# **CES-35 – Redes de Computadores e Internet**

## **Laboratório 1: Conhecendo protocolos - Wireshark**

**Nome:** Daniel Araujo Cavassani (COMP 25)

**Data:** 26/08/2024

### **1. Captura de pacotes**

* **Ação:** Iniciar captura no Wireshark após selecionar a interface de rede com acesso à internet.

### **2. Acesso ao site do Kurose**

* **Ação:** Acessar<http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html> no navegador e parar a captura.
* **Arquivo de captura salvo em formato pcapng:** Sim

### **3. Perguntas Gerais**

**3.A) Quais destes protocolos aparecem na lista de pacotes: TCP, QUIC, HTTP, DNS, UDP, TLS?**

* Resposta: TCP, QUIC, HTTP, DNS, UDP, TLS

**3.B) Quanto tempo transcorreu desde o envio do HTTP GET até o recebimento do HTTP OK?**

* Tempo decorrido: 126.654 ms

**3.C) Qual a utilidade dos campos User-Agent (HTTP GET) e Server (HTTP OK)?**

* Resposta:
  + O campo User-Agent no HTTP GET foi capturado como:

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36\r\n

Esse campo é utilizado para informar ao servidor detalhes sobre o navegador e o sistema operacional do cliente. No caso, ele indica que o cliente está utilizando o navegador Google Chrome, versão 128.0.0.0, no sistema operacional Windows 10 (64 bits). Além disso, o navegador usa o motor de renderização AppleWebKit, comum ao Safari. Essa informação é importante para que o servidor adapte a resposta ao cliente, garantindo compatibilidade e otimização da exibição do conteúdo de acordo com as especificações do navegador e sistema operacional.

* + Server: O campo Server na resposta HTTP OK foi capturado como:

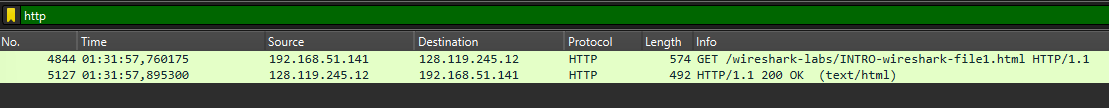
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod\_perl/2.0.11 Perl/v5.16.3\r\n

O campo Server informa ao cliente qual software está sendo utilizado no servidor. No caso, o servidor está rodando o Apache/2.4.6 em um sistema CentOS, com suporte a OpenSSL/1.0.2k-fips para criptografia segura, PHP/7.4.33 para execução de scripts PHP, e mod\_perl/2.0.11 para processar scripts Perl. Essas informações ajudam a entender a infraestrutura do servidor, o que pode ser útil para depuração e compatibilidade de sistemas.

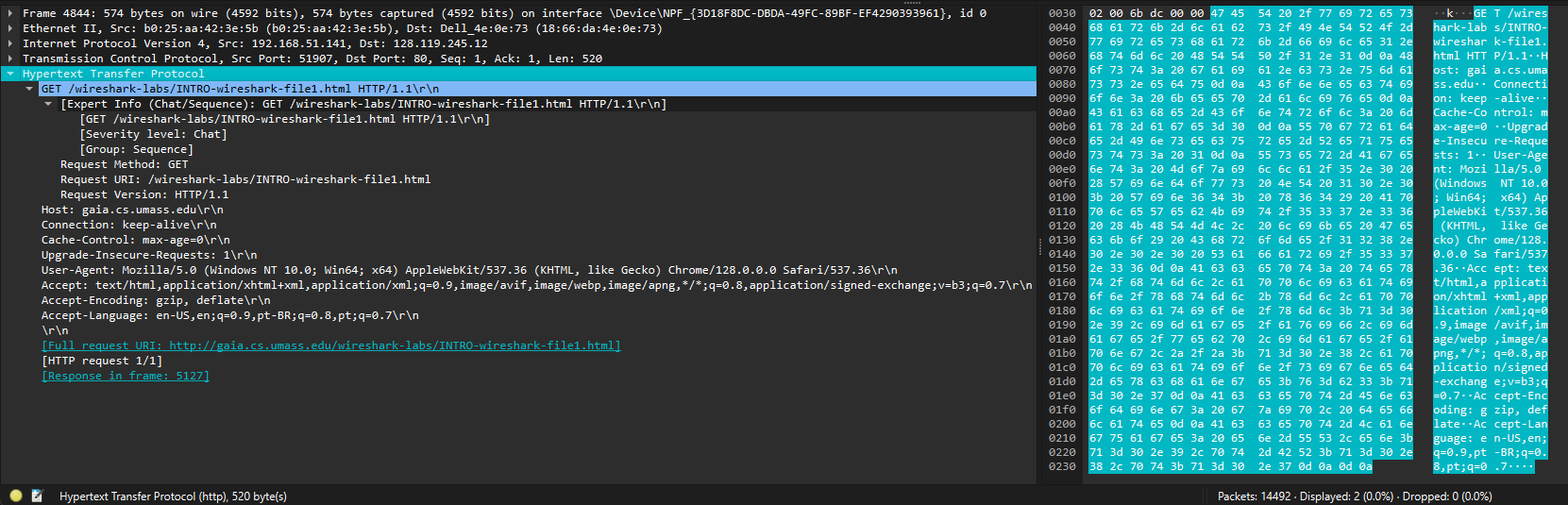
**3.D) Tamanho dos cabeçalhos e dados úteis da resposta HTTP OK:**

* Cabeçalho de Aplicação (HTTP): 357 bytes
* Cabeçalho de Transporte (TCP): 20 bytes
* Cabeçalho de Rede (IP): 20 bytes
* Cabeçalho de Enlace (Ethernet): 14 bytes
* Total de bytes dedicados aos cabeçalhos: 411 bytes
* Dados úteis: 81 bytes
* Porcentagem de dados úteis: 16.46%

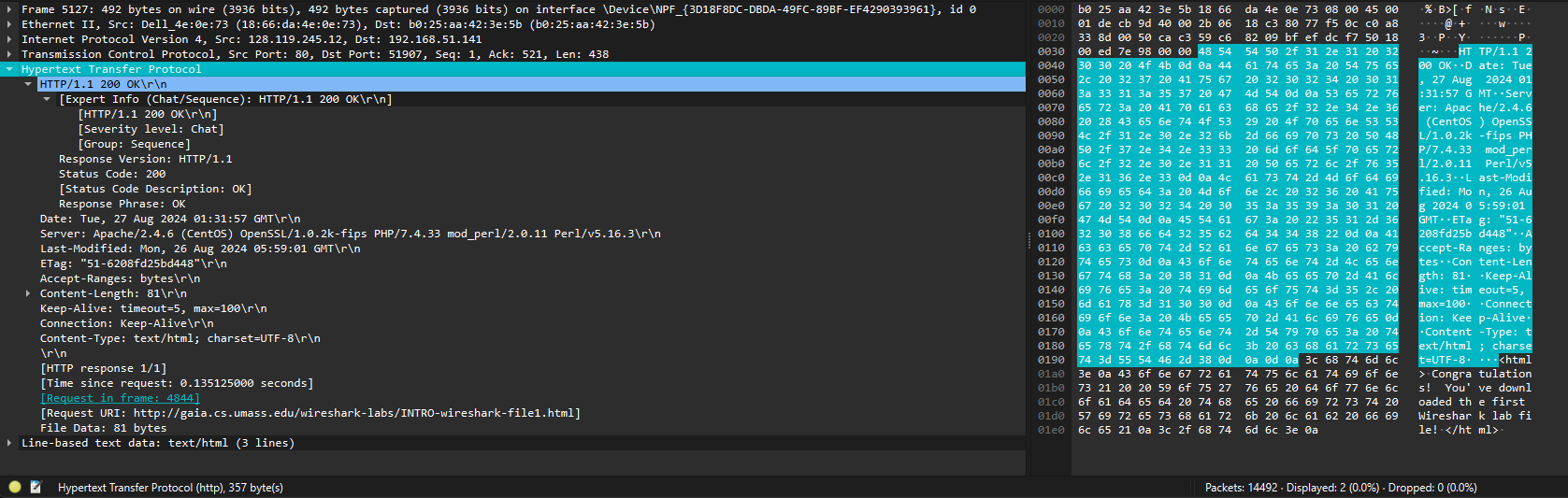
**3.E) Prints das mensagens HTTP GET e HTTP OK:**

****

* HTTP GET



* HTTP OK



**3.F) Explicação sobre a mensagem HTTP 1.1/304 Not Modified e solução:**

* Resposta:
  + Filtro aplicado: No campo de filtro do Wireshark, usei o seguinte filtro para capturar apenas as mensagens relacionadas ao IP do site do Kurose:

ip.addr == 128.119.245.12

* + Acesso à página: Após acessar novamente a URL http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html, recebemos um pacote de resposta com o código HTTP 1.1 304 Not Modified.
  + Explicação do código HTTP 304 Not Modified: O código 304 Not Modified indica que o recurso solicitado não foi modificado desde a última vez que o navegador fez a requisição. Isso ocorre porque o navegador enviou cabeçalhos de cache como If-Modified-Since ou If-None-Match, e o servidor constatou que a versão do conteúdo no cache do navegador ainda é válida. Como resultado, o servidor não transfere novamente os dados, economizando largura de banda.
  + Como evitar o HTTP 304 Not Modified: Para evitar receber a resposta 304 e forçar o servidor a enviar o conteúdo completo novamente, é possível:
    - Esvaziar o cache do navegador: Limpar o cache força o navegador a solicitar uma nova versão do recurso.
    - Utilizar o modo anônimo/privado: Nesse modo, o navegador não utiliza cache e sempre solicita o conteúdo completo.
    - Desativar temporariamente o cache: Nas ferramentas de desenvolvedor do navegador, é possível desativar o cache, garantindo que o servidor envie novamente os dados completos.

### **4. Camada de Transporte**

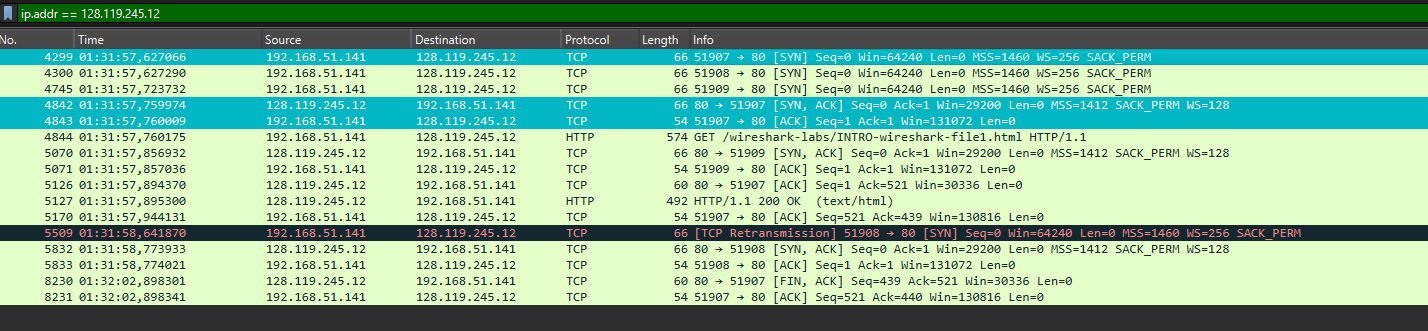
**4.A) Qual é o número da porta de destino e de origem para o HTTP GET?**

* Porta de origem: 51907
* Porta de destino: 80

**4.B) Campos da camada TCP encontrados no pacote:**

* Resposta:
  + Source Port (Porta de Origem): 51907
  + Destination Port (Porta de Destino): 80
  + TCP Segment Length: 520 bytes
  + Sequence Number (Número de Sequência): 1 (relative sequence number)
  + Acknowledgment Number (Número de Confirmação): 1 (relative acknowledgment number)
  + Header Length (Tamanho do Cabeçalho): 20 bytes (5 x 4 = 20 bytes)
  + Flags:
  + PSH (Push Flag)
  + ACK (Acknowledgment Flag)
  + Window Size (Tamanho da Janela): 512
  + Window Size Scaling Factor: 256
  + Checksum: 0xbdc6 (unverified)
  + Urgent Pointer: 0 (não utilizado)
  + TCP Payload: 520 bytes

**4.C) Flags no 3-way handshake:**

****

**Primeiro pacote (SYN)**:

* **Flag(s) de controle ligado(s)**: SYN
* **Pacote**: 4299
* **Descrição**: O primeiro pacote do handshake é enviado pelo cliente (192.168.51.141) para o servidor (128.119.245.12) com a flag **SYN** ativada, iniciando a conexão TCP.

**Segundo pacote (SYN, ACK)**:

* **Flag(s) de controle ligado(s)**: SYN, ACK
* **Pacote**: 4842
* **Descrição**: O segundo pacote é enviado pelo servidor (128.119.245.12) para o cliente (192.168.51.141) com as flags **SYN** e **ACK** ativadas, confirmando o recebimento do SYN do cliente e enviando o próprio SYN do servidor.

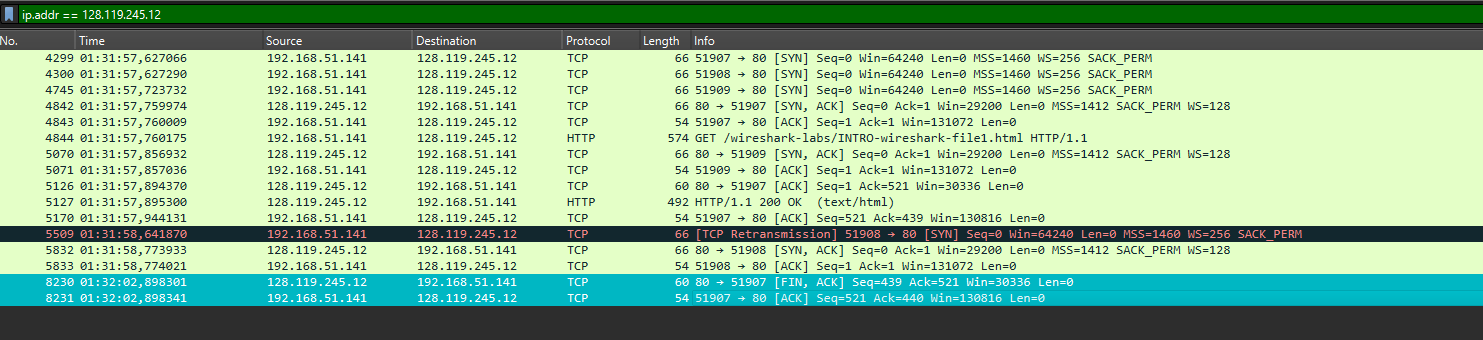
**Terceiro pacote (ACK)**:

* **Flag(s) de controle ligado(s)**: ACK
* **Pacote**: 4843
* **Descrição**: O terceiro pacote é enviado pelo cliente (192.168.51.141) para o servidor (128.119.245.12) com a flag **ACK** ativada, confirmando o recebimento do SYN do servidor e finalizando o handshake.

**4.D) Portas envolvidas no HTTP OK:**

* Porta de origem: 80
* Porta de destino: 51907

**4.E) Desconexão após transferência da página:**

****

### **Pacote nº 8230:**

### **Instante de tempo:** 01:32:02,898301

### **Fonte:** 192.168.51.141 (porta 51907)

### **Destino:** 128.119.245.12 (porta 80)

### **Flags:** [FIN, ACK]

### **Descrição:** O cliente sinaliza que deseja encerrar a conexão com o servidor enviando o pacote com as flags FIN e ACK ativadas.

### **Pacote nº 8231:**

### **Instante de tempo:** 01:32:02,898341

### **Fonte:** 128.119.245.12 (porta 80)

### **Destino:** 192.168.51.141 (porta 51907)

### **Flags:** [ACK]

### **Descrição:** O servidor responde ao pacote de FIN, ACK enviado pelo cliente com um pacote contendo apenas a flag ACK, confirmando o fechamento da conexão.

### No contexto da desconexão TCP, o pacote 8230 é o que contém a flag FIN, iniciando o fechamento da conexão na porta 51907.

### Em seguida, o servidor responde com um ACK (pacote 8231), encerrando a conexão conforme o protocolo TCP.

### Portanto, a desconexão aconteceu na porta 51907, envolvendo os pacotes 8230 e 8231.

### 

### **5. Endereços de rede**



**6. Camada de Rede**

**6.A) Endereços IP do gaia.cs.umass.edu e de seu computador:**

* IP do gaia.cs.umass.edu: 128.119.245.12
* IP de seu computador: 192.168.51.141

**6.B) O campo inet no ifconfig/ipconfig corresponde ao IP mostrado pelo Wireshark?**

* Resposta: Sim

**6.C) Campos da camada IP encontrados no pacote:**

* Resposta:
  + **Version**: 4
  + **Header Length**: 20 bytes
  + **Differentiated Services Field:** 0x00 (DSCP: CS0, ECN: Not-ECT)
  + **Total Length**: 560
  + **Identification**: 0x8dda (36314)
  + **Flags**: 0x2 (Don't fragment)
  + **Fragment Offset**: 0
  + **Time to Live (TTL)**: 128
  + **Protocol**: TCP (6)
  + **Header Checksum**: 0x0000 (validation disabled)
  + **Header Checksum Status**: Unverified
  + **Source Address**: 192.168.51.141
  + **Destination Address**: 128.119.245.12

### **7. Camada de Enlace**

**7.A) Endereços MAC de origem e destino:**

* **Endereço MAC de origem**: **b0:25:aa:42:3e:5b**
* **Endereço MAC de destino**: **18:66:da:4e:0e:73**

Esses são os endereços de origem e destino observados na camada Ethernet II. O endereço de origem é o da nossa placa de rede, e o endereço de destino é o próximo salto (que pode ser o roteador).

**7.B) Campos da camada MAC encontrados no pacote:**

* **Destination MAC Address**: 18:66:da:4e:0e:73
* **Source MAC Address**: b0:25:aa:42:3e:5b
* **Type**: IPv4 (0x0800)

Esses são os principais campos encontrados na camada de enlace (Ethernet) para o pacote HTTP GET.

**7.C) O endereço ether no ifconfig corresponde ao endereço de origem mostrado pelo Wireshark?**

O Endereço MAC de origem mostrado no Wireshark é b0:25:aa:42:3e:5b.

No resultado do comando ipconfig /all, o Endereço Físico (ou Physical Address) da sua interface Ethernet também é b0:25:aa:42:3e:5b.

Portanto, sim, o campo ether mostrado no ipconfig corresponde ao endereço de origem mostrado no Wireshark. Isso confirma que o seu sistema operacional está usando corretamente o endereço de sua placa de rede para montar os pacotes que emite para a rede.

### **8. Traceroute**

**8.A) Quantos saltos foram necessários até chegar ao site?**

* O traceroute (tracert no windows) mostra que foram necessários 25 saltos para chegar ao servidor gaia.cs.umass.edu (128.119.245.12).

**8.B) Há algum salto com tempo menor que o anterior?**

Sim, há passos em que o valor de tempo é menor que o anterior, por exemplo:

* **Salto 9 (114 ms)** para **Salto 8 (124 ms)**.
* **Salto 18 (126 ms)** para **Salto 17 (130 ms)**.

Isso pode acontecer por várias razões:

* **Rotas Assíncronas**: Os pacotes podem seguir rotas ligeiramente diferentes entre os saltos, resultando em tempos de resposta diferentes.
* **Congestionamento Temporário**: O congestionamento de rede em um salto pode causar um tempo de resposta maior, mas o congestionamento pode diminuir em saltos posteriores.
* **Otimização de Roteadores**: Alguns roteadores podem ser mais rápidos ao processar pacotes devido a configurações de cache, resultando em tempos menores nos saltos subsequentes.

**8.C) Saída do traceroute:**

