

**Instituto Superior de Engenharia**

**De Lisboa**

Licenciatura em Engenharia Informática e Multimédia

Redes de Computadores – 2024/2025 SV

**1ª Fase – Servidor Web**

Docente: Nuno Garcia

Realizado pelo grupo NG-16:  
Carolina Raposo n.º 51568

Carlos Simões n.º 51696  
Lara Camões n.º51742

*Lisboa, 29 de março de 2025*

**1. Introdução**

O presente relatório descreve as etapas tomadas e resultados alcançados durante a execução da Fase 1 do projeto da unidade curricular Redes de Computadores. Este projeto tem como objetivo principal a construção de uma rede de computadores, com fases progressivas de complexidade, desde a criação ode um servidor *web* até à implementação de uma rede corporativa típica.

A primeira fase do projeto foca na instalação e configuração de um servidor *web* num computador local, realização de testes de conectividade para garantir o funcionamento adequado deste servidor, utilização da ferramenta WireShark para capturar o acesso à web a partir de um *host* remoto e comparar os cabeçalhos HTTP enviados pelo cliente e pelo servidor e no desenvolvimento de um cliente web utilizando uma linguagem de programação à nossa escolha, no qual é estabelecida uma conexão TCP com o servidor e, então é realizado testes para verificar o seu funcionamento e analisar as várias respostas.

Neste relatório, iremos apresentar capturas das configurações e resultados obtidos, incluindo testes de conectividade, bem como o código-fonte do cliente web desenvolvido.

**2. Configuração do Servidor Web**

Para a implementação do servidor *web* local, foi utilizado o pacote XMAPP, que inclui o servidor Apache, MySQL, PHP, e Perl. O módulo Apache foi devidamente inicializado através do painel do controlo do XMAPP, como podemos ver na Figura 1, onde podemos ver que o seu correto funcionamento.

**Uma imagem com texto, captura de ecrã, software, Ícone de computador

Os conteúdos gerados por IA poderão estar incorretos.**

Figura 1 – Painel de controlo XMAPP

Após a inicialização do servidor Apache, procedemos à verificação do seu funcionamento utilizando dois métodos distintos.

Primeiramente, realizamos o teste de acesso local através do endereço *loopback* http://127.0.0.1/. Como mostra a Figura 2, este método permitiu confirmar que o serviço estava corretamente inicializado e respondendo às requisições HTTP na máquina local, sendo que podemos observar a página padrão do XMAPP.



Figura 2 – Teste de acesso local ao Servidor Web

Posteriormente, para verificar a acessibilidade do servidor por outros dispositivos na rede local, executamos o comando **ipconfig** para identificar o endereço IP da interface de rede ativa, neste caso, identificamos o endereço IPv4 na secção “Wireless LAN adapter Wi-Fi” da saída do comando, como mostra a Figura 3. Com este endereço (no formato http://192.168.1.221./), conseguimos aceder ao servidor a partir de um dispositivo móvel conectado à mesma rede, conforme demostrado na Figura 4.

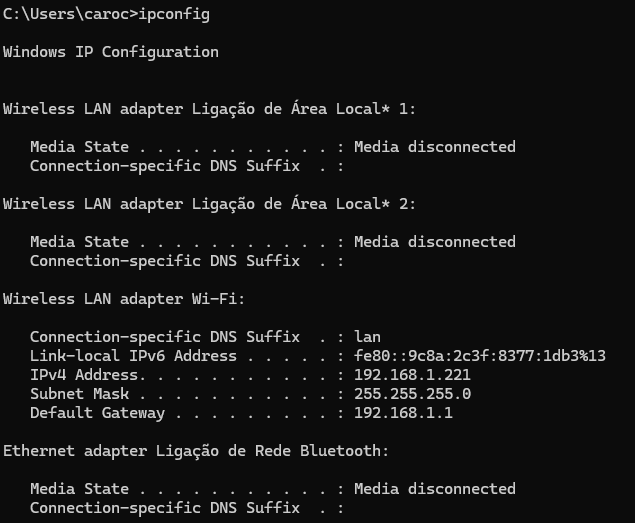


Figura 3 – Linha de comando após comando ipconfig

Figura 4 - Teste de acesso a um dispositivo móvel

**3. Captura de tráfego HTTP utilizando o Wireshark**

O objetivo desta parte foi capturar e analisar o tráfego **HTTP** entre um cliente e um servidor web, utilizando a ferramenta **Wireshark**.

Para isso, foi estabelecida uma comunicação com o servidor Apache previamente instalado, acedendo ao endereço **http://127.0.0.1/** a partir de um navegador. Durante esse acesso, o **Wireshark** foi utilizado para monitorizar e capturar os pacotes de rede trocados entre o cliente (browser) e o servidor.

Esta parte foi feita entre dois dispositivos, e foi utilizada a aplicação ***Radmin VPN*** para se poder estar numa rede virtual, e então fez-se a captura dos pacotes por meio da interface **Radmin VPN**.

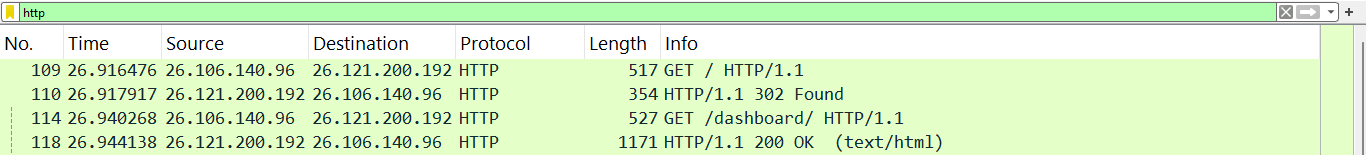
A seguinte tabela mostra os headers enviados e a sua explicação

Figura 5 - Verificação do pedido no Wireshark

Na linha 109, os headers enviados pelo cliente contêm informações essenciais sobre a requisição:

* **GET / HTTP/1.1\r\n** → Indica que o cliente está a solicitar um recurso (neste caso, a página inicial /) e define a versão do protocolo HTTP utilizada
* **Host: 26.121.200.192** → Especifica o servidor ao qual o pedido está a ser feito
* **Connection: keep-alive** → Indica que o cliente deseja manter a ligação TCP aberta para reutilização
* **DNT: 1** → O cliente solicita que não seja rastreado (Do Not Track)
* **Upgrade-Insecure-Requests: 1** → Informa que prefere uma resposta HTTPS se disponível
* **User-Agent: Mozilla/5.0 (...) Edg/134.0.0.0** → Identifica o navegador como Microsoft Edge, baseado no motor Chrome/WebKit
* **Accept: text/html, application/xml, image/webp, \*/\*;q=0.8** → Define os tipos de conteúdo que o cliente pode processar, priorizando HTML, XML e imagens modernas
* **Accept-Encoding: gzip, deflate** → Indica suporte a compressão de resposta para otimização de transferência
* **Accept-Language: en-GB,en;q=0.9,en-US;q=0.8** → Declara a preferência por conteúdos em inglês britânico, seguido de inglês geral e americano

Na linha 110, os headers enviados pelo servidor fornecem detalhes cruciais sobre a resposta à requisição do cliente:

* **HTTP/1.1 302 Found\r\n** → Indica que o servidor está a utilizar a versão **1.1 do protocolo HTTP** e o código de estado HTTP que indica que o recurso solicitado foi encontrado, mas **foi movido temporariamente para outro local**. O cliente deve seguir o novo URL indicado no header **Location**
* **Date: Sun, 30 Mar 2025 18:14:15 GMT** → Indica a data e hora em que o servidor processou a resposta
* **Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12** → Identifica o software utilizado pelo servidor, incluindo a versão do Apache, OpenSSL e PHP
* **X-Powered-By: PHP/8.2.12** → Informa que o servidor está a utilizar PHP para processar a página
* **Location: http://26.121.200.192/dashboard/** → Indica que o recurso solicitado foi movido temporariamente e que o cliente deve redirecionar para este novo URL
* **Content-Length: 0** → Especifica que a resposta não contém corpo, apenas os headers
* **Keep-Alive: timeout=5, max=100** → Define que a ligação TCP será mantida ativa por até 5 segundos e poderá ser reutilizada para até 100 requisições
* **Connection: Keep-Alive** → Confirma que o servidor deseja manter a ligação aberta para futuras comunicações
* **Content-Type: text/html; charset=UTF-8** → Especifica que, caso houvesse um corpo na resposta, ele estaria no formato HTML e codificado em UTF-8

Na linha 114, os headers enviados pelo cliente contêm informações essenciais sobre a nova requisição, efetuada após o redirecionamento:

* **Request Method: GET** → O cliente solicita o recurso /dashboard/ ao servidor utilizando o método **GET**.
* **Request URI: /dashboard/** → Indica o caminho do recurso solicitado dentro do servidor.
* **Request Version: HTTP/1.1** → Especifica que a requisição segue o protocolo **HTTP/1.1**.
* **Host: 26.121.200.192** → Define o endereço do servidor ao qual o pedido está a ser feito.
* **Connection: keep-alive** → O cliente solicita que a ligação TCP permaneça aberta para reutilização.
* **DNT: 1** → Indica que o cliente deseja evitar rastreamento (Do Not Track).
* **Upgrade-Insecure-Requests: 1** → Sugere que o cliente prefere HTTPS se estiver disponível.
* **User-Agent: Mozilla/5.0 (...) Edg/134.0.0.0** → Identifica o navegador e o sistema operativo do cliente (Microsoft Edge, baseado em Chrome/WebKit).
* **Accept: text/html, application/xhtml+xml, application/xml, image/webp, \*/\*;q=0.8** → Especifica os tipos de conteúdo aceites pelo cliente, priorizando HTML, XML e imagens modernas.
* **Accept-Encoding: gzip, deflate** → Indica que o cliente suporta compressão de resposta para otimizar a transferência de dados.
* **Accept-Language: en-GB,en;q=0.9,en-US;q=0.8** → Expressa a preferência por conteúdos em inglês britânico, seguido de inglês geral e americano.

Na linha 118, os headers enviados pelo servidor fornecem informações detalhadas sobre a resposta à requisição do cliente:

* **HTTP/1.1 200 OK** → O código de status **200 OK** indica que a requisição foi bem-sucedida e que o recurso solicitado está disponível.
* **Date: Sun, 30 Mar 2025 18:14:15 GMT** → A data e hora em que o servidor processou a resposta, em formato **GMT**.
* **Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12** → Identifica o software do servidor, incluindo a versão do Apache, OpenSSL (para encriptação) e PHP.
* **Last-Modified: Sun, 19 Nov 2023 11:10:25 GMT** → Indica a última vez que o conteúdo foi modificado, o que pode ser útil para cache e controle de versões.
* **ETag: "1443-60a7f6a8cca40"** → Um identificador único para a versão do conteúdo, utilizado para otimizar o cache.
* **Accept-Ranges: bytes** → Indica que o servidor suporta **intervalos de bytes** e que o cliente pode pedir apenas uma parte específica do recurso, se necessário.
* **Content-Length: 5187** → Especifica que o corpo da resposta tem **5187 bytes**, ou seja, o tamanho do conteúdo HTML enviado pelo servidor.
* **Keep-Alive: timeout=5, max=99** → Define que a ligação TCP será mantida ativa por até 5 segundos e pode ser reutilizada até 99 vezes para futuras requisições.
* **Connection: Keep-Alive** → Indica que a ligação TCP será mantida aberta após a resposta para possíveis requisições subsequentes.
* **Content-Type: text/html** → Especifica que o conteúdo da resposta é do tipo **HTML**, o que significa que o cliente deve interpretar os dados como uma página web.

**4. Criação de um Cliente Web**

Nesta etapa do trabalho, foi desenvolvido um cliente HTTP simples, utilizando apenas as bibliotecas de *socket* disponíveis na linguagem de programação escolhida, sem recorrer a bibliotecas HTTP prontas. O cliente deverá estabelecer uma ligação TCP com o servidor web, enviar uma requisição HTTP e processar a resposta recebida, exibindo-a ao utilizador.

O código desenvolvido foi o seguinte:

import java.io.\*;  
import java.net.\*;  
import java.util.Scanner;  
  
public class WebClient\_NG\_16 {  
  
 public Socket webClient(String serverAddress, int serverPort, String target) throws IOException {  
 //estabelecer a conexão TCP com o servidor  
 Socket socket = new Socket(serverAddress, serverPort);  
  
 //preparar a request HTTP  
 String request = "GET " + target + " HTTP/1.1\r\n" +  
 "Host: " + serverAddress + "\r\n" +  
 "Connection: close\r\n" + //fecho de comunicação após resposta  
 "\r\n"; //linha em branco para indicar o fim do header  
  
 //obter o fluxo de saída do socket para enviar os dados para o servidor  
 OutputStream outputStream = socket.getOutputStream();  
 //converter a String de request para bytes  
 outputStream.write(request.getBytes());  
 //garantir que os dados são todos enviados  
 outputStream.flush();  
  
 //obter o fluxo de entrada para receber os dados que o servidor enviou  
 InputStream inputStream = socket.getInputStream();  
 //criar um buffer para ler o fluxo de entrada  
 BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));  
 //variável para guardar cada linha da resposta  
 String line;  
 //objeto para concatenar todas as linhas  
 StringBuilder response = new StringBuilder();  
  
 //lê cada linha até o fluxo acabar  
 while ((line = reader.readLine()) != null) {  
 //adiciona ao StringBuilder cada linha, de modo a que cada uma fique numa linha diferente (\n)  
 response.append(line).append("\n");  
 }  
  
 //dar display da resposta do servidor  
 System.*out*.println("----------------- Resposta do Servidor ----------------");  
 System.*out*.println(response.toString());  
  
 //retornar a socket  
 return socket;  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 try (scanner) {  
 System.*out*.println("Insira o endereço de servidor: ");  
 String address = scanner.nextLine();  
 System.*out*.println("Insira o número da porta: ");  
 int port = scanner.nextInt();  
 scanner.nextLine();  
 System.*out*.println("Insira a target: ");  
 String target = scanner.nextLine();  
 WebClient\_NG\_16 client = new WebClient\_NG\_16();  
 Socket webclient = client.webClient(address, port, target);  
 //tratamento de errros  
 } catch (UnknownHostException e) {  
 System.*err*.println("Erro: Host desconhecido - " + e.getMessage());  
 } catch (IOException e) {  
 System.*err*.println("Erro I/O - " + e.getMessage());  
 }  
 }  
}

O código funciona da seguinte maneira:

1. Lê o IP, porta e *target* do utilizador
2. Estabelece uma conexão TCP com o servidor
3. Envia um pedido HTTP "GET" para obter uma página
4. Recebe e imprime a resposta HTTP
5. Trata erros de rede e entrada/saída

A classe foi desenvolvida de modo que:

* Seja estabelecida uma conexão TCP com um servidor web
* Um pedido HTTP (GET) seja enviado sem usar bibliotecas HTTP externas
* Haja a receção da resposta do servidor e que esta seja exibida
* Seja feita a captura de erros de rede e de entrada/saída

**5. Comparação das capturas**

Nesta última fase, foi pedida a analise e a comparação do tráfego criado pelo cliente web desenvolvido manualmente, com o tráfego capturado ao aceder ao mesmo servidor através de um navegador convencional.

Para isto, foi feita a captura usando o *Loopback Traffic Capture*, que se refere à captura de tráfego que circula internamente dentro do próprio sistema, sem sair para a rede física.

Primeiramente, iniciou-se o servidor Apache e colou-se 127.0.0.1 no browser (neste caso foi usado o Microsoft Edge), depois, antes de abrir o site, iniciou-se a captura no Wireshark, e por fim, abriu-se o site, ao qual o Wireshark capturou o seguinte:



Figura 6 – Pacotes da primeira captura

**Pacote 116**:

* Uma requisição **GET** foi enviada para / usando HTTP/1.1
* Isto significa que o cliente pediu a página principal (/) de um servidor web local

**Pacote 118**:

* O servidor respondeu com o código **302 Found**
* Este código significa que a página solicitada não está disponível diretamente e o cliente deve ser redirecionado para outro local

Após estas capturas, e ainda com o Wireshark a capturar, utilizou-se o cliente desenvolvido, sendo que o endereço do servidor foi o do computador que estava a fazer a captura, a porta foi a 80 e a *target* foi */dashboard*, diferente da *target* anterior “/”.

Ao correr o código, este mostrou o seguinte:

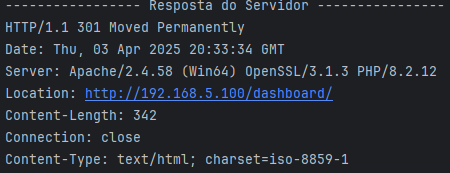


Figura 7 – Resposta do WebClient

No lado do Wireshark, as informações do *header* HTTP foram as mesmas:

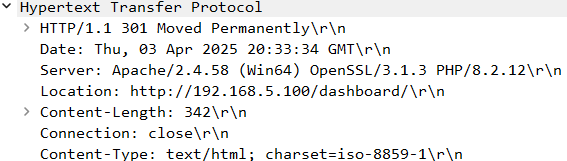
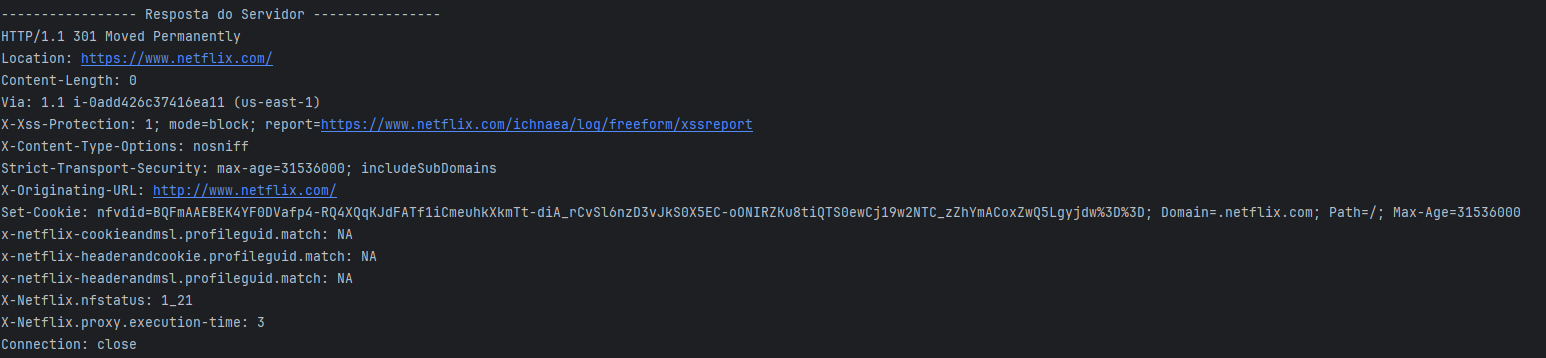


Figura 8 – Captura feita pelo Wireshark após correr o WebClient

Isto aconteceu porque, como se usou a *Loopback Traffic Capture*, capturou-se o tráfego interno do computador, ou seja, todas as comunicações entre aplicações locais, daí o *output* do *Java* ter sido igual ao pacote capturado pelo *Wireshark*.

**5.1. Testes com WebClient**

Para além do servidor Apache, testou-se o programa com outros servidores, como:



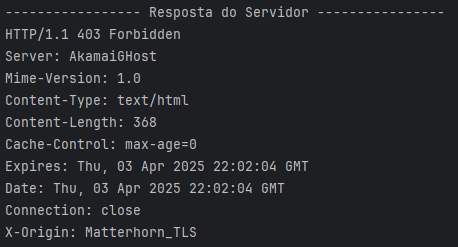


Figura 9 – Resposta do WebClient ao servidor da www.netflix.com

Figura 10 – Resposta do WebClient ao servidor da www.disney.com

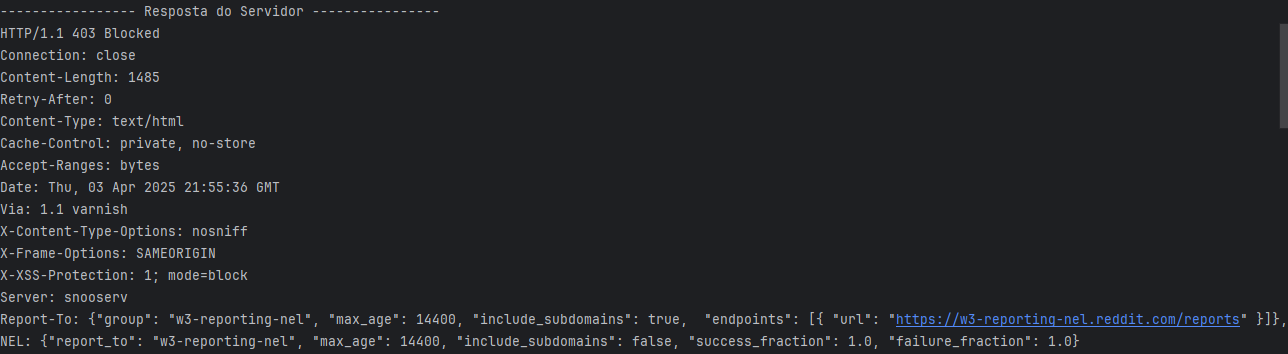


Figura 11 – Resposta do WebClient ao servidor do www.reddit.com

No caso do servidor da *Netflix*, o código recebido foi **301 *Moved* *Permanently***, que significa que a página ou recurso que o *WebClient* solicitou não está disponível no endereço atual, e que foi **permanentemente movido** para um novo endereço. O servidor então fornece o novo URL para onde o recurso foi movido, geralmente no cabeçalho *Location*.

Já a resposta ao servidor da *Disney* mostrou o código **403 *Forbidden***, que acontece quando o servidor reconhece a requisição, mas o cliente não tem permissão para aceder ao recurso solicitado. Isto pode acontecer por várias razões, como problemas de **autorização**, **restrições de acesso** ou **políticas de segurança**.

Por fim, ao tentar aceder ao *Reddit*, o *WebClient* apresentou o código **403 *Blocked***, ou seja, o servidor está a bloquear ativamente o acesso ao recurso solicitado. A razão para o bloqueio pode variar, mas está frequentemente relacionada com **políticas de segurança** mais rigorosas que estão a ser aplicadas no servidor ou na rede.