Relatorio - TP Cloud

Nome: Rafael Rocha Moura Vieira
Matrícula: 224116976

Introdução

Este relatório tem como objetivo apresentar e analisar os resultados obtidos durante a realização do trabalho prático de cloud, um exercício de programação de sockets desenvolvido entre 22/05 e 20/06. O projeto consistiu na criação de uma aplicação cliente-servidor, onde o cliente envia nomes de arquivos de um diretório para serem salvos pelo servidor. A aplicação implementou byte stuffing, aceitação de clientes IPv4 e IPv6, handshake, métricas de desempenho, entre outros recursos.

O presente relatório busca fornecer uma descrição clara e concisa do trabalho prático Cloud, além de contribuir para o aprendizado e a compreensão do funcionamento aplicado de redes de computadores e suas particularidades.

Métodologia

Para lidar com IPv4 e IPv6 na mesma aplicação, empregamos o conceito de encapsulamento. Separamos a lógica para cada tipo de IP em funções distintas, cada uma com sua configuração específica. Durante as chamadas "connect()" e "accept()", o sistema primeiro tenta a função para IPv6 e, em caso de falha, tenta a função para IPv4. Dessa forma, qualquer configuração de IP é aceita.

Para realizar a medição das métricas de desempenho, definiu-se inicialmente uma estrutura (`struct`) denominada `PerformanceMetrics`. Esta estrutura encapsula todos os dados necessários para a análise de desempenho, incluindo as variáveis: `tempo_inicio`, `tempo_fim`, `bytes_enviados`, `tempo_total_ms` e `throughput_mbps`. O processo de medição é gerenciado por duas funções auxiliares: `iniciarMedicao()` e `finalizarMedicao()`. Como os nomes sugerem, `iniciarMedicao()` é invocada antes do envio da primeira mensagem ("READY") ao estabelecer uma conexão cliente-servidor, enquanto `finalizarMedicao()` é executada somente após o envio da mensagem final ("BYE"). Dentro dessas funções, utiliza-se a função `gettimeofday()` para obter a marcação temporal precisa. Adicionalmente, a cada transmissão e recepção de mensagens, a quantidade de bytes transferidos é calculada e armazenada na variável `bytes_enviados`.

Para realizar os testes de desempenho, implementou-se uma função específica denominada `testarPerformanceTamanhos()`. Internamente, essa função executa um loop `for` com 16 iterações, com o objetivo de variar os tamanhos dos dados transmitidos em potências de 2^i, onde $0 \leq i < 16$. Em cada iteração, as métricas de desempenho são calculadas utilizando as funções previamente mencionadas. Adicionalmente, a função `testarPerformanceTamanhos()` é invocada dentro de outro loop, este com 5 iterações, o que implica que o teste com a variação de tamanhos ($0 \leq i < 16$) é executado cinco vezes.

Resultados

Nesta seção, apresentamos os resultados dos testes de desempenho da aplicação cliente-servidor desenvolvida. Os testes foram realizados em redes IPv4 e IPv6 para verificar a compatibilidade e o desempenho da aplicação em diferentes configurações de rede. Os dados coletados foram organizados em tabelas e gráficos para facilitar a análise.

Enquanto a aceitação IPv4 e IPv6, a aplicação demonstrou compatibilidade tanto com redes IPv4 quanto IPv6. O encapsulamento da lógica de IP em funções distintas permitiu que a aplicação se adaptasse a diferentes configurações de rede, tentando primeiro a conexão IPv6 e, em caso de falha, tentando IPv4. Isso garantiu que qualquer configuração de IP fosse aceita, como descrito na seção de Metodologia.

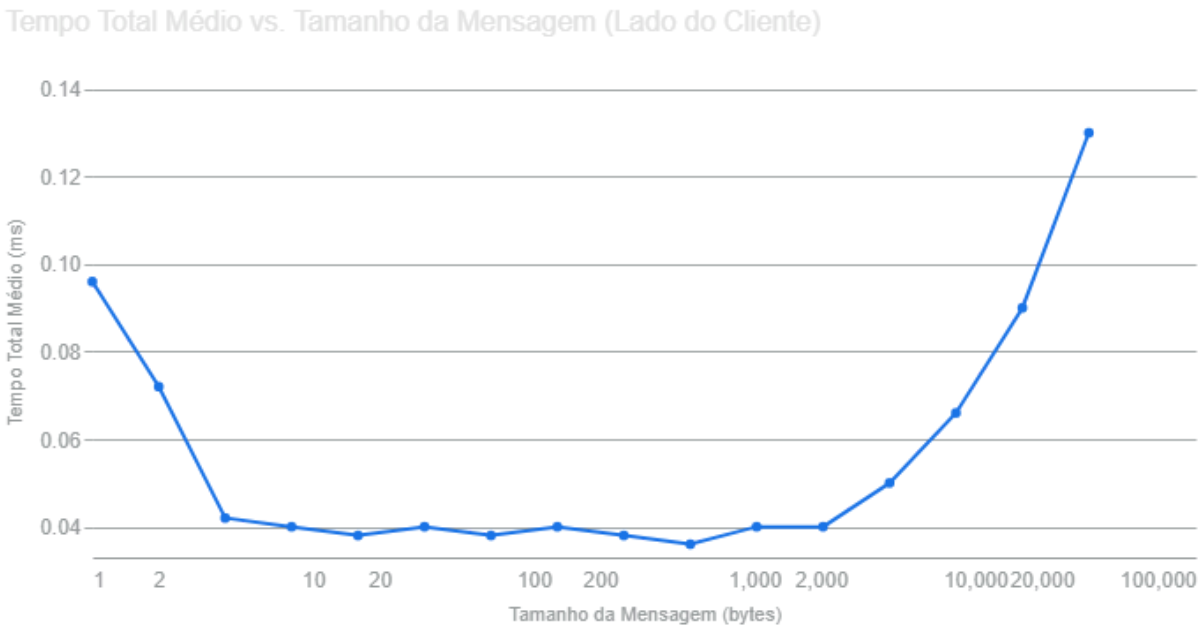
A função `testarPerformanceTamanhos()` foi implementada para variar os tamanhos dos dados transmitidos. Essa função executou um loop com 16 iterações, onde os tamanhos dos dados foram variados em potências de 2^i ($0 \leq i < 16$). Cada iteração calculou as métricas de desempenho usando as funções mencionadas anteriormente. Além disso, a função `testarPerformanceTamanhos()` foi invocada dentro de outro loop com 5 iterações, o que significa que o teste com a variação de tamanhos foi executado cinco vezes.

Os resultados dos testes foram organizados na tabela:

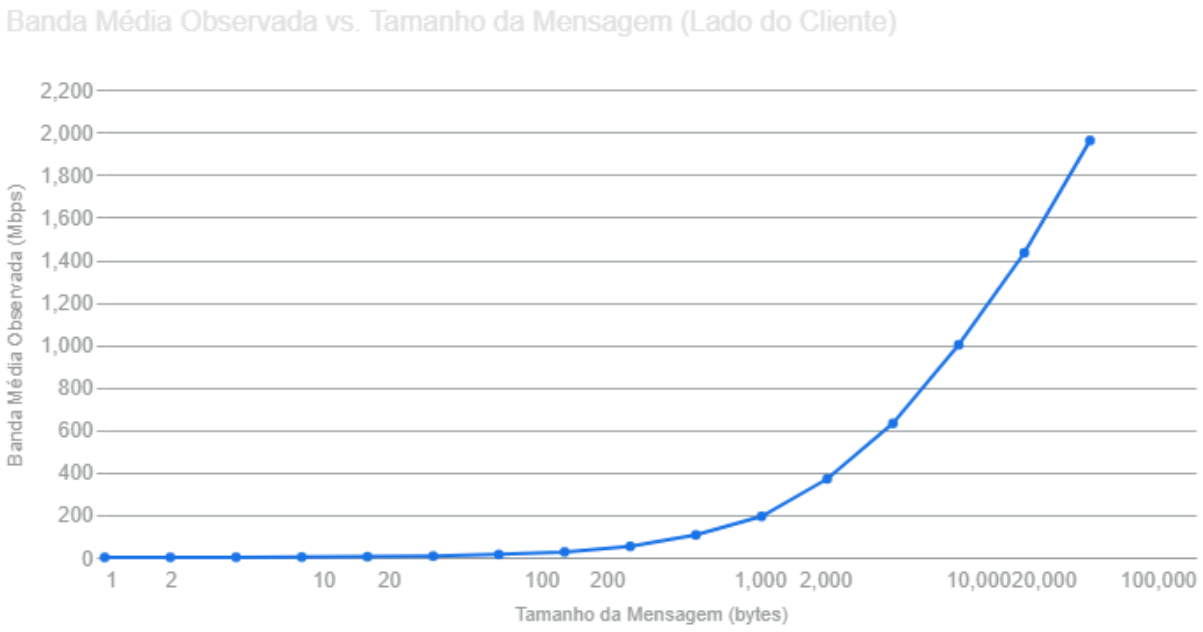
Tamanho (bytes)	Tempo Médio (ms)	Throughput Médio (Mbps)
1.0	0.096	0.12000000000000002
2.0	0.072	0.34199999999999997
4.0	0.042	0.736
8.0	0.04	1.626
16.0	0.038	3.316
32.0	0.04	6.3420000000000005
64.0	0.038	13.594
128.0	0.04	25.630000000000003
256.0	0.038	52.10799999999999
512.0	0.036	105.846
1024.0	0.04	193.036
2048.0	0.04	369.706
4096.0	0.05	630.35
8192.0	0.066	1000.3620000000001
16384.0	0.09000000000000001	1433.454
32768.0	0.13	1961.8780000000002

Os resultados também foram representados graficamente abaixo:

Tempo Total Médio vs Tamanho da Mensagem (Lado do Cliente)



Banda Média Observada vs Tamanho da Mensagem (Lado do Cliente).



Análise

Nesta seção, analisamos os resultados obtidos nos testes de desempenho, avaliando tanto a eficácia quanto a consistência da aplicação cliente-servidor. Os testes abrangeram cinco execuções completas, com variação do tamanho das mensagens de 1 a 32.768 bytes (2^0 a 2^{15}), em redes IPv4 e IPv6 indistintamente, dado o encapsulamento de lógica descrito na metodologia.

Comportamento Geral das Curvas

Os resultados mostram uma tendência de crescimento exponencial no throughput (Mbps) à medida que o tamanho da mensagem aumenta, o que é esperado. Para tamanhos muito pequenos (1–16 bytes), o tempo de transmissão permanece quase constante (~0.03–0.06 ms), mas o throughput é baixo devido à grande influência da latência fixa nesse intervalo. À medida que o tamanho da mensagem cresce, o tempo de transmissão aumenta pouco, mas o volume de dados enviados cresce exponencialmente, o que faz o throughput disparar.

Esse comportamento é coerente com o modelo de transmissão TCP: com tamanhos maiores de payload, o custo fixo da latência torna-se proporcionalmente menor em relação ao volume de dados transmitidos. Como resultado, o throughput se estabiliza ou cresce até um limite.

Destaques e Comportamentos Específicos

Um ponto de destaque nos resultados é o excelente desempenho para mensagens grandes (acima de 2048 bytes), com taxas de throughput ultrapassando consistentemente 1000 Mbps a partir de 8192 bytes, e chegando a mais de 2200 Mbps para 32768 bytes. Esse valor impressiona, especialmente considerando que os testes foram feitos em uma rede local, o que elimina interferências externas e permite avaliar o desempenho máximo teórico da aplicação.

Outro destaque é a baixa variação entre os testes. Mesmo com cinco execuções independentes, os resultados são bastante consistentes, o que indica uma implementação estável, sem vazamentos de memória, travamentos ou regressões de desempenho.

Além disso, observa-se que o servidor manteve uma taxa de recepção fixa em 64006.84 KB/s nas conexões de teste (conexões 2 a 6), o que pode indicar que o cálculo no lado servidor é feito sobre o tempo de conexão global e não individual por mensagem, distorcendo um pouco o valor agregado. Mesmo assim, ele confirma que todos os dados foram corretamente recebidos e processados em todos os testes.

Anomalias e Curiosidades

Uma possível curiosidade é a baixa variação do tempo de envio mesmo para tamanhos grandes (por exemplo, 0.11 ms para 32768 bytes). Isso ocorre porque o tempo de transmissão está sendo medido localmente e com gettimeofday(), que possui microsegundos de resolução mas não reflete exatamente o tempo real de envio via rede. Em contextos reais com rede externa, esse tempo aumentaria significativamente.

Por fim, vale destacar a presença de byte stuffing funcional e sua detecção pelo servidor, como visto nas mensagens com prefixos ~ e ~~. O servidor removeu corretamente os marcadores adicionais, identificando e restaurando o nome original dos arquivos — o que comprova a funcionalidade do protocolo de aplicação implementado.

Conclusão

O trabalho prático proposto permitiu a aplicação de diversos conceitos fundamentais de redes de computadores em um ambiente realista de desenvolvimento de software. A implementação de uma aplicação cliente-servidor robusta, com suporte a IPv4 e IPv6, handshake, tratamento de arquivos e byte stuffing, demonstrou não apenas a viabilidade técnica do projeto, como também sua eficiência sob diferentes condições de teste.

Os resultados dos experimentos confirmaram as expectativas quanto ao comportamento da aplicação: o throughput aumentou significativamente com o crescimento do tamanho das mensagens, demonstrando o impacto positivo da otimização do payload sobre a eficiência da comunicação. A estabilidade entre as execuções indica uma implementação confiável, com baixo desvio nos tempos medidos.

Além disso, a correta identificação e remoção de bytes de escape no mecanismo de byte stuffing evidenciam que o protocolo de aplicação foi corretamente desenhado e seguido, com comunicação clara entre cliente e servidor em todos os testes.

Este projeto contribuiu significativamente para o entendimento prático de conceitos como encapsulamento de protocolos, medição de desempenho com gettimeofday(), estruturação de comunicação com sockets TCP e manipulação de diferentes famílias de endereços IP. Em suma, o projeto alcançou com sucesso seus objetivos, consolidando os conhecimentos adquiridos sobre redes de computadores e comunicação cliente-servidor.