FUNDAÇÃO UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CAMPUS JUAZEIRO
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Breno Gabriel de Souza Coelho

Atividade HTTP e prática com Wireshark

Juazeiro, BA
2022

Breno Gabriel de Souza Coelho

Atividade HTTP e prática com Wireshark

Relatório requerido pelo professor Fábio Nelson de Sousa Pereira, como parte das exigências de avaliação da disciplina Redes de Computadores I, do curso de Engenharia da Computação, da Universidade do Vale do São Francisco.

Juazeiro, BA
2022

SUMÁRIO

INTRODUÇÃO 3

METODOLOGIA 5

DISCUSSÃO 6

CONCLUSÃO 12

REFERÊNCIA BIBLIOGRÁFICA 13

1 INTRODUÇÃO

HTTP (Hypertext Transfer Protocol) é um protocolo de transferência de pacotes de informação entre uma aplicação cliente (programa executado num sistema computacional) e uma aplicação servidora. Através dele é possível criar o fluxo de dados, no nível da camada de aplicação, que permite o funcionamento da WWW. Utilizado desde a década de 90, começou como uma norma simples que possuía somente o comando GET, na versão 0.9, e evoluiu com o tempo dando origem a versão 1.0, em 96, que já contava com uma série de adições importantes. Mas é na versão 1.1, de 97, que é consolidado o formato do protocolo. Alguns RCF's (Requests for Comments) foram publicados desde seu lançamento para melhorar a clareza da tecnologia. Em 2015 foi anunciado o lançamento do HTTP/2.0, e já existem planejamentos sobre uma futura versão 3.0. (BERNERS-LEE, TIM., 1993, apud MOGUL, JEFFREY C., 1995)

A versão 1.1 do HTTP apresentou algumas significativas melhoras em relação ao seu antecessor. Em um conjunto de testes comparativos realizados entre os dois mostraram que a versão 1.1, operando como pipelining, conseguiu superar a 1.0 em todas as situações, mesmo quando essa usava um conjunto de conexões em paralelo. Em termos de pacotes transmitidos, foram observados ganhos de pelo menos 2 pacotes, chegando até máximos de 10 em algumas situações. Tempos de espera também tiveram uma diferença, porém menos drástica, e mais dependente da qualidade de conexão do dispositivo. Esse mesmo estudo concluiu que para poder alcançar o máximo de eficácia no uso dessa versão é importante ter uma implementação adequada de tecnologias, em particular o uso de CSS *style sheets* e mais representações com png compactos. (NIELSEN et al, 1997).

De acordo com POPA, et al. (2010), o HTTP já provou, e continua a provar, que é um protocolo sólido e capaz de implementar boa parte das funções que se pretende para todas as arquiteturas de internet. Algumas de suas vantagens são o fato de ser um protocolo centralizador, apresentar suporte de middleboxes na forma de proxys diretos e reversos e outros. Contudo ele ainda possui algumas limitações, para o qual é sugerida a criação de uma extensão chamada S-GET, com objetivo de dar suporte para aplicações de baixa latência, como VoIP e chat.

Outra crítica recorrente é relacionada ao cache de rede, e a forma como arquivos precisam ser recolocados por completo sempre que modificados. Uma proposta cada vez mais forte sugere o uso de "delta-encoding", um processo em que somente a diferença entre um arquivo presente na rede em relação ao que se pretende adicionar é de fato enviada (método que seria usado quando o arquivo a ser adicionado possuí uma parte comum com aquele na rede). Estudos realizados em cima desse mecanismo mostram ganhos substanciais na velocidade de envio para esses subcasos recorrentes, além de melhor efeito quando combinado com uma compactação dos arquivos de envio. (MOGUL, JEFFREY C. et al, 1997)

O posicionamento de MOGUL, JEFFREY C. (1995) afirma que o http faz um uso ineficiente de recursos da rede, adicionando latências desnecessárias com a criação de um pacote TCP novo para cada requisição de sistema. O artigo trata de um conjunto de simulações de longa duração feitas com diferentes propostas de mudanças, muitas das quais se baseiam em utilizar uma mesma conexão TCP para múltiplos ciclos de pedido-resposta. Os resultados animadores, na palavra do autor, “demonstram o valor das conexões persistentes”.

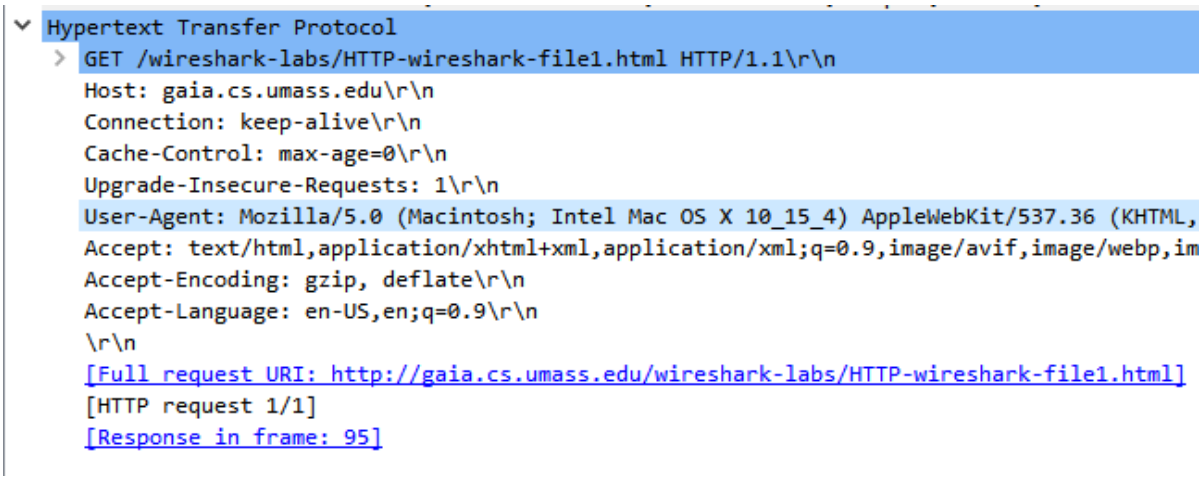
2 METODOLOGIA

Abaixo a lista de perguntas feitas no arquivo

1. Is your browser running HTTP version 1.0, 1.1, or 2? What version of HTTP is the server running?
2. What languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of your computer? What is the IP address of the gaia.cs.umass.edu server?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you are retrieving last modified at the server?
6. How many bytes of content are being returned to your browser?
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one
8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?
9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
10. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?
11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.
12. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?
13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
14. What is the status code and phrase in the response?
15. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?
16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
17. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.
18. What is the server’s response (status code and phrase) in response to the initial HTTP GET message from your browser?
19. When your browser’s sends the HTTP GET message for the second time, what new field is included in the HTTP GET message? .

3 DISCUSSÃO

Aqui estão as respostas para as questões apresentadas no arquivo “Wireshark HTTP v8.1”. Na primeira parte, o pedido feito ao servidor teve a seguinte estrutura:



1. Pela primeira linha em cada um (linha de pedido), vemos que a versão do http usada é a 1.1.
2. Analisando a linha com “Accept-Language” no pedido (enviado pelo browser) vemos que ele tem uma preferencia pelo Inglês americano (United States – US)
3. Isso não pode ser visto nos “prints” acima, mas na interface do wireshark, da tabela “Source” e “Destination” vemos que o IP de origem (do sistema usado para fazer esse packet) é 10.0.0.44, e o sistema destino é 128.119.245.12
4. Da linha de pedido da resposta, vemos que o código recebido foi “200” (o pedido foi processado e respondido sem problemas)
5. No pedido temos o cabeçalho “Last-modified”, onde lê-se que a última modificação ocorreu em 30/01/2021, um sábado, às 06:59

(coincidentemente, também é dia 30/01 enquanto escrevo isso, o que me fez ficar confuso se eu não teria modificado o arquivo de alguma forma. Mas eu só tinha confundido o ano mesmo, afinal é 2021 na resposta)

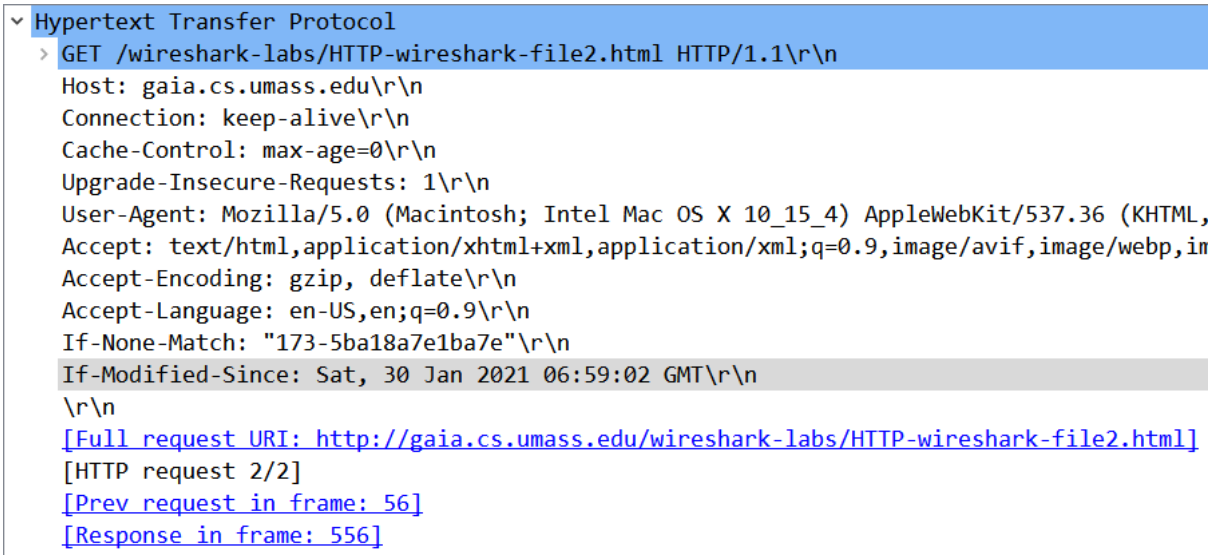
6. Pela última linha da resposta, 128 bytes.
7. Não, são idênticos (tanto o request com o response)

```
Hypertext Transfer Protocol
  GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, li
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image,
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9\r\n
  \r\n
  [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html]
  [HTTP request 1/2]
  [Response in frame: 58]
  [Next request in frame: 555]
```

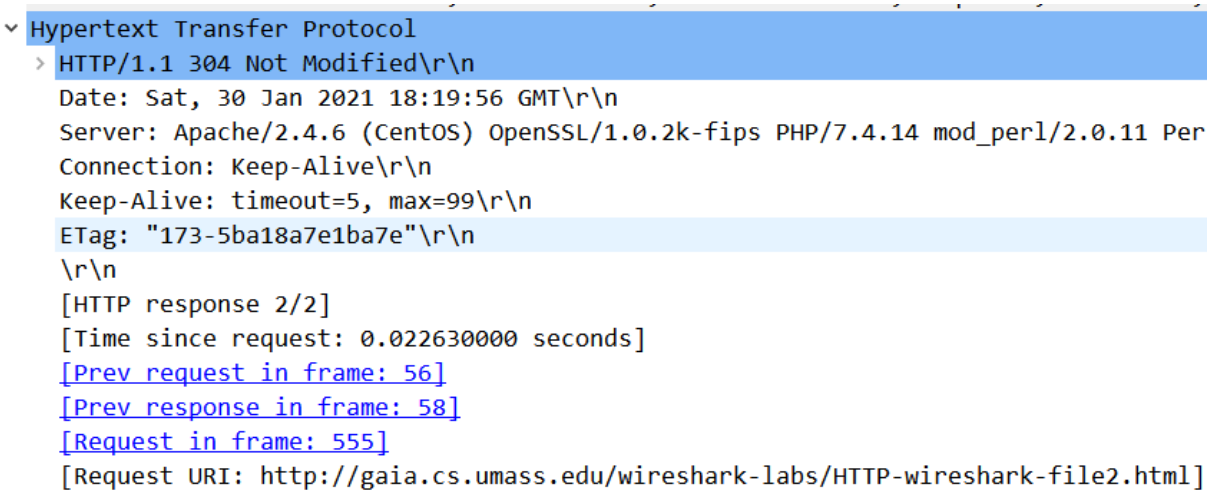
8. A imagem acima é o pedido feito pelo browser. Não existe nenhuma linha de cabeçalho com “If-Modified-Since”

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Sat, 30 Jan 2021 18:19:53 GMT\r\n
  Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.14 mod_perl/2.0.11 Perl
  Last-Modified: Sat, 30 Jan 2021 06:59:02 GMT\r\n
  ETag: "173-5ba18a7e1ba7e"\r\n
  Accept-Ranges: bytes\r\n
  Content-Length: 371\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  \r\n
  [HTTP response 1/2]
  [Time since request: 0.024609000 seconds]
  [Request in frame: 56]
  [Next request in frame: 555]
  [Next response in frame: 556]
  [Request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html]
  File Data: 371 bytes
```

9. A imagem acima é a resposta do servidor. O código 200 na linha de pedido mostra que ele foi processado com sucesso, e a linha de cabeçalho “Content-Type” mostra que o tipo do pacote foi um texto html. Junto com o fato de que ao final da resposta temos o tamanho do arquivo enviado (371 bytes), é certo dizer que recebemos um arquivo.



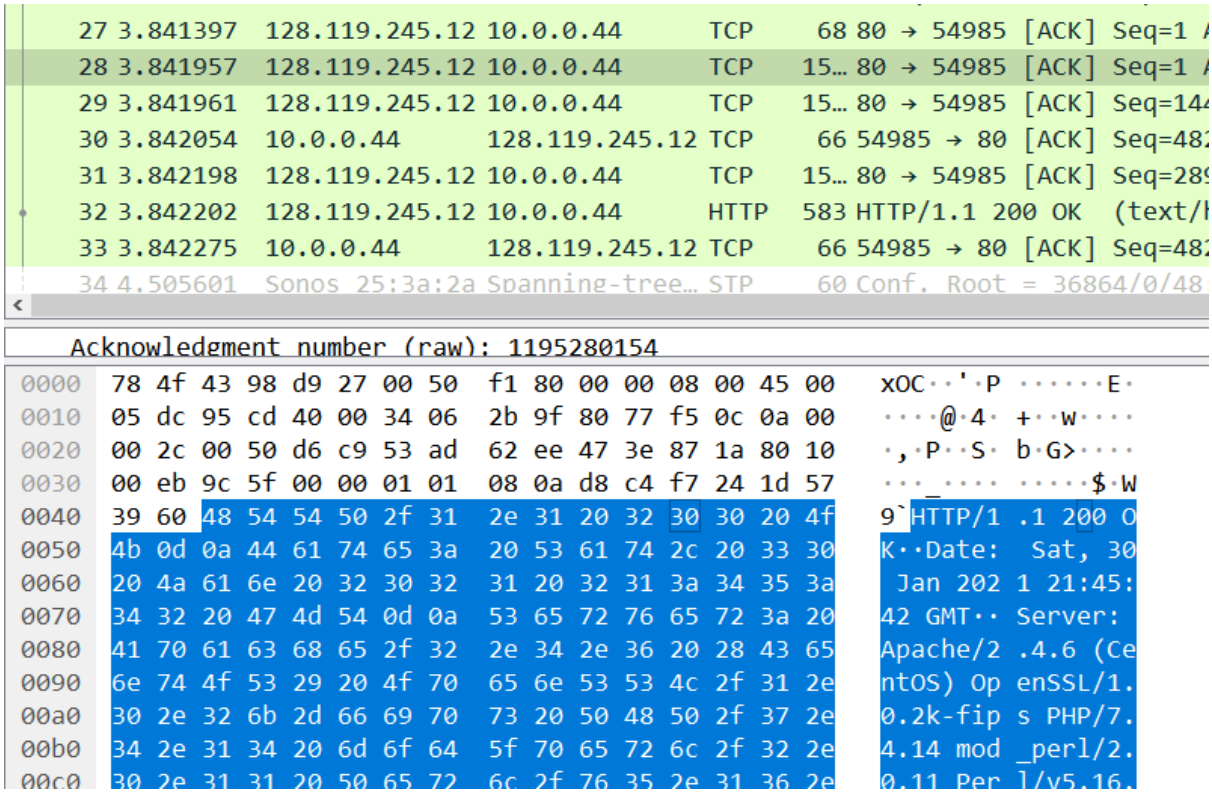
10. Sim, olhando a imagem acima podemos ver a linha com o “If-Modified-Since” (traduzindo literalmente, se modificado desde) seguido por: “Sat, 30 Jan 2021 06:59:02 GMT\r\n”



11. Agora, analisando a resposta acima, vemos que foi retornado um código 300, o que significa que o pedido não pode ser processado porque alguma ação adicional seria necessária. Nesse caso devido a condição “If-modified-since”, o servidor irá verificar se o arquivo buscado foi modificado desde a última vez que foi enviado para o navegador, caso negativo (que é o que ocorreu, pois foram dois pedidos feitos num intervalo muito curto), nada será enviado. Isso evita um reprocessamento desnecessário de pacotes. O código de retorno, 304, possui a mensagem “Not Modified”, indicando que o pacote não foi enviado não porque houve um erro de requisição ou sistema, mas simplesmente porque as condições apresentadas faziam com que ele não precisasse ser enviado caso não tivesse sido modificado.

```
26 3.813230  10.0.0.44      128.119.245.12 HTTP   547 GET /wireshark-labs/HTTP-wireshark-file3
27 3.841397  128.119.245.12 10.0.0.44      TCP     68 80 → 54985 [ACK] Seq=1 Ack=482 Win=30080
28 3.841957  128.119.245.12 10.0.0.44      TCP   15… 80 → 54985 [ACK] Seq=1 Ack=482 Win=30080
29 3.841961  128.119.245.12 10.0.0.44      TCP   15… 80 → 54985 [ACK] Seq=1449 Ack=482 Win=30
30 3.842054  10.0.0.44      128.119.245.12 TCP     66 54985 → 80 [ACK] Seq=482 Ack=2897 Win=12
31 3.842198  128.119.245.12 10.0.0.44      TCP   15… 80 → 54985 [ACK] Seq=2897 Ack=482 Win=30
32 3.842202  128.119.245.12 10.0.0.44      HTTP  583 HTTP/1.1 200 OK  (text/html)
```

12. Olhando a imagem acima, vemos que somente um packet com a mensagem GET foi enviado pelo browser. Podemos ver ainda que esse foi o pacote 26 trocado nessa captura.



13. Observando os dados enviados no pacote acima vemos que a linha de pedido da mensagem de resposta está em meio aos dados desse pacote, com o código do pedido e a frase associada – 200 OK -. Portanto a resposta é o pacote TCP de número 28.
14. 200 OK. O pedido foi recebido e processado com sucesso

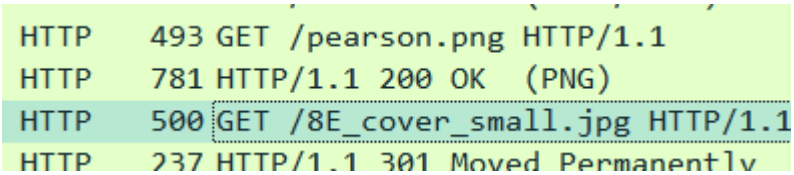
```
HTTP   547 GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1
TCP     68 80 → 54985 [ACK] Seq=1 Ack=482 Win=30080 Len=0 TSval=3636786978 TSecr=492255584
TCP   1514 80 → 54985 [ACK] Seq=1 Ack=482 Win=30080 Len=1448 TSval=3636786980 TSecr=492255584 [TCP segment of a reassembled PDU]
TCP   1514 80 → 54985 [ACK] Seq=1449 Ack=482 Win=30080 Len=1448 TSval=3636786980 TSecr=492255584 [TCP segment of a reassembled PDU]
TCP     66 54985 → 80 [ACK] Seq=482 Ack=2897 Win=128832 Len=0 TSval=492255612 TSecr=3636786980
TCP   1514 80 → 54985 [ACK] Seq=2897 Ack=482 Win=30080 Len=1448 TSval=3636786980 TSecr=492255584 [TCP segment of a reassembled PDU]
HTTP   583 HTTP/1.1 200 OK  (text/html)
```

15. Observando a imagem acima vemos que somente os pacotes 28, 29 e 31 possuem a terminação “TCP segment of a reassembled PDU”, ou segmento TCP de um PDU remontado. Como explicado no texto, a longa

mensagem presente no site foi quebrada em vários pacotes TCP enviados separadamente e depois juntados novamente. De toda forma, como a resposta HTTP vem logo em seguida, pacotes fora desse intervalo não estão relacionados ao pedido HTTP feito. Logo, o arquivo foi fragmentado em um conjunto de 3 pacotes TCP.

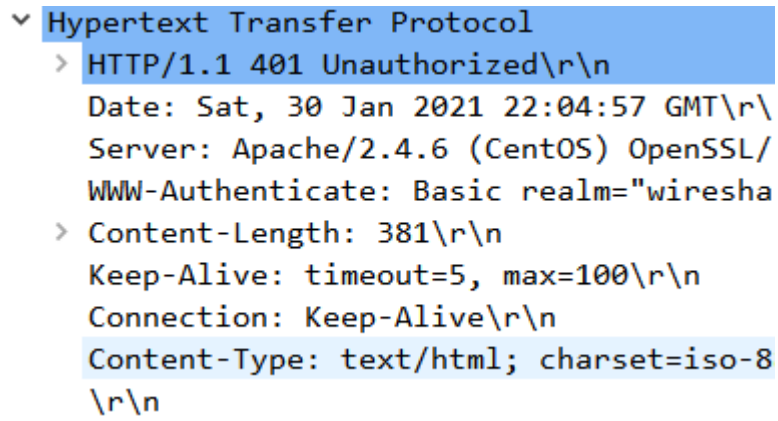
No.	Time	Source	Destination	Protocol	Length	Info
95	3.018199	10.0.0.44	128.119.245.12	HTTP	547	GET /wireshark-labs/HTTP-wireshark-f
97	3.048563	128.119.245.12	10.0.0.44	HTTP	1367	HTTP/1.1 200 OK (text/html)
99	3.072335	10.0.0.44	128.119.245.12	HTTP	493	GET /pearson.png HTTP/1.1
104	3.092770	128.119.245.12	10.0.0.44	HTTP	781	HTTP/1.1 200 OK (PNG)
118	3.309908	10.0.0.44	178.79.137.164	HTTP	500	GET /8E_cover_small.jpg HTTP/1.1
120	3.451822	178.79.137.164	10.0.0.44	HTTP	237	HTTP/1.1 301 Moved Permanently
144	3.757683	10.0.0.44	104.98.115.146	HTTP	361	GET /MFgwVqADAgEAME8wTTBLMAkGBSsOAwI
155	3.844477	104.98.115.146	10.0.0.44	OCSP	955	Response

16. Quatro mensagens. As duas primeiras foram endereçadas ao mesmo IP, 128.129.245.12, a terceira para 178.79.137.64 e a última para 104.98.115.146.



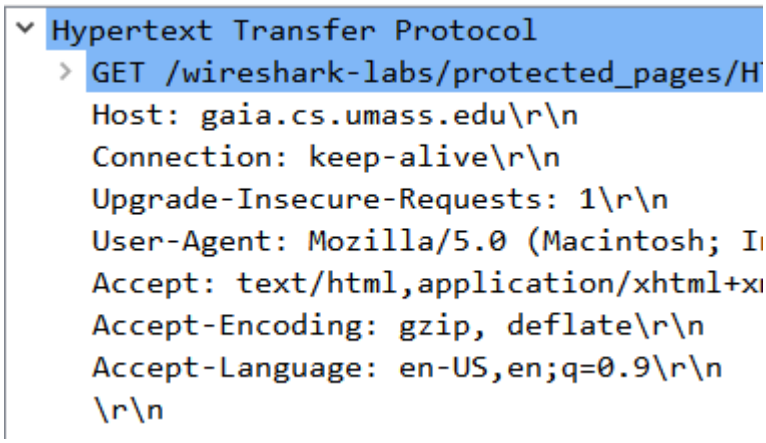
HTTP	493	GET /pearson.png HTTP/1.1
HTTP	781	HTTP/1.1 200 OK (PNG)
HTTP	500	GET /8E_cover_small.jpg HTTP/1.1
HTTP	237	HTTP/1.1 301 Moved Permanently

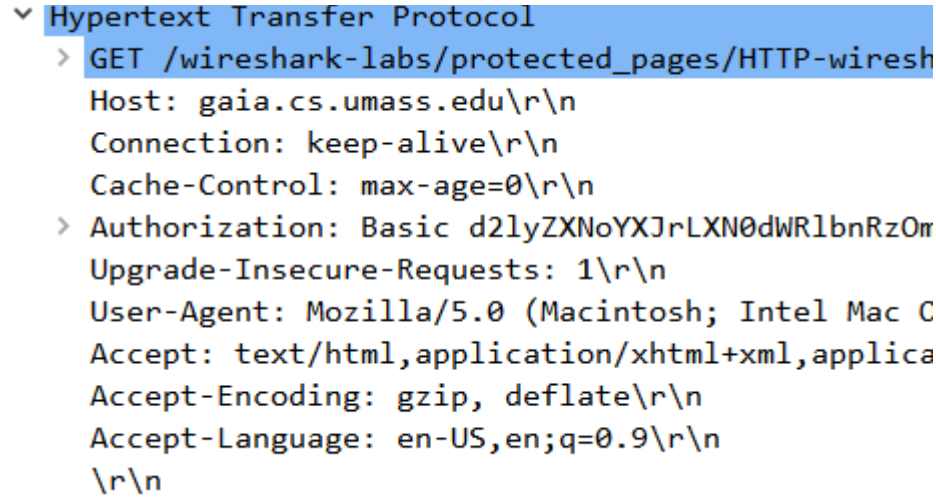
17. Pela porção acima vemos que duas mensagens GET separadas foram enviadas, a primeira para obter o ícone da Pearson e a segunda para obter a capa do livro.



```
Hypertext Transfer Protocol
  HTTP/1.1 401 Unauthorized\r\n
  Date: Sat, 30 Jan 2021 22:04:57 GMT\r\
  Server: Apache/2.4.6 (CentOS) OpenSSL/
  WWW-Authenticate: Basic realm="wiresha
  Content-Length: 381\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html; charset=iso-8
  \r\n
```

18. Pela imagem acima, 401 Unauthorized. Aparentemente os dados estavam errados





19. Acima estão o primeiro e o segundo resquest. Vemos que existem algumas linhas de cabeçalho novas no segundo: Cache-Control, Authorization e Upgrade-Insecure-Requests. Acredito que a principal seja o Authorization, onde lê-se um Basic. Possivelmente quando uma senha adequada foi adicionada o site, e então o browser, enviaram uma mensagem que permitia o retorno de um pacote adequado. Inclusive, a resposta do sistema dessa vez se dá através de um 200 OK.

4 CONCLUSÃO

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

5 REFERÊNCIA BIBLIOGRÁFICA

NIELSEN, HENRIK FRYSTYK et al. **Network performance effects of HTTP/1.1, CSS1, and PNG.** In: Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication. 1997.

POPA, LUCIAN; GHODSI, ALI; STOICA, ION. **HTTP as the Narrow Waist of the Future Internet.** In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. 2010.

MOGUL, JEFFREY C. et al. **Potential benefits of delta encoding and data compression for HTTP.** In: Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication. 1997.

MOGUL, JEFFREY C. **The case for persistent-connection HTTP.** ACM SIGCOMM Computer Communication Review, 1995.

BERNERS-LEE, TIM. **Hypertext transfer protocol: a stateless search, retrieve and manipulation protocol**, 1993.