Implementation of a Real-Time Transaction Collection and Processing System Using Websockets on Raspberry Pi

Aimilia Palaska 10453 - aimiliapm@ece.auth.gr
Academic Supervisor: Nikolaos Pitsianis

October 2, 2024

1 Introduction

This report is created as part of the Real-Time Embedded Systems course in Electrical and Computer Engineering department of Aristotle University of Thessaloniki. Its goal is the design and implementation of a real-time transaction collection and processing system, utilizing Websockets [1] on a Raspberry Pi [2]. The system is engineered to capture and process financial transaction data in real time, leveraging the lightweight and efficient architecture of the Raspberry Pi and the bidirectional communication capabilities of Websockets. By integrating these technologies, the system ensures low-latency data transmission and continuous updates, making it an ideal solution for applications that require rapid processing and dynamic response to transaction events.

The developed code in C, along with detailed comments that explain every step, can be found in the following GitHub Repository [3].

2 Implementation

2.1 Development

The program models a producer-consumer architecture where the main service establishes a WebSocket connection to interact with the Finnhub API [4], receiving real-time trade data for a set of predefined symbols. In this implementation, the symbols "GOOG", "AAPL", and "MSFT" were utilized. A producer thread is responsible for logging the incoming trade data into a JSON file in real time. Given the possibility of receiving multiple trades simultaneously, a threshold is enforced to limit the number of entries per call.

A consumer thread, initiated at the start of the program, operates periodically on a one-minute interval. Its task is to process the trade data in real time, deriving both the candlestick data (open, close, high, low, and total volume) for the most recent minute and the 15-minute moving average of the trade prices. The results are saved to separate JSON files, generating a total of three output files per symbol.

To ensure safe program termination and effective memory management, a signal handler function has been implemented, allowing for a graceful shutdown of all threads and processes, as well as a mutex to avoid deadlocks and overwriting.

2.2 Cross-Compilation

In terms of technicalities, the execution of the program required its cross-compilation in order to simulate an actual embedded system. For this, the libraries Jansson [5], zlib [6], OpenSSL [7] and libwebsockets [8] were trivial, as they provided the necessary functionality of JSON file and WebSocket handling. In order to execute the C code in a cross-compilation environment for the Raspberry Pi Zero 2W [2] the libraries needed to be configured for the Aarch64 Linux GNU Compiler [9], which was done according to a helpful tutorial by F. Papadakis [10].

3 Experimental Setup

The goal of the course is to collect data for 48 hours continuously. The first attempts resulted in stack smashing terminations, where the program was aborted due to memory handling issues. To mitigate these issues, the code incorporates several hard-coded parameters that are essential for ensuring the efficient and memory-conscious execution of the program.

- *NUM_SYMBOLS*: This parameter points to the number of symbols to which the WebSocket subscribes. A value of three was selected as a representative example to effectively demonstrate the benefits of thread parallelization.

- *NUM_THREADS*: This parameter indicates the number of trade data entries that the program can process per WebSocket callback. It is designed to prevent the concurrent creation of an excessive number of threads, while also avoiding information loss, as the WebSocket typically returns between one to ten transactions. The choice of three as the value for this parameter was decided

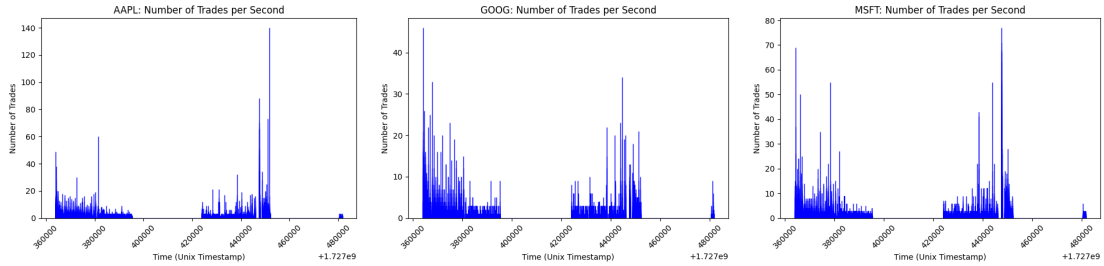

Figure 1: Number of trades received for each symbol over time.

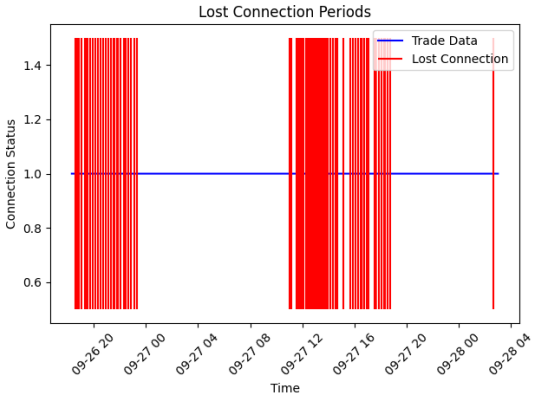

Figure 2: Representation of the WebSocket connection status throughout the 48-hour experimental period.

after initial testing, aiming to enable the Raspberry Pi to operate effectively over extended periods.

- *BUFFER_SIZE*: The final parameter is of particular significance, as it directly relates to memory management and aims to mitigate the risk of stack smashing incidents. Specifically, this parameter sets an upper limit on the number of transactions processed by the consumer and restricts the string sizes of input and output messages. Monitoring the WebSocket callbacks revealed that, within a 15-minute interval of the moving average, nearly all symbols did not exceed 1000 transactions, thus ensuring minimal information loss.

4 Results

Following the completion of the data collection phase, it was essential to extract and present the data through clear and analytical graphical representations. This task was carried out in a high-level environment using Python libraries, which efficiently support such functionalities. It is important to emphasize that this process does not constitute post-processing, as no additional data points were generated or inferred; rather, the visualizations strictly represent the data collected during the real-time experiment.

4.1 Trades over Time

For the first impression of the collected data, it is important to visualize the amount of trades per data received for each symbol, as shown in Figure 1. In this way, we get an indication of the periods of time with no connection as well as the frequency of each symbol.

4.2 Connection Disruption

The nature of the libwebsockets library combined with the technicalities of the FinnHub Server caused, in many cases, the connection to be broken. To overcome this and continue receiving trade information, a bash script was implemented in order to run the executable again in case it fails, until the 48 hour mark was reached. This obviously created some interruptions in the data collection pipeline. The periods of connection disruption, along with their duration, are shown in Figure 2. We observe that for more than 50% of the 48 hour execution period, the WebSocket was successfully established and receiving data.

4.3 Candlestick

A candlestick is a type of financial chart used to represent the price movement of an asset over a specific period of time. Each candlestick displays four key pieces of information: the opening price, the closing price, the highest price, and the lowest price during the time interval. The body of the candlestick shows the difference between the opening and closing prices, while the wicks (or shadows) represent the highs and lows. If the closing price is higher than the opening price, the candlestick is typically colored green or white (indicating a price increase), and if the closing price is lower, it's colored red or black (indicating a price decrease). Candlesticks are commonly used in technical analysis to identify price patterns and market trends.

For this implementation, the candlestick was calculated in real time with a period of one

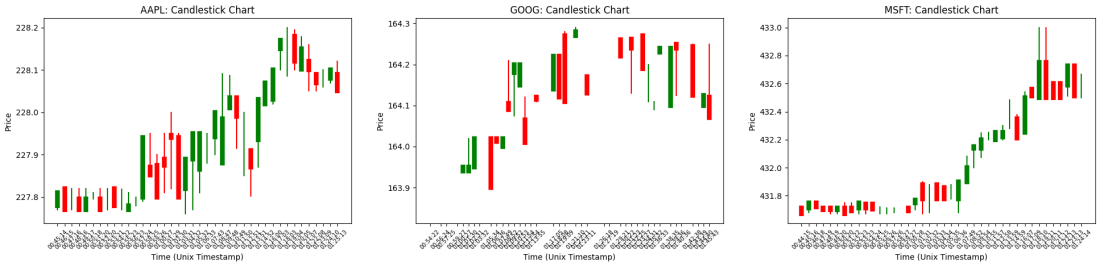

Figure 3: Candlestick of the three symbols for selective indices.

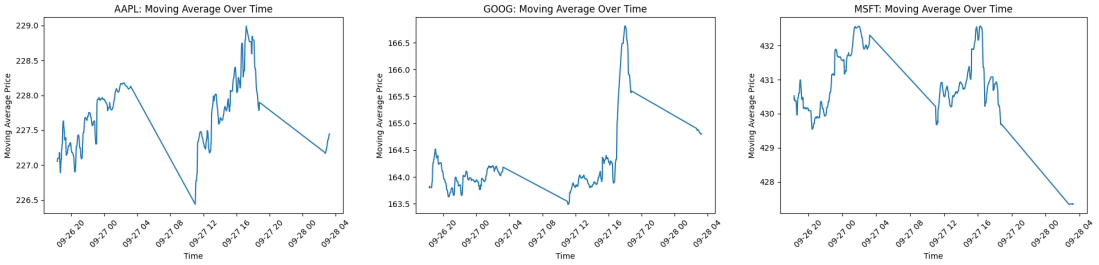

Figure 4: Delay distribution in milliseconds for the three symbols.

minute. Since it was not possible to show every single data point in the graphs, a range of indices was chosen. In Figure 3 we can see the candlestick for a relatively continuous flow of trades, noting the fluctuations of the price.

4.4 Moving Average

The moving average of the price of a trade is a statistical measure that smooths out price fluctuations by calculating the average price over a specific number of past time intervals. It helps identify trends by filtering out short-term volatility, providing a clearer view of the market's direction. Traders use it to spot trend reversals, assess momentum, and identify potential entry or exit points. A rising moving average suggests an upward trend, while a declining one indicates a downward trend, making it a valuable tool in technical analysis for trading decisions.

For this specific implementation, the moving average of the three symbols is shown in Figure 4, with the time periods of lost connection being visible by the abrupt fluctuations.

4.5 Delay

The program, designed to log the processing time in milliseconds for each symbol alongside statistical metrics, used the variable "d" to represent the delay in writing data at the end of each one-minute period. While this delay was theoretically expected to be negligible, the use of multiple threads and a mutex to prevent data overwriting increased the delay time. To analyze this, the distribution of delay periods was plotted in Figure 5. The distributions for all three symbols exhibit a tendency towards a Gaussian shape, which aligns with expectations given the large sample size. Notably, the maximum delays did not exceed one second, indicating that synchronization errors were limited to approximately 2% of the sampling period.

4.6 CPU Usage and Idleness

For the purpose of this report we are to assume that the CPU is used only during the processing of the trade data. In other words, anything related to the WebSocket connection, message receiving and write back are considered to affect the CPU usage minimally. Consequently, the overall CPU usage is calculated as the sum of the delay times - saved in the moving average JSON file and denoted with the symbol "d" - divided by the execution time, which is set to 24 hours to account for the lost connection periods. Its supplement is, of course, the CPU idle time percentage.

$$\text{CPU}_{\text{idle}} = 100\% - \frac{\sum_i delay_i}{t_{exec}}$$

In Table 1 we observe the percentage CPU Idle Time for the three symbols. The large values of Idle Time are expected since neither the procedure of maximum/minimum extraction nor the summation of the volumes is computationally heavy.

5 Conclusion

This implementation provided a valuable opportunity for practical exploration of real-time data

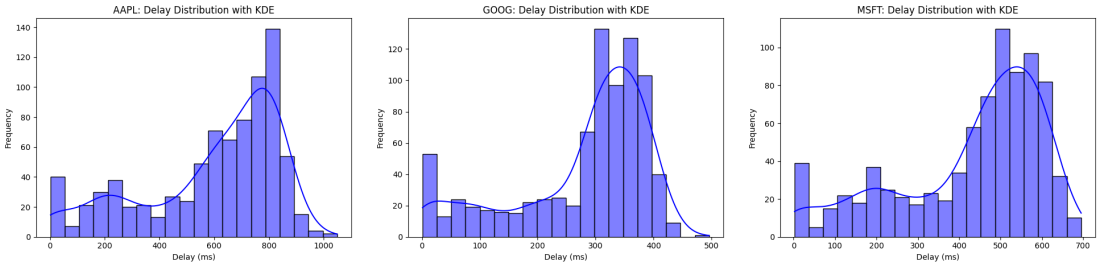

Figure 5: Delay distribution in milliseconds for the three symbols.

Symbol	CPU_{idle}
AAPL	99.43 %
GOOG	99.73 %
MSFT	99.59 %

Table 1: Percentage of Idle Time for the CPU calculated through the delays of each symbol.

processing within embedded systems. By leveraging key programming libraries and techniques, the project highlighted the significance of parallelization in optimizing performance on constrained hardware, such as the Raspberry Pi Zero 2W. The results yielded critical insights into connection instability and latency, underscoring the challenges of maintaining reliable data streams in real-time processing scenarios. Additionally, the experiment demonstrated the importance of efficient resource management and thread synchronization to mitigate performance bottlenecks. These findings emphasize the need for further refinement in embedded system designs to ensure robustness and scalability in real-world applications.

Acknowledgments

The open-source language model ChatGPT [11] was utilized in parts of these experiments. It generated portions of the code which were then tested and analyzed, as well as enhanced the overall readability and clarity of this report.

References

[1] I. Fette and A. Melnikov, "The Websocket Protocol." Internet Engineering Task Force (IETF), 2011. RFC 6455.

[2] R. P. Foundation, "Raspberry Pi Zero 2W," 2021.

[3] E. Palaska, "WebsocketRTES." https://github.com/emily-palaska/WebsocketRTES, 2024.

[4] Finnhub.io, "Finnhub - Free Realtime APIs for Stock, Forex and Cryptocurrency.." https://finnhub.io.

[5] P. Lehtinen, "Jansson: C Library for JSON," 2023. Version 2.14.

[6] J. loup Gailly and M. Adler, "zlib: General Purpose Compression Library," 2023. Version 1.2.13.

[7] O. S. Foundation, "Openssl: Full-Featured Toolkit for the TLS and SSL Protocols," 2023. Version 3.1.0.

[8] A. Green and Contributors, "libwebsockets: Lightweight C library for WebSockets," 2023. Version 4.3.

[9] G. Project, "Aarch64 Linux GNU Compiler Collection (GCC)," 2023. Version 12.2.0.

[10] F. Papadakis, "Tutorials-in-MD." https://github.com/Kou-ding/Tutorials-in-MD/blob/main/cross-compile.md, 2024. Accessed: 2024-09-05.

[11] OpenAI, "Chatgpt." https://chat.openai.com/, 2024.