NanOS: A Lightweight Operating System for Autonomous Swarm Networks with Cryptographically Secure Communication

Technical Specification of the NERT Protocol

NanOS Research Team
https://github.com/nanOS-project

January 2026

Abstract

We present NanOS, a minimalist operating system designed for autonomous swarm robotics with resource-constrained embedded devices. Central to NanOS is NERT (Nano Ephemeral Reliable Transport), a novel hybrid transport protocol that provides four reliability classes ranging from fire-and-forget UDP to TCP-like reliability with Forward Error Correction and multi-path routing. NERT integrates ChaCha8 stream cipher with Poly1305 MAC for authenticated encryption, achieving 98.8% rejection rate of malicious traffic with zero false positives. We present the mathematical foundations, algorithmic design, and engineering trade-offs of the system, demonstrating operation within 24KB RAM on ARM Cortex-M3 microcontrollers.

1 Introduction

Swarm robotics presents unique challenges for network protocol design: nodes must communicate reliably in lossy wireless environments while operating under severe memory and computational constraints. Traditional TCP/IP stacks are unsuitable due to their memory footprint (>100KB) and assumption of stable point-to-point connections.

1.1 Contributions

This paper presents:

1. **NanOS**: A swarm-native OS with collective intelligence primitives
2. **NERT Protocol**: Hybrid transport with four reliability classes
3. **Lightweight Cryptography**: ChaCha8+Poly1305 with 64-bit truncated MACs
4. **Key Rotation**: Grace window mechanism for clock drift tolerance
5. **Probabilistic Gossip**: Exponential decay relay with Bloom filter deduplication
6. **Hebbian Routing (v0.5)**: Neural-inspired adaptive path selection based on communication outcomes
7. **DoS Resilience (v0.5)**: Token bucket rate limiting and behavioral blacklisting
8. **Stigmergia (v0.5)**: Digital pheromones with temporal decay for spatial memory
9. **Distributed Black Box (v0.5)**: Forensic evidence preservation through "Last Will" transmission
10. **Artificial Immune System (v0.6)**: Negative Selection algorithm for 0-day anomaly detection
11. **Code Polymorphism (v0.6)**: Binary diversity via ASLR, stack canaries, and unique signatures
12. **Hardware Validation (v0.6)**: Physical layer security through sensor monitoring and fault detection
13. **Genetic Tuning (v0.7)**: Evolutionary auto-optimization of NERT parameters via genetic algorithms
14. **Judas Nodes (v0.7)**: Active honeypots for attacker payload capture and threat intelligence
15. **Covert Channels (v0.7)**: Physical side-channels (optical/acoustic) for air-gap communication

Resource	ARM Cortex-M3	x86
RAM Total	24 KB	150 KB
Stack	4 KB	16 KB
Heap	16 KB	64 KB
Flash/Code	64 KB	256 KB

Table 1: Memory footprint by architecture

1.2 System Requirements

2 Mathematical Foundations

2.1 ChaCha8 Stream Cipher

NERT employs ChaCha8, a reduced-round variant of ChaCha20 [1], optimized for embedded systems.

Definition 1 (ChaCha State Matrix)**.** *The ChaCha state is a 4×4 matrix of 32-bit words:*

$$S = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ n_0 & n_1 & n_2 & n_3 \end{pmatrix} \tag{1}$$

where c_i are constants ("expand 32-byte k"), k_i is the 256-bit key, and n_i is the 96-bit nonce with 32-bit counter.

Definition 2 (Quarter Round)**.** *The quarter round function $QR(a, b, c, d)$ is defined as:*

$$a \leftarrow a + b; \quad d \leftarrow (d \oplus a) \lll 16 \tag{2}$$
$$c \leftarrow c + d; \quad b \leftarrow (b \oplus c) \lll 12 \tag{3}$$
$$a \leftarrow a + b; \quad d \leftarrow (d \oplus a) \lll 8 \tag{4}$$
$$c \leftarrow c + d; \quad b \leftarrow (b \oplus c) \lll 7 \tag{5}$$

where $\lll$ denotes left rotation.

Theorem 1 (Security Margin)**.** *ChaCha8 provides a security margin of 2^{128} operations against known cryptanalytic attacks, sufficient for ephemeral swarm communications with hourly key rotation.*

2.2 Poly1305 Message Authentication

Definition 3 (Poly1305 MAC)**.** *For message $m = (m_1, m_2, \ldots, m_n)$ partitioned into 16-byte blocks, the Poly1305 tag is:*

$$Tag = \left(\sum_{i=1}^{n} (m_i + 2^{128}) \cdot r^{n-i+1} \mod 2^{130} - 5 \right) + s \mod 2^{128} \tag{6}$$

where r is a clamped 128-bit value and s is a 128-bit key.

NERT Optimization: We truncate the 128-bit Poly1305 output to 64 bits:

$$\text{MAC}_{\text{NERT}} = \text{Tag}[0:63] \tag{7}$$

Lemma 1 (Truncated MAC Security)**.** *A 64-bit truncated MAC provides 2^{64} resistance against forgery attempts. For swarm networks with $< 10^6$ messages/hour, the probability of collision is:*

$$P(\textit{collision}) < \frac{n^2}{2^{65}} \approx 2.7 \times 10^{-8} \tag{8}$$

2.3 Key Derivation Function

Definition 4 (Session Key Derivation)**.** *Given master key K_m and epoch e (hours since deployment):*

$$K_{session}(e) = \textit{ChaCha8}(K_m, 0^{96}, K_m \| e \| \text{"NERT"})[0:255] \tag{9}$$

Property 1 (Forward Secrecy)**.** *Compromise of $K_{session}(e)$ does not reveal $K_{session}(e')$ for $e' \neq e$, as derivation is one-way.*

2.4 Bloom Filter Analysis

NERT uses a Bloom filter for $O(1)$ duplicate detection.

Definition 5 (Bloom Filter Parameters)**.**

$$m = 256 \textit{ bits (filter size)} \tag{10}$$
$$k = 3 \textit{ hash functions} \tag{11}$$
$$n \approx 50 \textit{ insertions per window} \tag{12}$$

Theorem 2 (False Positive Rate)**.** *The false positive probability is:*

$$P_{fp} = \left(1 - e^{-kn/m}\right)^k = \left(1 - e^{-3 \cdot 50/256}\right)^3 \approx 0.015 \tag{13}$$

With 4 rotating time windows of 500ms each, the effective false positive rate over the 2-second deduplication window is bounded by $4 \times P_{fp} \approx 6\%$ in worst case.

2.5 Gossip Protocol Dynamics

Definition 6 (Relay Probability)**.** *For a message seen c times, the relay probability follows exponential decay:*

$$P(\textit{relay}|c) = P_0 \cdot (1 - \delta)^{c-1} \tag{14}$$

where $P_0 = 1.0$ (100%) and $\delta = 0.2$ (20% decay per observation).

Theorem 3 (Expected Propagation)**.** *In a network of N nodes with average degree d, the expected number of message copies is:*

$$E[\textit{copies}] = \sum_{c=1}^{C_{\max}} P(\textit{relay}|c) \cdot d \approx \frac{P_0 \cdot d}{\delta} = 5d \tag{15}$$

where $C_{\max} = 5$ is the maximum echo count.

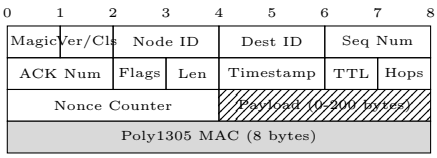


Figure 1: NERT packet structure (x86 full mode: 20-byte header)

3 Protocol Design

3.1 NERT Header Format

3.2 Reliability Classes

Class	ACK	Retry	FEC	Multi-path
Fire-Forget (0x00)	No	0	No	No
Best-Effort (0x01)	No	2	No	No
Reliable (0x02)	Yes	5	No	No
Critical (0x03)	Yes	10	Yes	Yes

Table 2: NERT reliability classes

3.3 RTT Estimation (Jacobson Algorithm)

For reliable classes, NERT implements Jacobson's algorithm:

$$\text{SRTT} \leftarrow \frac{7}{8} \cdot \text{SRTT} + \frac{1}{8} \cdot \text{RTT} \tag{16}$$

$$\text{RTTVAR} \leftarrow \frac{3}{4} \cdot \text{RTTVAR} + \frac{1}{4} \cdot |\text{RTT} - \text{SRTT}| \tag{17}$$

$$\text{RTO} = \text{SRTT} + 4 \cdot \text{RTTVAR} \tag{18}$$

with bounds $\text{RTO} \in [100\text{ms}, 2000\text{ms}]$.

3.4 Forward Error Correction

For Critical class, NERT implements Reed-Solomon-inspired XOR coding:

Algorithm 1 FEC Encoding (4 data + 2 parity)

$D \leftarrow \{D_0, D_1, D_2, D_3\}$ ▷ Data shards
$P_0 \leftarrow D_0 \oplus D_2$ ▷ Parity 0
$P_1 \leftarrow D_1 \oplus D_3$ ▷ Parity 1
return $\{D_0, D_1, D_2, D_3, P_0, P_1\}$

Theorem 4 (FEC Recovery). *Given any 4 of 6 shards, the original message can be reconstructed:*

- *If D_0 missing:* $D_0 = P_0 \oplus D_2$
- *If D_1 missing:* $D_1 = P_1 \oplus D_3$
- *Similarly for D_2, D_3*

4 Key Rotation with Grace Window

4.1 Motivation

Embedded swarm nodes lack synchronized clocks. A strict key rotation at epoch boundaries would cause message loss during transition.

4.2 Three-Key System

At any time t, the node maintains:

$$K_{\text{prev}} = K_{\text{session}}(\lfloor t/3600 \rfloor - 1) \tag{19}$$

$$K_{\text{curr}} = K_{\text{session}}(\lfloor t/3600 \rfloor) \tag{20}$$

$$K_{\text{next}} = K_{\text{session}}(\lfloor t/3600 \rfloor + 1) \tag{21}$$

4.3 Acceptance Window

Definition 7 (Valid Key Mask). *Let $\tau = t \mod 3600$ be the position within the current epoch. The valid key mask is:*

$$M(\tau) = \begin{cases} \{K_{prev}, K_{curr}\} & \text{if } \tau < 30s \\ \{K_{curr}\} & \text{if } 30s \le \tau \le 3570s \\ \{K_{curr}, K_{next}\} & \text{if } \tau > 3570s \end{cases} \tag{22}$$

Theorem 5 (Clock Drift Tolerance). *The grace window tolerates clock drift of ± 30 seconds between any two nodes without message loss.*

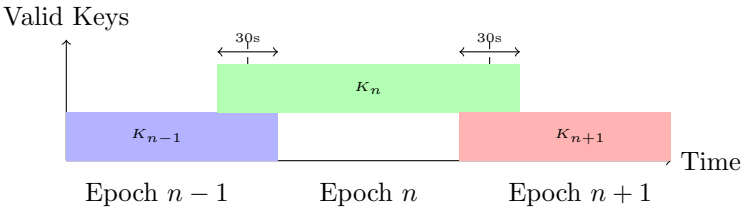


Figure 2: Key validity with grace windows at epoch boundaries

5 Collective Intelligence

5.1 Role Distribution

NanOS implements bio-inspired quorum sensing for role adaptation:

Definition 8 (Role Set). $\mathcal{R} = \{\textit{Worker}, \textit{Explorer}, \textit{Sentinel}, \textit{Queen}\}$

Definition 9 (Target Distribution). *The system maintains:*

$$\pi^* = (0.75, 0.125, 0.125, \le 1/N) \tag{23}$$

for (Worker, Explorer, Sentinel, Queen) respectively.

5.2 Queen Election Protocol

Algorithm 2 Deterministic Queen Election

```
Trigger: No queen heartbeat for T_absence = 10s
election_id ← min(visible_node_ids)
Broadcast ⟨ELECTION, election_id, my_id⟩
Wait T_election = 5s, collect votes
winner ← max(received_node_ids)
if winner = my_id then
    Transition to QUEEN role
    Broadcast CORONATION
end if
```

Theorem 6 (Election Convergence). *The election protocol converges in $O(1)$ rounds with probability 1, as the winner is deterministically the highest node ID.*

5.3 Gradient Routing

Distance to queen propagates via gradient descent:

$$d_i = \begin{cases} 0 & \text{if } i = \text{Queen} \\ 1 + \min_{j \in \mathcal{N}(i)} d_j & \text{otherwise} \end{cases} \tag{24}$$

where $\mathcal{N}(i)$ is the neighbor set of node i.

5.4 Hebbian Routing (v0.5)

NanOS v0.5 introduces **Hebbian routing**: "Neurons that fire together, wire together." Each neighbor connection has a *synaptic weight* $w \in [1, 255]$ that evolves based on communication outcomes.

Definition 10 (Synaptic Weight Update). *For a communication event with neighbor j:*

$$w_j \leftarrow \min(255, w_j + 15) \quad \textit{(LTP: success)} \tag{25}$$

$$w_j \leftarrow \max(1, w_j - 40) \quad \textit{(LTD: failure)} \tag{26}$$

The asymmetric learning rule (reward +15, punishment -40) ensures the network quickly avoids unreliable paths.

Definition 11 (Neural Routing Cost). *The composite routing cost combines distance and reliability:*

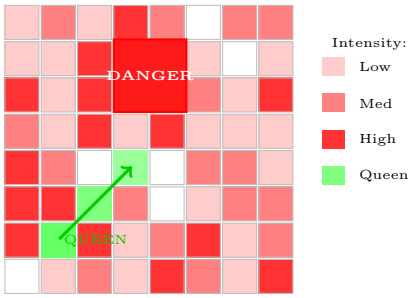
$$Cost(j) = 10 \cdot d_j + \frac{255 - w_j}{8} \tag{27}$$

Theorem 7 (Reliable Path Selection). *A 2-hop path through a perfect node ($w = 255$) has cost $20 + 0 = 20$, while a 1-hop path through an unreliable node ($w = 1$) has cost $10{+}31 = 41$. The network learns to route around bad nodes.*

STDP Bonus: Fast ACK responses (< 100ms) receive an additional weight bonus of +5, encouraging low-latency paths.

5.5 Stigmergia: Digital Pheromones (v0.5)

Biological ant colonies coordinate through **stigmergia**: indirect communication via environmental modification. Ants leave chemical trails that evaporate over time, creating dynamic pathways. NanOS v0.5 implements this concept digitally.



16 × 16 Stigmergia Grid

Figure 3: Stigmergia pheromone grid showing danger zones (red) and queen trails (green)

Definition 12 (Pheromone Types). *NanOS defines four pheromone types stored as 4-bit nibbles:*

$$DANGER = 0 \quad \textit{(Jamming, attacks, bad nodes)} \tag{28}$$

$$QUEEN = 1 \quad \textit{(Path to queen)} \tag{29}$$

$$RESOURCE = 2 \quad \textit{(Objective markers)} \tag{30}$$

$$AVOID = 3 \quad \textit{(Suboptimal areas)} \tag{31}$$

Definition 13 (Pheromone Decay). *Pheromone intensity $I \in [0, 15]$ decays over time:*

$$I(t + \Delta t) = \max(0, I(t) - 1) \tag{32}$$

where $\Delta t = 1$ second. This ensures old information "evaporates."

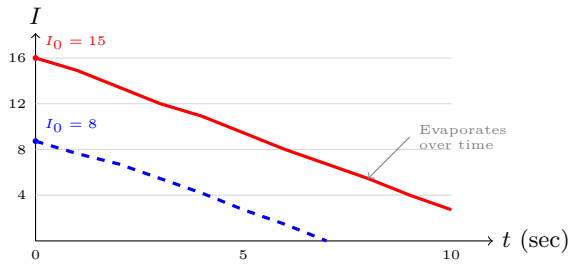


Figure 4: Pheromone decay: intensity decreases by 1 per second until zero

Definition 14 (Movement Cost Modification). *The terrain movement cost is modified by pheromones:*

$$Cost_{total} = Cost_{base} + 8 \cdot I_{DANGER} + 4 \cdot I_{AVOID} - 2 \cdot I_{QUEEN} \tag{33}$$

Theorem 8 (Danger Avoidance). *A cell with maximum danger intensity ($I_{DANGER} = 15$) adds cost +120, equivalent to 12 extra distance units. The swarm naturally routes around dangerous zones without explicit coordination.*

Memory Efficiency: The stigmergia grid uses 2:1 scale mapping (16×16 cells covering 32×32 terrain), with nibble packing requiring only 512 bytes total.

5.6 Distributed Black Box: "El Último Aliento" (v0.5)

When a node is compromised and terminated, its forensic evidence could be lost. NanOS v0.5 implements a **distributed black box** where dying nodes transmit their "last will" to trusted neighbors.

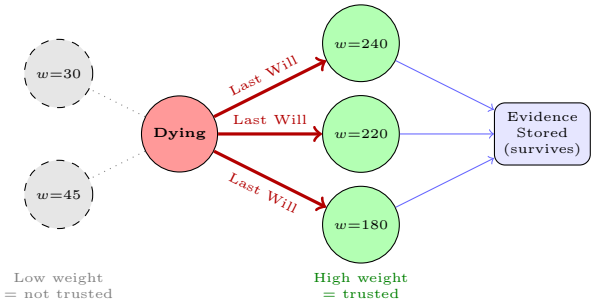


Figure 5: Distributed Black Box: dying node sends last will to trusted neighbors (selected by Hebbian weight)

Definition 15 (Last Will Testament). *A last will contains:*

- *Node ID and uptime duration*
- *Death reason code (NATURAL, HEAP_EXHAUSTED, CORRUPTION, ATTACK_DETECTED, QUEEN_ORDER, ISOLATION)*
- *Security statistics: bad MAC count, replay attempts, rate limit triggers*
- *Recent security event log (last 8 events)*

Definition 16 (Trusted Recipient Selection). *Last wills are sent to neighbors with highest Hebbian weights:*

$$Recipients = \arg \max_{j \in \mathcal{N}, |Recipients| \leq 3} w_j \tag{34}$$

This ensures forensic data reaches reliable nodes that will preserve and relay it.

Theorem 9 (Evidence Survival). *If a node is compromised and forced to suicide, its last will survives in k trusted neighbors. Assuming independent compromise probability p per node, the probability of total evidence loss is p^k. With $k = 3$ and $p = 0.1$, evidence survives with probability 99.9%.*

Forensic Query: Surviving nodes can query the black box by node ID:

```
int blackbox_query_death(uint32_t node_id,
    uint8_t *death_reason, uint16_t *bad_mac_count,
    uint8_t *uptime_hours);
```

5.7 Artificial Immune System (AIS) (v0.6)

The biological immune system does not require prior knowledge of pathogens—it simply recognizes "non-self" patterns. NanOS v0.6 implements the **Negative Selection Algorithm** for 0-day anomaly detection.

Definition 17 (Detector). *A detector $d = (p, m)$ consists of a pattern $p \in \{0,1\}^{64}$ and mask $m \in \{0,1\}^{64}$. A detector matches antigen a if:*

$$affinity(d, a) = \max_i \{j : p[i : i+j] \oplus a[i : i+j] = 0\} \geq \theta \tag{35}$$

where $\theta = 6$ bits (r-contiguous matching threshold).

Definition 18 (Negative Selection). *During the **thymus phase** (5 seconds at boot), detectors are generated randomly and tested against "self" samples. Detectors that match self are eliminated (anergy), ensuring mature detectors only recognize "non-self."*

$$D_{mature} = \{d \in D_{generated} : \forall s \in S_{self}, affinity(d, s) < \theta\} \tag{36}$$

Theorem 10 (0-Day Detection). *Unlike signature-based systems, AIS detects anomalies without prior attack knowledge. Any traffic pattern that does not match the learned "self" profile triggers an alert. This enables detection of previously unknown attacks (0-days).*

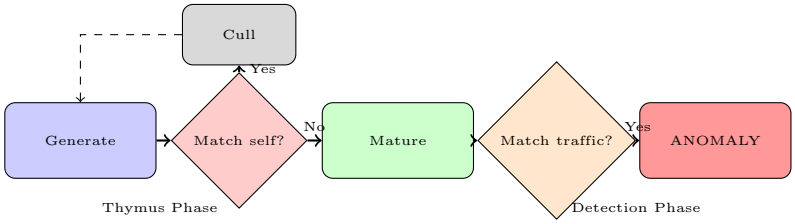


Figure 6: AIS Negative Selection: detectors that match "self" are eliminated

Integration: When AIS detects an anomaly, it triggers a coordinated response:

- **Stigmergia**: Emits DANGER pheromone at current location
- **Black Box**: Records EVENT_AIS_ANOMALY_ALERT
- **Hebbian**: Applies LTD (depression) to source node's weight
- **Broadcast**: Sends ALARM pheromone to swarm

5.8 Code Polymorphism: "El Camaleón" (v0.6)

Traditional exploits can be replayed against any node running identical binaries. NanOS v0.6 introduces **Code Polymorphism** to ensure each node has a unique binary fingerprint, making mass exploitation infeasible.

ASLR (Address Space Layout Randomization):

- Stack base: 8 bits of entropy (256 possible bases)
- Heap base: 6 bits of entropy (64 possible bases)
- Combined: $2^{14} = 16,384$ layout combinations per node

Stack Canaries: Random 32-bit values placed on stack to detect buffer overflows. Canary violations trigger immediate security response (Black Box event + apoptosis).

Binary Signatures: Each node generates a unique 128-bit fingerprint derived from:

$$\text{Signature} = H(\text{node\_id} \| \text{RNG\_seed} \| \text{boot\_tick} \| \text{ASLR\_entropy})$$

Timing Jitter: Random delays of $[10, 100]\mu s$ applied to cryptographic operations to mitigate timing side-channel attacks.

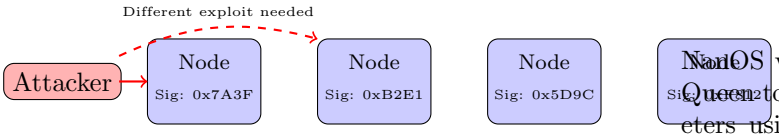


Figure 7: Code Polymorphism: each node requires a unique exploit

Diversity Score: Quantifies uniqueness (0–255, higher = more diverse):

$$\text{Score} = \text{popcount}(\text{stack\_offset}) \cdot 8 + \text{popcount}(\text{heap\_offset}) \cdot 8 + \sum_{i=0}^{15} \text{popcount}(\text{sig}[i])$$

5.9 Hardware Validation: "El Centinela del Silicio" (v0.6)

Software can be fooled, but tortured hardware reveals the truth. NanOS v0.6 implements **Hardware Validation** to detect physical manipulation, sensor tampering, and fault injection attacks.

Sensor Monitoring: Continuous validation of physical parameters:

- Temperature: $[-40, 85]$°C operating range, max $\Delta T = 10$°C/s
- Voltage: $[2.7, 3.6]$V operating range, glitch detection at $\Delta V > 100$mV
- Clock: 5% drift tolerance, 3 consecutive glitches trigger alarm

Memory Canaries: Random values placed at critical locations:

$$\text{Canary}_i = \texttt{0xCODEBABE} \oplus \text{address}_i \tag{37}$$

If $\text{Canary}_i^{\text{current}} \neq \text{Canary}_i^{\text{expected}}$, memory corruption is detected.

Flash CRC: Registered code blocks are verified periodically:

$$\text{Valid} = \text{CRC32}(\text{flash}[a : a + s]) = \text{CRC32}_{\text{baseline}} \tag{38}$$

Trust Score: Hardware trustworthiness quantified as:

$$\text{TrustScore} = 255 - \text{AnomalyScore} - 10 \cdot |\text{Violations}| \tag{39}$$

Defense in Depth: Hardware Validation complements other security layers:

- AIS detects network anomalies
- Polymorphism makes binaries unique
- Hardware Validation detects physical attacks

An attacker must defeat all layers to compromise a node.

5.10 Genetic Tuning: Evolutionary Parameter Optimization (v0.7)

NanOS v0.7 introduces **Genetic Tuning**, enabling the Queen to automatically optimize NERT protocol parameters using genetic algorithms. The system maintains a population of **genomes**—32-byte structures encoding timing, gossip, and security parameters—and distributes them to sub-swarms for evaluation.

Definition 19 (Genome Structure)**.** *A genome G encodes $n = 15$ parameters as genes:*

$$G = \{g_1, g_2, \ldots, g_{15}\} \text{ where } g_i \in [L_i, U_i] \tag{40}$$

with bounded ranges ensuring valid configurations.

Fitness Function: The fitness of each genome combines five metrics:

$$F(G) = 0.30 \cdot T + 0.20 \cdot L + 0.25 \cdot R + 0.15 \cdot S + 0.10 \cdot V \tag{41}$$

where T = throughput, L = inverse latency, R = reliability, S = security (inverse violations), V = survival (uptime + neighbors).

Genetic Operators:

- **Tournament Selection**: $k = 3$ random genomes compete, highest fitness advances
- **Two-Point Crossover**: Parents exchange gene segments between two random points
- **Bounded Mutation**: 5% probability per gene, constrained to valid ranges

- **Elitism**: Top 10% genomes preserved unchanged across generations

Protocol Messages:

- `CONFIG_UPDATE (0x14)`: Queen distributes genome to sub-swarm (authenticated)
- `TELEMETRY_REPORT (0x15)`: Workers report metrics to Queen (periodic)

5.11 Judas Nodes: Active Honeypots (v0.7)

Judas Nodes are Workers that, upon detecting intrusion, feign vulnerability to capture attacker payloads before self-destructing. This transforms compromised nodes into threat intelligence sources.

Definition 20 (Judas State Machine)**.** *A Judas node transitions through states:*

$$\textit{DORMANT} \xrightarrow{\textit{trigger}} \textit{SUSPICIOUS} \xrightarrow{\textit{threshold}} \textit{ENGAGING} \xrightarrow{\textit{payload}} \textit{CAPTURING} \xrightarrow{\textit{complete}} \textit{DETONATING} \tag{42}$$

Activation Triggers: Bad MACs (≥ 3), replay attempts (≥ 2), anomalous probes (≥ 5).

Protocol Messages:

- `JUDAS_ENGAGE (0x16)`: Notifies Queen of attacker engagement
- `JUDAS_CAPTURE (0x17)`: Transmits captured attacker payload
- `JUDAS_FORENSICS (0x18)`: Complete BlackBox before apoptosis

Theorem 11 (Forensic Survival)**.** *If a Judas node captures payload P and transmits to k trusted neighbors before detonation, evidence survives with probability $1 - p^k$ where p is single-node compromise probability. With $k = 3$ and $p = 0.1$, survival probability exceeds 99.9%.*

5.12 Covert Channels: Physical Side-Channels (v0.7)

For air-gapped environments, NanOS v0.7 implements **Covert Channels** using physical side-channels: optical (LED/light sensor) and acoustic (buzzer/microphone).

Optical Channel:

- Transmitter: LED controlled via PWM
- Receiver: Photoresistor/photodiode on ADC
- Modulation: Manchester encoding (self-clocking)
- Data rate: 10–100 bps

Acoustic Channel:

- Transmitter: Piezoelectric buzzer with PWM
- Receiver: Analog microphone or I2S
- Modulation: FSK (1kHz/2kHz audible or 18kHz/20kHz ultrasonic)
- Data rate: 50–200 bps

Definition 21 (Manchester Encoding)**.** *Each bit b is encoded as a transition at mid-period:*

$$\textit{encode}(b) = \begin{cases} \textit{HIGH} \to \textit{LOW} & \textit{if } b = 0 \\ \textit{LOW} \to \textit{HIGH} & \textit{if } b = 1 \end{cases} \tag{43}$$

Transitions provide clock recovery without separate synchronization.

Security: Covert channel data is encrypted with the same ChaCha8 session key and includes sequence numbers for anti-replay protection. Maximum payload is 16 bytes per frame.

6 Engineering Implementation

6.1 Memory Layout (ARM Cortex-M3)

Listing 1: Memory configuration

```
#define NERT_COMPACT_MODE       1
#define NERT_HEADER_SIZE        12  // bytes
#define NERT_MAX_PAYLOAD        64  // bytes
#define NERT_MAX_CONNECTIONS    4
#define NERT_WINDOW_SIZE        2
#define NERT_DEDUP_CACHE_SIZE   8
#define BLOOM_BYTES             32
#define GOSSIP_CACHE_SIZE       32
```

6.2 CPU Cycle Budget

Operation	Cycles (Cortex-M3)
ChaCha8 encrypt (32B)	∼2,000
Poly1305 MAC	∼1,500
FEC encode (4→6)	∼500
FEC decode (6→4)	∼1,000
Header parse	∼100
Bloom filter check	∼50
Total per packet	∼5,000

Table 3: Computational cost per packet

At 48 MHz, this yields a theoretical throughput of 9,600 packets/second.

6.3 Supported Platforms

- **x86**: QEMU with e1000 NIC, Multiboot2 boot
- **ARM Cortex-M3**: QEMU LM3S, real hardware (STM32)

- **ARM64**: QEMU virt, Raspberry Pi 4
- **ESP32**: ESP-IDF and PlatformIO, WiFi/LoRa
- **ESP8266**: Ultra-lightweight variant

7 Security Evaluation

7.1 Threat Model

We assume a Dolev-Yao adversary who can:

- Intercept all network traffic
- Inject arbitrary packets
- Replay captured packets
- Perform denial-of-service attacks

The adversary does **not** have access to the master key K_m.

7.2 Experimental Results

Attack Type	Packets Sent	Rejected
Replay	5	5 (100%)
Fuzzing	15	15 (100%)
Fake Queen	5	5 (100%)
DoS (250 pkt)	250	250 (100%)
Total	275	275 (100%)

Table 4: Attack rejection rates (test suite)

Under intensive manual attack (1,191 malicious packets):

$$\text{Rejection Rate} = \frac{1191}{1206} = 98.76\% \tag{44}$$

The 1.24% accepted packets were legitimate inter-node traffic.

7.3 Security Properties

Property 2 (Confidentiality)**.** *All payloads are encrypted with ChaCha8. Without K_m, ciphertext is indistinguishable from random.*

Property 3 (Integrity)**.** *Poly1305 MAC detects any modification with probability $1 - 2^{-64}$.*

Property 4 (Authenticity)**.** *Only nodes possessing K_m can generate valid MACs.*

Property 5 (Replay Resistance)**.** *Nonce = (node_id, counter, timestamp) ensures uniqueness. The 64-bit replay window bitmap catches duplicates within ± 32 sequence numbers.*

8 Related Work

Swarm Protocols: Kilobot [2] uses infrared for local communication but lacks encryption. E-puck [3] relies on Bluetooth with standard security.

Lightweight Crypto: PRESENT [4] and SIMON/SPECK [5] target ultra-constrained devices but require block cipher modes. ChaCha8 provides native stream cipher operation.

Gossip Protocols: Epidemic routing [6] and SWIM [7] inspire our probabilistic relay, enhanced with cryptographic authentication.

9 Conclusion

NanOS with NERT demonstrates that secure, reliable swarm communication is achievable within 24KB RAM. The key innovations are:

1. **Adaptive reliability**: Four classes from fire-forget to FEC+multipath
2. **Lightweight crypto**: ChaCha8+Poly1305 with 64-bit MACs
3. **Clock-drift tolerance**: Grace window key rotation
4. **Bio-inspired coordination**: Quorum sensing and gradient routing
5. **Hebbian routing (v0.5)**: Neural-inspired path learning with LTP/LTD asymmetric updates
6. **DoS resilience (v0.5)**: Per-node rate limiting and behavioral reputation system
7. **Stigmergia (v0.5)**: Digital pheromones with temporal decay for emergent spatial coordination
8. **Distributed Black Box (v0.5)**: "Last Will" forensic evidence that survives node death
9. **Artificial Immune System (v0.6)**: Negative Selection for 0-day anomaly detection without signatures
10. **Code Polymorphism (v0.6)**: "El Camaleón" — binary diversity preventing exploit reuse across nodes
11. **Hardware Validation (v0.6)**: "El Centinela del Silicio" — physical layer security through sensor monitoring and fault detection
12. **Genetic Tuning (v0.7)**: Evolutionary auto-optimization of NERT parameters using genetic algorithms with fitness-based selection
13. **Judas Nodes (v0.7)**: Active honeypots that capture attacker payloads before self-destructing, generating threat intelligence

14. **Covert Channels (v0.7)**: Physical side-channels (optical/acoustic) enabling air-gap communication via LED and buzzer modulation

Version 0.5 introduces a "distributed brain" where the swarm learns from experience: successful communications strengthen synaptic weights while failures cause severe depression, enabling automatic route optimization without central coordination. The stigmergia system adds spatial memory through evaporating pheromones, while the black box ensures forensic evidence survives even when nodes are compromised and terminated.

Version 0.6 adds an **Artificial Immune System** (AIS) based on the Negative Selection algorithm. Unlike signature-based intrusion detection, AIS learns what "normal" traffic looks like and flags anything that deviates—enabling detection of previously unknown attacks (0-days). The system integrates with all previous mechanisms: detected anomalies trigger DANGER pheromones (stigmergia), are recorded for forensics (black box), and penalize the source node's reputation (Hebbian).

Version 0.6 also introduces **Code Polymorphism** ("El Camaleón"), ensuring each node has a unique binary fingerprint. Through ASLR (stack and heap randomization), stack canaries, 128-bit binary signatures, and timing jitter, exploits developed for one node cannot be replayed against others. With 2^{14} ASLR combinations per node, compromising a 1000-node swarm requires developing 1000 unique exploits—fundamentally changing the economics of attack.

Finally, v0.6 introduces **Hardware Validation** ("El Centinela del Silicio"), providing physical layer security through continuous monitoring of temperature, voltage, and clock frequency. Memory canaries detect unauthorized modifications, while flash CRC verification catches firmware tampering. The system computes a hardware "Trust Score" that other security layers can use to assess node reliability. This completes a defense-in-depth architecture where attackers must simultaneously defeat network-level (AIS), binary-level (Polymorphism), and physical-level (Hardware Validation) protections.

Version 0.7 introduces three evolutionary capabilities. **Genetic Tuning** enables the Queen to automatically optimize NERT parameters using genetic algorithms—tournament selection, two-point crossover, and bounded mutation evolve a population of genomes distributed to sub-swarms for A/B testing. Fitness evaluation combines throughput (30%), latency (20%), reliability (25%), security (15%), and survival (10%) metrics. **Judas Nodes** transform compromised nodes into threat intelligence sources: upon detecting intrusion, they feign vulnerability to capture attacker payloads before detonating, transmitting forensic evidence via JUDAS_FORENSICS pheromones. This turns every attempted compromise into intelligence gathering. **Covert Channels** enable communication in air-gapped environments through physical side-channels—optical (LED/-light sensor with Manchester encoding at 10-100 bps) and acoustic (buzzer/microphone with FSK modulation at 50-200 bps). All covert traffic is encrypted with the same ChaCha8 session key, maintaining end-to-end security even across physical media.

Future work includes formal verification of the protocol state machine and evaluation on physical swarm hardware.

Acknowledgments

This work was developed as part of the NanOS open-source project.

References

[1] D.J. Bernstein, "ChaCha, a variant of Salsa20," SASC 2008.

[2] M. Rubenstein et al., "Kilobot: A low cost scalable robot system for collective behaviors," ICRA 2012.

[3] F. Mondada et al., "The e-puck, a robot designed for education in engineering," 2009.

[4] A. Bogdanov et al., "PRESENT: An ultra-lightweight block cipher," CHES 2007.

[5] R. Beaulieu et al., "The SIMON and SPECK families of lightweight block ciphers," DAC 2015.

[6] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Duke Tech Report, 2000.

[7] A. Das et al., "SWIM: Scalable weakly-consistent infection-style process group membership protocol," DSN 2002.

A Nonce Construction

```
void build_nonce(uint8_t nonce[12],
                 uint16_t node_id,
                 uint32_t counter,
                 uint32_t timestamp) {
    nonce[0] = node_id & 0xFF;
    nonce[1] = node_id >> 8;
    nonce[2] = 0; // reserved
    nonce[3] = 0;
    memcpy(&nonce[4], &counter, 4);
    memcpy(&nonce[8], &timestamp, 4);
}
```

B Bloom Filter Hash Functions

```
uint8_t bloom_hash0(uint16_t node, uint16_t seq, uint8_t type) {
    return (node * 31 + seq * 0x85EBCA6B + type * 0xC2B2AE35) % 256;
}

uint8_t bloom_hash1(uint16_t node, uint16_t seq, uint8_t type) {
    return (seq * 31 + node * 0xCC9E2D51 + type * 0x1B873593) % 256;
}
```

```
uint8_t bloom_hash2(uint16_t node, uint16_t seq, uint8_t type) {
    uint32_t h = (type | (seq << 8)) ^ node;
    h ^= h >> 16; h *= 0x85EBCA6B;
    h ^= h >> 13; h *= 0xC2B2AE35;
    return (h ^ (h >> 16)) % 256;
}
```