WorkStream: Desenvolvimento de Aplicação Cliente-Servidor - Projeto 1

Carlos Cauã Rocha da Silva - 231034304, José Neto Souza de Lima - 231018955,
Luca Heringer Megiorin - 231003390

***Resumo*—Para o primeiro projeto da disciplina de Redes de Computadores, foi desenvolvida uma aplicação da arquitetura Cliente-Servidor, utilizando em JavaScript. Neste trabalho foi promovida uma melhor compreensão da camada de aplicação e os protocolos de transmissão de dados, neste caso, o TCP.**

***Palavras-Chave*—aplicação, cliente, servidor, TCP**

I. Introdução

PROTOCOLOS são essenciais para o funcionamento da Internet e, consequentemente, para a transmissão de informações entre diferentes dispositivos. A partir disso, foi proposta a criação de uma nova aplicação que utilizasse os conceitos de sockets e protocolos para a emissão de pacotes, seguindo a estrutura cliente-servidor. Neste artigo, serão discutidas as formas em que o grupo projetou o novo serviço na rede, que foi nomeado de WorkStream e foi desenvolvido em JavaScript, utilizando o protocolo HTTP e TCP. O aplicativo trata-se de um grupo de conversas em que é possível estabelecer tarefas, as quais fornecem aos integrantes do grupo pontos de experiência (o conceito do serviço será melhor abordado na seção III deste artigo).

O artigo está dividido nas seguintes seções: Introdução I, que contextualiza o projeto, Fundamentação Téorica II, a qual descreve os conceitos utilizados para a realização do trabalho, a linguagem escolhida e uma breve descrição da interface por meio de imagens, Ambiente Experimental e Análise de Resultados III, que descreve a estrutura do serviço e as atividades por ele executadas, Conclusões IV, que apresentam os pensamentos finais quanto ao projeto, e Bibliografia 1, para as referências usadas no relatório. Após as referências consta o código fonte e o link para o pitch técnico da aplicação no Apêndice A.

II. Fundamentação Téorica

Para a criação do aplicativo, o grupo optou por usar os protocolos HTTP e TCP. A título de facilitar o desenvolvimento web, visto que a aplicação feita é acessada pelo browser, foi utilizada a linguagem JavaScript, aliada ao Node.js, para executar o código, e a biblioteca React. Nesta seção serão descritas as decisões quanto à linguagem (II-A), aos protocolos (II-B), mostrará partes da interface (II-C).

A. Linguagem e Bibliotecas

O serviço foi programado usando a versão Node 18 para executar o ambiente JavaScript. Apesar de facilitar na parte do desenvolvimento web, tal linguagem de programação abstrai os conceitos da criação de sockets e definição de suas portas, envolvendo-os na função fetch. Todavia, como será visto na seção de Ambiente Experimental e Análise de Resultados III, foi possível identificar todo esse processo de encapsulamento e envio de pacotes, então a escolha da linguagem teve um resultado positivo para o grupo.

Além disso, o ambiente Node.js permitiu o uso de bibliotecas que auxiliassem a construção da interface gráfica, como o React. Os arquivos com a extensão JSX (JavaScript XML), como visto na documentação oficial da biblioteca React [1], são uma extensão da sintaxe do JavaScript que integra a possibilidade da escrita semelhante a HTML em arquivos JavaScript, o que permite uma maior dinamicidade às páginas e facilita o desenvolvimento, visto que a lógica e o design dos elementos ficam no mesmo arquivo. O React também reduz a quantidade de pedidos que o cliente deve fazer ao servidor para a renderização de páginas: uma vez que o usuário entra em qualquer página, ele recebe o arquivo raiz JSX, que contém a topologia das páginas web, faz os pedidos dos outros componentes JSX e, a partir de então, qualquer outro pedido para a interface se resume a pedidos de arquivos gráficos, como imagens, mesmo se o usuário navegar para outra página do serviço.

Quanto ao acesso e armazenamento de informações, foi utilizada a biblioteca MySQL para a criação de um banco de dados, o que permitia ações como login e registro de usuários, guardar características do usuário e dos grupos de conversa e tarefas.

B. Protocolos Utilizados

Dado que o serviço é relativamente simples e não necessitava de uma velocidade ou de técnicas diferenciadas, ele se aproveita dos famosos protocolos HTTP, mais especificamente o HTTP/1.1, e TCP. Estes protocolos são usados por padrão pela função fetch do Node.js, o que será demonstrado na seção do Ambiente Experimental e Análise de Resultados III.

O protocolo TCP é fundamental para o funcionamento da internet, fazendo parte da camada de transporte. Com a sua versão mais atual publicada em agosto de 2022 na RFC 9293 [2], o protocolo é considerado confiável, ou seja, garante que as mensagens enviadas chegarão sem erros, sem perda e em ordem. Para atingir essa confiabilidade, o Protocolo de Controle de Transmissão (*Transmission Control Protocol* - TCP) tem como base uma máquina de estados que checa a integridade dos componentes e a ordem em que chegam a partir de números de sequência [3]. Após ser estabelecida

uma conexão entre o cliente e o servidor, o cabeçalho conterá não só os números de sequência, mas também os números de confirmação (*acknowledgement number*), que indicam qual é o número de sequência que o emissor espera receber da outra parte, permitindo-o identificar se o pacote enviado foi de fato recebido: a resposta de confirmação, ACK, deve conter o número de confirmação da mensagem recebida como o seu número de sequência. Quando um pacote é transmitido por meio do TCP, este mantém uma cópia do pacote em uma fila de retransmissão e um cronômetro será iniciado de forma que, caso haja uma confirmação antes de acabar o tempo do cronômetro (ou seja, acontecer um *timeout*), a cópia é deletada da fila, e, caso contrário, é reenviada - isso garante que o pacote sempre chegará ao seu destino.

Como visto no capítulo 2 do livro *Computer Networking: A Top-Down Approach*, 8ª edição, dos autores Jim Kurose e Keith Ross Pearson [4], o TCP é um protocolo orientado a conexão, isto é, para haver a troca de informações é feito um aperto de mão em três vias (chamado de *three way handshake*) entre as partes, que é representado pela troca de mensagens do tipo SYN (pedido de conexão do cliente para o servidor), SYN+ACK (reconhecimento do servidor de que chegou o pedido e confirmação da conexão) e, por fim, ACK (confirmação do cliente de que a conexão foi estabelecida). Isto garante que, de fato, as partes se conectaram e que os seus números de sequência estão sincronizados e, a partir de então, pode haver o envio dos pacotes, como descrito anteriormente. Vale lembrar que, ao estabelecer uma conexão, as portas dos clientes visam ser únicas, de modo que, ao chegar na porta do servidor, este poderá desmultiplexar a informação para o socket correto, isto é, cada cliente terá a sua interação contida em um socket diferente para garantir que pacotes não serão misturados. Ao finalizar a transação, há mais uma troca de pacotes indicando que não há mais dados a serem enviados (representado por FIN) e a conexão é encerrada.

A TABELA I abaixo mostra a estrutura do cabeçalho da camada de transporte presente na RFC 793 [3] criada para a primeria documentação do protocolo TCP. Nem todos os seus componentes serão abordados neste artigo, mas vale notar a presença do *checksum*, que serve para a checagem básica da corrupção de algum arquivo e impede o retorno de uma confirmação (ACK) caso esteja corrompido, acatando no processo de reenvio do pacote no caso de um *timeout*, como descrito previamente.

TABELA I
TABELA DA ESTRUTURA DO CABEÇALHO TCP

< ——— 32 bits ——— >			
Porta de Origem			Porta de Destino
Número de Sequência			
Número de Confirmação (Acknowledgment Number)			
Offset de Dados	Reservado para uso futuro	Flags: URG, ACK, PSH RST, SYN, FIN	Janela
Checksum			Ponteiro de Urgência
Opções (tamanho variável)			
Dados (não compõem o cabeçalho)			

Quanto ao protocolo da camada de aplicação, o HTTP/1.1 (com a versão mais recente sendo descrita na RFC 9110 [5], de junho de 2022) é um protocolo que não guarda estado (isto é, o servidor não retém informações sobre os pedidos anteriores do cliente) que é responsável pela troca de mensagens em hipertexto. O Protocolo de Transferência e Hipertexto (*Hypertext Transfer Protocol* - HTTP) corresponde à troca de mensagens no formato ASCII, ou seja, legíveis para humanos, utilizando o protocolo TCP para fazê-la. De forma mais específica, o HTTP/1.1 é considerado persistente por manter o socket TCP aberto até que seja indicado que deve ser fechado ou o tempo da conexão acabe (*timeout*). Isso tudo tornou o protocolo mais robusto, melhorando a escalabilidade de projetos, dado que para o caso em que a conexão não foi terminada, o tempo para receber uma informação não é prejudicado pelo tempo de estabelecer uma nova conexão.

Há várias informações que se pode descrever no cabeçalho de mensagens e respostas, como o *Keep-Alive*, o qual determina o tempo em que a conexão deve permanecer aberta e, no caso de *timeout*, fechá-la: isso é o padrão da função fetch. As mensagens de pedido e resposta têm diferentes identificadores. Dentre os métodos de pedido, o serviço implementado usou o método GET, responsável por pedir por informações, e o método POST, responsável por enviar informações recebidas do usuário. Para a resposta, a mensagem é acompanhada de um código de status, que indica o que ocorreu quando o pedido foi enviado, podendo dizer que tudo foi bem (representado pelo código 200, OK) ou descrevendo qual foi o erro (como o código 400, Bad Request).

C. Descrição da Interface

Quanto à interface, a aplicação focou principalmente em navegadores de computadores por questão de tempo. Apesar disso, é possível acessar o serviço web pelo navegador de celulares e interagir normalmente, mas, em razão de não ter sido projetado com celulares em mente, alguns elementos podem aparecer distorcidos ou cortados e, consequentemente, há modelos de celular que não conseguem interagir com todos eles. Nas imagens a seguir, é possível ver a interface para a tela de login (Figura 1), a qual permite que o usuário escreva seus credenciais para entrar no serviço, tela de registro (Figura 2), que permite que o usuário cadastre seus credenciais no serviço, e a tela inicial após o login (Figura 3), que aparece enquanto o usuário estiver com os seus dados registrados no serviço e não tiver selecionado um grupo de conversas.

Essas interfaces foram descritas usando React, como visto em II-A, de forma que componentes mais complexos, como a região de input de usuário, eram implementados em arquivos JSX diferente, permitindo a reutilização destes de uma forma simples. Vale notar que a tela de login e registro estão descritas no mesmo arquivo JSX que, por meio de arquivos CSS, controla a animação e aparição dos elementos relevantes para cada parte do serviço (mostrando a dinamicidade descrita em II-A). Na seção III serão descritas outras partes da interface conforme a explicação do ambiente e dos resultados obtidos.

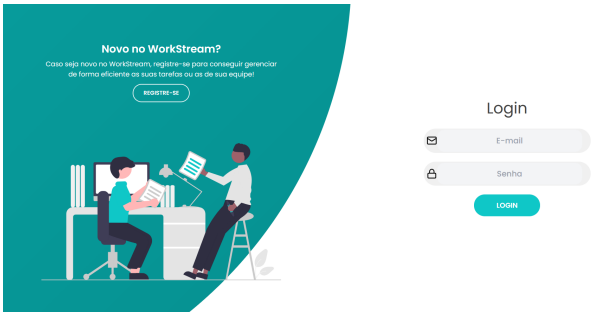

Figura 1: Interface da tela de login

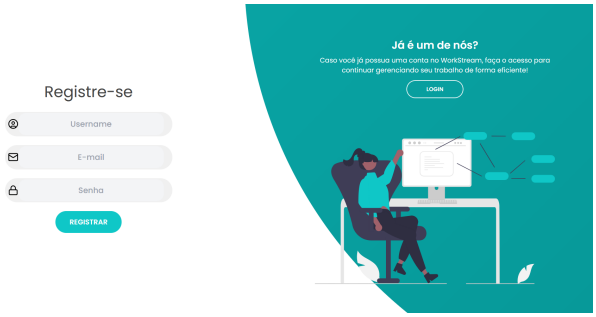

Figura 2: Interface da tela de registro

Figura 3: Interface da tela inicial após login

III. Ambiente Experimental e Análise de Resultados

A partir do embasamento teórico descrito na seção II, foi construída a aplicação. O WorkStream, nome dado ao serviço, se trata de um ambiente de conversa que pode ser utilizado por grupos de trabalho, turmas de uma escola ou até para grupos de amigos. Ao criar em um grupo, o administrador pode enviar tarefas aos usuários que estiverem cadastrados no grupo na hora da criação dela. Essas tarefas têm como recompensa pontos de experiência, também determinados pelo administrador, de forma que, ao cumprir uma tarefa, o usuário poderá subir de nível ao atingir o número de experiência necessário e ver a sua posição em relação aos outros componentes do grupo. Com isso, o WorkStream espera motivar o cumprimento de tarefas no ambiente escolar ou de trabalho de forma mais divertida, ou até proporcionar eventos de competições em um grupo de amigos. Nesta seção, será abordado quanto o hardware e software utilizados (III-A), a rede e elementos básicos do cliente e servidor (III-B), uma explicação resumida dos serviços (III-C) e um exemplo de um dos serviços (III-D).

A. Hardware e Software

Quanto ao hardware utilizado, o aplicativo foi implementado usando windows, então, como os testes foram feitos apenas nesse sistema operacional, não é possível garantir o funcionamento do servidor em outros sistemas operacionais. Todavia, por ser um serviço web para navegadores, clientes de qualquer tipo de dispositivo e sistema operacional, havendo apenas alguns casos em que o navegador do celular não renderiza os elementos de forma correta, dificultando o uso, como descrito na seção II.

Para o desenvolvimento do software, o grupo optou por usar mais uma ferramenta: o docker. Como dito em sua documentação oficial [6], o intuito do docker é reduzir o tempo de organização de máquinas e bibliotecas para priorizar na escrita do código, permitindo também que qualquer máquina que possua o aplicativo Docker Desktop [7] rode o programa (garantido se for do mesmo sistema operacional, visto que, mais uma vez, não foi testado em outros sitemas que não fossem o windows), facilitando o processo de produção e exportação.

B. Rede e Estrutura Básica

Com o hardware, software (III-A) e linguagem (II-A) definidos, a estrutura do serviço cliente-servidor foi separada em frontend (na porta 3000, responsável por tratar os pedidos relacionados à interface do cliente) e backend (na porta 5000, responsável por tratar os pedidos relacionados a operações com o servidor). Além disso, no backend foi colocado o banco de dados usando a biblioteca mysql, de forma que o servidor consegue sempre acessar suas informações por meio de *queries*.

No caso do servidor, ao executar o código, este será aberto apenas para a rede local, de forma que qualquer um conectado à rede é capaz de acessar como cliente a este servidor, usando o endereço IPv4 da máquina correspondente ao servidor e a porta 3000 (frontend): o link estaria na forma http://ipaddress:3000, em que `ipaddress` é o endereço IPv4 do servidor. A própria máquina do servidor é capaz de acessá-lo por um navegador como cliente usando o link descrito acima ou pelo endereço loopback: http://localhost:3000.

C. Serviços Oferecidos

A partir do banco de dados, é possível cadastrar, verificar a existência do usuário e entrar no serviço com as interfaces vistas na Figura 1 e na Figura 2. Por meio da criação de sessões presentes no JavaScrip, é criado um cookie que guarda o ID do usuário, o que permitirá a identificação dele para realizar qualquer ação na plataforma. O login e registro de usuários usam a mesma lógica, em que o cliente escreve as suas informações nos campos presentes na interface e, ao clicar no botão de login/registrar, é enviada uma mensagem HTTP com método POST (explicado em II-B), e o cliente recebe uma resposta 200 OK no caso do sucesso em cadastrar (para o registro) ou da existência e corretude de seus dados (para o login) e, neste último caso, ele é redirecionado para a página principal (Figura 3)

Após ser redirecionado para a página inicial, as únicas requisições que o cliente faz ao frontend são para receber as imagens necessárias para renderizar a interface do menu, visto que, por usar React, as informações da topologia dos elementos já estão presentes nos arquivos JSX que são recebidos ao abrir qualquer página do serviço (como visto na seção II-A). Dessa forma, o usuário pode abrir no canto esquerdo um menu que mostrará as informações do seu nome, nível, barra de experiência e grupos (Figura 4), as quais são obtidas por pedidos GET ao backend.

Figura 4: Interface do menu

Ainda na página inicial, o usuário pode criar ou entrar em um grupo. No caso da criação do grupo, ele fornecerá o nome, senha e uma breve descrição do grupo (Figura 5), e no caso de entrar em um grupo, precisará apenas de fornecer o nome do grupo e a senha (Figura 6).

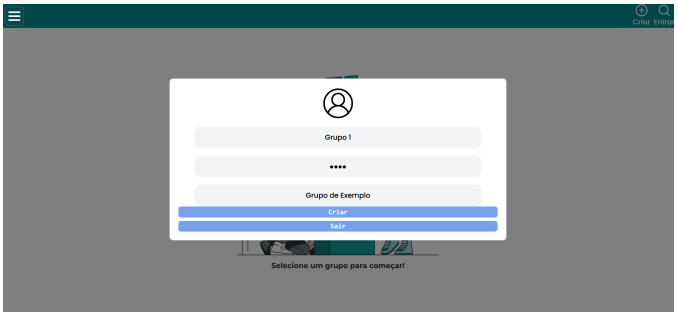

Figura 5: Criação do Grupo 1, com senha 1234 e descrição "Grupo de Exemplo"

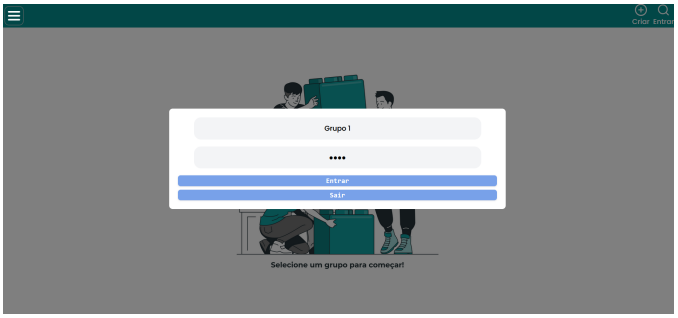

Figura 6: Usuário entrando no Grupo 1, com senha 1234

Ambos os processos irão enviar uma mensagem HTTP usando o método POST (como visto na seção II-B) e podem receber um 200 OK no caso de sucesso, ou outras mensagens de erro, no caso do grupo já existir ou de já estar em um grupo. O resultado é visto na Figura 7, em que, ao clicar no menu, um GET já foi feito para ver de quais grupos o usuário participa e a lista de grupos é atualizada de forma dinâmica, graças ao React.

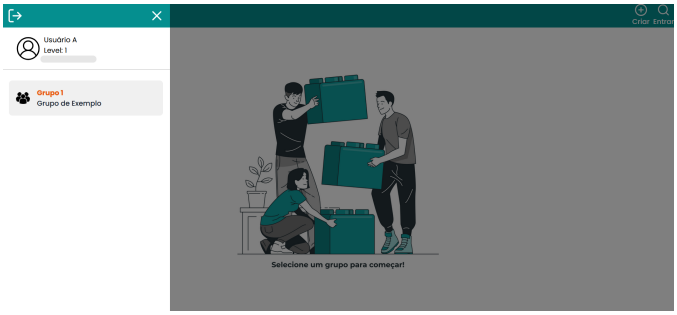

Figura 7: Menu com lista de grupos atualizada

A partir disso, é possível selecionar um dos grupos e a tela será atualizada para mostrar as conversas, participantes e tarefas – a funcionalidade do grupo de conversas será descrita no Exemplo de Serviço (III-D). Na aba de tarefas, o administrador poderá criar tarefas com uma recompensa (Figura 8), de forma que, com um método POST, a tarefa será guardada no banco de dados caso a resposta seja 200 OK. Qualquer outro usuário que estiver no grupo, ao recarregar a página, receberá a tarefa por meio de um GET, e poderá marcá-la como feita ao clicar nela (Figura 9), enviando uma mensagem HTTP POST que, ao retornar 200 OK atualiza o nível de experiência do usuário.

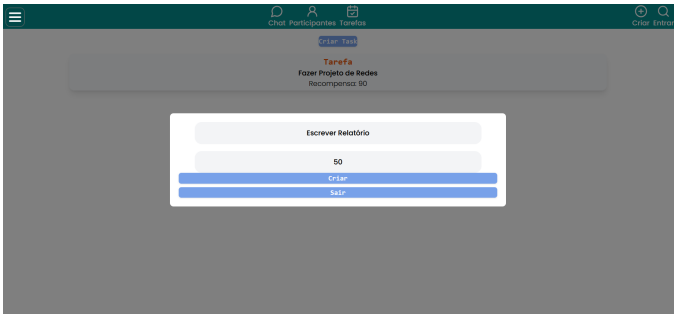

Figura 8: Criação da tarefa "Escrever Relatório", com recompensa de 50 pontos de experiência

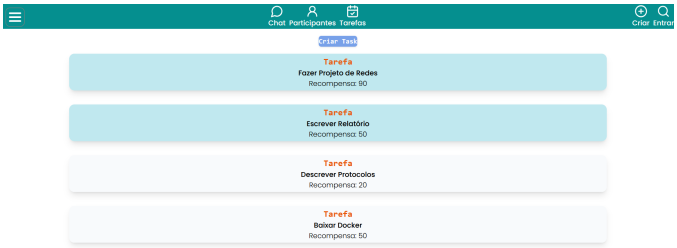

Figura 9: Aba de tarefas do administrador e as tarefas que ele já realizou (em azul) e está para realizar (branco)

Quanto à aba dos participantes, é feita uma requisição GET para o servidor, que retorna uma lista com os usuários pertencentes ao grupo na ordem do nível de cada caso a resposta seja 200 OK, podendo ser visto na Figura 10

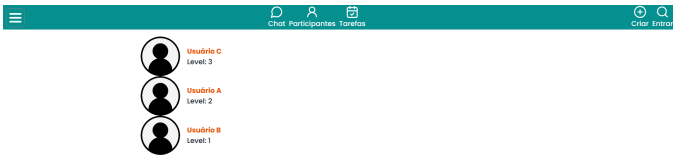

Figura 10: Aba dos participantes, com ranking de seus níveis

Por fim, na aba do menu (Figura 4 ou Figura 7), o usuário pode encerrar a sua sessão clicando no botão da esquerda. Com isso, é enviada uma mensagem POST que deleta a sessão do usuário (perdendo, assim, o ID do usuário) e o redireciona para a página do login no caso de uma resposta 200 OK. Vale notar que a página do menu principal (qualquer uma das figuras a partir da Figura 3 até a Figura 9) frequentemente realiza o pedido GET para checar se ainda existe uma sessão e, no caso negativo, redireciona o usuário à pagina de login.

Estes são os serviços do WorkStream de uma forma resumida. Todos eles usam o protocolo TCP e HTTP descritos em II-B e acessam o banco de dados para obter as informações necessárias, como mencionado em III-B e II-A.

D. Exemplo de Serviço: Grupo de Conversa

O serviço do grupo de conversa foi escolhido para ser descrito de forma mais detalhada. Para isso, foi utilizado o aplicativo Wireshark [8], que se trata de um farejador de pacotes, ou seja, é capaz de capturar os pacotes transmitidos e recebidos por uma máquina, para a melhor compreensão do funcionamento do aplicativo desenvolvido. Foi realizada a conexão entre um celular (cliente) e um computador (servidor) e os resultados obtidos estão descritos conforme o requerido no Quadro 1 das orientações do arquivo `projeto1-20242` fornecido pela professora Priscila Solis da disciplina de Redes de Computadores. A interface da aba de conversas é a seguinte:

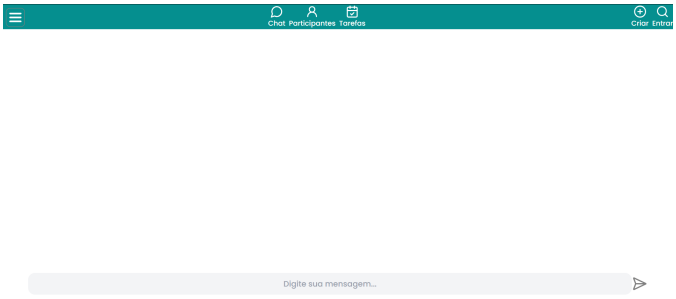

Figura 11: Interface do grupo de conversa

1) Identificação da versão ou tipo da aplicação no servidor que está executando: Ao escrever um texto para ser enviado na aba de mensagens, clicar para enviar, uma mensagem HTTP é gerada com o método POST e rota `ipaddress:5000/send-message`. Pelo Wireshark, foi identificada a mensagem HTTP, cujo conteúdo está descrito na Figura 12. Nessa figura, é possível verificar que a a aplicação que o servidor está executando se trata de um pedido HTTP/1.1 (pela linha `POST /groups/send-message HTTP/1.1\r\n`, é visível a versão do HTTP sendo utilizada)

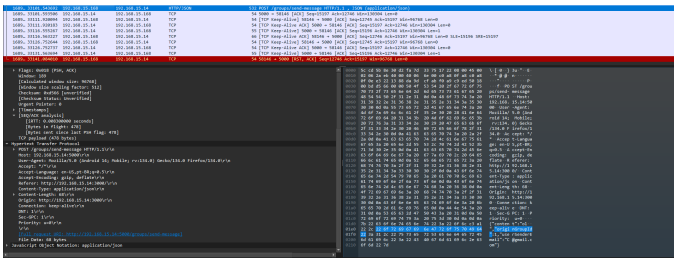

Figura 12: Captura de tela obtida ao enviar um texto no grupo de conversa: a mensagem HTTP está destacada no topo da imagem

2) Endereço IP dos clientes e do servidor: Ainda na Figura 12 é possível identificar que o endereço IP do servidor é 192.168.15.14 (visto na linha `Host: 192.168.15.14:5000\r\n`) e o endereço IP do cliente é 192.168.15.14 (visto na linha `Origin: http://192.168.15.14:3000\r\n`).

3) Tipo de Protocolo de Transporte: O protocolo de transporte usado pelo cliente e pelo servidor é o protocolo TCP, como é visto na Figura 13 abaixo e conforme as explicações em II-B.

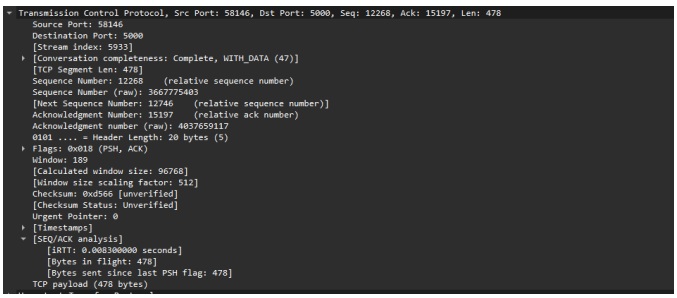

Figura 13: Captura de tela obtida ao enviar um texto no grupo de conversa: parte TCP

4) Porta de Destino do e de Origem do Cliente: Ainda na Figura 13 é identificada a porta de origem do cliente como 58146 (como visto em `Source Port: 58146`) e a porta de destino do cliente como 5000 (como visto em `Destination Port: 5000`). Isso corrobora com o que foi dito na parte teórica (II-B) quanto às portas e sockets TCP: a porta gerada para o cliente é única para ele, enquanto que a pertecente ao servidor é fixa (5000) e, quando outros clientes se conectarem, haverá um processo de desmultiplexação para sockets separados para cada porta de origem.

5) Carga útil do pacote: Por ser um pacote de mensagem pequeno, não houve necessidade de divi-lo em chunks, ou seja, o pacote mostrado na Figura 12 está completo. Observando esta figura (12) e confirmando na primeira linha da Figura 14, nota-se que a carga útil do pacote é de 532 bytes (visível em `Frame 1689298: 532 bytes on wire (4256 bits), 532 bytes captured (4256 bits) on interface \Device\NPF_{0FEC24F7-2B61-4901-A1FA-B9B891E21786}, id 0`). Checando mais abaixo na Figura 14, é visto que a mensagem de texto está sendo passada corretamente: o usuário, cujo email é C@email.com enviou um texto "olá"no grupo de ID 1. De fato, ao enviar a mensagem HTTP/1.1 com o método POST, foi criado um json com as informações `{content: userMessage, originGroupId: window.groupIdentificator, userSenderEmail: window.useremaillogado,}`, em que, neste caso, userMessage = "olá", originGroupId = 1 e userSenderEmail = C@email.com.

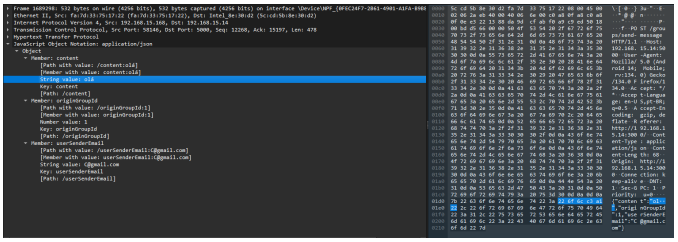

Figura 14: Captura de tela obtida ao enviar um texto no grupo de conversa: parte conteúdo da mensagem, destaque no texto enviado, "óla"

6) Encapsulamento: Por fim, é possível ver os cabeçalhos presentes no pacote HTTP: após a mensagem HTTP (Figura 12), há o cabeçalho da camada de transporte TCP (`Src Port: 58146, Dst Port: 5000, Seq: 12268, Ack: 15197, Len: 478`, destacado na Figura 15, em que é possível reconhecer outros valores descritos na TABELA I), da camada de rede (`Src: 192.168.15.168, Dst: 192.168.15.14`, destacado na Figura 16) e o da camada de enlace (`Src: fa:7d:33:75:17:22 (fa:7d:33:75:17:22), Dst: Intel_8e:30:d2 (5c:cd:5b:8e:30:d2)`, destacado na Figura 17). Como visto em aulas da disciplina e presente no capítulo 1 do livro do Kurose [4], há camadas na internet, cada uma com o seu protocolo: aplicação, transporte, rede, enlace e física. Cada uma dessas camadas é responsável por encapsular ou desencapsular as informações que recebem conforme a sua função, ou seja, a camada de transporte irá receber a mensagem da camada de aplicação e adicionar o seu cabeçalho, e irá receber o segmento da camada de redes e retirar seu cabeçalho após realizar os processos necessários, como a desmultiplexação, enviando a mensagem para a camada de aplicação. Da mesma forma a camada de rede e de enlace fazem o seu encapsulamento/desencapsulamento das informações recebidas da camada superior/inferior. Isso pode ser visto de forma didática na imagem presente no livro do Kurose [4], que é a Figura 18.

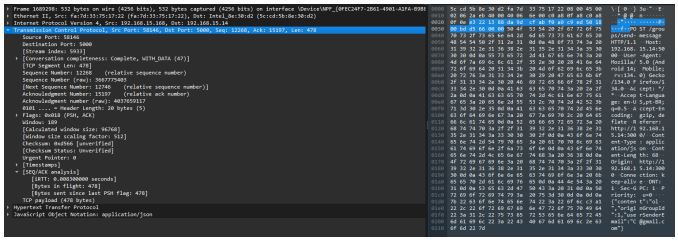

Figura 15: Captura de tela obtida ao enviar um texto no grupo de conversa: cabeçalho da camada de transporte TCP

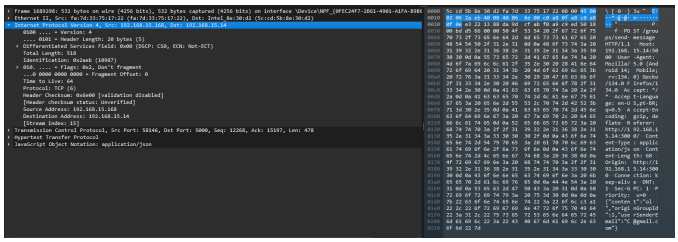

Figura 16: Captura de tela obtida ao enviar um texto no grupo de conversa: cabeçalho da camada de redes Internet Protocol version 4

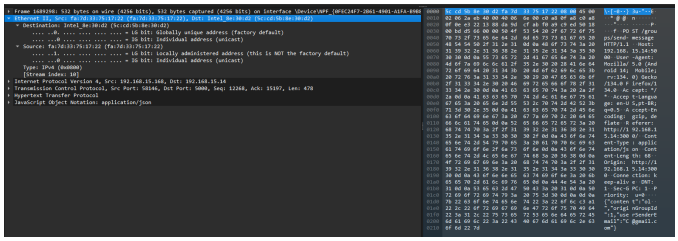

Figura 17: Captura de tela obtida ao enviar um texto no grupo de conversa: cabeçalho da camada de enlace Ethernet II

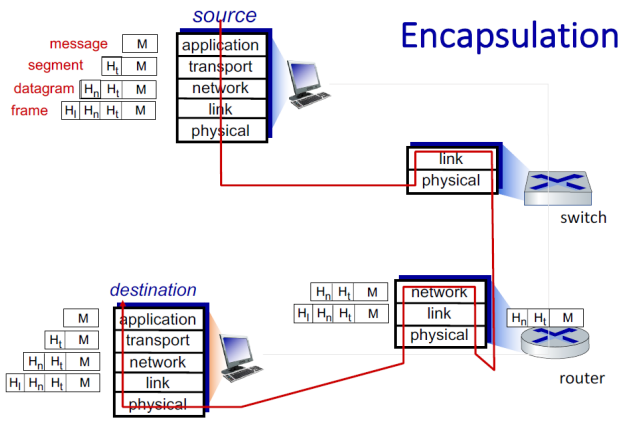

Figura 18: Representação do processo de encapsulamento e desencapsulamento de uma mensagem, presente no Capítulo 1 do livro *Computer Networking: A Top-Down Approach*, 8ª edição, dos autores Jim Kurose e Keith Ross Pearson [4]

Vale notar ainda sobre o envio de mensagens que a Figura 12 não mostra a abertura nem o encerramento do protocolo TCP, uma vez que, enquanto na aba da conversa dos grupos, um socket TCP fica aberto frequentemente realizando pedidos GET para obter mensagens recentes (é desse modo que os outros usuários receberão a mensagem enviada no procedimento descrito acima). O processo de conexão e pedido GET pode ser visto na Figura 19. Além disso, é visível logo abaixo do pacote HTTP na mesma Figura 12 que há um pacote TCP confirmando a chegada (ACK) do pacote HTTP acima dele, como previsto na seção II-B. Dessa forma, a interface de mensagens será atualizada e ficará como na Figura 20 abaixo.

As figuras utilizadas nesta seção estão disponíveis no Apêndice A para melhor visibilidade.

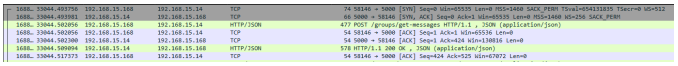

Figura 19: Handshake e GET de mensagens

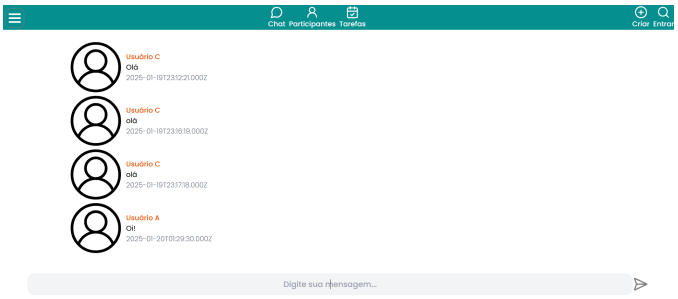

Figura 20: Interface do grupo de conversa após uma interação entre usuário C e A, a mensagem descrita na seção III-D corresponde à ultima mensagem enviada pelo usuário C ("olá")

IV. Conclusão

Com tudo o que foi descrito neste artigo é notável que a internet é uma rede complexa de protocolos e troca mensagens. O aplicativo desenvolvido neste projeto proporcionou um grande ganho de conhecimento que só seria possível compreender de forma prática como esta. Sendo assim, o grupo conseguiu aplicar os conhecimentos obtidos na aula quanto aos protocolos, linguagens de programação e a estrutura cliente servidor, como demonstrado no decorrer deste relatório. Os resultados obtidos podem ser vistos no pitch técnico e no codigo fonte presente no repositório (Apêndice A).

Referências

[1] Writing Markup with JSX – React. Disponível em: <https://react.dev/learn/writing-markup-with-jsx>. Acesso em: 18 de janeiro de 2025.
[2] EDDY, W. RFC 9293 - Transmission Control Protocol (TCP). Disponível em: <https://datatracker.ietf.org/doc/html/rfc9293>. Acesso em: 18 de janeiro de 2025.
[3] POSTEL, J. RFC 793 - Transmission control protocol. Disponível em: <https://datatracker.ietf.org/doc/html/rfc793>. Acesso em: 18 de janeiro de 2025.
[4] KUROSE, J. F.; ROSS, K. W. Computer networking : a top-down approach. 8. ed. Hoboken: Pearson, 2020.
[5] FIELDING, R. T.; NOTTINGHAM, M.;RESCHKE, J. RFC 9110 - HTTP Semantics. Disponível em: <https://datatracker.ietf.org/doc/html/rfc9110>. Acesso em: 18 de janeiro de 2025.
[6] Docker Desktop overview. Disponível em: <https://docs.docker.com/desktop/>. Acesso em: 18 jan. 2025.
[7] DOCKER. Enterprise Application Container Platform | Docker. Disponível em: <https://www.docker.com>. Acesso em: 18 jan. 2025.
[8] WIRESHARK FOUNDATION. Wireshark. Disponível em: <https://www.wireshark.org>. Acesso em: 18 jan. 2025.

Apêndice A
Links do Projeto

Aqui constam os links para o pitch técnico do aplicativo, do repositório contendo o código fonte para ele e das figuras utilizadas neste relatório. Para executar o servidor, é necessário apenas de ter instalado o aplicativo Docker Desktop [7] e, na raiz do projeto (onde está o arquivo `docker-compose.yml`), escrever o comando `docker compose up -build`, o que criará o servidor na rede local. Para acessar basta entrar na própria máquina com o link http://localhost:3000 ou com o link http://ipaddress:3000, em que `ipaddress` é o endereço IPv4 da máquina ou vpn em que o servidor está rodando (é necessário usar a segunda opção de link para entrar por outros dispositivos e checar se a firewall permite a interação na rede local com outros dispositivos)

- Link para pitch técnico: https://youtu.be/w5jFg5xnO20.

- Link para repositório: https://github.com/Schatten900/proj-rc.

- Link para figuras utilizadas no relatório: https://github.com/Schatten900/proj-rc/tree/relat%C3%B3rio.