

Design and Implementation of Lightweight Embedded HTTP Proxy Cache System

Nishita Saran¹, Sakshi Lagad², Priti Patil³, Dr. J.A. Desai⁴

¹Student, Department of Electronics and Telecommunication, Sinhgad College of Engineering

²Student, Department of Electronics and Telecommunication, Sinhgad College of Engineering

³Student, Department of Electronics and Telecommunication, Sinhgad College of Engineering

⁴Guide, Department of Electronics and Telecommunication, Sinhgad College of Engineering

Abstract - Railway stations rely on multiple digital display boards to provide real-time information such as train arrivals, departures, platform numbers, and status updates. In conventional architectures, each display unit independently communicates with a centralized railway server, leading to repetitive data requests, increased network congestion, higher latency, and excessive server load. These issues become more critical as the number of display units increases, affecting overall system efficiency and responsiveness.

To address these challenges, a lightweight embedded HTTP proxy cache system is implemented using the Raspberry Pi 3 platform. The proposed system acts as an intermediary between the central server and multiple display clients by intercepting HTTP requests and storing frequently accessed data in a local cache. When the same data is requested again, it is served directly from the cache instead of contacting the central server, thereby reducing redundant communication.

The system is developed using TCP/IP socket programming and supports concurrent client handling, ensuring reliable and scalable performance in real-time environments. The caching mechanism significantly reduces bandwidth consumption, improves response time, and minimizes server workload. Additionally, the architecture supports continuous data updates and efficient cache management, making it suitable for dynamic information systems.

The developed system demonstrates enhanced scalability and stable performance when multiple display boards operate simultaneously. Overall, the proposed solution provides a cost-effective, energy-efficient, and practical approach for centralized railway information systems and can be extended to other real-time data distribution applications.

Key Words: Embedded System; HTTP Proxy Cache; Raspberry Pi 3; Railway Information System; Client-Server Architecture

1.INTRODUCTION

Modern railway stations rely heavily on digital display boards to provide real-time train information such as arrival schedules, departure timings, platform numbers, and status updates. These display systems play a critical role in ensuring smooth passenger flow and effective communication. However, in conventional architectures, each display unit independently communicates with a centralized railway server to fetch updated information. As the number of display boards increases, this results in repetitive and redundant data requests, leading to excessive network traffic, increased latency, and higher computational load on the central server. Such inefficiencies become more prominent in large-scale railway stations where multiple display units operate simultaneously. Continuous communication with the central server not only consumes significant bandwidth but also affects system responsiveness, especially during peak hours. Therefore, there is a need for an efficient mechanism that can reduce redundant data transmission while maintaining real-time performance. To address these challenges, this paper proposes a lightweight embedded HTTP proxy cache system implemented using the Raspberry Pi 3 Model B. The system acts as an intermediate caching layer between the central railway server and multiple client display units. Instead of directly communicating with the main server, display units send requests to the proxy server, which checks for the availability of data in its local cache. If the requested data is already stored, it is served immediately; otherwise, the request is forwarded to the central server, and the response is cached for future use.

The proposed system utilizes HTTP-based communication and TCP/IP socket programming to ensure compatibility with existing network infrastructure. It also supports concurrent client handling, allowing multiple display units to access data simultaneously.

without performance degradation. By reducing redundant server requests, the system significantly lowers network congestion, improves response time, and enhances overall system efficiency. Furthermore, the solution is cost-effective, scalable, and easy to deploy, making it suitable not only for railway information systems but also for other centralized real-time display applications such as airports, bus terminals, and industrial monitoring systems.

2. METHODOLOGY

2.1. Introduction

The proposed system implements a lightweight embedded HTTP proxy cache mechanism for efficient data handling in railway information systems. It follows a client-proxy-server architecture, where multiple display units act as clients that continuously request real-time train information from a centralized server. Instead of direct communication with the central server, all requests are routed through a proxy server implemented on the Raspberry Pi 3. The proxy is developed using TCP/IP socket programming and HTTP protocol handling in a Linux environment. Upon receiving a request, the proxy parses the HTTP message and checks its local cache to determine whether the requested data is already available.

If the requested data is present in the cache (cache hit), it is served directly to the client, reducing response time and avoiding redundant communication with the central server. In the case of a cache miss, the proxy forwards the request to the central server, retrieves the response, stores it in the cache, and then sends it back to the client. The system supports periodic cache updates to maintain data freshness and uses multi-threading to handle multiple client requests simultaneously. This approach ensures efficient data distribution, reduced network congestion, improved scalability, and reliable real-time performance.

2.2. Components and Specifications

Hardware Components

1. Raspberry Pi 3 Model B

- Quad-core ARM Cortex-A53 processor @ 1.2 GHz
- 1 GB RAM
- Built-in Wi-Fi (802.11n) and Ethernet

2. MicroSD Card

- Storage: 16 GB / 32 GB
- Class 10 (used for OS and application storage)

3. Power Supply

- Input: 5V DC
- Current rating: 2.5 A

4. Router / Network

- LAN/Wi-Fi based network
- Frequency: 2.4 GHz
- Supports TCP/IP communication

5. Display Units (Client Systems)

- PC/monitor or embedded display

6. Central Server (Primary Server)

- HTTP server system (PC/Raspberry Pi)
- Operates on port (e.g., 8080)

7. Ethernet/Wi-Fi Interface

- Ethernet: 10/100 Mbps
- Wi-Fi: IEEE 802.11 b/g/n

Software Specifications

1. Operating System

- Linux-based OS (Raspberry Pi OS / Raspbian)

2. Programming Language

- C language for proxy and socket implementation

3. Networking Protocols

- TCP/IP for communication
- HTTP for request-response handling

4. Socket Programming

- Used for client-server communication
- Supports multiple connections

5. HTTP Communication

- GET and POST methods supported

6. Simulation Environment

- QEMU for virtual testing of Raspberry Pi

2.3. Algorithm

1. Initialize the Raspberry Pi, cache memory, and LED display.
2. Wait for HTTP requests from the display client.
3. Check the cache for the requested data.
4. If found (cache hit), serve it to the display client and show the information on LED.
5. If not found (cache miss), forward the request to the master server.
6. Receive the response and store it locally in the cache.

3. BLOCK DIAGRAM

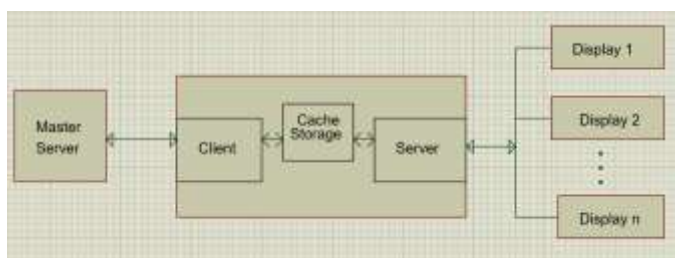


Fig3.1 Block Diagram of Lightweight Embedded HTTP Proxy Cache System

The proposed lightweight HTTP proxy cache system operates by continuously handling client requests and delivering real-time train information efficiently. When the system is initialized, the Raspberry Pi 3 configures the server environment and establishes communication with both the client display units and the central railway server. The display units periodically send HTTP requests to obtain updated train data. These requests are first received by the proxy server, which analyses the request and checks the local cache for the required information.

If the requested data is available in the cache (cache hit), the proxy server immediately returns the stored response to the client display, ensuring faster data delivery and reduced latency. In case of a cache miss, the proxy forwards the request to the central server, receives the response, stores it in the cache, and then sends it back to the client. The system also supports concurrent processing through multi-threading, allowing continuous data updates and simultaneous handling of multiple client requests. The processed data is finally displayed on the output display units, ensuring real-time visualization and efficient system performance.

4. RESULT AND DISCUSSION

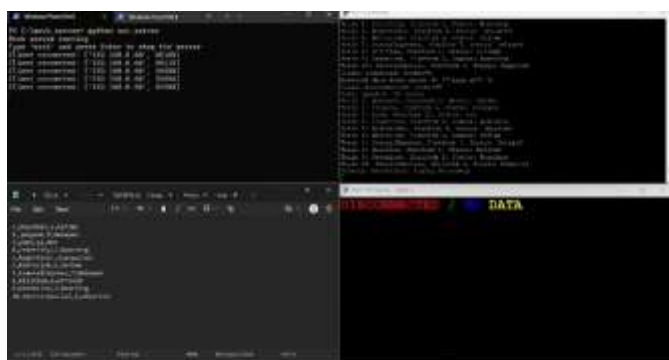


Fig 4.1 Multi-Client Communication, Cache Update and Display Output in Embedded Proxy System

This figure demonstrates the working of the embedded HTTP proxy cache system with multiple components operating simultaneously. The left terminal shows a

Python-based mock server running on the host system, which accepts multiple client connections and simulates real-time data transmission. The right terminal (PuTTY) represents the Raspberry Pi server, where incoming client requests are processed, data is received, and the cache is updated dynamically with train information such as train number, platform, and status. The bottom-left section shows the dataset file containing structured input data used for testing, while the bottom-right window represents the client-side display (RGB LED interface). The message “DISCONNECTED / NO DATA” indicates a temporary loss of connection or unavailable data from the server, demonstrating how the system handles communication failures gracefully. This result validates that the system supports concurrent client handling, real-time data updates, and cache management. It also highlights the robustness of the system in handling network issues by providing appropriate feedback on the display interface.



Fig 4.2 Result Analysis of Proxy Cache System

The implemented system demonstrates a significant improvement in overall performance by reducing response time for repeated data requests. Frequently accessed data is stored in the local cache and served directly to clients, eliminating the need to repeatedly fetch data from the central server. This results in faster data retrieval, improved user experience, and efficient utilization of system resources. By minimizing redundant data transfers, the system effectively reduces network traffic and bandwidth consumption. This decreases the load on the central server and ensures better scalability, especially in environments where multiple clients access the same data simultaneously. The use of an efficient cache replacement policy further enhances the system’s ability to manage limited storage effectively. The system was also tested under multiple client request scenarios, where it successfully handled concurrent connections without any noticeable delay or network congestion. This confirms the robustness of the system and its capability

to support real-time applications. The integration of multithreading enables parallel processing of requests, ensuring continuous and smooth operation. Additionally, the use of an embedded platform along with an LED display (simulated using an emulator) highlights the practical applicability of the system in real-world scenarios such as railway information display systems, smart stations, and IoT-based edge devices. The system consistently provided synchronized and accurate data across all client displays. Overall, the results validate that the proposed lightweight embedded HTTP proxy cache system is efficient, reliable, and scalable. It successfully achieves reduced latency, optimized bandwidth usage, and stable multi-client communication, making it a suitable solution for modern embedded and edge computing applications.

5. CONCLUSION

This research work successfully establishes that a lightweight and efficient HTTP proxy cache system can be implemented on a low-power embedded platform without compromising performance. The proposed system effectively minimizes redundant data transfers by storing and serving frequently accessed content locally, thereby reducing network congestion and significantly improving response time. The experimental results demonstrate that the system is capable of handling multiple client requests with stable and reliable performance, making it suitable for real-time and resource-constrained environments. Its cost-effectiveness and low power consumption further enhance its practicality for deployment in small-scale networks and embedded applications.

Moreover, this study highlights the increasing capability of embedded systems to undertake networking functionalities that were traditionally limited to high-performance servers. The integration of caching mechanisms with embedded platforms opens new possibilities in IoT and edge computing, where efficient data management and faster access are critical. In conclusion, the developed HTTP proxy cache system proves to be a scalable, efficient, and viable solution, contributing to the advancement of embedded networking technologies and offering a strong foundation for future enhancements and research in this domain.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our internal guide, the Project Coordinator, the Head of the

Department of Electronics & Telecommunication Engineering, and the Principal of the institution for their constant guidance, valuable suggestions, and continuous support throughout the course of this research work. Their encouragement and direction played a significant role in shaping the development and successful completion of this paper. We are deeply appreciative of the academic environment and opportunities provided, which enabled us to carry out this work effectively. Their motivation and assistance have been instrumental in enhancing our understanding and helping us overcome challenges during the research process.

REFERENCES

- [1] Kroeger, T., Long, D. D. E., & Mogul, J. C. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. USENIX Symposium, 1997.
- [2] Dias, R., Fiorese, A., Guardalben, L., Sargento, S. A distributed caching architecture for Over-the-Top content distribution. WONS, 2018.
- [3] Fielding, R. T., et al. Hypertext Transfer Protocol–HTTP/1.1. IETF RFC 2616, 1999.
- [4] Singh et al., Secure MQTT for the Internet of Things (IoT).
- [5] Li Chen, Design and Implementation of Lightweight Web Server for Embedded System.
- [6] Lerche, C., Laum, N., Niedermeier, C., Golasowski, F., Timmermann, D. Connecting the Web with the Web of Things: A CoAP-HTTP Proxy. IMIS, 2012.
- [7] ElAarag, H. Web Proxy Cache Replacement Strategies: Simulation, Implementation, and Performance Evaluation. Springer, 2013.
- [8] Ali, W., Shamsuddin, S. M., Ismail, A. S. Intelligent Web Proxy Caching Approaches Based on Machine Learning Techniques. Decision Support Systems, 2012.
- [9] Feldmann, A., Caceres, R., Douglass, F., Glass, G. Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments. IEEE INFOCOM, 1999.
- [10] ElAarag, H. Web Object-Based Storage Management in Proxy Caches. Future Generation Computer Systems, 2006.
- [11] Moise, I. A., Băicoianu, A. Optimizing Intensive Database Tasks Through Caching Proxy Mechanisms. arXiv, 2024.
- [12] Bayat, M., Jamali, M. A. J., Abbasi, M., Akbarpour, S. Enhancing Secure IoT Data Sharing through Dynamic Q-Learning and Blockchain at the Edge. Scientific Reports, 2025.