Universidade Federal
de São João del-Rei

Redes de Computadores
1º Semestre de 2025
Prof. Rafael Sachetto Oliveira

Bárbara Cardoso da Silva e Almeida

Diego Augusto Silva Castro

Samanta Ribeiro Freire

Relatório do Trabalho Prático 1 da disciplina de Redes de Computadores do curso de Ciência da Computação da Universidade Federal de São João del-Rei (UFSJ).

Maio de 2025

Sumário

1. Introdução

Este relatório descreve a implementação de uma aplicação cliente-servidor desenvolvida em linguagem C, com o propósito de realizar a transferência de arquivos entre cliente e servidor por meio dos protocolos TCP e UDP. A aplicação também realiza o cálculo do hash MD5 dos dados recebidos, a fim de verificar sua integridade, e mede a velocidade de download durante as transmissões. O principal objetivo do estudo é comparar o desempenho dos dois protocolos de transporte, considerando aspectos como taxa de transferência, confiabilidade na comunicação e ocorrência de perdas de pacotes.

2. Protocolos TCP e UDP

A camada de transporte do modelo TCP/IP é responsável por fornecer comunicação de ponta a ponta entre processos em sistemas distintos. Nessa camada, destacam-se dois protocolos principais: o *Transmission Control Protocol* (TCP) e o *User Datagram Protocol* (UDP). Ambos servem a propósitos distintos e possuem características específicas que influenciam diretamente no desempenho, confiabilidade e adequação a diferentes tipos de aplicações.

2.1. Protocolo TCP

O *Transmission Control Protocol* (TCP) é um protocolo orientado à conexão, projetado para garantir a entrega confiável de dados entre dois pontos da rede. Antes que qualquer dado seja transmitido, é necessário que haja o estabelecimento de uma conexão entre cliente e servidor, por meio de um processo conhecido como *three-way handshake*, que consiste na troca inicial de três mensagens (SYN, SYN-ACK e ACK).

Uma vez estabelecida a conexão, o TCP assegura que os dados transmitidos cheguem ao destino de forma ordenada, íntegra e sem duplicações. Para isso, o protocolo implementa uma série de mecanismos robustos, tais como:

- Numeração de sequência dos segmentos, permitindo a reordenação correta no destino.
- Confirmação (ACK) da recepção dos dados, possibilitando a retransmissão de pacotes perdidos.

- Controle de fluxo, por meio da janela deslizante, que ajusta dinamicamente a quantidade de dados que podem ser enviados antes de receber uma confirmação.
- Controle de congestionamento, para evitar a sobrecarga da rede, utilizando algoritmos como *slow start*, *congestion avoidance*, *fast retransmit* e *fast recovery*.

Essas funcionalidades tornam o TCP ideal para aplicações em que a confiabilidade e a integridade dos dados são essenciais, como em transferências de arquivos, e-mails e navegação web.

2.2. Protocolo UDP

O *User Datagram Protocol* (UDP), por sua vez, é um protocolo de transporte sem conexão e não confiável, baseado na transmissão de datagramas independentes. Não há estabelecimento prévio de conexão nem garantia de entrega, ordem ou integridade dos pacotes. O UDP simplesmente envia os dados, sem verificar se o receptor os recebeu corretamente.

Por não incorporar mecanismos como controle de fluxo, controle de congestionamento ou retransmissão automática, o UDP apresenta uma latência significativamente menor que o TCP. Essa leveza e simplicidade o tornam mais eficiente em cenários onde a velocidade é mais crítica que a confiabilidade, tais como:

- Aplicações de *streaming* de áudio e vídeo.
- Jogos online.
- Transmissões em tempo real, como VoIP.
- Protocolos de descoberta ou serviços multicast.

No entanto, a ausência de garantias do UDP implica que aplicações que dependem desse protocolo devem implementar, se necessário, seus próprios mecanismos de correção e verificação de integridade dos dados.

3. Implementação e Metodologia de Medição

A implementação do sistema foi dividida em três principais componentes: o cliente, o servidor e a biblioteca compartilhada, que contém funções auxiliares para manipulação de hashes, protocolos de comunicação e leitura da linha de comando. A comunicação entre o

cliente e o servidor é realizada por meio dos protocolos TCP e UDP. A seguir, serão detalhadas as principais partes do código para cada um dos componentes.

3.1. Cliente

O cliente é responsável por se conectar ao servidor e transferir os dados, medindo a taxa de download. O código a seguir mostra como a conexão é estabelecida:

```
void app(Shared sh) {
  if (connect(sh.sockfd, (struct sockaddr *)&sh.sock_addr,
              sizeof(sh.sock_addr))) {
    die("Failed to connect to the server: %s", strerror(errno));
  }

  struct timespec before;
  clock_gettime(CLOCK_MONOTONIC, &before);

  size_t total_bytes_read = sh.is_udp ? udp(sh) : tcp(sh);
  print("Total bytes: %ld", total_bytes_read);

  double total_time = time_elapsed(&before);
  print_download(total_bytes_read / total_time);
}
```

A função `app()` realiza a conexão com o servidor utilizando `connect()`. Após a conexão bem-sucedida, a função `clock_gettime()` é chamada para capturar o tempo inicial. Em seguida, dependendo do protocolo escolhido (TCP ou UDP), o cliente chama a função `udp()` ou `tcp()` para iniciar a recepção de dados.

3.1.1. Protocolo TCP

Para o protocolo TCP, a função `tcp()` é chamada para processar a recepção de dados. O cliente continua recebendo dados até que o servidor encerre a conexão. O código a seguir mostra a implementação:

```
size_t tcp(Shared sh) {
  ssize_t bytes_read, total_bytes_read = 0;

  while (1) {
    bytes_read = recv(sh.sockfd, sh.buffer, BUFSIZE, 0);
    if (bytes_read == 0)
      break;
    if (bytes_read == -1) {
      print("Server error");
      break;
    }
    total_bytes_read += bytes_read;

    sh_update_hash(sh, sh.buffer, bytes_read);
  }
  char *f = sh_finish_hash(sh);
  print("MD5: %s", f);
```

```
  return total_bytes_read;
}
```

O código utiliza a função `recv()` para receber dados do servidor. A variável `total_bytes_read` acumula o total de dados recebidos e, após a recepção dos dados, o hash MD5 é finalizado e impresso.

3.1.2. Protocolo UDP

Quando o protocolo UDP é selecionado, o cliente entra em um loop de recebimento de pacotes do servidor. O código a seguir mostra como os pacotes UDP são recebidos e processados, além de como o cliente calcula o hash MD5 dos dados recebidos:

```
size_t udp(Shared sh) {
  UDPPacket *pack = (UDPPacket *)sh.buffer;
  pack->packet_id = 0;

  send(sh.sockfd, NULL, 0, 0);

  ssize_t bytes_read = 0, total_bytes_read = 0;
  uint16_t expected_packet = 0, packets_lost = 0;
  while (1) {
    bytes_read = recv(sh.sockfd, pack, UDPPacketSize(BUFSIZE), 0);
    if (pack->packet_id == (uint16_t)-1) {
      break;
    }
    if (bytes_read == -1) {
      die("Error: %s", strerror(errno));
    }
    bytes_read -= UDPPacketSize(0);
    total_bytes_read += bytes_read;

    if (pack->packet_id != expected_packet) {
      packets_lost++;
      expected_packet = pack->packet_id;
    }
    if (packets_lost == 0) {
      sh_update_hash(sh, pack->body, bytes_read);
    }
    expected_packet++;
  }

  print("Packets Lost: %d", packets_lost);

  if (packets_lost == 0) {
    char *f = sh_finish_hash(sh);
    print("MD5: %s", f);
  } else {
    print("MD5: null");
  }

  return (size_t)total_bytes_read;
}
```

Nesse trecho, o cliente envia um pacote em branco para o servidor para solicitar o envio dos dados, e os pacotes recebidos são processados. A variável `expected_packet` controla a ordem

dos pacotes, e a variável `packets_lost` é incrementada sempre que um pacote é perdido. O hash MD5 é atualizado para cada pacote recebido, utilizando a função `sh_update_hash()`. O servidor indica a finalização enviando um pacote de número −1. Ao final, o hash MD5 total é impresso.

3.2. Servidor

O servidor é responsável por aceitar conexões dos clientes e enviar os dados solicitados. O servidor implementa os protocolos TCP e UDP de maneira semelhante ao cliente, com a diferença de que ele envia os dados ao invés de recebê-los.

3.2.1. Protocolo TCP

O código para o protocolo TCP no servidor é mostrado abaixo. Ele aceita uma conexão, envia os dados e calcula o hash MD5:

```
void tcp(int files, Shared sh) {
  listen(sh.sockfd, 1);
  while (files --> 0) {
    int bytes_read = 0;
    int client = accept(sh.sockfd, NULL, NULL);
    while ((bytes_read = fread(sh.buffer, 1, BUFSIZE, sh.file)) != 0) {
      ssize_t h = send(client, sh.buffer, bytes_read, 0);
      if (h == 0) {
        print("Client disconnected");
        break;
      } else if (h == -1) {
        die("Send Error: %s", strerror(errno));
      }
      sh_update_hash(sh, sh.buffer, bytes_read);
    }
    fseek(sh.file, 0, SEEK_SET);
    close(client);

    char *f = sh_finish_hash(sh);
    fseek(sh.file, 0, SEEK_SET);
    print("MD5: %s", f);
  }
}
```

Aqui, o servidor aceita uma conexão com `accept()`, lê os dados do arquivo com `fread()` e os envia ao cliente com `send()`. O hash MD5 é atualizado a cada bloco de dados enviado.

3.2.2. Protocolo UDP

No protocolo UDP, o servidor utiliza `recvfrom()` para aguardar clientes e `sendto()` para enviar os dados. A implementação é a seguinte:

```
void udp(int files, Shared sh) {
  UDPPacket *pack = (UDPPacket *)sh.buffer;
```

```
  while (files --> 0) {
    struct sockaddr_storage client_addr;
    socklen_t client_size = sizeof(client_addr);
    bzero(&client_addr, client_size);

    // wait for client
    if (recvfrom(sh.sockfd, NULL, 0, 0, (void *)&client_addr, &client_size) ==
        -1) {
      print("Recv Error: %s", strerror(errno));
      break;
    }

    uint16_t packet = 0;
    size_t bytes_read = 0;
    while (1) {
      bytes_read = fread(pack->body, 1, BUFSIZE, sh.file);
      if (bytes_read == 0) {
        break;
      }

      sh_update_hash(sh, pack->body, bytes_read);

      pack->packet_id = packet;
      ssize_t result;

      result = sendto(sh.sockfd, pack, UDPPacketSize(bytes_read), 0,
                      (void *)&client_addr, client_size);
      if (result == -1) {
        print("Send Error: %s", strerror(errno));
        break;
      }
      packet++;
    }
    pack->packet_id = -1;
    sendto(sh.sockfd, pack, UDPPacketSize(0), 0, (void *)&client_addr,
           client_size);

    char *f = sh_finish_hash(sh);
    fseek(sh.file, 0, SEEK_SET);
    printf("MD5: %s\n", f);
  }
}
```

A função `recvfrom()` é usada para aguardar o pedido de dados do cliente, e `sendto()` envia os dados do arquivo em pacotes UDP. A cada pacote enviado, o hash MD5 é atualizado. Ao final do arquivo, ele envia um pacote de número −1 para indicar ao cliente que acabou. Depois disso, mostra o hash MD5 para conferência.

3.3. Metodologia de Medição

A avaliação de desempenho da aplicação foi baseada na taxa de download, determinada pela razão entre o volume total de dados recebidos e o tempo necessário para a transferência. Para mensurar o tempo com precisão, utilizou-se a função `clock_gettime()` do sistema

operacional, que registra o instante imediatamente antes e depois da transmissão. A fórmula aplicada é mostrada abaixo:

$$\text{Velocidade de Download} = \frac{\text{Total de Bytes Lidos}}{\text{Tempo Total}}$$

Essa abordagem permitiu comparar o desempenho dos protocolos TCP e UDP sob as mesmas condições experimentais. No caso do UDP, também foi contabilizado o número de pacotes perdidos, fornecendo uma perspectiva adicional sobre a confiabilidade da transmissão.

O Gráfico 1 ilustra a média de dados transferidos por sessão para cada protocolo. Observa-se que o TCP apresenta uma média superior, com 336.92 Mb, enquanto o UDP alcança 284.875 Mb.

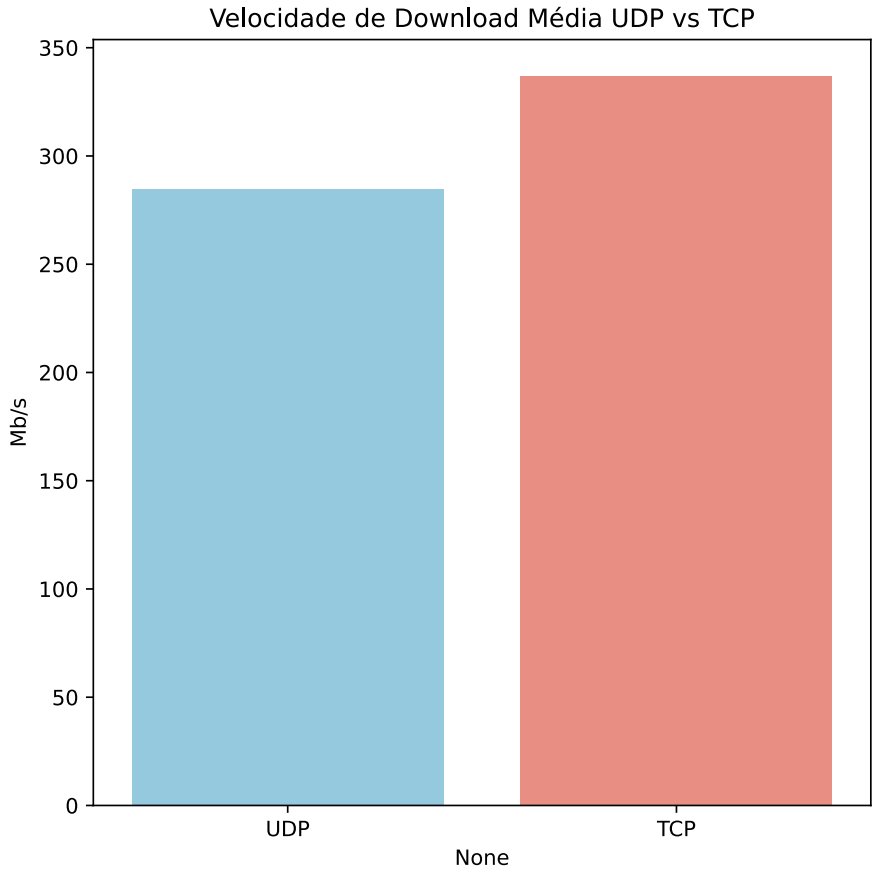

Figura 1: Comparação de Velocidade Média de Download (TCP vs UDP)

Complementando a análise gráfica, a Tabela 1 exibe os dados brutos utilizados na geração do gráfico anterior:

Teste	TCP	UDP
1	332.85	298.12
2	350.33	288.59
3	341.12	307.51
4	332.64	299.74
5	319.07	297.66
6	350.38	273.36
7	338.52	301.47
8	336.57	296.24
9	326.78	277.16
10	340.92	208.90

Tabela 1: Velocidade de Download dos Testes (TCP vs UDP)

A Tabela 2 apresenta um resumo estatístico detalhado das velocidades de download. O TCP demonstrou desempenho mais consistente, com valores concentrados em torno da média e menor desvio padrão. Em contraste, o UDP apresentou maior variação, com velocidades oscilando entre 208.9 Mb/s e 307.51 Mb/s, evidenciando sua instabilidade mesmo em redes confiáveis. A diferença de desempenho será elaborada nos subtópicos posteriores.

Estatística	TCP	UDP
Contagem (n)	10	10
Média	336.92	284.55
Desvio Padrão	9.74	28.78
Mínimo	319.07	208.9
25% Percentil	332.69	280.02
Mediana (50%)	337.55	296.95
75% Percentil	341.07	299.33
Máximo	350.38	307.51

Tabela 2: Resumo Estatístico das Velocidades de Download (TCP vs UDP)

4. Discussão

4.1. Perdas

Durante os testes realizados com os protocolos TCP e UDP, foram foram detectadas perdas de pacotes no protocolo UDP. Isso se deve ao fato de que, como mencionado previamente, o UDP não conta com mecanismos para mitigar isso. Nesses ambientes, a latência é extremamente baixa e não há interferência externa ou congestionamento, o que minimiza drasticamente as chances de perda de pacotes.

4.2. Velocidade

Enquanto o TCP apresentou uma velocidade média de 336.92 Mb/s, o UDP atingiu 284.55 Mb/s, correspondendo a 84.46% do desempenho do TCP. O TCP manteve velocidades consistentemente altas, variando entre 319.07 Mb/s e 350.38 Mb/s, com uma amplitude relativamente pequena (9.3%). Já o UDP, mesmo alcançando picos de 307.51 Mb/s em um dos testes, também registrou o valor mínimo de 208.9 Mb/s, revelando uma amplitude de variação maior (28.78%).

Essa diferença expressiva provavelmente é devida à otimização intensa que a pilha TCP apresenta nos sistemas operacionais modernos. Em contrapartida, o UDP, por ser mais utilizado para operações sensíveis à latência em vez de velocidade, apresenta desempenho levemente inferior. Isso é notável, considerando que o TCP possui bastante overhead e round--trips.

4.3. Comportamento Observado

Pode-se observar que o TCP demonstrou maior estabilidade, refletindo sua natureza orientada a conexão e mecanismos de controle de congestionamento. O UDP, por sua vez, apresentou flutuações mais acentuadas, característica esperada devido à ausência de controle de fluxo.

5. Conclusão Comparativa

Em síntese, enquanto o TCP se destaca pela velocidade e confiabilidade, sendo ideal para aplicações como transferência de arquivos (FTP), navegação web (HTTP/HTTPS) e envio de e-mails (SMTP), o UDP mostra maior volatilidade, sendo mais adequado para aplicações que priorizam latência mínima em detrimento da consistência, como chamadas de voz (VoIP), *streaming* ao vivo, jogos online e serviços de DNS. Logo, a escolha do protocolo deve considerar o contexto da aplicação, balanceando os requisitos de velocidade, confiabilidade e tempo de resposta.