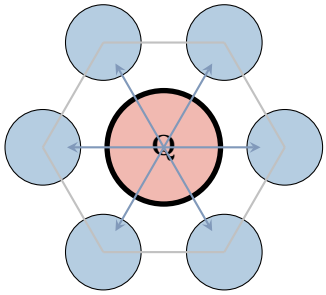


NanOS

Nano Operating System for Disposable Nodes

Manual Técnico

Versión 0.3

Incluye:

Arquitectura del Sistema
Protocolo de Feromonas
NERT: Ephemeral Reliable Transport
Guía de Implementación

NanOS Project
Enero 2026

NanOS Technical Manual
Version 0.3

"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:

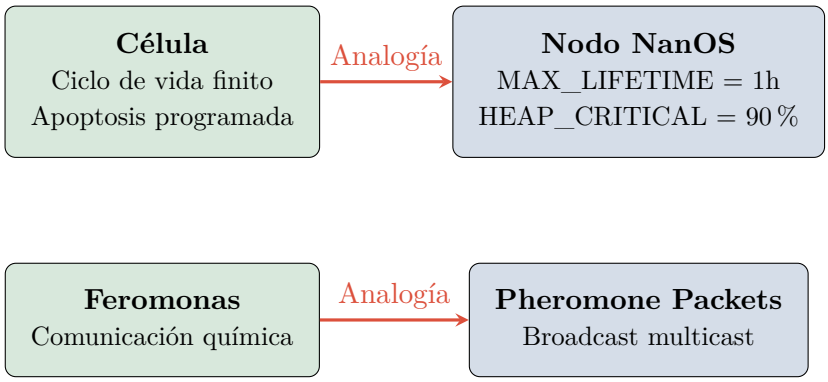


Figura 1.1: Analogía biológica de NanOS

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

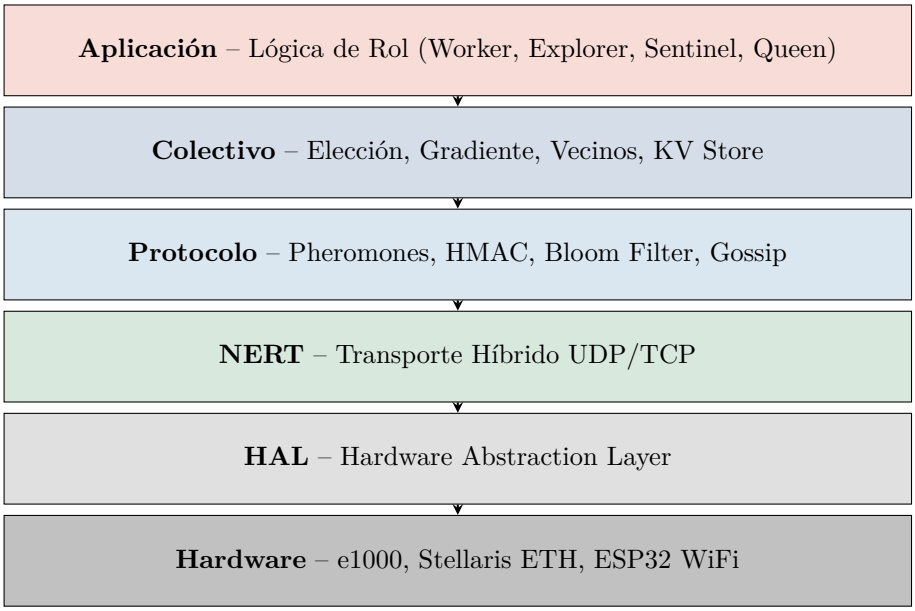


Figura 2.1: Arquitectura de capas de NanOS

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

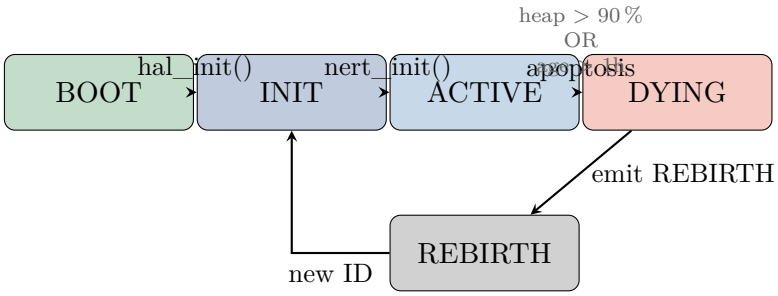


Figura 2.2: Ciclo de vida de un nodo NanOS

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

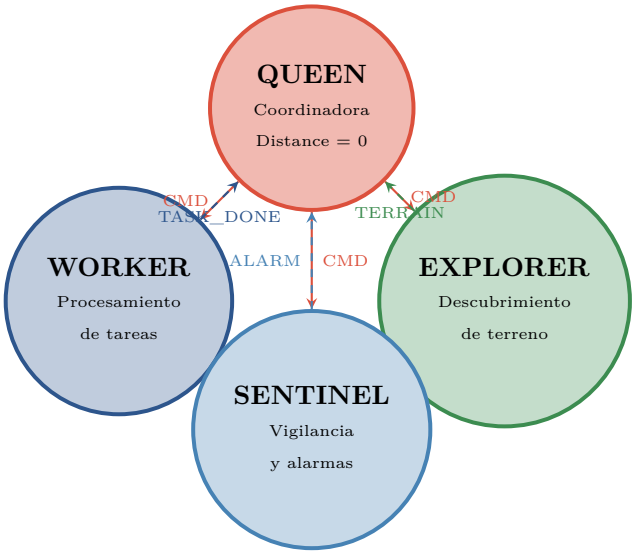


Figura 3.1: Sistema de roles en el enjambre

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:

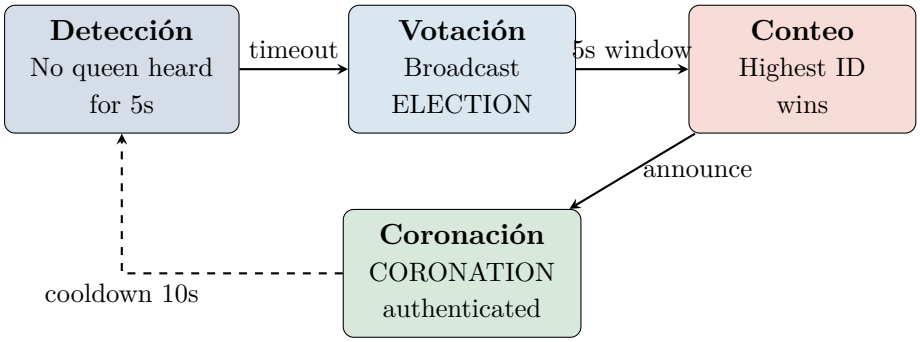


Figura 3.2: Fases del algoritmo de elección

```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)

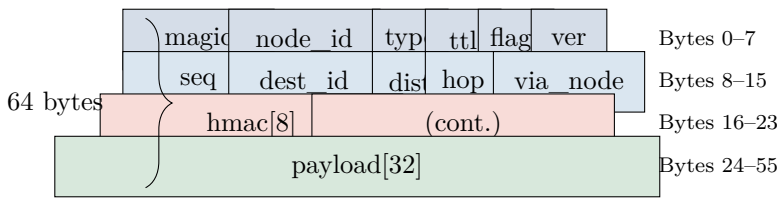


Figura 4.1: Estructura del paquete pheromone estándar

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:

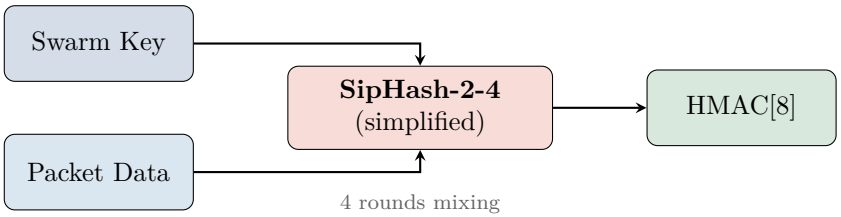


Figura 4.2: Proceso de generación de HMAC

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

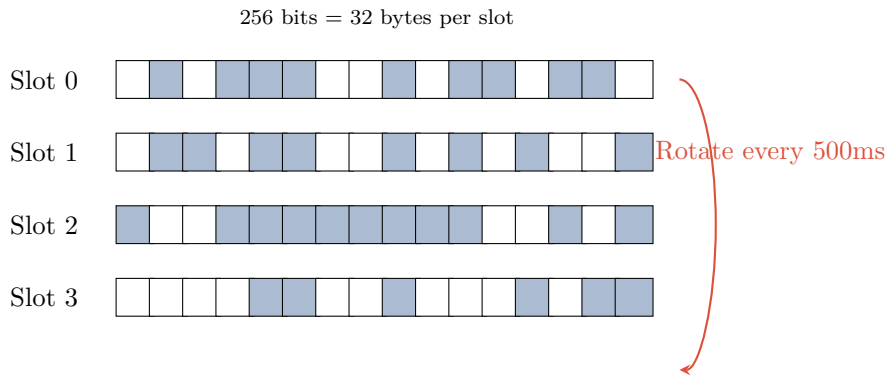


Figura 5.1: Visualización del Bloom filter con slots rotativos

5.2. Gossip Protocol

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

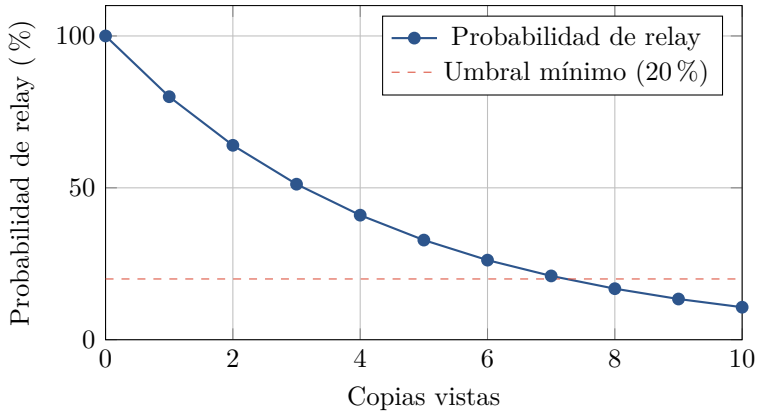


Figura 5.2: Decaimiento de probabilidad en el protocolo gossip

5.3. Gradient Routing

El enrutamiento por gradiente permite dirigir mensajes hacia la reina:

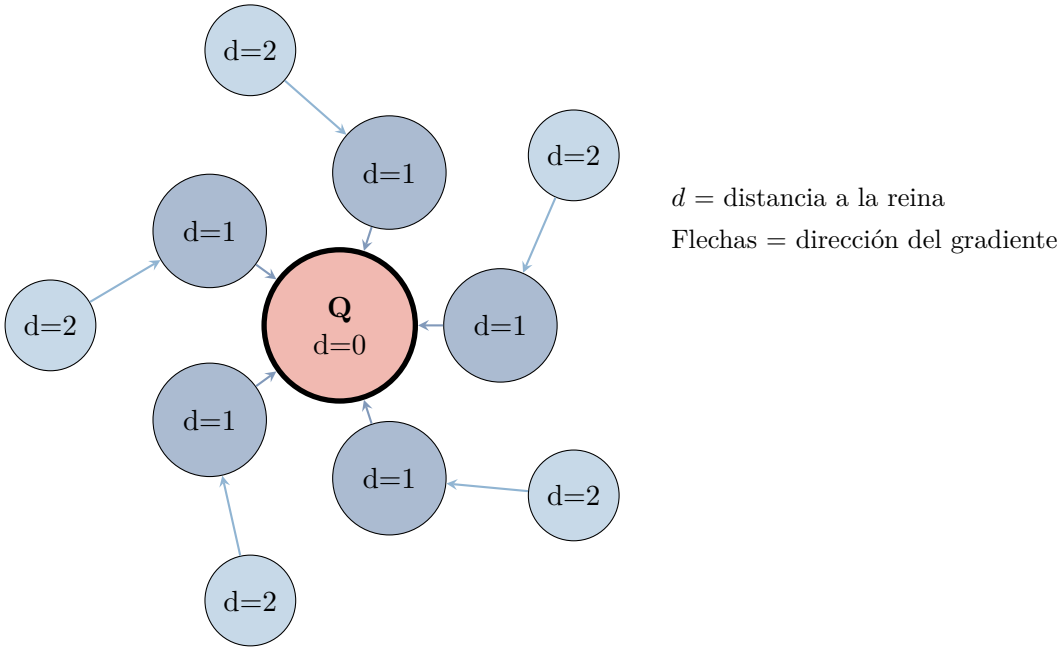


Figura 5.3: Enrutamiento por gradiente hacia la reina

Parte II

Protocolo NERT

Capítulo 6

Visión General de NERT

6.1. 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 6.1: Comparación de NERT con protocolos tradicionales

6.2. Clases de Confiabilidad

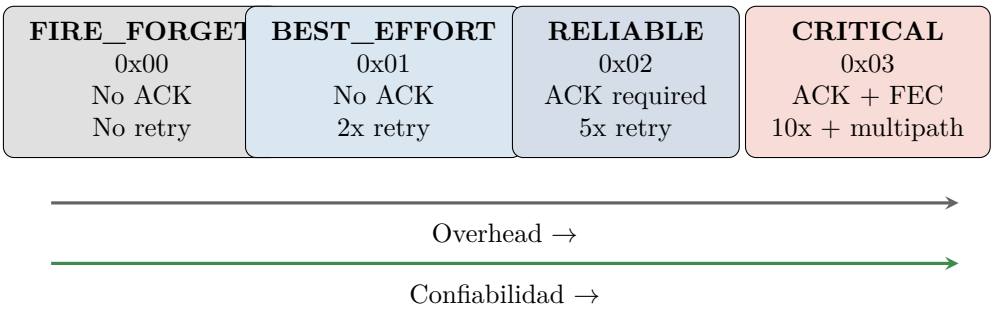


Figura 6.1: Clases de confiabilidad de NERT

Capítulo 7

Formato de Paquete NERT

7.1. Header Estándar (20 bytes)

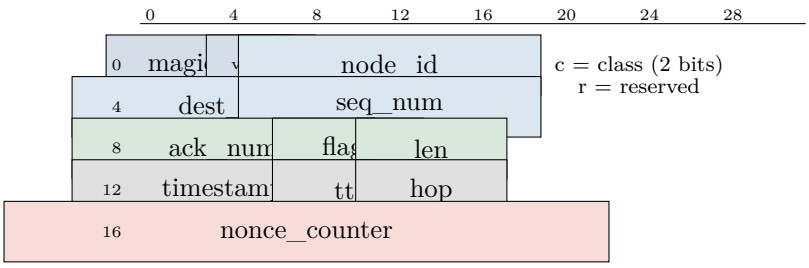


Figura 7.1: Header NERT estándar de 20 bytes

7.2. 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 7.1: Definición de bits del campo flags

7.3. Estructura Completa del Paquete

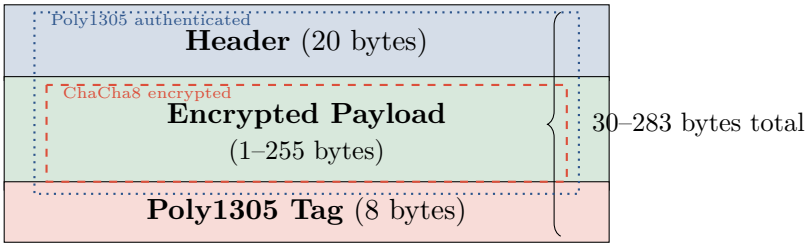


Figura 7.2: Estructura completa del paquete NERT

Capítulo 8

Criptografía

8.1. 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 8.1: Algoritmos criptográficos de NERT

8.2. Derivación de Claves

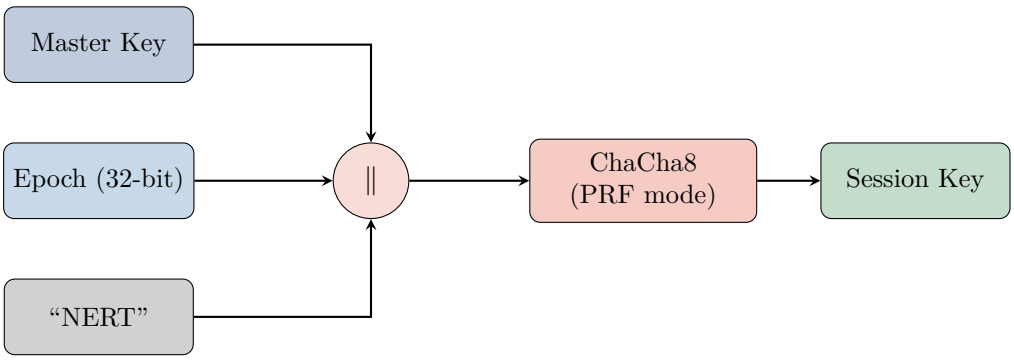
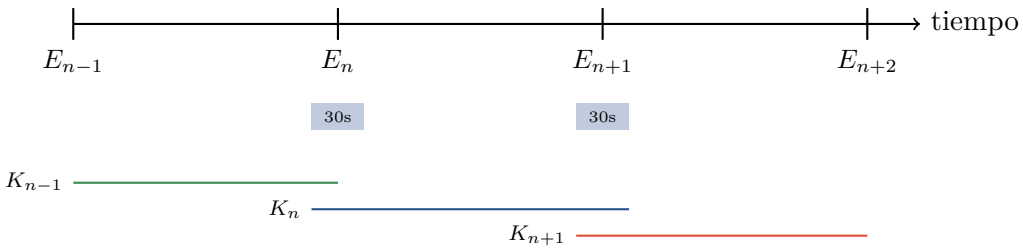


Figura 8.1: Proceso de derivación de clave de sesión

8.3. Ventana de Gracia para Rotación de Claves



Ventana de gracia: acepta claves de épocas adyacentes

Figura 8.2: 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.

8.4. Construcción del Nonce

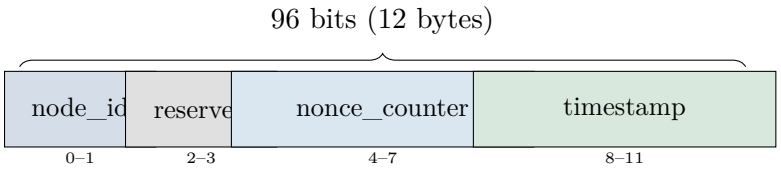


Figura 8.3: Estructura del nonce de 96 bits

Capítulo 9

Mecanismos de Confiabilidad

9.1. Máquina de Estados de Conexión

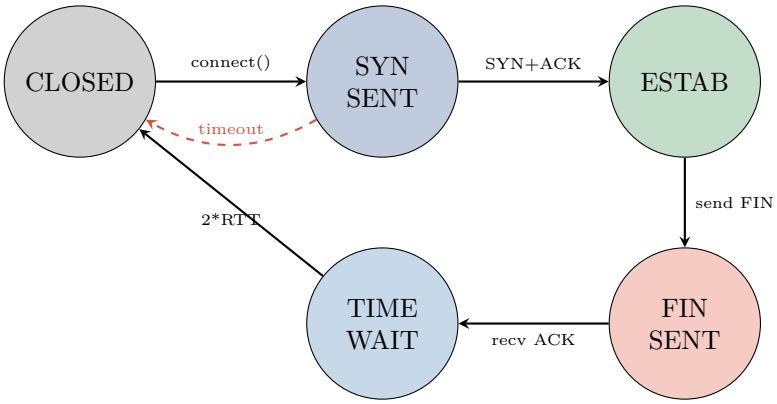


Figura 9.1: Máquina de estados de conexión NERT (simplificada)

9.2. Two-Way Handshake

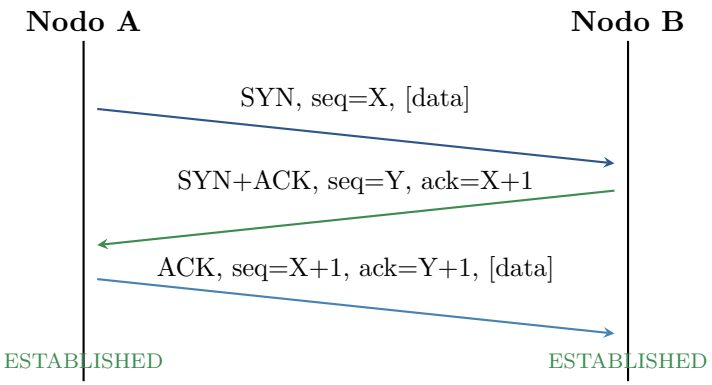


Figura 9.2: Handshake de dos vías de NERT

9.3. Selective ACK (SACK)

El SACK permite indicar eficientemente qué paquetes se han recibido:

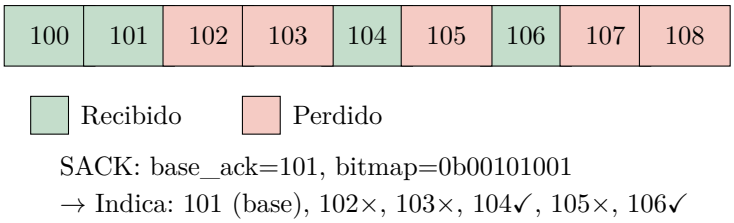


Figura 9.3: Ejemplo de Selective ACK

9.4. Retransmisión con Backoff Exponencial

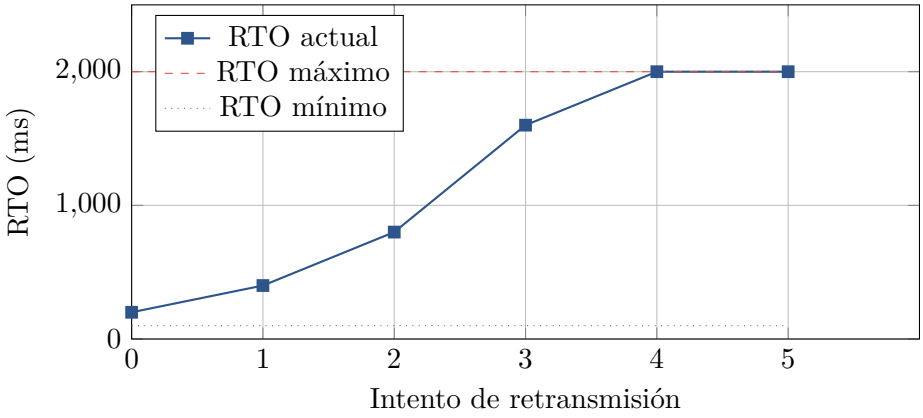


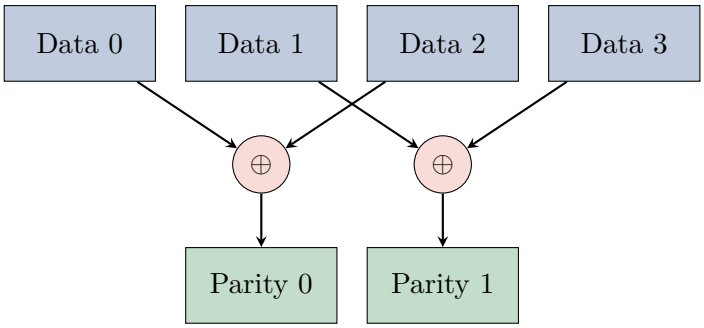
Figura 9.4: Crecimiento del RTO con backoff exponencial

Capítulo 10

Forward Error Correction

10.1. 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 10.1: Esquema FEC con paridad XOR

10.2. 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 10.1: Capacidad de recuperación del FEC

Capítulo 11

Multi-Path Transmission

11.1. Selección de Rutas

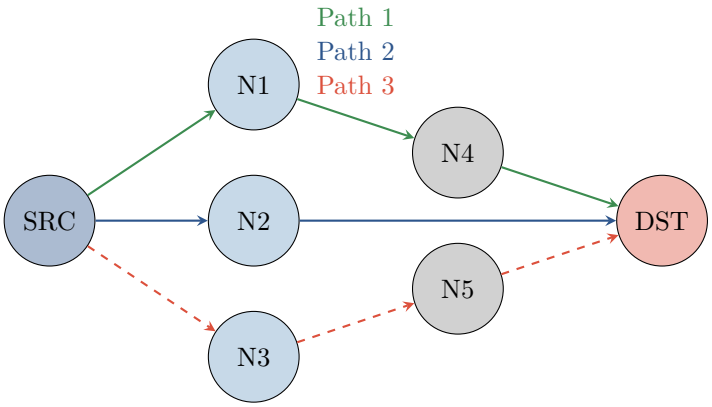


Figura 11.1: Transmisión multi-path con rutas diversas

Parte III

Guía de Implementación

Capítulo 12

Compilación y Despliegue

12.1. 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 12.1: Requisitos de compilación

12.2. Compilación x86

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

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

Listing 12.1: Compilación para QEMU x86

12.3. Compilación ARM

```
# Compilar para Stellaris
make PLATFORM=arm

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

Listing 12.2: Compilación para ARM Cortex-M3

12.4. Compilación ESP32

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

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

Listing 12.3: Compilación con PlatformIO

Capítulo 13

API de Programación

13.1. 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 13.1: Inicialización típica de NERT

13.2. 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 13.2: Ejemplos de envío con diferentes clases

Capítulo 14

Debugging y Monitoreo

14.1. 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 14.1: Acceso a estadísticas de protocolo

14.2. 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 14.1: 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

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.

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.

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.

Swarm
: Conjunto de nodos comunicándose colectivamente.

Sobre Este Documento

Este manual técnico fue generado como parte del proyecto NanOS v0.3. Cubre la arquitectura del sistema operativo y el protocolo NERT diseñado específicamente para comunicación entre nodos desechables.

NanOS Project
Los nodos mueren, el enjambre vive.
2026