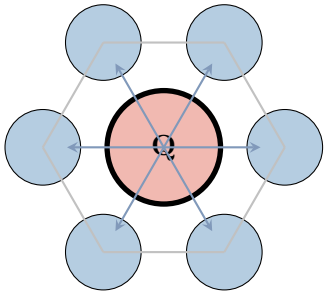


NanOS

Nano Operating System for Disposable Nodes

Manual Técnico

Versión 0.5

Incluye:

Arquitectura del Sistema
Protocolo de Feromonas
NERT: Ephemeral Reliable Transport
Enrutamiento Hebbiano (v0.5)
Stigmergia: Feromonas Digitales (v0.5)
Black Box Distribuida (v0.5)
Guía de Implementación

NanOS Project
Enero 2026

NanOS Technical Manual
Version 0.5

"Los nodos mueren, el enjambre vive."

Índice general

Índice de figuras

Índice de cuadros

Parte I

NanOS Core

Capítulo 1

Introducción

1.1. ¿Qué es NanOS?

NanOS es un sistema operativo minimalista diseñado para **nodos desechables** en entornos de computación distribuida. A diferencia de los sistemas operativos tradicionales que buscan estabilidad y persistencia, NanOS abraza la **efímera naturaleza** de sus nodos.

Filosofía de Diseño

- **Sin persistencia**: No hay disco, no hay filesystem. Todo es RAM volátil.
- **Ciclo de vida limitado**: Los nodos tienen muerte programada (apoptosis).
- **Regeneración**: Al morir, renacen con nueva identidad.
- **Inteligencia colectiva**: El comportamiento emerge del enjambre.

1.2. Inspiración Biológica

NanOS se inspira en sistemas biológicos como colonias de hormigas y organismos unicelulares. Cada nodo es análogo a una célula que:

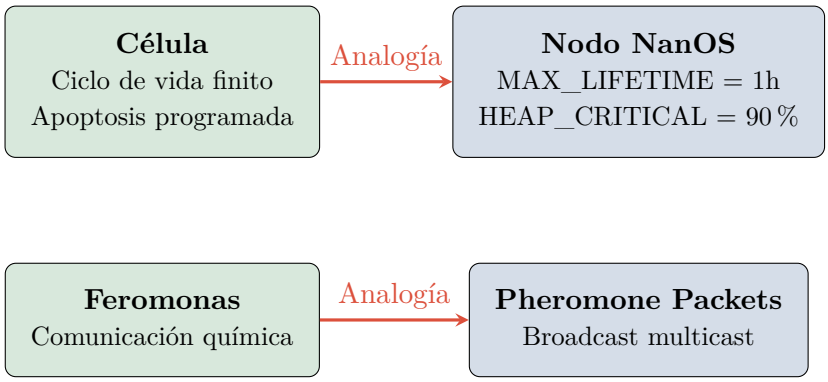


Figura 1.1: Analogía biológica de NanOS

1.3. Plataformas Soportadas

Plataforma	**Arquitectura**	**RAM Típica**	**Uso**
QEMU x86	i386/i686	128KB–1MB	Desarrollo/Testing
ARM Cortex-M3	ARMv7-M	64KB	IoT/Embedded
ESP32	Xtensa LX6	320KB	WiFi/Low-power
ARM64	ARMv8-A	1MB+	Servidores Edge

Cuadro 1.1: Plataformas soportadas por NanOS

Capítulo 2

Arquitectura del Sistema

2.1. Estructura de Capas

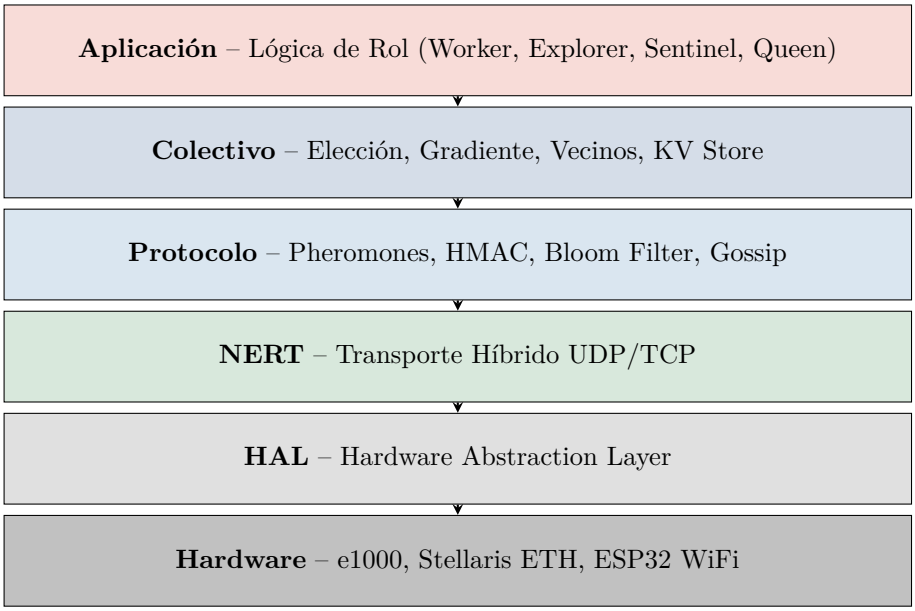


Figura 2.1: Arquitectura de capas de NanOS

2.2. Componentes del Kernel

2.2.1. Gestión de Memoria

NanOS utiliza un heap simple sin fragmentación externa mediante allocación de bloques de tamaño fijo.

```
#define HEAP_SIZE           0x10000     /* 64KB heap */
#define HEAP_CRITICAL_PCT   90          /* Muerte si > 90% */
#define BLOCK_SIZE          64          /* Bloques de 64 bytes */
```

Listing 2.1: Constantes de memoria

2.2.2. Timer y Scheduling

El kernel utiliza un timer de sistema (PIT en x86, SysTick en ARM) para:

- Mantener el contador de ticks global
- Verificar condiciones de apoptosis
- Rotar ventanas del Bloom filter
- Actualizar timeouts de conexiones

2.3. Ciclo de Vida del Nodo

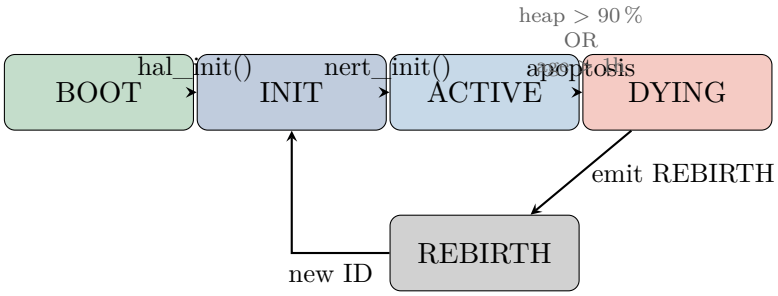


Figura 2.2: Ciclo de vida de un nodo NanOS

2.3.1. Condiciones de Apoptosis

Un nodo entra en estado DYING cuando:

1. **Memoria crítica**: Uso de heap $> 90\,\%$
2. **Antigüedad**: Tiempo de vida > `MAX_CELL_LIFETIME` (3600s)
3. **Comando externo**: Recibe `PHEROMONE_DIE` autenticado

Capítulo 3

Sistema de Roles

3.1. Roles Definidos

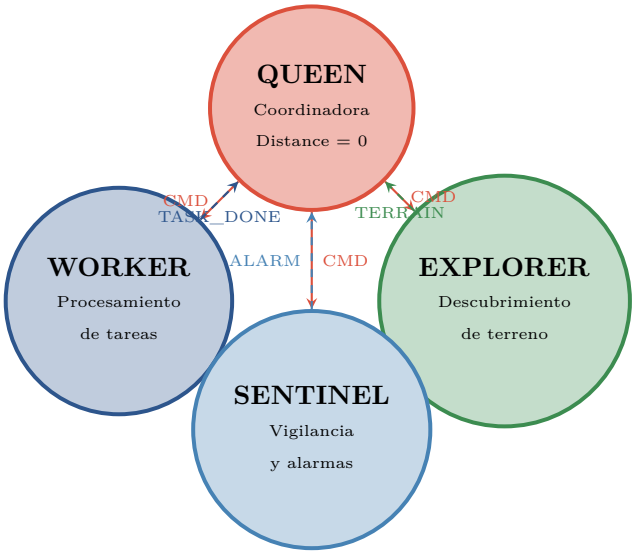


Figura 3.1: Sistema de roles en el enjambre

3.2. Transición de Roles

Los nodos pueden cambiar de rol dinámicamente basándose en:

- **Necesidades del enjambre**: Si hay déficit de exploradores, workers pueden transicionar
- **Muerte de la reina**: Se inicia proceso de elección
- **Comando de la reina**: Reasignación directa

Condición	Umbral	Acción
Sentinelas $< 10\,\%$	MIN_SENTINEL_RATIO	Worker $\rightarrow$ Sentinel
Exploradores $< 10\,\%$	MIN_EXPLORER_RATIO	Worker $\rightarrow$ Explorer
Sin reina detectada	QUEEN_TIMEOUT	Iniciar elección

Cuadro 3.1: Reglas de transición de roles

3.3. Algoritmo de Elección de Reina

El algoritmo de elección es **determinístico** y garantiza convergencia:

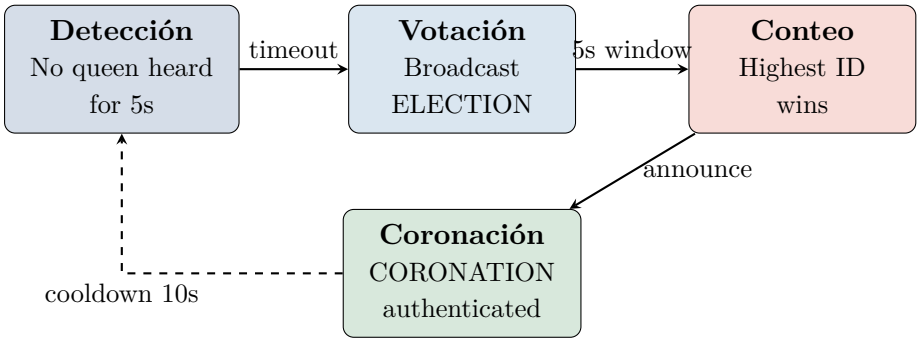


Figura 3.2: Fases del algoritmo de elección

```c
struct election_state {
    uint32_t election_id;       /* ID unico de eleccion */
    uint32_t started_at;        /* Tick de inicio */
    uint32_t my_vote;           /* A quien voto */
    uint32_t highest_vote_id;   /* Maximo ID visto */
    uint8_t  participating;     /* Participando? */
    uint8_t  phase;             /* 0=none, 1=voting, 2=counting */
};
```

Listing 3.1: Estructura de estado de elección

Capítulo 4

Protocolo de Feromonas

4.1. Tipos de Pheromones

Código	Nombre	Auth	Descripción
0x01	HELLO	No	Heartbeat con información de gradiente
0x02	DATA	No	Transporte de datos genérico
0x03	ALARM	No	Alerta de peligro detectado
0x04	ECHO	No	Respuesta/reconocimiento
0x05	ELECTION	No	Voto en elección de reina
0x06	CORONATION	**Sí**	Anuncio de nueva reina
0x10	QUEEN_CMD	**Sí**	Comando de la reina
0x20–0x22	KV_*	No	Key-Value store distribuido
0x30	TASK	No	Asignación de tarea
0x40	SENSOR	No	Lectura de sensor
0x70–0x73	MAZE_*	No	Exploración de laberinto
0x80–0x87	TERRAIN_*	No	Mapeo de terreno
0xFE	REBIRTH	**Sí**	Notificación de muerte/renacimiento
0xFF	DIE	**Sí**	Comando de terminación

Cuadro 4.1: Tipos de pheromones y sus características

4.2. Formato de Paquete

4.2.1. Paquete Estándar (64 bytes)

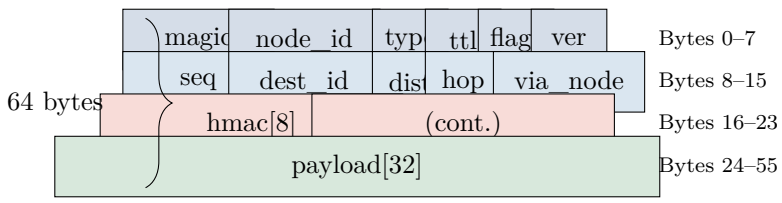


Figura 4.1: Estructura del paquete pheromone estándar

4.2.2. Paquete Compacto ARM (24 bytes)

Para plataformas con recursos limitados, se usa un formato compacto con **62 % menos overhead**:

```
1 struct nanos_pheromone_compact {
```

```
    uint8_t  magic;          /* 0xAA */
    uint16_t node_id;        /* 16-bit truncado */
    uint8_t  type;
    uint8_t  ttl_flags;      /* TTL(4) + flags(4) */
    uint8_t  seq;            /* 8-bit sequence */
    uint16_t dest_id;
    uint8_t  dist_hop;       /* distance(4) + hop(4) */
    uint8_t  payload[8];
    uint8_t  hmac[4];        /* 4-byte truncado */
    uint8_t  reserved[3];
};  /* Total: 24 bytes */
```

Listing 4.1: Estructura del paquete compacto

4.3. Autenticación HMAC

Los paquetes críticos requieren autenticación mediante un HMAC basado en SipHash simplificado:

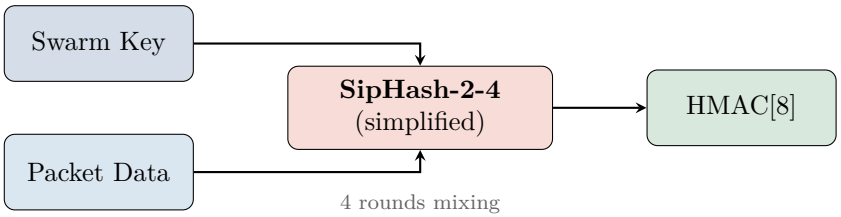


Figura 4.2: Proceso de generación de HMAC

Capítulo 5

Mecanismos de Red

5.1. Bloom Filter para Deduplicación

El Bloom filter proporciona deduplicación $O(1)$ con uso mínimo de memoria:

```
#define BLOOM_BITS       256      /* 32 bytes */
#define BLOOM_HASH_K     3        /* 3 funciones hash */
#define BLOOM_SLOTS      4        /* Ventanas rotativas */
#define BLOOM_WINDOW_MS  500      /* 500ms por ventana */
```

Listing 5.1: Configuración del Bloom filter

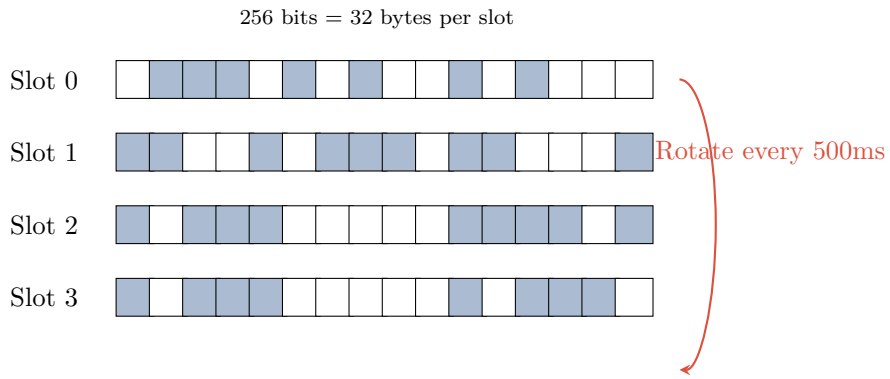


Figura 5.1: Visualización del Bloom filter con slots rotativos

5.2. Gossip Protocol

El protocolo de gossip controla la propagación de mensajes para evitar tormentas de broadcast:

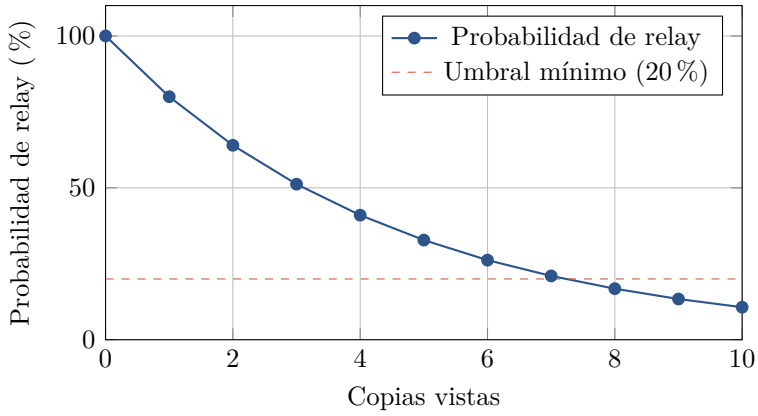


Figura 5.2: Decaimiento de probabilidad en el protocolo gossip

5.3. Gradient Routing

El enrutamiento por gradiente permite dirigir mensajes hacia la reina:

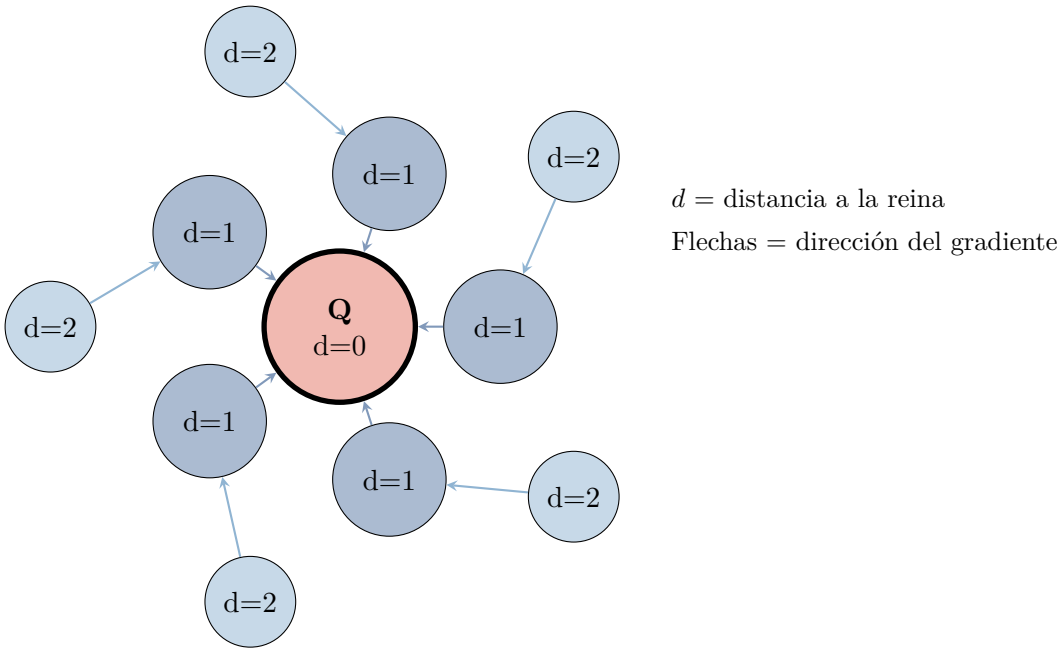


Figura 5.3: Enrutamiento por gradiente hacia la reina

5.4. Enrutamiento Hebbiano (v0.5)

La versión 0.5 introduce **Enrutamiento Hebbiano**, inspirado en la neurociencia: *"Las neuronas que se disparan juntas, se conectan juntas"*.

5.4.1. Peso Sináptico

Cada conexión con un vecino tiene un **peso sináptico** $w \in [1, 255]$:

```
struct neighbor_entry {
    uint32_t node_id;
    uint32_t last_seen;
    uint8_t  role;
```

```
    uint8_t  distance;
    uint16_t packets;
    uint8_t  synaptic_weight;  /* v0.5: 0-255, inicial: 128 */
};
```

Listing 5.2: Campos de peso sináptico

5.4.2. Regla de Aprendizaje

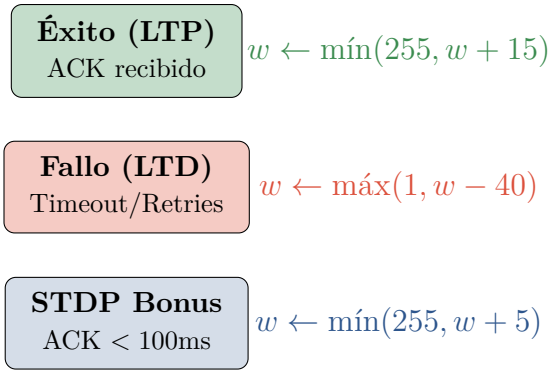


Figura 5.4: Reglas de aprendizaje Hebbiano

Asimetría Castigo/Recompensa

El castigo (-40) es casi 3 veces más severo que la recompensa ($+15$). Esto asegura que el enjambre aprenda rápidamente a evitar nodos poco confiables.

5.4.3. Fórmula de Costo Neural

El costo de ruta combina distancia y confiabilidad:

$$\text{Costo}(j) = 10 \cdot d_j + \frac{255 - w_j}{8} \tag{5.1}$$

Distancia	Peso	Costo
1 hop	255 (perfecto)	$10 + 0 = 10$
1 hop	128 (neutral)	$10 + 15 = 25$
1 hop	1 (muerto)	$10 + 31 = 41$
2 hops	255 (perfecto)	$20 + 0 = 20$

Cuadro 5.1: Ejemplos de cálculo de costo neural

Insight Clave

Una ruta de 2 saltos confiable (costo 20) es mejor que una ruta de 1 salto poco confiable (costo 41). El enjambre **aprende a rodear** nodos problemáticos.

Capítulo 6

Stigmergia: Feromonas Digitales (v0.5)

6.1. Inspiración Biológica

Las hormigas no memorizan mapas; dejan **químicos que se evaporan**. Este mecanismo de coordinación indirecta a través de modificación del ambiente se llama **stigmergia**. NanOS v0.5 implementa este concepto digitalmente.

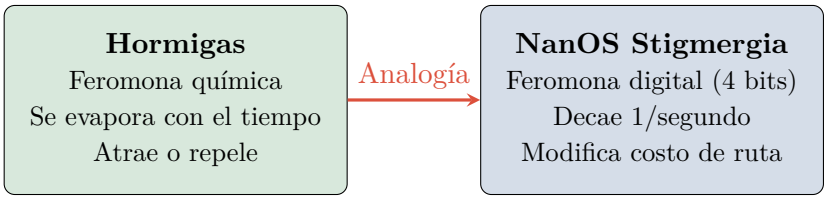


Figura 6.1: Analogía biológica de Stigmergia

6.2. Tipos de Feromonas

Código	Tipo	Efecto	Uso
0	DANGER	Repulsión (+8/nivel)	Jamming, ataques, nodos maliciosos
1	QUEEN	Atracción (-2/nivel)	Camino hacia la reina
2	RESOURCE	Neutral	Marcador de objetivos
3	AVOID	Repulsión (+4/nivel)	Zonas subóptimas (no peligrosas)

Cuadro 6.1: Tipos de feromonas digitales

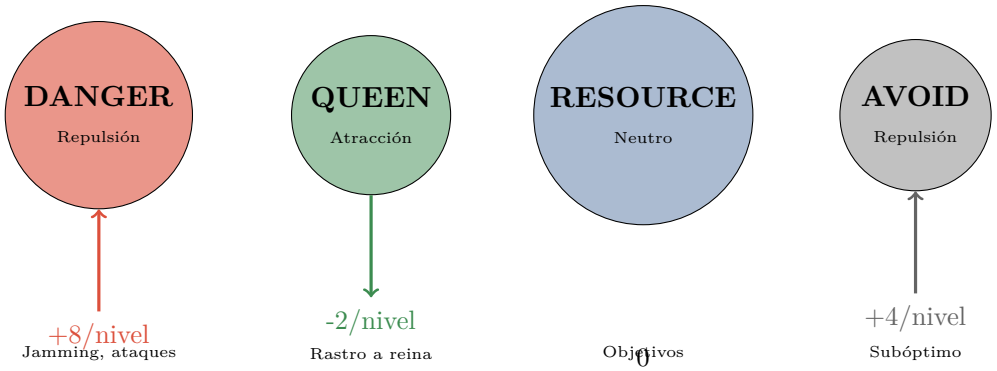


Figura 6.2: Tipos de feromonas y su efecto en el costo de movimiento

6.3. Estructura de Datos

```c
/* Cada celda almacena 4 tipos en 2 bytes (nibble packing) */
struct stigmergia_cell {
    uint8_t data[2];  /* [danger|queen], [resource|avoid] */
};

/* Grid 16x16 cubriendo terreno 32x32 (escala 2:1) */
struct stigmergia_cell pheromones[16][16];  /* 512 bytes total */

/* Acceso a intensidad (0-15) */
uint8_t stigmergia_get(uint8_t x, uint8_t y, uint8_t type);
void stigmergia_mark(uint8_t x, uint8_t y, uint8_t type, uint8_t
    intensity);
```

Listing 6.1: Almacenamiento de feromonas

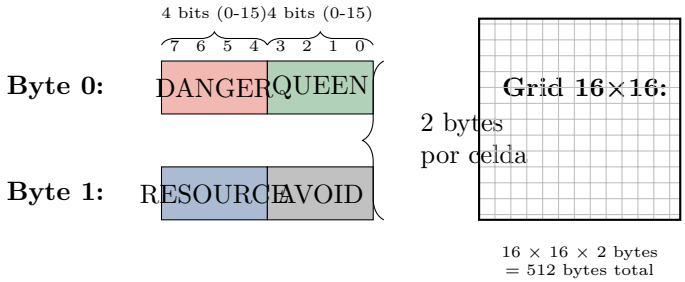


Figura 6.3: Nibble packing: 4 tipos de feromona en 2 bytes por celda

6.4. Decaimiento Temporal

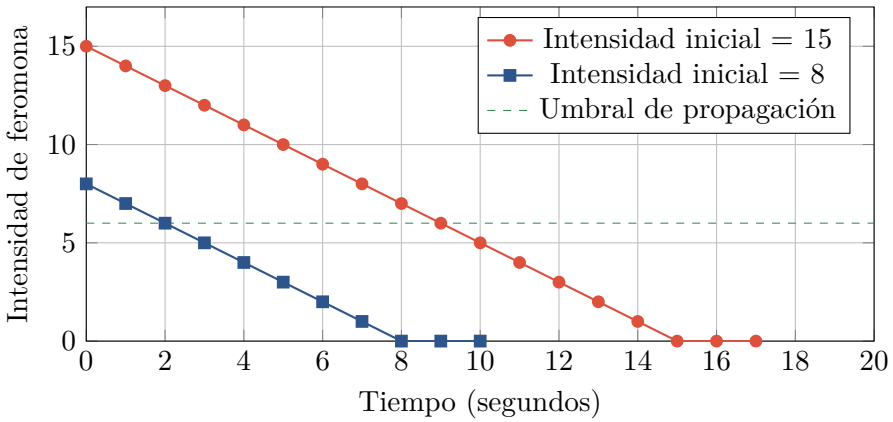


Figura 6.4: Decaimiento de feromonas: -1 cada segundo

Evaporación

La feromona **se pierde** si no es reforzada. Un peligro detectado hace 15 segundos ya no existe en el mapa. Esto evita información obsoleta.

6.5. Modificación de Costo de Movimiento

La presencia de feromonas modifica el costo de moverse a una celda:

$$\text{Costo}_{\text{total}} = \text{Costo}_{\text{base}} + 8 \cdot I_{\text{DANGER}} + 4 \cdot I_{\text{AVOID}} - 2 \cdot I_{\text{QUEEN}} \tag{6.1}$$

Situación	Feromonas	Modificador	Costo Final
Celda normal	Ninguna	0	10
Zona peligrosa	DANGER=15	+120	130
Cerca de reina	QUEEN=10	-20	-10 (atracción)
Zona subóptima	AVOID=8	+32	42
Mixta	DANGER=5, QUEEN=3	+40-6=+34	44

Cuadro 6.2: Ejemplos de modificación de costo

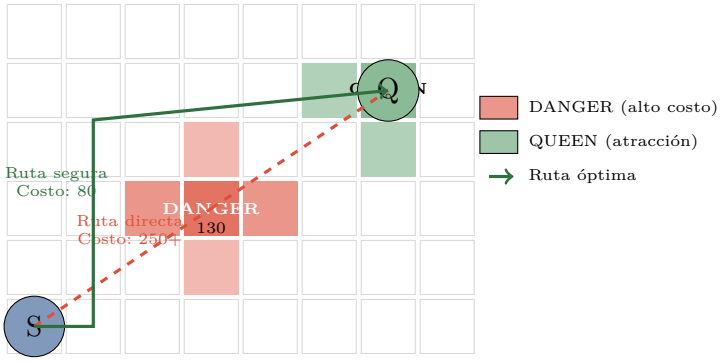


Figura 6.5: Efecto de feromonas en pathfinding: el enjambre rodea zonas peligrosas

6.6. Propagación de Feromonas

Las feromonas de alta intensidad se propagan a celdas vecinas:

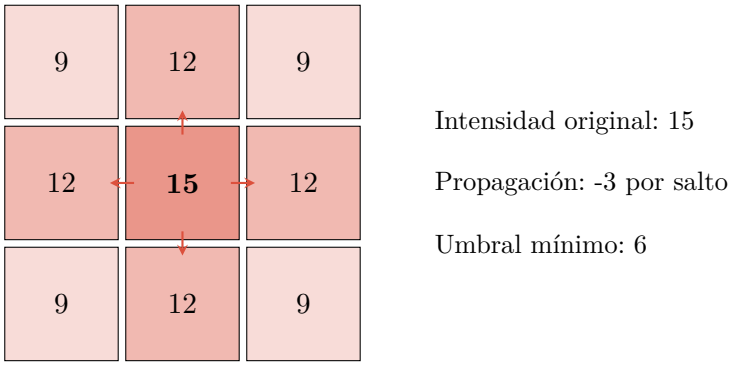


Figura 6.6: Propagación de feromona DANGER

6.7. API de Stigmergia

```c
/* Inicializar sistema (limpia todas las feromonas) */
void stigmergia_init(void);

/* Marcar feromona en coordenadas de terreno */
void stigmergia_mark(uint8_t terrain_x, uint8_t terrain_y,
                     uint8_t type, uint8_t intensity);

/* Obtener intensidad en coordenadas */
uint8_t stigmergia_get(uint8_t terrain_x, uint8_t terrain_y,
                       uint8_t type);

/* Aplicar decaimiento (llamar cada segundo) */
void stigmergia_decay(void);

/* Calcular modificador de costo para pathfinding */
int8_t stigmergia_cost_modifier(uint8_t terrain_x, uint8_t terrain_y);

/* Emitir feromona de peligro (convenience) */
void stigmergia_emit_danger(uint8_t intensity);

/* Emitir rastro hacia la reina */
void stigmergia_emit_queen_trail(void);
```

Listing 6.2: Funciones principales de Stigmergia

Capítulo 7

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

7.1. Problema: Evidencia Forense Perdida

Cuando un nodo es comprometido y terminado, su evidencia forense se pierde. Un atacante inteligente podría:

1. Comprometer un nodo
2. Extraer información sensible
3. Forzar su "suicidio" para eliminar rastros

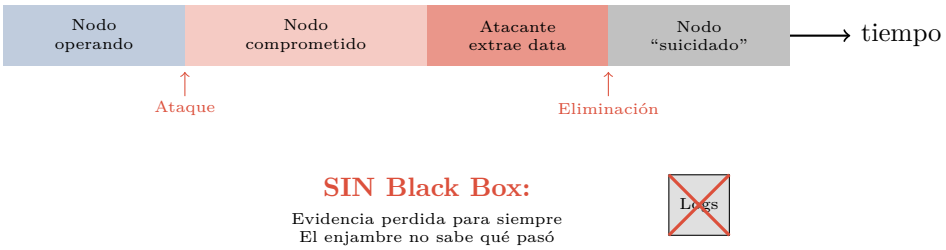


Figura 7.1: Escenario de ataque sin Black Box: la evidencia se pierde al morir el nodo

Escenario de Ataque

Sin Black Box: El atacante elimina el nodo y toda evidencia de compromiso desaparece. El enjambre no sabe qué pasó.

7.2. Solución: El Último Aliento

Antes de morir, cada nodo transmite un "testamento" (Last Will) a vecinos de confianza. Incluso si el nodo fue hackeado y suicidado, su evidencia sobrevive en el enjambre.

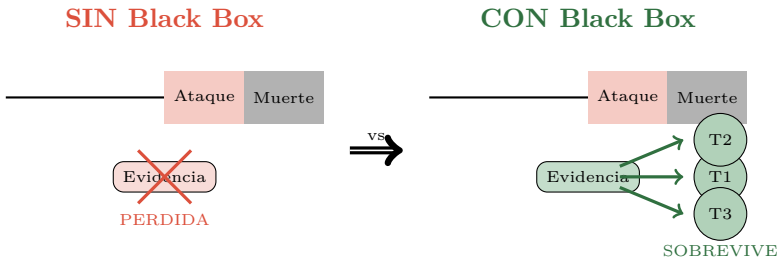


Figura 7.2: Comparación: sin Black Box la evidencia se pierde, con Black Box sobrevive

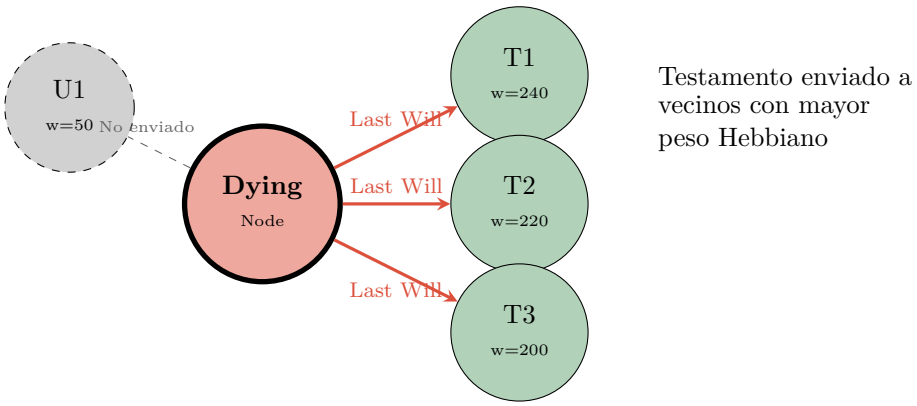


Figura 7.3: Transmisión del Último Aliento a vecinos de confianza

7.3. Contenido del Testamento

```c
/* Razones de muerte */
#define DEATH_NATURAL           0x00  /* Vejez/timeout normal */
#define DEATH_HEAP_EXHAUSTED    0x01  /* Sin memoria */
#define DEATH_CORRUPTION        0x02  /* Corrupcion detectada */
#define DEATH_ATTACK_DETECTED   0x03  /* Ataque en progreso */
#define DEATH_QUEEN_ORDER       0x04  /* Orden de la reina */
#define DEATH_ISOLATION         0x05  /* Sin contacto con enjambre */

/* Tipos de eventos de seguridad */
#define EVENT_BAD_MAC           0x01  /* MAC invalido recibido */
#define EVENT_REPLAY            0x02  /* Intento de replay */
#define EVENT_RATE_LIMIT        0x03  /* Rate limit excedido */
#define EVENT_BLACKLIST         0x04  /* Nodo en blacklist */
#define EVENT_JAMMING           0x05  /* Jamming detectado */
#define EVENT_CORRUPTION        0x06  /* Corrupcion de memoria */

/* Contenido del testamento */
struct last_will {
    uint32_t node_id;           /* ID del nodo que muere */
    uint8_t  death_reason;      /* DEATH_* codigo */
    uint8_t  uptime_hours;      /* Horas de vida */
    uint16_t bad_mac_count;     /* MACs invalidos recibidos */
    uint16_t replay_count;      /* Intentos de replay */
    uint16_t rate_limit_hits;   /* Veces rate limited */

    /* Ultimos 8 eventos de seguridad */
    struct {
```

```
        uint8_t  type;          /* EVENT_* tipo */
        uint16_t source_node;   /* Nodo relacionado */
        uint32_t timestamp;     /* Cuando ocurrio */
    } events[8];
};
```

Listing 7.1: Estructura del Last Will

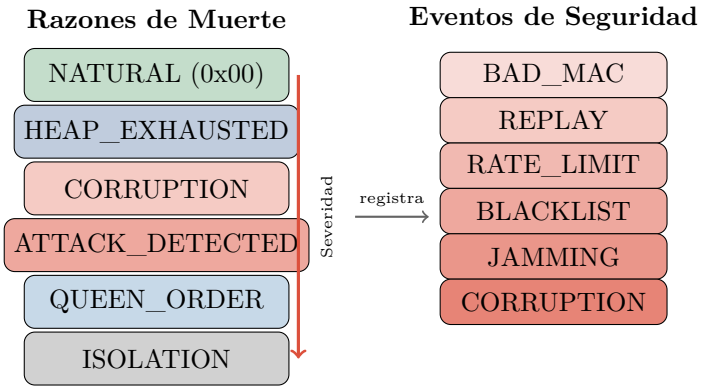


Figura 7.4: Códigos de muerte y eventos de seguridad registrados en el testamento

7.4. Flujo de Operación

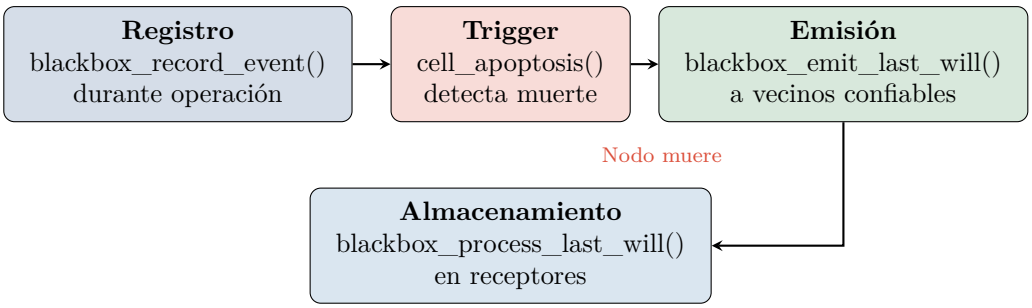


Figura 7.5: Flujo del sistema Black Box

7.5. Almacenamiento de Testamentos

Cada nodo almacena hasta 8 testamentos recibidos:

```
#define BLACKBOX_MAX_WILLS  8

/* Estructura de almacenamiento */
struct blackbox_storage {
    struct {
        uint32_t node_id;
        uint8_t  death_reason;
        uint8_t  uptime_hours;
        uint16_t bad_mac_count;
        uint16_t replay_count;
        uint8_t  priority;      /* Para relay y reemplazo */
        uint8_t  event_count;
    } wills[BLACKBOX_MAX_WILLS];

```

```
    uint8_t count;              /* Testamentos almacenados */
};
```

Listing 7.2: Almacenamiento de testamentos

7.6. API de Black Box

```
/* Inicializar sistema */
void blackbox_init(void);

/* Registrar evento de seguridad (durante operacion) */
void blackbox_record_event(uint8_t event_type, uint16_t source_node);

/* Emitir testamento antes de morir */
void blackbox_emit_last_will(uint8_t death_reason);

/* Procesar testamento recibido */
void blackbox_process_last_will(struct nanos_pheromone* pkt);

/* Consultar muerte de un nodo */
int blackbox_query_death(uint32_t node_id,
                         uint8_t *death_reason,
                         uint16_t *bad_mac_count,
                         uint8_t *uptime_hours);

/* Obtener numero de testamentos almacenados */
uint8_t blackbox_get_will_count(void);

/* Propagar testamentos criticos a vecinos */
void blackbox_relay_critical(void);

/* Imprimir resumen forense (debug) */
void blackbox_print_summary(void);
```

Listing 7.3: Funciones principales de Black Box

7.7. Ejemplo de Investigación Forense

```
void investigate_dead_node(uint32_t suspect_id) {
    uint8_t death_reason;
    uint16_t bad_mac;
    uint8_t uptime;

    if (blackbox_query_death(suspect_id, &death_reason,
                             &bad_mac, &uptime) == 0) {
        printf("Node 0x%08x death investigation:\n", suspect_id);
        printf("  Uptime: %d hours\n", uptime);

        switch (death_reason) {
            case DEATH_ATTACK_DETECTED:
                printf("  ALERT: Died under attack!\n");
                printf("  Bad MACs received: %d\n", bad_mac);
                break;
            case DEATH_CORRUPTION:
                printf("  Memory corruption detected\n");
```

```
                break;
            case DEATH_NATURAL:
                printf("  Normal lifecycle end\n");
                break;
        }
    } else {
        printf("No forensic data for node 0x%08x\n", suspect_id);
    }
}
```

Listing 7.4: Consulta forense de nodo muerto

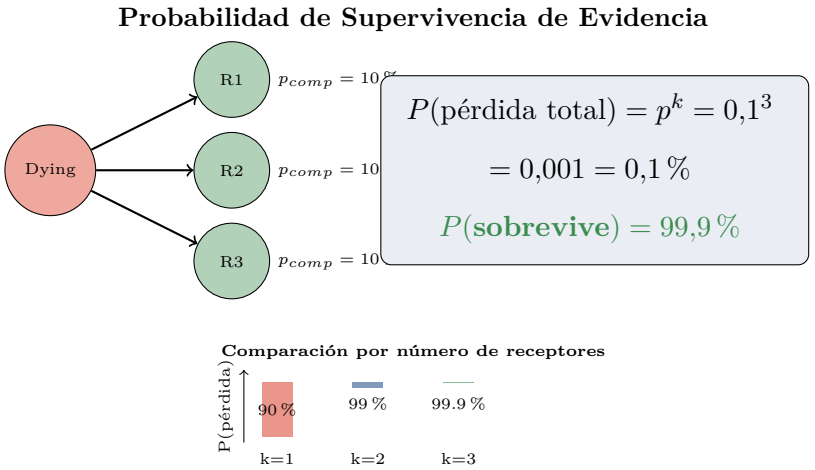


Figura 7.6: Supervivencia de evidencia: con 3 receptores, la probabilidad de pérdida es solo 0.1 %

Supervivencia de Evidencia

Con 3 receptores de confianza y probabilidad de compromiso 10 % por nodo, la evidencia sobrevive con probabilidad $1 - 0{,}1^3 = 99{,}9\,\%$. “Los muertos hablan a través de los vivos.”

Parte II

Protocolo NERT

Capítulo 8

Visión General de NERT

8.1. Motivación

Los protocolos tradicionales no satisfacen las necesidades de nodos desechables:

Característica	**TCP**	**UDP**	**NERT**
Confiabilidad	Siempre	Nunca	**Selectiva**
Encriptación	Opcional	No	**Obligatoria**
Handshake	3-way	Ninguno	**2-way**
FEC	No	No	**Opcional**
Multi-path	No	No	**Sí**
RAM/conexión	∼2KB	∼0	**∼100B**
Optimizado para efímeros	No	Parcial	**Sí**

Cuadro 8.1: Comparación de NERT con protocolos tradicionales

8.2. Clases de Confiabilidad

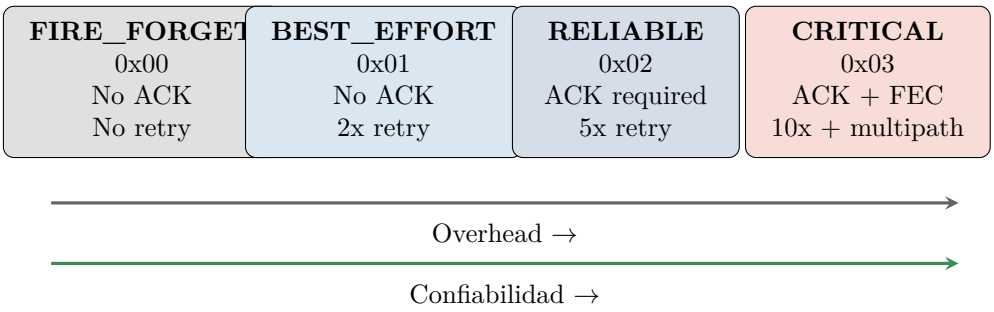


Figura 8.1: Clases de confiabilidad de NERT

Capítulo 9

Formato de Paquete NERT

9.1. Header Estándar (20 bytes)

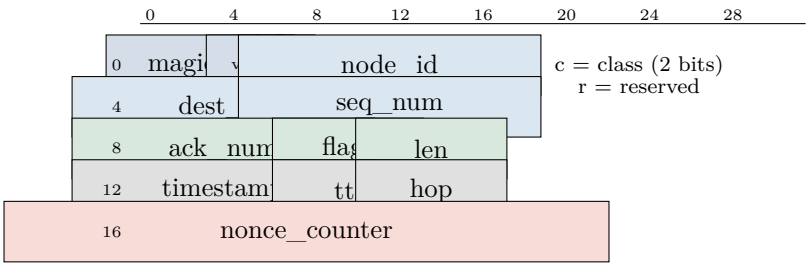


Figura 9.1: Header NERT estándar de 20 bytes

9.2. Campo de Flags

Bit	Nombre	Descripción
0	SYN	Inicio de conexión
1	ACK	Acknowledgment válido
2	FIN	Fin de conexión
3	RST	Reset/abort
4	ENC	Payload encriptado (siempre 1)
5	FEC	Incluye bloque FEC
6	FRAG	Paquete fragmentado
7	MPATH	Multi-path habilitado

Cuadro 9.1: Definición de bits del campo flags

9.3. Estructura Completa del Paquete

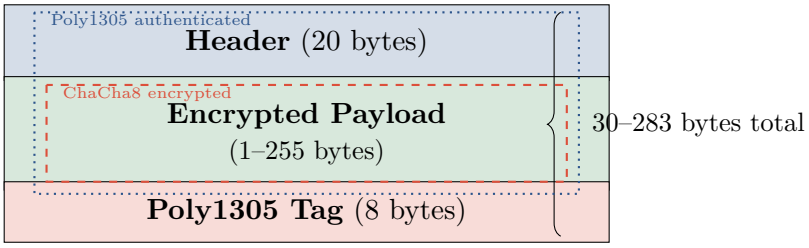


Figura 9.2: Estructura completa del paquete NERT

Capítulo 10

Criptografía

10.1. Algoritmos Utilizados

Función	Algoritmo	Parámetros
Encriptación	ChaCha8	256-bit key, 96-bit nonce
Autenticación	Poly1305	256-bit key, truncated to 64-bit
Key Derivation	ChaCha8-based PRF	Master key + epoch

Cuadro 10.1: Algoritmos criptográficos de NERT

10.2. Derivación de Claves

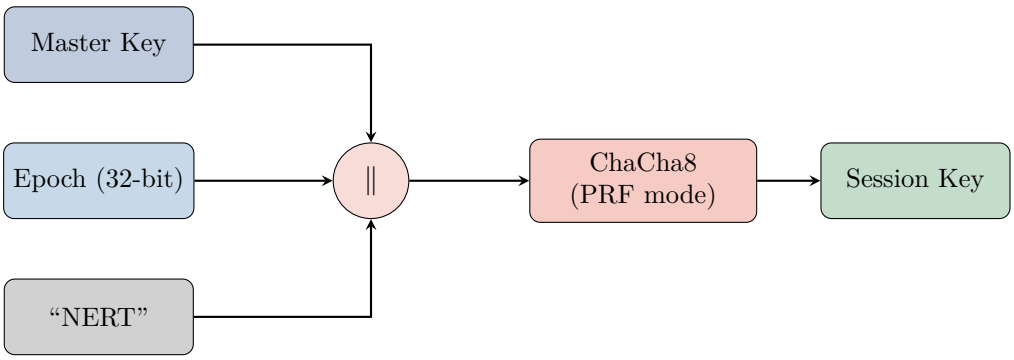
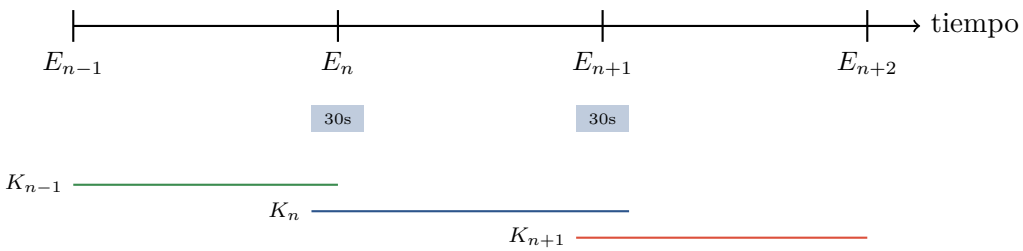


Figura 10.1: Proceso de derivación de clave de sesión

10.3. Ventana de Gracia para Rotación de Claves



Ventana de gracia: acepta claves de épocas adyacentes

Figura 10.2: Mecanismo de ventana de gracia para rotación de claves

Problema de Sincronización

Sin sincronización de tiempo, nodos con relojes desincronizados podrían rechazar paquetes válidos en el cambio de época. La ventana de gracia de 30 segundos permite tolerancia a drift de hasta ±30s entre nodos.

10.4. Construcción del Nonce

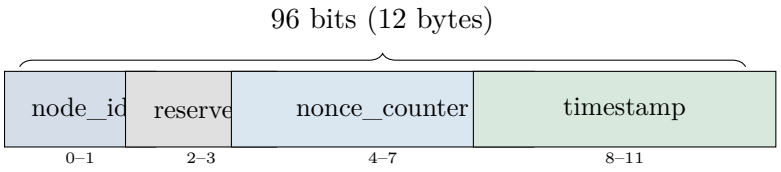


Figura 10.3: Estructura del nonce de 96 bits

Capítulo 11

Mecanismos de Confiabilidad

11.1. Máquina de Estados de Conexión

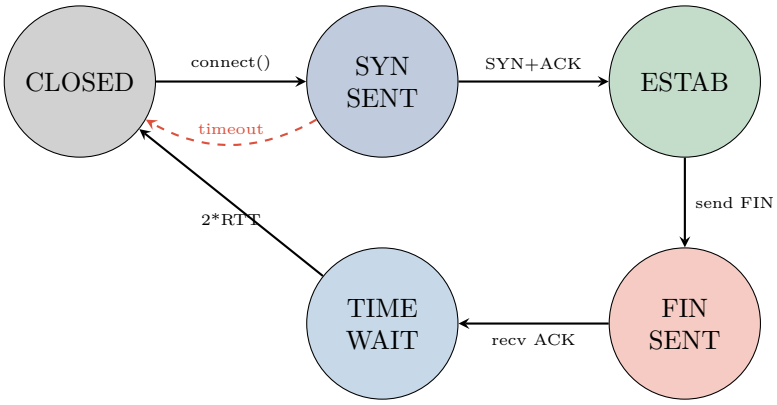


Figura 11.1: Máquina de estados de conexión NERT (simplificada)

11.2. Two-Way Handshake

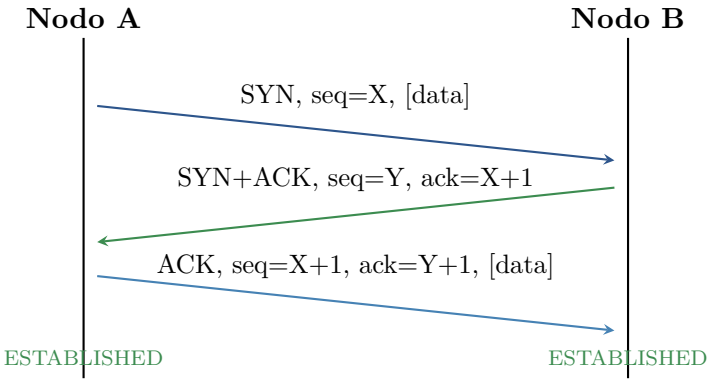


Figura 11.2: Handshake de dos vías de NERT

11.3. Selective ACK (SACK)

El SACK permite indicar eficientemente qué paquetes se han recibido:

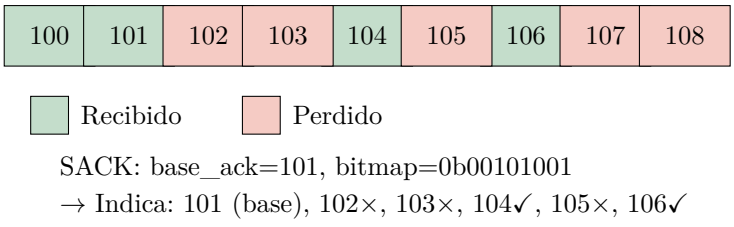


Figura 11.3: Ejemplo de Selective ACK

11.4. Retransmisión con Backoff Exponencial

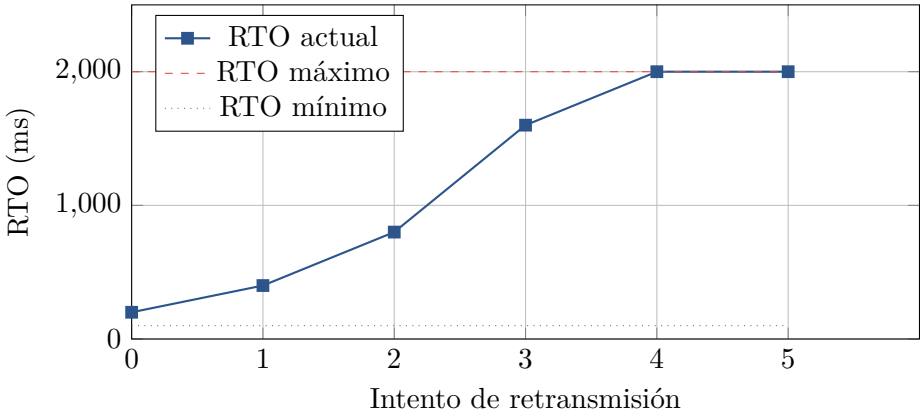


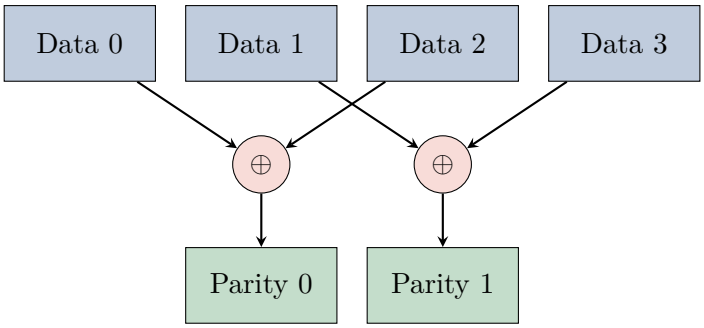
Figura 11.4: Crecimiento del RTO con backoff exponencial

Capítulo 12

Forward Error Correction

12.1. Esquema XOR Parity

Para la clase CRITICAL, NERT utiliza FEC basado en XOR:



$P_0 = D_0 \oplus D_2$

$P_1 = D_1 \oplus D_3$

Recuperación: $D_0 = P_0 \oplus D_2$, $D_2 = P_0 \oplus D_0$, etc.

Figura 12.1: Esquema FEC con paridad XOR

12.2. Capacidad de Recuperación

Pérdida	**Recuperable?**
1 data shard cualquiera	✓ Sí
2 data shards (grupos diferentes)	✓ Sí
2 data shards (mismo grupo)	× No
3+ data shards	× No

Cuadro 12.1: Capacidad de recuperación del FEC

Capítulo 13

Multi-Path Transmission

13.1. Selección de Rutas

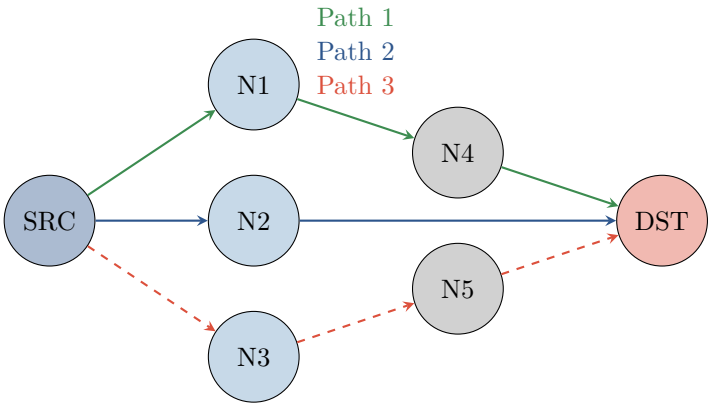


Figura 13.1: Transmisión multi-path con rutas diversas

Parte III

Guía de Implementación

Capítulo 14

Compilación y Despliegue

14.1. Requisitos

Componente	Versión Mínima
GCC (x86)	9.0
ARM GCC	arm-none-eabi-gcc 10.0
QEMU	6.0
PlatformIO	6.0 (para ESP32)
Make	4.0

Cuadro 14.1: Requisitos de compilación

14.2. Compilación x86

```
# Compilar kernel
make clean
make PLATFORM=x86

# Ejecutar en QEMU (3 nodos)
./scripts/run_swarm.sh 3
```

Listing 14.1: Compilación para QEMU x86

14.3. Compilación ARM

```
# Compilar para Stellaris
make PLATFORM=arm

# Ejecutar en QEMU
qemu-system-arm -M lm3s6965evb \
    -kernel build/nanos_arm.elf \
    -nographic
```

Listing 14.2: Compilación para ARM Cortex-M3

14.4. Compilación ESP32

```
cd platforms/esp32
pio run -e esp32dev

# Flash al dispositivo
pio run -t upload
```

Listing 14.3: Compilación con PlatformIO

Capítulo 15

API de Programación

15.1. Inicialización

```c
#include "nert.h"
#include "nanos.h"

void handle_pheromone(uint16_t sender, uint8_t type,
                      const void *data, uint8_t len) {
    switch (type) {
        case PHEROMONE_HELLO:
            neighbor_update(sender, data);
            break;
        case PHEROMONE_QUEEN_CMD:
            execute_command(data);
            break;
        // ... otros tipos
    }
}

void kernel_main(void) {
    hal_init();
    nert_init();
    nert_set_receive_callback(handle_pheromone);

    while (1) {
        nert_process_incoming();
        nert_timer_tick();
        nert_check_key_rotation();

        // Logica de aplicacion
        process_role_logic();

        hal_cpu_idle();
    }
}
```

Listing 15.1: Inicialización típica de NERT

15.2. Envío de Mensajes

```c
// Telemetria frecuente (no garantizada)
struct sensor_data sensor = {.temp = 25, .humidity = 60};
```

```c
nert_send_unreliable(0, PHEROMONE_SENSOR, &sensor, sizeof(sensor));

// Datos importantes (con retry)
struct kv_pair kv = {.key = "status", .value = "active"};
nert_send_best_effort(dest_id, PHEROMONE_KV_SET, &kv, sizeof(kv));

// Comando que requiere ACK
struct task_cmd task = {.action = ACTION_EXPLORE, .area = 5};
int conn = nert_send_reliable(worker_id, PHEROMONE_TASK,
                              &task, sizeof(task));

// Comando critico (FEC + multipath)
struct die_cmd die = {.target = rogue_node, .reason = SECURITY};
nert_send_critical(target_id, PHEROMONE_DIE, &die, sizeof(die));
```

Listing 15.2: Ejemplos de envío con diferentes clases

Capítulo 16

Debugging y Monitoreo

16.1. Estadísticas de NERT

```c
const struct nert_stats *stats = nert_get_stats();

printf("TX: %lu packets, %lu bytes\n",
       stats->tx_packets, stats->tx_bytes);
printf("RX: %lu packets, %lu duplicates\n",
       stats->rx_packets, stats->rx_duplicates);
printf("Retransmits: %lu\n", stats->tx_retransmits);
printf("Bad MAC: %lu, Replay blocked: %lu\n",
       stats->rx_bad_mac, stats->rx_replay_blocked);
printf("RTT: avg=%u, min=%u, max=%u\n",
       stats->avg_rtt, stats->min_rtt, stats->max_rtt);
```

Listing 16.1: Acceso a estadísticas de protocolo

16.2. Métricas Clave

Métrica	Valor Normal	Alerta Si
Duplicates/RX	$< 5\,\%$	$> 20\,\%$
Bad MAC rate	$< 0.1\,\%$	$> 1\,\%$
Retransmit rate	$< 10\,\%$	$> 30\,\%$
Avg RTT	$<$ 100ms	$>$ 500ms
Connection timeouts	0	$>$ 5/min

Cuadro 16.1: Umbrales de métricas para monitoreo

Apéndice A

Constantes de Configuración

Constante	Valor	Descripción
NERT_MAGIC	0x4E	Byte mágico ('N')
NERT_VERSION	0x10	Versión del protocolo
NERT_KEY_SIZE	32	Tamaño de clave en bytes
NERT_NONCE_SIZE	12	Tamaño de nonce en bytes
NERT_MAC_SIZE	8	Tamaño de tag MAC
NERT_MAX_CONNECTIONS	4–8	Conexiones simultáneas
NERT_WINDOW_SIZE	2–4	Tamaño de ventana TX
NERT_RETRY_TIMEOUT_MS	200	Timeout inicial
NERT_MAX_RETRIES	5	Reintentos (RELIABLE)
NERT_MAX_RETRIES_CRITICAL	10	Reintentos (CRITICAL)
NERT_KEY_ROTATION_SEC	3600	Rotación de clave (1h)
NERT_KEY_GRACE_WINDOW_MS	30000	Ventana de gracia (30s)
BLOOM_BITS	256	Bits del Bloom filter
BLOOM_SLOTS	4	Slots rotativos
BLOOM_WINDOW_MS	500	Ventana por slot
MAX_CELL_LIFETIME	3600000	Vida máxima del nodo (ms)
HEAP_CRITICAL_PCT	90	Umbral crítico de heap
Enrutamiento Hebbiano (v0.5)		
SYNAPSE_WEIGHT_MIN	1	Peso sináptico mínimo
SYNAPSE_WEIGHT_MAX	255	Peso sináptico máximo
SYNAPSE_WEIGHT_INITIAL	128	Peso inicial (neutral)
SYNAPSE_WEIGHT_THRESHOLD	32	Umbral para ruta saludable
SYNAPSE_LTP_INCREMENT	15	Recompensa por éxito (LTP)
SYNAPSE_LTD_DECREMENT	40	Castigo por fallo (LTD)
SYNAPSE_STDP_BONUS	5	Bonus por respuesta rápida
SYNAPSE_STDP_WINDOW_MS	100	Ventana STDP (ms)
SYNAPSE_DECAY_INTERVAL_MS	5000	Intervalo de decaimiento
SYNAPSE_DECAY_AMOUNT	1	Decremento por intervalo
Stigmergia (v0.5)		
STIGMERGIA_DANGER	0	Tipo: zona de peligro
STIGMERGIA_QUEEN	1	Tipo: camino a reina
STIGMERGIA_RESOURCE	2	Tipo: marcador de recurso
STIGMERGIA_AVOID	3	Tipo: zona a evitar

Continúa en la siguiente página...

Constante	Valor	Descripción
STIGMERGIA_INTENSITY_MAX	15	Intensidad máxima (4 bits)
STIGMERGIA_DECAY_INTERVAL_MS	1000	Decaimiento cada 1s
STIGMERGIA_DECAY_AMOUNT	1	Resta 1 por intervalo
STIGMERGIA_PROPAGATE_THRESHOLD	6	Min intensidad para propagar
STIGMERGIA_PROPAGATE_DECAY	3	Decremento al propagar
STIGMERGIA_DANGER_COST_MULT	8	Multiplicador costo DANGER
STIGMERGIA_AVOID_COST_MULT	4	Multiplicador costo AVOID
STIGMERGIA_QUEEN_COST_BONUS	2	Bonus atracción QUEEN
STIGMERGIA_SIZE	16	Grid 16x16 (escala 2:1)
Black Box (v0.5)		
BLACKBOX_MAX_WILLS	8	Testamentos almacenados
DEATH_NATURAL	0x00	Muerte natural (timeout)
DEATH_HEAP_EXHAUSTED	0x01	Sin memoria
DEATH_CORRUPTION	0x02	Corrupción detectada
DEATH_ATTACK_DETECTED	0x03	Bajo ataque
DEATH_QUEEN_ORDER	0x04	Orden de la reina
DEATH_ISOLATION	0x05	Sin contacto
EVENT_BAD_MAC	0x01	MAC inválido
EVENT_REPLAY	0x02	Intento de replay
EVENT_RATE_LIMIT	0x03	Rate limit excedido
EVENT_BLACKLIST	0x04	Nodo en blacklist
EVENT_JAMMING	0x05	Jamming detectado
EVENT_CORRUPTION	0x06	Corrupción memoria

Cuadro A.1: Constantes de configuración de NanOS/NERT

Apéndice B

Glosario

Apoptosis
: Muerte celular programada. En NanOS, el proceso por el cual un nodo termina su ejecución de forma controlada.

Black Box (v0.5)
: Sistema de preservación de evidencia forense. Los nodos emiten un "testamento" antes de morir que sobrevive en vecinos de confianza.

Bloom Filter
: Estructura de datos probabilística para pruebas de membresía con eficiencia de espacio.

Época (Epoch)
: Período de tiempo durante el cual una clave de sesión es válida (por defecto: 1 hora).

FEC
: Forward Error Correction. Técnica para recuperar datos perdidos sin retransmisión.

Gradiente
: Campo escalar que indica la distancia a la reina, usado para enrutamiento.

Gossip Protocol
: Protocolo de diseminación epidémica de información.

Grace Window
: Período en límites de época donde se aceptan claves de épocas adyacentes.

HMAC
: Hash-based Message Authentication Code.

LTD
: Long-Term Depression. Debilitamiento de conexiones sinápticas tras eventos negativos (timeouts, fallos de entrega).

LTP
: Long-Term Potentiation. Fortalecimiento de conexiones sinápticas tras eventos positivos (ACKs exitosos).

Nodo Desechable
: Nodo diseñado con vida útil limitada que regenera al morir.

NERT
: NanOS Ephemeral Reliable Transport.

Pheromone
Mensaje de comunicación entre nodos del enjambre.

SACK
Selective Acknowledgment.

STDP
Spike-Timing Dependent Plasticity. Mecanismo donde el tiempo de respuesta afecta el aprendizaje (respuestas rápidas refuerzan más).

Stigmergia (v0.5)
Coordinación indirecta a través de modificación del ambiente. En NanOS, feromonas digitales que decaen con el tiempo y modifican costos de movimiento. "Las hormigas no memorizan mapas."

Swarm
Conjunto de nodos comunicándose colectivamente.

Synaptic Weight
Peso sináptico. Valor [1-255] que representa la confiabilidad aprendida de una conexión con un vecino.

Sobre Este Documento

Este manual técnico fue generado como parte del proyecto NanOS v0.5. Cubre la arquitectura del sistema operativo, el protocolo NERT diseñado específicamente para comunicación entre nodos desechables, y el sistema de enrutamiento Hebbiano inspirado en neurociencia.

NanOS Project
Los nodos mueren, el enjambre vive.
2026