DAP SDK Test Framework - Полное Руководство

Асинхронное тестирование, моки и автоматизация тестов

Команда разработки Cellframe

27 октября 2025

Содержание

1 Информация о документе

Версия: 1.0.0
Дата: 27 октября 2025
Статус: Production Ready
Язык: Русский

1.1 История изменений

Версия	Дата	Изменения	Автор
1.0.0	2025-10-27	Первая версия полного руководства	Команда Cellframe

1.2 Авторские права

Этот документ описывает DAP SDK Test Framework, часть проекта Cellframe Network.

1.3 Лицензия

2 Часть I: Введение

2.1 1. Обзор

DAP SDK Test Framework - это production-ready инфраструктура тестирования для экосистемы блокчейна Cellframe. Она предоставляет комплексные инструменты для тестирования асинхронных операций, мокирования внешних зависимостей и обеспечения надёжного выполнения тестов на разных платформах.

2.1.1 1.1 Что такое DAP SDK Test Framework?

Полное решение для тестирования, включающее:

- **Async Testing Framework** - Инструменты для тестирования асинхронных операций с таймаутами
- **Mock Framework V4** - Мокирование функций без модификации кода
- **Auto-Wrapper System** - Автоматическая конфигурация линкера
- **Self-Tests** - 21 тест, валидирующий надёжность фреймворка

2.1.2 1.2 Зачем использовать этот фреймворк?

Проблема: Тестирование асинхронного кода сложно - Операции завершаются в непредсказуемое время - Сетевые задержки варьируются - Тесты могут зависать бесконечно - Внешние зависимости усложняют тестирование

Решение: Этот фреймворк предоставляет - □ Защиту от зависаний (глобальный + для каждой операции) - □ Эффективное ожидание (polling + condition variables) - □ Изоляцию зависимостей (мокирование) - □ Реалистичную симуляцию (задержки, ошибки) - □ Потокобезопасные операции - □ Кроссплатформенность

2.1.3 1.3 Ключевые возможности

Возможность	Описание	Польза
Global Timeout	alarm + siglongjmp	Предотвращает зависание CI/CD
Condition Polling	Конфигурируемые интервалы	Эффективное ожидание
pthread Helpers	Обёртки для condition variables	Потокобезопасная координация
Mock Framework	На основе линкера (`--wrap`)	Нулевой техдолг
Задержки	Fixed, Range, Variance	Реалистичная симуляция
Callbacks	Inline + Runtime	Динамическое поведение моков
Auto-Wrapper	Bash/PowerShell скрипты	Автоматическая настройка
Self-Tests	21 комплексный тест	Проверенная надёжность

2.1.4 1.4 Быстрое сравнение

Традиционный подход:

```
// ❌ Плохо: занятое ожидание, нет таймаута, трата CPU
while (!done) {
    usleep(10000);  // 10ms сон
}
```

C DAP Test Framework:

```
// ✅ Хорошо: эффективно, защита таймаутом, автоматическое логирование
DAP_TEST_WAIT_UNTIL(done == true, 5000, "Should complete");
```

2.1.5 1.5 Целевая аудитория

- Разработчики DAP SDK
- Контрибьюторы Cellframe SDK
- Разработчики VPN Client
- Все, кто тестирует асинхронный C код в экосистеме Cellframe

2.1.6 1.6 Предварительные требования

Необходимые знания: - Программирование на C - Базовое понимание асинхронных операций - Основы CMake - Концепции pthread (для продвинутых возможностей)

Необходимое ПО: - GCC 7+ или Clang 10+ (или MinGW на Windows) - CMake 3.10+ - Библиотека pthread - Linux, macOS, или Windows (частичная поддержка)

2.2 2. Быстрый Старт

2.2.1 2.1 Первый тест (5 минут)

Шаг 1: Создайте файл теста

```c
// my_test.c
#include "dap_test.h"
#include "dap_common.h"

#define LOG_TAG "my_test"

int main() {
    dap_common_init("my_test", NULL);

    // Код теста
    int result = 2 + 2;
    dap_assert_PIF(result == 4, "Math should work");

    log_it(L_INFO, "✓ Тест пройден!");

    dap_common_deinit();
    return 0;
}
```

Шаг 2: Создайте CMakeLists.txt

```
add_executable(my_test my_test.c)
target_link_libraries(my_test dap_core)
add_test(NAME my_test COMMAND my_test)
```

Шаг 3: Соберите и запустите

```
cd build
cmake ..
make my_test
./my_test
```

2.2.2 2.2 Добавление async таймаута (2 минуты)

```c
#include "dap_test.h"
#include "dap_test_async.h"
#include "dap_common.h"

#define LOG_TAG "my_test"
#define TIMEOUT_SEC 30

int main() {
    dap_common_init("my_test", NULL);

    // Добавьте глобальный таймаут
    dap_test_global_timeout_t timeout;
    if (dap_test_set_global_timeout(&timeout, TIMEOUT_SEC, "My Test")) {
        return 1;  // Таймаут сработал
    }
```

```
    // Ваши тесты здесь

    dap_test_cancel_global_timeout();
    dap_common_deinit();
    return 0;
}
```

Обновите CMakeLists.txt:

```
target_link_libraries(my_test dap_test dap_core pthread)
```

2.2.3 2.3 Добавление моков (5 минут)

```
#include "dap_test.h"
#include "dap_mock.h"
#include "dap_common.h"

#define LOG_TAG "my_test"

// Объявите мок
DAP_MOCK_DECLARE(external_api_call);

int main() {
    dap_common_init("my_test", NULL);
    dap_mock_init();

    // Настройте мок
    DAP_MOCK_SET_RETURN(external_api_call, (void*)42);

    // Запустите код, который вызывает external_api_call
    int result = my_code_under_test();

    // Проверьте
    assert(DAP_MOCK_GET_CALL_COUNT(external_api_call) == 1);

    dap_mock_deinit();
    dap_common_deinit();
    return 0;
}
```

Обновите CMakeLists.txt:

```
include(${CMAKE_CURRENT_SOURCE_DIR}/../test-framework/mocks/DAPMockAutoWrap.cmak

target_link_libraries(my_test dap_test dap_core pthread)

# Автогенерация --wrap флагов линкера
dap_mock_autowrap(my_test)
```

2.3 3. Справочник API

2.3.1 3.1 Async Testing API

2.3.1.1 Глобальный таймаут

```
int dap_test_set_global_timeout(
    dap_test_global_timeout_t *a_timeout,
    uint32_t a_timeout_sec,
    const char *a_test_name
);
// Возвращает: 0 при настройке, 1 если таймаут сработал

void dap_test_cancel_global_timeout(void);
```

2.3.1.2 Опрос условий

```
bool dap_test_wait_condition(
    dap_test_condition_cb_t a_condition,
    void *a_user_data,
    const dap_test_async_config_t *a_config
);
// Возвращает: true если условие выполнено, false при таймауте
```

2.3.1.3 pthread хелперы

```
void dap_test_cond_wait_init(dap_test_cond_wait_ctx_t *a_ctx);
bool dap_test_cond_wait(dap_test_cond_wait_ctx_t *a_ctx, uint32_t a_timeout_ms);
void dap_test_cond_signal(dap_test_cond_wait_ctx_t *a_ctx);
void dap_test_cond_wait_deinit(dap_test_cond_wait_ctx_t *a_ctx);
```

2.3.1.4 Утилиты времени

```
uint64_t dap_test_get_time_ms(void);  // Монотонное время в мс
void dap_test_sleep_ms(uint32_t a_delay_ms);  // Кроссплатформенный sleep
```

2.3.1.5 Макросы

```
DAP_TEST_WAIT_UNTIL(condition, timeout_ms, msg)
// Быстрое ожидание условия
```

2.3.2 3.2 Mock Framework API

Заголовочный файл: `dap_mock.h`

2.3.2.1 Инициализация фреймворка

```
int dap_mock_init(void);
// Инициализация мок-фреймворка (обязательно перед использованием моков)
// Возвращает: 0 при успехе

void dap_mock_deinit(void);
// Очистка мок-фреймворка
```

2.3.2.2 Макросы объявления моков Простое объявление (авто-включено, возврат 0):

```
DAP_MOCK_DECLARE(function_name);
```

С конфигурационной структурой:

```
DAP_MOCK_DECLARE(function_name, {
    .enabled = true,
    .return_value.l = 0xDEADBEEF,
    .delay = {
        .type = DAP_MOCK_DELAY_FIXED,
        .fixed_us = 1000
    }
});
```

Со встроенным callback:

```
DAP_MOCK_DECLARE(function_name, {.return_value.i = 0}, {
    // Тело callback - пользовательская логика для каждого вызова
    if (a_arg_count >= 1) {
        int arg = (int)(intptr_t)a_args[0];
        return (void*)(intptr_t)(arg * 2);  // Удваиваем входное значение
    }
    return (void*)0;
});
```

Для пользовательской обертки (без авто-генерации):

```
DAP_MOCK_DECLARE_CUSTOM(function_name, {
    .delay = {
        .type = DAP_MOCK_DELAY_VARIANCE,
        .variance = {.center_us = 100000, .variance_us = 50000}
    }
});
```

2.3.2.3 Конфигурационные структуры dap_mock_config_t:

```
typedef struct dap_mock_config {
    bool enabled;                       // Включить/выключить мок
    dap_mock_return_value_t return_value;  // Возвращаемое значение
    dap_mock_delay_t delay;             // Задержка выполнения
} dap_mock_config_t;

// По умолчанию: enabled=true, return=0, без задержки
#define DAP_MOCK_CONFIG_DEFAULT { \
    .enabled = true, \
    .return_value = {0}, \
    .delay = {.type = DAP_MOCK_DELAY_NONE} \
}
```

dap_mock_return_value_t:

```
typedef union dap_mock_return_value {
    int i;          // Для int, bool, малых типов
    long l;         // Для указателей (приведение через intptr_t)
```

```
    uint64_t u64;  // Для uint64_t, size_t (64-бит)
    void *ptr;     // Для void*, общих указателей
    char *str;     // Для char*, строк
} dap_mock_return_value_t;
```

dap_mock_delay_t:

```
typedef enum {
    DAP_MOCK_DELAY_NONE,      // Без задержки
    DAP_MOCK_DELAY_FIXED,     // Фиксированная задержка
    DAP_MOCK_DELAY_RANGE,     // Случайная в [min, max]
    DAP_MOCK_DELAY_VARIANCE   // Центр ± разброс
} dap_mock_delay_type_t;

typedef struct dap_mock_delay {
    dap_mock_delay_type_t type;
    union {
        uint64_t fixed_us;
        struct { uint64_t min_us; uint64_t max_us; } range;
        struct { uint64_t center_us; uint64_t variance_us; } variance;
    };
} dap_mock_delay_t;
```

2.3.2.4 Макросы управления

```
DAP_MOCK_ENABLE(func_name)
// Включить мок (перехват вызовов)

DAP_MOCK_DISABLE(func_name)
// Выключить мок (вызов реальной функции)

DAP_MOCK_RESET(func_name)
// Сбросить историю вызовов

DAP_MOCK_SET_RETURN(func_name, value)
// Установить возвращаемое значение (приведение через (void*))

DAP_MOCK_GET_CALL_COUNT(func_name)
// Получить количество вызовов мока

DAP_MOCK_WAS_CALLED(func_name)
// Возвращает true если был вызван хотя бы раз

DAP_MOCK_GET_ARG(func_name, call_idx, arg_idx)
// Получить конкретный аргумент из вызова
```

2.3.2.5 Макросы конфигурации задержек

```
DAP_MOCK_SET_DELAY_FIXED(func_name, microseconds)
DAP_MOCK_SET_DELAY_MS(func_name, milliseconds)
// Установить фиксированную задержку

DAP_MOCK_SET_DELAY_RANGE(func_name, min_us, max_us)
```

```
DAP_MOCK_SET_DELAY_RANGE_MS(func_name, min_ms, max_ms)
// Установить случайную задержку в диапазоне

DAP_MOCK_SET_DELAY_VARIANCE(func_name, center_us, variance_us)
DAP_MOCK_SET_DELAY_VARIANCE_MS(func_name, center_ms, variance_ms)
// Установить задержку с разбросом (например, 100мс ± 20мс)

DAP_MOCK_CLEAR_DELAY(func_name)
// Убрать задержку
```

2.3.2.6 Конфигурация callback

```
DAP_MOCK_SET_CALLBACK(func_name, callback_func, user_data)
// Установить пользовательскую функцию callback

DAP_MOCK_CLEAR_CALLBACK(func_name)
// Убрать callback (использовать return_value)

// Сигнатура callback:
typedef void* (*dap_mock_callback_t)(
    void **a_args,
    int a_arg_count,
    void *a_user_data
);
```

2.3.3 3.3 API пользовательских линкер-оберток

Заголовочный файл: `dap_mock_linker_wrapper.h`

2.3.3.1 Макрос DAP_MOCK_WRAPPER_CUSTOM Создает пользовательскую линкер-обертку с PARAM синтаксисом:

```
DAP_MOCK_WRAPPER_CUSTOM(return_type, function_name,
    PARAM(type1, name1),
    PARAM(type2, name2),
    ...
) {
    // Реализация пользовательской обертки
}
```

Возможности: - Автоматически генерирует сигнатуру функции - Автоматически создает массив void* аргументов с правильным приведением типов - Автоматически проверяет, включен ли мок - Автоматически выполняет настроенную задержку - Автоматически записывает вызов - Вызывает реальную функцию при выключенном моке

Пример:

```
DAP_MOCK_WRAPPER_CUSTOM(int, my_function,
    PARAM(const char*, path),
    PARAM(int, flags),
    PARAM(mode_t, mode)
) {
```

```
    // Ваша пользовательская логика здесь
    if (strcmp(path, "/dev/null") == 0) {
        return -1;  // Симуляция ошибки
    }
    return 0;  // Успех
}
```

Макрос PARAM: - Формат: `PARAM(type, name)` - Автоматически извлекает тип и имя - Правильно обрабатывает приведение к void* - Использует `_Generic()` для корректного приведения указателей

2.3.3.2 Упрощенные макросы оберток

Для распространенных типов возвращаемых значений:

```
DAP_MOCK_WRAPPER_INT(func_name, (params), (args))
DAP_MOCK_WRAPPER_PTR(func_name, (params), (args))
DAP_MOCK_WRAPPER_VOID_FUNC(func_name, (params), (args))
DAP_MOCK_WRAPPER_BOOL(func_name, (params), (args))
DAP_MOCK_WRAPPER_SIZE_T(func_name, (params), (args))
```

2.3.4 3.4 Интеграция с CMake

CMake модуль: `mocks/DAPMockAutoWrap.cmake`

```
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

# Автоматическое сканирование исходников и генерация --wrap флагов
dap_mock_autowrap(target_name)

# Альтернатива: явно указать исходные файлы
dap_mock_autowrap(TARGET target_name SOURCE file1.c file2.c)
```

Как работает: 1. Сканирует исходные файлы на наличие паттернов `DAP_MOCK_DECLARE` 2. Извлекает имена функций 3. Добавляет `-Wl,--wrap=function_name` к флагам линкера 4. Работает с GCC, Clang, MinGW

2.4 4. Полные примеры

2.4.1 4.1 Тест стейт-машины (Пример из реального проекта)

Пример из `cellframe-srv-vpn-client/tests/unit/test_vpn_state_handlers.c`:

```c
#include "dap_test.h"
#include "dap_mock.h"
#include "vpn_state_machine.h"
#include "vpn_state_handlers_internal.h"

#define LOG_TAG "test_vpn_state_handlers"

// Объявление моков с простой конфигурацией
DAP_MOCK_DECLARE(dap_net_tun_deinit);
DAP_MOCK_DECLARE(dap_chain_node_client_close_mt);
DAP_MOCK_DECLARE(vpn_wallet_close);

// Мок с конфигурацией возвращаемого значения
DAP_MOCK_DECLARE(dap_chain_node_client_connect_mt, {
    .return_value.l = 0xDEADBEEF
});

static vpn_sm_t *s_test_sm = NULL;

static void setup_test(void) {
    dap_mock_init();
    s_test_sm = vpn_sm_init();
    assert(s_test_sm != NULL);
}

static void teardown_test(void) {
    if (s_test_sm) {
        vpn_sm_deinit(s_test_sm);
        s_test_sm = NULL;
    }
    dap_mock_deinit();
}

void test_state_disconnected_cleanup(void) {
    log_it(L_INFO, "ТЕСТ: state_disconnected_entry() очистка");

    setup_test();

    // Настройка состояния с ресурсами
    s_test_sm->tun_handle = (void*)0x12345678;
    s_test_sm->wallet = (void*)0xABCDEF00;
    s_test_sm->node_client = (void*)0x22222222;

    // Включение моков
    DAP_MOCK_ENABLE(dap_net_tun_deinit);
    DAP_MOCK_ENABLE(vpn_wallet_close);
    DAP_MOCK_ENABLE(dap_chain_node_client_close_mt);
```

```
    // Вызов обработчика состояния
    state_disconnected_entry(s_test_sm);

    // Проверка выполнения очистки
    assert(DAP_MOCK_GET_CALL_COUNT(dap_net_tun_deinit) == 1);
    assert(DAP_MOCK_GET_CALL_COUNT(vpn_wallet_close) == 1);
    assert(DAP_MOCK_GET_CALL_COUNT(dap_chain_node_client_close_mt) == 1);

    teardown_test();
    log_it(L_INFO, "□ УСПЕХ");
}

int main() {
    dap_common_init("test_vpn_state_handlers", NULL);

    test_state_disconnected_cleanup();

    log_it(L_INFO, "Все тесты ПРОЙДЕНЫ □");
    dap_common_deinit();
    return 0;
}
```

2.4.2 4.2 Мок с callback

```
#include "dap_mock.h"

DAP_MOCK_DECLARE(dap_hash_fast, {.return_value.i = 0}, {
    if (a_arg_count >= 2) {
        uint8_t *data = (uint8_t*)a_args[0];
        size_t size = (size_t)a_args[1];
        uint32_t hash = 0;
        for (size_t i = 0; i < size; i++) {
            hash += data[i];
        }
        return (void*)(intptr_t)hash;
    }
    return (void*)0;
});

void test_hash() {
    uint8_t data[] = {1, 2, 3};
    uint32_t hash = dap_hash_fast(data, 3);
    assert(hash == 6);  // Callback суммирует байты
}
```

2.4.3 4.3 Мок с задержками выполнения

Пример из dap-sdk/net/client/test/test_http_client_mocks.h:

```
#include "dap_mock.h"
```

```
// Мок с задержкой variance: симулирует реалистичные колебания сети
// 100мс ± 50мс = диапазон 50-150мс
#define HTTP_CLIENT_MOCK_CONFIG_WITH_DELAY ((dap_mock_config_t){ \
    .enabled = true, \
    .delay = { \
        .type = DAP_MOCK_DELAY_VARIANCE, \
        .variance = { \
            .center_us = 100000,   /* центр 100мс */ \
            .variance_us = 50000   /* разброс ±50мс */ \
        } \
    } \
})

// Объявление мока с симуляцией сетевой задержки
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_request_full,
                        HTTP_CLIENT_MOCK_CONFIG_WITH_DELAY);

// Мок без задержки для операций очистки (мгновенное выполнение)
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_close_unsafe, {
    .enabled = true,
    .delay = {.type = DAP_MOCK_DELAY_NONE}
});
```

2.4.4 4.4 Пользовательская линкер-обертка (Продвинутый уровень)

Пример из `test_http_client_mocks.c` с использованием `DAP_MOCK_WRAPPER_CUSTOM`:

```
#include "dap_mock.h"
#include "dap_mock_linker_wrapper.h"
#include "dap_client_http.h"

// Объявление мока (регистрация во фреймворке)
DAP_MOCK_DECLARE_CUSTOM(dap_client_http_request_async,
                        HTTP_CLIENT_MOCK_CONFIG_WITH_DELAY);

// Реализация пользовательской обертки
DAP_MOCK_WRAPPER_CUSTOM(void, dap_client_http_request_async,
    PARAM(dap_worker_t*, a_worker),
    PARAM(const char*, a_uplink_addr),
    PARAM(uint16_t, a_uplink_port),
    PARAM(const char*, a_method),
    PARAM(const char*, a_path),
    PARAM(dap_client_http_callback_full_t, a_response_callback),
    PARAM(dap_client_http_callback_error_t, a_error_callback),
    PARAM(void*, a_callbacks_arg)
) {
    // Пользовательская логика мока - симуляция асинхронного поведения
    if (g_mock_http_response.should_fail && a_error_callback) {
        a_error_callback(g_mock_http_response.error_code, a_callbacks_arg);
    } else if (a_response_callback) {
        a_response_callback(
            g_mock_http_response.body,
```

```
            g_mock_http_response.body_size,
            g_mock_http_response.headers,
            a_callbacks_arg,
            g_mock_http_response.status_code
        );
    }
}
```

CMakeLists.txt:

```
# Подключение auto-wrap помощника
include(${CMAKE_SOURCE_DIR}/dap-sdk/test-framework/mocks/DAPMockAutoWrap.cmake)

add_executable(test_http_client
    test_http_client_mocks.c
    test_http_client_mocks.h
    test_main.c
)

target_link_libraries(test_http_client
    dap_test     # Тест-фреймворк с моками
    dap_core     # Библиотека DAP core
    pthread      # Поддержка многопоточности
)

# Автогенерация --wrap флагов линкера сканированием всех исходников
dap_mock_autowrap(test_http_client)
```

2.4.5 4.5 Динамическое поведение мока

```
// Мок, который меняет поведение на основе счетчика вызовов
// Симулирует нестабильную сеть: ошибка 2 раза, затем успех
DAP_MOCK_DECLARE(flaky_network_send, {.return_value.i = 0}, {
    int call_count = DAP_MOCK_GET_CALL_COUNT(flaky_network_send);

    // Ошибка в первых 2 вызовах (симуляция сетевых проблем)
    if (call_count < 2) {
        log_it(L_DEBUG, "Симуляция сетевого сбоя (попытка %d)", call_count + 1);
        return (void*)(intptr_t)-1;  // Код ошибки
    }

    // Успех с 3-го и последующих вызовов
    log_it(L_DEBUG, "Сетевой вызов успешен");
    return (void*)(intptr_t)0;  // Код успеха
});

void test_retry_logic() {
    // Тест функции с повторными попытками при ошибке
    int result = send_with_retry(data, 3);  // Максимум 3 попытки

    // Должен завершиться успешно на 3-й попытке
    assert(result == 0);
```

```
    assert(DAP_MOCK_GET_CALL_COUNT(flaky_network_send) == 3);

    log_it(L_INFO, "✓ Логика повторных попыток работает корректно");
}
```

2.5 5. Глоссарий

Асинхронная операция - Операция, завершающаяся в непредсказуемое будущее время

Auto-Wrapper - Система авто-генерации флагов линкера `--wrap` из исходников

Callback - Указатель на функцию, выполняемую при событии

Condition Polling - Повторная проверка условия до выполнения или таймаута

Condition Variable - pthread примитив для синхронизации потоков

Constructor Attribute - GCC атрибут для запуска функции до main()

Designated Initializers - C99 инициализация: `{.field = value}`

Global Timeout - Ограничение времени для всего набора тестов через SIGALRM

Linker Wrapping - `--wrap=func` перенаправляет вызовы в `__wrap_func`

Mock - Фальшивая реализация функции для тестирования

Monotonic Clock - Источник времени, не зависящий от системных часов

Poll Interval - Время между проверками условия

pthread - Библиотека POSIX threads

Return Value Union - Объединение для типобезопасных возвратов моков

Self-Test - Тест, проверяющий сам фреймворк тестирования

Thread-Safe - Корректно работает при конкурентном доступе

Timeout - Максимальное время ожидания

Union - C тип, хранящий разные типы в одной памяти

2.6 6. Решение проблем

2.6.1 Проблема: Тест зависает

Симптом: Тест выполняется бесконечно
Решение: Добавьте глобальный таймаут

```
dap_test_set_global_timeout(&timeout, 30, "Tests");
```

2.6.2 Проблема: Высокая загрузка CPU

Симптом: 100% CPU во время теста
Решение: Увеличьте интервал polling или используйте pthread helpers

```
cfg.poll_interval_ms = 500;  // Менее частый polling
```

2.6.3 Проблема: Мок не вызывается

Симптом: Выполняется реальная функция
Решение: Проверьте флаги линкера

```
make VERBOSE=1 | grep -- "--wrap"
```

2.6.4 Проблема: Неправильное возвращаемое значение

Симптом: Мок возвращает неожиданное значение
Решение: Используйте правильное поле union

```
.return_value.i = 42      // int
.return_value.l = 0xDEAD  // указатель
.return_value.ptr = ptr   // void*
```

2.6.5 Проблема: Нестабильные тесты

Симптом: Иногда проходят, иногда падают
Решение: Увеличьте таймаут, добавьте допуск

```
cfg.timeout_ms = 60000;  // 60 сек для сети
assert(elapsed >= 90 && elapsed <= 150);  // ±50мс допуск
```