Server – improvements

# 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 tactics

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.

## Evaluation

Service Rate before modification: ~180

Service Rate with 64-connection Semaphore: ~220

Service Rate with 64-connection Semaphore and 10ms connection interval: ~290

# Persistent HTTP request handling

Using the HTTP Connection header, we accept persistent requests.

## Affected tactics

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).

## Evaluation

Service Rate after previous modification: ~220

Service Rate with 10ms connection interval: ~450

# ThreadPools

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

## Affected tactics

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.

## Evaluation

Service Rate after previous modification: ~450

Service Rate with 10ms connection interval: ~1150