tôi làm hệ thống giám sát có cấu trúc như sau. Hãy giải thích các hàm có trong file code sau .

4 Nguồn

Monica

Giải thích chi tiết các hàm trong utils.cpp và utils.hpp

Dựa trên cấu trúc dự án **NetworkSecurityAI** của bạn, file `utils.cpp` và `utils.hpp` nằm trong thư mục `src/common/` và cung cấp các hàm tiện ích dùng chung cho toàn bộ hệ thống. Dưới đây là giải thích chi tiết:

📋 I. TỔNG QUAN

File `utils` cung cấp các nhóm chức năng chính:

- ⏰ **Time utilities**: Xử lý thời gian
- 🔤 **String utilities**: Xử lý chuỗi
- 🔐 **Hash utilities**: Tính toán hash/checksum
- 📁 **File utilities**: Thao tác với file/thư mục
- 💾 **Memory utilities**: Xử lý bộ nhớ và dữ liệu binary
- 🖥️ **System utilities**: Thông tin hệ thống
- 🎲 **Random utilities**: Tạo dữ liệu ngẫu nhiên
- ✅ **Validation utilities**: Kiểm tra tính hợp lệ
- 🔄 **Conversion utilities**: Chuyển đổi kiểu dữ liệu

📦 II. CHI TIẾT CÁC NHÓM HÀM

⏰ 1. TIME UTILITIES

`getCurrentTimestamp()`

cpp Sao chép

```
std::string getCurrentTimestamp()
```

- **Mục đích**: Lấy thời gian hiện tại dạng chuỗi có định dạng
- **Trả về**: `"2025-01-15 14:30:45.123"` (có milliseconds)
- **Ứng dụng**: Ghi log, timestamp cho events

`getCurrentTimestampMs()` / `getCurrentTimestampUs()`

cpp Sao chép

```
uint64_t getCurrentTimestampMs()  // Milliseconds
uint64_t getCurrentTimestampUs()  // Microseconds
```

- **Mục đích**: Lấy timestamp số nguyên từ epoch (1970-01-01)
- **Ứng dụng**: Đo hiệu năng, so sánh thời gian, lưu trong database

`formatTimestamp(uint64_t timestamp, const std::string& format)`

cpp Sao chép

```
std::string formatTimestamp(1642253445123, "%Y-%m-%d")
// Kết quả: "2022-01-15"
```

- **Mục đích**: Chuyển timestamp số thành chuỗi với format tùy chỉnh
- **Tham số format**: Theo chuẩn `strftime` (`%Y`, `%m`, `%d`, `%H`, `%M`, `%S`)

`timeDifference(uint64_t start, uint64_t end)`

cpp Sao chép

```
uint64_t diff = timeDifference(start_time, end_time);
```

- **Mục đích**: Tính khoảng cách thời gian giữa 2 timestamp
- **Ứng dụng**: Đo thời gian xử lý gói tin, phát hiện timeout

🔤 2. STRING UTILITIES

`split(const std::string& str, char delimiter)`

cpp Sao chép

```cpp
auto parts = Utils::split("192.168.1.1:8080", ':');
// parts = {"192.168.1.1", "8080"}
```

- **Mục đích**: Tách chuỗi thành vector theo ký tự phân cách
- **Ứng dụng**: Parse IP:Port, phân tích log, xử lý CSV

`trim(const std::string& str, const std::string& chars)`

cpp Sao chép

```cpp
std::string clean = Utils::trim("  hello  \n");
// clean = "hello"
```

- **Mục đích**: Loại bỏ ký tự thừa ở đầu/cuối chuỗi
- **Ứng dụng**: Làm sạch input từ config file, user input

`toLowerCase()` / `toUpperCase()`

cpp Sao chép

```cpp
std::string lower = Utils::toLowerCase("HTTP");  // "http"
std::string upper = Utils::toUpperCase("get");   // "GET"
```

- **Ứng dụng**: So sánh protocol không phân biệt hoa/thường

`startsWith()` / `endsWith()`

cpp Sao chép

```cpp
bool isHTTP = Utils::startsWith(url, "http://");
bool isPNG = Utils::endsWith(filename, ".png");
```

- **Ứng dụng**: Kiểm tra protocol, file extension

`replaceAll(const std::string& str, const std::string& from, const std::string& to)`

cpp Sao chép

```
std::string safe = Utils::replaceAll(sql, "'", "''");
```

- **Ứng dụng**: Escape ký tự đặc biệt, sanitize input

`join(const std::vector<std::string>& strings, const std::string& delimiter)`

cpp Sao chép

```
auto ips = {"192.168.1.1", "10.0.0.1"};
std::string list = Utils::join(ips, ", ");
// list = "192.168.1.1, 10.0.0.1"
```

- **Ứng dụng**: Tạo chuỗi từ danh sách (cho log, report)

🔐 3. HASH UTILITIES

`calculateMD5(const void* data, size_t length)`

cpp Sao chép

```
std::string hash = Utils::calculateMD5(packet_data, packet_len);
```

- **Mục đích**: Tính MD5 hash (128-bit)
- **Ứng dụng**:
  - Kiểm tra malware signature
  - Verify file integrity
  - Cache key cho flow analysis
- ⚠️ **Lưu ý**: MD5 không an toàn cho mật mã, chỉ dùng cho checksum

`calculateSHA256(const void* data, size_t length)`

cpp Sao chép

```
std::string hash = Utils::calculateSHA256(file_content.c_str(), file_content.size());
```

- **Mục đích**: Tính SHA-256 hash (256-bit)

- **Ứng dụng**:
  - Verify model AI/ML integrity
  - Secure signature matching
  - Threat database indexing
- ✅ **An toàn hơn MD5**

`calculateCRC32(const void* data, size_t length)`

cpp Sao chép

```cpp
uint32_t checksum = Utils::calculateCRC32(packet,
packet_size);
```

- **Mục đích**: Tính CRC32 checksum (nhanh)
- **Ứng dụng**:
  - Kiểm tra lỗi gói tin
  - Quick integrity check
  - Hash table key

`hashString(const std::string& str)`

cpp Sao chép

```cpp
uint64_t flow_hash = Utils::hashString(src_ip + ":" +
dst_ip);
```

- **Mục đích**: Hash nhanh dùng FNV-1a algorithm
- **Ứng dụng**:
  - Flow table indexing
  - Fast string comparison
  - Hash map key

📁 4. FILE UTILITIES

`fileExists() / directoryExists()`

cpp Sao chép

```
if (Utils::fileExists("/etc/nsai/rules.conf")) {
    // Load rules
}
```

- **Ứng dụng**: Kiểm tra config file, model file trước khi load

`createDirectory(const std::string& dirpath)`

cpp Sao chép

```
Utils::createDirectory("/var/log/nsai/2025/01");
```

- **Mục đích**: Tạo thư mục đệ quy (bao gồm parent directories)
- **Ứng dụng**: Tạo log directory theo ngày/tháng

`readFileToString()` / `readFileToBytes()`

cpp Sao chép

```
std::string config = Utils::readFileToString("config.yaml");
std::vector<uint8_t> model =
Utils::readFileToBytes("model.bin");
```

- **Ứng dụng**:
  - Load config files
  - Load AI/ML models
  - Load signature databases

`writeStringToFile()` / `writeBytesToFile()`

cpp Sao chép

```
Utils::writeStringToFile("alert.log", alert_msg, true);  //
append=true
Utils::writeBytesToFile("pcap/capture.pcap", packet_data,
false);
```

- **Ứng dụng**:
  - Ghi log
  - Lưu PCAP captures

- Export reports

`getFileSize()` / `getFileModificationTime()`

cpp Sao chép

```cpp
size_t size = Utils::getFileSize("threats.db");
uint64_t mtime =
Utils::getFileModificationTime("rules.conf");
```

- **Ứng dụng**:
  - Kiểm tra file đã thay đổi (reload config)
  - Monitor log rotation
  - Database size tracking

`getFileName()` / `getDirectoryName()` / `getFileExtension()`

cpp Sao chép

```cpp
std::string name = Utils::getFileName("/var/log/nsai.log");
// "nsai.log"
std::string dir =
Utils::getDirectoryName("/var/log/nsai.log");  // "/var/log"
std::string ext = Utils::getFileExtension("capture.pcap");
// ".pcap"
```

- **Ứng dụng**: Parse đường dẫn file

💾 5. MEMORY UTILITIES

`formatBytes(size_t bytes)`

cpp Sao chép

```cpp
std::string size = Utils::formatBytes(1536000);
// size = "1.46 MB"
```

- **Ứng dụng**: Hiển thị memory usage, traffic volume trong dashboard

`hexDump(const void* data, size_t length, std::ostream& os)`

cpp Sao chép

```
Utils::hexDump(packet_payload, payload_len, std::cout);
```

- **Kết quả**:

```
makefile
```

Sao chép

```
00000000: 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a
GET / HTTP/1.1..
00000010: 48 6f 73 74 3a 20 65 78 61 6d 70 6c 65 2e 63 6f
Host: example.co
```

- **Ứng dụng**: Debug packet payload, analyze malware

`bytesToHex()` / `hexToBytes()`

```
cpp
```

Sao chép

```
std::string hex = Utils::bytesToHex(data, len);  //
"48656c6c6f"
std::vector<uint8_t> bytes =
Utils::hexToBytes("48656c6c6f");  // "Hello"
```

- **Ứng dụng**:
  - Display packet data
  - Parse hex signatures từ rule files

`secureMemoryCompare(const void* ptr1, const void* ptr2, size_t length)`

```
cpp
```

Sao chép

```
bool match = Utils::secureMemoryCompare(password_hash,
stored_hash, 32);
```

- **Mục đích**: So sánh bộ nhớ **constant-time** (chống timing attack)
- **Ứng dụng**: So sánh hash, password, cryptographic keys

🖥️ 6. SYSTEM UTILITIES

`getCPUCount()`

```
cpp
```

Sao chép

```
int cores = Utils::getCPUCount();  // 8
// Tạo 8 worker threads cho DPDK
```

- **Ứng dụng**: Tối ưu số worker threads theo CPU cores

`getMemoryUsage()` / `getTotalSystemMemory()` / `getAvailableSystemMemory()`

cpp Sao chép

```
size_t used = Utils::getMemoryUsage();
size_t total = Utils::getTotalSystemMemory();
double usage_percent = (used * 100.0) / total;
```

- **Ứng dụng**:
  - Monitor resource usage
  - Trigger alerts khi memory cao
  - Dashboard metrics

`getHostname()` / `getProcessName()` / `getProcessId()`

cpp Sao chép

```
std::string host = Utils::getHostname();  // "firewall-01"
std::string proc = Utils::getProcessName();  // "nsai-
daemon"
int pid = Utils::getProcessId();  // 12345
```

- **Ứng dụng**: Log identification, distributed system tracking

`getCPUUsage()`

cpp Sao chép

```
double cpu = Utils::getCPUUsage();  // 45.2%
```

- **Mục đích**: Đọc từ `/proc/stat` (Linux)
- **Ứng dụng**: Performance monitoring, load balancing

🎲 7. RANDOM UTILITIES

`randomInt()` / `randomDouble()`

cpp Sao chép

```cpp
int port = Utils::randomInt(10000, 65535);
double threshold = Utils::randomDouble(0.0, 1.0);
```

- **Ứng dụng**:
  - Generate test data
  - Random sampling trong ML
  - Jitter cho rate limiting

`randomString(size_t length, const std::string& charset)`

cpp Sao chép

```cpp
std::string session_id = Utils::randomString(32);
// "aB3xK9mP..."
```

- **Ứng dụng**: Generate session IDs, tokens, test data

`generateUUID()`

cpp Sao chép

```cpp
std::string flow_id = Utils::generateUUID();
// "550e8400-e29b-41d4-a716-446655440000"
```

- **Ứng dụng**:
  - Unique flow identifiers
  - Alert IDs
  - Correlation IDs trong distributed tracing

✅ 8. VALIDATION UTILITIES

`isNumeric()` / `isInteger()` / `isFloat()`

cpp Sao chép

```
bool valid = Utils::isInteger("8080");  // true
bool valid_ip = Utils::isNumeric("192.168.1.1");  // false
(có dấu chấm)
```

- **Ứng dụng**: Validate config values, user input

`isValidEmail(const std::string& email)`

cpp Sao chép

```
bool valid = Utils::isValidEmail("admin@example.com");  //
true
```

- **Mục đích**: Regex validation cho email
- **Ứng dụng**: Validate alert recipients

🔄 9. CONVERSION UTILITIES

`stringToInt()` / `stringToDouble()` / `stringToBool()`

cpp Sao chép

```
int port = Utils::stringToInt(config["port"], 8080);  //
default=8080
bool enabled = Utils::stringToBool(config["enable_ai"],
true);
```

- **Mục đích**: Safe conversion với giá trị mặc định
- **Ứng dụng**: Parse config files an toàn (không throw exception)

🛠️ III. CÁC CLASS HỖ TRỢ

⏱️ Timer Class

cpp Sao chép

```
Timer timer("Packet Processing");
timer.start();

// ... xử lý gói tin ...

timer.stop();
```

```
double elapsed_ms = timer.getElapsedMs();
LOG_INFO("Processing took " << elapsed_ms << " ms");
```

Ứng dụng trong hệ thống của bạn:

- Đo thời gian xử lý ở mỗi tầng (Layer 1 IPS, Layer 2 AI)
- Benchmark signature matching (Aho-Corasick)
- Profiling AI inference time

🔒 Singleton Template

cpp Sao chép

```
class FlowManager : public Singleton<FlowManager> {
    // ...
};

// Sử dụng
FlowManager::getInstance().addFlow(flow);
```

Ứng dụng:

- `ConfigManager`, `Logger`, `ThreatDatabase` (chỉ cần 1 instance)
- Thread-safe initialization

🧹 ScopeGuard Class

cpp Sao chép

```
void processPacket(Packet* pkt) {
    ScopeGuard cleanup([pkt]() {
        delete pkt;  // Tự động cleanup khi ra khỏi scope
    });

    if (error) return;  // pkt vẫn được delete

    // ... xử lý ...
}
```

Ứng dụng:

- RAII cho DPDK memory pools

- Auto-release locks
- Cleanup temporary files

🎯 IV. ỨNG DỤNG TRONG HỆ THỐNG CỦA BẠN

1. Tầng 1 (IPS Layer)

cpp Sao chép

```cpp
// XDP Filter
uint64_t src_ip_hash = Utils::hashString(src_ip_str);
if (blacklist.contains(src_ip_hash)) {
    return XDP_DROP;
}

// Signature Matching
std::string payload_hex = Utils::bytesToHex(payload,
payload_len);
if (Utils::containsIgnoreCase(payload_hex,
malware_signature)) {
    Utils::hexDump(payload, payload_len, alert_log);
}
```

2. Tầng 2 (AI/ML Layer)

cpp Sao chép

```cpp
// Feature Extraction
Timer timer("Feature Extraction");
std::vector<double> features = extractFeatures(flow);
double elapsed = timer.getElapsedUs();

// Model Loading
if (Utils::fileExists("models/autoencoder.onnx")) {
    auto model_data =
Utils::readFileToBytes("models/autoencoder.onnx");
    std::string model_hash =
Utils::calculateSHA256(model_data.data(),
model_data.size());
    // Verify integrity
}
```

3. Dashboard & Logging

cpp Sao chép

```
// Log formatting
std::string timestamp = Utils::getCurrentTimestamp();
std::string log_msg = timestamp + " [ALERT] " +
                      "Threat detected from " + src_ip;
Utils::writeStringToFile("logs/alerts.log", log_msg, true);

// System metrics
double cpu = Utils::getCPUUsage();
std::string mem =
Utils::formatBytes(Utils::getMemoryUsage());
dashboard.updateMetrics(cpu, mem);
```

📊 V. BẢNG TÓM TẮT NHANH

Nhóm	Hàm quan trọng nhất	Ứng dụng chính
Time	`getCurrentTimestampMs()`	Timestamp cho flows, events
String	`split()`, `trim()`	Parse config, protocols
Hash	`calculateSHA256()`, `hashString()`	Signature matching, flow indexing
File	`readFileToBytes()`, `writeStringToFile()`	Load models, write logs
Memory	`hexDump()`, `formatBytes()`	Debug packets, monitor resources
System	`getCPUCount()`, `getMemoryUsage()`	Optimize threads, alerts
Random	`generateUUID()`	Flow IDs, session tracking

💡 VI. LƯU Ý QUAN TRỌNG

1. **Thread-safety**: Các hàm `random*()` sử dụng `thread_local` → an toàn cho multi-threading

2. **Error handling**: Các hàm conversion trả về giá trị mặc định thay vì throw exception
3. **Performance**:
   - `hashString()` dùng FNV-1a (nhanh hơn MD5/SHA256)
   - `secureMemoryCompare()` chống timing attack
4. **OpenSSL 3.0**: Code đã update dùng EVP API thay vì deprecated MD5/SHA256 functions

2 Nguồn

Monica

Các hàm này tạo thành **foundation layer** cho toàn bộ hệ thống NetworkSecurityAI của bạn, giúp code khác tập trung vào logic nghiệp vụ thay vì xử lý low-level utilities! 🚀

Giải thích chi tiết các hàm trong network_utils.cpp và network_utils.hpp

File `network_utils` cung cấp các tiện ích chuyên biệt cho xử lý mạng trong hệ thống **NetworkSecurityAI** của bạn. Đây là thành phần quan trọng cho **Layer 1 (IPS)** và các module phân tích traffic.

📋 I. TỔNG QUAN CÁC NHÓM CHỨC NĂNG

Nhóm	Mục đích	Ứng dụng chính
🌐 **IP Address**	Xử lý địa chỉ IP	Validation, conversion, classification
🔢 **Network Range**	Tính toán subnet	CIDR parsing, IP range checking
🔌 **Port**	Quản lý cổng	Service identification, port classification
📡 **Protocol**	Nhận dạng giao thức	TCP/UDP/ICMP detection
🏷️ **MAC Address**	Xử lý địa chỉ vật lý	Vendor lookup, normalization
Network Interface	Quản lý card mạng	Get IP/MAC, check status

Nhóm	Mục đích	Ứng dụng chính
🌍 **DNS**	Phân giải tên miền	Hostname ↔ IP conversion
🗺️ **Geolocation**	Định vị địa lý	Country/ISP lookup
📊 **Traffic Analysis**	Phân tích lưu lượng	Rate calculation, utilization

🌐 II. IP ADDRESS UTILITIES

1. Validation Functions

`isValidIPv4(const std::string& ip)`

cpp Sao chép

```cpp
bool valid = NetworkUtils::isValidIPv4("192.168.1.1");  // true
bool valid = NetworkUtils::isValidIPv4("999.999.999.999"); // false
```

- **Cơ chế**: Sử dụng `inet_pton()` để parse IP
- **Ứng dụng**:
  - Validate IP từ config file
  - Kiểm tra input từ blacklist/whitelist
  - Filter invalid packets

`isValidIPv6(const std::string& ip)`

cpp Sao chép

```cpp
bool valid = NetworkUtils::isValidIPv6("2001:0db8::1");  // true
```

- **Ứng dụng**: Hỗ trợ IPv6 traffic monitoring

2. Conversion Functions

`ipStringToInt(const std::string& ip)` ↔ `ipIntToString(uint32_t ip)`

cpp Sao chép

```cpp
uint32_t ip_int =
NetworkUtils::ipStringToInt("192.168.1.100");
// ip_int = 3232235876 (0xC0A80164)

std::string ip_str =
NetworkUtils::ipIntToString(3232235876);
// ip_str = "192.168.1.100"
```

Tại sao cần chuyển đổi?

- ✅ **So sánh nhanh**: `if (ip_int >= start && ip_int <= end)` nhanh hơn string
- ✅ **Tính toán subnet**: Dùng bitwise operations
- ✅ **Hash table key**: Integer làm key hiệu quả hơn

Ứng dụng trong hệ thống:

cpp Sao chép

```cpp
// XDP Filter - Fast blacklist lookup
uint32_t src_ip =
NetworkUtils::ipStringToInt(packet.src_ip);
if (blacklist_bitmap[src_ip >> 5] & (1 << (src_ip & 0x1F)))
{
    return XDP_DROP;
}
```

3. Classification Functions

`isPrivateIP(const std::string& ip)`

cpp Sao chép

```cpp
bool is_private = NetworkUtils::isPrivateIP("10.0.0.1");  //
true
bool is_private = NetworkUtils::isPrivateIP("8.8.8.8");   //
false
```

Phạm vi Private IP:

- `10.0.0.0/8` → `0x0A000000 - 0x0AFFFFFF`
- `172.16.0.0/12` → `0xAC100000 - 0xAC1FFFFF`
- `192.168.0.0/16` → `0xC0A80000 - 0xC0A8FFFF`

Ứng dụng:

```cpp
// Threat Detection - Ignore internal scans
if (NetworkUtils::isPrivateIP(dst_ip) &&
NetworkUtils::isPrivateIP(src_ip)) {
    severity = LOW;  // Internal traffic
} else {
    severity = HIGH; // External attack
}
```

`isLoopbackIP(const std::string& ip)`

```cpp
bool is_local = NetworkUtils::isLoopbackIP("127.0.0.1");  //
true
```

- **Phạm vi**: `127.0.0.0/8`
- **Ứng dụng**: Filter local testing traffic

`isMulticastIP(const std::string& ip)`

```cpp
bool is_multicast =
NetworkUtils::isMulticastIP("224.0.0.1");  // true
```

- **Phạm vi**: `224.0.0.0/4` (224.0.0.0 - 239.255.255.255)
- **Ứng dụng**: Detect multicast attacks (IGMP flooding)

🔢 III. NETWORK RANGE UTILITIES

1. CIDR Operations

`isIPInCIDR(const std::string& ip, const std::string& cidr)`

cpp Sao chép

```cpp
bool in_range = NetworkUtils::isIPInCIDR("192.168.1.50",
"192.168.1.0/24");
// true
```

Cách hoạt động:

cpp Sao chép

```cpp
// Parse CIDR: "192.168.1.0/24"
network = "192.168.1.0"
prefix_len = 24

// Calculate mask
mask = 0xFFFFFF00  // (255.255.255.0)

// Check
(ip & mask) == (network & mask)
```

Ứng dụng:

cpp Sao chép

```cpp
// Whitelist checking
std::vector<std::string> trusted_networks = {
    "10.0.0.0/8",
    "192.168.0.0/16"
};

for (const auto& cidr : trusted_networks) {
    if (NetworkUtils::isIPInCIDR(src_ip, cidr)) {
        return ALLOW;
    }
}
```

`expandCIDR(const std::string& cidr)`

cpp Sao chép

```cpp
auto ips = NetworkUtils::expandCIDR("192.168.1.0/30");
// ips = {"192.168.1.1", "192.168.1.2"}
// (Không bao gồm network address .0 và broadcast .3)
```

⚠️ **Giới hạn an toàn:**

- Chỉ expand tối đa **65,536 IPs** (tránh memory overflow)
- Không expand `/31` và `/32` (không có host addresses)

Ứng dụng:

```cpp
// Generate IP ranges for scanning detection
auto scanner_ips = NetworkUtils::expandCIDR("10.0.0.0/24");
for (const auto& ip : scanner_ips) {
    port_scan_tracker[ip] = 0;
}
```

2. Subnet Calculations

`calculateSubnetMask(int prefix_len)`

```cpp
uint32_t mask = NetworkUtils::calculateSubnetMask(24);
// mask = 0xFFFFFF00 (255.255.255.0)
```

Công thức: `mask = 0xFFFFFFFF << (32 - prefix_len)`

Prefix	Mask	Hosts
/8	255.0.0.0	16,777,214
/16	255.255.0.0	65,534
/24	255.255.255.0	254
/30	255.255.255.252	2

`calculateNetworkAddress()` / `calculateBroadcastAddress()`

```cpp
uint32_t ip = NetworkUtils::ipStringToInt("192.168.1.50");

uint32_t network = NetworkUtils::calculateNetworkAddress(ip,
```

```
24);
// 192.168.1.0

uint32_t broadcast =
NetworkUtils::calculateBroadcastAddress(ip, 24);
// 192.168.1.255
```

🔌 IV. PORT UTILITIES

1. Port Classification

`isWellKnownPort(int port)` / `isRegisteredPort()` / `isDynamicPort()`

cpp Sao chép

```
bool is_system = NetworkUtils::isWellKnownPort(80);      //
true (1-1023)
bool is_app = NetworkUtils::isRegisteredPort(8080);      //
true (1024-49151)
bool is_ephemeral = NetworkUtils::isDynamicPort(54321);  //
true (49152-65535)
```

Ứng dụng trong phân tích:

cpp Sao chép

```
// Detect port scanning
if (NetworkUtils::isWellKnownPort(dst_port) &&
    connection_attempts > 10) {
    alert("Port scan detected on critical services");
}
```

2. Service Identification

`getPortService(int port)`

cpp Sao chép

```
std::string service = NetworkUtils::getPortService(443);  //
"HTTPS"
std::string service = NetworkUtils::getPortService(3306); //
"MYSQL"
std::string service = NetworkUtils::getPortService(12345);
// "UNKNOWN"
```

Database tích hợp:

cpp Sao chép

```
{20, "FTP-DATA"}, {21, "FTP"}, {22, "SSH"}, {23, "TELNET"},
{25, "SMTP"}, {53, "DNS"}, {80, "HTTP"}, {443, "HTTPS"},
{3306, "MYSQL"}, {3389, "RDP"}, {5432, "POSTGRESQL"}
```

Ứng dụng:

cpp Sao chép

```
// Protocol anomaly detection
std::string expected_service =
NetworkUtils::getPortService(dst_port);
if (expected_service == "HTTP" && detected_protocol ==
"SSH") {
    alert("Protocol mismatch - possible tunneling");
}
```

📡 V. PROTOCOL UTILITIES

getProtocolName(int protocol_number) ↔ getProtocolNumber(const std::string& name)

cpp Sao chép

```
std::string proto = NetworkUtils::getProtocolName(6);   //
"TCP"
int proto_num = NetworkUtils::getProtocolNumber("UDP"); //
17
```

Protocol mapping:

cpp Sao chép

```
{1, "ICMP"}, {6, "TCP"}, {17, "UDP"}, {47, "GRE"},
{50, "ESP"}, {89, "OSPF"}, {132, "SCTP"}
```

isTCPProtocol() / isUDPProtocol() / isICMPProtocol()

cpp Sao chép

```
if (NetworkUtils::isTCPProtocol(packet.protocol)) {
    // Parse TCP flags
    analyzeTCPFlags(packet);
}
```

Ứng dụng trong Flow Assembly:

cpp Sao chép

```
// flow_assembly.cpp
FlowKey key;
key.protocol = packet.protocol;

if (NetworkUtils::isTCPProtocol(key.protocol)) {
    flow->state = analyzeTCPState(packet.tcp_flags);
} else if (NetworkUtils::isUDPProtocol(key.protocol)) {
    flow->state = STATELESS;
}
```

🏷️ VI. MAC ADDRESS UTILITIES

1. Validation & Normalization

`isValidMAC(const std::string& mac)`

cpp Sao chép

```
bool valid = NetworkUtils::isValidMAC("00:1A:2B:3C:4D:5E");
// true
bool valid = NetworkUtils::isValidMAC("00-1A-2B-3C-4D-5E");
// true
bool valid = NetworkUtils::isValidMAC("001A2B3C4D5E");
// true
```

Hỗ trợ 3 format:

- Colon: `XX:XX:XX:XX:XX:XX`
- Dash: `XX-XX-XX-XX-XX-XX`
- Plain: `XXXXXXXXXXXX`

`normalizeMACAddress(const std::string& mac)`

cpp Sao chép

```
std::string mac = NetworkUtils::normalizeMACAddress("00-1a-
2b-3c-4d-5e");
// mac = "00:1A:2B:3C:4D:5E"
```

Quy trình:

1. Remove separators (`:` , `-`)
2. Convert to uppercase
3. Add colons every 2 characters

2. Vendor Lookup

`getVendorFromMAC(const std::string& mac)`

cpp Sao chép

```
std::string vendor =
NetworkUtils::getVendorFromMAC("00:0C:29:12:34:56");
// vendor = "VMware"
```

OUI Database (Organizationally Unique Identifier):

cpp Sao chép

```
{"00:00:0C", "Cisco Systems"},
{"00:0C:29", "VMware"},
{"00:15:5D", "Microsoft Corporation"},
{"08:00:27", "PCS Systemtechnik GmbH"},  // VirtualBox
{"52:54:00", "QEMU/KVM"}
```

Ứng dụng:

cpp Sao chép

```
// Detect virtualization
std::string vendor =
NetworkUtils::getVendorFromMAC(src_mac);
if (vendor == "VMware" || vendor == "VirtualBox") {
    alert("Traffic from virtual machine detected");
}
```

VII. NETWORK INTERFACE UTILITIES

1. Interface Discovery

`getNetworkInterfaces()`

cpp Sao chép

```cpp
auto interfaces = NetworkUtils::getNetworkInterfaces();
// interfaces = {"lo", "eth0", "eth1", "wlan0"}
```

Cơ chế: Sử dụng `getifaddrs()` để enumerate interfaces

2. Interface Information

`getInterfaceIP(const std::string& interface)`

cpp Sao chép

```cpp
std::string ip = NetworkUtils::getInterfaceIP("eth0");
// ip = "192.168.1.100"
```

`getInterfaceMAC(const std::string& interface)`

cpp Sao chép

```cpp
std::string mac = NetworkUtils::getInterfaceMAC("eth0");
// mac = "00:1A:2B:3C:4D:5E"
```

Cơ chế: Sử dụng `ioctl()` với `SIOCGIFHWADDR`

`isInterfaceUp(const std::string& interface)`

cpp Sao chép

```cpp
bool is_up = NetworkUtils::isInterfaceUp("eth0");  // true
```

Ứng dụng trong DPDK Dispatcher:

cpp Sao chép

```cpp
// dpdk_dispatcher.cpp
auto interfaces = NetworkUtils::getNetworkInterfaces();
for (const auto& iface : interfaces) {
```

```
    if (NetworkUtils::isInterfaceUp(iface) &&
        iface.find("eth") != std::string::npos) {
        dpdk_ports.push_back(iface);
    }
}
```

🌍 VIII. DNS UTILITIES

1. Hostname Resolution

`resolveHostname(const std::string& ip)`

cpp Sao chép

```
std::string hostname =
NetworkUtils::resolveHostname("8.8.8.8");
// hostname = "dns.google"
```

Cơ chế: Reverse DNS lookup với `getnameinfo()`

`resolveIP(const std::string& hostname)`

cpp Sao chép

```
auto ips = NetworkUtils::resolveIP("google.com");
// ips = {"142.250.185.46", "142.250.185.78", ...}
```

Cơ chế: Forward DNS lookup với `getaddrinfo()`

2. Domain Validation

`isValidDomainName(const std::string& domain)`

cpp Sao chép

```
bool valid = NetworkUtils::isValidDomainName("example.com");
// true
bool valid =
NetworkUtils::isValidDomainName("invalid..com"); // false
```

Regex validation:

- Độ dài: 1-253 ký tự

- Label: `[a-zA-Z0-9] + -` (không bắt đầu/kết thúc bằng `-`)
- Max label length: 63 ký tự

Ứng dụng:

cpp Sao chép

```
// DNS tunneling detection
if (domain.length() > 100 &&
NetworkUtils::isValidDomainName(domain)) {
    alert("Possible DNS tunneling - abnormally long
domain");
}
```

🗺️ IX. GEOLOCATION UTILITIES

GeoLocation Struct

cpp Sao chép

```
struct GeoLocation {
    std::string country;        // "United States"
    std::string country_code;   // "US"
    std::string region;         // "California"
    std::string city;           // "Mountain View"
    double latitude;            // 37.4056
    double longitude;           // -122.0775
    std::string isp;            // "Google LLC"
    std::string organization;   // "Google Cloud"
};
```

getIPGeolocation(const std::string& ip)

cpp Sao chép

```
auto geo = NetworkUtils::getIPGeolocation("8.8.8.8");
// geo.country = "United States"
// geo.isp = "Google LLC"
```

⚠️ **Lưu ý**: Implementation hiện tại là placeholder. Trong production cần:

- **MaxMind GeoIP2**: Database thương mại
- **IP2Location**: Alternative database

- **API services**: ipapi.co, ipgeolocation.io

Ứng dụng:

cpp Sao chép

```cpp
// Geo-blocking
auto geo = NetworkUtils::getIPGeolocation(src_ip);
if (geo.country_code == "CN" || geo.country_code == "RU") {
    if (isHighRiskTraffic(packet)) {
        return DROP;
    }
}
```

📊 X. TRAFFIC ANALYSIS UTILITIES

1. Rate Calculations

`calculatePacketRate(uint64_t packet_count, uint64_t time_window_ms)`

cpp Sao chép

```cpp
double pps = NetworkUtils::calculatePacketRate(10000, 1000);
// pps = 10,000 packets/second
```

Công thức: `rate = (packet_count * 1000) / time_window_ms`

`calculateBitRate(uint64_t byte_count, uint64_t time_window_ms)`

cpp Sao chép

```cpp
double bps = NetworkUtils::calculateBitRate(125000, 1000);
// bps = 1,000,000 bps = 1 Mbps
```

Công thức: `rate = (byte_count * 8 * 1000) / time_window_ms`

2. Utilization Calculation

`calculateUtilization(uint64_t bytes, uint64_t time_ms, uint64_t bandwidth_bps)`

cpp Sao chép

```cpp
double util = NetworkUtils::calculateUtilization(
    125000000,  // 125 MB transferred
    1000,       // in 1 second
    1000000000  // on 1 Gbps link
);
// util = 100.0%
```

Ứng dụng:

cpp Sao chép

```cpp
// Bandwidth monitoring
double utilization = NetworkUtils::calculateUtilization(
    stats.total_bytes,
    time_window,
    interface_bandwidth
);

if (utilization > 80.0) {
    alert("High bandwidth utilization: " +
std::to_string(utilization) + "%");
}
```

📈 XI. NETWORKSTATSCOLLECTOR CLASS

`NetworkStats` Struct

cpp Sao chép

```cpp
struct NetworkStats {
    uint64_t total_packets;
    uint64_t total_bytes;
    uint64_t tcp_packets;
    uint64_t udp_packets;
    uint64_t icmp_packets;
    uint64_t other_packets;

    double packet_rate;
    double byte_rate;
    double average_packet_size;

    std::chrono::steady_clock::time_point start_time;
    std::chrono::steady_clock::time_point last_update;
};
```

Methods

`updateStats(size_t packet_size, uint8_t protocol)`

cpp Sao chép

```cpp
NetworkStatsCollector collector;

// Update for each packet
collector.updateStats(packet.size, packet.protocol);
```

Xử lý tự động:

- Phân loại protocol (TCP/UDP/ICMP/Other)
- Tính packet rate & byte rate
- Tính average packet size
- Thread-safe với `std::mutex`

`getStats()` / `resetStats()`

cpp Sao chép

```cpp
// Get current stats
NetworkStats stats = collector.getStats();
std::cout << "Total packets: " << stats.total_packets <<
std::endl;
std::cout << "Packet rate: " << stats.packet_rate << " pps"
<< std::endl;

// Reset counters
collector.resetStats();
```

Ứng dụng trong Dashboard

cpp Sao chép

```cpp
// dashboard_controller.cpp
NetworkStatsCollector global_stats;

void updateDashboard() {
    auto stats = global_stats.getStats();

    dashboard.setMetric("Total Packets",
stats.total_packets);
```

```
    dashboard.setMetric("Packet Rate", stats.packet_rate);
    dashboard.setMetric("Bandwidth",
formatBitRate(stats.byte_rate));

    // Protocol distribution
    dashboard.setPieChart("Protocols", {
        {"TCP", stats.tcp_packets},
        {"UDP", stats.udp_packets},
        {"ICMP", stats.icmp_packets}
    });
}
```

🎯 XII. ỨNG DỤNG TỔNG HỢP TRONG HỆ THỐNG

1. XDP Filter (Layer 1)

cpp Sao chép

```
// xdp_filter.cpp
int xdp_filter_prog(struct xdp_md *ctx) {
    uint32_t src_ip = get_src_ip(ctx);

    // Fast blacklist check
    if (is_blacklisted(src_ip)) {
        return XDP_DROP;
    }

    // Private IP bypass
    if (is_private_ip_range(src_ip)) {
        return XDP_PASS;
    }

    return XDP_PASS;
}
```

2. Flow Assembly (Layer 1)

cpp Sao chép

```
// flow_assembly.cpp
Flow* assembleFlow(Packet* packet) {
    FlowKey key;
    key.src_ip = NetworkUtils::ipStringToInt(packet-
>src_ip);
    key.dst_ip = NetworkUtils::ipStringToInt(packet-
>dst_ip);
```

```
    key.protocol = packet->protocol;

    // Protocol-specific handling
    if (NetworkUtils::isTCPProtocol(key.protocol)) {
        return assembleTCPFlow(packet);
    } else if (NetworkUtils::isUDPProtocol(key.protocol)) {
        return assembleUDPFlow(packet);
    }

    return nullptr;
}
```

3. Threat Detection (Layer 1)

cpp Sao chép

```
// threat_detection.cpp
ThreatLevel detectThreat(Flow* flow) {
    // Geolocation-based detection
    auto geo = NetworkUtils::getIPGeolocation(flow-
>src_ip_str);
    if (geo.country_code == "XX" && flow->dst_port == 22) {
        return HIGH_RISK;
    }

    // Port scan detection
    if (flow->unique_ports > 100 &&
        NetworkUtils::isWellKnownPort(flow->dst_port)) {
        return PORT_SCAN;
    }

    // Protocol anomaly
    std::string expected =
NetworkUtils::getPortService(flow->dst_port);
    if (expected != flow->detected_protocol) {
        return PROTOCOL_MISMATCH;
    }

    return NORMAL;
}
```

4. Feature Extraction (Layer 2 AI)

cpp Sao chép

```
// feature_extractor.cpp
std::vector<double> extractNetworkFeatures(Flow* flow) {
    std::vector<double> features;
```

```
    // IP features
    features.push_back(NetworkUtils::isPrivateIP(flow->src_ip_str) ? 1.0 : 0.0);
    features.push_back(NetworkUtils::isPrivateIP(flow->dst_ip_str) ? 1.0 : 0.0);

    // Port features
    features.push_back(NetworkUtils::isWellKnownPort(flow->dst_port) ? 1.0 : 0.0);

    // Protocol features
    features.push_back(NetworkUtils::isTCPProtocol(flow->protocol) ? 1.0 : 0.0);
    features.push_back(NetworkUtils::isUDPProtocol(flow->protocol) ? 1.0 : 0.0);

    // Traffic features
    features.push_back(flow->packet_rate);
    features.push_back(flow->byte_rate);
    features.push_back(flow->average_packet_size);

    return features;
}
```

💡 XIII. BEST PRACTICES & NOTES

1. Thread Safety

- ✅ `NetworkStatsCollector` sử dụng `std::mutex`
- ✅ Static functions là stateless → thread-safe

2. Error Handling

cpp Sao chép

```
// Safe conversion với validation
if (!NetworkUtils::isValidIPv4(ip_str)) {
    LOG_ERROR("Invalid IP: " + ip_str);
    return false;
}
```

3. Performance

- ✅ `ipStringToInt()` → Fast integer comparison
- ✅ `calculateSubnetMask()` → Bitwise operations

- ⚠️ `expandCIDR()` → Giới hạn 65K IPs

4. Security

```cpp
// Input validation trước khi xử lý
if (!NetworkUtils::isValidIPv4(user_input)) {
    return ERROR;
}
```

📊 XIV. BẢNG TÓM TẮT NHANH

Hàm	Input	Output	Ú
`isValidIPv4()`	"192.168.1.1"	`true`	V
`ipStringToInt()`	"192.168.1.1"	`3232235777`	F
`isPrivateIP()`	"10.0.0.1"	`true`	T c
`isIPInCIDR()`	IP, CIDR	`bool`	V
`getPortService()`	`443`	"HTTPS"	S ic
`getProtocolName()`	`6`	"TCP"	P
`normalizeMACAddress()`	MAC	"XX:XX:XX:XX:XX:XX"	V
`getInterfaceIP()`	"eth0"	"192.168.1.1"	Ir n
`calculatePacketRate()`	count, time	`pps`	P n

2 Nguồn

Các hàm này tạo thành **network processing layer** cho hệ thống IPS của bạn, cung cấp tất cả công cụ cần thiết để phân tích và xử lý traffic ở mức ...t layer! 🚀

Giải thích chi tiết PacketParser - Bộ phận tích gói tin mạng

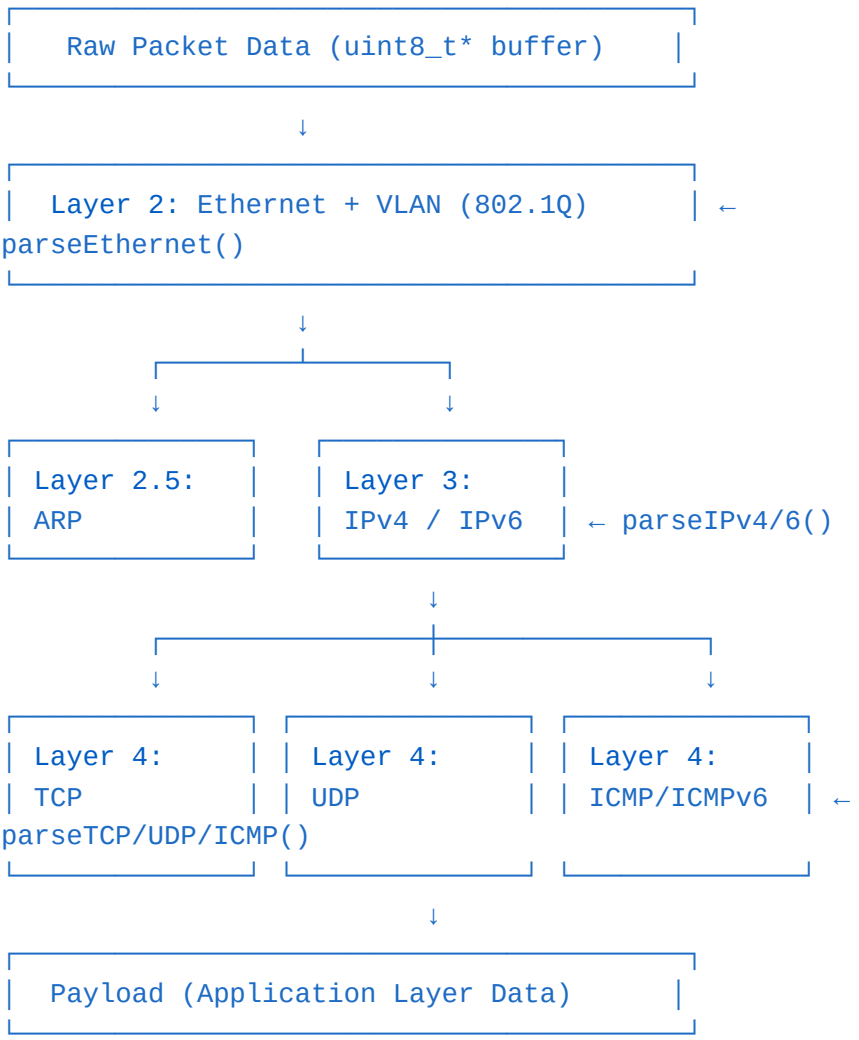
`PacketParser` là **thành phần cốt lõi** trong hệ thống **NetworkSecurityAI**, chịu trách nhiệm phân tích (parse) các gói tin mạng từ raw bytes thành cấu

trúc dữ liệu có tổ chức theo mô hình OSI.

📋 I. TỔNG QUAN KIẾN TRÚC

Mô hình phân tích theo tầng OSI

yaml Sao chép



🏗️ II. CẤU TRÚC DỮ LIỆU CHÍNH

1. ParsedPacket - Container chứa toàn bộ thông tin

cpp Sao chép

```cpp
struct ParsedPacket {
    // === METADATA ===
    uint64_t timestamp;          // Thời gian capture
(microseconds)
    size_t packet_size;          // Kích thước gói tin
```

```
    ProtocolType protocol_type;  // Loại protocol cao nhất

    // === LAYER 2: ETHERNET ===
    bool has_ethernet;
    EthernetHeader ethernet;     // MAC addresses, VLAN,
EtherType

    // === LAYER 2.5: ARP ===
    bool has_arp;
    ARPHeader arp;               // ARP request/reply

    // === LAYER 3: NETWORK ===
    bool has_ipv4;
    IPv4Header ipv4;             // IP addresses, TTL, flags

    bool has_ipv6;
    IPv6Header ipv6;             // IPv6 addresses, flow
label

    // === LAYER 4: TRANSPORT ===
    bool has_tcp;
    TCPHeader tcp;               // Ports, seq/ack, flags,
window

    bool has_udp;
    UDPHeader udp;               // Ports, length

    bool has_icmp;
    ICMPHeader icmp;             // Type, code, echo data

    // === PAYLOAD ===
    const uint8_t* payload;      // Pointer to application
data
    size_t payload_length;       // Payload size

    // === QUICK ACCESS FIELDS ===
    uint32_t src_ip, dst_ip;     // Fast access (IPv4)
    uint16_t src_port, dst_port; // Fast access (TCP/UDP)
    uint8_t tcp_flags;           // Fast flag checking
};
```

Tại sao có Quick Access Fields?

- ✅ **Performance**: Không cần kiểm tra `has_tcp` → truy cập trực tiếp
- ✅ **Backward compatibility**: Code cũ không cần refactor
- ✅ **Convenience**: `packet.src_ip` thay vì `packet.ipv4.src_ip`

2. ProtocolType Enum

cpp Sao chép

```cpp
enum class ProtocolType {
    UNKNOWN = 0,
    ETHERNET,    // Layer 2
    ARP,         // Layer 2.5
    IPV4,        // Layer 3
    IPV6,        // Layer 3
    TCP,         // Layer 4
    UDP,         // Layer 4
    ICMP,        // Layer 4
    ICMPV6,      // Layer 4
    IGMP, ESP, AH, SCTP, GRE  // Other protocols
};
```

Ứng dụng:

cpp Sao chép

```cpp
switch (packet.protocol_type) {
    case ProtocolType::TCP:
        analyzeTCPTraffic(packet);
        break;
    case ProtocolType::UDP:
        analyzeUDPTraffic(packet);
        break;
}
```

🔍 III. CÁC HÀM PARSE CHÍNH

1. parsePacket() - Entry Point

cpp Sao chép

```cpp
bool parsePacket(const uint8_t* data, size_t length,
ParsedPacket& parsed)
```

Workflow:

cpp Sao chép

```cpp
// 1. Initialize
parsed = ParsedPacket();
parsed.timestamp = Utils::getCurrentTimestampUs();
parsed.packet_size = length;

// 2. Parse Ethernet
```

```
if (!parseEthernet(data, length, parsed, offset)) return
false;

// 3. Parse based on EtherType
switch (ntohs(parsed.ethernet.ether_type)) {
    case ETH_P_IP:    // 0x0800
        parseIPv4() → parseTCP/UDP/ICMP()
        break;
    case ETH_P_IPV6:  // 0x86DD
        parseIPv6() → parseTCP/UDP/ICMPv6()
        break;
    case ETH_P_ARP:   // 0x0806
        parseARP()
        break;
}

// 4. Set payload
parsed.payload = data + offset;
parsed.payload_length = length - offset;

// 5. Copy quick access fields
copyQuickAccessFields(parsed);
```

Ứng dụng:

cpp Sao chép

```
// DPDK Dispatcher
void processBatch(struct rte_mbuf** pkts, uint16_t nb_pkts)
{
    for (int i = 0; i < nb_pkts; i++) {
        uint8_t* pkt_data = rte_pktmbuf_mtod(pkts[i],
uint8_t*);
        size_t pkt_len = rte_pktmbuf_pkt_len(pkts[i]);

        ParsedPacket parsed;
        if (parser.parsePacket(pkt_data, pkt_len, parsed)) {
            processPacket(parsed);
        }
    }
}
```

2. parseEthernet() - Layer 2

cpp Sao chép

```
bool parseEthernet(const uint8_t* data, size_t length,
                   ParsedPacket& parsed, size_t& offset)
```

Cấu trúc Ethernet Frame:

scss Sao chép

```
┌──────────────┬──────────────┬──────────┬──────────┬─────┐
| Dst MAC (6B) | Src MAC (6B) | Type (2B) | Payload  | FCS |
└──────────────┴──────────────┴──────────┴──────────┴─────┘
```

Xử lý VLAN (802.1Q):

cpp Sao chép

```cpp
// Check for VLAN tag
if (ntohs(parsed.ethernet.ether_type) == ETH_P_8021Q) {  // 0x8100
    parsed.ethernet.has_vlan = true;

    // Parse VLAN tag (4 bytes)
    // ┌──────────┬────┬──────────────┬──────────────┐
    // | PCP (3) | DEI| VID (12)     | EtherType (16)|
    // └──────────┴────┴──────────────┴──────────────┘
    uint16_t tci = ntohs(vlan_tag[0]);
    parsed.ethernet.vlan_priority = (tci >> 13) & 0x07;
    parsed.ethernet.vlan_id = tci & 0x0FFF;
    parsed.ethernet.ether_type = vlan_tag[1];  // Real EtherType

    offset += 4;
}
```

Validation:

cpp Sao chép

```cpp
bool validateEthernet(const struct ethhdr* eth_header, size_t length) {
    uint16_t eth_type = ntohs(eth_header->h_proto);

    // Known EtherTypes
    switch (eth_type) {
        case ETH_P_IP:      // 0x0800
        case ETH_P_IPV6:    // 0x86DD
        case ETH_P_ARP:     // 0x0806
        case ETH_P_8021Q:   // 0x8100 (VLAN)
            return true;
        default:
            return eth_type >= 0x0600;  // Valid range
```

```
    }
}
```

Ứng dụng:

cpp Sao chép

```
// VLAN-based filtering
if (parsed.ethernet.has_vlan) {
    if (parsed.ethernet.vlan_id == MANAGEMENT_VLAN) {
        priority = HIGH;
    } else if (parsed.ethernet.vlan_id == GUEST_VLAN) {
        apply_rate_limiting();
    }
}
```

3. parseIPv4() - Layer 3

cpp Sao chép

```
bool parseIPv4(const uint8_t* data, size_t length,
               ParsedPacket& parsed, size_t& offset)
```

Cấu trúc IPv4 Header:

css Sao chép

```
 0                   1                   2
3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0 1
┌────────┬────────┬──────────────────┬──────────────────────────────────
│Version│  IHL  │      TOS       │         Total Length
│
├────────┴────────┴──────────────────┼──────────────────┬──────────────
│        Identification          │Flags│  Fragment Offset
│
├──────────────────┬─────────────────┼──────────────────┴──────────────
│      TTL       │   Protocol    │       Header Checksum
│
├──────────────────┴─────────────────┴──────────────────────────────────
│                     Source IP Address
│
├──────────────────────────────────────────────────────────────────────
│                   Destination IP Address
```

```
|
└──────────────────────────────────────────────────────────────────────────────
```

Extract Flags & Fragment:

cpp Sao chép

```cpp
void extractIPv4Flags(const struct iphdr* ip_header,
IPv4Header& ipv4) {
    uint16_t flags_and_frag = ntohs(ip_header->frag_off);

    // Flags (3 bits)
    // ┌─────┬─────┬─────┐
    // │ 0   │ DF  │ MF  │
    // └─────┴─────┴─────┘
    ipv4.dont_fragment = (flags_and_frag & 0x4000) != 0;  //
DF bit
    ipv4.more_fragments = (flags_and_frag & 0x2000) != 0; //
MF bit

    // Fragment offset (13 bits)
    ipv4.fragment_offset = flags_and_frag & 0x1FFF;

    // Is fragmented?
    ipv4.is_fragmented = (ipv4.fragment_offset != 0) ||
ipv4.more_fragments;
}
```

Validation:

cpp Sao chép

```cpp
bool validateIPv4(const struct iphdr* ip_header, size_t
length) {
    // Check version
    if (ip_header->version != 4) return false;

    // Check header length
    size_t header_len = ip_header->ihl * 4;  // IHL in 32-
bit words
    if (header_len < 20 || header_len > length) return
false;

    // Check total length
    uint16_t total_len = ntohs(ip_header->tot_len);
    if (total_len < header_len) return false;

    // Check TTL
```

```
    if (ip_header->ttl == 0) return false;

    return true;
}
```

Ứng dụng:

cpp Sao chép

```
// Fragment reassembly detection
if (parsed.ipv4.is_fragmented) {
    FragmentKey key = {
        parsed.ipv4.src_ip,
        parsed.ipv4.dst_ip,
        parsed.ipv4.identification
    };

    fragment_tracker[key].push_back(parsed);

    if (!parsed.ipv4.more_fragments) {
        // Last fragment - reassemble
        reassembleFragments(key);
    }
}

// TTL anomaly detection
if (parsed.ipv4.ttl < 10) {
    alert("Suspicious low TTL - possible spoofing");
}
```

4. parseTCP() - Layer 4

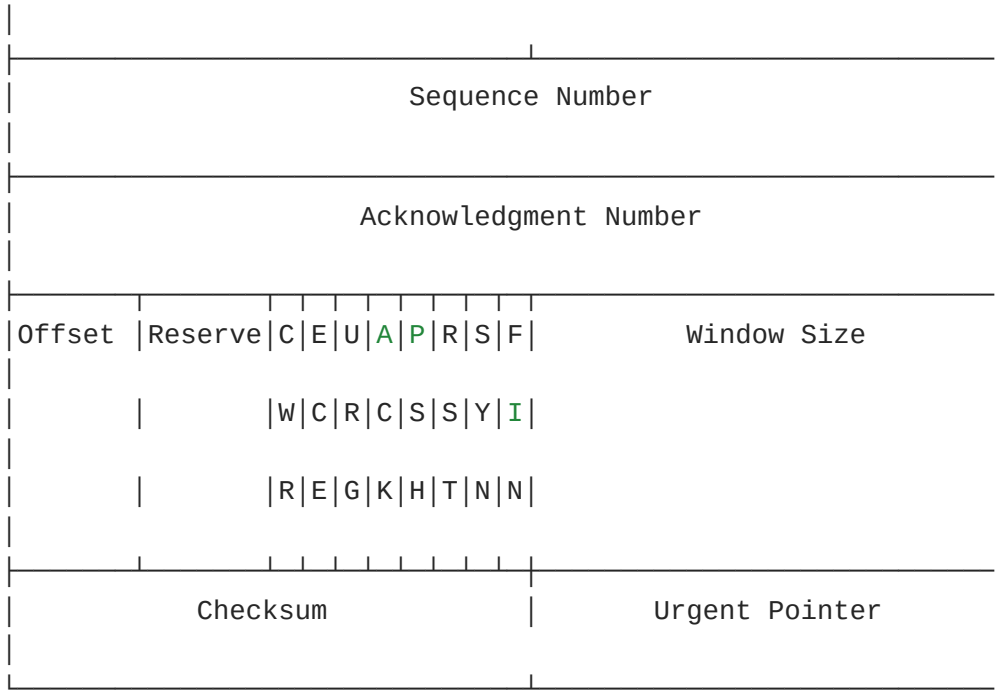
cpp Sao chép

```
bool parseTCP(const uint8_t* data, size_t length,
              ParsedPacket& parsed, size_t& offset)
```

Cấu trúc TCP Header:

css Sao chép

```
 0                   1                   2
3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0 1
┌─────────────────────────────────┬─────────────────────────────────┐
│          Source Port            │       Destination Port
```



Extract TCP Flags:

cpp Sao chép

```cpp
void extractTCPFlags(const struct tcphdr* tcp_header,
TCPHeader& tcp) {
    // Extract from struct
    tcp.flag_fin = tcp_header->fin;
    tcp.flag_syn = tcp_header->syn;
    tcp.flag_rst = tcp_header->rst;
    tcp.flag_psh = tcp_header->psh;
    tcp.flag_ack = tcp_header->ack;
    tcp.flag_urg = tcp_header->urg;

    // ECE and CWR from raw bytes (byte 13)
    const uint8_t* tcp_bytes = reinterpret_cast<const
uint8_t*>(tcp_header);
    uint8_t flags_byte = tcp_bytes[13];
    tcp.flag_ece = (flags_byte & 0x40) != 0;
    tcp.flag_cwr = (flags_byte & 0x80) != 0;

    // Build combined flags byte
    tcp.flags = 0;
    if (tcp.flag_fin) tcp.flags |= 0x01;
    if (tcp.flag_syn) tcp.flags |= 0x02;
    if (tcp.flag_rst) tcp.flags |= 0x04;
    if (tcp.flag_psh) tcp.flags |= 0x08;
    if (tcp.flag_ack) tcp.flags |= 0x10;
    if (tcp.flag_urg) tcp.flags |= 0x20;
    if (tcp.flag_ece) tcp.flags |= 0x40;
```

```
    if (tcp.flag_cwr) tcp.flags |= 0x80;
}
```

TCP Options Parsing:

cpp Sao chép

```
// Calculate TCP header length
size_t tcp_header_len = tcp_header->doff * 4;  // Data
offset in 32-bit words

// Extract options if present
if (tcp_header_len > sizeof(struct tcphdr)) {
    parsed.tcp.has_options = true;
    size_t options_len = tcp_header_len - sizeof(struct
tcphdr);

    // Copy options (max 40 bytes)
    parsed.tcp.options_length = std::min(options_len,
sizeof(parsed.tcp.options));
    std::memcpy(parsed.tcp.options,
                data + offset + sizeof(struct tcphdr),
                parsed.tcp.options_length);
}
```

Validation:

cpp Sao chép

```
bool validateTCP(const struct tcphdr* tcp_header, size_t
length) {
    // Check header length
    size_t header_len = tcp_header->doff * 4;
    if (header_len < 20 || header_len > length) return
false;

    // Check ports
    if (ntohs(tcp_header->source) == 0 || ntohs(tcp_header-
>dest) == 0) {
        return false;
    }

    // Check invalid flag combinations
    bool syn = tcp_header->syn;
    bool fin = tcp_header->fin;
    bool rst = tcp_header->rst;

    if (rst && (syn || fin)) return false;  // RST + SYN/FIN
invalid
    if (syn && fin) return false;           // SYN + FIN
```

```
invalid

    return true;
}
```

Ứng dụng:

cpp Sao chép

```
// TCP state tracking
if (parsed.tcp.flag_syn && !parsed.tcp.flag_ack) {
    // SYN - New connection
    flow_state = SYN_SENT;
} else if (parsed.tcp.flag_syn && parsed.tcp.flag_ack) {
    // SYN-ACK - Connection response
    flow_state = SYN_RECEIVED;
} else if (parsed.tcp.flag_fin) {
    // FIN - Connection closing
    flow_state = FIN_WAIT;
}

// Attack detection
if (parsed.tcp.flag_syn && parsed.tcp.flag_fin) {
    alert("SYN-FIN scan detected");  // Invalid combination
}

if (parsed.tcp.flag_rst && parsed.tcp.window_size == 0) {
    alert("Possible TCP RST attack");
}
```

5. parseUDP() - Layer 4

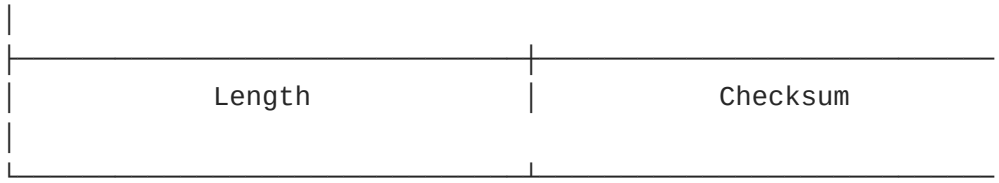
cpp Sao chép

```
bool parseUDP(const uint8_t* data, size_t length,
              ParsedPacket& parsed, size_t& offset)
```

Cấu trúc UDP Header (đơn giản hơn TCP):

Sao chép

```
 0                   1                   2
3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0 1
┌─────────────────────────────────┬─────────────────────────────────┐
│          Source Port            │       Destination Port
```



Validation:

cpp Sao chép

```cpp
bool validateUDP(const struct udphdr* udp_header, size_t
length) {
    // Check destination port
    if (ntohs(udp_header->dest) == 0) return false;

    // Check length
    uint16_t udp_len = ntohs(udp_header->len);
    if (udp_len < 8 || udp_len > length) return false;

    return true;
}
```

Ứng dụng:

cpp Sao chép

```cpp
// DNS detection
if (parsed.udp.dst_port == 53 || parsed.udp.src_port == 53)
{
    parseDNS(parsed.payload, parsed.payload_length);
}

// DHCP detection
if (parsed.udp.dst_port == 67 || parsed.udp.dst_port == 68)
{
    parseDHCP(parsed.payload, parsed.payload_length);
}

// UDP flood detection
if (udp_packet_rate > threshold) {
    alert("Possible UDP flood attack");
}
```

6. parseICMP() - Layer 4

cpp Sao chép

```
bool parseICMP(const uint8_t* data, size_t length,
               ParsedPacket& parsed, size_t& offset)
```

Cấu trúc ICMP Header:

```
css
```

Sao chép

```
 0                   1                   2
3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0 1
┌─────────────────┬─────────────────┬─────────────────────────────
│      Type       │      Code       │           Checksum
│
├─────────────────┴─────────────────┴─────────────────────────────
│                    Rest of Header (varies)
│
└─────────────────────────────────────────────────────────────────

// For Echo Request/Reply (Type 8/0):
┌─────────────────────────────────┬─────────────────────────────
│         Identifier              │        Sequence Number
│
└─────────────────────────────────┴─────────────────────────────
```

ICMP Types:

```
cpp
```

Sao chép

```
std::string icmpTypeToString(uint8_t type) {
    switch (type) {
        case 0:  return "Echo Reply";
        case 3:  return "Destination Unreachable";
        case 5:  return "Redirect";
        case 8:  return "Echo Request";
        case 11: return "Time Exceeded";
        case 13: return "Timestamp Request";
        case 14: return "Timestamp Reply";
        default: return "Unknown (" + std::to_string(type) +
")";
    }
}
```

Ứng dụng:

```
cpp
```

Sao chép

```
// Ping detection
if (parsed.icmp.type == ICMP_ECHO) {
    LOG_INFO("Ping request from " <<
ipv4ToString(parsed.ipv4.src_ip));
}

// ICMP flood detection
if (icmp_packet_count > 1000 && parsed.icmp.type ==
ICMP_ECHO) {
    alert("Possible ICMP flood (Ping flood)");
}

// Traceroute detection
if (parsed.icmp.type == ICMP_TIME_EXCEEDED) {
    LOG_INFO("Traceroute detected");
}
```

7. parseARP() - Layer 2.5

cpp Sao chép

```
bool parseARP(const uint8_t* data, size_t length,
              ParsedPacket& parsed, size_t& offset)
```

Cấu trúc ARP Packet:

scss Sao chép

```
 0                   1                   2
3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0 1
┌─────────────────────────────────┬─────────────────────────────────
│       Hardware Type (1)         │       Protocol Type
(0x0800)   │
├─────────────────┬───────────────┼─────────────────────────────────
│  HW Size (6)    │ Proto Size(4) │          Opcode (1 or 2)
│
├─────────────────┴───────────────┴─────────────────────────────────
│                    Sender MAC Address (6 bytes)
│
├──────────────────────────────────────────────────────────────────
│                    Sender IP Address (4 bytes)
│
├──────────────────────────────────────────────────────────────────
│                    Target MAC Address (6 bytes)
│
├──────────────────────────────────────────────────────────────────
```

```
|                    Target IP Address (4 bytes)
|
└──────────────────────────────────────────────────────────────────
```

ARP Opcodes:

cpp Sao chép

```cpp
std::string arpOpcodeToString(uint16_t opcode) {
    switch (opcode) {
        case 1: return "ARP Request";
        case 2: return "ARP Reply";
        case 3: return "RARP Request";
        case 4: return "RARP Reply";
        default: return "Unknown";
    }
}
```

Ứng dụng:

cpp Sao chép

```cpp
// ARP spoofing detection
if (parsed.arp.opcode == 2) {  // ARP Reply
    std::string sender_ip =
ipv4ToString(parsed.arp.sender_ip);
    std::string sender_mac =
macToString(parsed.arp.sender_mac);

    if (arp_cache[sender_ip] != "" && arp_cache[sender_ip]
!= sender_mac) {
        alert("ARP spoofing detected! IP " + sender_ip +
              " changed MAC from " + arp_cache[sender_ip] +
              " to " + sender_mac);
    }

    arp_cache[sender_ip] = sender_mac;
}

// Gratuitous ARP detection
if (parsed.arp.opcode == 2 &&
    parsed.arp.sender_ip == parsed.arp.target_ip) {
    LOG_INFO("Gratuitous ARP detected");
}
```

🛠️ IV. UTILITY FUNCTIONS

1. Conversion Functions

ipv4ToString() / ipv6ToString()

cpp Sao chép

```
std::string ip_str =
PacketParser::ipv4ToString(parsed.ipv4.src_ip);
// "192.168.1.100"

std::string ip6_str =
PacketParser::ipv6ToString(parsed.ipv6.src_ip);
// "2001:0db8:85a3::8a2e:0370:7334"
```

macToString()

cpp Sao chép

```
std::string mac =
PacketParser::macToString(parsed.ethernet.src_mac);
// "00:1a:2b:3c:4d:5e"
```

tcpFlagsToString()

cpp Sao chép

```
std::string flags =
PacketParser::tcpFlagsToString(parsed.tcp.flags);
// "SYN|ACK" or "PSH|ACK" or "FIN|ACK"
```

2. getPacketSummary() - One-line Description

cpp Sao chép

```
std::string summary =
PacketParser::getPacketSummary(parsed);
```

Ví dụ output:

ini Sao chép

```
[TCP] 192.168.1.100 -> 8.8.8.8 [54321 -> 443] Flags: SYN (60
bytes)
[UDP] 10.0.0.5 -> 10.0.0.1 [12345 -> 53] (512 bytes)
```

```
[ICMP] 192.168.1.1 -> 192.168.1.100 Echo Request (84 bytes)
[ARP] ARP Request 192.168.1.100 -> 192.168.1.1
```

Ứng dụng trong logging:

cpp Sao chép

```cpp
// Quick logging
LOG_INFO(PacketParser::getPacketSummary(parsed));

// Dashboard display
dashboard.addPacketLog(PacketParser::getPacketSummary(parsed)
```

🎯 V. ỨNG DỤNG TRONG HỆ THỐNG NETWORKSECURITYAI

1. DPDK Dispatcher (Layer 1)

cpp Sao chép

```cpp
// dpdk_dispatcher.cpp
void DPDKDispatcher::processBatch(struct rte_mbuf** pkts,
uint16_t nb_pkts) {
    PacketParser parser;

    for (uint16_t i = 0; i < nb_pkts; i++) {
        // Get packet data
        uint8_t* pkt_data = rte_pktmbuf_mtod(pkts[i],
uint8_t*);
        size_t pkt_len = rte_pktmbuf_pkt_len(pkts[i]);

        // Parse packet
        ParsedPacket parsed;
        if (!parser.parsePacket(pkt_data, pkt_len, parsed))
{
            stats_.malformed_packets++;
            continue;
        }

        // Quick filtering
        if (blacklist_.contains(parsed.src_ip)) {
            rte_pktmbuf_free(pkts[i]);
            stats_.blocked_packets++;
            continue;
        }
```

```
        // Send to next stage
        flow_assembly_->processPacket(parsed);
    }
}
```

2. Flow Assembly (Layer 1)

cpp Sao chép

```
// flow_assembly.cpp
Flow* FlowAssembly::assembleFlow(const ParsedPacket& packet)
{
    // Create flow key
    FlowKey key;
    key.src_ip = packet.src_ip;
    key.dst_ip = packet.dst_ip;
    key.src_port = packet.src_port;
    key.dst_port = packet.dst_port;
    key.protocol = packet.ip_protocol;

    // Find or create flow
    Flow* flow = flow_table_[key];
    if (!flow) {
        flow = new Flow(key);
        flow_table_[key] = flow;
    }

    // Update flow statistics
    flow->packet_count++;
    flow->byte_count += packet.packet_size;
    flow->last_seen = packet.timestamp;

    // TCP state tracking
    if (packet.has_tcp) {
        updateTCPState(flow, packet.tcp);
    }

    return flow;
}
```

3. Signature Matching (Layer 1)

cpp Sao chép

```
// signature_matcher.cpp
bool SignatureMatcher::matchPacket(const ParsedPacket&
packet) {
    // HTTP detection
```

```
    if (packet.has_tcp && (packet.dst_port == 80 ||
packet.dst_port == 8080)) {
        if (packet.payload_length > 0) {
            std::string payload(reinterpret_cast<const
char*>(packet.payload),
                                packet.payload_length);

            // Check for SQL injection patterns
            if (aho_corasick_.search(payload,
sql_injection_patterns)) {
                alert("SQL injection attempt detected");
                return true;
            }
        }
    }

    // DNS tunneling detection
    if (packet.has_udp && packet.dst_port == 53) {
        if (packet.payload_length > 512) {
            alert("Suspicious large DNS query - possible
tunneling");
            return true;
        }
    }

    return false;
}
```

4. Feature Extraction (Layer 2 AI)

cpp Sao chép

```
// feature_extractor.cpp
std::vector<double> FeatureExtractor::extractFeatures(const
ParsedPacket& packet) {
    std::vector<double> features;

    // === NETWORK FEATURES ===
    features.push_back(packet.packet_size);
    features.push_back(packet.payload_length);
    features.push_back(packet.ip_ttl);

    // === PROTOCOL FEATURES ===
    features.push_back(packet.protocol_type ==
ProtocolType::TCP ? 1.0 : 0.0);
    features.push_back(packet.protocol_type ==
ProtocolType::UDP ? 1.0 : 0.0);

    // === TCP FEATURES ===
    if (packet.has_tcp) {
```

```
        features.push_back(packet.tcp.window_size);
        features.push_back(packet.tcp.flag_syn ? 1.0 : 0.0);
        features.push_back(packet.tcp.flag_ack ? 1.0 : 0.0);
        features.push_back(packet.tcp.flag_fin ? 1.0 : 0.0);
        features.push_back(packet.tcp.flag_rst ? 1.0 : 0.0);
    } else {
        features.insert(features.end(), 5, 0.0);
    }

    // === PORT FEATURES ===

features.push_back(NetworkUtils::isWellKnownPort(packet.dst_p
? 1.0 : 0.0);

    return features;
}
```

5. Logging & Dashboard

cpp Sao chép

```
// logger.cpp
void Logger::logPacket(const ParsedPacket& packet, const
std::string& action) {
    std::stringstream log;

    log << Utils::getCurrentTimestamp() << " ";
    log << "[" << action << "] ";
    log << PacketParser::getPacketSummary(packet);

    if (packet.has_tcp) {
        log << " Seq=" << packet.tcp.seq_number;
        log << " Ack=" << packet.tcp.ack_number;
        log << " Win=" << packet.tcp.window_size;
    }

    writeLog(log.str());
}

// dashboard_controller.cpp
void DashboardController::updatePacketStats(const
ParsedPacket& packet) {
    stats_.total_packets++;
    stats_.total_bytes += packet.packet_size;

    switch (packet.protocol_type) {
        case ProtocolType::TCP:
            stats_.tcp_packets++;
            break;
```

```
            case ProtocolType::UDP:
                stats_.udp_packets++;
                break;
            case ProtocolType::ICMP:
                stats_.icmp_packets++;
                break;
        }

        // Update real-time chart
        dashboard_.addDataPoint(packet.timestamp,
    packet.packet_size);
    }
```

📊 VI. PERFORMANCE CONSIDERATIONS

1. Zero-Copy Design

cpp Sao chép

```
// Payload không được copy - chỉ lưu pointer
parsed.payload = data + offset;  // ✅ Fast
// NOT: parsed.payload = new uint8_t[payload_length]; ❌
Slow
```

2. Quick Access Fields

cpp Sao chép

```
// Fast access without branching
if (packet.src_ip == blacklisted_ip) {  // ✅ Direct access
    drop();
}

// vs slower version
if (packet.has_ipv4 && packet.ipv4.src_ip == blacklisted_ip)
{  // ❌ Branch
    drop();
}
```

3. Validation Early Exit

cpp Sao chép

```
// Fail fast
if (!validateEthernet(...)) return false;  // Stop
immediately
```

```
if (!validateIPv4(...)) return false;
if (!validateTCP(...)) return false;
```

💡 VII. BEST PRACTICES

*1. Always Check has_ Flags**

cpp Sao chép

```
// ✅ Correct
if (parsed.has_tcp) {
    uint16_t port = parsed.tcp.dst_port;
}

// ❌ Wrong - may access uninitialized data
uint16_t port = parsed.tcp.dst_port;
```

2. Handle Fragmented Packets

cpp Sao chép

```
if (parsed.is_fragmented) {
    // Don't analyze payload until reassembled
    return;
}
```

3. Validate Before Use

cpp Sao chép

```
if (parsed.payload_length > 0 && parsed.payload != nullptr)
{
    analyzePayload(parsed.payload, parsed.payload_length);
}
```

🎓 VIII. TÓM TẮT

Component	Purpose	Key Features
parsePacket()	Entry point	Layer-by-layer parsing
parseEthernet()	Layer 2	MAC, VLAN support
parseIPv4/6()	Layer 3	IP addresses, fragmentation

2 Nguồn

Component	Purpose	Key Features
parseTCP()	Layer 4	Flags, seq/ack, options
parseUDP()	Layer 4	Ports, length
parseICMP()	Layer 4	Type, code, echo data
parseARP()	Layer 2.5	MAC-IP mapping
Utilities	Helpers	String conversion, validation

Giải thích chi tiết FlowManager - Quản lý luồng mạng

`FlowManager` là thành phần **quan trọng nhất** trong **Layer 1 (IPS)**, chịu trách nhiệm **tracking và phân tích các network flows** (luồng mạng) - tức là chuỗi các packets liên quan đến nhau trong một phiên giao tiếp.

PacketParser là **nền tảng** cho toàn bộ hệ thống phân tích traffic, cung cấp cấu trúc dữ liệu chuẩn hóa cho tất cả các module xử lý tiếp theo! 🚀

📋 I. KHÁI NIỆM NETWORK FLOW

Flow là gì?

Network Flow = Tập hợp các packets có cùng **5-tuple**:

```
css                                                    Sao chép

┌──────────────────────────────────────────────────────┐
│  Flow = (Src IP, Dst IP, Src Port, Dst Port, Proto)  │
└──────────────────────────────────────────────────────┘
```

Ví dụ:

```
yaml                                                   Sao chép

Flow 1: 192.168.1.100:54321 → 8.8.8.8:443 (TCP)
  - Packet 1: SYN
  - Packet 2: SYN-ACK
  - Packet 3: ACK
  - Packet 4-100: Data transfer
  - Packet 101: FIN

Flow 2: 10.0.0.5:12345 → 10.0.0.1:53 (UDP)
  - Packet 1: DNS Query
  - Packet 2: DNS Response
```

Tại sao cần Flow Management?

- ✅ **Stateful inspection**: Theo dõi trạng thái kết nối (TCP handshake, teardown)

- ✅ **Attack detection**: Phát hiện port scan, DDoS, data exfiltration
- ✅ **Traffic analysis**: Tính toán throughput, latency, packet loss
- ✅ **Protocol identification**: Nhận dạng ứng dụng (HTTP, SSH, DNS...)

🏗️ II. CẤU TRÚC DỮ LIỆU

1. FlowKey - Flow Identifier

cpp Sao chép

```cpp
struct FlowKey {
    uint32_t src_ip;      // Source IP (network byte order)
    uint32_t dst_ip;      // Destination IP
    uint16_t src_port;    // Source port
    uint16_t dst_port;    // Destination port
    uint8_t protocol;     // Protocol (TCP=6, UDP=17,
ICMP=1)

    bool operator==(const FlowKey& other) const;
    size_t hash() const;
};
```

Hash Function Implementation

cpp Sao chép

```cpp
size_t FlowKey::hash() const {
    size_t h1 = std::hash<uint32_t>{}(src_ip);
    size_t h2 = std::hash<uint32_t>{}(dst_ip);
    size_t h3 = std::hash<uint16_t>{}(src_port);
    size_t h4 = std::hash<uint16_t>{}(dst_port);
    size_t h5 = std::hash<uint8_t>{}(protocol);

    // XOR combination with bit shifts
    return h1 ^ (h2 << 1) ^ (h3 << 2) ^ (h4 << 3) ^ (h5 <<
4);
}
```

Tại sao dùng hash?

- ✅ **Fast lookup**: O(1) average time trong `unordered_map`
- ✅ **Memory efficient**: Không cần lưu string keys
- ✅ **Collision resistant**: XOR + bit shifts giảm collision

Ứng dụng:

cpp Sao chép

```cpp
// Fast flow lookup
FlowKey key = {src_ip, dst_ip, src_port, dst_port,
protocol};
auto flow = flow_table_[key];  // O(1) lookup
```

2. FlowInfo - Flow Metadata

cpp Sao chép

```cpp
struct FlowInfo {
    // === IDENTIFICATION ===
    FlowKey key;                    // 5-tuple identifier

    // === TIMING ===
    uint64_t first_seen;            // First packet
timestamp (μs)
    uint64_t last_seen;             // Last packet timestamp
(μs)
    uint64_t flow_duration;         // Duration in μs

    // === COUNTERS ===
    uint64_t packet_count;          // Total packets
    uint64_t byte_count;            // Total bytes

    // === STATISTICS ===
    std::vector<uint64_t> packet_intervals;  // Inter-
arrival times
    std::vector<size_t> packet_sizes;        // Packet size
distribution

    // === FLAGS ===
    bool is_bidirectional;          // Traffic in both
directions?
    bool is_suspicious;             // Flagged as
suspicious?
    uint8_t threat_level;           // 0-10 threat score

    // === APPLICATION LAYER ===
    std::string application_protocol;  // HTTP, SSH, DNS,
etc.
    std::vector<uint8_t> payload_sample;  // First 64 bytes
};
```

Giải thích các trường:

Field	Purpose	Example
`first_seen`	Flow start time	`1234567890000` (µs)
`last_seen`	Last activity	`1234567900000` (µs)
`flow_duration`	`last_seen - first_seen`	`10000000` (10s)
`packet_intervals`	Time between packets	`[100, 150, 120, ...]` (µs)
`packet_sizes`	Size distribution	`[60, 1500, 1500, ...]` (bytes)
`is_bidirectional`	Client ↔ Server	`true` (normal), `false` (scan)
`threat_level`	Risk score	`0` (safe) to `10` (critical)

3. FlowKeyHash - Custom Hash Functor

cpp Sao chép

```cpp
struct FlowKeyHash {
    size_t operator()(const FlowKey& key) const {
        return key.hash();
    }
};
```

Ứng dụng trong unordered_map:

cpp Sao chép

```cpp
std::unordered_map<FlowKey, std::shared_ptr<FlowInfo>,
FlowKeyHash> flows_;
```

🔧 III. FLOWMANAGER CLASS

1. Constructor & Configuration

cpp Sao chép

```
FlowManager::FlowManager()
    : max_flows_(100000),           // Max 100K concurrent
flows
      flow_timeout_ms_(300000),     // 5 minutes timeout
      total_packets_(0),
      total_bytes_(0)
{
}
```

Configuration Methods:

cpp Sao chép

```
void setMaxFlows(size_t max_flows);        // Limit memory
usage
void setFlowTimeout(uint64_t timeout_ms);  // Cleanup
interval
```

Ví dụ:

cpp Sao chép

```
FlowManager flow_mgr;
flow_mgr.setMaxFlows(50000);      // Limit to 50K flows
flow_mgr.setFlowTimeout(60000);   // 1 minute timeout
```

2. processPacket() - Core Function

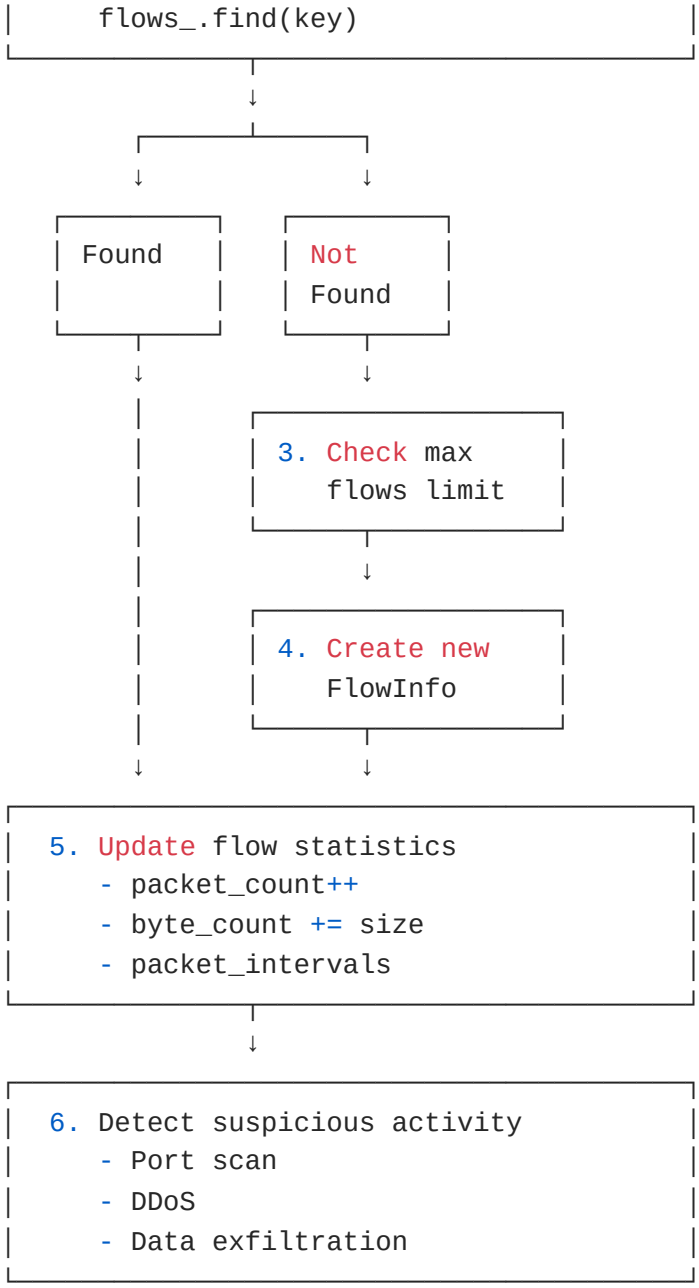
cpp Sao chép

```
std::shared_ptr<FlowInfo> processPacket(const ParsedPacket&
packet)
```

Workflow:

sql Sao chép

```
┌─────────────────────────────────────────┐
│  1. Create FlowKey from packet          │
│     (normalize direction)               │
└─────────────────┬───────────────────────┘
                  ↓
┌─────────────────────────────────────────┐
│  2. Lookup flow in hash table           │
```



3. createFlowKey() - Flow Normalization

cpp Sao chép

```
FlowKey createFlowKey(const ParsedPacket& packet)
```

Bidirectional Flow Matching:

cpp Sao chép

```
// Normalize direction: smaller IP/port first
bool forward = (packet.src_ip < packet.dst_ip) ||
               (packet.src_ip == packet.dst_ip &&
                packet.src_port < packet.dst_port);
```

```
if (forward) {
    key.src_ip = packet.src_ip;
    key.dst_ip = packet.dst_ip;
    key.src_port = packet.src_port;
    key.dst_port = packet.dst_port;
} else {
    // Reverse direction
    key.src_ip = packet.dst_ip;
    key.dst_ip = packet.src_ip;
    key.src_port = packet.dst_port;
    key.dst_port = packet.src_port;
}
```

Tại sao cần normalize?

Ví dụ không normalize:

arduino Sao chép

```
Client → Server: 192.168.1.100:54321 → 8.8.8.8:443
Server → Client: 8.8.8.8:443 → 192.168.1.100:54321

❌ Tạo 2 flows khác nhau!
```

Với normalize:

ini Sao chép

```
Both directions: 8.8.8.8:443 ↔ 192.168.1.100:54321

✅ Cùng 1 flow → is_bidirectional = true
```

4. updateFlowStats() - Statistics Tracking

cpp Sao chép

```
void updateFlowStats(std::shared_ptr<FlowInfo> flow,
                     const ParsedPacket& packet)
```

Update Counters:

cpp Sao chép

```
flow->packet_count++;
flow->byte_count += packet.packet_size;
```

```
flow->last_seen = packet.timestamp;
flow->flow_duration = flow->last_seen - flow->first_seen;
```

Track Packet Intervals:

cpp Sao chép

```
if (flow->last_seen > 0) {
    uint64_t interval = packet.timestamp - flow->last_seen;
    flow->packet_intervals.push_back(interval);

    // Keep only last 100 intervals (memory optimization)
    if (flow->packet_intervals.size() > 100) {
        flow->packet_intervals.erase(flow-
>packet_intervals.begin());
    }
}
```

Track Packet Sizes:

cpp Sao chép

```
flow->packet_sizes.push_back(packet.packet_size);
if (flow->packet_sizes.size() > 100) {
    flow->packet_sizes.erase(flow->packet_sizes.begin());
}
```

Detect Bidirectional Traffic:

cpp Sao chép

```
if (!flow->is_bidirectional) {
    bool is_reverse = (packet.src_ip == flow->key.dst_ip &&
                       packet.dst_ip == flow->key.src_ip);
    if (is_reverse) {
        flow->is_bidirectional = true;
    }
}
```

Capture Payload Sample:

cpp Sao chép

```
if (flow->payload_sample.empty() && packet.payload_length >
0) {
    size_t sample_size = std::min(64,
packet.payload_length);
```

```
    flow->payload_sample.assign(packet.payload,
                                packet.payload +
sample_size);

    identifyApplicationProtocol(flow);
}
```

5. identifyApplicationProtocol() - DPI

cpp Sao chép

```
void identifyApplicationProtocol(std::shared_ptr<FlowInfo>
flow)
```

HTTP Detection:

cpp Sao chép

```
if (std::memcmp(payload.data(), "GET ", 4) == 0 ||
    std::memcmp(payload.data(), "POST", 4) == 0 ||
    std::memcmp(payload.data(), "HTTP", 4) == 0) {
    flow->application_protocol = "HTTP";
    return;
}
```

TLS/SSL Detection:

cpp Sao chép

```
// TLS handshake: 0x16 0x03 0x00-0x03
if (payload[0] == 0x16 && payload[1] == 0x03 &&
    (payload[2] >= 0x00 && payload[2] <= 0x03)) {
    flow->application_protocol = "TLS/SSL";
    return;
}
```

SSH Detection:

cpp Sao chép

```
if (std::memcmp(payload.data(), "SSH-", 4) == 0) {
    flow->application_protocol = "SSH";
    return;
}
```

Port-based Detection:

cpp Sao chép

```cpp
if (flow->key.dst_port == 53 || flow->key.src_port == 53) {
    flow->application_protocol = "DNS";
}
if (flow->key.dst_port == 21) {
    flow->application_protocol = "FTP";
}
if (flow->key.dst_port == 25 || flow->key.dst_port == 587) {
    flow->application_protocol = "SMTP";
}
```

Ứng dụng:

cpp Sao chép

```cpp
// Protocol-based filtering
if (flow->application_protocol == "HTTP" &&
    !isAllowedHTTPHost(flow)) {
    blockFlow(flow);
}

// Detect protocol mismatch
if (flow->key.dst_port == 80 &&
    flow->application_protocol == "SSH") {
    alert("SSH tunneling over HTTP port detected!");
}
```

6. detectSuspiciousActivity() - Threat Detection

cpp Sao chép

```cpp
void detectSuspiciousActivity(std::shared_ptr<FlowInfo>
flow)
```

A. Port Scan Detection

cpp Sao chép

```cpp
// Many packets, small size, unidirectional
if (flow->packet_count > 10 && !flow->is_bidirectional) {
    double avg_size = (double)flow->byte_count / flow-
>packet_count;
    if (avg_size < 100) {
        flow->is_suspicious = true;
```

```
        flow->threat_level = std::max(flow->threat_level,
(uint8_t)3);
    }
}
```

Logic:

- **Port scan** gửi nhiều SYN packets (60-80 bytes)
- Không có response → `is_bidirectional = false`
- Average packet size nhỏ

Ví dụ:

ini Sao chép

```
Attacker → Target:
  192.168.1.100:54321 → 10.0.0.1:22   (SYN, 60 bytes)
  192.168.1.100:54322 → 10.0.0.1:23   (SYN, 60 bytes)
  192.168.1.100:54323 → 10.0.0.1:80   (SYN, 60 bytes)
  ... (100+ packets)

✅ Detected: packet_count=100, avg_size=60,
is_bidirectional=false
```

B. DDoS Detection

cpp Sao chép

```
// High packet rate
if (flow->packet_count > 100 && flow->flow_duration > 0) {
    double pps = (double)(flow->packet_count * 1000) / flow-
>flow_duration;
    if (pps > 1000) {  // > 1000 packets/second
        flow->is_suspicious = true;
        flow->threat_level = std::max(flow->threat_level,
(uint8_t)5);
    }
}
```

Logic:

- **DDoS** tạo ra packet rate cực cao
- Normal traffic: 10-100 pps

- Attack traffic: 1000+ pps

Ví dụ:

ini Sao chép

```
Botnet → Server:
  10.0.0.1:80 receives 50,000 packets in 10 seconds

  pps = (50000 * 1000) / 10000 = 5000 pps

✅ Detected: threat_level=5 (High)
```

C. Covert Channel Detection

cpp Sao chép

```cpp
// Very regular packet intervals
if (flow->packet_intervals.size() > 10) {
    double avg = calculateAverage(flow->packet_intervals);
    double variance = calculateVariance(flow-
>packet_intervals, avg);

    // Low variance = very regular timing
    if (variance < 10 && avg > 0) {
        flow->is_suspicious = true;
        flow->threat_level = std::max(flow->threat_level,
(uint8_t)2);
    }
}
```

Logic:

- **Covert channels** sử dụng timing patterns để truyền data
- Normal traffic có variance cao (user interaction)
- Automated traffic có variance thấp

Ví dụ:

ini Sao chép

```
Normal traffic intervals: [100, 250, 50, 300, 150] µs → High
variance
Covert channel: [100, 100, 100, 100, 100] µs → Low variance
✅ Suspicious
```

D. Data Exfiltration Detection

cpp Sao chép

```cpp
// Large data transfer to uncommon ports
if (flow->byte_count > 10000000 &&  // > 10MB
    flow->key.dst_port > 1024 &&
    flow->key.dst_port != 8080 &&
    flow->key.dst_port != 8443) {
    flow->is_suspicious = true;
    flow->threat_level = std::max(flow->threat_level,
(uint8_t)4);
}
```

Logic:

- **Data exfiltration** thường dùng non-standard ports
- Large data transfer (>10MB) là dấu hiệu đáng ngờ

Ví dụ:

yaml Sao chép

```yaml
Internal Server → External IP:
  10.0.0.5:12345 → 203.0.113.50:9999 (50MB transferred)

✅ Detected: Large transfer to uncommon port 9999
```

7. cleanupExpiredFlows() - Memory Management

cpp Sao chép

```cpp
size_t cleanupExpiredFlows(uint64_t timeout_ms = 300000)  //
5 minutes
```

Implementation:

cpp Sao chép

```cpp
auto current_time =
std::chrono::duration_cast<std::chrono::milliseconds>(
    std::chrono::system_clock::now().time_since_epoch()
).count();
```

```cpp
    size_t removed_count = 0;

    for (auto it = flows_.begin(); it != flows_.end();) {
        if (isFlowExpired(it->second, current_time)) {
            it = flows_.erase(it);
            removed_count++;
        } else {
            ++it;
        }
    }

    return removed_count;
```

Expiration Check:

cpp Sao chép

```cpp
bool isFlowExpired(const std::shared_ptr<FlowInfo>& flow,
                   uint64_t current_time) {
    return (current_time - flow->last_seen) >
flow_timeout_ms_;
}
```

Ứng dụng:

cpp Sao chép

```cpp
// Periodic cleanup (every 60 seconds)
std::thread cleanup_thread([&]() {
    while (running) {

std::this_thread::sleep_for(std::chrono::seconds(60));
        size_t removed = flow_mgr.cleanupExpiredFlows();
        LOG_INFO("Cleaned up " << removed << " expired
flows");
    }
});
```

8. Query Functions

`getFlow()`

cpp Sao chép

```cpp
std::shared_ptr<FlowInfo> getFlow(const FlowKey& key);
```

`getAllFlows()`

cpp Sao chép

```cpp
std::vector<std::shared_ptr<FlowInfo>> getAllFlows();
```

`getSuspiciousFlows()`

cpp Sao chép

```cpp
std::vector<std::shared_ptr<FlowInfo>> getSuspiciousFlows()
{
    std::vector<std::shared_ptr<FlowInfo>> result;

    for (const auto& pair : flows_) {
        if (pair.second->is_suspicious || pair.second-
>threat_level > 0) {
            result.push_back(pair.second);
        }
    }

    // Sort by threat level (highest first)
    std::sort(result.begin(), result.end(),
        [](const auto& a, const auto& b) {
            return a->threat_level > b->threat_level;
        });

    return result;
}
```

Ứng dụng:

cpp Sao chép

```cpp
// Dashboard: Show top threats
auto threats = flow_mgr.getSuspiciousFlows();
for (size_t i = 0; i < std::min(10, threats.size()); i++) {
    auto flow = threats[i];
    dashboard.addThreat(
        ipToString(flow->key.src_ip),
        ipToString(flow->key.dst_ip),
        flow->threat_level
    );
}
```

🎯 IV. ỨNG DỤNG TRONG HỆ THỐNG

1. Integration with DPDK Dispatcher

cpp Sao chép

```cpp
// dpdk_dispatcher.cpp
FlowManager flow_mgr;

void processBatch(struct rte_mbuf** pkts, uint16_t nb_pkts)
{
    for (int i = 0; i < nb_pkts; i++) {
        ParsedPacket parsed;
        parser.parsePacket(pkt_data, pkt_len, parsed);

        // Process flow
        auto flow = flow_mgr.processPacket(parsed);

        // Check if suspicious
        if (flow->is_suspicious) {
            if (flow->threat_level >= 5) {
                // Drop high-threat traffic
                rte_pktmbuf_free(pkts[i]);
                continue;
            } else {
                // Log medium-threat traffic
                logger.logThreat(flow);
            }
        }

        // Forward to next stage
        sendToNextStage(pkts[i]);
    }
}
```

2. Real-time Monitoring Dashboard

cpp Sao chép

```cpp
// dashboard_controller.cpp
void updateFlowStats() {
    auto all_flows = flow_mgr.getAllFlows();

    // Top talkers
    std::sort(all_flows.begin(), all_flows.end(),
        [](const auto& a, const auto& b) {
            return a->byte_count > b->byte_count;
        });

    dashboard.setTopTalkers(all_flows, 10);

    // Protocol distribution
```

```
    std::map<std::string, size_t> proto_dist;
    for (const auto& flow : all_flows) {
        proto_dist[flow->application_protocol]++;
    }
    dashboard.setProtocolDistribution(proto_dist);

    // Active threats
    auto threats = flow_mgr.getSuspiciousFlows();
    dashboard.setActiveThreats(threats);
}
```

3. Automated Response System

cpp Sao chép

```
// response_engine.cpp
void analyzeFlows() {
    auto suspicious = flow_mgr.getSuspiciousFlows();

    for (const auto& flow : suspicious) {
        if (flow->threat_level >= 7) {
            // Critical threat - block immediately
            firewall.blockIP(flow->key.src_ip);
            alert.sendCritical("Blocked IP: " +
ipToString(flow->key.src_ip));
        }
        else if (flow->threat_level >= 4) {
            // Medium threat - rate limit
            firewall.rateLimitIP(flow->key.src_ip, 100);  //
100 pps
            alert.sendWarning("Rate limited IP: " +
ipToString(flow->key.src_ip));
        }
        else {
            // Low threat - monitor
            monitor.watchIP(flow->key.src_ip);
        }
    }
}
```

4. Flow Export (NetFlow/IPFIX)

cpp Sao chép

```
// flow_exporter.cpp
void exportFlows() {
    auto flows = flow_mgr.getAllFlows();
```

```
    for (const auto& flow : flows) {
        NetFlowRecord record;
        record.src_ip = flow->key.src_ip;
        record.dst_ip = flow->key.dst_ip;
        record.src_port = flow->key.src_port;
        record.dst_port = flow->key.dst_port;
        record.protocol = flow->key.protocol;
        record.packets = flow->packet_count;
        record.bytes = flow->byte_count;
        record.start_time = flow->first_seen;
        record.end_time = flow->last_seen;

        netflow_collector.send(record);
    }
}
```

📊 V. PERFORMANCE OPTIMIZATION

1. Memory Management

cpp Sao chép

```
// Limit max flows
setMaxFlows(100000);  // 100K flows max

// Automatic cleanup
if (flows_.size() >= max_flows_) {
    cleanupExpiredFlows();

    // If still at max, remove oldest
    if (flows_.size() >= max_flows_) {
        auto oldest = std::min_element(flows_.begin(),
flows_.end(),
            [](const auto& a, const auto& b) {
                return a.second->last_seen < b.second-
>last_seen;
            });
        flows_.erase(oldest);
    }
}
```

Memory footprint per flow:

python Sao chép

```
FlowKey: 13 bytes
FlowInfo base: ~100 bytes
packet_intervals (100 entries): 800 bytes
```

```
packet_sizes (100 entries): 800 bytes
payload_sample: 64 bytes
Total: ~1.8 KB per flow

100K flows = ~180 MB RAM
```

2. Thread Safety

cpp Sao chép

```cpp
// All public methods use mutex
std::lock_guard<std::mutex> lock(flows_mutex_);
```

Optimization: Use `std::shared_mutex` for read-heavy workloads:

cpp Sao chép

```cpp
// Multiple readers
std::shared_lock<std::shared_mutex> lock(flows_mutex_);
auto flow = flows_[key];

// Single writer
std::unique_lock<std::shared_mutex> lock(flows_mutex_);
flows_[key] = new_flow;
```

3. Hash Table Performance

cpp Sao chép

```cpp
// Reserve capacity to avoid rehashing
flows_.reserve(max_flows_);

// Load factor tuning
flows_.max_load_factor(0.75);  // Default, can tune
```

💡 VI. BEST PRACTICES

1. Regular Cleanup

cpp Sao chép

```cpp
// Run cleanup every 60 seconds
std::thread([&]() {
```

```
    while (running) {
        sleep(60);
        flow_mgr.cleanupExpiredFlows();
    }
}).detach();
```

2. Threat Level Tuning

cpp Sao chép

```
// Adjust thresholds based on network profile
if (is_datacenter) {
    ddos_threshold = 10000;  // Higher for datacenter
} else {
    ddos_threshold = 1000;   // Lower for office network
}
```

3. Flow Export

cpp Sao chép

```
// Export before cleanup
auto flows = flow_mgr.getAllFlows();
exportToSIEM(flows);
flow_mgr.cleanupExpiredFlows();
```

📈 VII. TÓM TẮT

Feature	Purpose	Benefit
5-tuple tracking	Flow identification	Stateful inspection
Bidirectional matching	Normalize flows	Accurate statistics
Packet intervals	Timing analysis	Covert channel detection
Payload sampling	Protocol identification	DPI capabilities
Threat scoring	Risk assessment	Automated response
Memory limits	Resource control	Scalability

2 Nguồn

Feature	Purpose	Benefit
Automatic cleanup	Prevent memory leak	Long-term stability

Giải thích chi tiết ConfigManager - Quản lý cấu hình hệ thống

`ConfigManager` là **singleton component** chịu trách nhiệm **quản lý toàn FlowManager** là **trái tim** của hệ thống IPS, cung cấp **stateful tracking** và **bộ cấu hình** của hệ thống NetworkSecurityAI, hỗ trợ **đa nguồn cấu hình behavioral analysis** cho tất cả network traffic! 🚀
(file, environment, command-line) với **thread-safety** và **hot-reload**.

📋 I. KIẾN TRÚC TỔNG QUAN

Configuration Sources Priority

scss Sao chép

```
┌─────────────────────────────────────────────────┐
│         Configuration Loading Priority          │
│  (Higher number = Higher priority, overrides lower) │
└─────────────────────────────────────────────────┘
                        ↓
┌─────────────────────────────────────────────────┐
│  1. Default Values (initializeDefaults())       │
│     - Hardcoded in constructor                  │
│     - Lowest priority                           │
└─────────────────────────────────────────────────┘
                        ↓
┌─────────────────────────────────────────────────┐
│  2. Config File (loadFromFile())                │
│     - JSON/YAML format                          │
│     - Overrides defaults                        │
└─────────────────────────────────────────────────┘
                        ↓
┌─────────────────────────────────────────────────┐
│  3. Environment Variables (loadFromEnvironment()) │
│     - NETSEC_* prefix                           │
│     - Overrides file config                     │
└─────────────────────────────────────────────────┘
                        ↓
┌─────────────────────────────────────────────────┐
│  4. Command Line Arguments (loadFromCommandLine()) │
│     - --key=value format                        │
│     - Highest priority                          │
└─────────────────────────────────────────────────┘
```

Ví dụ:

cpp Sao chép

```
ConfigManager& config = ConfigManager::getInstance();

// 1. Load defaults
config.initializeDefaults();  // network.interface = "eth0"

// 2. Load from file
config.loadFromFile("config.json");  // network.interface =
"eth1"

// 3. Load from environment
// export NETSEC_NETWORK_INTERFACE=eth2
config.loadFromEnvironment("NETSEC_");  // network.interface
= "eth2"

// 4. Load from command line
// ./app --network.interface=eth3
config.loadFromCommandLine(argc, argv);  //
network.interface = "eth3" ✅ FINAL
```

🏗️ II. CẤU TRÚC DỮ LIỆU

1. ConfigType Enum

cpp Sao chép

```
enum class ConfigType {
    STRING,    // std::string
    INTEGER,   // int
    DOUBLE,    // double
    BOOLEAN,   // bool
    ARRAY,     // std::vector<std::string>
    OBJECT     // Nested config (future use)
};
```

2. ConfigEntry Struct

cpp Sao chép

```
struct ConfigEntry {
    // === DATA ===
    std::any value;                              // Type-
erased value
    ConfigType type;                             // Type
identifier
    std::string description;                     // Human-
readable description
```

```
    // === FLAGS ===
    bool is_required;                       // Must be
set before use
    bool is_readonly;                       // Cannot
be modified after init

    // === VALIDATION ===
    std::function<bool(const std::any&)> validator;  //
Custom validator

    // Constructors
    ConfigEntry();
    ConfigEntry(const std::any& val, ConfigType t,
                const std::string& desc = "",
                bool required = false,
                bool readonly = false);
};
```

Giải thích các trường:

Field	Purpose	Example
`value`	Actual config value	`"eth0"` , `8080` , `true`
`type`	Type for validation	`ConfigType::STRING`
`description`	Documentation	`"Network interface name"`
`is_required`	Must be set	`true` for `network.interface`
`is_readonly`	Immutable after init	`true` for `system.thread_pool_size`
`validator`	Custom validation	`[](auto v) { return port > 0; }`

Ví dụ sử dụng:

cpp Sao chép

```
ConfigEntry entry(
    "eth0",                         // value
    ConfigType::STRING,             // type
```

```
    "Network interface for capture",  // description
    true,                             // required
    false                             // readonly
);

entry.validator = [](const std::any& val) {
    std::string iface = std::any_cast<std::string>(val);
    return !iface.empty() && iface.length() < 16;
};
```

3. ConfigKeys Namespace (Constants)

cpp Sao chép

```
namespace ConfigKeys {
    // System
    const std::string SYSTEM_LOG_LEVEL = "system.log_level";
    const std::string SYSTEM_LOG_FILE = "system.log_file";
    const std::string SYSTEM_MAX_LOG_SIZE =
"system.max_log_size";
    const std::string SYSTEM_THREAD_POOL_SIZE =
"system.thread_pool_size";
    const std::string SYSTEM_ENABLE_DEBUG =
"system.enable_debug";

    // Network
    const std::string NETWORK_INTERFACE =
"network.interface";
    const std::string NETWORK_CAPTURE_BUFFER_SIZE =
"network.capture_buffer_size";
    const std::string NETWORK_PACKET_TIMEOUT =
"network.packet_timeout";
    const std::string NETWORK_MAX_PACKET_SIZE =
"network.max_packet_size";
    const std::string NETWORK_PROMISCUOUS_MODE =
"network.promiscuous_mode";

    // Detection
    const std::string DETECTION_ENABLE_DPI =
"detection.enable_dpi";
    const std::string DETECTION_SIGNATURE_FILE =
"detection.signature_file";
    const std::string DETECTION_ANOMALY_THRESHOLD =
"detection.anomaly_threshold";
    const std::string DETECTION_MAX_FLOWS =
"detection.max_flows";
    const std::string DETECTION_FLOW_TIMEOUT =
"detection.flow_timeout";

    // AI/ML
```

```
    const std::string AI_MODEL_PATH = "ai.model_path";
    const std::string AI_ENABLE_GPU = "ai.enable_gpu";
    const std::string AI_BATCH_SIZE = "ai.batch_size";
    const std::string AI_CONFIDENCE_THRESHOLD =
"ai.confidence_threshold";
    const std::string AI_UPDATE_INTERVAL =
"ai.update_interval";

    // Database
    const std::string DB_HOST = "db.host";
    const std::string DB_PORT = "db.port";
    const std::string DB_NAME = "db.name";
    const std::string DB_USERNAME = "db.username";
    const std::string DB_PASSWORD = "db.password";
    const std::string DB_MAX_CONNECTIONS =
"db.max_connections";
    const std::string DB_CONNECTION_TIMEOUT =
"db.connection_timeout";

    // Alerts
    const std::string ALERT_ENABLE_EMAIL =
"alert.enable_email";
    const std::string ALERT_EMAIL_SMTP_HOST =
"alert.email.smtp_host";
    const std::string ALERT_EMAIL_SMTP_PORT =
"alert.email.smtp_port";
    const std::string ALERT_EMAIL_USERNAME =
"alert.email.username";
    const std::string ALERT_EMAIL_PASSWORD =
"alert.email.password";
    const std::string ALERT_EMAIL_RECIPIENTS =
"alert.email.recipients";
    const std::string ALERT_WEBHOOK_URL =
"alert.webhook_url";
    const std::string ALERT_SEVERITY_THRESHOLD =
"alert.severity_threshold";

    // Performance
    const std::string PERF_ENABLE_PROFILING =
"perf.enable_profiling";
    const std::string PERF_STATS_INTERVAL =
"perf.stats_interval";
    const std::string PERF_MAX_MEMORY_USAGE =
"perf.max_memory_usage";
    const std::string PERF_MAX_CPU_USAGE =
"perf.max_cpu_usage";
}
```

Ứng dụng:

cpp Sao chép

```
// Type-safe access
std::string iface =
config.getString(ConfigKeys::NETWORK_INTERFACE);
int buffer_size =
config.getInt(ConfigKeys::NETWORK_CAPTURE_BUFFER_SIZE);
bool enable_dpi =
config.getBool(ConfigKeys::DETECTION_ENABLE_DPI);
```

🔧 III. CORE FUNCTIONALITY

1. Singleton Pattern

cpp Sao chép

```
class ConfigManager : public Singleton<ConfigManager> {
    friend class Singleton<ConfigManager>;

private:
    ConfigManager();  // Private constructor
    ~ConfigManager();

    // Delete copy/move
    ConfigManager(const ConfigManager&) = delete;
    ConfigManager& operator=(const ConfigManager&) = delete;
};
```

Ứng dụng:

cpp Sao chép

```
// Global access
ConfigManager& config = ConfigManager::getInstance();

// Thread-safe initialization (C++11 magic statics)
// First call initializes, subsequent calls return same
instance
```

2. Thread Safety - Read-Write Lock

cpp Sao chép

```
private:
    mutable std::shared_mutex config_mutex_;  // Reader-
writer lock
```

```
    std::unordered_map<std::string, ConfigEntry>
config_map_;
```

Read Operations (Multiple readers allowed):

cpp Sao chép

```
std::string getString(const std::string& key,
                      const std::string& default_value)
const {
    std::shared_lock<std::shared_mutex> lock(config_mutex_);
// Shared lock

    auto it = config_map_.find(key);
    if (it == config_map_.end()) return default_value;

    return std::any_cast<std::string>(it->second.value);
}
```

Write Operations (Exclusive lock):

cpp Sao chép

```
bool setString(const std::string& key, const std::string&
value) {
    std::unique_lock<std::shared_mutex> lock(config_mutex_);
// Exclusive lock

    config_map_[key] = ConfigEntry(value,
ConfigType::STRING);
    return true;
}
```

Performance:

- ✅ **Multiple readers**: No blocking between readers
- ✅ **Single writer**: Blocks all readers/writers
- ✅ **Read-heavy workload**: Optimal for config access pattern

3. initializeDefaults() - Default Configuration

cpp Sao chép

```
void ConfigManager::initializeDefaults() {
    // System defaults
```

```
    setString(ConfigKeys::SYSTEM_LOG_LEVEL, "INFO",
              "System log level (DEBUG, INFO, WARN,
ERROR)");
    setString(ConfigKeys::SYSTEM_LOG_FILE, "netsec.log",
              "System log file path");
    setInt(ConfigKeys::SYSTEM_MAX_LOG_SIZE, 100 * 1024 *
1024,
           "Maximum log file size in bytes");  // 100MB
    setInt(ConfigKeys::SYSTEM_THREAD_POOL_SIZE,
           std::thread::hardware_concurrency(),
           "Thread pool size");
    setBool(ConfigKeys::SYSTEM_ENABLE_DEBUG, false,
            "Enable debug mode");

    // Network defaults
    setString(ConfigKeys::NETWORK_INTERFACE, "eth0",
              "Network interface for packet capture");
    setInt(ConfigKeys::NETWORK_CAPTURE_BUFFER_SIZE, 64 *
1024 * 1024,
           "Capture buffer size in bytes");  // 64MB
    setInt(ConfigKeys::NETWORK_PACKET_TIMEOUT, 1000,
           "Packet capture timeout in milliseconds");
    setBool(ConfigKeys::NETWORK_PROMISCUOUS_MODE, true,
            "Enable promiscuous mode");

    // Detection defaults
    setBool(ConfigKeys::DETECTION_ENABLE_DPI, true,
            "Enable Deep Packet Inspection");
    setDouble(ConfigKeys::DETECTION_ANOMALY_THRESHOLD, 0.8,
              "Anomaly detection threshold");
    setInt(ConfigKeys::DETECTION_MAX_FLOWS, 100000,
           "Maximum concurrent flows");

    // AI defaults
    setString(ConfigKeys::AI_MODEL_PATH, "models/",
              "AI model directory path");
    setBool(ConfigKeys::AI_ENABLE_GPU, false,
            "Enable GPU acceleration");
    setDouble(ConfigKeys::AI_CONFIDENCE_THRESHOLD, 0.7,
              "AI confidence threshold");

    // Set required keys
    setRequired(ConfigKeys::NETWORK_INTERFACE, true);
    setRequired(ConfigKeys::DB_HOST, true);

    // Set readonly keys
    setReadonly(ConfigKeys::SYSTEM_THREAD_POOL_SIZE, true);
}
```

Tại sao cần defaults?

- ✅ **Fallback values**: Hệ thống vẫn chạy được mà không cần config file
- ✅ **Documentation**: Mô tả ý nghĩa từng config key
- ✅ **Type safety**: Định nghĩa type cho mỗi key

4. loadFromFile() - JSON Configuration

cpp Sao chép

```cpp
bool ConfigManager::loadFromFile(const std::string&
config_file) {
    try {
        std::ifstream file(config_file);
        if (!file.is_open()) {
            std::cerr << "Cannot open config file: " <<
config_file << std::endl;
            return false;
        }

        std::stringstream buffer;
        buffer << file.rdbuf();
        file.close();

        return loadFromJson(buffer.str());
    }
    catch (const std::exception& e) {
        std::cerr << "Error loading config file: " <<
e.what() << std::endl;
        return false;
    }
}
```

Example config.json:

json Sao chép

```json
{
    "system": {
        "log_level": "DEBUG",
        "log_file": "/var/log/netsec.log",
        "max_log_size": 209715200,
        "thread_pool_size": 8,
        "enable_debug": true
    },
    "network": {
        "interface": "eth1",
        "capture_buffer_size": 134217728,
        "packet_timeout": 500,
        "promiscuous_mode": true
```

```
    },
    "detection": {
        "enable_dpi": true,
        "signature_file": "/etc/netsec/signatures.yaml",
        "anomaly_threshold": 0.85,
        "max_flows": 200000,
        "flow_timeout": 600
    },
    "ai": {
        "model_path": "/opt/netsec/models/",
        "enable_gpu": true,
        "batch_size": 64,
        "confidence_threshold": 0.75
    },
    "db": {
        "host": "192.168.1.100",
        "port": 5432,
        "name": "netsec_prod",
        "username": "netsec_admin",
        "password": "SecurePass123!",
        "max_connections": 20
    },
    "alert": {
        "enable_email": true,
        "email": {
            "smtp_host": "smtp.company.com",
            "smtp_port": 587,
            "username": "alerts@company.com",
            "password": "EmailPass123!",
            "recipients": [
                "security-team@company.com",
                "admin@company.com"
            ]
        },
        "webhook_url":
"https://hooks.slack.com/services/XXX",
        "severity_threshold": "HIGH"
    }
}
```

5. loadFromJson() - JSON Parsing

cpp Sao chép

```
bool ConfigManager::loadFromJson(const std::string&
json_content) {
    try {
        // Remove comments (JSON doesn't support comments
officially)
        std::string clean_json =
```

```
removeJsonComments(json_content);

        json j = json::parse(clean_json);

        // Parse recursively (handles nested objects)
        return parseJsonRecursive(j.dump(), "");
    }
    catch (const json::parse_error& e) {
        std::cerr << "JSON parse error: " << e.what() <<
std::endl;
        return false;
    }
}
```

removeJsonComments() - Comment Support

cpp Sao chép

```
std::string ConfigManager::removeJsonComments(const
std::string& json_content) {
    std::string result;
    std::istringstream iss(json_content);
    std::string line;
    bool in_multiline_comment = false;

    while (std::getline(iss, line)) {
        // Remove single-line comments (//)
        size_t comment_pos = line.find("//");
        if (comment_pos != std::string::npos &&
!in_multiline_comment) {
            line = line.substr(0, comment_pos);
        }

        // Handle multi-line comments (/* */)
        if (!in_multiline_comment) {
            size_t start_pos = line.find("/*");
            if (start_pos != std::string::npos) {
                in_multiline_comment = true;
                line = line.substr(0, start_pos);
            }
        }
        if (in_multiline_comment) {
            size_t end_pos = line.find("*/");
            if (end_pos != std::string::npos) {
                in_multiline_comment = false;
                line = line.substr(end_pos + 2);
            } else {
                line.clear();
            }
        }

        // Trim whitespace
```

```
            line.erase(0, line.find_first_not_of(" \t"));
            line.erase(line.find_last_not_of(" \t") + 1);

            if (!line.empty()) {
                result += line + "\n";
            }
        }

        return result;
    }
```

Ví dụ JSON với comments:

json Sao chép

```
{
    // System configuration
    "system": {
        "log_level": "INFO",  // DEBUG, INFO, WARN, ERROR
        /*
         * Thread pool size
         * Auto-detect CPU cores
         */
        "thread_pool_size": 8
    }
}
```

parseJsonRecursive() - Nested Objects

cpp Sao chép

```
bool ConfigManager::parseJsonRecursive(const std::string&
json_str,
                                       const std::string&
prefix) {
    try {
        json j = json::parse(json_str);

        for (auto it = j.begin(); it != j.end(); ++it) {
            std::string key = prefix.empty() ? it.key()
                                             : prefix + "."
+ it.key();

            if (it.value().is_object()) {
                // Recursive call for nested objects
                parseJsonRecursive(it.value().dump(), key);
            }
            else if (it.value().is_string()) {
                setString(key, it.value().get<std::string>
```

```
());
            }
            else if (it.value().is_number_integer()) {
                setInt(key, it.value().get<int>());
            }
            else if (it.value().is_number_float()) {
                setDouble(key, it.value().get<double>());
            }
            else if (it.value().is_boolean()) {
                setBool(key, it.value().get<bool>());
            }
            else if (it.value().is_array()) {
                std::vector<std::string> arr;
                for (const auto& item : it.value()) {
                    if (item.is_string()) {
                        arr.push_back(item.get<std::string>
());
                    } else {
                        arr.push_back(item.dump());
                    }
                }
                setStringArray(key, arr);
            }
        }

        return true;
    }
    catch (const std::exception& e) {
        std::cerr << "Error parsing JSON: " << e.what() <<
std::endl;
        return false;
    }
}
```

Flattening Example:

json Sao chép

```
{
    "network": {
        "interface": "eth0",
        "capture": {
            "buffer_size": 64000000,
            "timeout": 1000
        }
    }
}
```

Flattened keys:

ini Sao chép

```ini
network.interface = "eth0"
network.capture.buffer_size = 64000000
network.capture.timeout = 1000
```

6. loadFromEnvironment() - Environment Variables

cpp Sao chép

```cpp
void ConfigManager::loadFromEnvironment(const std::string&
prefix) {
    extern char** environ;

    if (environ == nullptr) {
        std::cerr << "Environment variables not available"
<< std::endl;
        return;
    }

    for (char** env = environ; *env != nullptr; ++env) {
        std::string env_var(*env);
        size_t eq_pos = env_var.find('=');
        if (eq_pos == std::string::npos) continue;

        std::string key = env_var.substr(0, eq_pos);
        std::string value = env_var.substr(eq_pos + 1);

        // Check prefix
        if (!prefix.empty()) {
            if (key.length() < prefix.length() ||
                key.substr(0, prefix.length()) != prefix) {
                continue;
            }
            key = key.substr(prefix.length());
        }

        // Convert: NETWORK_INTERFACE → network.interface
        std::transform(key.begin(), key.end(), key.begin(),
::tolower);
        std::replace(key.begin(), key.end(), '_', '.');

        // Auto-detect type
        if (value == "true" || value == "false") {
            setBool(key, value == "true");
        }
        else if (std::regex_match(value, std::regex(R"(^-?
\d+$)"))) {
            setInt(key, std::stoi(value));
        }
        else if (std::regex_match(value, std::regex(R"(^-?
```

```
\d+\.\d+$"))) {
            setDouble(key, std::stod(value));
        }
        else {
            setString(key, value);
        }
    }
}
```

Ví dụ:

bash Sao chép

```
# Export environment variables
export NETSEC_NETWORK_INTERFACE=eth2
export NETSEC_NETWORK_CAPTURE_BUFFER_SIZE=134217728
export NETSEC_DETECTION_ENABLE_DPI=true
export NETSEC_AI_CONFIDENCE_THRESHOLD=0.85

# Load in application
config.loadFromEnvironment("NETSEC_");

# Results:
# network.interface = "eth2"
# network.capture.buffer.size = 134217728
# detection.enable.dpi = true
# ai.confidence.threshold = 0.85
```

7. loadFromCommandLine() - CLI Arguments

cpp Sao chép

```
void ConfigManager::loadFromCommandLine(int argc, char*
argv[]) {
    for (int i = 1; i < argc; ++i) {
        std::string arg(argv[i]);

        // Handle --key=value format
        if (arg.substr(0, 2) == "--" && arg.find('=') !=
std::string::npos) {
            size_t eq_pos = arg.find('=');
            std::string key = arg.substr(2, eq_pos - 2);
            std::string value = arg.substr(eq_pos + 1);

            // Replace hyphens with dots
            std::replace(key.begin(), key.end(), '-', '.');

            // Auto-detect type
            if (value == "true" || value == "false") {
```

```
                setBool(key, value == "true");
            }
            else if (std::regex_match(value, std::regex(R"
(^-?\d+$)"))) {
                setInt(key, std::stoi(value));
            }
            else if (std::regex_match(value, std::regex(R"
(^-?\d+\.\d+$)"))) {
                setDouble(key, std::stod(value));
            }
            else {
                setString(key, value);
            }
        }
        // Handle --key value format
        else if (arg.substr(0, 2) == "--" && i + 1 < argc) {
            std::string key = arg.substr(2);
            std::string value = argv[i + 1];
            i++;  // Skip next argument

            std::replace(key.begin(), key.end(), '-', '.');

            // Auto-detect type (same as above)
            // ...
        }
    }
}
```

Ví dụ:

bash Sao chép

```
# Format 1: --key=value
./netsec \
    --network.interface=eth3 \
    --detection.max-flows=200000 \
    --ai.enable-gpu=true \
    --ai.confidence-threshold=0.9

# Format 2: --key value
./netsec \
    --network.interface eth3 \
    --detection.max-flows 200000 \
    --ai.enable-gpu true
```

8. setValue() - Generic Setter with Validation

cpp Sao chép

```
bool ConfigManager::setValue(const std::string& key, const
std::any& value,
                             ConfigType type, const
std::string& description) {
    // Validate key format
    if (!isValidKey(key)) {
        std::cerr << "Invalid key: " << key << std::endl;
        return false;
    }

    std::unique_lock<std::shared_mutex> lock(config_mutex_);

    // Check readonly
    auto it = config_map_.find(key);
    if (it != config_map_.end() && it->second.is_readonly) {
        std::cerr << "Cannot modify readonly key: " << key
<< std::endl;
        return false;
    }

    // Validate type
    if (!isValidType(value, type)) {
        std::cerr << "Invalid type for key: " << key <<
std::endl;
        return false;
    }

    std::any old_value;
    bool key_existed = (it != config_map_.end());
    if (key_existed) {
        old_value = it->second.value;
    }

    // Create entry
    ConfigEntry entry(value, type, description);
    if (key_existed) {
        entry.is_required = it->second.is_required;
        entry.is_readonly = it->second.is_readonly;  //
Preserve readonly
        entry.validator = it->second.validator;
    }

    // Custom validation
    if (entry.validator && !entry.validator(value)) {
        std::cerr << "Validation failed for key: " << key <<
std::endl;
        return false;
    }

    config_map_[key] = entry;
    lock.unlock();
```

```
    // Notify change
    notifyChange(key, key_existed ? old_value : std::any{},
value);

    return true;
}
```

9. Getter Methods with Type Safety

cpp Sao chép

```
std::string ConfigManager::getString(const std::string& key,
                                     const std::string&
default_value) const {
    std::shared_lock<std::shared_mutex> lock(config_mutex_);

    auto it = config_map_.find(key);
    if (it == config_map_.end()) return default_value;

    try {
        if (it->second.type == ConfigType::STRING) {
            return std::any_cast<std::string>(it-
>second.value);
        }
        // Type conversion fallback
        return anyToString(it->second.value, it-
>second.type);
    }
    catch (const std::bad_any_cast& e) {
        std::cerr << "Type mismatch for key: " << key <<
std::endl;
        return default_value;
    }
}

int ConfigManager::getInt(const std::string& key, int
default_value) const {
    std::shared_lock<std::shared_mutex> lock(config_mutex_);

    auto it = config_map_.find(key);
    if (it == config_map_.end()) return default_value;

    try {
        if (it->second.type == ConfigType::INTEGER) {
            return std::any_cast<int>(it->second.value);
        }
        // Try conversion from string
        if (it->second.type == ConfigType::STRING) {
            return std::stoi(std::any_cast<std::string>(it-
>second.value));
```

```
        }
        return default_value;
    }
    catch (...) {
        return default_value;
    }
}
```

🎯 IV. ADVANCED FEATURES

1. Validation System

cpp Sao chép

```cpp
// Set custom validator
config.setValidator(ConfigKeys::NETWORK_INTERFACE,
    [](const std::any& val) {
        std::string iface = std::any_cast<std::string>(val);
        // Check if interface exists
        return NetworkUtils::interfaceExists(iface);
    });

config.setValidator(ConfigKeys::DB_PORT,
    [](const std::any& val) {
        int port = std::any_cast<int>(val);
        return port > 0 && port < 65536;
    });

config.setValidator(ConfigKeys::AI_CONFIDENCE_THRESHOLD,
    [](const std::any& val) {
        double threshold = std::any_cast<double>(val);
        return threshold >= 0.0 && threshold <= 1.0;
    });
```

2. Change Notification System

cpp Sao chép

```cpp
using ConfigChangeCallback = std::function<void(
    const std::string& key,
    const std::any& old_value,
    const std::any& new_value
)>;

// Register callback
config.registerChangeCallback("network.interface",
    [](const std::string& key, const std::any& old_val,
```

```
const std::any& new_val) {
        std::string old_iface = std::any_cast<std::string>
(old_val);
        std::string new_iface = std::any_cast<std::string>
(new_val);

        LOG_INFO("Network interface changed: " << old_iface
<< " → " << new_iface);

        // Reinitialize packet capture
        packet_capture.stop();
        packet_capture.setInterface(new_iface);
        packet_capture.start();
    });
```

3. Hot Reload - File Watcher

cpp Sao chép

```
// Enable file watching
config.enableFileWatcher("config.json", 5000);  // Check
every 5 seconds

// Automatic reload on file change
void ConfigManager::fileWatcherThread() {
    while (!file_watcher_stop_) {
        std::this_thread::sleep_for(

std::chrono::milliseconds(file_watcher_interval_));

        auto current_time =
std::filesystem::last_write_time(watched_file_);

        if (current_time > last_file_modification_time_) {
            LOG_INFO("Config file changed, reloading...");
            loadFromFile(watched_file_);
            last_file_modification_time_ = current_time;
        }
    }
}
```

4. Export Configuration

cpp Sao chép

```
// Export to JSON
std::string json_config = config.exportToJson();
std::cout << json_config << std::endl;
```

```
// Save to file
config.saveToFile("config_backup.json");
```

💡 V. USAGE EXAMPLES

Complete Initialization

cpp Sao chép

```
int main(int argc, char* argv[]) {
    ConfigManager& config = ConfigManager::getInstance();

    // 1. Initialize defaults
    config.initializeDefaults();

    // 2. Load from file
    if (!config.loadFromFile("config.json")) {
        LOG_WARN("Failed to load config file, using
defaults");
    }

    // 3. Override with environment
    config.loadFromEnvironment("NETSEC_");

    // 4. Override with command line
    config.loadFromCommandLine(argc, argv);

    // 5. Validate required keys
    if (!config.validateAll()) {
        LOG_ERROR("Missing required configuration");
        return 1;
    }

    // 6. Use configuration
    std::string iface =
config.getString(ConfigKeys::NETWORK_INTERFACE);
    int buffer_size =
config.getInt(ConfigKeys::NETWORK_CAPTURE_BUFFER_SIZE);
    bool enable_gpu =
config.getBool(ConfigKeys::AI_ENABLE_GPU);

    LOG_INFO("Starting with interface: " << iface);
    LOG_INFO("Buffer size: " << buffer_size);
    LOG_INFO("GPU enabled: " << (enable_gpu ? "Yes" :
"No"));

    return 0;
}
```

📊 VI. TÓM TẮT

Feature	Purpose	Benefit
Multi-source loading	File + Env + CLI	Flexible deployment
Type safety	`std::any` + validation	Prevent runtime errors
Thread safety	`std::shared_mutex`	Concurrent access
Hot reload	File watcher	No restart needed
Change callbacks	Event notification	Dynamic reconfiguration
Validation	Custom validators	Data integrity
Readonly keys	Immutable config	Prevent accidental changes
Required keys	Mandatory config	Fail-fast on missing config

2 Nguồn

Giải thích chi tiết Logger System - Hệ thống ghi log đa tầng

ConfigManager là **trung tâm điều khiển** của toàn bộ hệ thống, đảm bảo cấu hình được **quản lý an toàn** và **linh hoạt**! 🚀

`Logger` là **hệ thống logging chuyên nghiệp** với **multiple appenders**, **async processing**, **thread-safety**, và **flexible formatting** - thiết kế theo pattern **Strategy + Observer**.

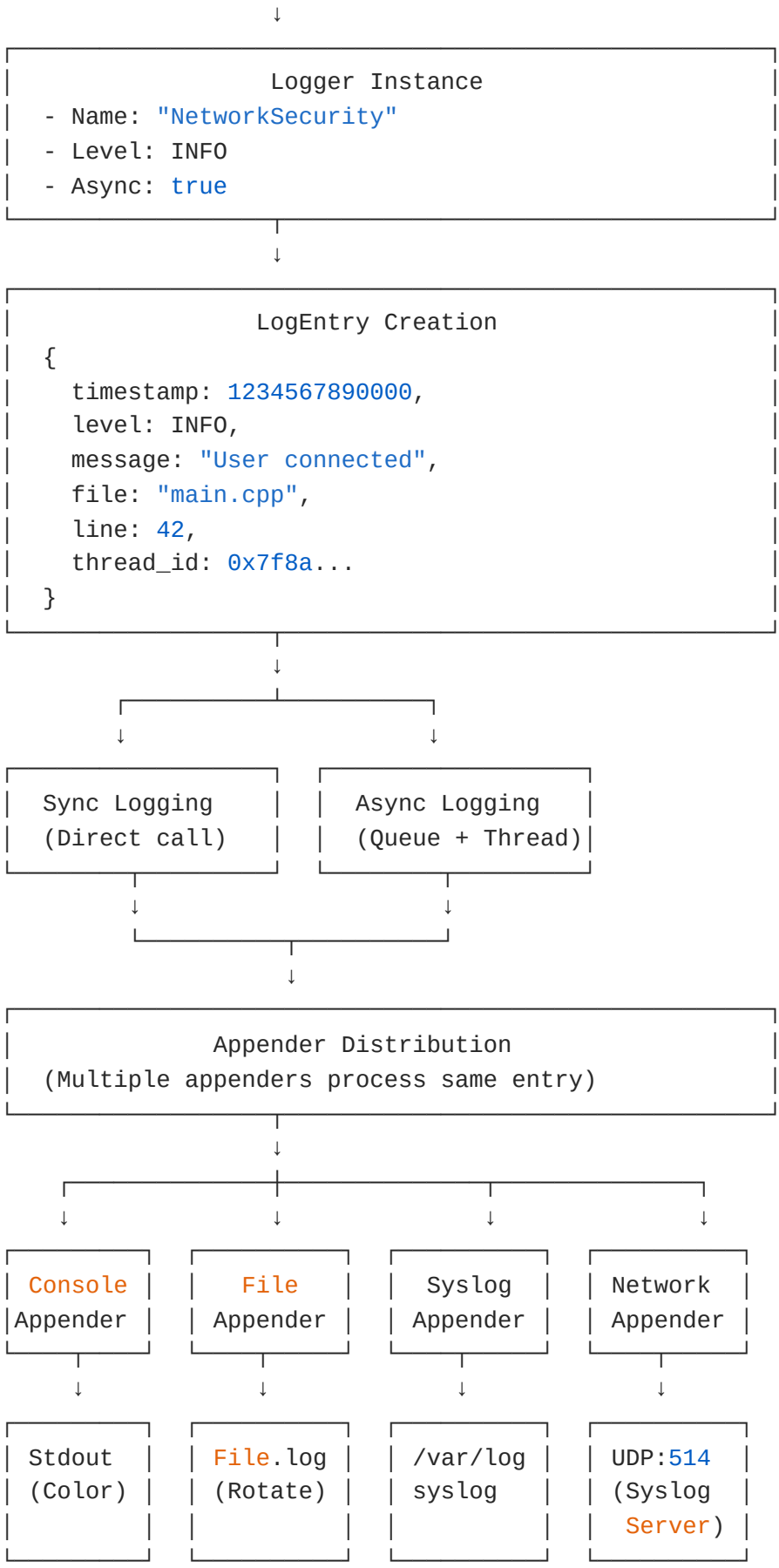
📋 I. KIẾN TRÚC TỔNG QUAN

Logger Architecture

arduino

Sao chép

```
┌─────────────────────────────────────────────────┐
│                  Application Code               │
│  LOG_INFO("User connected");                    │
│  LOG_ERROR("Database error: {}", error_msg);    │
└─────────────────────┬───────────────────────────┘
```



🏗️ II. CORE DATA STRUCTURES

1. LogLevel Enum

cpp Sao chép

```cpp
enum class LogLevel : int {
    TRACE = 0,  // Detailed debug info (function calls, variables)
    DEBUG = 1,  // Debug information (state changes, decisions)
    INFO  = 2,  // General information (startup, shutdown, events)
    WARN  = 3,  // Warning messages (deprecated API, recoverable errors)
    ERROR = 4,  // Error messages (failed operations, exceptions)
    FATAL = 5,  // Fatal errors (system crash, data corruption)
    OFF   = 6   // Disable all logging
};
```

Level Hierarchy:

sql Sao chép

```sql
TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF

If level = INFO:
  ✅ INFO, WARN, ERROR, FATAL are logged
  ❌ TRACE, DEBUG are filtered out
```

Ứng dụng:

cpp Sao chép

```cpp
// Development
logger.setLevel(LogLevel::TRACE);  // Log everything

// Production
logger.setLevel(LogLevel::WARN);   // Only warnings and errors

// Maintenance
logger.setLevel(LogLevel::OFF);    // Disable logging
```

2. LogEntry Struct

cpp Sao chép

```
struct LogEntry {
    // === TIMING ===
    uint64_t timestamp;              // Microseconds since
epoch

    // === IDENTIFICATION ===
    LogLevel level;                  // Log level
    std::string logger_name;         // Logger instance name

    // === MESSAGE ===
    std::string message;             // Actual log message
    std::string formatted_message;   // Pre-formatted
message (cache)

    // === SOURCE LOCATION ===
    std::string file;                // Source file
(__FILE__)
    std::string function;            // Function name
(__FUNCTION__)
    int line;                        // Line number
(__LINE__)

    // === CONTEXT ===
    std::thread::id thread_id;       // Thread that created
log

    // Constructors
    LogEntry();
    LogEntry(LogLevel lvl, const std::string& name, const
std::string& msg,
             const std::string& f = "", const std::string&
func = "", int l = 0);
};
```

Ví dụ LogEntry:

cpp Sao chép

```
LogEntry entry(
    LogLevel::ERROR,                 // level
    "DatabaseManager",               // logger_name
    "Connection timeout after 30s",  // message
    "db_manager.cpp",                // file
    "connect()",                     // function
    142                              // line
);

// timestamp: 1234567890123456 (auto-generated)
// thread_id: 0x7f8a1234abcd (auto-detected)
```

3. LogFormatter Interface

cpp Sao chép

```cpp
class LogFormatter {
public:
    virtual ~LogFormatter() = default;

    // Format log entry to string
    virtual std::string format(const LogEntry& entry) = 0;

    // Clone for thread-safety
    virtual std::unique_ptr<LogFormatter> clone() const = 0;
};
```

4. DefaultFormatter - Pattern-based Formatting

cpp Sao chép

```cpp
class DefaultFormatter : public LogFormatter {
public:
    DefaultFormatter(const std::string& pattern =
        "[{timestamp}] [{level}] [{logger}] {message}");

    std::string format(const LogEntry& entry) override;
    std::unique_ptr<LogFormatter> clone() const override;

private:
    std::string pattern_;
};
```

Pattern Placeholders:

cpp Sao chép

```cpp
{timestamp}  → "2024-01-15 10:30:45.123456"
{level}      → "INFO", "ERROR", etc.
{logger}     → Logger name
{message}    → Log message
{file}       → Source file name
{function}   → Function name
{line}       → Line number
{thread}     → Thread ID (hex)
```

Implementation:

cpp Sao chép

```
std::string DefaultFormatter::format(const LogEntry& entry)
{
    std::string result = pattern_;

    // Replace {timestamp}
    size_t pos = result.find("{timestamp}");
    if (pos != std::string::npos) {
        result.replace(pos, 11,
Utils::formatTimestamp(entry.timestamp));
    }

    // Replace {level}
    pos = result.find("{level}");
    if (pos != std::string::npos) {
        result.replace(pos, 7,
logLevelToString(entry.level));
    }

    // Replace {logger}
    pos = result.find("{logger}");
    if (pos != std::string::npos) {
        result.replace(pos, 8, entry.logger_name);
    }

    // Replace {message}
    pos = result.find("{message}");
    if (pos != std::string::npos) {
        result.replace(pos, 9, entry.message);
    }

    // ... (similar for other placeholders)

    return result;
}
```

Ví dụ:

cpp Sao chép

```
// Pattern
DefaultFormatter fmt("[{timestamp}] [{level}] {message}");

// Input
LogEntry entry(LogLevel::INFO, "App", "Server started");

// Output
"[2024-01-15 10:30:45.123456] [INFO] Server started"
```

Custom Patterns:

cpp Sao chép

```cpp
// Simple
"[{level}] {message}"
// Output: "[INFO] Server started"

// Detailed
"[{timestamp}] [{level}] [{logger}] {file}:{line} -
{message}"
// Output: "[2024-01-15 10:30:45] [ERROR] DB db.cpp:142 -
Connection failed"

// Production
"{timestamp}|{level}|{logger}|{message}"
// Output: "2024-01-15 10:30:45|INFO|App|Server started"
```

5. JsonFormatter - Structured Logging

cpp Sao chép

```cpp
class JsonFormatter : public LogFormatter {
public:
    JsonFormatter(bool pretty_print = false);
    std::string format(const LogEntry& entry) override;

private:
    bool pretty_print_;
};
```

Implementation:

cpp Sao chép

```cpp
std::string JsonFormatter::format(const LogEntry& entry) {
    std::ostringstream oss;

    if (pretty_print_) {
        oss << "{\n";
        oss << "  \"timestamp\": " << entry.timestamp <<
",\n";
        oss << "  \"level\": \"" <<
logLevelToString(entry.level) << "\",\n";
        oss << "  \"logger\": \"" << entry.logger_name <<
"\",\n";
        oss << "  \"message\": \"" << entry.message <<
"\",\n";
        oss << "  \"file\": \"" << entry.file << "\",\n";
        oss << "  \"function\": \"" << entry.function <<
"\",\n";
```

```
            oss << "  \"line\": " << entry.line << ",\n";
            oss << "  \"thread_id\": \"" << entry.thread_id <<
"\"\n";
            oss << "}";
        }
        else {
            oss << "{";
            oss << "\"timestamp\":" << entry.timestamp << ",";
            oss << "\"level\":\"" <<
logLevelToString(entry.level) << "\",";
            oss << "\"logger\":\"" << entry.logger_name <<
"\",";
            oss << "\"message\":\"" << entry.message << "\",";
            oss << "\"file\":\"" << entry.file << "\",";
            oss << "\"function\":\"" << entry.function << "\",";
            oss << "\"line\":" << entry.line << ",";
            oss << "\"thread_id\":\"" << entry.thread_id <<
"\"";
            oss << "}";
        }

        return oss.str();
    }
```

Output Example:

json Sao chép

```
{
  "timestamp": 1234567890123456,
  "level": "ERROR",
  "logger": "DatabaseManager",
  "message": "Connection timeout",
  "file": "db_manager.cpp",
  "function": "connect",
  "line": 142,
  "thread_id": "0x7f8a1234abcd"
}
```

Ứng dụng: ELK Stack, Splunk, CloudWatch Logs

bash Sao chép

```
# Elasticsearch query
GET /logs/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "level": "ERROR" }},
```

```
            { "range": { "timestamp": { "gte": "now-1h" }}}
        ]
      }
    }
}
```

🔧 III. LOG APPENDERS

1. LogAppender Interface

cpp Sao chép

```cpp
class LogAppender {
public:
    virtual ~LogAppender() = default;

    // Append log entry
    virtual bool append(const LogEntry& entry) = 0;

    // Flush buffered logs
    virtual void flush() = 0;

    // Close appender
    virtual void close() = 0;

    // Clone for thread-safety
    virtual std::unique_ptr<LogAppender> clone() const = 0;

    // Configuration
    void setFormatter(std::unique_ptr<LogFormatter>
formatter);
    void setLevel(LogLevel level);
    LogLevel getLevel() const;

protected:
    std::unique_ptr<LogFormatter> formatter_;
    LogLevel level_ = LogLevel::TRACE;
};
```

2. ConsoleAppender - Terminal Output

cpp Sao chép

```cpp
class ConsoleAppender : public LogAppender {
public:
    ConsoleAppender(bool use_colors = true);
```

```
    bool append(const LogEntry& entry) override;
    void flush() override;
    void close() override;

private:
    bool use_colors_;
    mutable std::mutex console_mutex_;

    std::string getColorCode(LogLevel level) const;
    std::string getResetCode() const;
};
```

Color Mapping:

cpp Sao chép

```
std::string ConsoleAppender::getColorCode(LogLevel level)
const {
    switch (level) {
        case LogLevel::TRACE:   return "\033[37m";  // White
        case LogLevel::DEBUG:   return "\033[36m";  // Cyan
        case LogLevel::INFO:    return "\033[32m";  // Green
        case LogLevel::WARN:    return "\033[33m";  //
Yellow
        case LogLevel::ERROR:   return "\033[31m";  // Red
        case LogLevel::FATAL:   return "\033[35m";  //
Magenta
        default:                return "\033[0m";   // Reset
    }
}

std::string ConsoleAppender::getResetCode() const {
    return "\033[0m";
}
```

Implementation:

cpp Sao chép

```
bool ConsoleAppender::append(const LogEntry& entry) {
    if (entry.level < level_) {
        return false;  // Filter by level
    }

    std::lock_guard<std::mutex> lock(console_mutex_);

    std::string formatted = formatter_->format(entry);

    if (use_colors_) {
        std::cout << getColorCode(entry.level)
```

```
                    << formatted
                    << getResetCode()
                    << std::endl;
    }
    else {
        std::cout << formatted << std::endl;
    }

    return true;
}
```

Ví dụ Output:

bash Sao chép

```
# With colors
[2024-01-15 10:30:45] [INFO] Server started        # Green
[2024-01-15 10:30:46] [WARN] High memory usage     # Yellow
[2024-01-15 10:30:47] [ERROR] Database error       # Red

# Without colors
[2024-01-15 10:30:45] [INFO] Server started
[2024-01-15 10:30:46] [WARN] High memory usage
[2024-01-15 10:30:47] [ERROR] Database error
```

3. FileAppender - File Output with Rotation

cpp Sao chép

```
class FileAppender : public LogAppender {
public:
    FileAppender(const std::string& filename, bool
append_mode = true);
    ~FileAppender();

    bool append(const LogEntry& entry) override;
    void flush() override;
    void close() override;

    // Rotation configuration
    void setMaxFileSize(size_t max_size);
    void setMaxFiles(int max_files);
    void enableRotation(bool enable);

private:
    std::string filename_;
    std::ofstream file_stream_;
    mutable std::mutex file_mutex_;
```

```
    size_t max_file_size_;
    int max_files_;
    bool rotation_enabled_;
    size_t current_file_size_;

    void rotateFile();
    std::string getRotatedFilename(int index) const;
};
```

File Rotation Logic:

cpp Sao chép

```
void FileAppender::rotateFile() {
    file_stream_.close();

    // Remove oldest file
    if (max_files_ > 0) {
        std::string oldest = getRotatedFilename(max_files_);
        std::remove(oldest.c_str());

        // Rotate existing files
        for (int i = max_files_ - 1; i >= 1; i--) {
            std::string old_name = getRotatedFilename(i);
            std::string new_name = getRotatedFilename(i +
1);
            std::rename(old_name.c_str(), new_name.c_str());
        }

        // Rename current file
        std::string backup = getRotatedFilename(1);
        std::rename(filename_.c_str(), backup.c_str());
    }

    // Open new file
    file_stream_.open(filename_, std::ios::out);
    current_file_size_ = 0;
}

std::string FileAppender::getRotatedFilename(int index)
const {
    return filename_ + "." + std::to_string(index);
}
```

Rotation Example:

sql Sao chép

```
Before rotation (app.log reaches 10MB):
  app.log       (10MB) ← Current file
```

```
  app.log.1     (10MB)
  app.log.2     (10MB)
  app.log.3     (10MB)

After rotation:
  app.log       (0KB)  ← New file
  app.log.1     (10MB) ← Old app.log
  app.log.2     (10MB) ← Old app.log.1
  app.log.3     (10MB) ← Old app.log.2
  app.log.4 deleted    ← Old app.log.3 removed
```

Ứng dụng:

cpp Sao chép

```cpp
// Create rotating file appender
auto file_appender = std::make_unique<FileAppender>
("app.log");
file_appender->setMaxFileSize(10 * 1024 * 1024);  // 10MB
file_appender->setMaxFiles(5);                     // Keep 5
backups
file_appender->enableRotation(true);

logger.addAppender(std::move(file_appender));
```

4. RotatingFileAppender - Simplified Rotation

cpp Sao chép

```cpp
class RotatingFileAppender : public FileAppender {
public:
    RotatingFileAppender(const std::string& filename,
                         size_t max_size = 10 * 1024 * 1024,
// 10MB
                         int max_files = 5);
};
```

Implementation:

cpp Sao chép

```cpp
RotatingFileAppender::RotatingFileAppender(const
std::string& filename,
                                           size_t max_size,
                                           int max_files)
    : FileAppender(filename, true) {
    setMaxFileSize(max_size);
```

```
    setMaxFiles(max_files);
    enableRotation(true);
}
```

Ứng dụng:

cpp Sao chép

```
// One-liner setup
auto appender = std::make_unique<RotatingFileAppender>(
    "app.log",
    50 * 1024 * 1024,  // 50MB per file
    10                  // Keep 10 files
);

logger.addAppender(std::move(appender));
```

5. SyslogAppender - System Log Integration

cpp Sao chép

```
class SyslogAppender : public LogAppender {
public:
    SyslogAppender(const std::string& ident =
"NetworkSecurity",
                   int facility = LOG_LOCAL0);
    ~SyslogAppender();

    bool append(const LogEntry& entry) override;
    void flush() override;
    void close() override;

private:
    std::string ident_;
    int facility_;
    bool opened_;

    int getSyslogPriority(LogLevel level) const;
};
```

Implementation:

cpp Sao chép

```
#ifdef __linux__
SyslogAppender::SyslogAppender(const std::string& ident, int
facility)
```

```
    : ident_(ident), facility_(facility), opened_(false) {
    if (!formatter_) {
        formatter_ = std::make_unique<DefaultFormatter>(
            "{level} {logger}: {message}"
        );
    }
    openlog(ident_.c_str(), LOG_PID | LOG_CONS, facility_);
    opened_ = true;
}

bool SyslogAppender::append(const LogEntry& entry) {
    if (entry.level < level_ || !opened_) {
        return false;
    }

    std::string formatted = formatter_->format(entry);
    int priority = getSyslogPriority(entry.level);
    syslog(priority, "%s", formatted.c_str());

    return true;
}

int SyslogAppender::getSyslogPriority(LogLevel level) const
{
    switch (level) {
        case LogLevel::TRACE:
        case LogLevel::DEBUG:   return LOG_DEBUG;
        case LogLevel::INFO:    return LOG_INFO;
        case LogLevel::WARN:    return LOG_WARNING;
        case LogLevel::ERROR:   return LOG_ERR;
        case LogLevel::FATAL:   return LOG_CRIT;
        default:                return LOG_INFO;
    }
}
#endif
```

Ứng dụng:

cpp Sao chép

```
// Send logs to syslog
auto syslog_appender = std::make_unique<SyslogAppender>(
    "NetworkSecurityAI",  // ident
    LOG_LOCAL0            // facility
);

logger.addAppender(std::move(syslog_appender));

// View logs
// journalctl -t NetworkSecurityAI
// tail -f /var/log/syslog | grep NetworkSecurityAI
```

6. NetworkAppender - Remote Logging

cpp Sao chép

```cpp
class NetworkAppender : public LogAppender {
public:
    NetworkAppender(const std::string& host, int port);
    ~NetworkAppender();

    bool append(const LogEntry& entry) override;
    void flush() override;
    void close() override;

private:
    std::string host_;
    int port_;
    int socket_fd_;
    mutable std::mutex network_mutex_;

    bool initSocket();
};
```

Ứng dụng: Centralized logging

cpp Sao chép

```cpp
// Send logs to remote syslog server
auto network_appender = std::make_unique<NetworkAppender>(
    "192.168.1.100",  // Syslog server IP
    514               // Syslog UDP port
);

// Use JSON formatter for structured logging
network_appender-
>setFormatter(std::make_unique<JsonFormatter>());

logger.addAppender(std::move(network_appender));
```

🎯 IV. LOGGER CLASS - MAIN INTERFACE

Basic Logging Methods

cpp Sao chép

```cpp
class Logger {
public:
    explicit Logger(const std::string& name);
```

```
    // Basic logging
    void trace(const std::string& message, const
std::string& file = "",
               const std::string& function = "", int line =
0);
    void debug(const std::string& message, ...);
    void info(const std::string& message, ...);
    void warn(const std::string& message, ...);
    void error(const std::string& message, ...);
    void fatal(const std::string& message, ...);

    // Formatted logging (printf-style)
    template<typename... Args>
    void info(const std::string& format, Args&&... args);

    // Configuration
    void setLevel(LogLevel level);
    void addAppender(std::unique_ptr<LogAppender> appender);
    void removeAllAppenders();

    // Async control
    void enableAsyncLogging(size_t queue_size = 10000);
    void disableAsyncLogging();

private:
    std::string name_;
    LogLevel level_;
    std::vector<std::unique_ptr<LogAppender>> appenders_;

    // Async logging
    bool async_enabled_;
    std::queue<LogEntry> log_queue_;
    std::thread worker_thread_;
    std::mutex queue_mutex_;
    std::condition_variable queue_cv_;
    std::atomic<bool> stop_flag_;

    void log(const LogEntry& entry);
    void asyncWorker();
};
```

Macros for Easy Logging

cpp Sao chép

```
// Automatic file/line/function capture
#define LOG_TRACE(logger, msg) \
    logger.trace(msg, __FILE__, __FUNCTION__, __LINE__)
```

```
#define LOG_DEBUG(logger, msg) \
    logger.debug(msg, __FILE__, __FUNCTION__, __LINE__)

#define LOG_INFO(logger, msg) \
    logger.info(msg, __FILE__, __FUNCTION__, __LINE__)

#define LOG_WARN(logger, msg) \
    logger.warn(msg, __FILE__, __FUNCTION__, __LINE__)

#define LOG_ERROR(logger, msg) \
    logger.error(msg, __FILE__, __FUNCTION__, __LINE__)

#define LOG_FATAL(logger, msg) \
    logger.fatal(msg, __FILE__, __FUNCTION__, __LINE__)
```

Ứng dụng:

cpp Sao chép

```
Logger logger("MyApp");

LOG_INFO(logger, "Server started");
// Output: [2024-01-15 10:30:45] [INFO] [MyApp] main.cpp:42
- Server started

LOG_ERROR(logger, "Connection failed");
// Output: [2024-01-15 10:30:46] [ERROR] [MyApp]
network.cpp:128 - Connection failed
```

💡 V. COMPLETE USAGE EXAMPLE

cpp Sao chép

```
int main() {
    // Create logger
    Logger logger("NetworkSecurityAI");
    logger.setLevel(LogLevel::DEBUG);

    // Add console appender (colored)
    auto console = std::make_unique<ConsoleAppender>(true);
    console->setFormatter(std::make_unique<DefaultFormatter>
(
        "[{timestamp}] [{level}] {message}"
    ));
    logger.addAppender(std::move(console));

    // Add rotating file appender
    auto file = std::make_unique<RotatingFileAppender>(
        "app.log", 10 * 1024 * 1024, 5
```

```
    );
    logger.addAppender(std::move(file));

    // Add syslog appender
    auto syslog = std::make_unique<SyslogAppender>
("NetSecAI");
    logger.addAppender(std::move(syslog));

    // Enable async logging
    logger.enableAsyncLogging(10000);

    // Use logger
    LOG_INFO(logger, "Application started");
    LOG_DEBUG(logger, "Initializing components");
    LOG_WARN(logger, "High memory usage detected");
    LOG_ERROR(logger, "Failed to connect to database");

    return 0;
}
```