CS3031 - Advanced Telecommunications

Web Proxy Server - Report

Introduction

The task of this assignment was to develop a web proxy server which would run on a local machine fetching items from the web on behalf of a web client instead of the client fetching them directly. This proxy should also implement caching and access control. The proxy must be able to respond to both HTTP, HTTPS requests and also support WebSocket connections. The proxy should also allow for web pages to be blocked and unblocked dynamically via a management console. Requests must be cached as they are received in order to save bandwidth and relay responses efficiently.

- Brandon Dooley (#16327446)

Approach

I decided to build my web proxy server in Node.JS, the reason being that it was designed to build scalable network applications. Node.JS allows for users to efficiently set up client-server architectures and provides many extremely useful libraries that I felt greatly benefitted this use case.

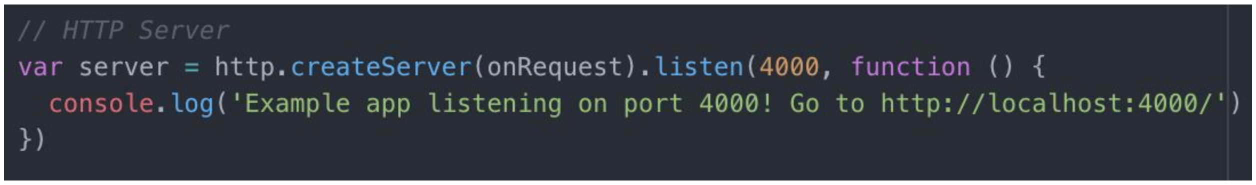
"Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node are free from worries of dead-locking the process, since there are no locks. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, scalable systems are very reasonable to develop in Node." - *Node.JS Website*

Node.JS is now a widely used framework across modern web servers and API's. It sits on the application layer and allows the handling of network packets and requests to be dealt with in a seamlessly easy and efficient manner.

Design

Web Server

For my proxy server I built a web server using Node.JS's HTTP library.

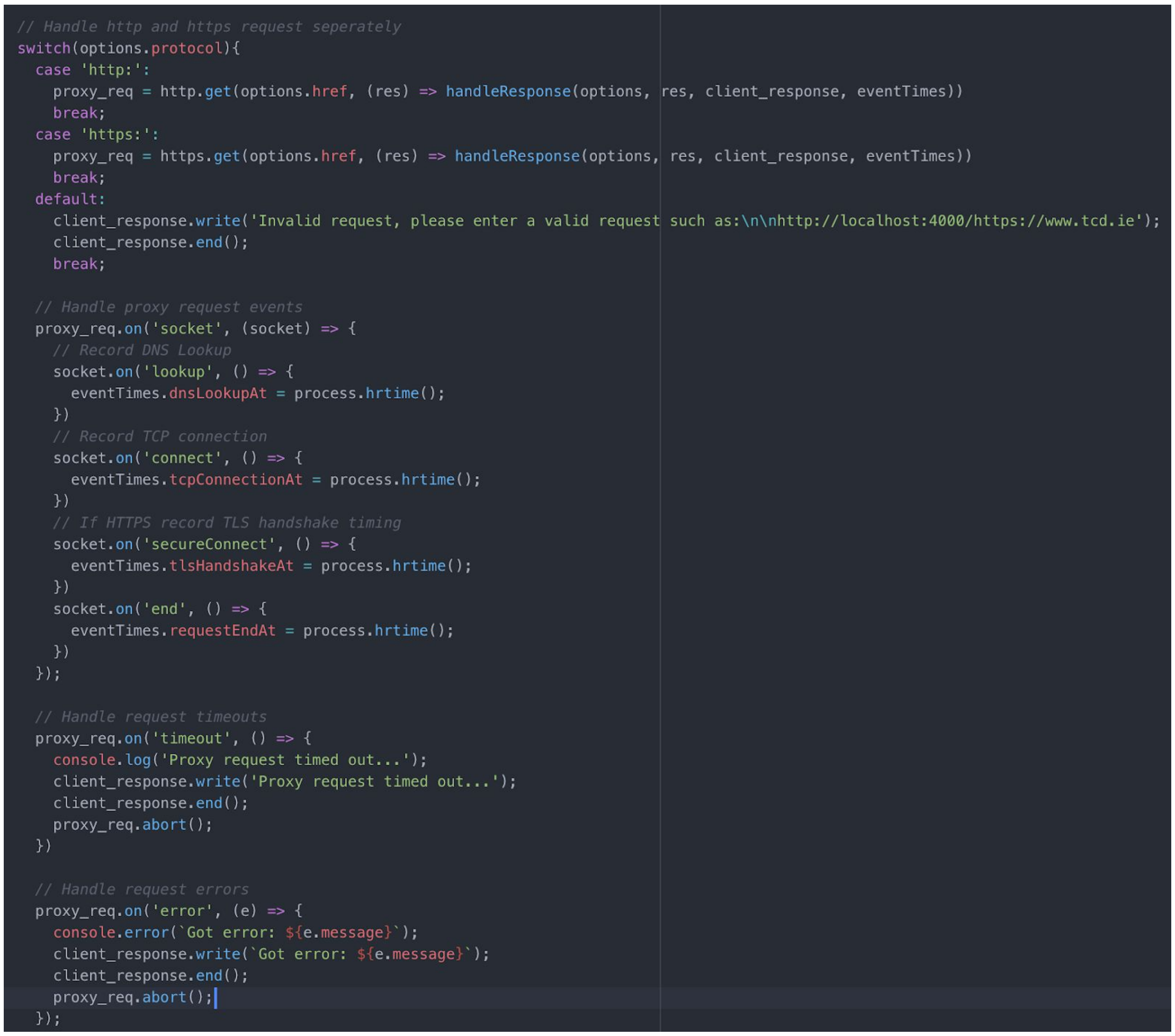


```
// HTTP Server
var server = http.createServer(onRequest).listen(4000, function () {
  console.log('Example app listening on port 4000! Go to http://localhost:4000/')
})
```

This creates an asynchronous server listening on port 4000 of the local machine. Upon receiving requests it uses a callback function named *onRequest*.

Handling Requests

As mentioned above when the server receives a request on port 4000 it then passes this request to the callback function *onRequest*. Within this callback the first thing that is that the URL is not blocked. This URL blocker was implemented using a basic hashtable. The next thing that is checked is the protocol of the request i.e HTTP or HTTPS. Based on that it then uses either Node.JS's HTTP or HTTPS library to send the request. Once it has received a response it then uses a callback function called *handleResponse* to correctly handle the response and implement caching etc.

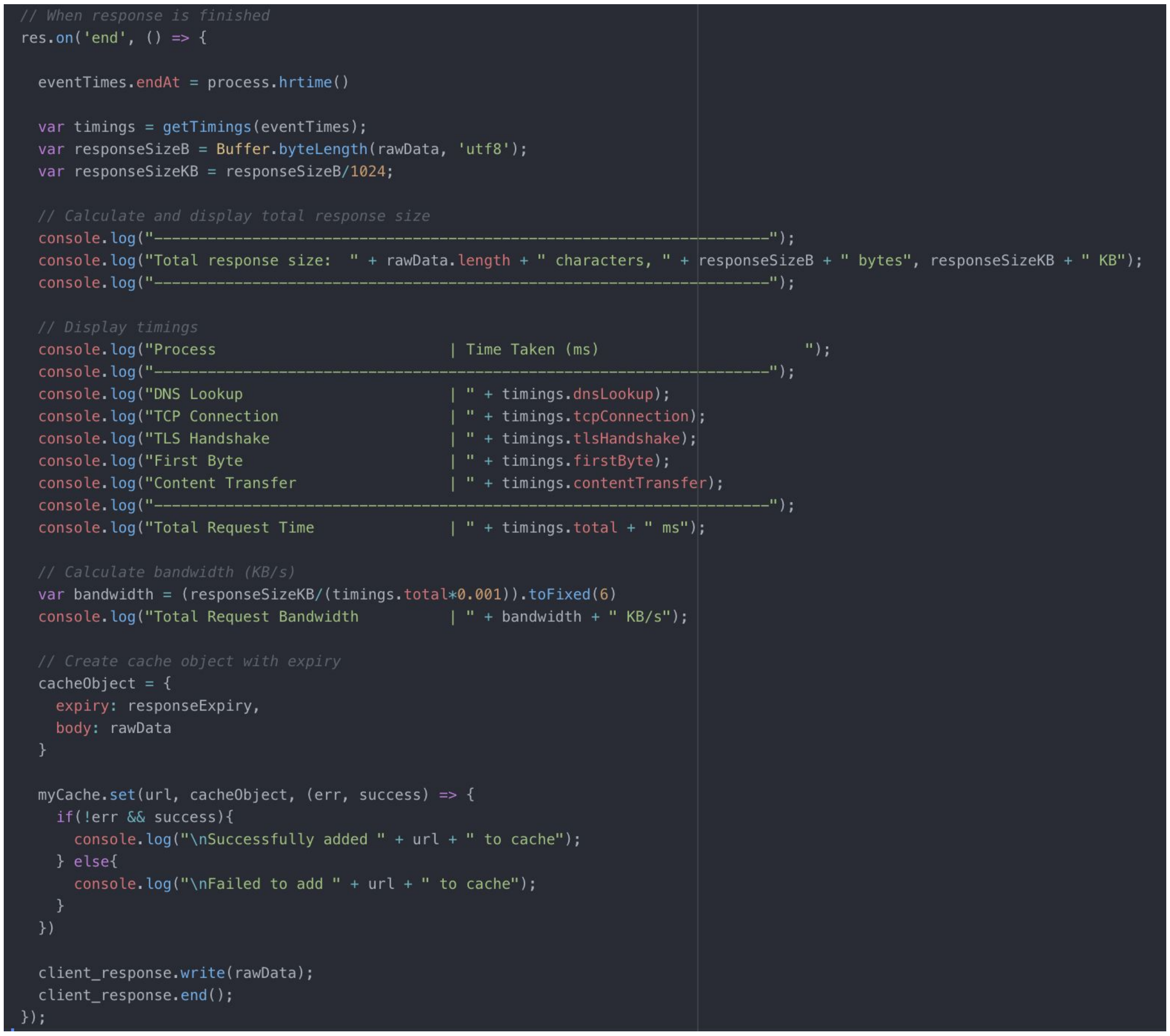


```
// Handle http and https request seperately
switch(options.protocol){
  case 'http:':
    proxy_req = http.get(options.href, (res) => handleResponse(options, res, client_response, eventTimes))
    break;
  case 'https:':
    proxy_req = https.get(options.href, (res) => handleResponse(options, res, client_response, eventTimes))
    break;
  default:
    client_response.write('Invalid request, please enter a valid request such as:\n\nhttp://localhost:4000/https://www.tcd.ie');
    client_response.end();
    break;

  // Handle proxy request events
  proxy_req.on('socket', (socket) => {
    // Record DNS Lookup
    socket.on('lookup', () => {
      eventTimes.dnsLookupAt = process.hrtime();
    })
    // Record TCP connection
    socket.on('connect', () => {
      eventTimes.tcpConnectionAt = process.hrtime();
    })
    // If HTTPS record TLS handshake timing
    socket.on('secureConnect', () => {
      eventTimes.tlsHandshakeAt = process.hrtime();
    })
    socket.on('end', () => {
      eventTimes.requestEndAt = process.hrtime();
    })
  });

  // Handle request timeouts
  proxy_req.on('timeout', () => {
    console.log('Proxy request timed out...');
    client_response.write('Proxy request timed out...');
    client_response.end();
    proxy_req.abort();
  })

  // Handle request errors
  proxy_req.on('error', (e) => {
    console.error(`Got error: ${e.message}`);
    client_response.write(`Got error: ${e.message}`);
    client_response.end();
    proxy_req.abort();
  });
```

Handling Responses

Responses from the proxied requests are handled using the *handleResponse* function. The first thing that is checked from the response is the status code within the header. If the status code of the response is not 200 (success code) then a response is sent to the client informing them of an error and displaying the relevant error message.

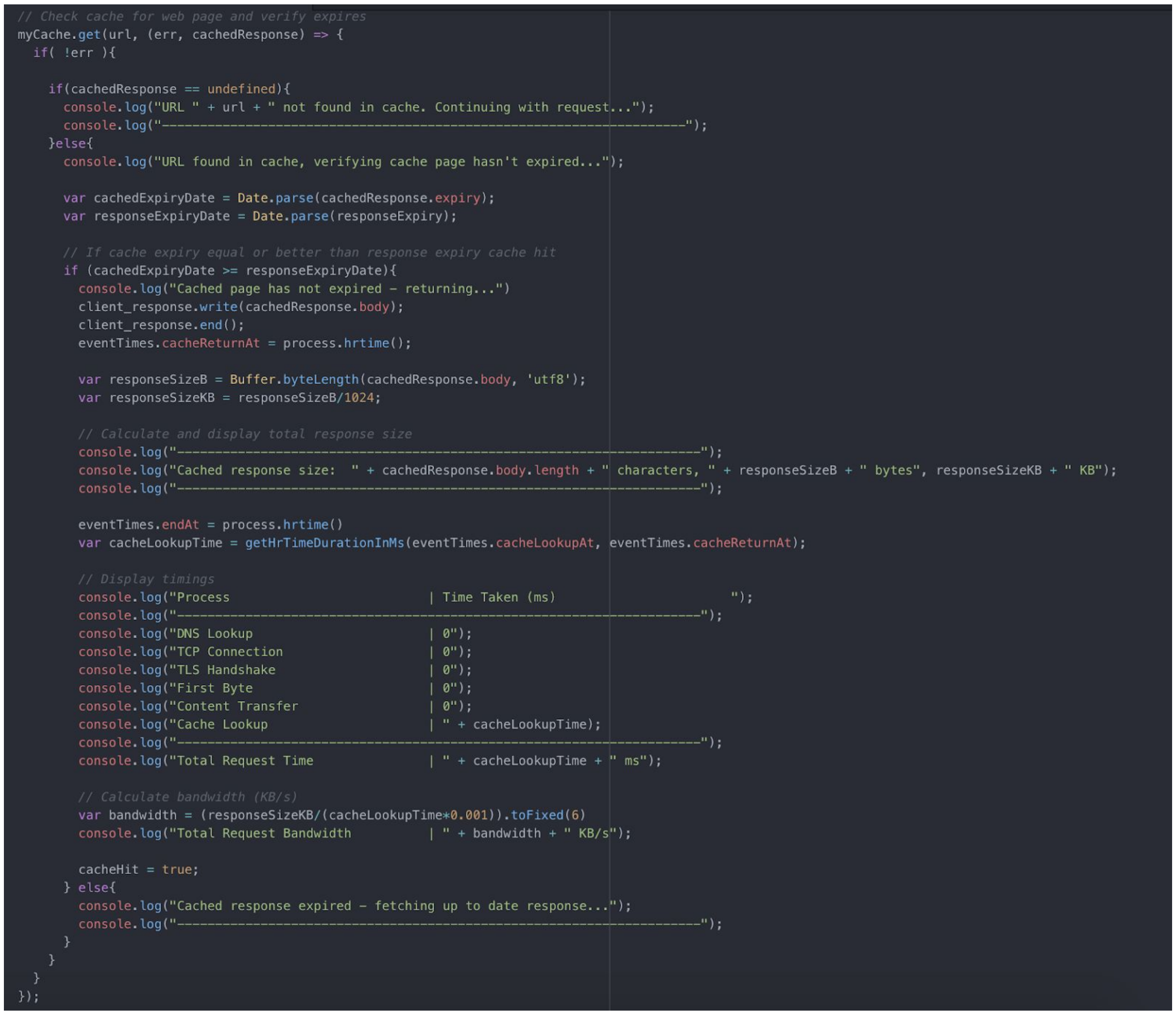
 If the status code is in fact 200 and we have received a successful response for our given proxied request the first thing that is done is that the encoding of the result is set to UTF8. We then create an empty variable to house the subsequent packets as it is more than likely that are response will be send over multiple packets. Node.JS triggers an event for every time a chunk of data is received and we then append this to the variable which is housing the data. Node.JS also notifies the server on the event of the end of transmission. Once this event has been triggered the servers response is finished and we can relay this response back to our proxies client. The proxy also calculates the bandwidth of the request based on timings and the size of the response and graphically displays this back to the user. The proxy also adds this response to the cache with the request URL being the key.



```
// When response is finished
res.on('end', () => {

  eventTimes.endAt = process.hrtime()

  var timings = getTimings(eventTimes);
  var responseSizeB = Buffer.byteLength(rawData, 'utf8');
  var responseSizeKB = responseSizeB/1024;

  // Calculate and display total response size
  console.log("---------------------------------------------------------------------");
  console.log("Total response size:  " + rawData.length + " characters, " + responseSizeB + " bytes", responseSizeKB + " KB");
  console.log("---------------------------------------------------------------------");

  // Display timings
  console.log("Process                          | Time Taken (ms)                       ");
  console.log("---------------------------------------------------------------------");
  console.log("DNS Lookup                       | " + timings.dnsLookup);
  console.log("TCP Connection                   | " + timings.tcpConnection);
  console.log("TLS Handshake                    | " + timings.tlsHandshake);
  console.log("First Byte                       | " + timings.firstByte);
  console.log("Content Transfer                 | " + timings.contentTransfer);
  console.log("---------------------------------------------------------------------");
  console.log("Total Request Time               | " + timings.total + " ms");

  // Calculate bandwidth (KB/s)
  var bandwidth = (responseSizeKB/(timings.total*0.001)).toFixed(6)
  console.log("Total Request Bandwidth          | " + bandwidth + " KB/s");

  // Create cache object with expiry
  cacheObject = {
    expiry: responseExpiry,
    body: rawData
  }

  myCache.set(url, cacheObject, (err, success) => {
    if(!err && success){
      console.log("\nSuccessfully added " + url + " to cache");
    } else{
      console.log("\nFailed to add " + url + " to cache");
    }
  })

  client_response.write(rawData);
  client_response.end();
});
```

Caching Requests

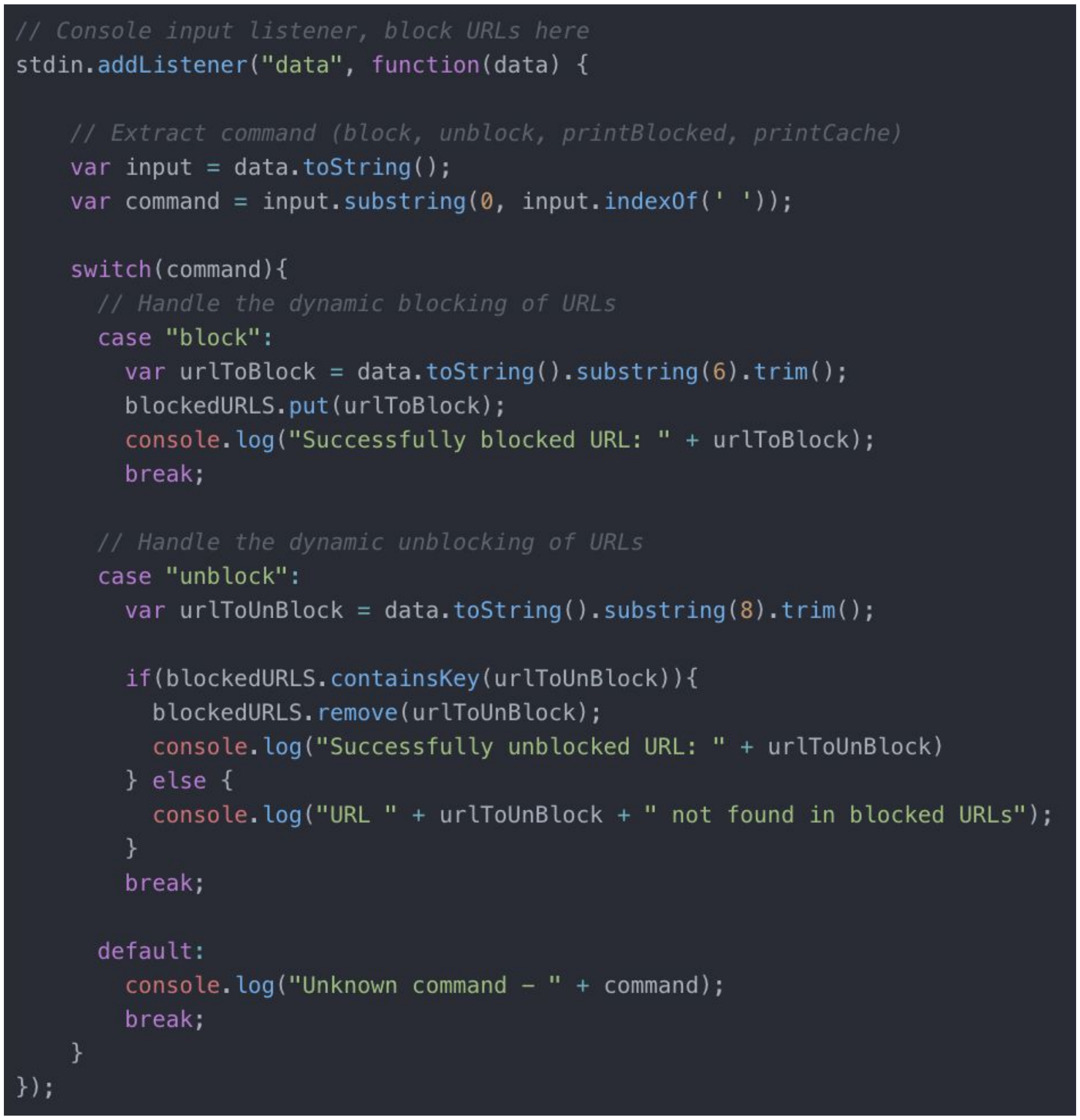
In order to preserve bandwidth and unnecessary requests being sent over the network the proxy server uses a cache to save previously completed requests. The cache was implemented using a Node.JS library node-cache. When a request is received and is not blocked the proxy checks the cache to see if there is an entry that matches the request URL. If an entry is not found the proxy continues on with the request as detailed above. However, if a cache entry is found for the URL (cache hit) then the proxy extracts the expiry timestamp from the cached object and compares it with the current timestamp. If the cached request has expired then the proxy fetches an up-to-date response from the server and then proceeds as described above. If the cached request has not expired it is returned to the web client without having to send a subsequent request to the requested address thus saving bandwidth.



```javascript
// Check cache for web page and verify expires
myCache.get(url, (err, cachedResponse) => {
  if( !err ){

    if(cachedResponse == undefined){
      console.log("URL " + url + " not found in cache. Continuing with request...");
      console.log("---------------------------------------------------------------------");
    }else{
      console.log("URL found in cache, verifying cache page hasn't expired...");

      var cachedExpiryDate = Date.parse(cachedResponse.expiry);
      var responseExpiryDate = Date.parse(responseExpiry);

      // If cache expiry equal or better than response expiry cache hit
      if (cachedExpiryDate >= responseExpiryDate){
        console.log("Cached page has not expired - returning...")
        client_response.write(cachedResponse.body);
        client_response.end();
        eventTimes.cacheReturnAt = process.hrtime();

        var responseSizeB = Buffer.byteLength(cachedResponse.body, 'utf8');
        var responseSizeKB = responseSizeB/1024;

        // Calculate and display total response size
        console.log("---------------------------------------------------------------------");
        console.log("Cached response size:  " + cachedResponse.body.length + " characters, " + responseSizeB + " bytes", responseSizeKB + " KB");
        console.log("---------------------------------------------------------------------");

        eventTimes.endAt = process.hrtime()
        var cacheLookupTime = getHrTimeDurationInMs(eventTimes.cacheLookupAt, eventTimes.cacheReturnAt);

        // Display timings
        console.log("Process                          | Time Taken (ms)                       ");
        console.log("---------------------------------------------------------------------");
        console.log("DNS Lookup                       | 0");
        console.log("TCP Connection                   | 0");
        console.log("TLS Handshake                    | 0");
        console.log("First Byte                       | 0");
        console.log("Content Transfer                 | 0");
        console.log("Cache Lookup                     | " + cacheLookupTime);
        console.log("---------------------------------------------------------------------");
        console.log("Total Request Time               | " + cacheLookupTime + " ms");

        // Calculate bandwidth (KB/s)
        var bandwidth = (responseSizeKB/(cacheLookupTime*0.001)).toFixed(6)
        console.log("Total Request Bandwidth          | " + bandwidth + " KB/s");

        cacheHit = true;
      } else{
        console.log("Cached response expired - fetching up to date response...");
        console.log("---------------------------------------------------------------------");
      }
    }
  }
});
```

Blocking/Unblocking URLs

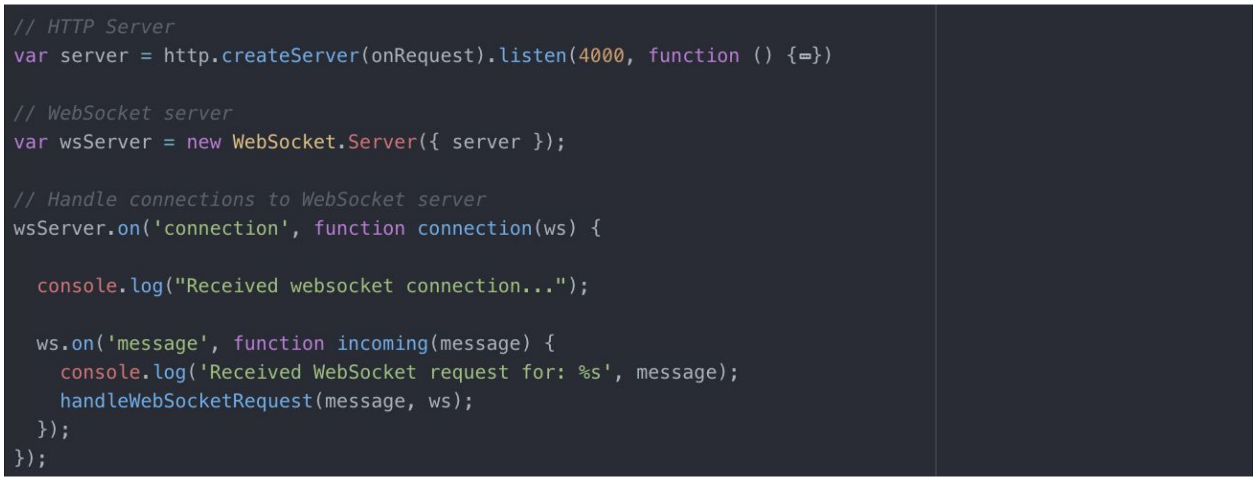
In order for the proxy user/admin to be able to dynamically block and unblock URLs a management console was implemented via the basic terminal that the proxy is being run through. This was done by adding a listener to the *stdIn* of the terminal which allowed for the proxy to detect input based on an event being triggered. Once it received data from the user it would trigger a callback function that would perform the desired functionality of blocking and unblocking URLs.



```
// Console input listener, block URLs here
stdin.addListener("data", function(data) {

    // Extract command (block, unblock, printBlocked, printCache)
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    switch(command){
      // Handle the dynamic blocking of URLs
      case "block":
        var urlToBlock = data.toString().substring(6).trim();
        blockedURLS.put(urlToBlock);
        console.log("Successfully blocked URL: " + urlToBlock);
        break;

      // Handle the dynamic unblocking of URLs
      case "unblock":
        var urlToUnBlock = data.toString().substring(8).trim();

        if(blockedURLS.containsKey(urlToUnBlock)){
          blockedURLS.remove(urlToUnBlock);
          console.log("Successfully unblocked URL: " + urlToUnBlock)
        } else {
          console.log("URL " + urlToUnBlock + " not found in blocked URLs");
        }
        break;

      default:
        console.log("Unknown command - " + command);
        break;
    }
});
```

WebSocket Connections

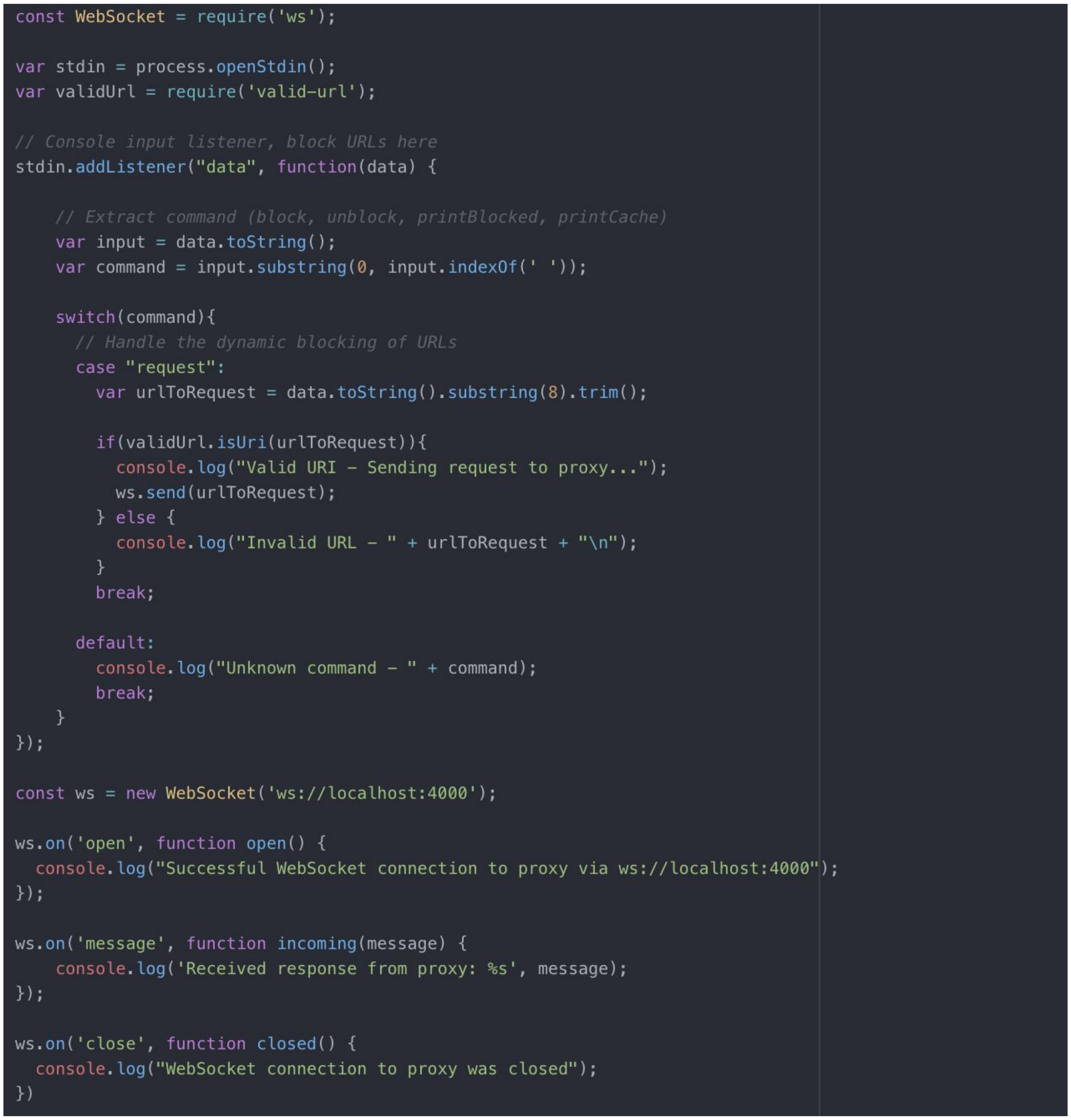
The proxy can also accept WebSocket connections and allow requests to be sent over this connection between the client and the proxy. The WebSocket server is created using the Node.JS ws library. This WebSocket connection allows for a two-way communication between the client and the server. It also allows for the WebSocket server to be bound to the previously built proxy server allowing for requests and responses to be handled in the exact same manner as before just with a different method of relaying the response to the WebSocket client and parsing the request.



```
// HTTP Server
var server = http.createServer(onRequest).listen(4000, function () {⇔})

// WebSocket server
var wsServer = new WebSocket.Server({ server });

// Handle connections to WebSocket server
wsServer.on('connection', function connection(ws) {

  console.log("Received websocket connection...");

  ws.on('message', function incoming(message) {
    console.log('Received WebSocket request for: %s', message);
    handleWebSocketRequest(message, ws);
  });
});
```

WebSocket Client

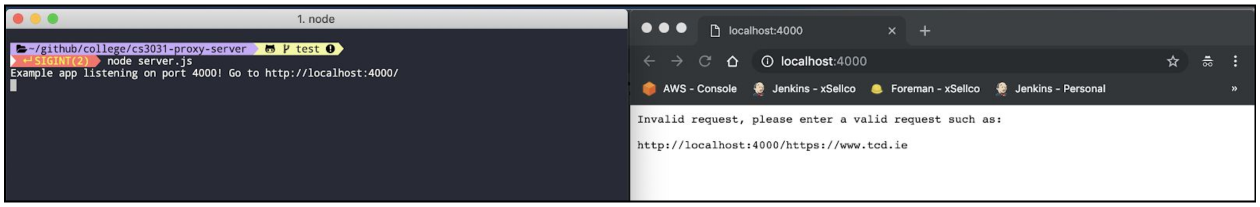
The following code details an example of how a client could be built in Node.JS to communicate with the proxy server.



```javascript
const WebSocket = require('ws');

var stdin = process.openStdin();
var validUrl = require('valid-url');

// Console input listener, block URLs here
stdin.addListener("data", function(data) {

    // Extract command (block, unblock, printBlocked, printCache)
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    switch(command){
      // Handle the dynamic blocking of URLs
      case "request":
        var urlToRequest = data.toString().substring(8).trim();

        if(validUrl.isUri(urlToRequest)){
          console.log("Valid URI - Sending request to proxy...");
          ws.send(urlToRequest);
        } else {
          console.log("Invalid URL - " + urlToRequest + "\n");
        }
        break;

      default:
        console.log("Unknown command - " + command);
        break;
    }
});

const ws = new WebSocket('ws://localhost:4000');

ws.on('open', function open() {
  console.log("Successful WebSocket connection to proxy via ws://localhost:4000");
});

ws.on('message', function incoming(message) {
    console.log('Received response from proxy: %s', message);
});

ws.on('close', function closed() {
  console.log("WebSocket connection to proxy was closed");
})
```

Screenshots

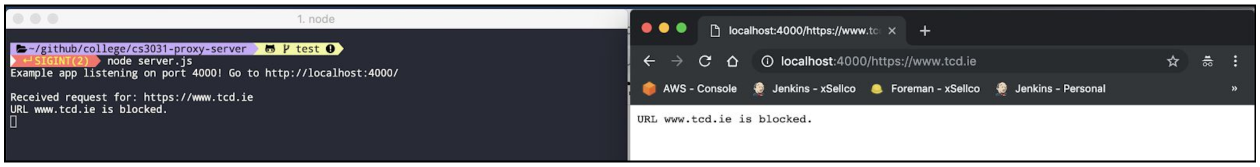
Invalid Request



```
~/github/college/cs3031-proxy-server  test
SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/
```

```
Invalid request, please enter a valid request such as:

http://localhost:4000/https://www.tcd.ie
```

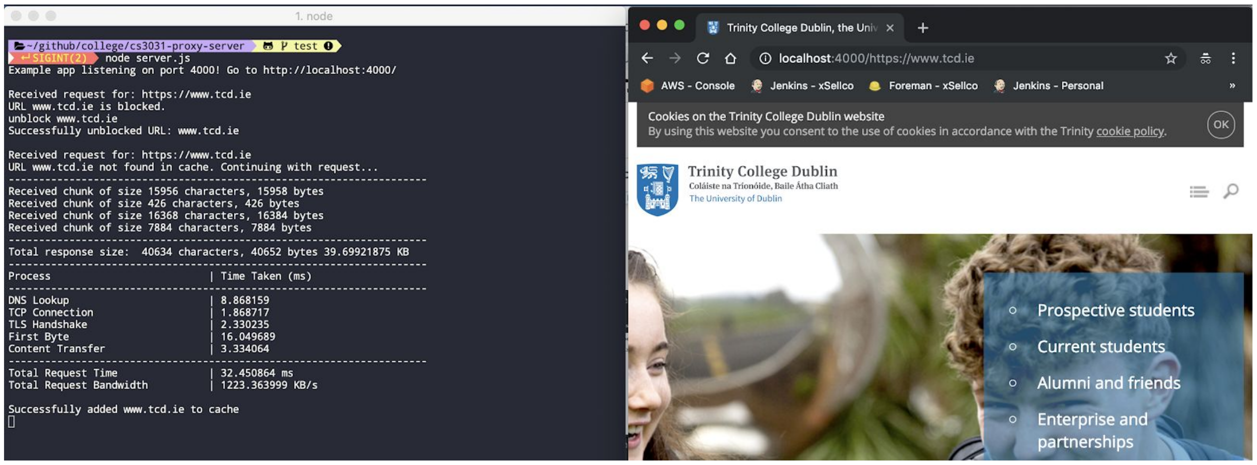
Blocked URL



```
~/github/college/cs3031-proxy-server  test
SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/

Received request for: https://www.tcd.ie
URL www.tcd.ie is blocked.
```

```
URL www.tcd.ie is blocked.
```

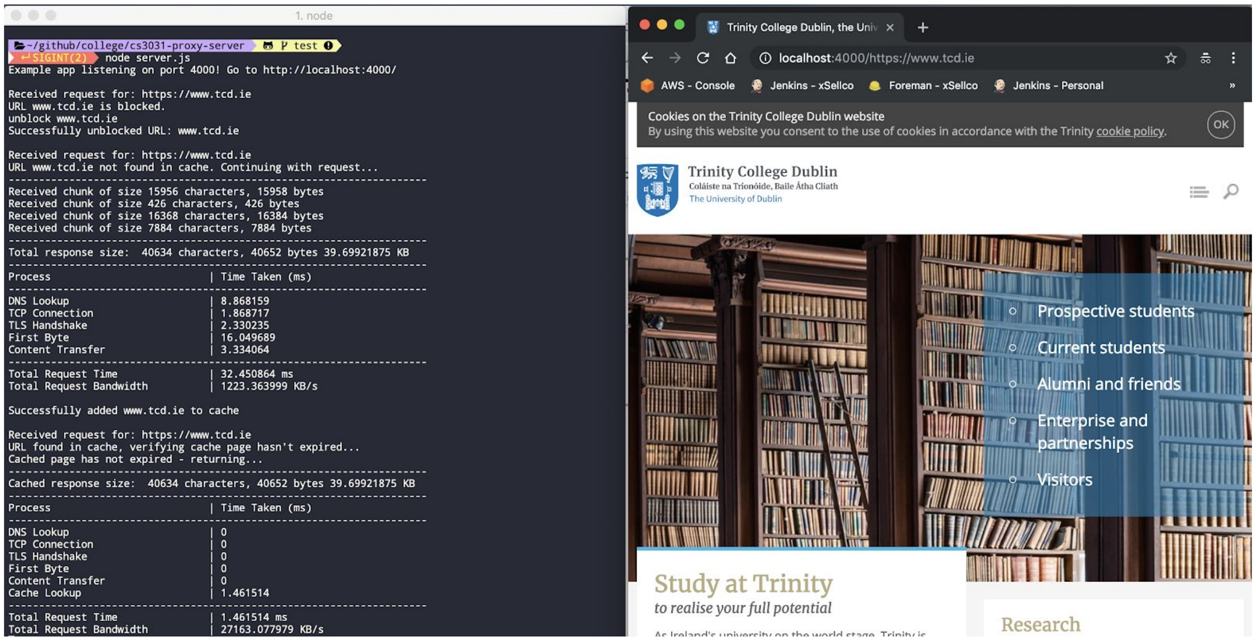
Unblocking URL



```
~/github/college/cs3031-proxy-server  test
SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/

Received request for: https://www.tcd.ie
URL www.tcd.ie is blocked.
unblock www.tcd.ie
Successfully unblocked URL: www.tcd.ie

Received request for: https://www.tcd.ie
URL www.tcd.ie not found in cache. Continuing with request...
---------------------------------------------------------------------
Received chunk of size 15956 characters, 15958 bytes
Received chunk of size 426 characters, 426 bytes
Received chunk of size 16368 characters, 16384 bytes
Received chunk of size 7884 characters, 7884 bytes
---------------------------------------------------------------------
Total response size:  40634 characters, 40652 bytes 39.69921875 KB
---------------------------------------------------------------------
Process                          | Time Taken (ms)
---------------------------------------------------------------------
DNS Lookup                       | 8.868159
TCP Connection                   | 1.868717
TLS Handshake                    | 2.330235
First Byte                       | 16.049689
Content Transfer                 | 3.334064
---------------------------------------------------------------------
Total Request Time               | 32.450864 ms
Total Request Bandwidth          | 1223.363999 KB/s

Successfully added www.tcd.ie to cache
```

Cookies on the Trinity College Dublin website
By using this website you consent to the use of cookies in accordance with the Trinity cookie policy.

Cached Response



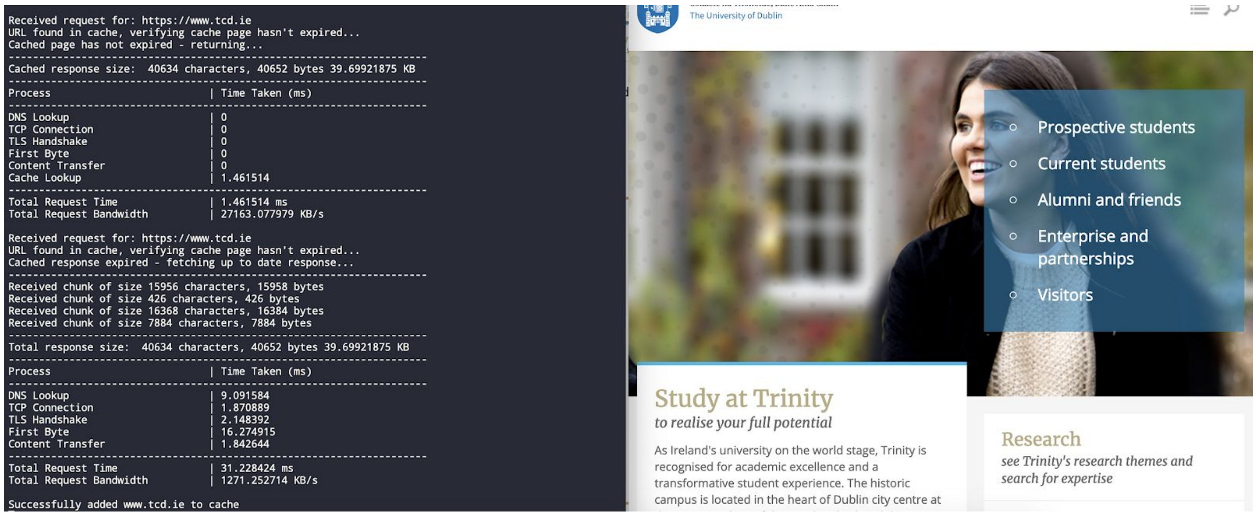
```
~/github/college/cs3031-proxy-server  test
↵ SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/

Received request for: https://www.tcd.ie
URL www.tcd.ie is blocked.
unblock www.tcd.ie
Successfully unblocked URL: www.tcd.ie

Received request for: https://www.tcd.ie
URL www.tcd.ie not found in cache. Continuing with request...
--------------------------------------------------------------------
Received chunk of size 15956 characters, 15958 bytes
Received chunk of size 426 characters, 426 bytes
Received chunk of size 16368 characters, 16384 bytes
Received chunk of size 7884 characters, 7884 bytes
--------------------------------------------------------------------
Total response size:  40634 characters, 40652 bytes 39.69921875 KB
--------------------------------------------------------------------
Process                          | Time Taken (ms)
--------------------------------------------------------------------
DNS Lookup                       | 8.868159
TCP Connection                   | 1.868717
TLS Handshake                    | 2.330235
First Byte                       | 16.049689
Content Transfer                 | 3.334064
--------------------------------------------------------------------
Total Request Time               | 32.450864 ms
Total Request Bandwidth          | 1223.363999 KB/s

Successfully added www.tcd.ie to cache

Received request for: https://www.tcd.ie
URL found in cache, verifying cache page hasn't expired...
Cached page has not expired - returning...
--------------------------------------------------------------------
Cached response size:  40634 characters, 40652 bytes 39.69921875 KB
--------------------------------------------------------------------
Process                          | Time Taken (ms)
--------------------------------------------------------------------
DNS Lookup                       | 0
TCP Connection                   | 0
TLS Handshake                    | 0
First Byte                       | 0
Content Transfer                 | 0
Cache Lookup                     | 1.461514
--------------------------------------------------------------------
Total Request Time               | 1.461514 ms
Total Request Bandwidth          | 27163.077979 KB/s
```

1. node
Trinity College Dublin, the Univ
localhost:4000/https://www.tcd.ie
AWS - Console
Jenkins - xSellco
Foreman - xSellco
Jenkins - Personal
Cookies on the Trinity College Dublin website
By using this website you consent to the use of cookies in accordance with the Trinity cookie policy.
OK
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin
Prospective students
Current students
Alumni and friends
Enterprise and partnerships
Visitors
Study at Trinity
to realise your full potential
Research

From this we can also see that the difference in bandwidth between a cached and non-cached response is significantly greater:

- **Non-Cached Request:** 1223.36 KB/s
- **Cached Request:** 27163.07 KB/s

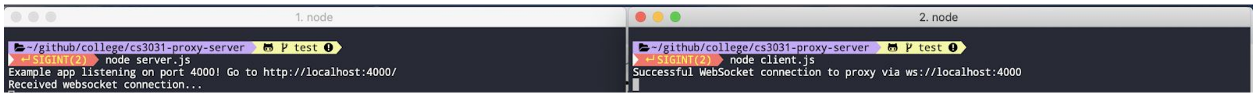
Expired Cache Request



```
Received request for: https://www.tcd.ie
URL found in cache, verifying cache page hasn't expired...
Cached page has not expired - returning...
--------------------------------------------------------------------
Cached response size:  40634 characters, 40652 bytes 39.69921875 KB
--------------------------------------------------------------------
Process                          | Time Taken (ms)
--------------------------------------------------------------------
DNS Lookup                       | 0
TCP Connection                   | 0
TLS Handshake                    | 0
First Byte                       | 0
Content Transfer                 | 0
Cache Lookup                     | 1.461514
--------------------------------------------------------------------
Total Request Time               | 1.461514 ms
Total Request Bandwidth          | 27163.077979 KB/s

Received request for: https://www.tcd.ie
URL found in cache, verifying cache page hasn't expired...
Cached response expired - fetching up to date response...
--------------------------------------------------------------------
Received chunk of size 15956 characters, 15958 bytes
Received chunk of size 426 characters, 426 bytes
Received chunk of size 16368 characters, 16384 bytes
Received chunk of size 7884 characters, 7884 bytes
--------------------------------------------------------------------
Total response size:  40634 characters, 40652 bytes 39.69921875 KB
--------------------------------------------------------------------
Process                          | Time Taken (ms)
--------------------------------------------------------------------
DNS Lookup                       | 9.091584
TCP Connection                   | 1.870889
TLS Handshake                    | 2.148392
First Byte                       | 16.274915
Content Transfer                 | 1.842644
--------------------------------------------------------------------
Total Request Time               | 31.228424 ms
Total Request Bandwidth          | 1271.252714 KB/s

Successfully added www.tcd.ie to cache
```

The University of Dublin
Prospective students
Current students
Alumni and friends
Enterprise and partnerships
Visitors
Study at Trinity
to realise your full potential
As Ireland's university on the world stage, Trinity is recognised for academic excellence and a transformative student experience. The historic campus is located in the heart of Dublin city centre at
Research
see Trinity's research themes and search for expertise

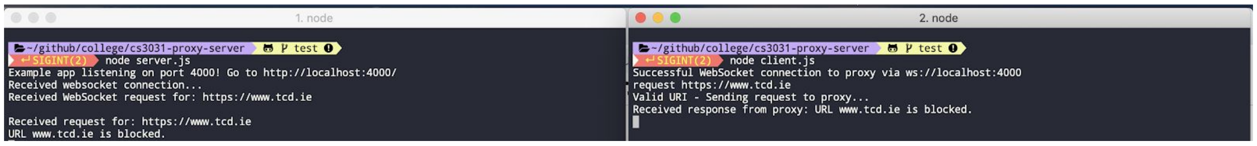
WebSocket Connection



```
1. node
~/github/college/cs3031-proxy-server  test
↵ SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/
Received websocket connection...
```

```
2. node
~/github/college/cs3031-proxy-server  test
↵ SIGINT(2)  node client.js
Successful WebSocket connection to proxy via ws://localhost:4000
```

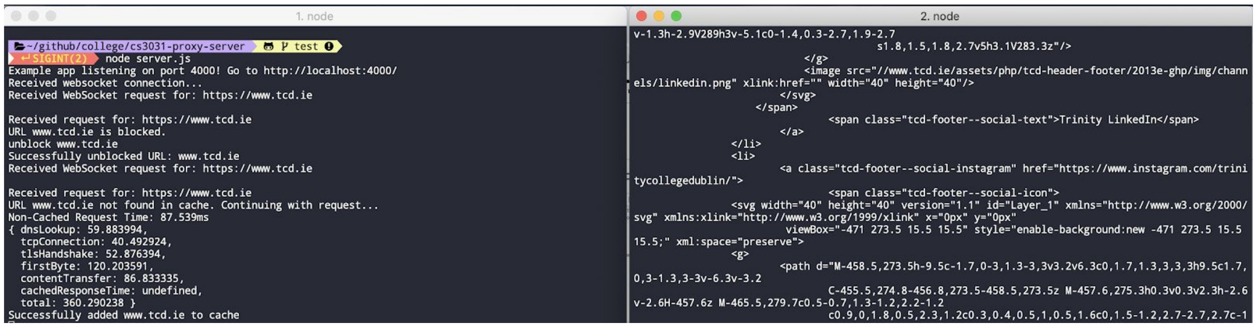
WebSocket Request (Blocked)



```
1. node
~/github/college/cs3031-proxy-server  test
↵ SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/
Received websocket connection...
Received WebSocket request for: https://www.tcd.ie

Received request for: https://www.tcd.ie
URL www.tcd.ie is blocked.
```

```
2. node
~/github/college/cs3031-proxy-server  test
↵ SIGINT(2)  node client.js
Successful WebSocket connection to proxy via ws://localhost:4000
request https://www.tcd.ie
Valid URI - Sending request to proxy...
Received response from proxy: URL www.tcd.ie is blocked.
```

WebSocket Request (Unblocked)



```
1. node
~/github/college/cs3031-proxy-server  test
↵ SIGINT(2)  node server.js
Example app listening on port 4000! Go to http://localhost:4000/
Received websocket connection...
Received WebSocket request for: https://www.tcd.ie

Received request for: https://www.tcd.ie
URL www.tcd.ie is blocked.
unblock www.tcd.ie
Successfully unblocked URL: www.tcd.ie
Received WebSocket request for: https://www.tcd.ie

Received request for: https://www.tcd.ie
URL www.tcd.ie not found in cache. Continuing with request...
Non-Cached Request Time: 87.539ms
{ dnsLookup: 59.883994,
  tcpConnection: 40.492924,
  tlsHandshake: 52.876394,
  firstByte: 120.203591,
  contentTransfer: 86.833335,
  cachedResponseTime: undefined,
  total: 360.290238 }
Successfully added www.tcd.ie to cache
```

```
2. node
v-1.3h-2.9V289h3v-5.1c0-1.4,0.3-2.7,1.9-2.7
                                        s1.8,1.5,1.8,2.7v5h3.1V283.3z"/>
                            </g>
                            <image src="//www.tcd.ie/assets/php/tcd-header-footer/2013e-ghp/img/chann
els/linkedin.png" xlink:href="" width="40" height="40"/>
                        </svg>
                    </span>
                                <span class="tcd-footer--social-text">Trinity LinkedIn</span>
                        </a>
                </li>
                <li>
                        <a class="tcd-footer--social-instagram" href="https://www.instagram.com/trini
tycollegedublin/">
                                <span class="tcd-footer--social-icon">
                <svg width="40" height="40" version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/
svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
                         viewBox="-471 273.5 15.5 15.5" style="enable-background:new -471 273.5 15.5
15.5;" xml:space="preserve">
                <g>
                        <path d="M-458.5,273.5h-9.5c-1.7,0-3,1.3-3,3v3.2v6.3c0,1.7,1.3,3,3,3h9.5c1.7,
0,3-1.3,3-3v-6.3v-3.2
                                C-455.5,274.8-456.8,273.5-458.5,273.5z M-457.6,275.3h0.3v0.3v2.3h-2.6
v-2.6H-457.6z M-465.5,279.7c0.5-0.7,1.3-1.2,2.2-1.2
                                c0.9,0,1.8,0.5,2.3,1.2c0.3,0.4,0.5,1,0.5,1.6c0,1.5-1.2,2.7-2.7,2.7c-1
```

Code Dump (Server.js)

```javascript
var fs = require('fs');
var https = require('https');
var http = require('http');
var WebSocket = require('ws');
var URL = require('url');
var NodeCache = require( "node-cache" );
var SimpleHashTable = require('simple-hashtable');
var stdin = process.openStdin();

// Create cache, never delete files unless updated by request
const myCache = new NodeCache({ stdTTL: 3600, checkperiod: 3600 });
const blockedURLS = new SimpleHashTable();

// Constants for timing
const NS_PER_SEC = 1e9
const MS_PER_NS = 1e6

// Block TCD to start with
blockedURLS.put('www.tcd.ie', 'blocked');

// Console input listener, block URLs here
stdin.addListener("data", function(data) {

    // Extract command (block, unblock, printBlocked, printCache)
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    switch(command){
      // Handle the dynamic blocking of URLs
      case "block":
        var urlToBlock = data.toString().substring(6).trim();
        blockedURLS.put(urlToBlock);
        console.log("Successfully blocked URL: " + urlToBlock);
        break;

      // Handle the dynamic unblocking of URLs
      case "unblock":
        var urlToUnBlock = data.toString().substring(8).trim();

        if(blockedURLS.containsKey(urlToUnBlock)){
          blockedURLS.remove(urlToUnBlock);
          console.log("Successfully unblocked URL: " + urlToUnBlock)
        } else {
          console.log("URL " + urlToUnBlock + " not found in blocked URLs");
        }

        break;

      default:
        console.log("Unknown command - " + command);
        break;
    }
});
```

```
// Handle responses
function handleResponse(options, res, client_response, eventTimes){

  // Extract URL from options
  var url = options.hostname;

  // Check if URL is blocked
  if(blockedURLS.containsKey(url)){
    console.log("URL " + url + " is blocked.");
    client_response.write("URL " + url + " is blocked.");
    client_response.end();
    return;
  }

  // Extract status code and expires from response
  const { statusCode } = res;
  const responseExpiry = res.headers['expires'];

  let error;

  //If no 200 status received, error
  if (statusCode !== 200) {
    error = new Error('Request Failed.\n' +
    `Status Code: ${statusCode}`);
  }

  // Handle response error
  if (error) {
    console.error(error.message);
    client_response.write(error.message);
    client_response.end();
    return;
  }

  var cacheHit = false;

  // Start cache lookup time
  eventTimes.cacheLookupAt = process.hrtime()

  // Check cache for web page and verify expires
  myCache.get(url, (err, cachedResponse) => {
    if( !err ){

      if(cachedResponse == undefined){
        console.log("URL " + url + " not found in cache. Continuing with request...");
        console.log("---------------------------------------------------------------------");
      }else{
        console.log("URL found in cache, verifying cache page hasn't expired...");

        var cachedExpiryDate = Date.parse(cachedResponse.expiry);
        var responseExpiryDate = Date.parse(responseExpiry);

        // If cache expiry equal or better than response expiry cache hit
        if (cachedExpiryDate >= responseExpiryDate){
          console.log("Cached page has not expired - returning...")
          client_response.write(cachedResponse.body);
          client_response.end();
          eventTimes.cacheReturnAt = process.hrtime();

          var responseSizeB = Buffer.byteLength(cachedResponse.body, 'utf8');
          var responseSizeKB = responseSizeB/1024;

          // Calculate and display total response size
          console.log("---------------------------------------------------------------------");
          console.log("Cached response size:  " + cachedResponse.body.length + " characters, " +
responseSizeB + " bytes", responseSizeKB + " KB");
          console.log("---------------------------------------------------------------------");

          eventTimes.endAt = process.hrtime()
          var cacheLookupTime = getHrTimeDurationInMs(eventTimes.cacheLookupAt,
eventTimes.cacheReturnAt);

          // Display timings
          console.log("Process                          | Time Taken (ms)                       ");
          console.log("---------------------------------------------------------------------");
          console.log("DNS Lookup                       | 0");
          console.log("TCP Connection                   | 0");
          console.log("TLS Handshake                    | 0");
          console.log("First Byte                       | 0");
          console.log("Content Transfer                 | 0");
          console.log("Cache Lookup                     | " + cacheLookupTime);
          console.log("---------------------------------------------------------------------");
          console.log("Total Request Time               | " + cacheLookupTime + " ms");

          // Calculate bandwidth (KB/s)
          var bandwidth = (responseSizeKB/(cacheLookupTime*0.001)).toFixed(6)
          console.log("Total Request Bandwidth          | " + bandwidth + " KB/s");

          cacheHit = true;
        } else{
          console.log("Cached response expired - fetching up to date response...");
          console.log("---------------------------------------------------------------------");
        }
      }
    }
  });

  // If url not found in cache, continue with request and cache new response
  if(!cacheHit){
    res.setEncoding('utf8');

    let rawData = '';

    // When first byte recieved
    res.once('readable', () => {
      eventTimes.firstByteAt = process.hrtime()
      console.time('firstByteAt')
    })

    // When data is received
    res.on('data', (chunk) => {
      rawData += chunk;
      console.log("Received chunk of size " + chunk.length + " characters, " + Buffer.byteLength(chunk,
'utf8') + " bytes");
    });

    // When response is finished
    res.on('end', () => {

      eventTimes.endAt = process.hrtime()

      var responseSizeB = Buffer.byteLength(rawData, 'utf8');
      var responseSizeKB = responseSizeB/1024;

      // Calculate and display total response size
      console.log("---------------------------------------------------------------------");
      console.log("Total response size:  " + rawData.length + " characters, " + responseSizeB + "
bytes", responseSizeKB + " KB");
      console.log("---------------------------------------------------------------------");

      var timings = getTimings(eventTimes);

      // Display timings
      console.log("Process                          | Time Taken (ms)                       ");
      console.log("---------------------------------------------------------------------");
      console.log("DNS Lookup                       | " + timings.dnsLookup);
      console.log("TCP Connection                   | " + timings.tcpConnection);
      console.log("TLS Handshake                    | " + timings.tlsHandshake);
      console.log("First Byte                       | " + timings.firstByte);
      console.log("Content Transfer                 | " + timings.contentTransfer);
      console.log("---------------------------------------------------------------------");
      console.log("Total Request Time               | " + timings.total + " ms");

      // Calculate bandwidth (KB/s)
      var bandwidth = (responseSizeKB/(timings.total*0.001)).toFixed(6)
      console.log("Total Request Bandwidth          | " + bandwidth + " KB/s");


      // Create cache object with expiry
      cacheObject = {
        expiry: responseExpiry,
        body: rawData
      }

      myCache.set(url, cacheObject, (err, success) => {
        if(!err && success){
          console.log("\nSuccessfully added " + url + " to cache");
        } else{
          console.log("\nFailed to add " + url + " to cache");
        }
      })

      client_response.write(rawData);
      client_response.end();
    });
  }
}
```

```
// Handle requests
function onRequest(client_request, client_response) {

    // Record specific event times here
    const eventTimes = {
      // use process.hrtime() as it's not a subject of clock drift
      startAt: process.hrtime(),
      dnsLookupAt: undefined,
      tcpConnectionAt: undefined,
      tlsHandshakeAt: undefined,
      firstByteAt: undefined,
      endAt: undefined,
      cacheLookupAt: undefined,
      cacheReturnAt: undefined
    }

    var options = URL.parse(client_request.url.substring(1), true);

    // Only handle HTTP and HTTPS requests
    if(options.protocol == 'http:' || options.protocol == 'https:'){

      // Filter out favicon and assets requests
      if(options.path != 'favicon.ico' && options.hostname != 'assets'){
        console.log('\nReceived request for: ' + options.protocol + '//'+ options.hostname);

        var proxy_req = null;

        // Handle http and https request seperately
        switch(options.protocol){
          case 'http:':
            proxy_req = http.get(options.href, (res) => handleResponse(options, res, client_response,
eventTimes));
            break;
          case 'https:':
            proxy_req = https.get(options.href, (res) => handleResponse(options, res, client_response,
eventTimes))
            .on('socket', (socket) => {
              // Record DNS Lookup
              socket.on('lookup', () => {
                eventTimes.dnsLookupAt = process.hrtime();
              })
              // Record TCP connection
              socket.on('connect', () => {
                eventTimes.tcpConnectionAt = process.hrtime();
              })
              // If HTTPS record TLS handshake timing
              socket.on('secureConnect', () => {
                eventTimes.tlsHandshakeAt = process.hrtime();
              })
              socket.on('end', () => {
                eventTimes.requestEndAt = process.hrtime();
              })
            });
            break;
          default:
            client_response.write('Invalid request, please enter a valid request such
as:\n\nhttp://localhost:4000/https://www.tcd.ie');
            client_response.end();
            break;

          // Handle proxy request events
          // Once a socket is assigned to the proxy request
          proxy_req.on('socket', (socket) => {
            // Record DNS Lookup
            socket.on('lookup', () => {
              eventTimes.dnsLookupAt = process.hrtime();
            })
            // Record TCP connection
            socket.on('connect', () => {
              eventTimes.tcpConnectionAt = process.hrtime();
            })
            // If HTTPS record TLS handshake timing
            socket.on('secureConnect', () => {
              eventTimes.tlsHandshakeAt = process.hrtime();
            })
            socket.on('end', () => {
              eventTimes.requestEndAt = process.hrtime();
            })
          });

          // Handle request timeouts
          proxy_req.on('timeout', () => {
            console.log('Proxy request timed out...');
            client_response.write('Proxy request timed out...');
            client_response.end();
            proxy_req.abort();
          })

          // Handle request errors
          proxy_req.on('error', (e) => {
            console.error(`Got error: ${e.message}`);
            client_response.write(`Got error: ${e.message}`);
            client_response.end();
            proxy_req.abort();
          });
        }
      } else{
        client_response.end();
      }
    } else{
      client_response.write('Invalid request, please enter a valid request such
as:\n\nhttp://localhost:4000/https://www.tcd.ie');
      client_response.end();
    }
}
```

```
// Handle WebSocket requests
function handleWebSocketRequest(url, ws){

  // Record specific event times here
  const eventTimes = {
    // use process.hrtime() as it's not a subject of clock drift
    startAt: process.hrtime(),
    dnsLookupAt: undefined,
    tcpConnectionAt: undefined,
    tlsHandshakeAt: undefined,
    firstByteAt: undefined,
    endAt: undefined,
    cacheLookupAt: undefined,
    cacheReturnAt: undefined
  }

  var options = URL.parse(url, true);

  // Only handle HTTP and HTTPS requests
  if(options.protocol == 'http:' || options.protocol == 'https:'){

    // Filter out favicon and assets requests
    if(options.path != 'favicon.ico' && options.hostname != 'assets'){
      console.log('\nReceived request for: ' + options.protocol + '//'+ options.hostname);

      var proxy_req = null;

      // Handle http and https request seperately
      switch(options.protocol){
        case 'http:':
          proxy_req = http.get(options.href, (res) => handleWebSocketResponse(options, res, ws,
eventTimes));
          break;
        case 'https:':
          proxy_req = https.get(options.href, (res) => handleWebSocketResponse(options, res, ws,
eventTimes))
          .on('socket', (socket) => {
            // Record DNS Lookup
            socket.on('lookup', () => {
              eventTimes.dnsLookupAt = process.hrtime();
            })
            // Record TCP connection
            socket.on('connect', () => {
              eventTimes.tcpConnectionAt = process.hrtime();
            })
            // If HTTPS record TLS handshake timing
            socket.on('secureConnect', () => {
              eventTimes.tlsHandshakeAt = process.hrtime();
            })
            socket.on('end', () => {
              eventTimes.requestEndAt = process.hrtime();
            })
          });
          break;
        default:
          ws.send('Invalid request, please enter a valid request such
as:\n\nhttp://localhost:4000/https://www.tcd.ie');
          break;

        // Handle proxy request events
        // Once a socket is assigned to the proxy request
        proxy_req.on('socket', (socket) => {
          // Record DNS Lookup
          socket.on('lookup', () => {
            eventTimes.dnsLookupAt = process.hrtime();
          })
          // Record TCP connection
          socket.on('connect', () => {
            eventTimes.tcpConnectionAt = process.hrtime();
          })
          // If HTTPS record TLS handshake timing
          socket.on('secureConnect', () => {
            eventTimes.tlsHandshakeAt = process.hrtime();
          })
          socket.on('end', () => {
            eventTimes.requestEndAt = process.hrtime();
          })
        });

        // Handle request timeouts
        proxy_req.on('timeout', () => {
          console.log('Proxy request timed out...');
          ws.send('Proxy request timed out...');
          proxy_req.abort();
        })

        // Handle request errors
        proxy_req.on('error', (e) => {
          console.error(`Got error: ${e.message}`);
          ws.send(`Got error: ${e.message}`);
          proxy_req.abort();
        });
      }
    }
  } else{
    ws.send('Invalid request, please enter a valid request such
as:\n\nhttp://localhost:4000/https://www.tcd.ie');
  }
}
```

```
// Handle WebSocket responses
function handleWebSocketResponse(options, res, ws, eventTimes){

  // Extract URL from options
  var url = options.hostname;

  // Check if URL is blocked
  if(blockedURLS.containsKey(url)){
    console.log("URL " + url + " is blocked.");
    ws.send("URL " + url + " is blocked.");
    return;
  }

  // Extract status code and expires from response
  const { statusCode } = res;
  const responseExpiry = res.headers['expires'];

  let error;

  //If no 200 status received, error
  if (statusCode !== 200) {
    error = new Error('Request Failed.\n' +
    `Status Code: ${statusCode}`);
  }

  // Handle response error
  if (error) {
    console.error(error.message);
    ws.send(error.message);
    return;
  }

  var cacheHit = false;

  // Check cache for web page and verify expires
  myCache.get(url, (err, cachedResponse) => {
    if( !err ){

      if(cachedResponse == undefined){
        console.log("URL " + url + " not found in cache. Continuing with request...");
      }else{
        console.log("URL found in cache, verifying cache page hasn't expired...");

        console.log("Cache object expiry: " + cachedResponse.expiry);
        console.log("Response expiry: " + responseExpiry);

        var cachedExpiryDate = Date.parse(cachedResponse.expiry);
        var responseExpiryDate = Date.parse(responseExpiry);

        // If cache expiry equal or better than response expiry cache hit
        if (cachedExpiryDate >= responseExpiryDate){
          console.time('Cached Request Time');
          console.log("Cached page has not expired - returning...")
          ws.send(cachedResponse.body);
          console.timeEnd('Cached Request Time');
          cacheHit = true;
        } else{
          console.log("Cached response expired - fetching up to date response...");
        }
      }
    }
  });

  // If url not found in cache, continue with request and cache new response
  if(!cacheHit){
    res.setEncoding('utf8');

    let rawData = '';
    console.time('Non-Cached Request Time');

    // When first byte recieved
    res.once('readable', () => {
      eventTimes.firstByteAt = process.hrtime()
      console.time('firstByteAt')
    })

    // When data is received
    res.on('data', (chunk) => { rawData += chunk; });

    // When response is finished
    res.on('end', () => {

      console.timeEnd('Non-Cached Request Time');
      eventTimes.endAt = process.hrtime()
      var timings = getTimings(eventTimes);
      console.log(timings);

      // Create cache object with expiry
      cacheObject = {
        expiry: responseExpiry,
        body: rawData
      }

      myCache.set(url, cacheObject, (err, success) => {
        if(!err && success){
          console.log("Successfully added " + url + " to cache");
        } else{
          console.log("Failed to add " + url + " to cache");
        }
      })

      ws.send(rawData);
    });
  }
}
```

```
// Calculates all of the stored timing values
function getTimings (eventTimes) {
  return {
    // There is no DNS lookup with IP address
    dnsLookup: eventTimes.dnsLookupAt !== undefined ? getHrTimeDurationInMs(eventTimes.startAt,
eventTimes.dnsLookupAt) : undefined,
    tcpConnection: getHrTimeDurationInMs(eventTimes.dnsLookupAt || eventTimes.startAt,
eventTimes.tcpConnectionAt),
    tlsHandshake: eventTimes.tlsHandshakeAt !== undefined ?
(getHrTimeDurationInMs(eventTimes.tcpConnectionAt, eventTimes.tlsHandshakeAt)) : undefined,
    firstByte: getHrTimeDurationInMs((eventTimes.tlsHandshakeAt || eventTimes.tcpConnectionAt),
eventTimes.firstByteAt),
    contentTransfer: getHrTimeDurationInMs(eventTimes.firstByteAt, eventTimes.endAt),
    cachedResponseTime: (eventTimes.cacheLookupAt != undefined && eventTimes.cacheReturnAt != undefined)
? getHrTimeDurationInMs(eventTimes.cacheLookupAt, eventTimes.cacheReturnAt) : undefined,
    total: getHrTimeDurationInMs(eventTimes.startAt, eventTimes.endAt)
  }
}

// Converts times to ms
function getHrTimeDurationInMs (startTime, endTime) {
  const secondDiff = endTime[0] - startTime[0]
  const nanoSecondDiff = endTime[1] - startTime[1]
  const diffInNanoSecond = secondDiff * NS_PER_SEC + nanoSecondDiff

  return diffInNanoSecond / MS_PER_NS
}

// HTTP Server
var server = http.createServer(onRequest).listen(4000, function () {
  console.log('Example app listening on port 4000! Go to http://localhost:4000/')
})

// WebSocket server
var wsServer = new WebSocket.Server({ server });

// Handle connections to WebSocket server
wsServer.on('connection', function connection(ws) {

  console.log("Received websocket connection...");

  ws.on('message', function incoming(message) {
    console.log('Received WebSocket request for: %s', message);
    handleWebSocketRequest(message, ws);
  });
});
```

Code Dump (Client.js)

```
const WebSocket = require('ws');

var stdin = process.openStdin();
var validUrl = require('valid-url');

// Console input listener, block URLs here
stdin.addListener("data", function(data) {

    // Extract command (block, unblock, printBlocked, printCache)
    var input = data.toString();
    var command = input.substring(0, input.indexOf(' '));

    switch(command){
      // Handle the dynamic blocking of URLs
      case "request":
        var urlToRequest = data.toString().substring(8).trim();

        if(validUrl.isUri(urlToRequest)){
          console.log("Valid URI - Sending request to proxy...");
          ws.send(urlToRequest);
        } else {
          console.log("Invalid URL - " + urlToRequest + "\n");
        }
        break;

      default:
        console.log("Unknown command - " + command);
        break;
    }
});

const ws = new WebSocket('ws://localhost:4000');

ws.on('open', function open() {
  console.log("Successful WebSocket connection to proxy via ws://localhost:4000");
});

ws.on('message', function incoming(message) {
    console.log('Received response from proxy: %s', message);
});

ws.on('close', function closed() {
  console.log("WebSocket connection to proxy was closed");
})
```

All of the above code can be found in the following GitHub Repository: cs3031-proxy-server

- Brandon Dooley (#16327446)