

CSU34031 Project 1

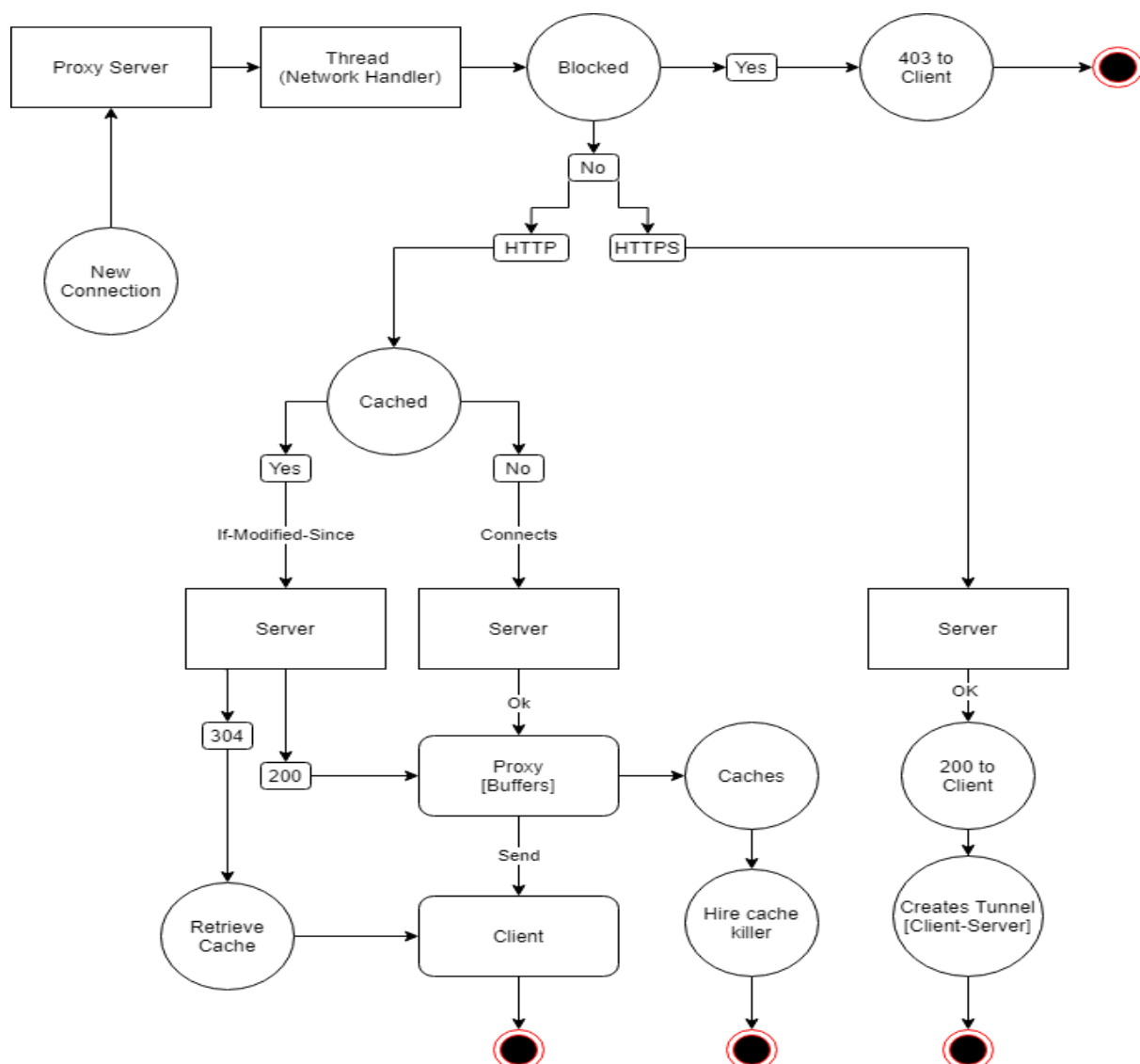
Web Proxy Server

Objective

The objective of the exercise is to implement a Web Proxy Server. A Web proxy is a local server, which fetches items from the Web on behalf of a Web client instead of the client fetching them directly. This allows for caching of pages and access control.

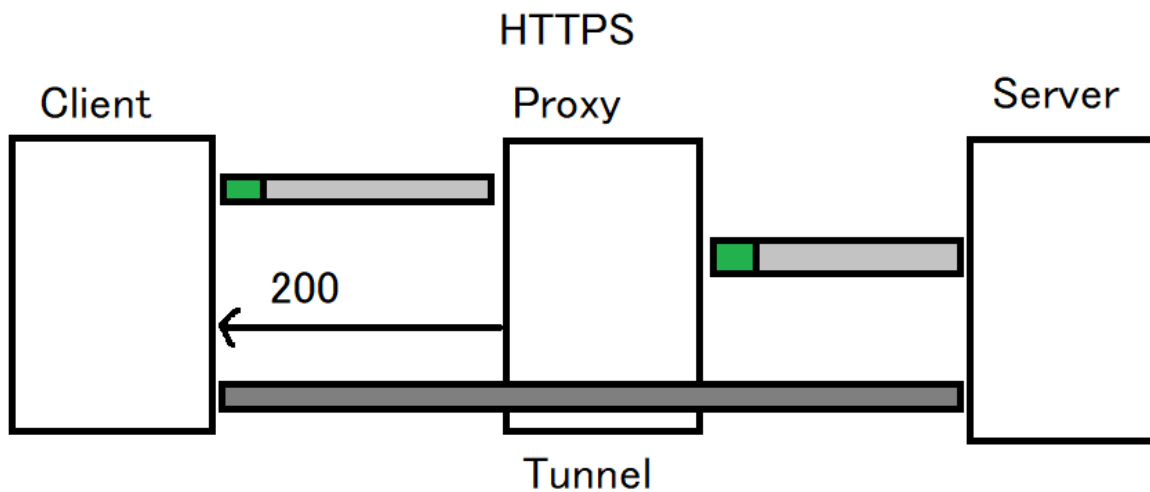
Overview

Action Flow



HTTPS

The web proxy creates a connection (tunnel) between the Client and the Server without doing any additional processing on the proxy server.

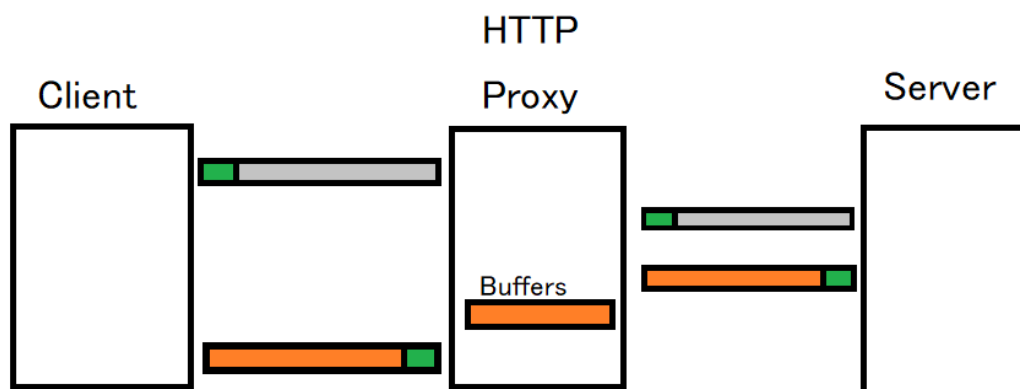


```
// 200 to Client
buffrw.WriteString("HTTP/1.1 200 Connection Established\r\n\r\n")
buffrw.Flush()
// Connection to client (tunnelling)
go io.Copy(serverCon, buffrw)
io.Copy(buffrw, serverCon)
```

2021/03/14 14:07:12 **HTTPS** Connection Established: **example.com:443** [99.204ms]

HTTP

The web proxy makes an HTTP call on behalf of the Client, buffers the response from the Server and then send it back to the Client.



2021/03/14 14:03:23 **HTTP** Completed Transmission: **http://example.com/** [218.8578ms]

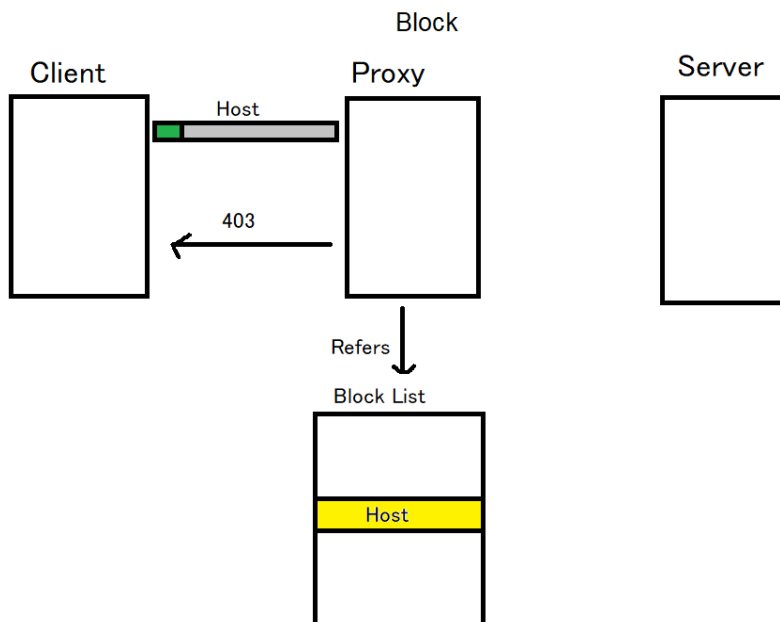
18315028
Yi Xiang Tan

Blocking

The web proxy maintains a block list.

If The host of a request from the Client, the request is blocked and a 403 is returned to indicate that the request is forbidden.

```
if !blockList[host] {  
    // ...  
} else {  
    w.WriteHeader(http.StatusForbidden)  
}
```



One can block a host by using the following command

```
block example.com
```

and the following to unblock

```
unblock example.com
```

and the following to list the entries in the block list



```
list
```

Here's the result of blocking, visiting, listing, and unblocking.

```
block example.com  
2021/03/14 14:01:27 BLOCKED example.com
```

(The red on 'B' is explicitly defined for clarity)



18315028
Yi Xiang Tan

Status	Method	Domain	File
403	GET	 example.com	/
403	GET	 example.com	favicon.ico

```
2021/03/14 14:02:03 BLOCKED example.com
```

```
list
2021/03/14 14:02:49 map[example.com:true]
```

```
unblock example.com
2021/03/14 14:03:10 UNBLOCKED example.com
```

200	GET	 example.com	favicon.ico
200	GET	 example.com	/

Cache

Implementation

The web proxy attempts to cache HTTP responses to save time and bandwidth.

This is done by caching the responses on the proxy server based on the request URI and time (in case of cache update)

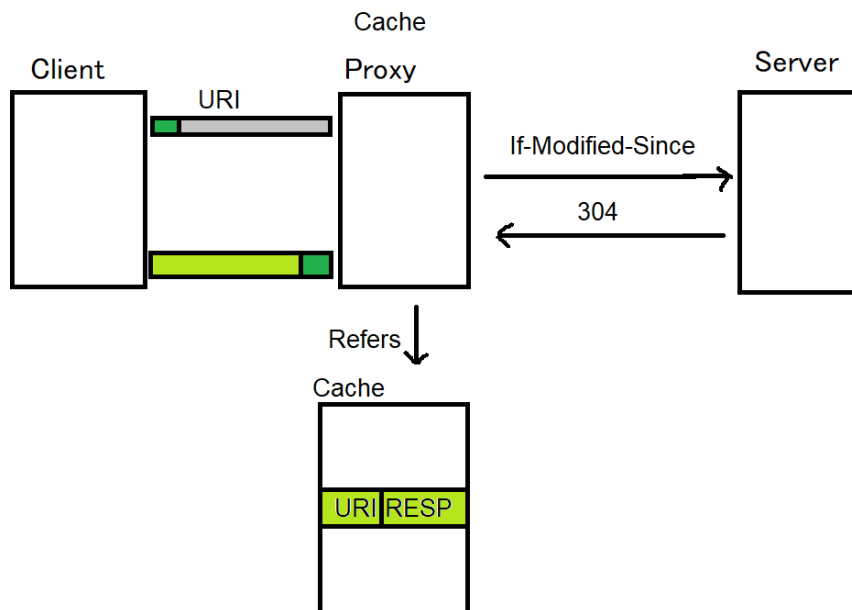
```
type cacheItem struct {
    body    []byte
    headers http.Header
    date    string
}
```

and hiring a killer to kill the cache after CACHE_EXPIRY duration if the timestamp of the cache entry is not updated.

```
time.Sleep(CACHE_EXPIRY)
    if cache[URI].date == date {
        delete(cache, URI)
    }
```

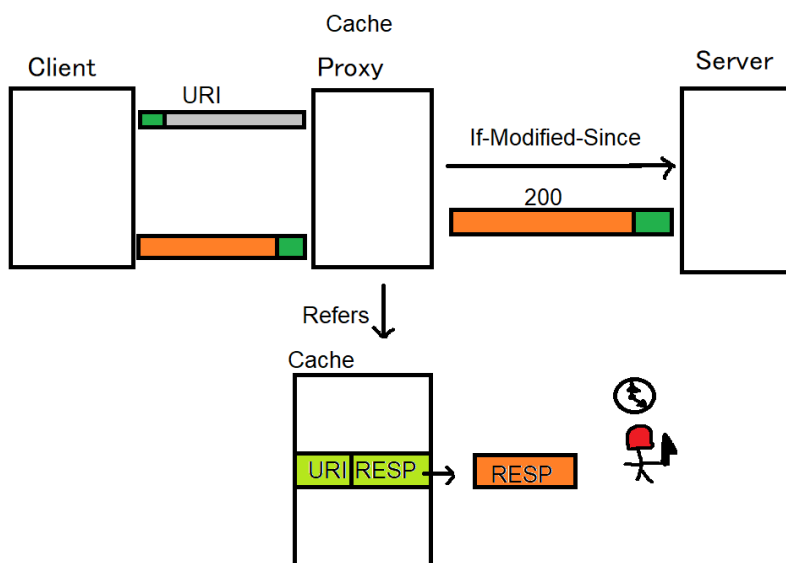
When an HTTP request gets passed to the proxy server, the proxy server checks if the response for the request is in the cache.

If it is, the proxy server sends a request to the Server with "If-Modified-Since" header, and if the Server returns a 304, the proxy server sends the cached response.



Otherwise, the proxy server sends the new response from the Server to the Client and updates the cache with the new response and hire a new cache killer.

Uncached requests will be cached in the same way, with proxy sending a request to the Server without the “If-Modified-Since” header.

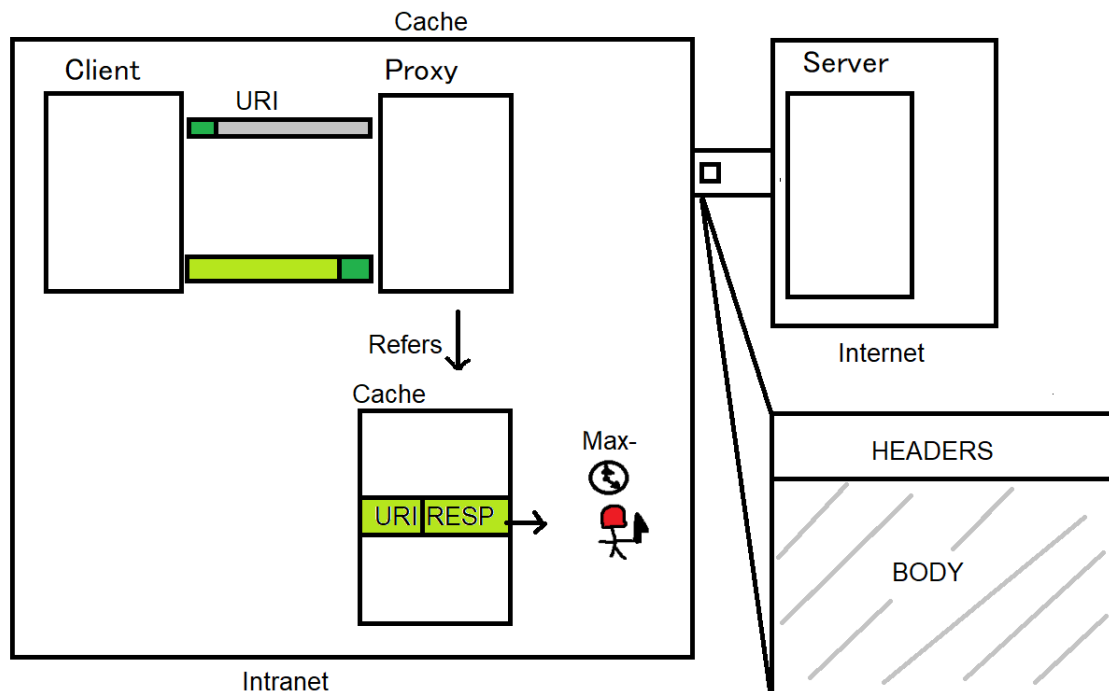


2021/03/14 14:32:29 Killing cache for <http://www.universitytimes.ie/wp-includes/js/twemoji.js> registered at Sun, 14 Mar 2021 14:30:59 GMT

Benefits

This would help in saving time and bandwidth as the request with the “If-Modified-Since” header and the response is relatively smaller than a normal request with payload, and the transfer speed within an intranet is usually faster than the internet, and the bandwidth cost is close to none.

Imagine an HTTP request to a file that is 1TB long (food for thoughts)



The benefit is stored in

```
type cacheSaving struct {  
    dataSaved      int  
    timeSaved      time.Duration  
    lastUncachedTime time.Duration  
}
```

mapped to request URI.

The data saved is calculated by the number of bytes of the response body.
The time saved is calculated by the difference between the latest uncached HTTP time and cached HTTP time, which is updated every time the cache is updated.

```
2021/03/14 14:43:09 HTTP Completed Transmission: http://universitytimes.ie/ [365.9548ms] 64035 bytes CACHED [593.8156ms]
```

We saved *64035 bytes* and *593.8156ms* by using the cache.

We could also list all the savings (all URIs) from caching with

```
saved
```

Example

```
saved  
2021/03/14 14:52:24 total data saved 1638238 bytes, total time saved 6.1139252s
```

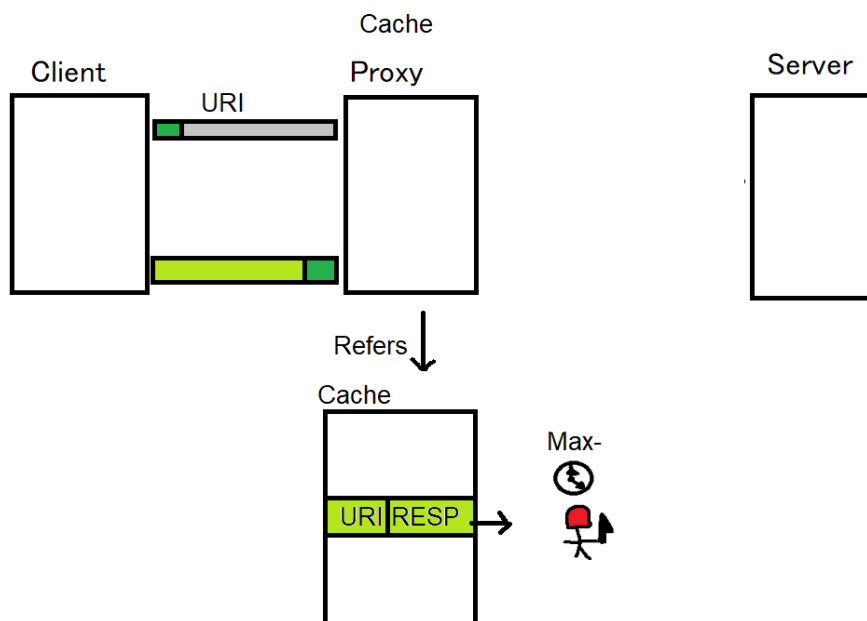
Actual Cache-Control

The actual way of [cache control](#), however, is to use the header

18315028

Yi Xiang Tan

Cache-Control: max-age=<seconds>



where we return the cached response if it was not stale(killed), without sending a request with “If-Modified-Since” header, and only send the request with the header if it was stale(killed).

This was not implemented as not all request contains the “Cache-Control” headers.

Threading

The proxy server spawns a new thread for every request.

This is supported out of the box with Golang

```
networkHandler := http.HandlerFunc(networkHandler)
// Create a thread of networkHandler for each connection
http.ListenAndServe(":8080", networkHandler)
```

where for every connection to *port 8080*, a *networkHandler* is spawned (i.e. a *networkHandler* thread for every connection).

Management Terminal

The management terminal is integrated with the terminal.

18315028

Yi Xiang Tan

```
Proxy Console [:8080]
2021/03/14 14:59:02 HTTP Completed Transmission: http://detectportal.firefox.com/success.txt [116.8636ms]
2021/03/14 14:59:02 HTTP Completed Transmission: http://detectportal.firefox.com/success.txt?ipv4 [19.2229ms]
2021/03/14 14:59:02 HTTP Completed Transmission: http://detectportal.firefox.com/success.txt?ipv6 [21.2563ms]
2021/03/14 14:59:02 HTTPS Connection Established: firefox.settings.services.mozilla.com:443 [120.0343ms]
2021/03/14 14:59:02 HTTPS Connection Established: snippets.cdn.mozilla.net:443 [70.5164ms]
2021/03/14 14:59:03 HTTPS Connection Established: push.services.mozilla.com:443 [341.9928ms]
2021/03/14 14:59:03 HTTPS Connection Established: profile.accounts.firefox.com:443 [223.2434ms]
2021/03/14 14:59:03 HTTPS Connection Established: api.accounts.firefox.com:443 [216.4594ms]
2021/03/14 14:59:03 HTTPS Connection Established: incoming.telemetry.mozilla.org:443 [246.4459ms]
2021/03/14 14:59:03 HTTP Completed Transmission: http://ocsp.digicert.com/ [87.273ms]
2021/03/14 14:59:03 HTTP Completed Transmission: http://ocsp.digicert.com/ [37.5469ms] 5 bytes CACHED [49.7261ms]
```

Commands (case sensitive)

```
block <host>
```

Adds *host* to the block list

```
unblock <host>
```

Remove *host* from the block list

```
list
```

List all entries in the block list

```
saved
```

Show the total bandwidth and time saved with caching

18315028
Yi Xiang Tan

Code

```
package main

import (
    "bufio"
    "fmt"
    "io"
    "log"
    "net"
    "net/http"
    "os"
    "strconv"
    "strings"
    "time"
)

var blockList map[string]bool = make(map[string]bool)
var cacheSavings map[string]*cacheSaving = make(map[string]*cacheSaving)
var cache map[string]cacheItem = make(map[string]cacheItem)
var CACHE_EXPIRY = 90 * time.Second

// Struct to store cache with identifier
type cacheItem struct {
    body    []byte
    headers http.Header
    date    string
}

// Struct to store information about cache savings
type cacheSaving struct {
    dataSaved      int
    timeSaved      time.Duration
    lastUncachedTime time.Duration
}

func colorOutput(str string, color string) string {
    colorCode := ""
    switch color {
    case "green":
        colorCode = "32"
    case "gray":
    }
```

18315028

Yi Xiang Tan

```
        colorCode = "37"
    case "red":
        colorCode = "91"
    case "yellow":
        colorCode = "93"
    case "cyan":
        colorCode = "96"
    }
    colorCode = "\033[" + colorCode + "m"
    return colorCode + str + "\033[0m"
}

func httpsHandler(w http.ResponseWriter, req *http.Request) {
    startTimer := time.Now()
    req.URL.Scheme = "https"

    hj, ok := w.(http.Hijacker)
    if !ok {
        http.Error(w, "webserver doesn't support hijacking",
http.StatusInternalServerError)
        return
    }
    clientCon, buffrw, err := hj.Hijack()
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    server := req.URL.Host
    serverCon, err := net.Dial("tcp", server)

    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // Doesn't work because connect is hijacked
    // w.Write([]byte("HTTP/1.1 200 Connection Established\r\n\r\n"))
    buffrw.WriteString("HTTP/1.1 200 Connection Established\r\n\r\n")
    buffrw.Flush()

    // Logging
    elapsed := time.Since(startTimer)
```

18315028

Yi Xiang Tan

```
    log.Printf("%s Connection Established: %s [%s]",
        colorOutput("HTTPS", "green"), colorOutput(server, "yellow"),
        colorOutput(elapsed.String(), "cyan"))
    // Logging

    // Connection to client
    go io.Copy(serverCon, buffrw)
    io.Copy(buffrw, serverCon)

    serverCon.Close()
    clientCon.Close()
}

func httpHandler(w http.ResponseWriter, req *http.Request) {
    startTimer := time.Now()
    URI := req.RequestURI
    cachedRes, exist := cache[URI]
    // If request/response is cached
    if exist {
        // Add header to check if cache is fresh
        location, _ := time.LoadLocation("GMT")
        formattedTime := time.Now().In(location).Format(http.TimeFormat)

        client := &http.Client{}
        req, err := http.NewRequest("GET", URI, nil)
        if err != nil {
            log.Printf("%s\n", err)
        }
        req.Header.Add("If-Modified-Since", formattedTime)
        resp, err := client.Do(req)
        if err != nil {
            log.Printf("%s\n", err)
        }
        defer resp.Body.Close()
        // If not modified, use cache
        if resp.StatusCode == 304 {
            for k, v := range cachedRes.headers {
                for _, vv := range v {
                    w.Header().Set(k, vv)
                }
            }
            fmt.Fprint(w, string(cachedRes.body))
        }
    }
}
```

```
        // Update time and bandwidth saved
        elapsed := time.Since(startTimer)
        savingPointer := cacheSavings[URI]
        timeSaved := savingPointer.lastUncachedTime - elapsed
        cachedBodyLen := len(cachedRes.body)
        savingPointer.timeSaved += timeSaved
        savingPointer.dataSaved += cachedBodyLen

        // Logging
        if timeSaved > 0 {
            log.Printf("%s Completed Transmission: %s [%s] %s [%s]",
                colorOutput("HTTP", "green"), colorOutput(URI, "yellow"),
                colorOutput(elapsed.String(), "cyan"),
                colorOutput(strconv.Itoa(cachedBodyLen)+" bytes CACHED",
                    "gray"), colorOutput(timeSaved.String(), "green"))
        } else {
            log.Printf("%s Completed Transmission: %s [%s] %s [%s]",
                colorOutput("HTTP", "green"), colorOutput(URI, "yellow"),
                colorOutput(elapsed.String(), "cyan"),
                colorOutput(strconv.Itoa(cachedBodyLen)+" bytes CACHED",
                    "gray"), colorOutput((-timeSaved).String(), "red"))
        }
        // Logging

        return
    }
}

// Send to client
resp, err := http.Get(URI)
if err != nil {
    log.Printf("%s\n", err)
}
defer resp.Body.Close()
// Copies all headers
for k, v := range resp.Header {
    for _, vv := range v {
        w.Header().Set(k, vv)
    }
}
}
```

18315028

Yi Xiang Tan

```
// Copy body and pass to client
body, err := io.ReadAll(resp.Body)
if err != nil {
    log.Println(err)
}
fmt.Fprint(w, string(body))
elapsed := time.Since(startTimer)
// Logging
log.Printf("%s Completed Transmission: %s [%s]",
    colorOutput("HTTP", "green"), colorOutput(URI, "yellow"),
colorOutput(elapsed.String(), "cyan"))
// Logging

// Cache and set cache expiry
respDate := resp.Header.Get("date")
cachedData := cacheItem{
    body:    body,
    headers: resp.Header,
    date:    respDate,
}
cache[URI] = cachedData
savingPointer, exist := cacheSavings[URI]
if !exist {
    resourceSavedPointer := &cacheSaving{
        dataSaved:    0,
        timeSaved:    0,
        lastUncachedTime: elapsed,
    }
    cacheSavings[URI] = resourceSavedPointer
} else {
    savingPointer.lastUncachedTime = elapsed
}

go func(date string) {
    time.Sleep(CACHE_EXPIRY)
    cachedResp := cache[URI]
    if cachedResp.date == date {
        delete(cache, URI)
    }

    // Logging
```

18315028

Yi Xiang Tan

```
        log.Printf("%s for %s registered at %s\n", colorOutput("Killing  
cache", "red"), colorOutput(URI, "yellow"), colorOutput(cachedResp.date,  
"cyan"))  
        // Logging  
    }  
}(respDate)  
}  
  
func networkHandler(w http.ResponseWriter, req *http.Request) {  
    host := req.Host  
    // If not in blockList  
    if !blockList[host] {  
        // If HTTPS  
        if req.Method == "CONNECT" {  
            httpsHandler(w, req)  
            // If HTTP  
        } else {  
            // Handles connections to server  
            httpHandler(w, req)  
        }  
        // Return 403 if in blockList  
    } else {  
        log.Printf("%s %s\n", colorOutput("BLOCKED", "red"),  
colorOutput(req.Host, "yellow"))  
        w.WriteHeader(http.StatusForbidden)  
    }  
}  
  
func CLIHandler() {  
    fmt.Println("Proxy Console [:8080]")  
    reader := bufio.NewReader(os.Stdin)  
  
    for {  
        text, err := reader.ReadString('\n')  
        if err != nil {  
            log.Println(err)  
        }  
        // convert CRLF to LF  
        text = strings.Replace(text, "\r\n", "", -1)  
        arguments := strings.Split(text, " ")  
        // To show blocklist  
        if arguments[0] == "list" {
```

```
        log.Printf("%v\n", blockList)
        // To block
    } else if arguments[0] == "block" {
        _, exist := blockList[arguments[1]]
        if exist {
            log.Printf("%s %s", colorOutput("ALREADY BLOCKED", "red"),
colorOutput(arguments[1], "yellow"))
        } else {
            blockList[arguments[1]] = true
            log.Printf("%s%s %s", colorOutput("B", "red"),
colorOutput("LOCKED", "green"), colorOutput(arguments[1], "yellow"))
        }
        // To unblock
    } else if arguments[0] == "unblock" {
        _, exist := blockList[arguments[1]]
        if exist {
            delete(blockList, arguments[1])
            log.Printf("%s %s", colorOutput("UNBLOCKED", "green"),
colorOutput(arguments[1], "yellow"))
        } else {
            log.Printf("%s %s", colorOutput("NOT BLOCKED", "red"),
colorOutput(arguments[1], "yellow"))
        }
        // To list data/time saved
    } else if arguments[0] == "saved" {
        totalDataSaved := 0
        totalTimeSaved := time.Duration(0)
        for _, saving := range cacheSavings {
            totalDataSaved += saving.dataSaved
            totalTimeSaved += saving.timeSaved
        }
        log.Printf("total data saved %s bytes, total time saved %v",
colorOutput(strconv.Itoa(totalDataSaved), "green"),
colorOutput(totalTimeSaved.String(), "green"))

    } else {
        log.Printf("%s %s %s | list | %s\n", colorOutput("WRONG INPUT:",
"red"), colorOutput("(un)?block", "cyan"), colorOutput("HOST", "yellow"),
colorOutput("saved", "green"))
    }
}
```

18315028
Yi Xiang Tan

```
}  
  
func main() {  
    go CLIHandler()  
    networkHandler := http.HandlerFunc(networkHandler)  
    // Create a thread of networkHandler for each connection  
    http.ListenAndServe(":8080", networkHandler)  
}
```