UnB
Redes de Computadores — 2025.1
Projeto 1 - Turma 01

Giovanni Daldegan
232002520
Ciência da Computação
Universidade de Brasília
Brasília, DF

Rodrigo Rafik
232009502
Ciência da Computação
Universidade de Brasília
Brasília, DF

Rute Fernandes
232009549
Ciência da Computação
Universidade de Brasília
Brasília, DF

***Resumo*—O presente relatório descreve o processo de desenvolvimento do ChatWeb 2.0, uma aplicação web de chat em tempo real inspirada na estética e nas funcionalidades da internet do início dos anos 2000. Realizado no âmbito da disciplina de Redes de Computadores da Universidade de Brasília, o projeto visa aprofundar a compreensão do funcionamento de aplicações em rede, com ênfase na observação prática dos protocolos de comunicação e na arquitetura cliente-servidor.**

***Index Terms*—ChatWeb 2.0, aplicações web, protocolos de rede, comunicação em tempo real**

***Abstract*—This report describes the development process of ChatWeb 2.0, a real-time web chat application inspired by the aesthetics and functionalities of the early 2000s internet. Conducted within the scope of the Computer Networks course at the University of Brasília, the project aims to deepen understanding of network applications operation, with emphasis on practical observation of communication protocols and client-server architecture.**

***Index Terms*—ChatWeb 2.0, web applications, network protocols, real-time communication**

1. Introdução

Este projeto consiste no desenvolvimento do ChatWeb 2.0, uma aplicação web de chat em tempo real inspirada na estética e funcionalidades da internet do início dos anos 2000. Implementado como requisito da disciplina de Redes de Computadores da Universidade de Brasília, o projeto visa demonstrar na prática os conceitos fundamentais de comunicação em rede, protocolos de transporte e arquiteturas cliente-servidor.

O ChatWeb 2.0 combina tecnologias modernas (Flask-SocketIO, WebSockets, JavaScript ES6) com um design nostálgico que remete ao Windows XP e às interfaces web clássicas. A aplicação permite que múltiplos usuários se conectem simultaneamente, criem salas de chat temáticas e troquem mensagens instantaneamente, proporcionando uma experiência autêntica da web dos anos 2000.

Figura 1. Logotipo do projeto no estilo gráfico “y2k”

O sistema foi projetado como uma ferramenta educacional que ilustra o funcionamento de protocolos de rede modernos em um contexto prático e visual. A interface utiliza a fonte Monaspace Neon, cores características do Windows XP e elementos visuais com efeitos de sombra e relevo que remetem aos sistemas operacionais clássicos.

O projeto aborda conceitos como gerenciamento de conexões WebSocket, sincronização de estado distribuído, arquitetura orientada a eventos e análise de tráfego de rede em tempo real. Através da implementação e análise deste sistema, é possível compreender como as comunicações bidirecionais modernas funcionam e os desafios envolvidos no desenvolvimento de aplicações web em tempo real.

2. Fundamentação Teórica

O ChatWeb 2.0 foi desenvolvido com base no modelo de arquitetura cliente-servidor, no qual o servidor central é responsável por gerenciar conexões, autenticações e o fluxo de mensagens, enquanto os clientes interagem com o sistema por meio de uma interface gráfica. Essa organização segue o princípio clássico da comunicação em redes, conforme discutido por Kurose e Ross em Computer Networking: A Top-Down Approach[1], onde o servidor centraliza a lógica da aplicação e os clientes atuam como consumidores do serviço.

Dessa forma, a aplicação foi estruturada em duas partes principais para o desenvolvimento: front-end e back-end.

O front-end do ChatWeb 2.0 foi construído inteiramente com HTML, CSS e JavaScript, adotando um design visual

inspirado na estética da internet do início dos anos 2000. A interface oferece três telas principais para a interação do usuário: uma tela de login; uma tela com a lista de salas de chat disponíveis, que também permite a criação de novas salas; e, por fim, a tela de chat propriamente dita, onde os usuários podem enviar e receber mensagens em tempo real.

Para o back-end da aplicação, utilizamos a linguagem Python em conjunto com o framework Flask[2] para a construção de um servidor baseado na biblioteca Werkzeug[3]. Além disso, foi empregada a extensão Flask-SocketIO[5], que implementa o protocolo Socket.IO[4] para aplicações Flask. Essa biblioteca permite a comunicação bidirecional em tempo real entre cliente e servidor, viabilizando o envio e recebimento de mensagens instantâneas sem a necessidade de recarregamento da página. No lado do cliente, a biblioteca JavaScript Socket.IO foi utilizada para estabelecer conexões persistentes com o servidor e gerenciar os eventos de comunicação. Ela fornece uma API simples para emitir e escutar eventos personalizados, facilitando o desenvolvimento de interações em tempo real.

A identificação única de usuários e salas é feita por meio da biblioteca UUID, uma ferramenta nativa do Python utilizada para gerar identificadores únicos universais (UUIDs). Com ela, é possível garantir que cada cliente e sala de chat tenha um ID exclusivo, evitando conflitos e assegurando a integridade dos dados, mesmo em cenários com múltiplas conexões simultâneas.

Do ponto de vista da arquitetura de rede, a aplicação utiliza o protocolo WebSocket como principal meio de comunicação, permitindo conexões full-duplex contínuas sobre o TCP, o que reduz a latência e melhora a eficiência da troca de dados. Sobre ele, o Socket.IO atua como uma camada de abstração, oferecendo funcionalidades extras como reconexão automática, salas virtuais, eventos personalizados e suporte a diferentes formatos de dados. A conexão é iniciada via HTTP e, em seguida, atualizada para WebSocket, garantindo uma comunicação estável, rápida e confiável entre cliente e servidor.

3. Experimento

3.1. Ambiente Experimental

O desenvolvimento do ChatWeb 2.0 foi conduzido em um ambiente controlado que permitiu a análise detalhada dos protocolos de rede envolvidos na comunicação cliente-servidor. A configuração experimental foi estruturada para proporcionar observação prática dos conceitos teóricos estudados na disciplina.

Infraestrutura de Desenvolvimento: O sistema foi implementado utilizando uma arquitetura cliente-servidor moderna, com backend em Python e frontend em JavaScript. O servidor Flask-SocketIO foi configurado para aceitar conexões na porta 5000, permitindo acesso de múltiplos dispositivos na rede local através do protocolo WebSocket.

Ambiente de Rede: Os experimentos foram realizados em rede local (LAN), com o servidor executando em uma máquina Linux e clientes conectando através de navegadores web modernos. Esta configuração isolada permitiu análise detalhada do tráfego de rede sem interferências externas, facilitando a observação dos protocolos de comunicação em diferentes camadas.

3.2. Análise dos Resultados

Ao monitorar os pacotes transmitidos entre servidor e clientes durante o uso da aplicação, através de software próprio para a análise de tráfego de rede, foi possível destacar os pacotes gerados pela comunicação entre os dispositivos.

A ferramenta permitiu a inspeção das requisições HTTP dos arquivos presentes no servidor (elementos HTML, imagens, códigos em JavaScript, arquivos CSS) realizadas por parte dos clientes, além das mensagens de comunicação do SocketIO entre cliente e servidor. O tráfego desses dados permitiu que os clientes fizessem uso pleno dos recursos da aplicação: registrassem-se com nome, criassem salas de bate-papo e trocassem mensagens por elas.

No experimento conduzido, o laptop que se comunicou com o servidor será chamado de Cliente 1 e o celular, Cliente 2. A seção seguinte destrincha os resultados obtidos.

Análise dos Resultados Obtidos

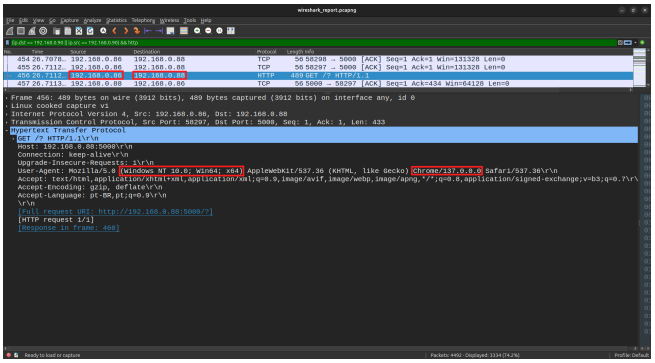

Figura 2. Requisição HTTP do índice do serviço pelo Cliente 1

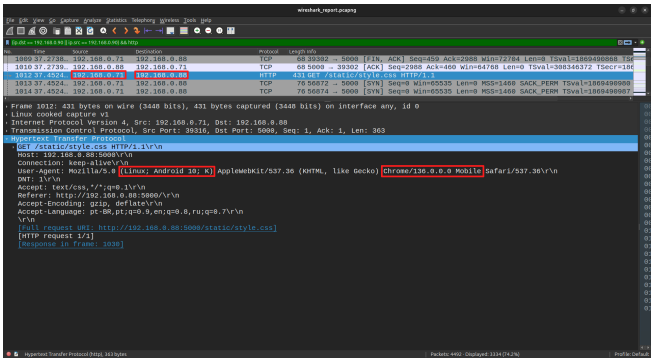

Figura 3. Requisição HTTP do índice pelo Cliente 2

A. Nas figuras 2 e 3, é possível notar as requisições HTTP do índice da página do serviço pelos Clientes 1 e 2,

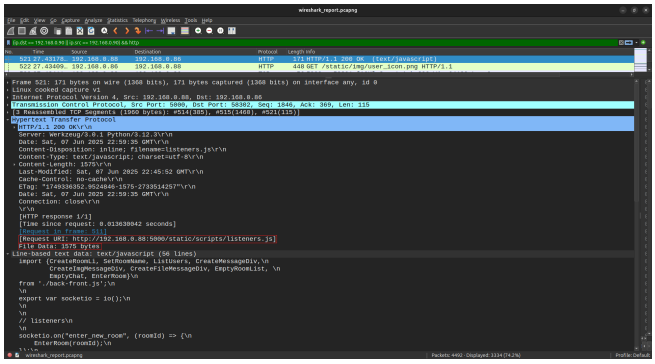

Figura 4. Resposta HTTP do servidor à requisição do arquivo “listeners.js” pelo Cliente 1

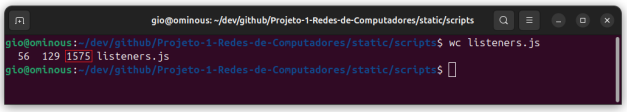

Figura 5. Tamanho em bytes do arquivo listeners.js, obtido pelo comando “wc”

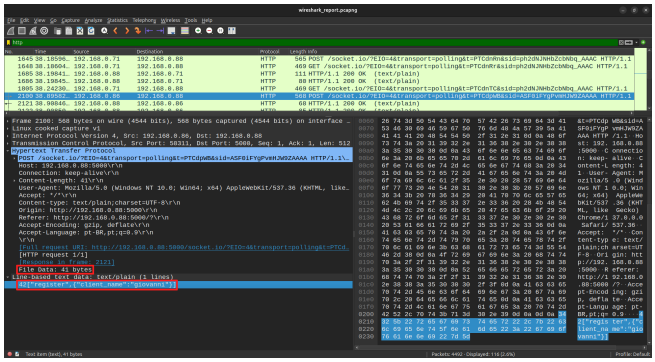

Figura 6. Mensagem gerada pelo SocketIO presente na aplicação do Cliente 1, registrando-o com o nome “giovanni”

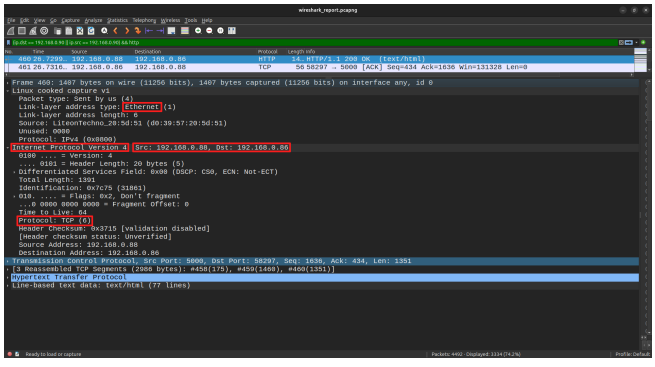

Figura 7. Cabeçalhos de enlace e rede da resposta à primeira requisição do Cliente 1

respectivamente. O primeiro opera em um Windows 10, com o navegador Chrome; e o segundo em um Android 10 baseado em Linux, também acessando o serviço pelo navegador

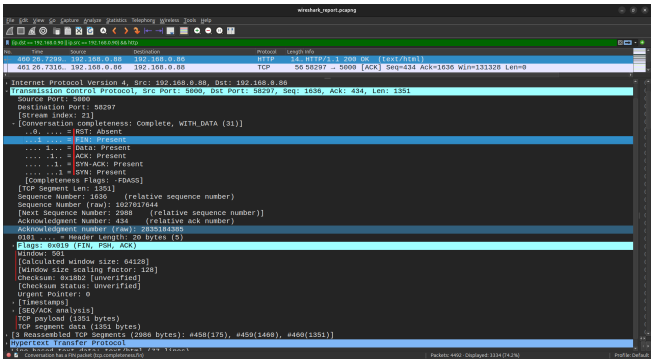

Figura 8. Cabeçalho de transporte da resposta à primeira requisição do Cliente 1

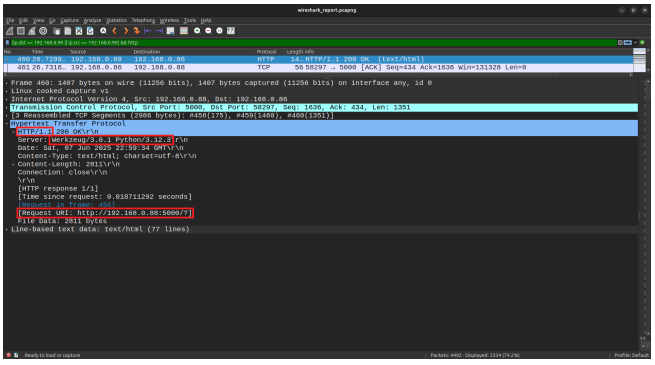

Figura 9. Cabeçalho de aplicação da resposta à primeira requisição do Cliente 1

Chrome (Mobile). Na figura 9, é visível que o servidor está operando na linguagem Python 3.12.3, representando uma aplicação Werkzeug na versão 3.0.1.

B. Nas figuras 2 e 3 estão representados os IPs dos clientes (192.168.0.86 e 192.168.0.71) e do servidor (192.168.0.88) operando na porta 5000.

C. Na figura 4, a resposta do servidor à requisição do arquivo tem uma carga útil de 1575 bytes, que coincide com o tamanho em bytes do arquivo no diretório local do servidor, como explícito na figura 5. Já na figura 6, o Cliente 1 envia uma mensagem gerada pelo SocketIO, pedindo o registro do cliente com o nome “giovanni”. Isso demonstra que a troca de pacotes entre clientes e servidor e sua carga útil correspondem ao funcionamento esperado da aplicação desenvolvida.

D. Analisando os dados da resposta do servidor ao Cliente 1 que contém o índice da página HTML da aplicação, pode-se ver: na figura 7, os dados dos cabeçalhos das camadas de enlace e rede, que denotam, respectivamente, a tecnologia utilizada para transmissão (Ethernet), e uso do IPv4 no endereçamento de origem e destino do segmento encapsulado; na figura 8, o cabeçalho da camada de transporte, com as flags TCP (ACK, RST, SYN, FIN, etc.), dados da janela de transmissão, tamanho do cabeçalho e checksum da mensagem encapsulada; e, por fim, na figura 9, o cabeçalho HTTP/1.1, com o tamanho da mensagem enviada, o URI ao

qual foi feita a requisição (com IP e porta) e o código da mensagem (200, OK), entre outras informações.

4. Conclusão

O desenvolvimento do ChatWeb 2.0 proporcionou uma compreensão prática dos conceitos fundamentais de redes de computadores, demonstrando a viabilidade das tecnologias WebSocket para comunicação em tempo real. A implementação combinou funcionalidade técnica moderna com design nostálgico do Windows XP, enquanto a análise detalhada do tráfego com Wireshark revelou o funcionamento prático da pilha TCP/IP. O projeto evidenciou o valor de abordagens práticas e criativas no ensino de sistemas distribuídos.

Referências

[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson, 2021.

[2] "Flask Documentation," [Online]. Available: https://flask.palletsprojects.com/en/stable/

[3] "Werkzeug Documentation," [Online]. Available: https://werkzeug.palletsprojects.com/en/stable/

[4] "Documentação SocketIO," [Online]. Available: https://socket.io/pt-br/docs/v4/

[5] "Flask-SocketIO Documentation," [Online]. Available: https://flask-socketio.readthedocs.io/en/latest/index.html

[6] "Gabrielcarvfer - Redes-de-Computadores-UnB," [Online]. Available: https://github.com/Gabrielcarvfer/Redes-de-Computadores-UnB

[7] "MDS Web Docs - References," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web