# Custom Network Proxy Server (HTTP + HTTPS CONNECT)

Project Report

## 1. Abstract

This project implements a custom forward proxy server using Python socket programming. The proxy supports HTTP request forwarding, concurrent multi-client handling via a thread-per-connection model, domain-based blocking using a configurable blocklist, structured logging, and HTTPS support using CONNECT tunneling.

## 2. Objectives

- Implement a forward proxy using TCP sockets (HTTP/1.1).
- Handle multiple clients concurrently using threads.
- Parse HTTP requests to extract method, host, port, and path.
- Forward requests to destination servers and relay responses back.
- Implement rule-based filtering (blocklist) with 403 responses.
- Maintain logs for allowed/blocked requests and transferred byte counts.
- Support HTTPS through CONNECT tunneling (optional extension).

## 3. System Architecture

The proxy runs as a TCP server listening on 127.0.0.1:8888. For each incoming client connection, a dedicated thread is spawned to handle the session.

### 3.1 HTTP Forwarding Flow

1. Accept client TCP connection (browser/curl).
2. Receive request headers until CRLF CRLF (\r\n\r\n).
3. Parse request line and headers (method, target, HTTP version, Host header).
4. Resolve destination host/port/path from absolute-URI or Host + relative-path.
5. Apply blocklist policy. If blocked, respond with 403 Forbidden.
6. Open TCP connection to destination host:port.
7. Rewrite request line to server-form: METHOD /path HTTP/1.1.
8. Forward headers (+ body if Content-Length exists).
9. Relay the destination response back to the client until the server closes.
10. Log the result (ALLOW/BLOCKED/ERROR) and bytes transferred.

### 3.2 HTTPS CONNECT Tunneling Flow

11. Client sends CONNECT host:port HTTP/1.1 to the proxy.
12. Proxy parses the CONNECT target host and port (typically 443).
13. Apply blocklist policy to the CONNECT host. If blocked, respond 403 Forbidden.
14. Proxy opens a TCP connection to host:port.
15. Proxy replies: HTTP/1.1 200 Connection Established.
16. Create a transparent bidirectional tunnel (proxy does not decrypt TLS).
17. Log tunnel bytes in both directions (client→server and server→client).

## 4. Implementation Details

### 4.1 Concurrency Model

A thread-per-connection model is used. The main thread accepts connections and spawns a daemon thread for each client session. This approach is simple and satisfies the requirement to handle multiple clients.

### 4.2 Request Parsing

The proxy reads until \r\n\r\n to obtain complete HTTP headers. Headers are parsed into a dictionary using lowercase keys. The proxy supports both absolute-URI form (common when using a proxy) and relative-path form using the Host header.

### 4.3 Forwarding and Relay

For HTTP requests, the proxy connects to the destination server and forwards a rewritten request whose request line contains only the path. Responses are streamed back to the client in chunks, avoiding full buffering in memory. Total relayed bytes are recorded for logging.

### 4.4 Blocklist Filtering

Blocked domains are defined in config/blocked.txt. Matching is case-insensitive and includes parent-domain blocking (blocking example.com also blocks www.example.com). Blocked requests return HTTP/1.1 403 Forbidden and are logged.

### 4.5 Logging

Each request generates a log entry containing UTC timestamp, decision (ALLOW/BLOCKED/ERROR), client address, target, and byte counts. CONNECT tunnels log c2s and s2c byte counts.

## 5. Testing and Results

The proxy was tested using curl with explicit proxy settings. The following tests were executed:

| Test | Command | Expected Result |
|---|---|---|

| | | |
|---|---|---|
| HTTP Allow | curl -v http://example.com -x http://127.0.0.1:8888 | 200 OK, HTML returned, ALLOW log with bytes |
| HTTP Block | curl -v http://example.com -x http://127.0.0.1:8888 (example.com in blocked.txt) | 403 Forbidden, BLOCKED log |
| HTTPS Allow | curl -v https://example.com -x http://127.0.0.1:8888 | 200 Connection Established, then 200 OK, CONNECT log with c2s/s2c |
| HTTPS Block | curl -v https://example.com -x http://127.0.0.1:8888 (example.com in blocked.txt) | 403 Forbidden, CONNECT tunnel fails, BLOCKED CONNECT log |

Screenshots of outputs and corresponding logs are attached in the Appendix.

## 6. Limitations

- Chunked request bodies are not fully supported (request-body handling assumes Content-Length).
- No caching layer is implemented.
- No authentication or user access control is implemented.
- Designed for HTTP/1.1 over TCP; does not support HTTP/2 or QUIC.

## 7. Future Improvements

- Add LRU-based HTTP response caching with validation headers (ETag/Last-Modified).
- Use a thread pool or async I/O for better scalability.
- Add optional authentication and access control lists (ACLs).
- Extend HTTP edge-case handling (chunked requests, pipelining).

## Appendix A: Evidence (Screenshots)

Here are the screenshots of tests.

- HTTP Allow: curl output (200 OK + HTML).

```
D:\repo>curl -v http://example.com -x http://127.0.0.1:8888
*   Trying 127.0.0.1:8888...
* Established connection to 127.0.0.1 (127.0.0.1 port 8888) from 127.0.0.1 port 58052
* using HTTP/1.x
> GET http://example.com/ HTTP/1.1
> Host: example.com
> User-Agent: curl/8.16.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Request completely sent off
< HTTP/1.1 200 OK
< Date: Thu, 08 Jan 2026 12:00:04 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< CF-RAY: 9bab8849981345af-DEL
< Last-Modified: Sat, 03 Jan 2026 05:43:21 GMT
< Allow: GET, HEAD
< Accept-Ranges: bytes
< Age: 5834
< cf-cache-status: HIT
< Server: cloudflare
<
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initia
l-scale=1"><style>body{background:#eee;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5em}div{
opacity:0.8}a:link,a:visited{color:#348}</style><body><div><h1>Example Domain</h1><p>This domain is for use in documentatio
n examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a>
</div></body></html>
* shutting down connection #0
```

- HTTP Block: curl output (403 Forbidden).

```
D:\repo>curl -v http://example.com -x http://127.0.0.1:8888
*   Trying 127.0.0.1:8888...
* Established connection to 127.0.0.1 (127.0.0.1 port 8888) from 127.0.0.1 port 58056
* using HTTP/1.x
> GET http://example.com/ HTTP/1.1
> Host: example.com
> User-Agent: curl/8.16.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 403 Forbidden
< Connection: close
< Content-Type: text/plain
< Content-Length: 33
<
403 Forbidden (Blocked by proxy)
* we are done reading and this is set to close, stop send
* abort upload
* shutting down connection #0
```

- HTTPS Allow: curl output showing 200 Connection Established and final 200 OK.

```
D:\repo>curl -v https://example.com -x http://127.0.0.1:8888
*   Trying 127.0.0.1:8888...
* CONNECT: no ALPN negotiated
* allocate connect buffer
* Establish HTTP proxy tunnel to example.com:443
> CONNECT example.com:443 HTTP/1.1
> Host: example.com:443
> User-Agent: curl/8.16.0
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 Connection Established
<
* CONNECT phase completed
* CONNECT tunnel established, response 200
* schannel: disabled automatic use of client certificate
* ALPN: curl offers http/1.1
* ALPN: server accepted http/1.1
* Established connection to 127.0.0.1 (127.0.0.1 port 8888) from 127.0.0.1 port 58058
* using HTTP/1.x
> GET / HTTP/1.1
> Host: example.com
> User-Agent: curl/8.16.0
> Accept: */*
>
* Request completely sent off
* schannel: remote party requests renegotiation
* schannel: renegotiating SSL/TLS connection
* schannel: SSL/TLS connection renegotiated
< HTTP/1.1 200 OK
< Date: Thu, 08 Jan 2026 12:00:38 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< CF-RAY: 9bab89216dcb0c66-DEL
< last-modified: Sat, 03 Jan 2026 05:43:21 GMT
< allow: GET, HEAD
< Accept-Ranges: bytes
< Age: 6722
< cf-cache-status: HIT
< Server: cloudflare
<
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee;width:60vw;margin:15vh auto;f
ont-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{color:#348}</style><body><div><h1>Example Domain</h1><p>This domain is for use in documentation examples wi
thout needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a></div></body></html>
* Connection #0 to host 127.0.0.1:8888 left intact
```

- HTTPS Block: curl output showing CONNECT blocked (403).

```
D:\repo>curl -v https://example.com -x http://127.0.0.1:8888
*   Trying 127.0.0.1:8888...
* CONNECT: no ALPN negotiated
* allocate connect buffer
* Establish HTTP proxy tunnel to example.com:443
> CONNECT example.com:443 HTTP/1.1
> Host: example.com:443
> User-Agent: curl/8.16.0
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 403 Forbidden
< Connection: close
< Content-Type: text/plain
< Content-Length: 33
<
* CONNECT tunnel failed, response 403
* closing connection #0
curl: (56) CONNECT tunnel failed, response 403
```

- logs/proxy.log showing ALLOW/BLOCKED entries for HTTP and CONNECT (including bytes/c2s/s2c).

```
2026-01-08 12:00:04 UTC | ALLOW   | 127.0.0.1:58052 -> GET example.com:80/ | bytes=822
2026-01-08 12:00:22 UTC | BLOCKED | 127.0.0.1:58056 -> GET example.com:80/
2026-01-08 12:00:39 UTC | ALLOW   | 127.0.0.1:58058 -> CONNECT example.com:443 | c2s=659 s2c=5699
2026-01-08 12:00:56 UTC | BLOCKED | 127.0.0.1:58060 -> CONNECT example.com:443
```