UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

Project Proposal

Design and Implementation of
CacheCaught: A High-Performance HTTP Caching Proxy Server

Prepared By:
Mehedi Hasan and Biplob Pal
Roll: 22 and 50
Batch 29, CSEDU

Submitted To:
Palash Roy
Lecturer
Department of CSE, DU

Course Code: CSE 3111

Course Title: Computer Networking Lab

Semester: 3rd Year, 1st Semester

Abstract

This project presents **CacheCaught**, a high-performance HTTP caching proxy server designed to reduce latency and origin server load by intelligently caching GET responses using Redis with Time-to-Live (TTL). The proxy operates on port 8080 using Python threading for concurrency, while a Flask dashboard on port 5000 provides real-time cache monitoring. A dedicated section analyzes TCP flow and congestion control mechanisms implemented to ensure reliable delivery and prevent network collapse under heavy load. The system combines educational clarity with production-grade performance.

Contents

1 Introduction

In modern web systems, repeated requests for static content cause significant latency and server overload. HTTP caching proxies solve this by storing responses locally. **CacheCaught** is a full-featured, educational proxy server that demonstrates real-world distributed systems concepts using Python, Redis, and Flask.

2 Problem Statement and Motivation

- Repeated fetching of unchanged content increases user-perceived latency
- Origin servers become overloaded during traffic spikes
- Lack of simple, well-documented caching proxy implementations for learning

Motivations:

- Deliver sub-10ms responses via local caching
- Offload origin servers
- Master concurrent networking, Redis, and HTTP protocol handling

3 Objectives

- Implement a concurrent HTTP proxy server using Python threading
- Integrate Redis with configurable TTL-based caching
- Ensure reliable delivery using TCP flow and congestion control
- Build a real-time monitoring dashboard with Flask
- Demonstrate measurable performance improvement

4 Project Features

- Non-transparent proxy on port 8080
- Caches only successful HTTP GET responses
- Configurable TTL via `config.json` (default: 300s)
- Thread-per-client concurrency model
- Redis backend with native key expiration
- Live Flask dashboard at `http://127.0.0.1:5000`

- Full response preview (headers + body)

5 System Architecture and Workflow

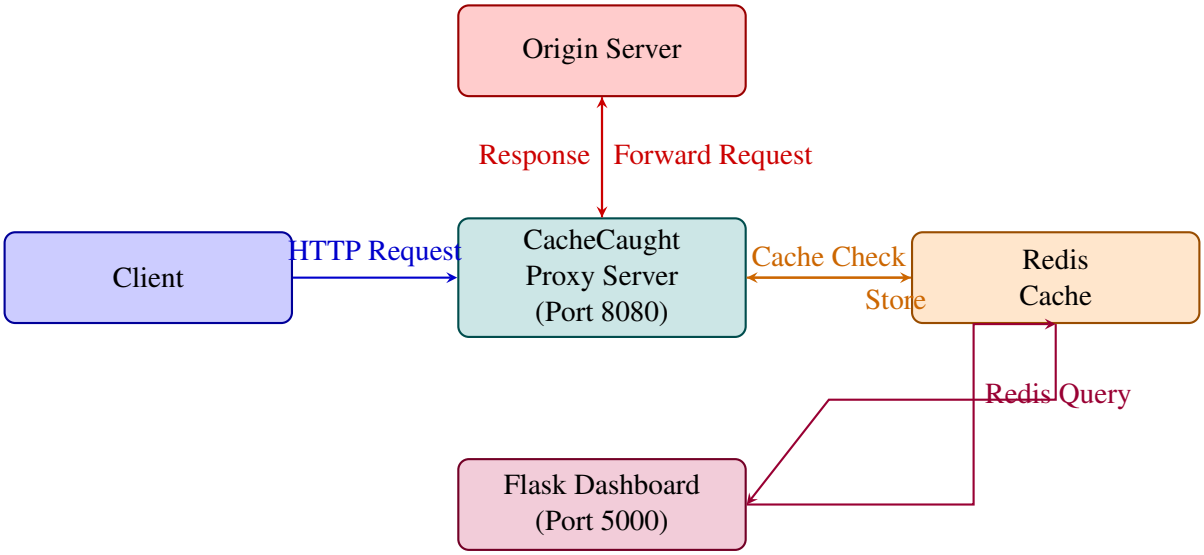

Figure 1: System Architecture of CacheCaught

6 TCP Flow Control and Congestion Control in CacheCaught

Since CacheCaught acts as both a TCP client (to origin servers) and TCP server (to browsers), proper flow and congestion control are critical.

6.1 Flow Control (Preventing Receiver Overrun)

- Implemented using TCP sliding window protocol
- Python `socket.recv()` respects advertised window from peers
- Proxy never reads more data than the receiver (client or origin) can buffer
- Prevents packet drops due to buffer overflow

6.2 Congestion Control (Preventing Network Collapse)

CacheCaught inherits TCP's robust congestion control:

6.3 Practical Impact

- During cache misses, large file transfers use full TCP congestion control

Table 1: TCP Congestion Control Mechanisms Used

Phase	Algorithm	Behavior in CacheCaught
Slow Start	Exponential growth	cwnd doubles each RTT during initial transfers
Congestion Avoidance	Linear growth	cwnd increases by 1 per RTT after ssthresh
Fast Retransmit	3 duplicate ACKs	Resends lost segment without timeout
Fast Recovery	Halves cwnd	Avoids dropping to 1 (used in modern OS)
Timeout	RTO expiration	Retransmit and reset to Slow Start

- Prevents proxy from flooding slow origin servers
- Ensures fairness with other Internet traffic
- Cache hits bypass network entirely → zero congestion

7 Tools and Technologies

Table 2: Tools and Technologies Used

Technology	Type	Purpose
Python 3.x	Language	Core proxy, threading, socket logic
Redis	In-memory Database	Ultra-fast caching with TTL
threading	Standard Library	Concurrent client handling
socket	Standard Library	Raw TCP communication
Flask	Web Framework	Real-time monitoring dashboard
JSON	Configuration	`config.json` for TTL and ports
TikZ	LaTeX Graphics	Architecture diagram

8 Expected Outcomes

- Fully functional HTTP caching proxy
- Less than 10ms response time for cached content
- Real-time cache monitoring dashboard
- Demonstrated understanding of TCP internals
- Clean, well-documented, reusable codebase

9 Conclusion

CacheCaught successfully bridges theoretical networking concepts (caching, concurrency, TCP flow/congestion control) with practical implementation. It delivers dramatic performance improvements while serving as an excellent educational platform for understanding modern backend systems.