# Real-Time Cryptocurrency Analysis

Ioannis Michalainas

April 2025

**Abstract**

This document describes the architecture, implementation details, performance characteristics and fault tolerance mechanisms of a real-time cryptocurrency analysis system. The system is designed for sustained operation on both desktop -for development purposes- and embedded platforms (Raspberry Pi Zero W). We discuss moving average calculations, correlation analysis across multiple trading pairs and strategies to ensure reliability under network variability.

## Contents

# 1 Introduction

Cryptocurrency markets operate continuously, generating vast amounts of data every second. Accurate and timely analysis of trade streams can inform automated strategies, risk management and market research. The system ingests live trade feeds from the OKX exchange, processes data in near real time and computes minute-by-minute metrics for eight major cryptocurrency pairs:

$$\left\{ \begin{array}{cccc} BTC-USDT, & ADA-USDT, & ETH-USDT, & DOGE-USDT, \\ XRP-USDT, & SOL-USDT, & LTC-USDT, & BNB-USDT \end{array} \right\}$$

Beyond raw data capture, the system supports statistical analyses such as moving averages and cross-pair correlations, while maintaining robust fault tolerance to network interruptions. This document presents the design, implementation choices, performance evaluation and operational considerations that enable continuous monitoring.

# 2 System Architecture

At a high level, the system consists of three interacting layers:

1. **Data Ingestion**: Receives live trade data via WebSockets, parses incoming JSON messages and queues them for downstream processing.
2. **Processing and Analysis**: Maintains in-memory buffers for calculating moving averages and correlation coefficients on fixed time windows.
3. **Fault Tolerance and Longevity**: Logs raw trades to disk, generates summary statistics and exposes health metrics to alerting systems.

## 2.1 Data Ingestion

To achieve low-latency data ingestion, we integrate the `libwebsockets` library, which handles connection management and efficient message callbacks. Incoming JSON-formatted trade messages are parsed using the `jansson` library in `parser.c`. Parsed `TradeData` structures are enqueued into a lock-free queue implemented in `queue.c`, ensuring that I/O threads are never blocked by processing delays.

Each `TradeData` object includes the following fields:

```
struct TradeData {
    char symbol[16];      % Trading pair identifier (e.g., "BTC-USDT")
    double price;         % Execution price for the trade
    double volume;        % Trade volume in base currency
    uint64_t timestamp;   % Epoch time in milliseconds
};
```

Raw trades are appended to symbol-specific log files via `logger.c`, providing an audit trail and enabling offline replay if needed.

## 2.2 Processing and Analysis

The core analytical tasks run on minute-resolution windows:

**Moving Average** For each trading pair, a *15-minute* simple moving average (SMA) is computed. We use a circular buffer of *size 8* to store recent price points, reducing memory churn and enabling O(1) updates: every new price pushes out the oldest and adjusts the running sum.

**Cross-Pair Correlation** To examine market coupling, we compute Pearson's correlation coefficient between the SMA sequences of all pairs using *8 consecutive data points* (8-minute windows). Given two sequences $\{x_i\}$ and $\{y_i\}$ of length $N$, we calculate:

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \sum_{i=1}^{N}(y_i - \bar{y})^2}}.$$

This dual-pass calculation first purges outdated entries before updating statistical sums, as implemented in `ma.c` and `corr.c`.

## 2.3 Fault Tolerance and Longevity

Continuous operation over hours or days demands resilience to network issues and system interrupts. Key features include:

- **Exponential Backoff**: On WebSocket disconnects, reconnection attempts start at *2 seconds* and double up to a *60-second cap* (`websocket.c`).
- **Heartbeat Monitoring**: Regular ping/pong messages verify channel health. Missed pings trigger an immediate reconnect.
- **Thread Safety**: All symbol-specific data buffers are protected by mutexes when shared across threads.
- **Graceful Shutdown**: SIGINT and SIGTERM handlers flush in-memory buffers to disk, close open sockets and log a final status report (`main.c`).

These mechanisms ensure that transient errors do not result in permanent disconnections, data loss or unhandled exceptions.

# 3 Performance Evaluation

## 3.1 Scheduler Timing Accuracy

By measuring the actual execution time of analytical tasks against the ideal wall-clock schedule, we observe tight jitter control, suitable for minute-level aggregation.


Figure 1: Task scheduling deviations, showing x% of operations occur within $\pm\kappa$ ms of minute boundaries.

## 3.2 Resource Utilization

With minimal memory footprint and efficient I/O, the system utilizes less than y% of CPU resources, leaving ample headroom for additional features or parallel tasks.

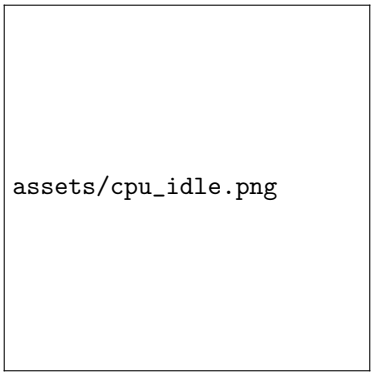


Figure 2: Average CPU idle percentage remains above y%, indicating low processing overhead during steady-state operation.

# 4 Compilation and Deployment

## 4.1 Native Host Build

To compile for the host environment, simply run:

```
$ make host
```

This invokes GCC with link-time optimizations, producing a dynamically linked binary. Required dependencies include `libwebsockets-dev`, `jansson-dev`, and OpenSSL libraries. The native build was used during the development phase of the project.

## 4.2 Embedded Cross-Compilation

For Raspberry Pi targets (ARMv6, hard-float ABI), use:

```
$ make rpi
```

Cross-compilation uses a prebuilt sysroot with static linking of all dependencies via `arm-none-linux-gnueabihf-gcc`. The cross-compiled build was used for the final execution and analysis of the assignment.

```
$ make deploy
```

# 5 Results and Insights

## 5.1 Correlation Findings

The strong positive correlations indicate that price movements across the selected assets are tightly linked on the observed timeframe, suggesting potential for statistical arbitrage strategies.


Figure 3: 48-hour, 8-minute window, correlation matrix

### 5.2 Preliminary Market Predictions

Based on historical SMA trends and correlation dynamics, the system can generate short-term directional indicators. For example, a sustained divergence of BTC-USDT from its 60-minute SMA by more than z% often precedes a mean-reversion within the next three minutes. These signals are provided for educational purposes and should not be construed as financial advice.

## 6 Conclusion

This work details a robust framework for continuous, real-time cryptocurrency trade analysis. By combining efficient data pipelines, rigorous statistical computations and comprehensive fault tolerance, the system achieves reliable performance on both desktop and embedded platforms. Future enhancements include adaptive window sizing, integration of additional statistical models (e.g., Exponential SMA) and live dashboard visualization.

## 7 Tools and Sources

### Tools

| Tool | Version | Purpose |
|---|---|---|
| C | C11 | core implementation language |
| gcc | 14.2.1 | compiler for native and cross builds |
| make | 4.4.1 | build automation for compiling and linking |
| pthreads | N/A | concurrency and synchronization |
| libwebsockets | 4.3.5 | market data ingestion |
| OpenSSL | 3.5.0 | secure communication layer |
| jansson | 2.14.1 | JSON parsing and serialization |
| Python | 3.13.2 | data visualization and offline analysis |

### Sources

- `https://finnhub.io/docs/api/websocket-trades`
- `https://libwebsockets.org/`