mechanism
- DAG structure and synchronization protocol
- Blockchain integration API
- Error handling for every identified edge case
- Test vectors with expected outputs
- Main event loop and message dispatch (ES-10)
- Distance estimation methods (ES-11)
- Transport layer: BLE mesh, WiFi, LoRa (ES-12)
- Cryptographic initialization and device enrollment (ES-13)
- Flash layout and persistence strategy (ES-14)
- Rate limiting and DoS protection (ES-15)
- LPWAN gateway trust model (ES-16)

MVP Scope: Stigmergic Consensus (§3) + Pentastratic Immunity (§4) + Metabolic State (§5) + Quorum Sensing (§7). Morphogenetic Topology, HGT, and Entropic Valuation are specified at interface level only.

Target Platform: ESP32 (Nano node) + nRF52840 (Dust node) over BLE mesh + WiFi.

What is this document? (Plain-Language Summary)

This is the builder's manual for KRILL. While the Research Paper explains *why* and *what*, this Engineering Specification explains *how* — down to individual bytes on the wire.

If you're a developer, here's what you'll find:

- **Exact data formats** — every message is specified byte-by-byte with offsets, sizes, and encoding
- **State machines** — every component has explicit states, transitions, and timeouts
- **Pseudocode** — copy-paste-ready algorithms in Rust-like syntax
- **Test vectors** — concrete inputs with expected outputs so you can verify your implementation
- **Hardware targets** — ESP32 (Nano node, $4) and nRF52840 (Dust node, $2) over BLE mesh + WiFi + LoRa

MVP = 4 of 9 subsystems: Stigmergic Consensus + Immune System + Metabolic State + Quorum Sensing. This is enough for a working prototype. The other 5 subsystems have interface stubs.

Key sections for getting started:

- **ES-1** — data types and constants (start here)
- **ES-10** — main event loop (how it all fits together)
- **ES-12** — transport layer (BLE/WiFi/LoRa configuration)
- **ES-13** — device enrollment (how a new device joins)
- **ES-9** — test vectors (verify your code)

ES-1. Core Data Types and Constants

ES-1.1 Primitive Types

```
Type         Size    Encoding        Notes
─────────────────────────────────────────────────────
u8           1B      unsigned
u16          2B      little-endian
u32          4B      little-endian
u64          8B      little-endian
i16          2B      little-endian   signed, for readings
i32          4B      little-endian   signed
f16          2B      IEEE 754 half   for sensor values
f32          4B      IEEE 754 single
timestamp    4B      u32             seconds since epoch (Unix, wraps 2106)
duration     4B      u32             seconds
```

```
hash20       20B     SHA-1 truncated for DAG references (NOT security-
critical)
hash32       32B     BLAKE2s-256     for integrity/identity
sig64        64B     Ed25519         compact signature
pubkey32     32B     Ed25519         public key
device_id    8B      hash32[0..8]   truncated identity hash
reading_t    2B      i16             fixed-point: value × 100 (e.g., 2310 =
23.10°C)
weight_t     1B      u8              0-255 maps to 0.0-1.0 (w = raw/255)
confidence_t 1B      u8              0-255 maps to 0.0-1.0
```

ES-1.2 Global Constants

```
PROTOCOL_VERSION        = 0x01
MAGIC_BYTES             = 0x4B 0x42 ("KB" = KRILL-BIA)

// Pheromone decay
LAMBDA_DEFAULT          = 0.00116   // decay rate, T½ = 10 min → λ =
ln(2)/600
LAMBDA_LPWAN            = 0.0000013 // T½ ~6.2 days for LoRa (λ_mesh / 900)

// Immune thresholds (σ multiples, stored as u8 × 0.1)
SIGMA_PYREXIA           = 10        // 1.0σ
SIGMA_INFLAMMATION      = 20        // 2.0σ
SIGMA_REJECTION         = 30        // 3.0σ
DRIFT_THRESHOLD         = 50        // 5.0σ (NK audit)

// Quorum sensing
THETA_LOW               = 3         // neighbors for QUORUM mode
THETA_HIGH              = 10        // neighbors for SWARM mode
HYSTERESIS_DELTA        = 2

// Timing
NEONATAL_DURATION       = 1209600   // 14 days in seconds
EPOCH_DURATION          = 60        // 1 epoch = 60 seconds
PRESENCE_INTERVAL       = 10        // seconds between presence pulses
TRACE_INTERVAL_DEFAULT  = 5         // seconds between trace deposits
NK_AUDIT_PROBABILITY    = 3         // p = 3/255 ≈ 0.012 per epoch

// Metabolic half-lives (seconds)
T_HALF_EPHEMERAL        = 600       // 10 minutes
T_HALF_OPERATIONAL      = 86400     // 1 day
T_HALF_STRUCTURAL       = 7776000  // 90 days
T_HALF_ARCHIVAL         = 0xFFFFFFFF // infinite (never decays, compresses)

// Limits
MAX_NEIGHBORS           = 32
MAX_DAG_ENTRIES         = 4096      // per local DAG (Nano); 256 (Dust)
MAX_PROFILE_BINS        = 50
PROFILE_SIZE_BYTES      = 500       // per monitored device
REJECTION_RATE_LIMIT    = 51        // max 20% = 51/255 of cluster per
epoch
```

ES-1.3 Quantity Type Enumeration

```
enum QuantityType : u8 {
    TEMPERATURE         = 0x01,
    HUMIDITY            = 0x02,
    PRESSURE            = 0x03,
    LIGHT               = 0x04,
    ACCELERATION        = 0x05,
    CO2                 = 0x06,
    SOIL_MOISTURE       = 0x07,
    VIBRATION           = 0x08,
    VOLTAGE             = 0x09,
    CURRENT             = 0x0A,
    SOUND_LEVEL         = 0x0B,
    PM25                = 0x0C,
    WATER_FLOW          = 0x0D,
    // 0x0E-0x7F: reserved for future physical quantities
    BINARY_DOOR         = 0x80,     // discrete: open/close
    BINARY_MOTION       = 0x81,     // discrete: detected/not
    BINARY_BUTTON       = 0x82,     // discrete: pressed/not
    // 0x83-0xFE: reserved for future discrete types
    CUSTOM              = 0xFF      // user-defined
}
```

ES-1.4 Physical Bounds Table (compiled into firmware)

```
struct PhysicalBounds {
    quantity:           QuantityType,
    absolute_min:       i16,        // reading_t: × 100
    absolute_max:       i16,        // reading_t: × 100
    max_spatial_grad:   u16,        // × 100 per meter
    max_temporal_rate:  u16,        // × 100 per second
    typical_sigma:      u16,        // × 100, manufacturer uncertainty
}

// Compiled table (Stratum 0 innate rules):
BOUNDS_TABLE[] = {
    // qty              abs_min     abs_max     spat_grad   temp_rate
sigma
    { TEMPERATURE,      -27315,     100000,     500,        83,         50
},
    //                  -273.15°C   1000.00°C   5.00°C/m    0.83°C/s
0.50°C
    { HUMIDITY,         0,          10000,      1000,       83,         200
},
    //                  0.00%       100.00%     10.00%/m    0.83%/s
2.00%
    { PRESSURE,         30000,      110000,     1,          0,          10
},
    //                  300.00hPa   1100.00hPa  0.01hPa/m   ~0
0.10hPa
    { CO2,              0,          500000,     10000,      83,
3000},
    //                  0ppm        5000.00ppm  100.00/m    0.83/s
30.00ppm
    { LIGHT,            0,          10000000,   -1,         -1,         -1
},
    //                  (highly variable — innate only checks absolute
bounds)
    { SOUND_LEVEL,      0,          19400,      2000,       4000,       150
},
    //                  0.00dBA     194.00dBA   20.00/m     40.00/s
1.50dBA
    { PM25,             0,          100000,     5000,       16667,
1000},
    //                  0μg/m³      1000.00     50.00/m     166.67/s
10.00
}
// -1 = check disabled for that dimension (use only absolute bounds)
```

ES-2. Wire Formats

All messages share a common header. Multi-byte values are little-endian. All messages are signed.

ES-2.1 Message Header (12 bytes)

```
Offset  Size  Field            Description
──────────────────────────────────────────
0       2     magic            0x4B42 ("KB")
2       1     version          PROTOCOL_VERSION (0x01)
3       1     msg_type         MessageType enum
4       8     sender_id        device_id of sender
──────────────────────────────────────────
Total: 12 bytes
```

ES-2.2 Message Types

```
enum MessageType : u8 {
    // Stigmergic Consensus
    TRACE_DEPOSIT       = 0x01,     // sensor reading + pheromone trace
    CONSISTENCY_ALERT   = 0x02,     // physical consistency violation
detected

    // Quorum Sensing
    PRESENCE_PULSE      = 0x10,     // minimal heartbeat
    MODE_ANNOUNCE       = 0x11,     // current operating mode

    // Immune System
    PYREXIA_NOTICE      = 0x20,     // 1-2σ deviation noticed
    INFLAMMATION_QUERY  = 0x21,     // 2-3σ, requesting neighbor
confirmation
    INFLAMMATION_REPLY  = 0x22,     // neighbor's response to query
    REJECTION_NOTICE    = 0x23,     // >3σ, device isolated
    NK_AUDIT_REQUEST    = 0x24,     // random audit challenge
    NK_AUDIT_RESPONSE   = 0x25,     // audit result
    VACCINE             = 0x26,     // attack signature propagation
    PROFILE_SYNC        = 0x27,     // behavioral profile exchange between
neighbors

    // Metabolic State
    HEARTBEAT           = 0x30,     // proof of life (refreshes STRUCTURAL
state)
    DEATH_CERTIFICATE   = 0x31,     // device apoptosis announcement
    DATA_RETRACTION     = 0x32,     // data apoptosis

    // HGT (future)
    MODULE_ADVERTISE    = 0x40,
    MODULE_REQUEST      = 0x41,
    MODULE_TRANSFER     = 0x42,
    MODULE_FITNESS      = 0x43,

    // Clock sync
    TIME_SYNC_REQUEST   = 0x50,
    TIME_SYNC_RESPONSE  = 0x51,
}
```

ES-2.3 TRACE_DEPOSIT (primary message — most frequent)

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x01
12      4     timestamp           u32, seconds since epoch
16      1     quantity_type       QuantityType enum
17      2     reading             i16, fixed-point × 100
19      1     confidence          u8, 0-255 → 0.0-1.0
20      1     weight              u8, immunological weight
21      1     seq_num             u8, wrapping sequence number (monotonic
counter)
22      4     prev_hash           hash20[0..4], link to previous own trace
(DAG parent)
24      64    signature           Ed25519(sender_privkey, bytes[0..26])
──────────────────────────────────────────────────
Total: 90 bytes
```

Dust node compressed variant (for LPWAN):

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       2     sender_id_short     device_id[0..2] (disambiguated by
gateway)
2       2     reading             i16, fixed-point × 100
4       2     timestamp_delta     u16, seconds since last trace
6       1     confidence          u8
7       1     checksum            CRC-8 of bytes[0..7]
──────────────────────────────────────────────────
Total: 8 bytes (fits LoRa SF12 payload)
```

ES-2.4 PRESENCE_PULSE (quorum sensing)

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x10
12      1     mode                current mode: 0=SOLO, 1=QUORUM, 2=SWARM
13      1     neighbor_count      u8, observed neighbor count
14      64    signature           Ed25519
──────────────────────────────────────────────────
Total: 78 bytes
```

ES-2.5 CONSISTENCY_ALERT

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x02
12      8     suspect_id          device_id of suspect device
20      2     suspect_reading     i16
22      2     expected_max        i16, max permissible reading given
physics
24      2     expected_min        i16, min permissible reading given
physics
26      1     severity            0=PYREXIA, 1=INFLAMMATION, 2=REJECTION
27      64    signature           Ed25519
──────────────────────────────────────────────────
Total: 91 bytes
```

ES-2.6 INFLAMMATION_QUERY

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x21
12      8     suspect_id          device_id under investigation
20      1     quantity_type       QuantityType
21      2     suspect_reading     i16
23      2     reporter_reading    i16, sender's own recent reading
25      4     timestamp           when anomaly observed
29      64    signature           Ed25519
──────────────────────────────────────────────────
Total: 93 bytes
```

ES-2.7 INFLAMMATION_REPLY

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x22
12      8     suspect_id          device_id under investigation
20      1     verdict             0=CONFIRM_ANOMALY, 1=DENY_ANOMALY,
2=ABSTAIN
21      2     own_reading         i16, responder's own reading of same
quantity
23      1     own_confidence      u8
24      64    signature           Ed25519
──────────────────────────────────────────────────
Total: 88 bytes
```

ES-2.8 VACCINE

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x26
12      4     origin_timestamp    when attack was first detected
16      8     origin_cluster      cluster identifier (hash of founding
device set)
24      1     attack_type         enum: 0=VALUE_SPIKE, 1=SLOW_DRIFT,
2=PATTERN_CHANGE,
                                       3=FLOOD, 4=IMPERSONATION

25      1     hops_from_origin    u8, incremented at each relay (TTL = 10 -
hops)
26      20    attack_signature    condensed feature vector of the attack:
                                    [quantity_type(1), deviation_mean(2),
deviation_std(2),
                                     temporal_pattern(4),
affected_dims_bitfield(2),
                                     reserved(9)]
46      1     vaccine_strength    u8, decays: strength = initial × e^{-
hops/d₀}
47      64    signature           Ed25519 of originating node (NOT relaying
node)
──────────────────────────────────────────────────
Total: 111 bytes
```

ES-2.9 DEATH_CERTIFICATE

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x31
12      1     reason              0=SELF_COMPROMISE, 1=PUF_ANOMALY,
2=BOOT_FAIL,
                                  3=OPERATOR_COMMAND, 4=ENERGY_DEPLETED
13      4     timestamp           time of death
17      32    old_pubkey_hash     hash of public key being invalidated
49      64    signature           signed with OLD key (proof of voluntary
death)
──────────────────────────────────────────────────
Total: 113 bytes
```

ES-2.10 HEARTBEAT (proof of life)

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x30
12      4     timestamp           current time
16      4     uptime              seconds since last boot
20      1     battery_level       u8, 0-255 (0xFF = mains powered)
21      1     mode                current quorum mode
22      2     dag_tip_hash        hash20[0..2] of latest DAG entry
24      64    signature           Ed25519
──────────────────────────────────────────────────
Total: 88 bytes
```

ES-2.11 PROFILE_SYNC (behavioral profile exchange)

```
Offset  Size  Field               Description
──────────────────────────────────────────────────
0       12    header              msg_type = 0x27
12      8     subject_id          device being profiled
20      2     profile_version     u16, incremented on update
22      1     component_id        which profile component:
                                    0=VALUES, 1=TEMPORAL, 2=COMM,
                                    3=ENERGY, 4=CORRELATION
23      1     bin_count           number of histogram bins (≤50)
24      N     bins                bin_count × 2 bytes (u16 counts)
24+N    64    signature           Ed25519
──────────────────────────────────────────────────
Total: 88 + N bytes (max 188 bytes for 50 bins)
```

ES-3. State Machines

ES-3.1 Node Lifecycle State Machine

```
                      ┌────────────┐
           ┌────────→│  OFFLINE    │←────────┐
           │          └─────┬──────┘          │
           │                │ power on +          │ energy depleted /
           │                │ PUF extract         │ operator shutdown
```

```
         │                 ▼                    │
         │         ┌────────────┐               │
         │         │  BOOTING   │               │
         │         └──────┬─────┘               │
         │                │ PUF OK +            │
         │                │ crypto init         │
         │                ▼                     │
         │         ┌────────────┐               │
    death cert     │  NEONATAL  │───────────────┤
    issued         │  (14 days) │  PUF fail     │
         │         └──────┬─────┘               │
         │                │ neonatal_timer      │
         │                │ expires             │
         │                ▼                     │
         │         ┌────────────┐               │
         │    ┌───→│  NORMAL    │←──────┐       │
         │    │    └──┬───┬──┬──┘       │       │
         │    │       │   │  │          │       │
         │    │  1-2σ │   │  │ >3σ+deny │       │
         │    │       │   │  │          │       │
         │    │       ▼   │  ▼          │       │
         │    │  ┌───────┐│ ┌─────────┐ │       │
         │    │  │PYREXIA││ │REJECTION│ │       │
         │    │  └──┬────┘│ └───┬─────┘ │       │
         │    │     │     │     │       │       │
         │    │  <1σ│     │  remediate  │       │
         │    │     │     │  success    │       │
         │    │     └─────┤     │       │       │
         │    │           │     └───────┘       │
         │    │      2-3σ │                     │
         │    │           ▼                     │
         │    │    ┌─────────────┐              │
         │    │    │INFLAMMATION │              │
         │    │    └──┬────┬─────┘              │
         │    │       │    │                    │
         │    │  confirm   deny+>3σ             │
         │    │  (update   │                    │
         │    │  profile)  └──→REJECTION        │
         │    │       │                         │
         │    └───────┘                         │
         │                                      │
         │    ┌─────────────┐                   │
         │    │ QUARANTINE  │←─ REJECTION       │
         │    │ (isolated)  │   timeout         │
         │    └─────┬───────┘  (72h)            │
         │          │                           │
         │   auto-remediate                     │
         │   success → NEONATAL (restart)       │
         │   fail    → APOPTOTIC ───────────────┘
         │          │
         │    ┌─────▼───────┐
         └────│  APOPTOTIC  │
              │ (death cert)│
              └─────────────┘
```

State definitions:

```
struct NodeState {
    state:              enum { OFFLINE, BOOTING, NEONATAL, NORMAL,
                               PYREXIA, INFLAMMATION, REJECTION,
                               QUARANTINE, APOPTOTIC },
    entered_at:         timestamp,      // when entered current state

    // Neonatal tracking
    neonatal_start:     timestamp,
    profile_frozen:     bool,           // true after neonatal ends

    // Immune tracking
    deviation_history:  RingBuffer<f16, 64>,   // last 64 deviation values
    consecutive_pyrexia: u16,           // epochs in PYREXIA
    inflammation_query_id: u32,         // active query, or 0
    inflammation_replies: [InflammationReply; MAX_NEIGHBORS],
    inflammation_reply_count: u8,

    // Quarantine
    quarantine_reason:  u8,
    remediation_attempts: u8,           // max 3

    // Metabolic
    last_heartbeat:     timestamp,
    immunological_weight: weight_t,     // current w(i,t)
}
```

Transition rules (pseudocode):

```
fn transition(state: &mut NodeState, event: Event) {
    match (state.state, event) {
        // BOOTING
        (BOOTING, PUF_OK)          ⇒ state.state = NEONATAL;
                                     state.neonatal_start = now();
                                     state.immunological_weight = 77; //
0.3 × 255
        (BOOTING, PUF_FAIL)        ⇒ state.state = APOPTOTIC;
                                     broadcast(DEATH_CERTIFICATE,
PUF_ANOMALY);

        // NEONATAL → NORMAL
        (NEONATAL, TIMER_EXPIRED)  ⇒ if now() - state.neonatal_start ≥
NEONATAL_DURATIO
N {
                                       state.state = NORMAL;
                                       state.profile_frozen = true;
                                       state.immunological_weight = 204;
// 0.8 × 255
                                       freeze_self_signature();
                                     }

        // NORMAL → PYREXIA
        (NORMAL, DEVIATION(d))     ⇒ if d ≥ SIGMA_PYREXIA && d <
SIGMA_INFLA
MMATION {
                                       state.state = PYREXIA;
                                       state.entered_at = now();
                                       state.consecutive_pyrexia = 0;
                                       increase_monitoring_frequency(2);
                                     }

        // NORMAL → INFLAMMATION (skip PYREXIA for sudden large deviation)
        (NORMAL, DEVIATION(d))     ⇒ if d ≥ SIGMA_INFLAMMATION && d <
SIGMA_
REJECTION {
                                       state.state = INFLAMMATION;
                                       broadcast_inflammation_query();
                                     }

        // NORMAL → REJECTION (extreme deviation)
        (NORMAL, DEVIATION(d))     ⇒ if d ≥ SIGMA_REJECTION {
                                       // Don't auto-reject: always query
neighbors first
                                       state.state = INFLAMMATION;
                                       broadcast_inflammation_query();
                                     }

        // PYREXIA → NORMAL (recovery)
        (PYREXIA, DEVIATION(d))    ⟹ if d < SIGMA_PYREXIA {
                                       state.state = NORMAL;
                                       restore_monitoring_frequency();
                                     }

        // PYREXIA → INFLAMMATION (escalation)
        (PYREXIA, DEVIATION(d))    ⟹ if d ⩾ SIGMA_INFLAMMATION {
                                       state.state = INFLAMMATION;
                                       broadcast_inflammation_query();
                                     }

        // PYREXIA timeout: auto-escalate after 6 hours
        (PYREXIA, TIMER_EXPIRED)   ⟹ if now() - state.entered_at > 21600 {
                                       state.state = INFLAMMATION;
                                       broadcast_inflammation_query();
                                     }

        // INFLAMMATION: collect replies
        (INFLAMMATION, REPLY(r))   ⟹ {

state.inflammation_replies[state.inflammation_reply_cou
nt] = r;
                                       state.inflammation_reply_count +=
1;
                                       evaluate_inflammation();
                                     }

        // INFLAMMATION timeout: decide with partial replies after 5
minutes
        (INFLAMMATION, TIMER_EXPIRED) ⟹ if now() - state.entered_at > 300
{
                                           evaluate_inflammation();
                                         }

        // REJECTION → QUARANTINE after 72 hours
        (REJECTION, TIMER_EXPIRED) ⟹ if now() - state.entered_at > 259200
{
                                       state.state = QUARANTINE;
                                       state.remediation_attempts = 0;
                                     }

        // QUARANTINE: attempt remediation
        (QUARANTINE, REMEDIATE_OK) ⟹ state.state = NEONATAL; // restart
lifecycle
                                     state.neonatal_start = now();
                                     state.immunological_weight = 77;

        (QUARANTINE, REMEDIATE_FAIL) ⟹ {
                                          state.remediation_attempts += 1;
                                          if state.remediation_attempts ⩾
3 {
                                            state.state = APOPTOTIC;
                                            broadcast(DEATH_CERTIFICATE,
SELF_COMPROMISE);
                                          }
                                       }

        // Heartbeat miss: any state → APOPTOTIC
        (_, HEARTBEAT_TIMEOUT)     => if now() - state.last_heartbeat >
T_HALF_STRUCTURAL
 {
                                        state.state = APOPTOTIC;
                                      }

        _ => {} // no transition
    }
}

fn evaluate_inflammation() {
    let total = state.inflammation_reply_count;
    if total == 0 { return; } // wait for at least one reply

    let confirm_count = count(replies where verdict == CONFIRM_ANOMALY);
    let deny_count = count(replies where verdict == DENY_ANOMALY);

    if deny_count > total / 2 {
        // Neighbors deny the change → suspect is anomalous
        if latest_deviation >= SIGMA_REJECTION {
            state.state = REJECTION;
            state.immunological_weight = 0;
            isolate_from_consensus();
        } else {
            // Moderate anomaly, neighbors disagree
            state.state = PYREXIA; // downgrade, watch more
        }
    } else if confirm_count > total / 2 {
        // Neighbors confirm the change → environment shifted
        update_profile(latest_readings);
        state.state = NORMAL;
    } else {
        // Inconclusive → stay in INFLAMMATION, wait for more replies
        // Auto-resolve after 5 min timeout with best available data
    }
}
```

ES-3.2 Quorum Sensing State Machine

```
              ┌──────────────────────────┐
              │                            │
    ┌─────────▼──┐    ┌──────────┐    ┌───┴──────┐
    │   SOLO      │───→│  QUORUM   │───→│  SWARM    │
    │ (<3 nbrs)  │     │ (3-10)    │     │ (>10)     │
    │             │←───│           │←───│           │
    └────────────┘    └──────────┘    └──────────┘

Transitions (with hysteresis on UPWARD transitions, simple threshold on
DOWNWARD):
  SOLO → QUORUM:    neighbor_count ≥ THETA_LOW + HYSTERESIS_DELTA  (≥ 5)
  QUORUM → SOLO:    neighbor_count < THETA_LOW                     (< 3)
  QUORUM → SWARM:   neighbor_count ≥ THETA_HIGH + HYSTERESIS_DELTA (≥ 12)
  SWARM → QUORUM:   neighbor_count < THETA_HIGH - HYSTERESIS_DELTA (< 8)
```

```
struct QuorumState {
    mode:                enum { SOLO, QUORUM, SWARM },
    neighbor_count:      u8,               // smoothed over 3 epochs (EMA)
    neighbor_table:      [NeighborEntry; MAX_NEIGHBORS],
    last_mode_change:    timestamp,
    min_transition_interval: u32,       // 120 seconds (prevent
oscillation)
}
```

```
struct NeighborEntry {
    device_id:          device_id,
    last_seen:          timestamp,
    rssi:               i8,             // signal strength (for distance
est.)
    mode:               u8,             // neighbor's reported mode
    weight:             weight_t,       // neighbor's immunological weight
    alive:              bool,           // false if last_seen > 3 ×
PRESENCE_INTERVAL
}

fn update_quorum(qs: &mut QuorumState) {
    // Prune dead neighbors
    for n in qs.neighbor_table {
        if now() - n.last_seen > 3 * PRESENCE_INTERVAL {
            n.alive = false;
        }
    }

    let alive_count = count(qs.neighbor_table where alive == true);

    // EMA smoothing (α = 0.3)
    qs.neighbor_count = (0.3 * alive_count + 0.7 * qs.neighbor_count) as
u8;

    // Rate limit transitions
    if now() - qs.last_mode_change < qs.min_transition_interval { return; }

    match qs.mode {
        SOLO => {
            if qs.neighbor_count >= THETA_LOW + HYSTERESIS_DELTA {
                qs.mode = QUORUM;
                qs.last_mode_change = now();
            }
        }
        QUORUM => {
            if qs.neighbor_count < THETA_LOW {
                qs.mode = SOLO;
                qs.last_mode_change = now();
            } else if qs.neighbor_count >= THETA_HIGH + HYSTERESIS_DELTA {
                qs.mode = SWARM;
                qs.last_mode_change = now();
            }
        }
        SWARM => {
            if qs.neighbor_count < THETA_HIGH - HYSTERESIS_DELTA {
                qs.mode = QUORUM;
                qs.last_mode_change = now();
            }
        }
    }
}
```

ES-3.3 HGT Module Lifecycle (interface only — MVP deferred)

```
    UNKNOWN → ADVERTISED → REQUESTED → TRANSFERRING → SANDBOX → PRODUCTION
                                                         │
                                                         ├→ REJECTED
(rollback)
                                                         └→ REVOKED (key
compromise)
```

ES-4. Algorithms

ES-4.1 Physical Consistency Check

```
fn check_physical_consistency(
    my_reading: i16,        // reading_t × 100
    their_reading: i16,     // reading_t × 100
    quantity: QuantityType,
    distance_cm: u32,       // estimated distance in centimeters
    time_delta_ms: u32,     // time difference in milliseconds
) → ConsistencyResult {

    let bounds = BOUNDS_TABLE.get(quantity);
    if bounds.is_none() {
        return ConsistencyResult::UNKNOWN; // no bounds for this quantity
    }
    let b = bounds.unwrap();

    // Step 1: Innate check (Stratum 0)
    if my_reading < b.absolute_min || my_reading > b.absolute_max {
        return ConsistencyResult::REJECT_INNATE;
    }

    // Step 2: Compute maximum permissible difference
    let distance_m = distance_cm as f32 / 100.0;
    let time_delta_s = time_delta_ms as f32 / 1000.0;

    let epsilon_spatial = if b.max_spatial_grad < 0 {
        i16::MAX // disabled, skip spatial check
    } else {
        (b.max_spatial_grad as f32 * distance_m) as i16
    };

    let epsilon_temporal = if b.max_temporal_rate < 0 {
        i16::MAX
    } else {
        (b.max_temporal_rate as f32 * time_delta_s) as i16
    };

    let sigma_sum = 2 * b.typical_sigma; // σ_i + σ_j (assume same sensor
type)

    let max_diff = epsilon_spatial
        .saturating_add(epsilon_temporal)
        .saturating_add(sigma_sum as i16);

    // Step 3: Compare
    let actual_diff = (my_reading - their_reading).abs();

    if actual_diff ≤ max_diff {
        ConsistencyResult::CONSISTENT
    } else {
        let deviation_sigma = (actual_diff - max_diff) as f32
                              / (b.typical_sigma as f32);
        if deviation_sigma < 1.0 {
            ConsistencyResult::CONSISTENT // within noise
        } else if deviation_sigma < 2.0 {
            ConsistencyResult::PYREXIA(deviation_sigma)
        } else if deviation_sigma < 3.0 {
            ConsistencyResult::INFLAMMATION(deviation_sigma)
        } else {
            ConsistencyResult::REJECTION(deviation_sigma)
        }
    }
}

enum ConsistencyResult {
    CONSISTENT,
    REJECT_INNATE,              // Stratum 0: physically impossible
    PYREXIA(f32),               // 1-2σ: elevated, watch
    INFLAMMATION(f32),          // 2-3σ: significant, query neighbors
    REJECTION(f32),             // >3σ: extreme, isolate after confirmation
    UNKNOWN,                    // no bounds available
}
```

ES-4.2 Behavioral Fingerprint (Stratum 1)

Profile construction (run on Nano node for each monitored Dust node):

```
struct BehavioralProfile {
    subject_id:         device_id,

    // Component 0: Value distribution (histogram)
    value_hist:         [u16; MAX_PROFILE_BINS],  // 50 bins
    value_min:          i16,                       // histogram range min
    value_max:          i16,                       // histogram range max
    value_count:        u32,                       // total readings
ingested
    value_mean:         f32,                       // running mean
    value_var:          f32,                       // running variance
(Welford's)

    // Component 1: Temporal pattern
    hourly_activity:    [u16; 24],    // readings per hour-of-day bucket
    avg_interval_ms:    u32,          // mean inter-reading interval
    interval_var_ms:    u32,          // variance of intervals

    // Component 2: Communication pattern
    unique_neighbors:   u8,           // distinct neighbors contacted per
day
    msg_rate_per_epoch: f16,          // messages per epoch (smoothed)

    // Component 3: Energy signature
    battery_trend:      i8,           // -128..+127: mV/day change
    active_energy_ratio: f16,         // energy during active vs sleep

    // Component 4: Cross-correlation with neighbors
    correlation_scores: [f16; MAX_NEIGHBORS], // Pearson corr with each
neighbor

    // Metadata
    profile_version:    u16,
    last_updated:       timestamp,
    is_frozen:          bool,         // true = self-signature (immutable)
}

// Total size: ~500 bytes (as specified in architecture doc)
```

Anomaly detection algorithm (Mahalanobis-inspired, dimension-reduced):

```
fn compute_deviation(
    profile: &BehavioralProfile,
    new_reading: &TraceDeposit,
    recent_readings: &RingBuffer<TraceDeposit, 32>,
) -> f32 {
    // Deviation score: weighted sum of per-dimension z-scores

    let mut total_deviation = 0.0_f32;
    let mut total_weight = 0.0_f32;

    // Dimension 0: Value deviation (weight: 0.4)
    let z_value = if profile.value_var > 0.0 {
        ((new_reading.reading as f32 / 100.0) - profile.value_mean).abs()
        / profile.value_var.sqrt()
    } else {
        0.0
    };
    total_deviation += 0.4 * z_value;
    total_weight += 0.4;

    // Dimension 1: Timing deviation (weight: 0.2)
    if let Some(prev) = recent_readings.last() {
        let interval_ms = (new_reading.timestamp - prev.timestamp) * 1000;
        let z_timing = if profile.interval_var_ms > 0 {
            (interval_ms as f32 - profile.avg_interval_ms as f32).abs()
            / (profile.interval_var_ms as f32).sqrt()
        } else {
            0.0
        };
        total_deviation += 0.2 * z_timing;
        total_weight += 0.2;
    }

    // Dimension 2: Neighbor correlation deviation (weight: 0.3)
    // Compare reading against expected value from correlated neighbors
    let expected_from_neighbors = compute_neighbor_expectation(
        profile, recent_readings
    );
    if let Some(expected) = expected_from_neighbors {
        let z_corr = (new_reading.reading as f32 / 100.0 - expected).abs()
                     / (profile.value_var.sqrt() + 0.01); // avoid div/0
        total_deviation += 0.3 * z_corr;
        total_weight += 0.3;
    }

    // Dimension 3: Communication pattern deviation (weight: 0.1)
    // (computed per epoch, not per reading — skip here, add at epoch
boundary)

    // Normalize
    if total_weight > 0.0 {
        total_deviation / total_weight
    } else {
        0.0
    }
}

fn compute_neighbor_expectation(
    profile: &BehavioralProfile,
    recent_readings: &RingBuffer<TraceDeposit, 32>,
) -> Option<f32> {
    // Weighted average of neighbor readings, weighted by historical
correlation
    let mut weighted_sum = 0.0_f32;
    let mut weight_sum = 0.0_f32;

    for (i, neighbor) in neighbor_table.iter().enumerate() {
        if !neighbor.alive { continue; }
        let corr = profile.correlation_scores[i] as f32;
        if corr.abs() < 0.1 { continue; } // ignore uncorrelated neighbors

        if let Some(their_reading) = get_latest_reading(neighbor.device_id)
{
            weighted_sum += corr * (their_reading as f32 / 100.0);
            weight_sum += corr.abs();
        }
    }

    if weight_sum > 0.5 { // need sufficient correlated data
        Some(weighted_sum / weight_sum)
    } else {
        None
    }
}
```

Profile update (Welford's online algorithm):

```
fn update_profile(profile: &mut BehavioralProfile, reading: &TraceDeposit)
{
    if profile.is_frozen { return; } // self-signature: never update

    let value = reading.reading as f32 / 100.0;

    // Welford's online mean + variance
    profile.value_count += 1;
    let n = profile.value_count as f32;
    let delta = value - profile.value_mean;
    profile.value_mean += delta / n;
    let delta2 = value - profile.value_mean;
    profile.value_var += (delta * delta2 - profile.value_var) / n;
    // Note: for running profile (not frozen), use EMA instead for
adaptivity:
    // profile.value_mean = 0.99 * profile.value_mean + 0.01 * value;
    // profile.value_var  = 0.99 * profile.value_var  + 0.01 * (value -
profile.value_mean)^2;

    // Update histogram
    let bin_width = (profile.value_max - profile.value_min) as f32
                  / MAX_PROFILE_BINS as f32;
    let bin_idx = ((reading.reading - profile.value_min) as f32 /
bin_width) as usize;
    let bin_idx = bin_idx.clamp(0, MAX_PROFILE_BINS - 1);
    profile.value_hist[bin_idx] =
profile.value_hist[bin_idx].saturating_add(1);

    // Update temporal pattern
    let hour = (reading.timestamp % 86400) / 3600;
    profile.hourly_activity[hour as usize] += 1;

    profile.last_updated = reading.timestamp;
    profile.profile_version += 1;
}
```

ES-4.3 NK Random Audit (Stratum 4)

```
fn maybe_run_nk_audit(
    device_id: device_id,
    self_signature: &BehavioralProfile,   // frozen at end of neonatal
    current_profile: &BehavioralProfile,  // mutable, current
    epoch_rng: u8,                        // random byte from RNG
) -> Option<NkAuditResult> {

    // Probability check: p_audit ≈ 0.012
    if epoch_rng > NK_AUDIT_PROBABILITY { return None; }

    // Compare current profile against immutable self-signature
    let drift = compute_profile_drift(self_signature, current_profile);

    if drift > DRIFT_THRESHOLD as f32 / 10.0 {
        Some(NkAuditResult::DRIFT_DETECTED {
            device_id,
            drift_sigma: drift,
            dimensions: identify_drifted_dimensions(self_signature,
current_profile),
        })
    } else {
        Some(NkAuditResult::PASS)
    }
}

fn compute_profile_drift(
    baseline: &BehavioralProfile,
    current: &BehavioralProfile,
) -> f32 {
    // L2 distance across all dimensions, normalized by baseline variance

    let mut drift_sq = 0.0_f32;

    // Value distribution drift
    let mean_drift = (current.value_mean - baseline.value_mean).abs()
                    / (baseline.value_var.sqrt() + 0.01);
    drift_sq += mean_drift * mean_drift;

    // Variance drift (has the noise profile changed?)
    let var_ratio = if baseline.value_var > 0.01 {
        current.value_var / baseline.value_var
    } else {
        1.0
    };
    let var_drift = (var_ratio - 1.0).abs() * 2.0; // scaled: 50% change =
1σ
    drift_sq += var_drift * var_drift;

    // Timing drift
    let timing_drift = if baseline.avg_interval_ms > 0 {
        ((current.avg_interval_ms as f32 - baseline.avg_interval_ms as
f32).abs())
        / (baseline.interval_var_ms as f32).sqrt().max(1.0)
    } else {
        0.0
    };
    drift_sq += timing_drift * timing_drift;

    // Communication pattern drift
    let comm_drift = (current.unique_neighbors as f32 -
baseline.unique_neighbors as f32).abs()
 / 3.0;
```

```
    drift_sq += comm_drift * comm_drift;

    drift_sq.sqrt() // total multi-dimensional drift in σ units
}

fn identify_drifted_dimensions(
    baseline: &BehavioralProfile,
    current: &BehavioralProfile,
) -> u8 {
    // Bitfield: bit 0 = value, bit 1 = timing, bit 2 = comm, bit 3 =
energy
    let mut dims = 0u8;

    let mean_drift = (current.value_mean - baseline.value_mean).abs()
                     / (baseline.value_var.sqrt() + 0.01);
    if mean_drift > 2.0 { dims |= 0x01; }

    let timing_drift = (current.avg_interval_ms as f32 -
baseline.avg_interval_ms as f32).abs()
                       / (baseline.interval_var_ms as f32).sqrt().max(1.0);
    if timing_drift > 2.0 { dims |= 0x02; }

    let comm_drift = (current.unique_neighbors as f32 -
baseline.unique_neighbors as f32).abs()
 / 3.0;
    if comm_drift > 2.0 { dims |= 0x04; }

    dims
}
```

ES-4.4 Entropic Data Valuation (interface only — runs on Full nodes)

```
fn compute_data_value(
    reading: &TraceDeposit,
    local_readings: &[TraceDeposit],    // recent readings from neighbors
    health: weight_t,                    // immunological weight of sender
) -> f32 {
    // Approximate I(r; World | Network_State) using local entropy
reduction

    let prior_entropy = estimate_entropy(local_readings);

    // Add new reading to set, recompute entropy
    let mut augmented = local_readings.to_vec();
    augmented.push(*reading);
    let posterior_entropy = estimate_entropy(&augmented);

    let information_gain = (prior_entropy - posterior_entropy).max(0.0);
    let health_f = health as f32 / 255.0;

    information_gain * health_f
}

fn estimate_entropy(readings: &[TraceDeposit]) -> f32 {
    // Histogram-based entropy estimation
    // Bin width = typical_sigma for the quantity type
    if readings.is_empty() { return 0.0; }

    let n = readings.len() as f32;
    let bounds = BOUNDS_TABLE.get(readings[0].quantity_type);
    let bin_width = bounds.map(|b| b.typical_sigma).unwrap_or(100) as f32;
    // Build histogram
    let mut bins: HashMap<i16, u32> = HashMap::new();
    for r in readings {
        let bin = r.reading / bin_width as i16;
        *bins.entry(bin).or_insert(0) += 1;
    }

    // Shannon entropy
    let mut h = 0.0_f32;
    for &count in bins.values() {
        let p = count as f32 / n;
        if p > 0.0 {
            h -= p * p.log2();
        }
    }
    h
}
```

ES-4.5 Pheromone Field Computation

```
struct PheromoneField {
    entries:    [PheromoneEntry; MAX_DAG_ENTRIES],
    count:      u16,
    lambda:     f32,    // decay rate (LAMBDA_DEFAULT or LAMBDA_LPWAN)
}

struct PheromoneEntry {
    device_id:  device_id,
    reading:    i16,
    timestamp:  timestamp,
    weight:     weight_t,
    quantity:   QuantityType,
    seq:        u8,
}

fn query_field(
    field: &PheromoneField,
    quantity: QuantityType,
    now: timestamp,
) -> FieldQuery {
    let mut weighted_sum = 0.0_f32;
    let mut weight_sum = 0.0_f32;
    let mut count = 0u16;
    let mut min = i16::MAX;
    let mut max = i16::MIN;

    for entry in field.entries[..field.count as usize].iter() {
        if entry.quantity != quantity { continue; }

        let age = (now - entry.timestamp) as f32;
        let decay = (-field.lambda * age).exp(); // e^{-λ(t-t_i)}

        if decay < 0.03 { continue; } // below 3% = effectively dead

        let w = (entry.weight as f32 / 255.0) * decay;
        weighted_sum += w * (entry.reading as f32);
        weight_sum += w;
        count += 1;
        min = min.min(entry.reading);
        max = max.max(entry.reading);
    }
```

```rust
    FieldQuery {
        consensus_value: if weight_sum > 0.0 {
            Some((weighted_sum / weight_sum) as i16)
        } else {
            None
        },
        active_traces: count,
        range: (min, max),
        total_weight: weight_sum,
    }
}

fn deposit_trace(
    field: &mut PheromoneField,
    trace: &TraceDeposit,
) {
    // Garbage collect: remove entries with decay < 3%
    let now = trace.timestamp;
    let mut write_idx = 0;
    for read_idx in 0..field.count as usize {
        let age = (now - field.entries[read_idx].timestamp) as f32;
        let decay = (-field.lambda * age).exp();
        if decay >= 0.03 {
            field.entries[write_idx] = field.entries[read_idx];
            write_idx += 1;
        }
    }
    field.count = write_idx as u16;

    // Add new entry
    if (field.count as usize) < MAX_DAG_ENTRIES {
        field.entries[field.count as usize] = PheromoneEntry {
            device_id: trace.sender_id,
            reading: trace.reading,
            timestamp: trace.timestamp,
            weight: trace.weight,
            quantity: trace.quantity_type,
            seq: trace.seq_num,
        };
        field.count += 1;
    } else {
        // Evict oldest entry
        let mut oldest_idx = 0;
        let mut oldest_time = u32::MAX;
        for i in 0..field.count as usize {
            if field.entries[i].timestamp < oldest_time {
                oldest_time = field.entries[i].timestamp;
                oldest_idx = i;
            }
        }
        field.entries[oldest_idx] = PheromoneEntry {
            device_id: trace.sender_id,
            reading: trace.reading,
            timestamp: trace.timestamp,
            weight: trace.weight,
            quantity: trace.quantity_type,
            seq: trace.seq_num,
        };
    }
}
```

ES-4.6 Morphogenetic Signals (interface only — runs on Nano+)

```
struct MorphogeneticState {
    activator:      f16,    // A: local "more connectivity needed" signal
    inhibitor:      f16,    // I: local "capacity saturated" signal

    // Parameters (from governance or self-tuning)
    d_a:            f16,    // activator diffusion rate (SLOW)
    d_i:            f16,    // inhibitor diffusion rate (FAST, d_i > d_a)
    margin:         f16,    // decision margin
}

fn update_morphogenetic(
    state: &mut MorphogeneticState,
    packet_drop_rate: f16,      // local metric
    latency_ratio: f16,         // current / threshold
    energy_reserve: f16,        // 0.0-1.0
    neighbor_A: &[f16],         // activator from neighbors
    neighbor_I: &[f16],         // inhibitor from neighbors
    dt: f32,                    // time step
) {
    // Reaction terms
    let f_a = packet_drop_rate + (latency_ratio - 1.0).max(0.0); // need
drives activator
    let f_i = (1.0 - energy_reserve) + bandwidth_saturation();     // cost
drives inhibitor

    // Diffusion: discrete Laplacian on neighbor graph
    let laplacian_A = neighbor_A.iter().map(|a| *a as f32).sum::<f32>() /
neighbor_A.len(
) as f32
                      - state.activator as f32;
    let laplacian_I = neighbor_I.iter().map(|i| *i as f32).sum::<f32>() /
neighbor_I.len(
) as f32
                      - state.inhibitor as f32;

    // Euler step for reaction-diffusion
    let new_A = state.activator as f32
                + dt * (state.d_a as f32 * laplacian_A + f_a as f32);
    let new_I = state.inhibitor as f32
                + dt * (state.d_i as f32 * laplacian_I + f_i as f32);

    // Clamp to [0, 1]
    state.activator = new_A.clamp(0.0, 1.0) as f16;
    state.inhibitor = new_I.clamp(0.0, 1.0) as f16;
}

fn morphogenetic_decision(state: &MorphogeneticState) -> TopologyAction {
    let diff = state.activator as f32 - state.inhibitor as f32;
    if diff > state.margin as f32 {
        TopologyAction::EXPAND  // increase range/frequency/relay
    } else if diff < -(state.margin as f32) {
        TopologyAction::CONTRACT // decrease range/frequency/relay
    } else {
        TopologyAction::HOLD
    }
}
```

ES-5. DAG Structure and Synchronization

ES-5.1 Local DAG Structure

Each node maintains a local DAG. The DAG is **not shared globally** — it is a local data structure representing the pheromone field.

```
struct LocalDAG {
    entries:        [DAGEntry; MAX_DAG_ENTRIES],
    count:          u16,
    tip_hash:       hash20,         // hash of latest entry
    my_last_seq:    u8,             // my latest sequence number
}

struct DAGEntry {
    hash:           hash20,         // SHA-1(entry contents)
    parent_hash:    hash20,         // previous entry by SAME device
    device_id:      device_id,
    timestamp:      timestamp,
    quantity:       QuantityType,
    reading:        i16,
    confidence:     confidence_t,
    weight:         weight_t,
    seq:            u8,
}
// Size per entry: 20 + 20 + 8 + 4 + 1 + 2 + 1 + 1 + 1 = 58 bytes
// Dust (256 entries): 256 × 58 = ~14.5 KB (stored in flash, not RAM)
// Nano (4096 entries): 4096 × 58 = ~232 KB (stored in flash)
```

ES-5.2 DAG Operations

```
fn dag_append(dag: &mut LocalDAG, trace: &TraceDeposit) → hash20 {
    let parent = dag_find_latest_by_device(dag, trace.sender_id);

    let entry = DAGEntry {
        hash:           sha1_truncated(&serialize(trace, parent)),
        parent_hash:    parent.map(|p| p.hash).unwrap_or([0u8; 20]),
        device_id:      trace.sender_id,
        timestamp:      trace.timestamp,
        quantity:       trace.quantity_type,
        reading:        trace.reading,
        confidence:     trace.confidence,
        weight:         trace.weight,
        seq:            trace.seq_num,
    };

    // Insert (with GC if full)
    if dag.count as usize ≥ MAX_DAG_ENTRIES {
        dag_gc(dag, trace.timestamp);
    }

    dag.entries[dag.count as usize] = entry;
    dag.count += 1;
    dag.tip_hash = entry.hash;

    entry.hash
}

fn dag_gc(dag: &mut LocalDAG, now: timestamp) {
    // Remove entries older than 5 × T½_EPHEMERAL (effectively dead)
    let cutoff = now.saturating_sub(5 * T_HALF_EPHEMERAL);

    let mut write = 0usize;
    for read in 0..dag.count as usize {
        if dag.entries[read].timestamp >= cutoff {
            dag.entries[write] = dag.entries[read];
            write += 1;
        }
    }
    dag.count = write as u16;

    // If still full after time-based GC, evict lowest-weight entries
    if dag.count as usize >= MAX_DAG_ENTRIES {
        // Sort by weight × recency, keep top 75%
        dag.entries[..dag.count as usize].sort_by(|a, b| {
            let score_a = (a.weight as u32) * (a.timestamp - cutoff);
            let score_b = (b.weight as u32) * (b.timestamp - cutoff);
            score_b.cmp(&score_a)
        });
        dag.count = (dag.count as f32 * 0.75) as u16;
    }
}
```

ES-5.3 DAG Synchronization on Reconnect

When a node reconnects after being offline:

```
fn dag_sync(my_dag: &mut LocalDAG, neighbor: &Connection) {
    // Step 1: Exchange tip hashes
    send(neighbor, TimeSyncRequest { my_tip: my_dag.tip_hash });
    let their_tip = receive(neighbor); // TimeSyncResponse

    // Step 2: If tips match, no sync needed
    if my_dag.tip_hash == their_tip { return; }

    // Step 3: Find divergence point
    // Send my last N entry hashes (N = 32 for efficiency)
    let my_recent: Vec<hash20> = my_dag.entries[..]
        .iter().rev().take(32).map(|e| e.hash).collect();
    send(neighbor, my_recent);

    // Step 4: Neighbor responds with entries I'm missing
    let missing_entries: Vec<DAGEntry> = receive(neighbor);

    // Step 5: Validate and insert
    for entry in missing_entries {
        // Verify hash integrity
        if sha1_truncated(&serialize_entry(&entry)) != entry.hash {
            continue; // corrupted, skip
        }
        // Check if we already have it
        if dag_contains(my_dag, entry.hash) {
            continue;
        }
        // Insert (bypasses normal trace processing — these are historical)
        dag_insert_historical(my_dag, entry);
    }
}
```

ES-5.4 Dust Node DAG (Minimal)

Dust nodes maintain a **micro-DAG**: only their own traces + most recent trace from each neighbor.

```
struct DustDAG {
    my_traces:          RingBuffer<DAGEntry, 64>,   // last 64 own traces
    neighbor_latest:    [DAGEntry; MAX_NEIGHBORS],  // 1 per neighbor
    neighbor_count:     u8,
}
// RAM: 64 × 58 + 32 × 58 = ~5.6 KB (stored in flash on 32KB device)
// Active RAM: ~200 bytes (current trace + working memory)
```

ES-6. Blockchain Integration API

ES-6.1 Nano → Full Node Interface

Nano nodes batch traces and submit to Full nodes for on-chain settlement.

```
struct TraceBatch {
    batch_id:           u32,
    nano_id:            device_id,      // submitting Nano node
    epoch_start:        timestamp,
    epoch_end:          timestamp,
    trace_count:        u16,
    traces:             [BatchedTrace; 256], // max 256 per batch
    aggregate:          BatchAggregate,
    signature:          sig64,          // Nano's signature over batch
}

struct BatchedTrace {
    device_id:          device_id,
    quantity:           QuantityType,
    reading_mean:       i16,            // mean of readings this epoch
    reading_count:      u8,             // how many readings
    confidence:         confidence_t,   // weighted average confidence
    anomaly_flags:      u8,             // bitfield: 0=none, 1=pyrexia,
2=inflam, 4=reject
}
// Size per trace: 8 + 1 + 2 + 1 + 1 + 1 = 14 bytes
// Batch of 256: 256 × 14 + overhead = ~3.7 KB

struct BatchAggregate {
    total_devices:      u16,
    healthy_devices:    u16,            // weight > 0.5
    anomalies_detected: u16,
    rejections:         u8,
    field_entropy:      f16,            // pheromone field entropy
}
```

ES-6.2 On-Chain vs Off-Chain

```
ON-CHAIN (via Full node BFT consensus):
  - BatchAggregate (summary statistics)
  - Device identity events: registration, death certificates, key rotations
  - PLANKTON settlement: rewards per device per epoch
  - Governance parameter changes
  - Module hashes (HGT audit trail)

OFF-CHAIN (local DAGs, never on-chain):
  - Individual sensor readings (traces)
  - Behavioral profiles
  - Pheromone field state
  - Immune system state (anomaly history, inflammation queries)
  - Raw presence pulses

SETTLEMENT (per epoch):
  - Nano submits TraceBatch to Full node
  - Full node validates: batch signature OK, device_ids are registered,
    anomaly_flags are consistent with known immune state
  - Full node computes PLANKTON rewards:
      for each trace in batch:
        if anomaly_flags == 0:
          reward = base_reward × data_value(trace)
        else:
          reward = 0 (anomalous data not rewarded)
  - Full node submits settlement TX to blockchain:
      PLANKTON_net(nano) = Σ rewards - bandwidth_cost(batch)
```

ES-6.3 PLANKTON Settlement Formula

```
fn compute_plankton_settlement(
    batch: &TraceBatch,
    plankton_staked: u64,           // Nano's staked PLANKTON
    bandwidth_cost_per_byte: u64,   // current network rate
) -> i64 {
    let bandwidth_cost = batch.byte_size() as u64 *
bandwidth_cost_per_byte;

    let mut total_reward: u64 = 0;
    for trace in &batch.traces {
        if trace.anomaly_flags == 0 {
            let value = compute_data_value_simple(trace);
            total_reward += value;
        }
    }

    // Net = staked - cost + rewards
    plankton_staked as i64 - bandwidth_cost as i64 + total_reward as i64
}
```

ES-7. Error Handling and Edge Cases

ES-7.1 Nano Node Crash (Dust nodes lose supervisor)

```
SCENARIO: Nano node monitoring 100 Dust nodes crashes.

DETECTION: Dust nodes stop receiving presence pulses from Nano.
  After 3 × PRESENCE_INTERVAL (30s): Nano marked as dead in neighbor table.

IMMEDIATE EFFECT:
  - Dust nodes lose Strata 1-2 protection (behavioral fingerprint, adaptive
immunity)
  - Dust nodes retain Stratum 0 (innate) — hardcoded, no external
dependency
  - Profiles stored on crashed Nano are temporarily unavailable

RECOVERY PROTOCOL:
  1. Quorum sensing detects reduced neighbor count → may transition SWARM →
QUORUM
  2. Other Nano nodes in range detect orphaned Dust nodes (Dust nodes
sending traces
     but no Nano acknowledging)
  3. Nearest healthy Nano node adopts orphaned Dust nodes:
     - Begins building new profiles (enters partial neonatal for these
devices)
     - Uses cross-reference with other Nano nodes that may have partial
profiles
  4. If crashed Nano recovers: profile data restored from flash → immediate
resumption
  5. If crashed Nano is permanently lost: profiles rebuild over neonatal
period (14 days)

PROFILE REDUNDANCY:
  Each device's profile is stored by MULTIPLE Nano nodes (all Nano nodes in
range).
  Typical redundancy: 2-4 copies. Single Nano failure ≠ profile loss.

  Profile adoption protocol:
    Nano_new sends PROFILE_SYNC request to other Nano nodes for orphaned
device_ids.
    Receiving Nano responds with profile data.
    Nano_new merges profiles (weighted average, preferring higher-version
profiles).
```

ES-7.2 Neonatal Period During Network Partition

```
SCENARIO: Device in neonatal period gets partitioned from network.

PROBLEM: Neonatal timer continues, but no neighbors are building profile.

HANDLING:
  - Neonatal timer PAUSES when quorum mode = SOLO (no neighbors).
  - Timer resumes when at least THETA_LOW neighbors are detected.
  - Remaining neonatal duration stored in persistent state:
      neonatal_remaining = NEONATAL_DURATION - time_with_neighbors
  - Device stays in neonatal (w = 0.3) until total observed-by-neighbors
time
    reaches NEONATAL_DURATION.

  Edge case: device is partitioned for 6 months, then reconnects.
    - Neonatal timer has barely advanced.
    - Device starts fresh neonatal from near-zero progress.
    - This is CORRECT: the network has no basis to trust this device yet.
```

ES-7.3 PUF Degradation (Aging Silicon)

```
SCENARIO: PUF response shifts over years due to silicon aging.

DETECTION:
  - Every boot, device extracts PUF response and derives key.
  - Fuzzy extractor tolerates up to 15% bit error rate.
  - If raw PUF bit error rate exceeds 12% (approaching limit):
    → WARNING flag set internally.
  - If bit error rate exceeds 15% (fuzzy extractor fails):
    → PUF extraction fails → BOOTING → PUF_FAIL → APOPTOTIC.
    → Device issues death certificate, reboots with new PUF enrollment.

MONITORING:
  - PUF bit error rate logged each boot:
      puf_ber = hamming_distance(raw_response, enrolled_reference) /
total_bits
  - Trend tracking: if BER increases by >1% per year, operator warned.
  - Proactive re-enrollment: operator can trigger PUF re-enrollment before
failure.
    New enrollment = new identity = new neonatal period.

COST: PUF re-enrollment requires physical access (to reset helper data in
flash).
  Cannot be done remotely — this is a feature, not a bug (prevents remote
identity theft).
```

ES-7.4 Conflicting Profiles from Multiple Neighbors

```
SCENARIO: Two Nano nodes have different profiles for the same Dust device.

CAUSE: Nanos observed device at different times, from different
perspectives.

RESOLUTION:
  fn merge_profiles(
      profile_a: &BehavioralProfile,
      profile_b: &BehavioralProfile,
  ) -> BehavioralProfile {
      // Higher version number = more recent observations
      let (primary, secondary) = if profile_a.profile_version >
profile_b.profile_version {
          (profile_a, profile_b)
      } else {
          (profile_b, profile_a)
      };

      // Weighted merge: 70% primary, 30% secondary
      let mut merged = primary.clone();
      merged.value_mean = 0.7 * primary.value_mean + 0.3 *
secondary.value_mean;
      merged.value_var = 0.7 * primary.value_var + 0.3 *
secondary.value_var
                         + 0.21 * (primary.value_mean -
secondary.value_mean).powi(2);
      // (variance of mixture: Var = Σ w_i(σ_i² + μ_i²) - μ_mix²)

      // Histogram: weighted merge
      for i in 0..MAX_PROFILE_BINS {
          merged.value_hist[i] = (0.7 * primary.value_hist[i] as f32
                                + 0.3 * secondary.value_hist[i] as f32) as
u16;
      }

      merged.profile_version = primary.profile_version + 1;
      merged
  }
```

ES-7.5 HGT Module Late Failure (post-sandbox)

```
SCENARIO: Module passes 24h sandbox but causes issues after 30 days.

DETECTION:
  - Continuous fitness monitoring (not just sandbox period).
  - fitness(M) recomputed every epoch from all devices running M.
  - If fitness(M) drops below 0.5 after initial acceptance:
    → MODULE_DEGRADATION alert

RESPONSE:
  1. Module quarantined: no new installations.
  2. Devices running M: prompted to report detailed diagnostics.
  3. If fitness(M) drops below 0.3:
     → Automatic rollback on all devices.
     → Module marked as REVOKED in DAG.
  4. If fitness(M) recovers above 0.6:
     → Quarantine lifted, monitoring continues.

TIMELINE:
  - Detection: depends on fitness reporting interval (1 epoch = 60s)
  - Rollback: automatic, 1 epoch after decision
  - Total exposure: up to 30 days of degraded operation
  - Mitigation: canary deployment limits exposure to 10% of compatible
devices
```

ES-7.6 Boundary Environment: Undeclared Boundary During Neonatal

```
SCENARIO: Two sensors on opposite sides of a wall, deployed simultaneously.
Neither knows about the boundary. Both in neonatal period.

WHAT HAPPENS:
  1. During neonatal, both sensors build profiles independently.
  2. Both are marked w = 0.3 (neonatal weight).
  3. After neonatal, they begin cross-checking.
  4. FIRST CROSS-CHECK: Huge difference detected (e.g., 25°C).
  5. Both trigger INFLAMMATION.
  6. Neighbors are queried. If other neighbors exist on EACH SIDE:
     → Each side's neighbors CONFIRM their local reading.
     → Result: BOUNDARY PROFILE created automatically.
  7. If no other neighbors (just these two):
     → Inconclusive. Both remain in INFLAMMATION.
     → Timeout: after 5 min, both downgraded to PYREXIA (watch).
     → Over subsequent days, the consistent difference is learned.
     → Boundary profile emerges from repeated observation.

LEARNING TIME: 3-7 days of consistent boundary observations before
  boundary profile confidence exceeds 0.8.
```

ES-8. Test Vectors

ES-8.1 Physical Consistency Check

```
TEST VECTOR 1: Normal reading
  Input:
    my_reading = 2310        (23.10°C)
    their_reading = 2340     (23.40°C)
    quantity = TEMPERATURE
    distance_cm = 200        (2 meters)
    time_delta_ms = 5000     (5 seconds)

    Computation:
      epsilon_spatial = 500 × 2.0 = 1000 (reading_t units, i.e., 10.00°C
max gradient over 2m)
      epsilon_temporal = 83 × 5.0 = 415 (reading_t units, i.e., 4.15°C max
change over 5s)
    sigma_sum = 2 × 50 = 100
    max_diff = 1000 + 415 + 100 = 1515
    actual_diff = |2310 - 2340| = 30
    30 ≤ 1515 → CONSISTENT ✓

TEST VECTOR 2: Fire event
  Input:
    my_reading = 2310        (23.10°C)
    their_reading = 8500     (85.00°C)
    quantity = TEMPERATURE
    distance_cm = 100        (1 meter)
    time_delta_ms = 2000     (2 seconds)

  Computation:
      epsilon_spatial = 500 × 1.0 = 500
      epsilon_temporal = 83 × 2.0 = 166
    sigma_sum = 100
    max_diff = 500 + 166 + 100 = 766
    actual_diff = |2310 - 8500| = 6190
    excess = 6190 - 766 = 5424
    deviation_sigma = 5424 / 50 = 108.48σ
    → REJECTION(108.48) ✓
    (Note: neighbor confirmation will determine if this is a real fire
     or a compromised sensor)

TEST VECTOR 3: Slight anomaly
  Input:
    my_reading = 2310        (23.10°C)
    their_reading = 2480     (24.80°C)
    quantity = TEMPERATURE
    distance_cm = 50         (0.5 meters)
    time_delta_ms = 1000     (1 second)

  Computation:
      epsilon_spatial = 500 × 0.5 = 250
      epsilon_temporal = 83 × 1.0 = 83
    sigma_sum = 100
    max_diff = 250 + 83 + 100 = 433
    actual_diff = |2310 - 2480| = 170
    170 ≤ 433 → CONSISTENT ✓
    (Even 1.7°C difference at 0.5m is physically normal)

TEST VECTOR 4: Boundary case (PYREXIA)
  Input:
    my_reading = 2310        (23.10°C)
    their_reading = 2620     (26.20°C)
    quantity = TEMPERATURE
    distance_cm = 30         (0.3 meters, very close)
    time_delta_ms = 500      (0.5 seconds)

  Computation:
      epsilon_spatial = 500 × 0.3 = 150
      epsilon_temporal = 83 × 0.5 = 42 (truncated from 41.5)
    sigma_sum = 100
    max_diff = 150 + 42 + 100 = 292
    actual_diff = |2310 - 2620| = 310
    excess = 310 - 292 = 18
    deviation_sigma = 18 / 50 = 0.36σ
    0.36 < 1.0 → CONSISTENT ✓
    (Just barely within noise)

TEST VECTOR 5: CO₂ anomaly
  Input:
    my_reading = 42000       (420.00 ppm)
    their_reading = 89000    (890.00 ppm)
    quantity = CO2
    distance_cm = 300        (3 meters)
    time_delta_ms = 10000    (10 seconds)

  Computation:
      epsilon_spatial = 10000 × 3.0 = 30000
      epsilon_temporal = 83 × 10.0 = 830
    sigma_sum = 2 × 3000 = 6000
    max_diff = 30000 + 830 + 6000 = 36830
    actual_diff = |42000 - 89000| = 47000
    excess = 47000 - 36830 = 10170
    deviation_sigma = 10170 / 3000 = 3.39σ
    → REJECTION(3.39) — triggers INFLAMMATION query
```

ES-8.2 Behavioral Fingerprint Deviation

```
TEST VECTOR 6: Normal device
  Profile:
    value_mean = 23.10
    value_var = 0.25 (σ = 0.50)
    avg_interval_ms = 5000
    interval_var_ms = 250000 (σ = 500)
    correlation with neighbor A = 0.92
    neighbor A latest reading = 23.30

  New reading: 23.40°C at interval 4800ms

  z_value = |23.40 - 23.10| / 0.50 = 0.60
  z_timing = |4800 - 5000| / 500 = 0.40
  expected_from_neighbors = 0.92 × 23.30 / 0.92 = 23.30
  z_corr = |23.40 - 23.30| / 0.50 = 0.20

  total_deviation = (0.4 × 0.60 + 0.2 × 0.40 + 0.3 × 0.20) / 0.9
                  = (0.24 + 0.08 + 0.06) / 0.9
                  = 0.42σ → NORMAL ✓

TEST VECTOR 7: Compromised device (value manipulation)
  Profile: same as above

  New reading: 25.60°C at interval 5100ms
  Neighbor A latest: 23.20°C (environment hasn't changed)

  z_value = |25.60 - 23.10| / 0.50 = 5.00
  z_timing = |5100 - 5000| / 500 = 0.20
  z_corr = |25.60 - 23.20| / 0.50 = 4.80

  total_deviation = (0.4 × 5.00 + 0.2 × 0.20 + 0.3 × 4.80) / 0.9
                  = (2.00 + 0.04 + 1.44) / 0.9
                  = 3.87σ → REJECTION ✓

TEST VECTOR 8: Environmental change (HVAC turned on)
  Profile: same as above

  New reading: 25.60°C at interval 5100ms
  Neighbor A latest: 25.40°C (neighbor ALSO shifted)

  z_value = |25.60 - 23.10| / 0.50 = 5.00
  z_timing = |5100 - 5000| / 500 = 0.20
  z_corr = |25.60 - 25.40| / 0.50 = 0.40

  total_deviation = (0.4 × 5.00 + 0.2 × 0.20 + 0.3 × 0.40) / 0.9
                  = (2.00 + 0.04 + 0.12) / 0.9
                  = 2.40σ → INFLAMMATION ✓

  Inflammation query: neighbors confirm shift → PROFILE UPDATE → NORMAL
  (This correctly distinguishes real environmental change from attack)
```

ES-8.3 Pheromone Field Query

```
TEST VECTOR 9: Healthy field
  Field entries (all TEMPERATURE):
    Device A: reading=2310, weight=204(0.80), age=10s
    Device B: reading=2340, weight=204(0.80), age=15s
    Device C: reading=2290, weight=255(1.00), age=5s
    Device D: reading=2320, weight=77(0.30),  age=8s  (neonatal)

  λ = 0.00116

  Decay factors:
    A: e^(-0.00116 × 10) = 0.9885
    B: e^(-0.00116 × 15) = 0.9827
    C: e^(-0.00116 × 5)  = 0.9942
    D: e^(-0.00116 × 8)  = 0.9908

  Weights (w × decay):
    A: 0.80 × 0.9885 = 0.7908
    B: 0.80 × 0.9827 = 0.7862
    C: 1.00 × 0.9942 = 0.9942
    D: 0.30 × 0.9908 = 0.2972

  Weighted sum = 0.7908×2310 + 0.7862×2340 + 0.9942×2290 + 0.2972×2320
               = 1826.7 + 1839.7 + 2276.7 + 689.5
               = 6632.6
  Weight sum = 0.7908 + 0.7862 + 0.9942 + 0.2972 = 2.8684

  Consensus value = 6632.6 / 2.8684 = 2312 (23.12°C)
  Active traces = 4
  Range = (2290, 2340)

  Note: neonatal device D has LOW influence (0.2972 vs avg ~0.86)
  This is correct: untested devices contribute less to consensus.

TEST VECTOR 10: Decayed field (old traces)
  Same entries but all 30 minutes old:

  Decay at 1800s: e^(-0.00116 × 1800) = e^(-2.088) = 0.1238
  Weights:
    A: 0.80 × 0.1238 = 0.0990
    B: 0.80 × 0.1238 = 0.0990
    C: 1.00 × 0.1238 = 0.1238
    D: 0.30 × 0.1238 = 0.0371

  Total weight = 0.3589 (very low — field is stale)
  Consensus value still computable but LOW CONFIDENCE.

  After 50 minutes (3000s): decay = e^(-3.48) = 0.031
  → Below 0.03 threshold → entries garbage-collected.
```

ES-8.4 NK Audit Drift Detection

```
TEST VECTOR 11: Slow drift attack (0.1σ per day, 100 days)
  Self-signature (frozen):
    value_mean = 23.10
    value_var = 0.25 (σ = 0.50)
    avg_interval_ms = 5000
    interval_var_ms = 250000
    unique_neighbors = 8

  Current profile (after 100 days of 0.1σ/day drift):
    value_mean = 28.10 (drifted +5.00°C = +10.0σ)
    value_var = 0.30 (slightly widened)
    avg_interval_ms = 5200
    interval_var_ms = 280000
    unique_neighbors = 7

  Drift computation:
    mean_drift = |28.10 - 23.10| / 0.50 = 10.00σ
    var_drift = |0.30/0.25 - 1.0| × 2.0 = 0.40σ
    timing_drift = |5200 - 5000| / 500 = 0.40σ
    comm_drift = |7 - 8| / 3.0 = 0.33σ

    total_drift = √(10² + 0.4² + 0.4² + 0.33²) = √(100 + 0.16 + 0.16 +
0.11)
                = √100.43 = 10.02σ

    DRIFT_THRESHOLD = 5.0σ
    10.02 > 5.0 → DRIFT_DETECTED ✓
    drifted_dimensions = 0x01 (value dimension only, > 2σ)

  Detection timing:
    p_audit = 0.012 per epoch (60s)
    Expected detection: 1/0.012 ≈ 83 epochs ≈ 83 minutes
    At 1 audit/day: detection after ~83 days
    But with p_audit per EPOCH (not per day): detection within hours.
```

ES-8.5 Quorum Sensing Transitions

```
TEST VECTOR 12: Startup sequence (SOLO→QUORUM requires EMA ≥ 5,
QUORUM→SWARM ≥ 12)
  t=0:    neighbor_count=0, mode=SOLO
  t=10:   1 neighbor detected, count=0.3×1+0.7×0=0.30, SOLO (< 5)
  t=20:   3 neighbors, count=0.3×3+0.7×0.30=1.11, SOLO
  t=30:   5 neighbors, count=0.3×5+0.7×1.11=2.28, SOLO
  t=40:   7 neighbors, count=0.3×7+0.7×2.28=3.70, SOLO (< 5)
  t=50:   10 neighbors, count=0.3×10+0.7×3.70=5.59, QUORUM (≥ 5) ✓
  t=60:   12 neighbors, count=0.3×12+0.7×5.59=7.51, QUORUM (< 12)
  t=70:   15 neighbors, count=0.3×15+0.7×7.51=9.76, QUORUM (< 12)
  t=80:   18 neighbors, count=0.3×18+0.7×9.76=12.23, SWARM (≥ 12) ✓

  Transition back:
  t=90:   10 neighbors, count=0.3×10+0.7×12.23=11.56, SWARM (≥ 8)
  t=100:  5 neighbors, count=0.3×5+0.7×11.56=9.59, SWARM (≥ 8)
  t=110:  3 neighbors, count=0.3×3+0.7×9.59=7.61, QUORUM (< 8) ✓
  t=120:  1 neighbor, count=0.3×1+0.7×7.61=5.63, QUORUM (≥ 3)
  t=130:  0 neighbors, count=0.3×0+0.7×5.63=3.94, QUORUM (≥ 3)
  t=140:  0 neighbors, count=0.3×0+0.7×3.94=2.76, SOLO (< 3) ✓

  Note: EMA smoothing prevents rapid oscillation. Hysteresis on SOLO→QUORUM
  (threshold 5 vs 3) means you need sustained neighbor presence to enter
QUORUM.
```

ES-8.6 Wire Format Serialization

```
TEST VECTOR 13: TRACE_DEPOSIT serialization
  Input:
    sender_id = 0xA1B2C3D4E5F60718
    timestamp = 1707800000 (2024-02-13 ~07:33 UTC)
    quantity = TEMPERATURE (0x01)
    reading = 2310 (23.10°C)
    confidence = 204 (0.80)
    weight = 204 (0.80)
    seq_num = 42
      prev_hash = 0xABCD1234 (4 bytes)

    Serialized (hex), before signature:
      4B 42                        // magic
// version
// msg_type = TRACE_DEPOSIT
      18 07 F6 E5 D4 C3 B2 A1     // sender_id (little-endian)
      80 6C C7 65                  // timestamp (1707800000 LE)
// quantity = TEMPERATURE
      06 09                        // reading = 2310 (LE: 0x0906)
      CC                           // confidence = 204
      CC                           // weight = 204
      2A                           // seq_num = 42
      34 12 CD AB                  // prev_hash[0..4] (little-endian)
      [64 bytes signature]

    Total: 90 bytes ✓
```

ES-9. Implementation Priorities (MVP Roadmap)

```
PHASE 1 — Minimum Viable Stigmergy (standalone, no blockchain):
  ✦ Wire format serialization/deserialization
  ✦ TRACE_DEPOSIT + PRESENCE_PULSE messages
  ✦ Physical consistency check (ES-4.1)
  ✦ Local DAG (ES-5.1, ES-5.2)
  ✦ Pheromone field (ES-4.5)
  ✦ Quorum sensing state machine (ES-3.2)
  ✦ Innate immunity (Stratum 0) — compiled bounds table
  Target: 2 ESP32s + BLE mesh, temperature sensors
  Validation: Test vectors ES-8.1, ES-8.3, ES-8.5, ES-8.6
```

```
PHASE 2 — Pentastratic Immunity:
  ✦ Behavioral profile construction (ES-4.2)
  ✦ Anomaly detection (ES-4.2 compute_deviation)
  ✦ Node lifecycle state machine (ES-3.1)
  ✦ INFLAMMATION_QUERY/REPLY protocol
  ✦ NK random audit (ES-4.3)
  ✦ Neonatal period management
  Target: 10+ ESP32s, simulated attacks
  Validation: Test vectors ES-8.2, ES-8.4

PHASE 3 — Metabolic State + Blockchain Integration:
  ✦ State decay (half-life computation per state type)
  ✦ Apoptosis protocols (device, state, data)
  ✦ TraceBatch assembly and submission (ES-6.1)
  ✦ PLANKTON settlement (ES-6.3)
  Target: Integration with KRILL blockchain testnet

PHASE 4 — Advanced Mechanisms:
  ✦ Morphogenetic topology (ES-4.6)
  ✦ HGT protocol (ES-3.3)
  ✦ Entropic data valuation (ES-4.4)
  ✦ Vaccine propagation
  ✦ Boundary learning
  Target: 100+ node testbed
```

ES-10. Main Loop and Event Dispatch

Every BIA node runs a single cooperative event loop. There are no threads — all work is done in time-sliced tasks within a single loop iteration. This section defines exactly what happens each iteration and at what intervals.

ES-10.1 Timer Configuration

```
// All intervals in milliseconds
TIMER_SENSOR_READ        = 5000      // read sensor every 5s
(TRACE_INTERVAL_DEFAULT × 1000)
TIMER_PRESENCE_PULSE     = 10000     // broadcast presence every 10s
TIMER_EPOCH_TICK         = 60000     // epoch boundary every 60s
TIMER_DAG_GC             = 300000    // garbage collect DAG every 5 min
TIMER_FLASH_FLUSH        = 60000     // persist critical state every 60s
TIMER_QUORUM_UPDATE      = 10000     // update quorum state every 10s
TIMER_HEARTBEAT          = 30000     // send heartbeat every 30s (metabolic
proof-of-life)
TIMER_PYREXIA_CHECK      = 21600000  // 6h pyrexia timeout check
TIMER_INFLAMMATION_TO    = 300000    // 5 min inflammation timeout
```

ES-10.2 Main Loop (Nano Node)

```
fn main() {
    // —— Phase 0: Hardware Init ——
    init_hardware();                    // GPIO, SPI, I2C, BLE radio, WiFi
    let puf_result = extract_puf();     // see ES-13
    if puf_result.is_err() {
        broadcast(DEATH_CERTIFICATE, PUF_ANOMALY);
        enter_deep_sleep_forever();
    }
    let (privkey, pubkey, device_id) =
derive_identity(puf_result.unwrap());

    // ── Phase 1: Restore State from Flash ──
    let mut state = load_state_from_flash();  // see ES-14
    if state.is_none() {
        // First boot: fresh state
        state = Some(NodeState::new_neonatal(device_id, now()));
    }
    let mut state = state.unwrap();
    let mut quorum =
load_quorum_from_flash().unwrap_or(QuorumState::new());
    let mut dag = load_dag_from_flash().unwrap_or(LocalDAG::new());
    let mut field = PheromoneField::new(LAMBDA_DEFAULT);
    let mut profiles: HashMap<device_id, BehavioralProfile> =
load_profiles_from_flash();

    // ── Phase 2: Transport Init ──
    let transport = init_transport(&device_id, &pubkey);  // see ES-12
    transport.start_advertising();
    transport.start_scanning();

    // ── Phase 3: Clock Sync ──
    if quorum.neighbor_count > 0 {
        request_time_sync(&transport);
    }

    // ── Phase 4: Main Event Loop ──
    let mut last_sensor_read = 0u32;
    let mut last_presence = 0u32;
    let mut last_epoch = 0u32;
    let mut last_dag_gc = 0u32;
    let mut last_flash_flush = 0u32;
    let mut last_quorum_update = 0u32;
    let mut last_heartbeat = 0u32;

    loop {
        let now_ms = millis();  // hardware monotonic clock
        let now_s = now_ms / 1000;

        // ── 4a: Process incoming messages (non-blocking) ──
        while let Some(msg) = transport.recv_nonblocking() {
            if !rate_limiter.check(msg.sender_id, msg.msg_type) {  // ES-15
                continue;  // rate limited, drop
            }
            if !verify_signature(&msg) {
                continue;  // invalid signature, drop
            }
            dispatch_message(&mut state, &mut quorum, &mut dag, &mut field,
                           &mut profiles, &msg, &transport);
        }

        // ── 4b: Sensor read + trace deposit ──
        if now_ms - last_sensor_read ≥ TIMER_SENSOR_READ {
            last_sensor_read = now_ms;

            let reading = read_sensor();
            if reading.is_ok() {
                let reading = reading.unwrap();

                // Innate check on own reading (Stratum 0)
                let self_check = check_absolute_bounds(reading,
sensor_quantity);
```

```
                if self_check == REJECT_INNATE {
                    // Own sensor is broken — enter PYREXIA for self
                    transition(&mut state, DEVIATION(3.0));
                } else {
                    // Build and broadcast trace
                    let trace = build_trace_deposit(
                        &device_id, reading, sensor_quantity,
                        state.immunological_weight, &mut dag, &privkey
                    );
                    transport.broadcast(&trace);
                    deposit_trace(&mut field, &trace);
                    dag_append(&mut dag, &trace);
                }
            }
        }

        // ── 4c: Presence pulse ──
        if now_ms - last_presence >= TIMER_PRESENCE_PULSE {
            last_presence = now_ms;
            let pulse = build_presence_pulse(&device_id, &quorum,
&privkey);
            transport.broadcast(&pulse);
        }

        // ── 4d: Quorum update ──
        if now_ms - last_quorum_update >= TIMER_QUORUM_UPDATE {
            last_quorum_update = now_ms;
            update_quorum(&mut quorum);

            // Adjust lambda based on mode
            field.lambda = match quorum.mode {
                SOLO   => LAMBDA_DEFAULT * 2.0,   // faster decay when
alone
                QUORUM => LAMBDA_DEFAULT,
                SWARM  => LAMBDA_DEFAULT * 0.5,   // slower decay in dense
network
            };
        }

        // ── 4e: Epoch boundary processing ──
        if now_ms - last_epoch >= TIMER_EPOCH_TICK {
            last_epoch = now_ms;
            epoch_tick(&mut state, &mut quorum, &mut dag, &mut field,
                      &mut profiles, &transport, &privkey);
        }

        // ── 4f: DAG garbage collection ──
        if now_ms - last_dag_gc >= TIMER_DAG_GC {
            last_dag_gc = now_ms;
            dag_gc(&mut dag, now_s);
        }

        // ── 4g: Heartbeat ──
        if now_ms - last_heartbeat >= TIMER_HEARTBEAT {
            last_heartbeat = now_ms;
            let hb = build_heartbeat(&device_id, now_s, &dag, &quorum,
&privkey
);
            transport.broadcast(&hb);
            state.last_heartbeat = now_s;
        }

        // ── 4h: Persist to flash ──
        if now_ms - last_flash_flush >= TIMER_FLASH_FLUSH {
            last_flash_flush = now_ms;
            flush_to_flash(&state, &quorum, &dag, &profiles);  // ES-14
        }

        // ── 4i: State timeout checks ──
        check_state_timeouts(&mut state, now_s, &transport, &privkey);

        // ── 4j: Sleep until next event ──
        // Calculate next wake time
        let next_event = [
            last_sensor_read + TIMER_SENSOR_READ,
            last_presence + TIMER_PRESENCE_PULSE,
            last_epoch + TIMER_EPOCH_TICK,
            last_quorum_update + TIMER_QUORUM_UPDATE,
        ].iter().min().unwrap() - now_ms;

        if next_event > 10 && transport.recv_queue_empty() {
            light_sleep(next_event.min(100));  // max 100ms sleep, wake on
radio interrupt
        }
    }
}
```

ES-10.3 Message Dispatch

```
fn dispatch_message(
    state: &mut NodeState,
    quorum: &mut QuorumState,
    dag: &mut LocalDAG,
    field: &mut PheromoneField,
    profiles: &mut HashMap<device_id, BehavioralProfile>,
    msg: &Message,
    transport: &Transport,
) {
    match msg.msg_type {
        TRACE_DEPOSIT => {
            // 1. Record in DAG
            dag_append(dag, &msg.as_trace());

            // 2. Deposit in pheromone field
            deposit_trace(field, &msg.as_trace());

            // 3. Physical consistency check against own latest reading
            if let Some(my_latest) = get_my_latest_reading(dag) {
                let distance = estimate_distance(msg.sender_id, quorum);
// ES-11
                let result = check_physical_consistency(
                    my_latest.reading, msg.as_trace().reading,
                    msg.as_trace().quantity_type,
                    distance, (now() - my_latest.timestamp) * 1000
                );
                handle_consistency_result(state, result, msg.sender_id,
transport);
            }

            // 4. Behavioral profile update (if not in neonatal)
            if state.profile_frozen {
                let profile = profiles.entry(msg.sender_id)
                    .or_insert(BehavioralProfile::new(msg.sender_id));
                let deviation = compute_deviation(profile, &msg.as_trace(),
                    &get_recent_readings(dag, msg.sender_id));

                // Feed deviation into state machine
                transition(state, DEVIATION(deviation));

                // Update profile (only if not anomalous)
                if deviation < SIGMA_INFLAMMATION as f32 / 10.0 {
                    update_profile(profile, &msg.as_trace());
                }
            }
        }

        PRESENCE_PULSE => {
            // Update neighbor table
            let entry = &mut quorum.neighbor_table;
            upsert_neighbor(entry, msg.sender_id, msg.as_pulse().mode,
                          transport.last_rssi(), now());
        }

        INFLAMMATION_QUERY => {
            // Am I a neighbor of the suspect?
            let query = msg.as_inflammation_query();
            if let Some(my_reading) = get_my_latest_for_quantity(dag,
query.quantity_type) {
                let verdict = evaluate_inflammation_locally(
                    my_reading, query.suspect_reading, query.quantity_type
                );
                let reply = build_inflammation_reply(
                    &device_id, query.suspect_id, verdict,
                    my_reading, &privkey
                );
                transport.send_to(msg.sender_id, &reply);
            }
        }

        INFLAMMATION_REPLY => {
            transition(state, REPLY(msg.as_inflammation_reply()));
        }

        VACCINE => {
            let vaccine = msg.as_vaccine();
            if vaccine.hops_from_origin < 10 {  // TTL check
                // Apply vaccine to local immune state
                apply_vaccine(profiles, &vaccine);
                // Relay with incremented hop count
                let relayed = vaccine.with_incremented_hops();
                transport.broadcast(&relayed);
            }
        }

        DEATH_CERTIFICATE => {
            let cert = msg.as_death_cert();
            // Remove device from neighbor table
            remove_neighbor(&mut quorum.neighbor_table, cert.sender_id);
            // Mark all traces from this device as untrusted
            invalidate_traces(dag, field, cert.sender_id);
            // Remove profile
            profiles.remove(&cert.sender_id);
        }

        HEARTBEAT => {
            upsert_neighbor(&mut quorum.neighbor_table, msg.sender_id,
                          msg.as_heartbeat().mode, transport.last_rssi(),
now());
        }

        PROFILE_SYNC => {
            let sync = msg.as_profile_sync();
            if let Some(local_profile) = profiles.get(&sync.subject_id) {
                if sync.profile_version > local_profile.profile_version {
                    // Merge: remote is newer
                    let merged = merge_profiles(local_profile,
&sync.to_profile());
                    profiles.insert(sync.subject_id, merged);
                }
            } else {
                // New device we haven't profiled yet — adopt
                profiles.insert(sync.subject_id, sync.to_profile());
            }
        }

        _ => {} // unknown message type, ignore
    }
}
```

ES-10.4 Epoch Tick

```
fn epoch_tick(
    state: &mut NodeState,
    quorum: &mut QuorumState,
    dag: &mut LocalDAG,
    field: &mut PheromoneField,
    profiles: &mut HashMap<device_id, BehavioralProfile>,
    transport: &Transport,
    privkey: &[u8; 32],
) {
    // 1. Update communication pattern in profiles
    for (did, profile) in profiles.iter_mut() {
        let msg_count = count_messages_this_epoch(dag, *did);
        // EMA update
        profile.msg_rate_per_epoch = 0.9 * profile.msg_rate_per_epoch
                                    + 0.1 * msg_count as f16;
    }

    // 2. NK Random Audit (Stratum 4)
    let rng_byte = hardware_rng_byte();
    for (did, profile) in profiles.iter() {
        if let Some(self_sig) = get_self_signature(did) {
            if let Some(result) = maybe_run_nk_audit(*did, self_sig,
profile, rng_byte) {
                match result {
                    NkAuditResult::DRIFT_DETECTED { device_id, drift_sigma,
dimensions } =>
{
                        // Broadcast alert
                        let alert = build_consistency_alert(
                            device_id, 0, 0, 0, REJECTION as u8, privkey
                        );
                        transport.broadcast(&alert);
                        // Local action
                        transition(state, DEVIATION(drift_sigma));
                    }
                    NkAuditResult::PASS => {} // all good
                }
            }
        }
    }

    // 3. Neonatal timer check
    if state.state == NEONATAL {
        if quorum.mode != SOLO {
            // Only count time when we have neighbors observing us
            let elapsed = now() - state.neonatal_start;
            if elapsed >= NEONATAL_DURATION {
                transition(state, TIMER_EXPIRED);
            }
        }
    }

    // 4. Batch assembly for blockchain (if Full node connection available)
    if quorum.mode == SWARM || quorum.mode == QUORUM {
        if let Some(batch) = assemble_trace_batch(dag, field, profiles) {
            submit_to_full_node(batch, transport);
        }
    }

    // 5. Profile sync with neighbors (every 10th epoch = 10 min)
    static mut epoch_counter: u32 = 0;
    epoch_counter += 1;
    if epoch_counter % 10 == 0 {
        for (did, profile) in profiles.iter() {
            let sync_msg = build_profile_sync(did, profile, privkey);
            transport.broadcast(&sync_msg);
        }
    }
}
```

ES-10.5 State Timeout Check

```
fn check_state_timeouts(
    state: &mut NodeState,
    now_s: u32,
    transport: &Transport,
    privkey: &[u8; 32],
) {
    match state.state {
        PYREXIA => {
            // Auto-escalate after 6 hours
            if now_s - state.entered_at > 21600 {
                transition(state, TIMER_EXPIRED);
            }
        }
        INFLAMMATION => {
            // Decide with partial replies after 5 minutes
            if now_s - state.entered_at > 300 {
                evaluate_inflammation();
            }
        }
        REJECTION => {
            // Move to QUARANTINE after 72 hours
            if now_s - state.entered_at > 259200 {
                transition(state, TIMER_EXPIRED);
            }
        }
        QUARANTINE => {
            // Attempt auto-remediation every 24 hours
            if now_s - state.entered_at > 86400 *
(state.remediation_attempts as u32 + 1) {
                let result = attempt_remediation(state);
                transition(state, result);
            }
        }
        _ => {
            // Check heartbeat timeout (any state)
            if now_s - state.last_heartbeat > T_HALF_STRUCTURAL {
                state.state = APOPTOTIC;
                let cert = build_death_certificate(ENERGY_DEPLETED,
privkey);
                transport.broadcast(&cert);
            }
        }
    }
}
```

ES-10.6 Dust Node Main Loop (Simplified)

```
fn main_dust() {
    // Dust nodes have NO behavioral profiling, NO NK audit, NO
inflammation queries.
    // They only: read sensor, deposit trace, listen for alerts.

    init_hardware_lpwan();
    let (privkey, pubkey, device_id) = derive_identity_dust();
    let transport = init_lpwan_transport();

    loop {
        // 1. Read sensor
        let reading = read_sensor();

        // 2. Innate check (Stratum 0 only)
        if check_absolute_bounds(reading, sensor_quantity) == REJECT_INNATE
{
            enter_deep_sleep(60);  // sensor broken, retry in 60s
            continue;
        }

        // 3. Build compressed trace (8 bytes for LPWAN)
        let trace = build_compressed_trace(
            device_id, reading, &mut seq_counter
        );
        transport.send(&trace);

        // 4. Check for incoming alerts/vaccines (rare)
        if let Some(msg) = transport.recv_with_timeout(500) {
            match msg.msg_type {
                VACCINE => apply_vaccine_dust(&msg);
                REJECTION_NOTICE => {
                    if msg.suspect_id == device_id {
                        enter_quarantine_dust();
                    }
                }
                _ => {}
            }
        }

        // 5. Deep sleep until next reading
        enter_deep_sleep(TRACE_INTERVAL_DEFAULT * 1000);
    }
}
```

ES-11. Distance Estimation

ES-11.1 Problem Statement

`check_physical_consistency()` requires `distance_cm` between two devices. In a wireless mesh network, exact distance is rarely known. This section specifies how distance is estimated.

ES-11.2 Estimation Methods (ordered by preference)

```
enum DistanceMethod : u8 {
    STATIC_CONFIG   = 0,    // operator-provided during deployment
    RSSI_MODEL      = 1,    // radio signal strength model
    ROUND_TRIP_TIME = 2,    // BLE connection interval timing
    UNKNOWN         = 3,    // fallback: assume MAX_REASONABLE_DISTANCE
}
```

ES-11.3 Method 0: Static Configuration (preferred for fixed deployments)

```
DEPLOYMENT MANIFEST (loaded into flash at provisioning time):

struct DeploymentEntry {
    device_id:      device_id,
    position_cm:    [i32; 3],       // (x, y, z) in centimeters, relative
to deployment origin
    floor:          u8,             // floor number (for multi-story
buildings)
    zone:           u16,            // logical zone identifier
}

fn static_distance(a: device_id, b: device_id, manifest: &
[DeploymentEntry]) → Option&l
t;u32> {
    let pos_a = manifest.iter().find(|e| e.device_id == a)?.position_cm;
    let pos_b = manifest.iter().find(|e| e.device_id == b)?.position_cm;

    let dx = (pos_a[0] - pos_b[0]) as f32;
    let dy = (pos_a[1] - pos_b[1]) as f32;
    let dz = (pos_a[2] - pos_b[2]) as f32;

    Some((dx*dx + dy*dy + dz*dz).sqrt() as u32)  // distance in cm
}

STORAGE COST: 21 bytes per device. For 100-device deployment = 2.1 KB.
Fits in Nano flash easily. Dust nodes store only their own position.
```

ES-11.4 Method 1: RSSI-Based Model (for ad-hoc deployments)

```
// Log-distance path loss model:
//   RSSI = RSSI_ref - 10 × n × log10(d / d_ref)
// Where:
//   RSSI_ref = RSSI at reference distance d_ref (1 meter)
//   n = path loss exponent (environment-dependent)
//   d = estimated distance
// Calibration constants (stored in flash, adjustable per-deployment):
RSSI_REF_1M             = -59       // dBm at 1 meter (BLE typical, measured during calibration
)
PATH_LOSS_EXPONENT_INDOOR = 2.7     // typical indoor (2.0 = free space, 3.5 = heavy walls)
PATH_LOSS_EXPONENT_OUTDOOR = 2.2    // typical outdoor
RSSI_FLOOR              = -95       // below this, signal is noise

fn rssi_to_distance_cm(rssi: i8, environment: Environment) → u32 {
    if rssi ≥ RSSI_REF_1M {
        return 100;  // closer than 1m, clamp to 1m (RSSI model unreliable close-up)
    }
    if rssi ≤ RSSI_FLOOR {
        return MAX_REASONABLE_DISTANCE;  // too weak to estimate
    }

    let n = match environment {
        INDOOR  ⇒ PATH_LOSS_EXPONENT_INDOOR,
        OUTDOOR ⇒ PATH_LOSS_EXPONENT_OUTDOOR,
    };

    // d = d_ref × 10^((RSSI_ref - RSSI) / (10 × n))
    let exponent = (RSSI_REF_1M as f32 - rssi as f32) / (10.0 * n);
    let distance_m = 10.0_f32.powf(exponent);

    (distance_m * 100.0) as u32  // convert to centimeters
}

// ACCURACY: ±50% indoors (walls, reflections cause multipath fading)
// MITIGATION: Use median of last 8 RSSI readings to smooth fluctuations

struct RSSITracker {
    history:    RingBuffer<i8, 8>,
}

fn smoothed_rssi(tracker: &RSSITracker) → i8 {
    let mut sorted = tracker.history.as_slice().to_vec();
    sorted.sort();
    sorted[sorted.len() / 2]  // median (more robust than mean against outliers)
}

MAX_REASONABLE_DISTANCE = 3000      // 30 meters (cm). Used when estimation fails.
```

ES-11.5 Method 2: Round-Trip Time (BLE only, high accuracy)

```
// BLE connection events have precise timing. Round-trip can estimate distance.
// Speed of light: ~3.3 ns/meter. BLE clock resolution: ~1 μs = ~300m.
// NOT practical for sub-meter accuracy over BLE.
// RESERVED for future UWB (Ultra-Wideband) support where RTT gives ±10cm accuracy.
// For now: UNUSED. Placeholder for UWB-enabled hardware.
```

ES-11.6 Composite Distance Estimation

```
fn estimate_distance(
    their_id: device_id,
    quorum: &QuorumState,
) -> u32 {
    // Priority 1: Static config
    if let Some(d) = static_distance(my_id(), their_id,
&DEPLOYMENT_MANIFEST) {
        return d;
    }

    // Priority 2: RSSI model
    if let Some(neighbor) = find_neighbor(quorum, their_id) {
        if neighbor.rssi > RSSI_FLOOR {
            let tracker = get_rssi_tracker(their_id);
            return rssi_to_distance_cm(smoothed_rssi(tracker),
CURRENT_ENVIRONMENT);
        }
    }

    // Priority 3: Unknown — use conservative default
    // Using MAX_REASONABLE_DISTANCE (30m) makes consistency checks
LENIENT.
    // This is intentional: we'd rather miss an anomaly than false-alarm
when
    // distance is unknown. The behavioral fingerprint (Stratum 1) will
catch
    // attacks that spatial consistency misses.
    MAX_REASONABLE_DISTANCE
}

// IMPORTANT: distance estimation quality affects ONLY spatial gradient
checks.
// Temporal gradient checks and behavioral profiling are distance-
independent.
// Even with ±50% RSSI error, the system remains secure because:
//   1. Temporal checks catch rapid changes regardless of distance
//   2. Behavioral fingerprint catches sustained deviations
//   3. Neighbor correlation catches isolated anomalies
//   4. NK audit catches slow drift
// Distance is a "nice to have" for spatial gradient — not a single point
of failure.
```

ES-12. Transport Layer

ES-12.1 Transport Abstraction

```
// All BIA code talks to a Transport trait. Concrete implementations are
per-hardware.
// This allows the same BIA logic to run on BLE mesh, WiFi, LoRa, or
simulation.

trait Transport {
    fn broadcast(&self, msg: &[u8]);                    // send to all
neighbors
    fn send_to(&self, target: device_id, msg: &[u8]);  // send to specific
device
    fn recv_nonblocking(&self) -> Option<Message>;      // poll for
incoming messa
ge
    fn recv_with_timeout(&self, ms: u32) -> Option<Message>;
    fn last_rssi(&self) → i8;                          // RSSI of last
received message
    fn start_advertising(&self);
    fn start_scanning(&self);
    fn recv_queue_empty(&self) → bool;
}
```

ES-12.2 BLE Mesh Implementation (Primary for Nano+Dust)

```
BLUETOOTH CONFIGURATION:
  Specification:          Bluetooth Mesh (Mesh Profile 1.0.1)
  Model:                  Vendor-specific model (Company ID: to be
assigned)
  Service UUID:           0xKB01 (placeholder, real UUID = 128-bit)

  Advertising:
    Type:                 Connectable + Scannable Undirected (ADV_IND)
    Interval:             100ms (during active phase), 1000ms (during sleep
phase)
    TX Power:             0 dBm (default), adjustable -20 to +8 dBm per
morphogenetic signal

  Scanning:
    Window:               30ms
    Interval:             60ms (50% duty cycle during active)
    Active scan:          Yes (to get scan response data)

  Connection:
    Not used for data exchange (connectionless mesh)
    BLE Mesh uses advertising-based bearer

  MTU:
    Advertising PDU payload: 31 bytes (legacy) / 255 bytes (extended
advertising)
    For extended advertising (BT 5.0+): 255 bytes → all messages fit in 1
PDU
    For legacy advertising: messages > 31 bytes need SEGMENTATION

SEGMENTATION (legacy BLE only):
    Header: [segment_index: u8, total_segments: u8, msg_id: u16]  = 4 bytes
overhead
    Payload per segment: 31 - 4 = 27 bytes
    TRACE_DEPOSIT (90B): ceil(90/27) = 4 segments → 4 × advertising events
    PRESENCE_PULSE (78B): ceil(78/27) = 3 segments

    Reassembly:
      Receiver maintains reassembly buffer per (sender_id, msg_id) pair.
      Timeout: 500ms. If not all segments received → discard.
      Max concurrent reassemblies: 8 (per neighbor).

    NOTE: With BT 5.0 extended advertising (255B), segmentation is
unnecessary.
    Phase 1 MVP SHOULD target BT 5.0 devices to avoid segmentation
complexity.

RELAY:
    BLE Mesh relay is ENABLED on Nano nodes.
    Nano nodes relay messages for Dust nodes that can't reach each other
directly.
    Relay TTL: 3 hops (adjustable via governance).
    Relay cache: last 32 message hashes to prevent relay loops.

NETWORK KEY:
    All BIA devices in a deployment share a Network Key (NetKey).
    NetKey is provisioned during device enrollment (ES-13).
    NetKey provides network-layer encryption (AES-CCM).
    Application-layer integrity: Ed25519 signatures (per ES-2).

    NOTE: NetKey prevents non-BIA devices from injecting messages into the
mesh,
    but does NOT prevent compromised BIA devices from sending invalid data.
    That's what the immune system is for.
```

ES-12.3 WiFi Implementation (Nano-to-Full, Nano-to-Nano long range)

```
WIFI CONFIGURATION:
  Protocol:                ESP-NOW (connectionless, peer-to-peer over WiFi
PHY)
  Channel:                 Auto-negotiated, follows BLE mesh channel
  Encryption:              CCMP (WPA2-equivalent, built into ESP-NOW)
  Max payload:             250 bytes (ESP-NOW limit) → all messages fit

  Peer management:
    Max ESP-NOW peers: 20 (ESP32 hardware limit)
    Peers = Nano nodes in range (discovered via BLE mesh)
    Fallback: standard WiFi + UDP multicast if ESP-NOW peer limit reached

  UDP Multicast (fallback):
    Address:               239.75.66.73 (0xEF.0x4B.0x42.0x49 = "KBIA" in
ASCII offsets)
    Port:                  47201
    TTL:                   1 (link-local only)
    No ACK, no retransmit (same as BLE mesh — at-most-once delivery)

NANO → FULL NODE (batch submission):
    Protocol:              TCP over WiFi
    Port:                  47202
    TLS:                   Required (TLS 1.3, server authenticates with
Ed25519 cert)
    Message format:        Length-prefixed (4-byte LE length + TraceBatch
payload)
    Keepalive:             30s TCP keepalive
    Reconnect:             Exponential backoff 1s → 2s → 4s → ... → 300s max
```

ES-12.4 LoRa/LPWAN Implementation (Dust nodes)

```
LORA CONFIGURATION:
  Frequency:               Region-dependent (EU868: 868 MHz, US915: 915 MHz)
  Spreading Factor:        SF12 (max range, min data rate)
  Bandwidth:               125 kHz
  Coding Rate:             4/5
  Max payload:             51 bytes (SF12, EU868) → compressed trace (8B)
fits easily
  Duty cycle:              1% (EU regulation) → max 1 message per ~7 seconds
at SF12

  MESSAGE FLOW (Dust → Gateway → Nano):
    1. Dust node wakes from deep sleep
    2. Reads sensor
    3. Builds compressed trace (8 bytes, ES-2.3)
    4. Transmits via LoRa
    5. Gateway (co-located with Nano or standalone) receives
    6. Gateway wraps compressed trace into full TRACE_DEPOSIT:
       - Expands sender_id_short (2B) to full device_id (8B) from
registration table
       - Adds gateway-provided timestamp (if Dust has no RTC)
       - Signs with GATEWAY'S key (not Dust's — Dust has no Ed25519)
       - Sets confidence field to reflect gateway vouching (see ES-16)
    7. Gateway injects into BLE/WiFi mesh as normal TRACE_DEPOSIT
    8. Nano processes like any other trace (but with gateway trust
discount)

  DOWNLINK (rare):
    - Gateway → Dust: VACCINE messages (condensed to 8 bytes)
    - Gateway → Dust: REJECTION_NOTICE (4 bytes: header + device_id_short)
    - Max 1 downlink per 10 uplinks (duty cycle budget)
```

ES-12.5 Multicast vs Unicast Rules

```
MESSAGE DELIVERY MODE:
  BROADCAST (BLE mesh advertising, ESP-NOW broadcast, UDP multicast):
    TRACE_DEPOSIT          — all neighbors need it for consensus
    PRESENCE_PULSE         — discovery protocol
    HEARTBEAT              — proof of life
    MODE_ANNOUNCE          — quorum state advertisement
    VACCINE                — epidemic propagation
    DEATH_CERTIFICATE      — network-wide invalidation
    CONSISTENCY_ALERT      — network awareness

  UNICAST (BLE mesh unicast, ESP-NOW peer, TCP):
    INFLAMMATION_QUERY     — sent to specific neighbors of suspect
    INFLAMMATION_REPLY     — reply to specific querier
    NK_AUDIT_REQUEST       — challenge to specific device
    NK_AUDIT_RESPONSE      — response to challenger
    PROFILE_SYNC           — targeted exchange between Nano nodes
    TIME_SYNC_REQUEST      — request to nearest clock source
    TIME_SYNC_RESPONSE     — response to requester
    MODULE_* (HGT)         — targeted module exchange
    TraceBatch             — Nano → specific Full node (TCP)
```

ES-13. Cryptographic Initialization and Device Enrollment

ES-13.1 PUF-Based Key Derivation

```
BOOT SEQUENCE (cryptographic):

1. SRAM PUF EXTRACTION:
   - Power on ESP32
   - Read uninitialized SRAM (first 4KB, before any write)
   - SRAM cells settle to device-specific pattern based on manufacturing
variation
   - Raw PUF response: 4096 bytes = 32768 bits

2. FUZZY EXTRACTION (BCH-based):
   - Fuzzy extractor parameters:
       n = 32768 (PUF response length in bits)
       k = 256 (extracted key length in bits)
       t = 5120 (error correction capacity = 15.6% BER tolerance)

   - ENROLLMENT (first boot only):
       a. Read raw PUF response R
       b. Generate random 256-bit seed S
       c. Encode S using BCH(32768, 256, 5120) → codeword C
       d. Compute helper data: W = R ⊕ C
       e. Store W in flash (public — knowing W without R reveals nothing
about S)
       f. Derive key pair from S

   - RECONSTRUCTION (every subsequent boot):
       a. Read raw PUF response R' (noisy version of R)
       b. Load helper data W from flash
       c. Compute C' = R' ⊕ W  (noisy codeword)
       d. BCH decode C' → C (corrects up to t=5120 bit errors)
       e. Recover S = BCH_message(C)
       f. Derive same key pair from S

   - BIT ERROR RATE MONITORING:
       ber = hamming_distance(R', R_enrolled) / 32768
       Store ber_history[8] in flash (last 8 boots).
       If ber > 0.12 (approaching 0.156 limit): set PUF_DEGRADATION_WARNING
flag.
       If ber > 0.156: BCH decode fails → PUF_FAIL → APOPTOTIC.

3. KEY DERIVATION (from seed S):
   - signing_privkey = HKDF-SHA256(S, salt="krill-bia-sign", info="ed25519-
v1", len=32)
   - signing_pubkey  = Ed25519_pubkey(signing_privkey)
   - device_id       = BLAKE2s(signing_pubkey)[0..8]  // truncated to 8
bytes
   - network_auth_key = HKDF-SHA256(S, salt="krill-bia-net", info="mesh-
auth-v1", len=16)
     // Used for BLE mesh network authentication during provisioning

STORAGE:
   - Flash (persistent):
       helper_data_W:      4096 bytes (PUF helper data)
       ber_history:        8 bytes
       enrolled_flag:      1 byte (0x00 = first boot, 0xFF = enrolled)
   - RAM (derived each boot):
       signing_privkey:    32 bytes (NEVER written to flash)
       signing_pubkey:     32 bytes
       device_id:          8 bytes
       network_auth_key:   16 bytes
```

ES-13.2 Device Enrollment Protocol

```
ENROLLMENT FLOW (new device joining network):

1. PROVISIONER ROLE:
   - Any Nano node in NORMAL state with weight ≥ 0.8 can act as
provisioner.
   - Provisioner must be connected to ≥ 3 other Nano nodes (QUORUM or
SWARM).
   - In Phase 1 MVP: enrollment requires physical button press on
provisioner
     (prevents remote enrollment of rogue devices).

2. PROTOCOL:
   NEW DEVICE (Neonatal)              PROVISIONER (Nano)
   ────────────────                  ─────────────────
   Boots, extracts PUF, derives keys
   Enters NEONATAL state
   Broadcasts ENROLLMENT_REQUEST:
     { device_id, pubkey, capabilities,
       sensor_types[], hw_version }
                                      ──→ Receives request
                                      ──→ Operator presses ENROLL button
                                          (or: auto-accept if deployment
manifest
                                           includes this device_id)
                                      ──→ Validates: device_id not already
registered
                                      ──→ Generates ENROLLMENT_RESPONSE:
                                          { provisioner_id,
network_key_encrypted,
                                            time_sync, deployment_zone,
                                            initial_neighbor_list[] }
                                      ←──
   Receives response
   Decrypts network_key using
     Diffie-Hellman(my_privkey, provisioner_pubkey)
   Sets system clock from time_sync
   Stores network_key in RAM
   Enters BLE mesh with network_key
   Begins NEONATAL period

3. ENROLLMENT MESSAGE (new wire format):

   MessageType ENROLLMENT_REQUEST  = 0x60
   MessageType ENROLLMENT_RESPONSE = 0x61

   ENROLLMENT_REQUEST:
   Offset  Size  Field               Description
   ──────────────────────────────────────────────────
12    header              msg_type = 0x60
32    pubkey              Ed25519 public key
1     capability_flags    bit 0=BLE, 1=WiFi, 2=LoRa, 3=UWB
1     sensor_count        number of sensors
N     sensor_types        N × QuantityType (1 byte each)
   46+N    2     hw_version          hardware revision
   48+N    64    signature           Ed25519(self-signed, proves key
ownership)
   ──────────────────────────────────────────────────

   ENROLLMENT_RESPONSE:
   Offset  Size  Field               Description
   ──────────────────────────────────────────────────
12    header              msg_type = 0x61
8     target_device_id    who this response is for
32    provisioner_pubkey  for DH key exchange
32    network_key_enc     NetKey encrypted with DH shared secret
4     time_sync           current timestamp
2     deployment_zone     zone assignment
64    signature           Ed25519(provisioner key)
   ──────────────────────────────────────────────────

4. KEY ROTATION:
   - Normal operation: keys are NEVER rotated (PUF-derived, stable).
   - If PUF degrades: device issues DEATH_CERTIFICATE with old key,
     undergoes physical re-enrollment with new PUF extraction → new
identity.
   - If network key is compromised: provisioner broadcasts KEY_ROTATION
```

```
message
     with new NetKey encrypted to each device's pubkey individually.
     Deadline: 24 hours. Devices that don't rotate are excluded from mesh.
```

ES-13.3 Dust Node Enrollment (Simplified)

```
Dust nodes CANNOT run Ed25519 (insufficient compute/RAM on nRF52832 class).

DUST ENROLLMENT:
  1. Dust node has factory-provisioned 128-bit symmetric key (AES-128).
  2. During deployment, operator registers (dust_id, symmetric_key) in
gateway.
  3. Gateway maintains lookup table: dust_id_short (2B) → { device_id (8B),
aes_key (16B) }.
  4. Dust traces are authenticated via CRC-8 (ES-2.3) — NOT
cryptographically signed.
  5. Gateway verifies CRC, wraps in full TRACE_DEPOSIT, signs with
gateway's Ed25519 key.
  6. Trust chain: Dust → symmetric auth → Gateway → Ed25519 → Network.

LIMITATION: If gateway is compromised, all Dust traces through that gateway
are suspect.
  Mitigation: Multiple gateways per zone, cross-validation between gateway-
submitted traces.

STORAGE (Gateway):
  Per Dust device: 2 + 8 + 16 = 26 bytes.
  For 1000 Dust devices: 26 KB. Fits easily in Nano flash.
```

ES-14. Flash Layout and Persistence

ES-14.1 Flash Partitioning (ESP32 — 4MB Flash)

```
PARTITION TABLE:
  Offset      Size        Name            Content
  ────────────────────────────────────────────────────────────
  0x000000    0x008000    nvs             Non-Volatile Storage (ESP-IDF
NVS)
  0x008000    0x001000    phy_init        WiFi/BLE calibration data
  0x009000    0x001000    otadata         OTA partition tracking
  0x010000    0x180000    app0            Firmware image A (1.5 MB)
  0x190000    0x180000    app1            Firmware image B (OTA, 1.5 MB)
  0x310000    0x001000    krill_meta      BIA metadata (enrolled flag, PUF
helper, etc.)
  0x311000    0x040000    krill_dag       DAG storage (256 KB)
  0x351000    0x020000    krill_profiles  Behavioral profiles (128 KB)
  0x371000    0x004000    krill_state     Node state + quorum state (16 KB)
  0x375000    0x001000    krill_deploy    Deployment manifest (4 KB)
  0x376000    0x08A000    krill_reserve   Reserved for future (552 KB)
  ────────────────────────────────────────────────────────────
  Total:      0x400000    (4 MB)
```

ES-14.2 krill_meta Partition Layout

```
Offset  Size    Field                   Description
──────────────────────────────────────────────────────
0x000   1       enrolled_flag           0x00=first boot, 0xFF=enrolled
0x001   1       protocol_version        PROTOCOL_VERSION at time of
enrollment
0x002   2       hw_version              hardware revision
0x004   4096    puf_helper_data_W       fuzzy extractor helper data
0x1004  8       ber_history             last 8 boot BER values (u8 each,
×255 = percentage)
0x100C  32      pubkey_cache            cached public key (for
verification, not security)
0x102C  8       device_id_cache         cached device_id
0x1034  4       enrollment_timestamp    when device was enrolled
0x1038  2       deployment_zone         assigned zone
0x103A  16      network_key_enc         encrypted NetKey (decrypted at boot
via PUF)
0x104A  4       boot_count              total boot counter
0x104E  4       last_boot_timestamp     when device last booted
0x1052  1       puf_warning_flag        0x00=OK, 0x01=BER>12% warning
0x1053  1       reserved
──────────────────────────────────────────────────────
Total: ~4180 bytes (fits in 4KB partition with margin)
```

ES-14.3 State Persistence Strategy

```
WHAT IS PERSISTED (and when):

1. CRITICAL STATE (persisted every TIMER_FLASH_FLUSH = 60s):
   - NodeState.state, entered_at, neonatal_start
   - NodeState.immunological_weight
   - NodeState.last_heartbeat
   - NodeState.remediation_attempts
   - QuorumState.mode, last_mode_change

2. DAG (persisted incrementally):
   - Each new DAG entry is appended to krill_dag partition
   - DAG GC rewrites the partition (compaction)
   - Format: sequential DAGEntry structs (58 bytes each)
   - Header: [entry_count: u16, tip_hash: hash20, checksum: u32]
   - Max entries: 256KB / 58B = ~4413 entries (capped at MAX_DAG_ENTRIES =
4096)

3. PROFILES (persisted every 10th epoch = 10 min):
   - Serialized HashMap<device_id, BehavioralProfile>
   - Format: [profile_count: u16, then sequential (device_id,
BehavioralProfile) pairs]
   - Max profiles: 128KB / 500B = ~256 profiles

4. NEIGHBOR TABLE (NOT persisted — rebuilt from presence pulses):
   - Neighbors are transient; rebuilding from scratch takes ~30 seconds
   - Not worth flash wear for data that changes every 10 seconds

FLASH WEAR MANAGEMENT:
   - ESP32 flash endurance: ~100,000 write cycles per sector (4KB)
   - At 1 write/minute to krill_state: 100,000 min = ~69 days per sector
   - MITIGATION: Use NVS library (wear-leveling built-in) for state
partition
   - DAG partition: wear-leveled via rotating write pointer + compaction
   - Profiles: written every 10 min → 100,000 × 10 min = ~694 days = ~1.9
years
```

```
   - For 5-year lifetime: implement sector rotation (3 sectors minimum)

BOOT SEQUENCE (state restoration):
   1. Read krill_meta: verify enrolled_flag, increment boot_count
   2. Extract PUF, derive keys
   3. Read krill_state: restore NodeState and QuorumState
   4. Read krill_dag: restore LocalDAG, rebuild PheromoneField from DAG
entries
   5. Read krill_profiles: restore behavioral profiles
   6. If any partition has invalid checksum: start with empty state for
that component
      (DAG and profiles will rebuild naturally; state defaults to NEONATAL)
   7. Resume main loop
```

ES-14.4 Dust Node Flash Layout (nRF52 — 256KB Flash)

```
PARTITION TABLE (nRF52832, 256KB flash):
  Offset      Size        Name            Content
  ────────────────────────────────────────────────────────────
  0x000000    0x01C000    softdevice      BLE softdevice (112 KB)
  0x01C000    0x01C000    application     Firmware (112 KB)
  0x038000    0x001000    dust_meta       Enrolled flag, symmetric key,
device_id (4 KB)
  0x039000    0x003800    dust_dag        Micro-DAG (14 KB, ~240 entries)
  0x03C800    0x001000    dust_state      Minimal state (4 KB)
  0x03D800    0x002800    dust_reserve    Reserved (10 KB)
  ────────────────────────────────────────────────────────────
  Total:      0x040000    (256 KB)

Active RAM usage: ~2 KB (current trace, working buffers, radio stack)
```

ES-15. Rate Limiting and DoS Protection

ES-15.1 Rate Limiter Design

```
// Per-device, per-message-type token bucket rate limiter.
// Prevents flood attacks from compromised devices.

struct RateLimiter {
    buckets: HashMap<(device_id, MessageType), TokenBucket>,
}

struct TokenBucket {
    tokens:         u16,        // current token count
    max_tokens:     u16,        // bucket capacity
    refill_rate:    u16,        // tokens per epoch (60s)
    last_refill:    timestamp,
}

// Rate limits per message type (tokens per epoch):
RATE_LIMITS = {
    TRACE_DEPOSIT:      15,     // max 15 traces/min (normal: 12 at 5s
interval)
    PRESENCE_PULSE:     8,      // max 8 pulses/min (normal: 6 at 10s
interval)
    CONSISTENCY_ALERT:  5,      // max 5 alerts/min
    INFLAMMATION_QUERY: 3,      // max 3 queries/min (expensive operation)
    INFLAMMATION_REPLY: 10,     // max 10 replies/min (responding to
multiple queries)
    VACCINE:            2,      // max 2 vaccines/min (epidemic propagation
is slow)
    HEARTBEAT:          4,      // max 4 heartbeats/min (normal: 2 at 30s
interval)
    DEATH_CERTIFICATE:  1,      // max 1 death cert/min (should be very
rare)
    PROFILE_SYNC:       2,      // max 2 syncs/min
    ENROLLMENT_REQUEST: 1,      // max 1 enrollment/min per device
    // All others:      5,      // default
}

fn check_rate_limit(
    limiter: &mut RateLimiter,
    sender: device_id,
    msg_type: MessageType,
    now: timestamp,
) → bool {
    let key = (sender, msg_type);
    let bucket = limiter.buckets.entry(key).or_insert_with(|| {
        let limit = RATE_LIMITS.get(&msg_type).unwrap_or(&5);
        TokenBucket {
            tokens: *limit,
            max_tokens: *limit,
            refill_rate: *limit,
            last_refill: now,
        }
    });

    // Refill tokens
    let elapsed_epochs = (now - bucket.last_refill) / EPOCH_DURATION;
    if elapsed_epochs > 0 {
        bucket.tokens = (bucket.tokens + bucket.refill_rate *
elapsed_epochs as u16)
                        .min(bucket.max_tokens);
        bucket.last_refill = now;
    }

    // Consume token
    if bucket.tokens > 0 {
        bucket.tokens -= 1;
        true  // allowed
    } else {
        false // rate limited — drop message
    }
}
```

ES-15.2 Duplicate Detection

```
// Sequence number tracking prevents replay attacks and duplicate
processing.

struct DuplicateDetector {
    seen: HashMap<device_id, SeqTracker>,
}

struct SeqTracker {
    last_seq:       u8,
    seen_bitmap:    u32,    // bitmap of last 32 sequence numbers
}

fn is_duplicate(
    detector: &mut DuplicateDetector,
    sender: device_id,
    seq_num: u8,
) -> bool {
    let tracker = detector.seen.entry(sender).or_insert(SeqTracker {
        last_seq: seq_num.wrapping_sub(1),
        seen_bitmap: 0,
    });

    let delta = seq_num.wrapping_sub(tracker.last_seq);

    if delta == 0 {
        return true;  // exact duplicate
    }

    if delta <= 32 {
        // Recent message — check bitmap
        let bit = 1u32 << (delta - 1);
        if tracker.seen_bitmap & bit != 0 {
            return true;  // already seen
        }
        tracker.seen_bitmap |= bit;
        return false;
    }

    if delta > 32 && delta < 224 {
        // Far ahead — accept, reset tracker (missed messages in between)
        tracker.last_seq = seq_num;
        tracker.seen_bitmap = 0;
        return false;
    }

    // delta >= 224: likely a wrapped-around old message, reject
    true
}
```

ES-15.3 Flood Detection and Temporary Blacklist

```
struct FloodDetector {
    counters:       HashMap<device_id, FloodCounter>,
    blacklist:      HashMap<device_id, timestamp>,  // blacklisted until
timestamp
}

struct FloodCounter {
    total_this_epoch: u16,
    rejected_this_epoch: u16,   // rate-limited messages count
}

FLOOD_THRESHOLD         = 50        // if >50 messages rejected in 1 epoch
→ blacklist
BLACKLIST_DURATION      = 300       // 5 minutes

fn check_flood(
    detector: &mut FloodDetector,
    sender: device_id,
    was_rate_limited: bool,
    now: timestamp,
) -> bool {
    // Check blacklist
    if let Some(&until) = detector.blacklist.get(&sender) {
        if now < until {
            return false;  // still blacklisted, drop silently
        }
        detector.blacklist.remove(&sender);
    }

    let counter = detector.counters.entry(sender).or_insert(FloodCounter {
        total_this_epoch: 0,
        rejected_this_epoch: 0,
    });

    counter.total_this_epoch += 1;
    if was_rate_limited {
        counter.rejected_this_epoch += 1;
    }

    if counter.rejected_this_epoch > FLOOD_THRESHOLD {
        // Blacklist this device
        detector.blacklist.insert(sender, now + BLACKLIST_DURATION);
        // Optionally: broadcast CONSISTENCY_ALERT about flooding device
        return false;
    }

    true
}

// Reset counters at epoch boundary:
fn reset_flood_counters(detector: &mut FloodDetector) {
    detector.counters.clear();
}
```

ES-15.4 BLE Mesh Layer Protection

```
MESH-LEVEL PROTECTIONS (in addition to application-level rate limiting):

1. NETWORK LAYER ENCRYPTION:
   - BLE Mesh encrypts all messages with NetKey (AES-128-CCM)
   - Devices without NetKey cannot inject messages
   - Protects against external (non-BIA) attackers

2. RELAY CACHE:
   - Each Nano node maintains a relay cache of last 32 message hashes
   - Prevents relay amplification attacks (same message relayed in circles)
   - Hash: BLAKE2s(msg[0..header_size + 8])[0..4] = 4-byte truncated hash

3. TTL ENFORCEMENT:
   - All relayed messages have TTL decremented
   - Messages with TTL=0 are NOT relayed
   - Default TTL: 3 (covers ~3 hops / ~30m BLE range / ~90m effective
range)
   - Vaccine messages: TTL=10 (wider propagation needed)

4. ADVERTISING RATE LIMIT:
   - BLE specification limits advertising to minimum 20ms interval
   - Hardware enforced — cannot be bypassed by compromised firmware
   - At 100ms advertising interval: max 10 advertisements/second per device
   - This is a natural hardware-level DoS ceiling
```

ES-16. LPWAN Gateway Trust Model

ES-16.1 Problem Statement

Dust nodes communicate via LPWAN (LoRa) through a gateway. The gateway wraps unsigned 8-byte compressed traces into signed 90-byte TRACE_DEPOSITs. This creates a trust dependency on the gateway. If a gateway is compromised, it can forge traces for any Dust device it manages.

ES-16.2 Trust Architecture

```
TRUST LEVELS (reflected in confidence field):

confidence_t values for gateway-vouched traces:
  DIRECT_SIGNED       = 255    // device signed with own Ed25519 key (Nano
nodes)
  GATEWAY_VOUCHED_OK  = 180    // gateway vouches, ≥2 gateways agree
  GATEWAY_VOUCHED_1   = 128    // single gateway vouching (default for
LPWAN)
  GATEWAY_UNVERIFIED  = 64     // gateway vouches but CRC check failed /
timing anomaly
  GATEWAY_SUSPECT     = 0      // gateway under investigation

IMPACT ON BIA MECHANISMS:
  - Pheromone field: confidence is multiplied into weight, so gateway-
vouched traces
    have 50-70% influence of directly-signed traces.
  - Behavioral profile: gateway-vouched traces update profile normally, but
anomaly
    thresholds are relaxed by 20% (to account for gateway timing jitter).
  - NK audit: drift threshold raised by 1.0σ for Dust devices
(DRIFT_THRESHOLD = 6.0σ
    instead of 5.0σ) because Dust profiles are noisier.
```

ES-16.3 Gateway Verification Protocol

```
CROSS-GATEWAY VALIDATION:

If a Dust device is in range of multiple gateways (ideal deployment):

  Gateway A receives compressed trace from Dust device D
  Gateway B receives same compressed trace from Dust device D

  Both gateways independently produce TRACE_DEPOSITs:
    trace_A = { sender=D, reading=X, timestamp=T_a, sig=sig_A }
    trace_B = { sender=D, reading=X, timestamp=T_b, sig=sig_B }

  Nano node receives both:
    if trace_A.reading == trace_B.reading:
      confidence = GATEWAY_VOUCHED_OK (180)    // corroborated by 2
gateways
    else:
      // Gateways disagree — one may be compromised
      confidence = GATEWAY_UNVERIFIED (64)
      trigger INFLAMMATION for both gateways
```

```
GATEWAY HEALTH MONITORING (run on Nano nodes):

struct GatewayProfile {
    gateway_id:         device_id,
    dust_devices:       [device_id; 256],   // Dust devices this gateway
serves
    dust_count:         u16,
    traces_received:    u32,                // total traces received via
this gateway
    anomalies_flagged:  u16,                // traces that triggered immune
response
    anomaly_rate:       f16,                // anomalies / total (EMA
smoothed)
    last_cross_validated: timestamp,        // when another gateway last
corroborated
}

fn evaluate_gateway_trust(
    gw: &mut GatewayProfile,
    trace_was_anomalous: bool,
) -> confidence_t {
    gw.traces_received += 1;
    if trace_was_anomalous {
        gw.anomalies_flagged += 1;
    }

    // EMA anomaly rate
    let instant_rate = if trace_was_anomalous { 1.0 } else { 0.0 };
    gw.anomaly_rate = 0.99 * gw.anomaly_rate + 0.01 * instant_rate;

    if gw.anomaly_rate > 0.20 {
        // >20% anomaly rate from this gateway — suspect
        GATEWAY_SUSPECT  // 0: effectively removes all Dust traces via this
gateway
    } else if gw.anomaly_rate > 0.05 {
        GATEWAY_UNVERIFIED  // 64: reduced trust
    } else if now() - gw.last_cross_validated < 3600 {
        GATEWAY_VOUCHED_OK  // 180: recently corroborated by another
gateway
    } else {
        GATEWAY_VOUCHED_1  // 128: normal single-gateway trust
    }
}
```

ES-16.4 Gateway Registration

```
GATEWAY ENROLLMENT:
  Gateways are Nano-class devices with LoRa radio in addition to BLE/WiFi.
  They enroll like any Nano node (ES-13.2) plus:

  1. Operator registers gateway with its Dust device table:
       for each Dust device:
         register_dust(dust_id_short, dust_device_id, dust_aes_key)

  2. Gateway advertises GATEWAY_CAPABILITY in MODE_ANNOUNCE:
       capabilities_flags bit 4 = LPWAN_GATEWAY

  3. Other Nano nodes recognize gateway and create GatewayProfile.

  4. Gateway periodically broadcasts GATEWAY_STATUS:
       { gateway_id, dust_device_count, uptime, anomaly_rate_self_report }
       (This is informational — Nano nodes compute their own anomaly rate.)
```

```
DUST_ID_SHORT COLLISION HANDLING:
  sender_id_short is 2 bytes = 65536 possible values.
  For deployments < 1000 Dust devices: collision probability < 1%.
  If collision detected (same id_short, different traces):
    Gateway disambiguates by LoRa device address (DevAddr in LoRaWAN) or
    timing pattern (each Dust device has slightly different crystal
frequency,
    creating a natural "timing fingerprint" at ±5ppm resolution).
```

This engineering specification is a companion to the KRILL-BIA research paper (v1.0, February 2026). It provides the implementation-level detail needed to build a working prototype. Where the research paper says "what" and "why," this document says "how" and "exactly how many bytes."

Version 0.2 — Updated with: main loop (ES-10), distance estimation (ES-11), transport layer (ES-12), crypto init (ES-13), flash layout (ES-14), rate limiting (ES-15), LPWAN gateway trust (ES-16). Fixed: prev_hash 2B→4B, quorum hysteresis consistency, test vector comments.

Supporting This Work

This engineering specification — and the KRILL project as a whole — is open and freely available. If you implement this protocol, use it in a product, or build research on top of it, please consider supporting further development:

Ethereum / ERC-20 / Base / Arbitrum / Polygon:

```
0x0BC290355c0B16B5B247701B7BC9AB2E1e61ffa7
```

What your support enables:

- Reference firmware for ESP32 (Nano) and nRF52840 (Dust)
- Hardware test lab with 50+ device mesh for validation
- Formal TLA+ model of consensus and NK audit protocols
- Bounty program for community-discovered spec bugs

Code contributions are equally welcome — see the project repository for open issues and contribution guidelines.

KRILL Bio-Inspired Architecture — February 2026