Implementation

# Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Tactic | Availability | Performance | Security |
| Asynchronous | X | X |  |
| Persistent |  | X | X |
| ThreadPools | X | X |  |
| DoS Detection | X | X | X |
| Host Blocking |  |  | X |
| 304 Not Modified | X |  |  |
| (5 overlaps) | 4 | 4 | 3 |

# Asynchronous connection handling

Using Semaphore and await Listener.AcceptSocketAsync(), we are able to allow the server to accept multiple connections and handle them in an asynchronous manner.

## Affected quality attributes

Availability – The server can simultaneously serve as many clients as the semaphore is configured to allow, at the expense of slower response times.

Performance – The server can limit the number of connections it simultaneously serves to decrease response times.

# Persistent HTTP request handling

Using the HTTP Connection header, we accept persistent requests.

## Affected quality attributes

Performance – The server can keep persistent requests alive, which reduces round-trip time and allows for less resource turnover (which has an additional time benefit).

Security – The server can complete the TCP handshake process fewer times per client when persistent connections are used.

# ThreadPools

Using ThreadPool.QueueUserWorkItem, we remove much of the overhead of manually starting threads.

## Affected quality attributes

Availability – The server can serve more connections per second because of decreased overhead.

Performance – The server can better handle repeated connection requests using a pool of threads managed by the framework.

# DoS Detection

Using synchronized access to a public Dictionary<IPEndPoint, int>, we can track the amount of connections to a single client at any given time. If that number exceeds 25, we immediately close the socket without sending a response.

## Affected quality attributes

Availability – The server can serve more unique clients per second because it is less tied up with repeated requests from misbehaved clients.

Performance – The server can use file system resources for valid requests and isn’t forced to continually send a misbehaving client the same response.

Security – Misbehaving clients are recognized and throttled.

# Host Blocking

Using synchronized access to a public List<IPAddress>, we can manually or programmatically maintain a list of IP addresses not permitted to use the server. If those hosts connect, we immediately close the socket without sending a response.

## Affected quality attributes

Security – Misbehaving clients are recognized and blocked.

# 304 Not Modified

By checking last modified dates and using the 304 Not Modified header, we prevent sending pages that the client has already cached.

## Affected quality attributes

Availability – The server can serve more unique clients per second because it doesn’t have to lock on the same file for repeated requests.

Evaluations

# Availability

|  |  |  |
| --- | --- | --- |
| Measurement | Before | After |
| Non-Threaded DoS Client Service Rate | ~180 | ~1150 |
| ThreadPool DoS Client Service Rate (default settings) | Client unusable because of server crashes and file access issues | ~5-10 |

# Performance

|  |  |  |
| --- | --- | --- |
| Measurement | Before | After |
| Miss Rate (measured using exception counter in ThreadPool DoS Client) | Unmeasureable: server crashes due to file access issues | 100 threads \*  2 connections per second \*  30 seconds /  870 misses =  **14.5%** |
| Throughput/Service Rate | See Availability | See Availability |

# Security

|  |  |  |
| --- | --- | --- |
| Measurement | Before | After |
| Average Detection Time (measured using 5 trial runs and setting breakpoint at DoS detected line of ConnectionHandler) | n/a | 3 seconds, assuming 100 active connections from same IP Address is an “attack” |

Reaction and recovery time are basically unmeasureable – the DoS detection code just closes the socket and returns the thread to the pool.