CSE 4074 – Programming Assignment Report

Socket Programming – HTTP Server and Proxy Server

Project Members:

Cem Eren Kula 150120059

Doğukan Onmaz 150120071

Ferhat Sirkeci 150120067

HTTP Server

Overview

The HttpServer.py file implements a simple HTTP server capable of handling client requests. It leverages multi-threading to handle multiple connections concurrently. The server responds to GET requests, checks the validity of the request URI, and generates appropriate HTTP responses.

Features

1. **Multi-threaded Design**:

- Each client connection is handled in a separate thread, ensuring the server can process multiple requests simultaneously.

2. **Request Parsing**:

- Validates the HTTP method (supports GET only).
- Checks the URI for integer values within a specified range (100 to 20,000).
- Handles non-integer URIs (e.g., favicon.ico).

3. **Response Generation**:

- Constructs HTTP responses based on the status of the request (200 OK, 400 Bad Request, or 501 Not Implemented).
- Allows for optional saving of responses to HTML files in a “Responses” directory.

4. **Port Configurability**:

- The port on which the server listens can be specified via command-line arguments.

Code Details

1. **Request Parsing**:

- The parse_request() function validates the request method and URI.
- Unsupported methods return a 501 Not Implemented status, while invalid URIs return a 400 Bad Request.

2. **Response Generation**:

- The generate_response() function creates HTML responses based on the status code.
- If enabled, it saves responses as files in the Responses directory.

3. **Concurrency**:

- The handle_client() function uses threading to ensure independent handling of client connections.
- A shared counter (response_counter) tracks the number of responses generated, protected by a thread-safe lock.

4. **Server Setup**:

- The start_server() function initializes a socket and listens for incoming connections.
- Client connections are accepted and handled in separate threads.

Potential Enhancements

We can add a cache mechanism to store frequently requested responses, reducing processing time for repeated requests.

Proxy Server

Overview

ProxyServer.py implements a simple proxy server designed to forward HTTP requests to a specified web server. The proxy enforces restrictions on allowed hosts and ports, ensuring controlled access to backend resources. It employs multi-threading to handle multiple client connections concurrently.

Features

1. **Request Handling**:

- Parses incoming HTTP requests, extracting the method, URI, and HTTP version.
- Handles HTTP requests with the http:// scheme, routing them to the appropriate destination.
- Verifies host and port against predefined allowed values.

2. **Request Forwarding**:

- Rewrites the request line and headers before forwarding them to the target web server.
- Establishes a connection to the web server and sends the modified request.

3. **Response Transmission**:

- Receives responses from the web server and relays them back to the client without modification.

4. **Access Control**:

- Implements strict access controls by allowing requests only to specified hosts and ports (localhost:8080 in this implementation).
- Returns an HTTP 403 Forbidden status for disallowed requests.

5. **Multi-threading**:

- Uses threading to handle multiple client connections simultaneously, ensuring high responsiveness.

Code Details

1. **Request Parsing**:

- Extracts and validates the method, URI, and HTTP version.
- Parses the host and port from the URI for routing.

2. **Access Control**:

- The handle_client() function checks if the requested host and port match the allowed configuration (ALLOWED_HOST and ALLOWED_PORT).
- Disallowed requests return a 403 Forbidden response.

3. **Request Forwarding**:

- Constructs a modified HTTP request with updated headers.
- Forwards the request to the specified backend server using a new socket connection.

4. **Response Handling**:

- Relays responses from the backend server back to the client.
- Returns a 502 Bad Gateway response if the backend server is unreachable.

5. **Thread Management**:

- Each client connection is managed in a separate thread using Python's threading module.
- Threads operate independently to ensure concurrent handling of requests.

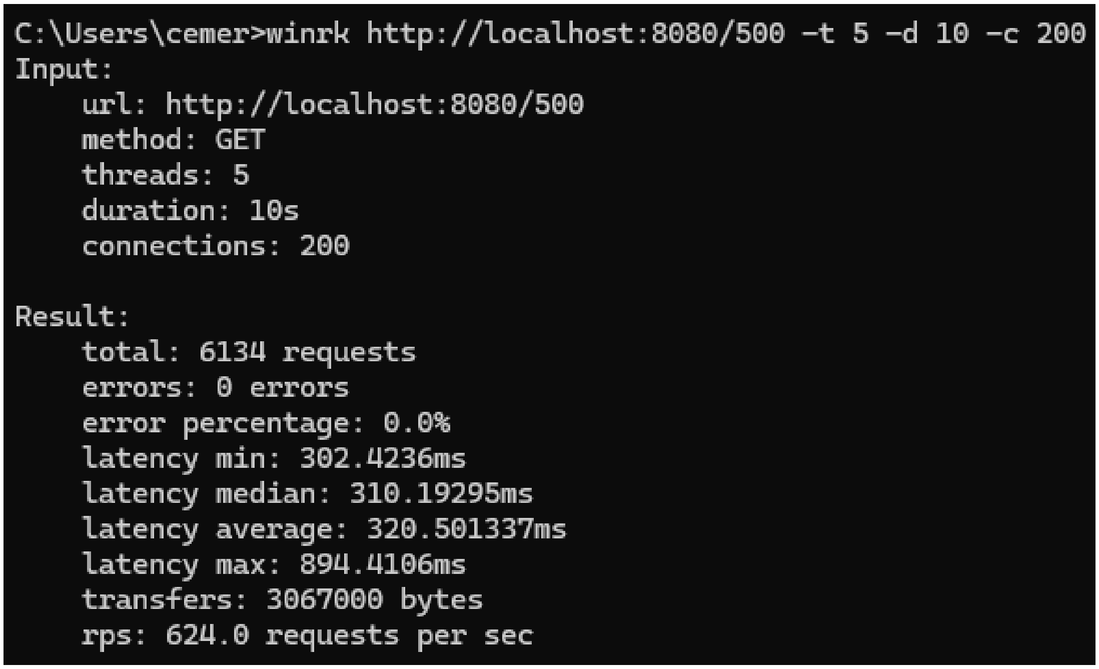
Potential Enhancements

We can also add caching to proxy server.

Stress Testing Proxy Server

<u>Note: In first 6 test we are only changing Number of Threads.</u>

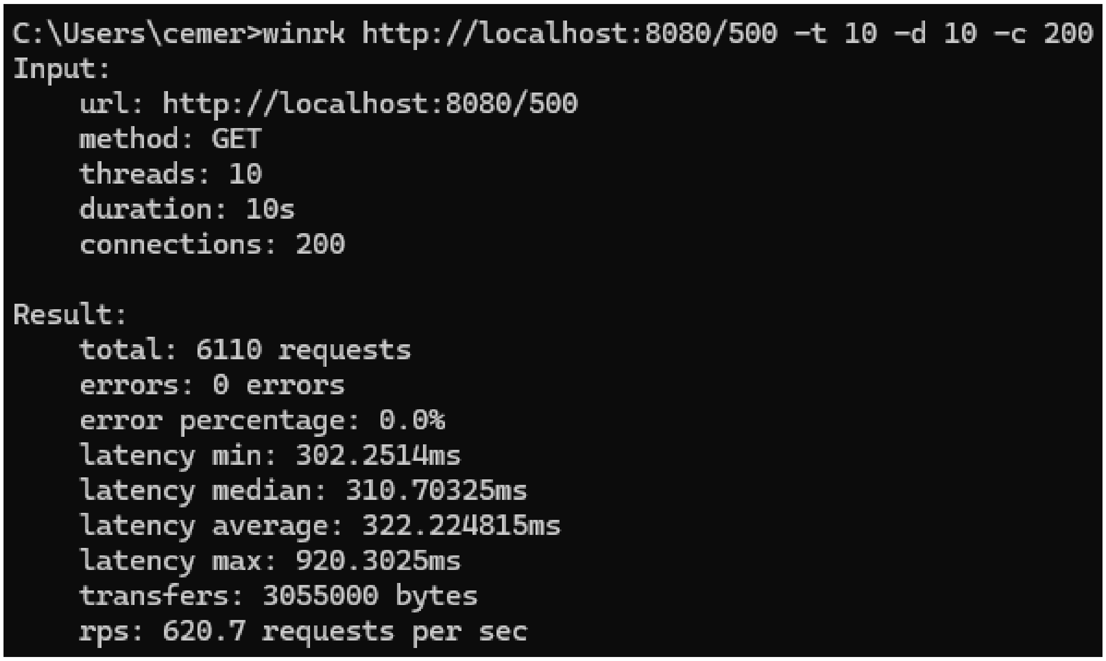
1. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 200

Result:
    total: 6134 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 302.4236ms
    latency median: 310.19295ms
    latency average: 320.501337ms
    latency max: 894.4106ms
    transfers: 3067000 bytes
    rps: 624.0 requests per sec
```

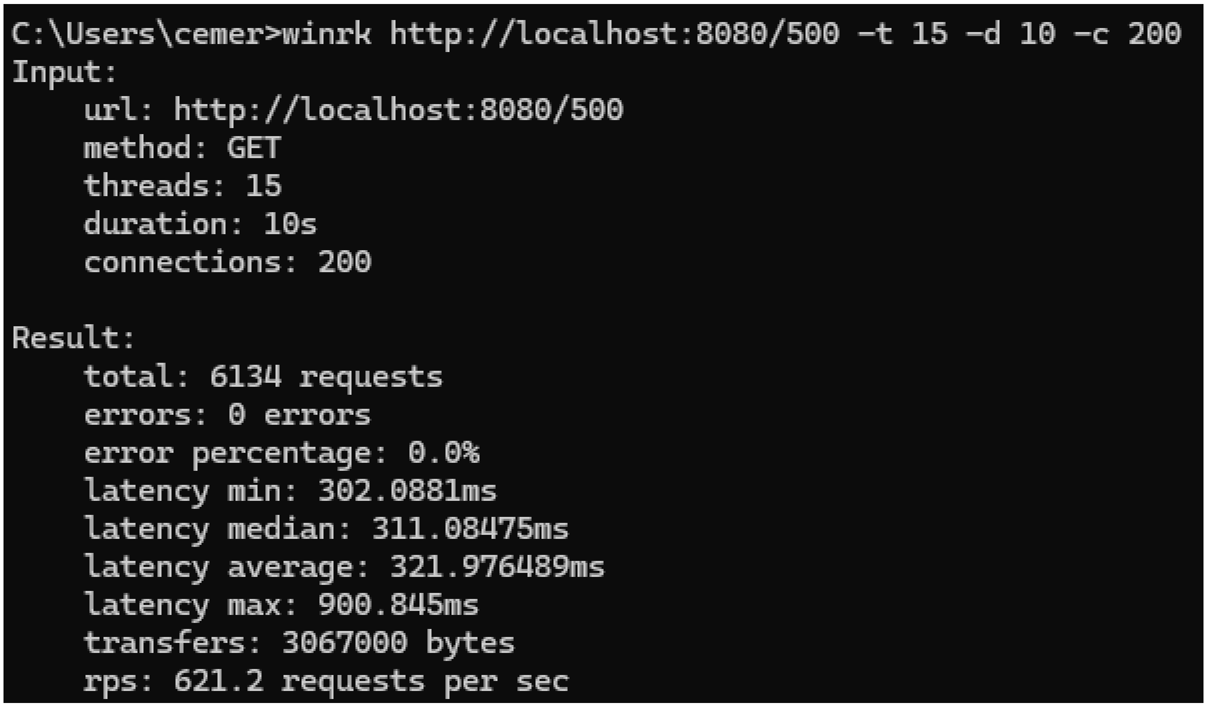
2. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 10 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 10
    duration: 10s
    connections: 200

Result:
    total: 6110 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 302.2514ms
    latency median: 310.70325ms
    latency average: 322.224815ms
    latency max: 920.3025ms
    transfers: 3055000 bytes
    rps: 620.7 requests per sec
```

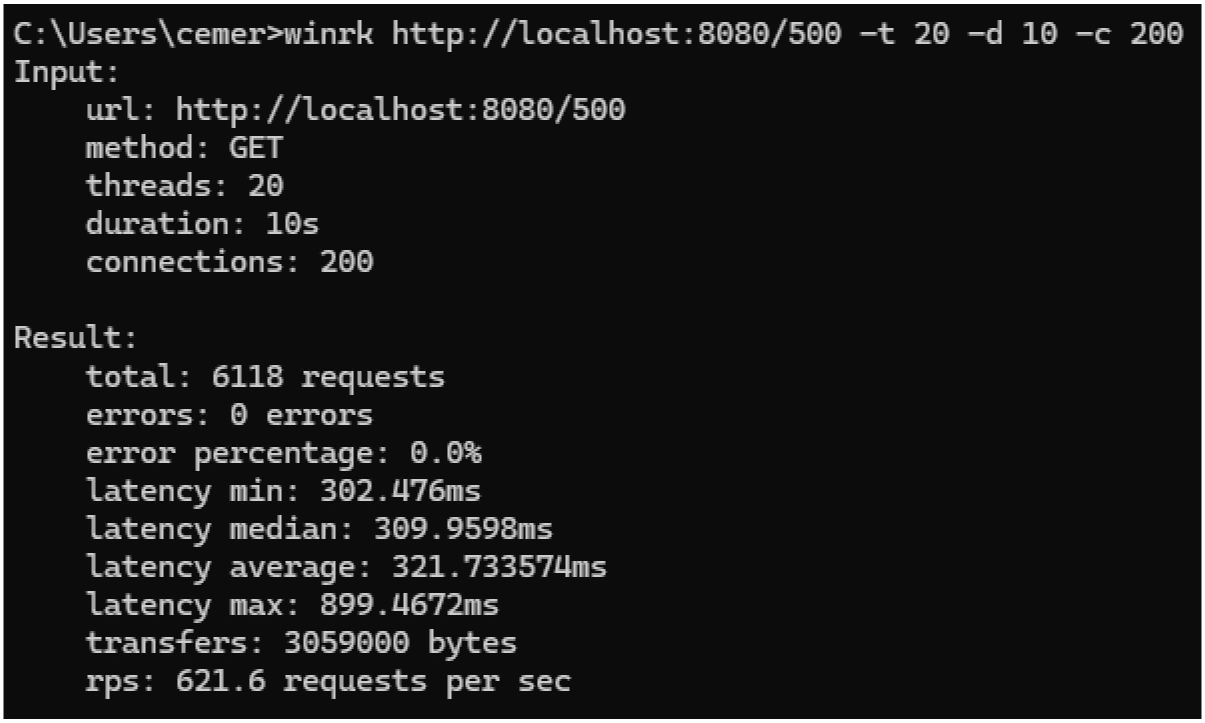
3. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 15 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 15
    duration: 10s
    connections: 200

Result:
    total: 6134 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 302.0881ms
    latency median: 311.08475ms
    latency average: 321.976489ms
    latency max: 900.845ms
    transfers: 3067000 bytes
    rps: 621.2 requests per sec
```

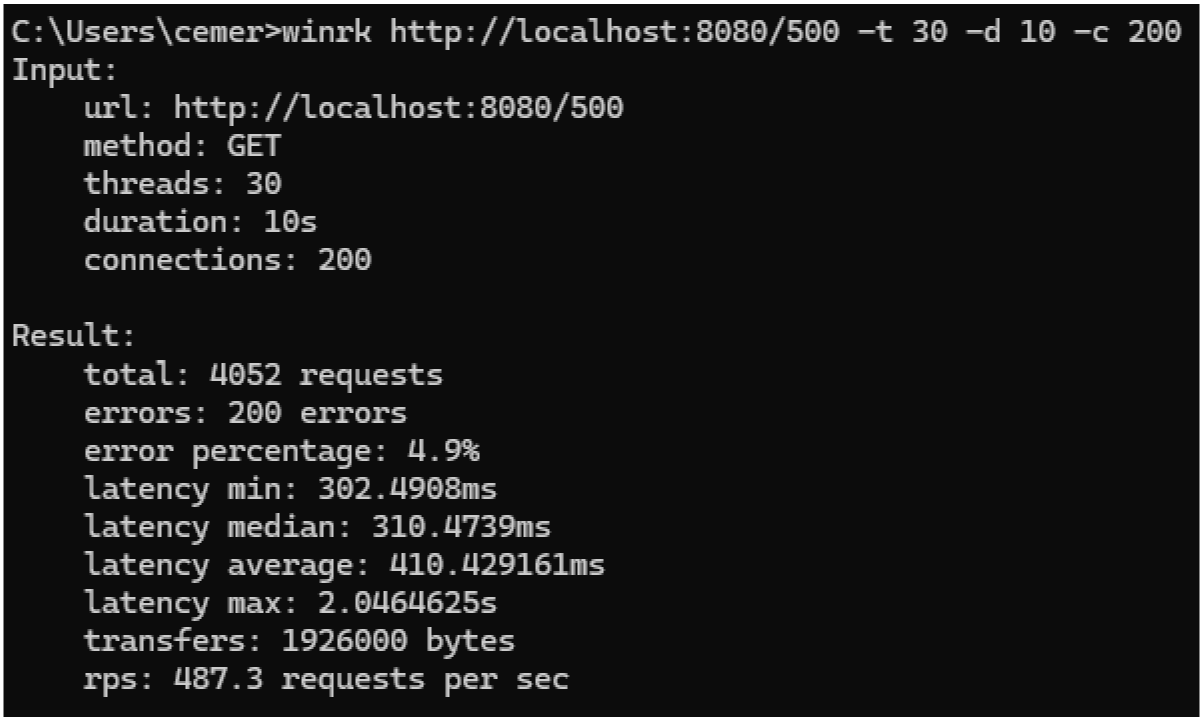
4. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 20 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 20
    duration: 10s
    connections: 200

Result:
    total: 6118 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 302.476ms
    latency median: 309.9598ms
    latency average: 321.733574ms
    latency max: 899.4672ms
    transfers: 3059000 bytes
    rps: 621.6 requests per sec
```

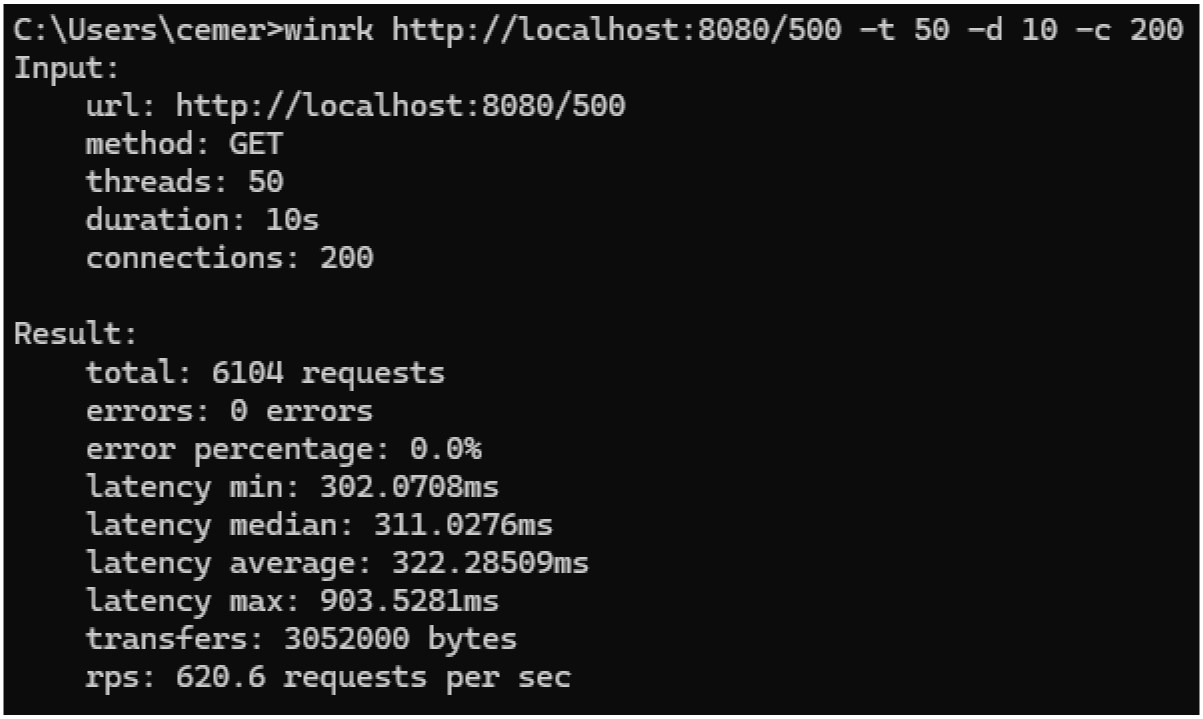
5. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 30 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 30
    duration: 10s
    connections: 200

Result:
    total: 4052 requests
    errors: 200 errors
    error percentage: 4.9%
    latency min: 302.4908ms
    latency median: 310.4739ms
    latency average: 410.429161ms
    latency max: 2.0464625s
    transfers: 1926000 bytes
    rps: 487.3 requests per sec
```

6. **Test**

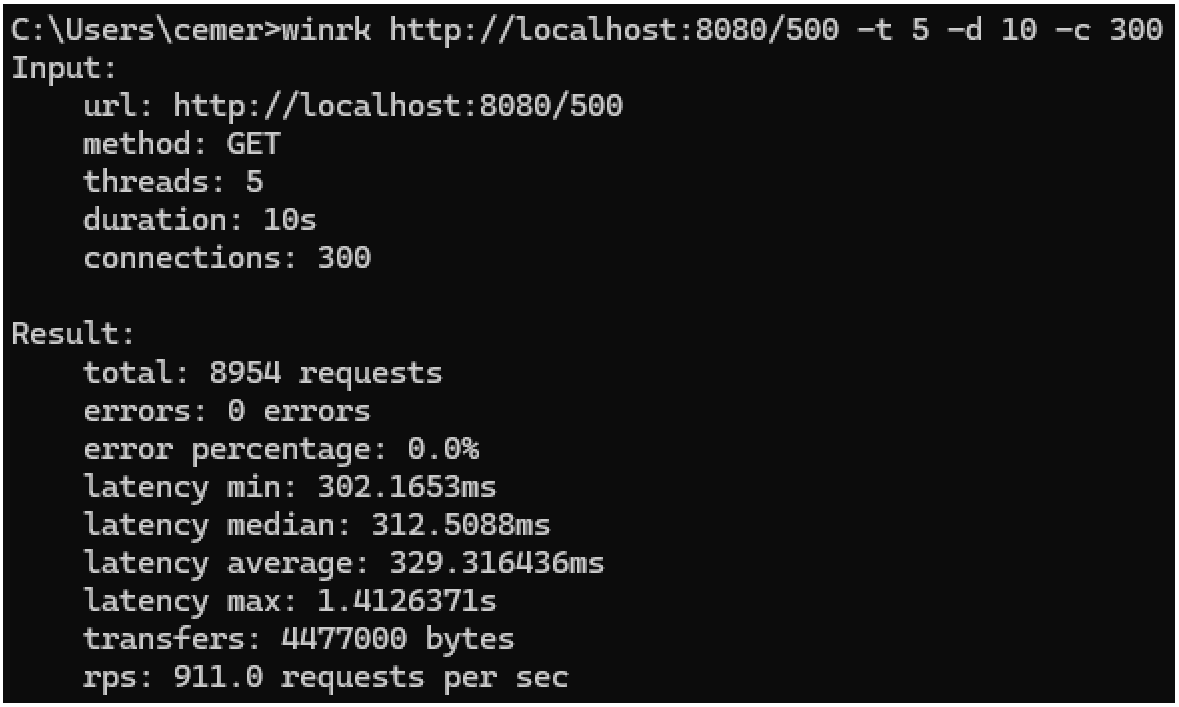


```
C:\Users\cemer>winrk http://localhost:8080/500 -t 50 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 50
    duration: 10s
    connections: 200

Result:
    total: 6104 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 302.0708ms
    latency median: 311.0276ms
    latency average: 322.28509ms
    latency max: 903.5281ms
    transfers: 3052000 bytes
    rps: 620.6 requests per sec
```

Note: Now we are going to change parallel connection number.

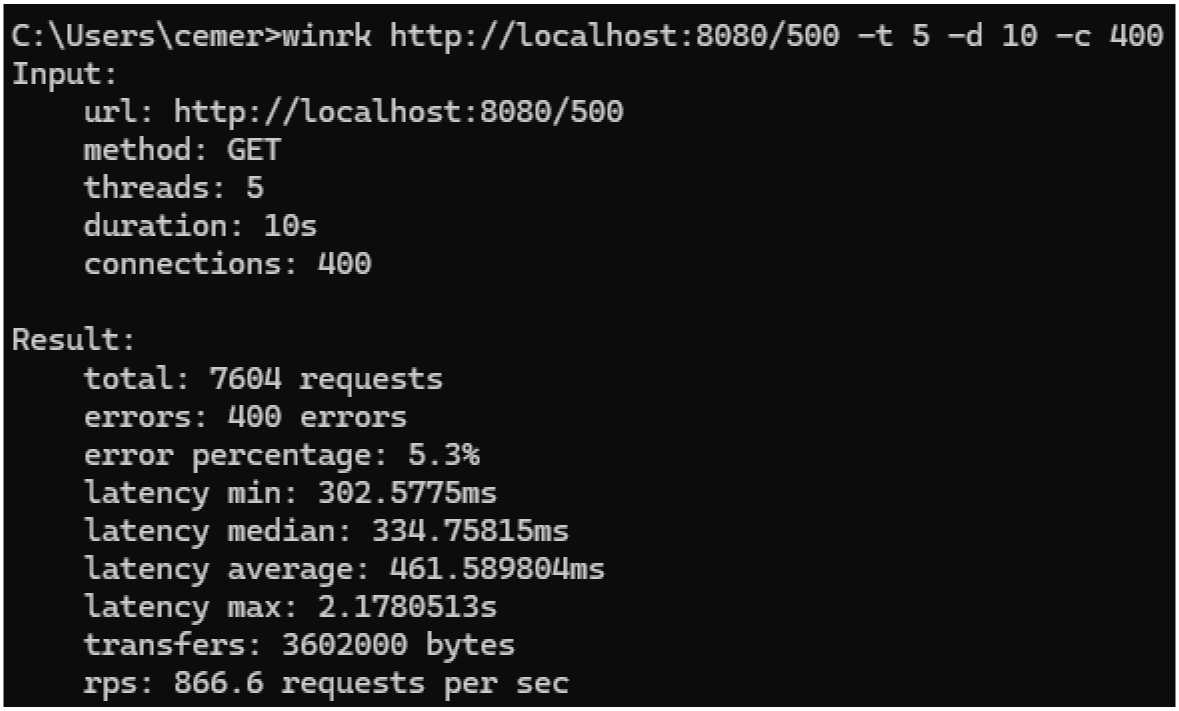
7. Test



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 300
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 300

Result:
    total: 8954 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 302.1653ms
    latency median: 312.5088ms
    latency average: 329.316436ms
    latency max: 1.4126371s
    transfers: 4477000 bytes
    rps: 911.0 requests per sec
```

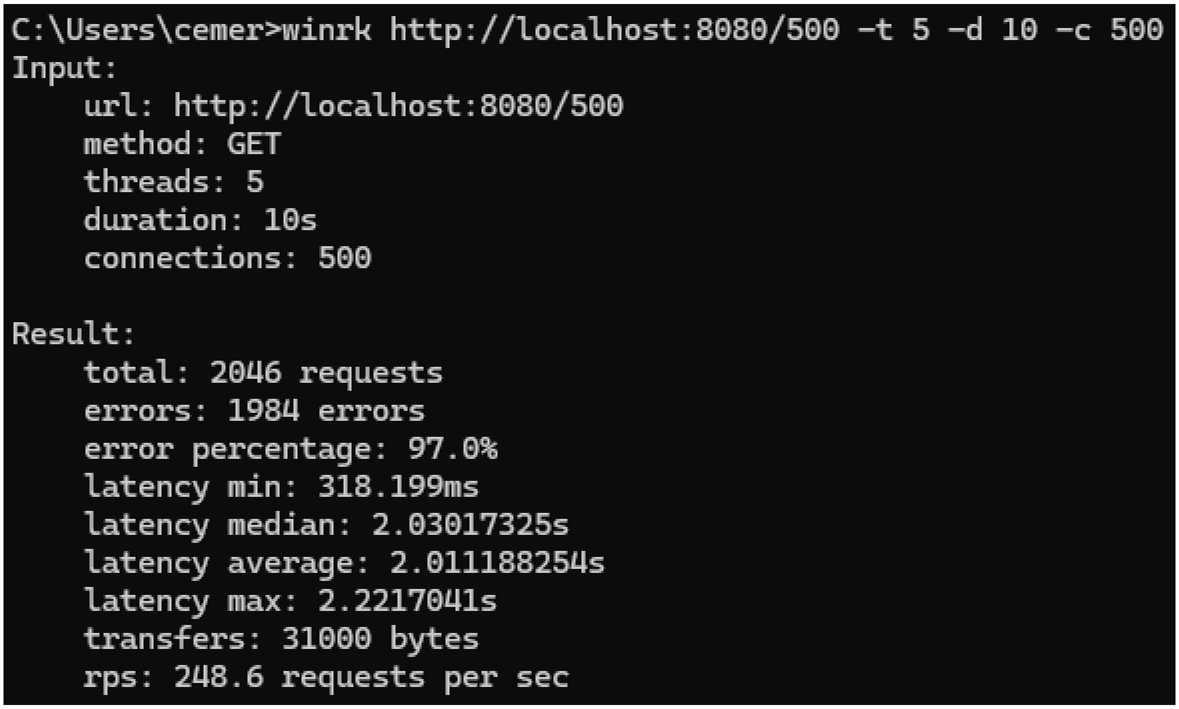
8. Test



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 400
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 400

Result:
    total: 7604 requests
    errors: 400 errors
    error percentage: 5.3%
    latency min: 302.5775ms
    latency median: 334.75815ms
    latency average: 461.589804ms
    latency max: 2.1780513s
    transfers: 3602000 bytes
    rps: 866.6 requests per sec
```

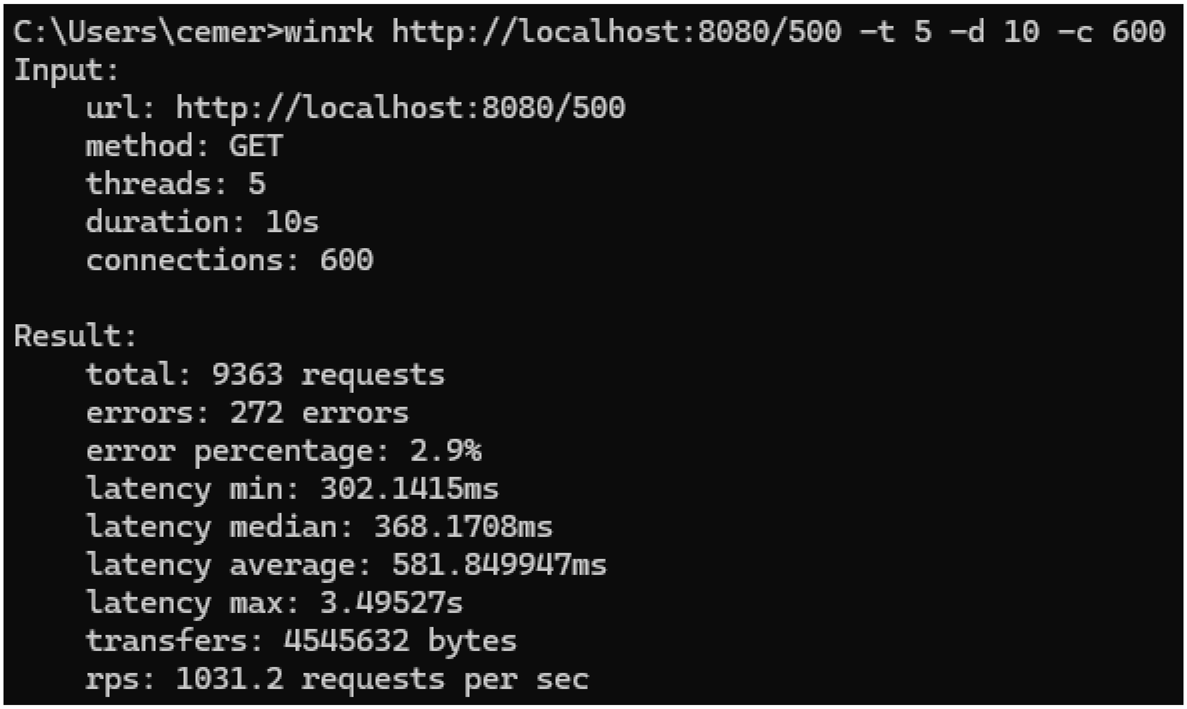
9. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 500
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 500

Result:
    total: 2046 requests
    errors: 1984 errors
    error percentage: 97.0%
    latency min: 318.199ms
    latency median: 2.03017325s
    latency average: 2.011188254s
    latency max: 2.2217041s
    transfers: 31000 bytes
    rps: 248.6 requests per sec
```

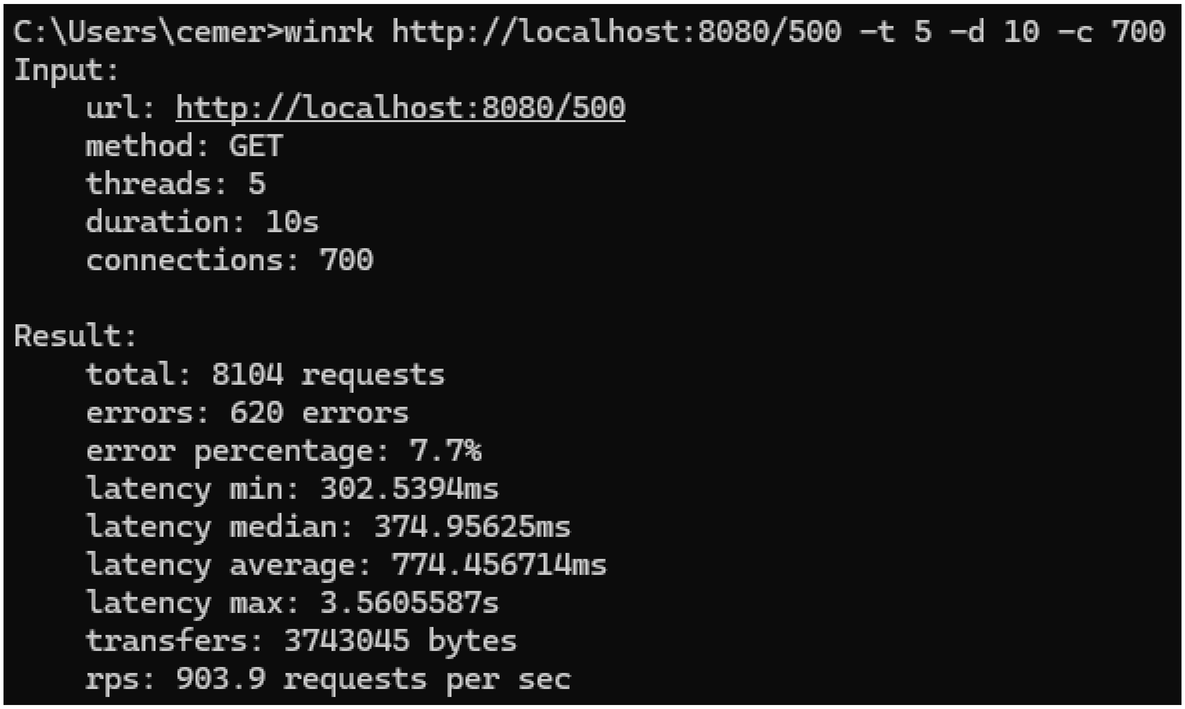
10. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 600
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 600

Result:
    total: 9363 requests
    errors: 272 errors
    error percentage: 2.9%
    latency min: 302.1415ms
    latency median: 368.1708ms
    latency average: 581.849947ms
    latency max: 3.49527s
    transfers: 4545632 bytes
    rps: 1031.2 requests per sec
```

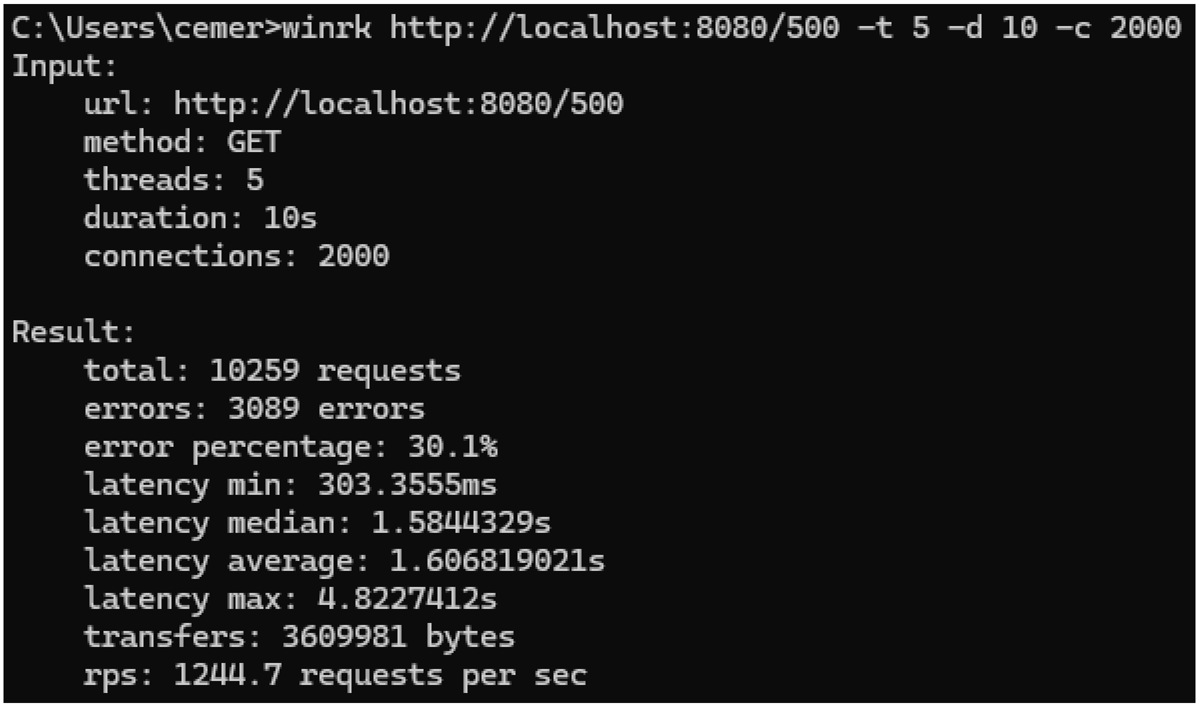
11. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 700
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 700

Result:
    total: 8104 requests
    errors: 620 errors
    error percentage: 7.7%
    latency min: 302.5394ms
    latency median: 374.95625ms
    latency average: 774.456714ms
    latency max: 3.5605587s
    transfers: 3743045 bytes
    rps: 903.9 requests per sec
```

12. **Test**

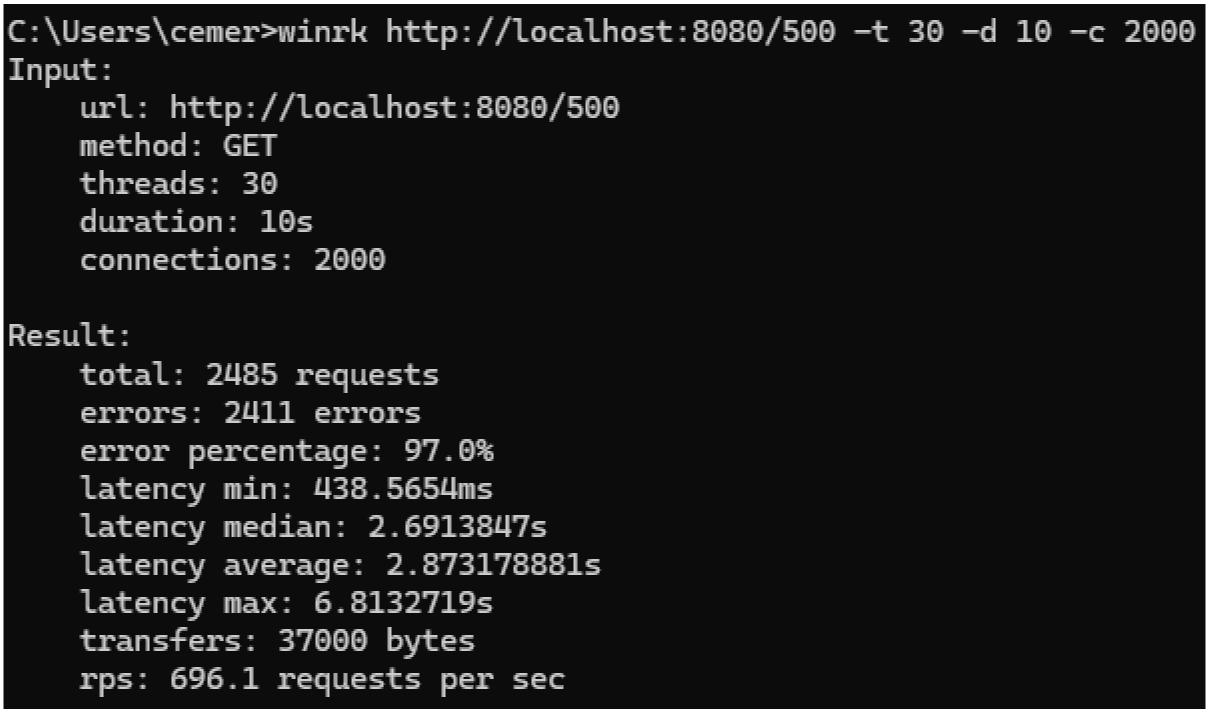


```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 2000
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 2000

Result:
    total: 10259 requests
    errors: 3089 errors
    error percentage: 30.1%
    latency min: 303.3555ms
    latency median: 1.5844329s
    latency average: 1.606819021s
    latency max: 4.8227412s
    transfers: 3609981 bytes
    rps: 1244.7 requests per sec
```

Note: Now we are going to change parallel connection number and thread number at the same time.

13. **Test**



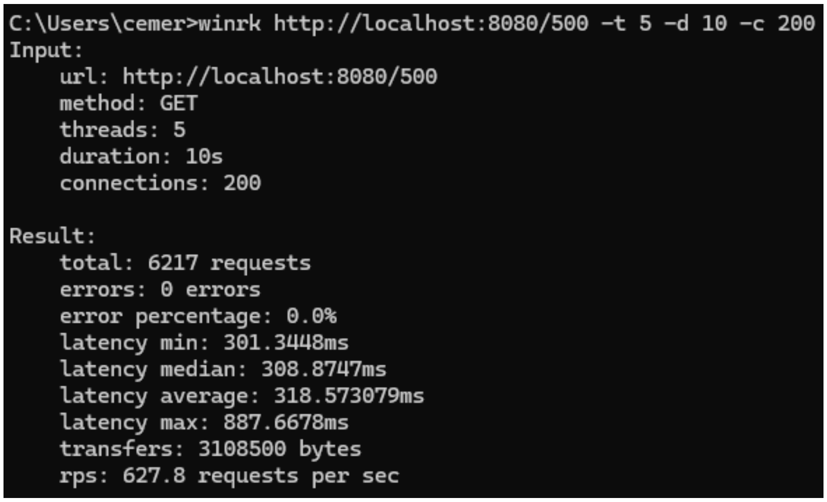
```
C:\Users\cemer>winrk http://localhost:8080/500 -t 30 -d 10 -c 2000
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 30
    duration: 10s
    connections: 2000

Result:
    total: 2485 requests
    errors: 2411 errors
    error percentage: 97.0%
    latency min: 438.5654ms
    latency median: 2.6913847s
    latency average: 2.873178881s
    latency max: 6.8132719s
    transfers: 37000 bytes
    rps: 696.1 requests per sec
```

Stress Testing HTTP Server

Note: We are going to change only Number of Threads.

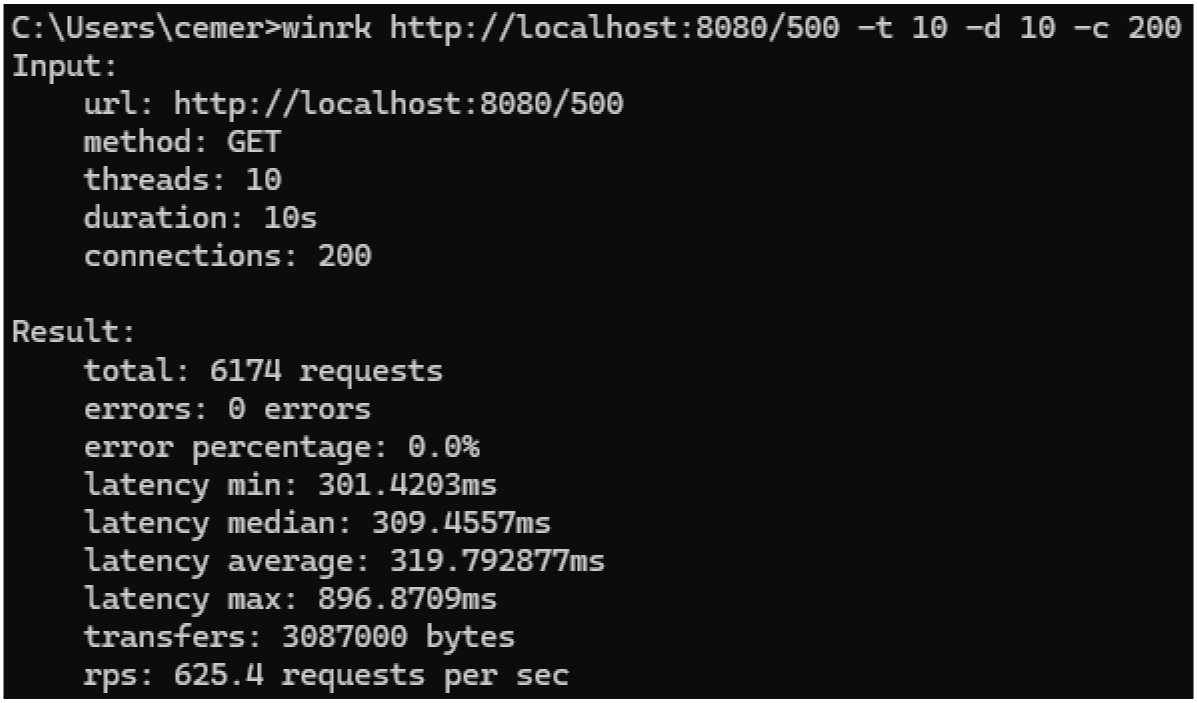
1. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 200

Result:
    total: 6217 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 301.3448ms
    latency median: 308.8747ms
    latency average: 318.573079ms
    latency max: 887.6678ms
    transfers: 3108500 bytes
    rps: 627.8 requests per sec
```

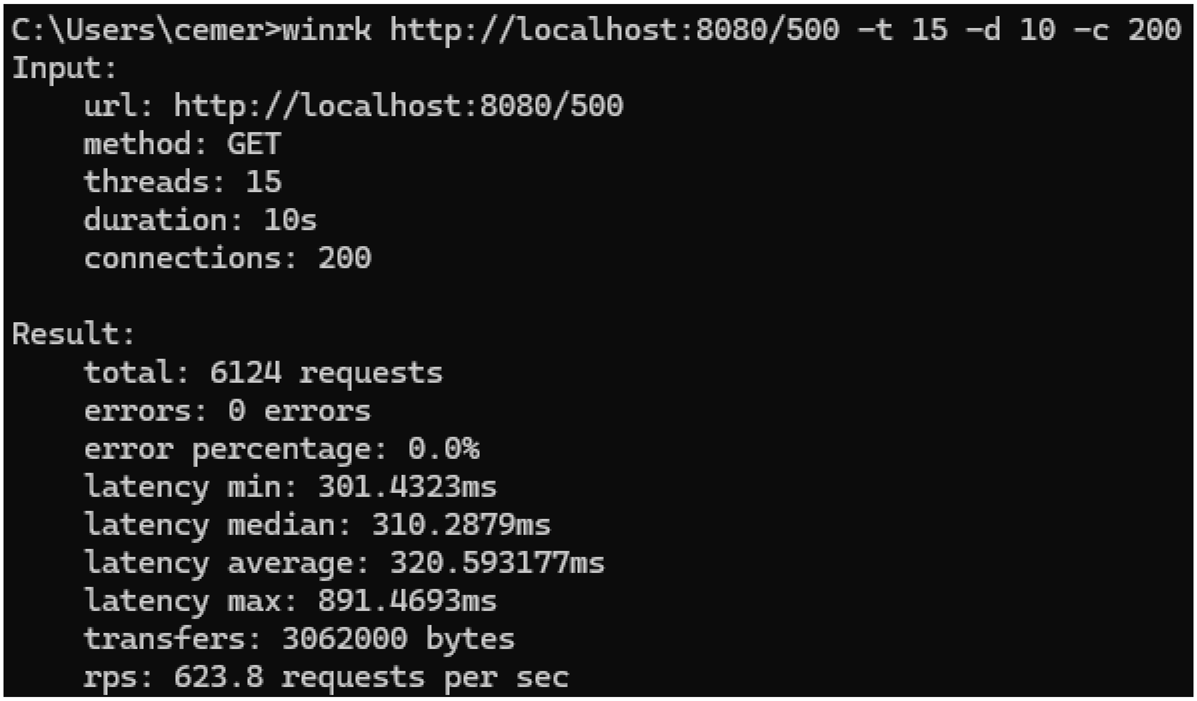
2. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 10 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 10
    duration: 10s
    connections: 200

Result:
    total: 6174 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 301.4203ms
    latency median: 309.4557ms
    latency average: 319.792877ms
    latency max: 896.8709ms
    transfers: 3087000 bytes
    rps: 625.4 requests per sec
```

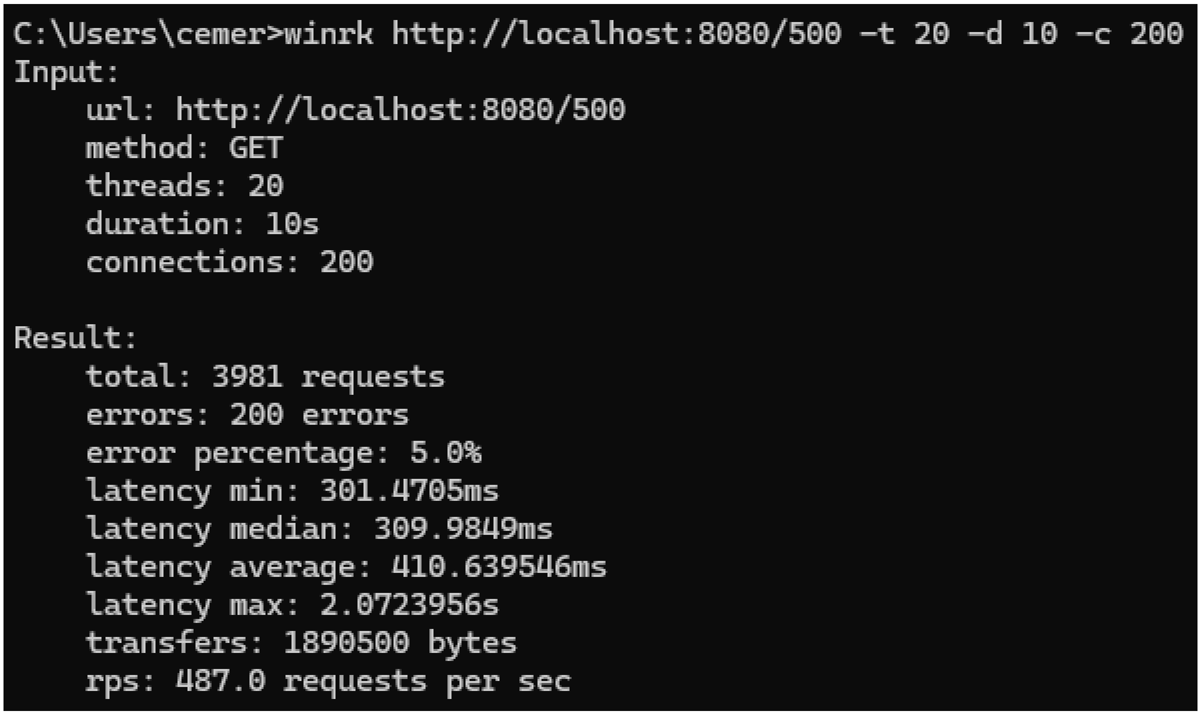
3. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 15 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 15
    duration: 10s
    connections: 200

Result:
    total: 6124 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 301.4323ms
    latency median: 310.2879ms
    latency average: 320.593177ms
    latency max: 891.4693ms
    transfers: 3062000 bytes
    rps: 623.8 requests per sec
```

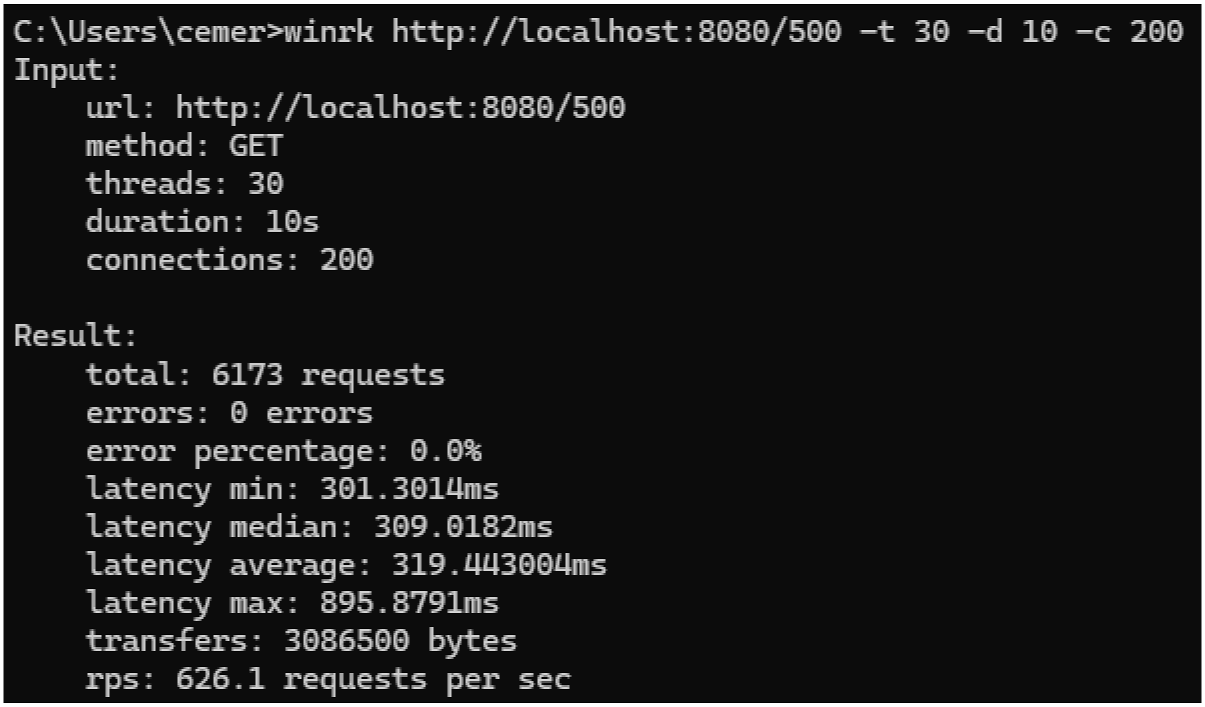
4. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 20 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 20
    duration: 10s
    connections: 200

Result:
    total: 3981 requests
    errors: 200 errors
    error percentage: 5.0%
    latency min: 301.4705ms
    latency median: 309.9849ms
    latency average: 410.639546ms
    latency max: 2.0723956s
    transfers: 1890500 bytes
    rps: 487.0 requests per sec
```

5. **Test**

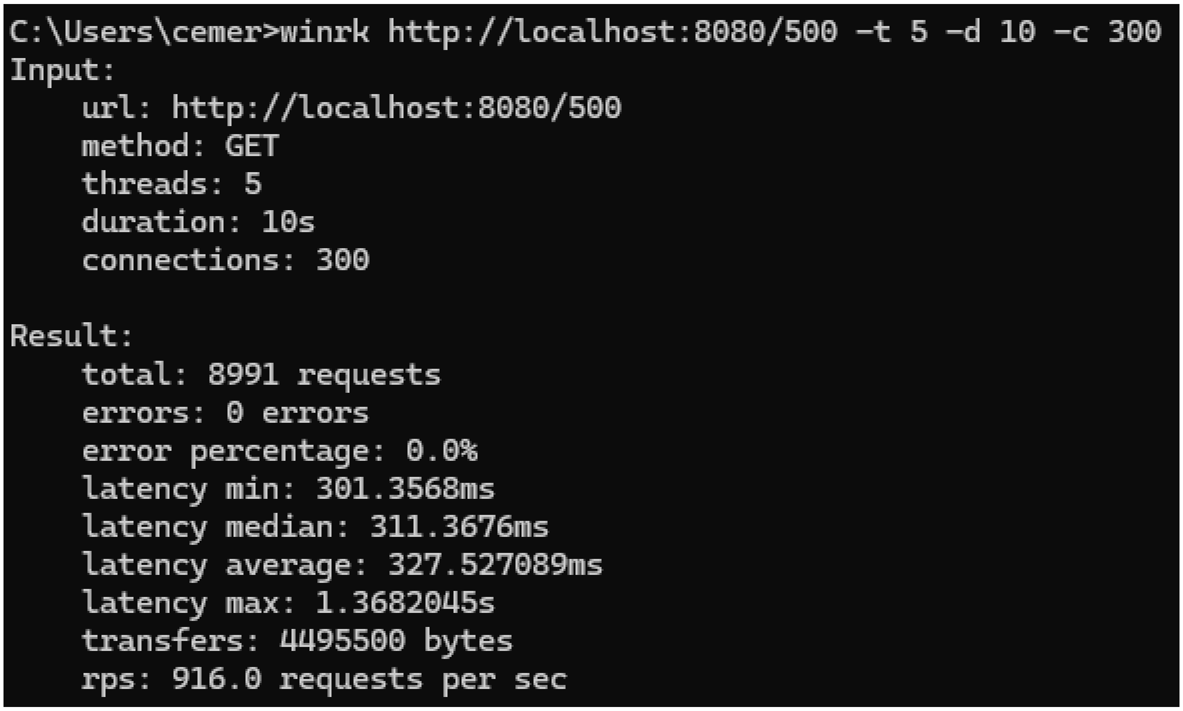


```
C:\Users\cemer>winrk http://localhost:8080/500 -t 30 -d 10 -c 200
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 30
    duration: 10s
    connections: 200

Result:
    total: 6173 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 301.3014ms
    latency median: 309.0182ms
    latency average: 319.443004ms
    latency max: 895.8791ms
    transfers: 3086500 bytes
    rps: 626.1 requests per sec
```

Note: Now we are going to change only parallel connection number.

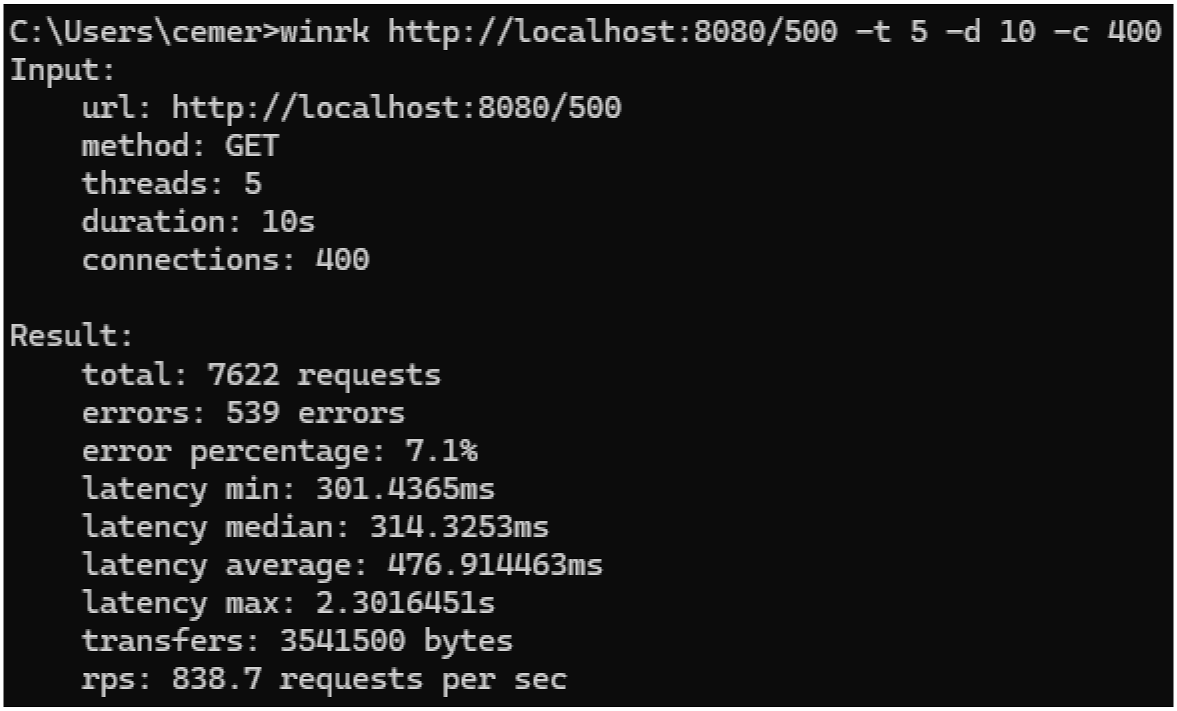
6. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 300
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 300

Result:
    total: 8991 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 301.3568ms
    latency median: 311.3676ms
    latency average: 327.527089ms
    latency max: 1.3682045s
    transfers: 4495500 bytes
    rps: 916.0 requests per sec
```

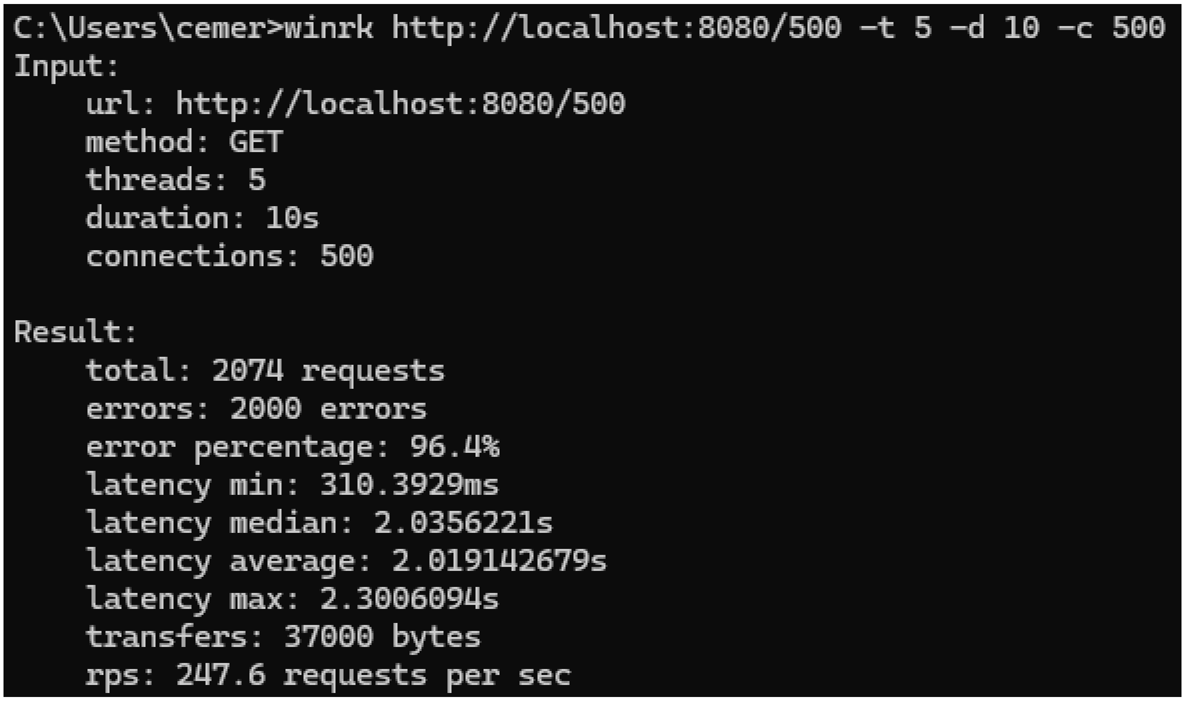
7. **Test**



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 400
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 400

Result:
    total: 7622 requests
    errors: 539 errors
    error percentage: 7.1%
    latency min: 301.4365ms
    latency median: 314.3253ms
    latency average: 476.914463ms
    latency max: 2.3016451s
    transfers: 3541500 bytes
    rps: 838.7 requests per sec
```

8. Test

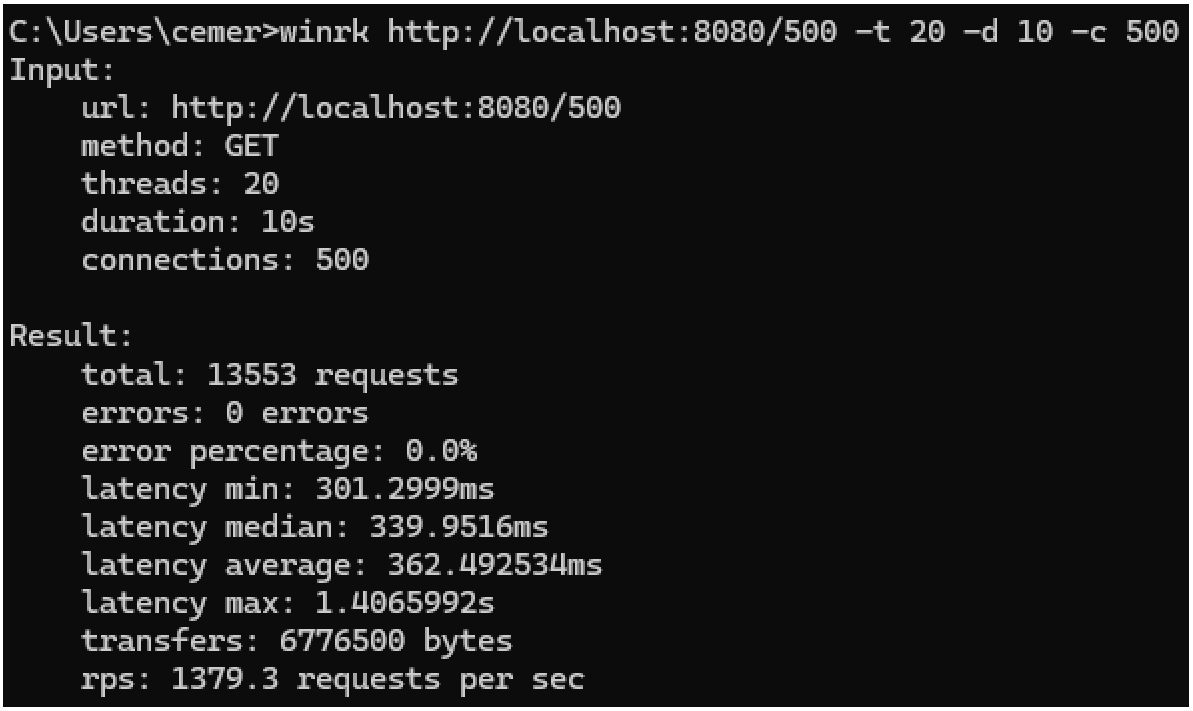


```
C:\Users\cemer>winrk http://localhost:8080/500 -t 5 -d 10 -c 500
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 5
    duration: 10s
    connections: 500

Result:
    total: 2074 requests
    errors: 2000 errors
    error percentage: 96.4%
    latency min: 310.3929ms
    latency median: 2.0356221s
    latency average: 2.019142679s
    latency max: 2.3006094s
    transfers: 37000 bytes
    rps: 247.6 requests per sec
```

Note: Now we are going to also increase thread number with 500 connections.

9. Test



```
C:\Users\cemer>winrk http://localhost:8080/500 -t 20 -d 10 -c 500
Input:
    url: http://localhost:8080/500
    method: GET
    threads: 20
    duration: 10s
    connections: 500

Result:
    total: 13553 requests
    errors: 0 errors
    error percentage: 0.0%
    latency min: 301.2999ms
    latency median: 339.9516ms
    latency average: 362.492534ms
    latency max: 1.4065992s
    transfers: 6776500 bytes
    rps: 1379.3 requests per sec
```

Conclusion

When conducting stress tests on the proxy server, the following behaviors were observed:

1. **Impact of Increasing Parallel Connections**:
   - As the number of parallel connections increases, both the error rate and average latency rise significantly.
   - This occurs because the server struggles to manage a higher volume of simultaneous requests with a fixed number of threads.
2. **Effect of Increasing Thread Count**:
   - By increasing the number of threads, the server can handle a greater degree of concurrency more efficiently.
   - Higher thread counts allow the server to respond to simultaneous requests with reduced latency and error rates.
3. **Thread Count vs. Latency and Errors**:
   - Simply increasing the thread count without increasing the number of parallel connections does not affect latency or error rates.
   - This demonstrates that the server's performance is primarily limited by the demand for concurrency rather than the thread pool size under low-load conditions.

These insights highlight the importance of tuning thread count relative to expected traffic patterns for optimal performance during high-concurrency scenarios.