Comparação de Desempenho entre TCP e UDP na Transferência de Arquivos

Ana Luiza da Rocha Herrmann[1], **Letícia Torres**[1]

[1]Ciência da Computação – Universidade Estadual do Oeste do Paraná (UNIOESTE)
85819-110 – Cascavel – PR – Brasil

Abstract. *TCP and UDP protocols are widely used as a delivery service between end-devices processes. Their particularities cause them to be applied for different purposes, however it's possible, with a few modifications, use them both for the same goals. It were applied the both protocols for file transfer, which needs reliable data transfer, in a client-server communication. Developing with Java Language and its libraries, it was possible to analyse TCP's performance by different sizes of files and packages, however it was not feasible to compare the both protocols due to UDP client issues to receive packages.*

Resumo. *Os protocolos TCP e UDP são amplamente utilizados como um serviço de entrega entre dois processos de sistemas finais. Suas particularidades fazem com que eles sejam aplicados para diferentes fins, porém é possível com algumas modificações, utilizá-los para os mesmos objetivos. Foram aplicados os dois protocolos para realizar transferência de arquivos, que necessita de entrega confiável, em uma comunicação cliente-servidor. Utilizando a linguagem Java e suas bibliotecas, foi possível analisar o desempenho do TCP conforme tamanhos de arquivos e pacotes diferentes, entretanto não foi possível realizar a comparação de desempenho entre os dois protocolos devido à problemas com o recebimento de pacotes do cliente UDP.*

1. Introdução

A rede TCP/IP disponibiliza dois protocolos de transportes para a camada de aplicação: o UDP (*User Datagram Protocol*) [RFC 768 2017] e o TPC (*Transmission Control Protocol*) [RFC 739 2017]. A função principal destes protocolos é ampliar o serviço de entrega entre dois sistemas finais para um serviço de entrega entre dois processos que rodam em sistemas finais.

Tanto o UDP quanto o TCP, além de realizarem entrega de dados processo a processo, fornecem verificação de integridade com a inclusão de um campo para detecção de erros em seus segmentos. Note que o UDP não fornece ao usuário nada além desses dois serviços, fazendo com que seja um serviço não confiável, pois não garante que esses dados sejam entregues de maneira intacta ao processo destino. O TCP oferece entrega confiável de dados por meio de controle de fluxo, números de sequência, reconhecimento e temporizadores. Outras duas particularidades do TCP é que ele oferece (i) um serviço orientado à conexão, isto é, antes que ocorra o envio de dados de um processo a outro, primeiro esses processos enviam segmentos para estabelecer parâmetros de transferência de dados; e (ii) controle de congestionamento, a fim de evitar uma quantidade excessiva de tráfego nos enlaces e comutadores devido a outras conexões TCP [Kurose and Ross 2013].

2. Descrição do Problema

Devido à simplicidade do UDP, ele é recomendado para aplicações que realizam troca de dados em tempo real, que são pouco sensíveis à perda e atraso na rede, como jogos, *streaming* e videoconferência. Já o TCP pode ser utilizado por aplicações que requerem uma entrega confiável de dados, como transferência de arquivos e aplicações Web.

É possível utilizar ambos os protocolos para a mesma finalidade. Porém, as diferenças entre os dois pode causar uma diferença de desempenho. Ao utilizar o TCP como *streaming* de vídeo por exemplo, mesmo que haja entrega confiável o vídeo irá congelar nos momentos de perda de pacotes.

O objetivo deste trabalho é comparar e analisar o desempenho dos protocolos TCP e UDP na transferência de arquivos de diferentes tamanhos com diferentes pacotes. Como a transferência de arquivos exige garantia de entrega, será necessário implementar um protocolo para tal no UDP. Com isso, será avaliado se o atributo velocidade do UDP permanece.

3. Implementação

3.1. TCP

A implementação dos protocolos foi feita utilizando a Linguagem Java. Os trechos de código a seguir mostram somente as partes cruciais para a conexão e transferência de arquivo entre Servidor e Cliente. O código completo pode ser acessado através do repositório no Github (`https://github.com/herrmannluiza/Redes_TCP_UDP`).

Código 1. Servidor TCP

```
ServerSocket serverSocket = new ServerSocket(PORTA);
Socket socket = serverSocket.accept();
System.out.println("Conectado com: " +
                    socket.getInetAddress().getHostAddress());
File send = new File(args[0]);
long fileLength = send.length();
FileInputStream fileSend = new FileInputStream(send);
BufferedInputStream buffFile = new BufferedInputStream(fileSend);
OutputStream output = socket.getOutputStream();

byte[] fileChunk;
long postionPtr = 0;

long startTime = System.nanoTime();
while (postionPtr != fileLength) {
  if (fileLength - postionPtr >= sizePacket)
    postionPtr += sizePacket;
  else {
    sizePacket = (int) (fileLength - postionPtr);
    postionPtr = fileLength;
  }
  fileChunk = new byte[sizePacket];
  buffFile.read(fileChunk, 0, sizePacket);
  output.write(fileChunk);
}
long endTime = System.nanoTime();
```

No Código 1, as duas primeiras linhas instanciam os objetos ServerSocket e Socket, responsáveis por atender requisições de clientes e comunicar-se com um cliente no momento em que a conexão for aceita, respectivamente. Para que se possa ler o arquivo byte a byte de acordo com o tamanho do pacote, utilizou-se a classe BufferedInputStream[1] devido ao seu desempenho na leitura de bytes, pois seu método read() lê uma quantia de 8192 bytes e armazena em um buffer, realizando ”tamanho de arquivo / 8192”chamadas ao SO. A escrita do conteúdo do arquivo e envio ao socket é feita com o objeto instanciado na linha 9. As linhas 15 a 20 tratam da divisão do arquivo em pacotes para transferência utilizando um ponteiro que lê o arquivo pacote a pacote, ou o restante do arquivo caso seja menor que o tamanho do pacote. Após, é realizada leitura, e escrita e envio do arquivo ao Cliente.

Código 2. Cliente TCP

```
1  Socket socket = new Socket(SERVIDOR, PORTA);
2  FileOutputStream file = new FileOutputStream(args[0]);
3  BufferedOutputStream buffer = new BufferedOutputStream(file);
4  InputStream in = socket.getInputStream();
5
6  byte[] fileChunk = new byte[sizePacket];
7  int bytesRead = 0;
8
9  while ((bytesRead = in.read(fileChunk)) != -1) {
10     buffer.write(fileChunk, 0, bytesRead);
11 }
```

Similar ao código do Servidor, a primeira linha instancia um objeto Socket para comunicar-se com a porta e o endereço do servidor. Para escrever em arquivo é utilizada a classe BufferedOutputStream e para ler e receber os pacotes do servidor, InputStream. As linhas 9, 10 e 11 correspondem a leitura e escrita do arquivo recebido enquanto não atingir final de stream (-1)[2].

3.2. UDP

Código 3. Server UDP

```
1  DatagramSocket serverSocket = new DatagramSocket(PORTA);
2  DatagramPacket receivePacket = new DatagramPacket(receiveData, ↩
       receiveData.length);
3  serverSocket.receive(receivePacket);
4  System.out.println(new String(receiveData, 0, receiveData.length, "↩
       UTF-8"));
5
6  while (positionPtr != fileLength) {
7
8      if (fileLength - positionPtr >= sizePacket) {
9          positionPtr += sizePacket;
10     } else {
11         sizePacket = (int) (fileLength - positionPtr);
12         positionPtr = fileLength;
13     }
```

[1]https://docs.oracle.com/javase/7/docs/api/java/io/BufferedInputStream.html
[2]https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html#read()

```
14
15      fileChunk = new byte[sizePacket];
16      buff_file.read(fileChunk, 0, sizePacket);
17      DatagramPacket packet = new DatagramPacket(fileChunk, sizePacket,↩
            IPClient, port);
18
19      try {
20          connection = new DatagramSocket();
21          connection.send(packet);
22          serverSocket.setSoTimeout(2000);
23          connection.close();
24      } catch (SocketTimeoutException so){
25          connection = new DatagramSocket();
26          connection.send(packet);
27          serverSocket.setSoTimeout(2000);
28          connection.close();
29      }
30  }
```

A primeira linha do código do Servidor UDP instancia um socket UDP em uma dada porta, similar ao TCP, porém a classe é um DatagramSocket, pois o UDP não é orientado a conexão. Como o UDP não é orientado a conexão, a abordagem utilizada foi fazer o cliente realizar uma solicitação ao Servidor indicando para quem enviar o arquivo. As linhas de 6 a 30 implementam o envio do arquivo ao cliente: da mesma maneira que o TCP, o ponteiro lê o arquivo pacote a pacote, ou o restante do arquivo caso seja menor que o tamanho do pacote. O código no escopo do *try-catch* foi uma tentativa de gerar um código com garantia de entrega. Porém, a mensagem de confirmação de recebimento que deveria ser enviada pelo cliente não estava sendo confirmada. Em todos os casos a exceção *SocketTimeoutException* era gerada.

Código 4. Cliente UDP

```
1  DatagramPacket sendPacket = new DatagramPacket(reqMsg, reqMsg.length,↩
       IPAddress, PORTA);
2  clientSocket.send(sendPacket);
3
4  byte[] receiveData = new byte[4];
5  DatagramPacket receivePacket = new DatagramPacket(receiveData, ↩
       receiveData.length);
6  clientSocket.receive(receivePacket);
7
8  while(positionPtr != fileLength){
9      receivePacket = new DatagramPacket(fileChunk, sizePacket);
10     clientSocket.receive(receivePacket);
11
12     if (fileLength - positionPtr >= sizePacket) {
13         positionPtr += sizePacket;
14     } else {
15         sizePacket = (int) (fileLength - positionPtr);
16         positionPtr = fileLength;
17     }
18     buffer.write(fileChunk, 0, sizePacket);
19 }
20 clientSocket.close();
```

Aqui o cliente primeiramente envia a solicitação citada no Código 3 ao servidor. Após, recebe os dados do servidor e os lê escrevendo em buffer, da mesma maneira que o Código 2.

3.3. Ambiente de Execução

Os protocolos foram implementados e executados na máquina cuja configuração é mostrada na Tabela 1.

Tabela 1. Configurações da Máquina

RAM	8GB
HD	103GB
CPU	AMD FX(tm)-8120 Eight-Core x 8
GPU	Gallium 0.4 on AMD BARTS
SO	ubuntu 14.04 64-bit

3.4. Exemplos de Execução

São definidos dois parâmetros tanto para o Cliente quanto para o Servidor: o nome do arquivo e o tamanho do pacote.

```
java Servidor <arquivo_para_envio> <tamanho_do_pacote>
```

```
java Cliente <arquivo_de_saida> <tamanho_do_pacote>
```

Um exemplo de execução Cliente-Servidor TCP é mostrado nas Figuras 1 e 2. A saída dos Servidores mostra o tempo necessário para a transferência do arquivo e o número de blocos gerados a partir do tamanho do pacote para cada arquivo.

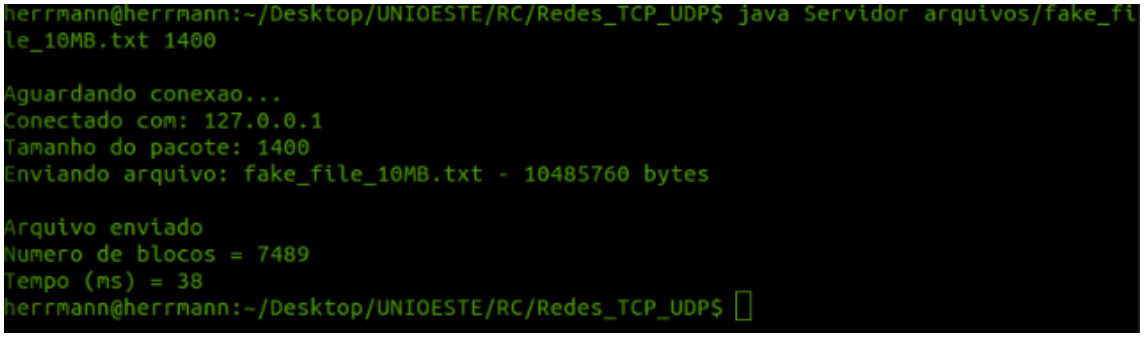

Figura 1. Execução Servidor TCP

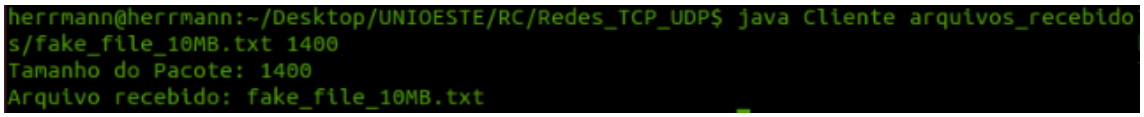

Figura 2. Execução Cliente TCP

Um exemplo de execução Cliente-Servidor UDP é mostrado nas Figuras

O servidor UDP mostra ainda uma verificação de envio de pacotes ao cliente, informando respectivamente o ponteiro do arquivo (que se encontra no final do arquivo), o tamanho do arquivo e o tamanho do último pacote.

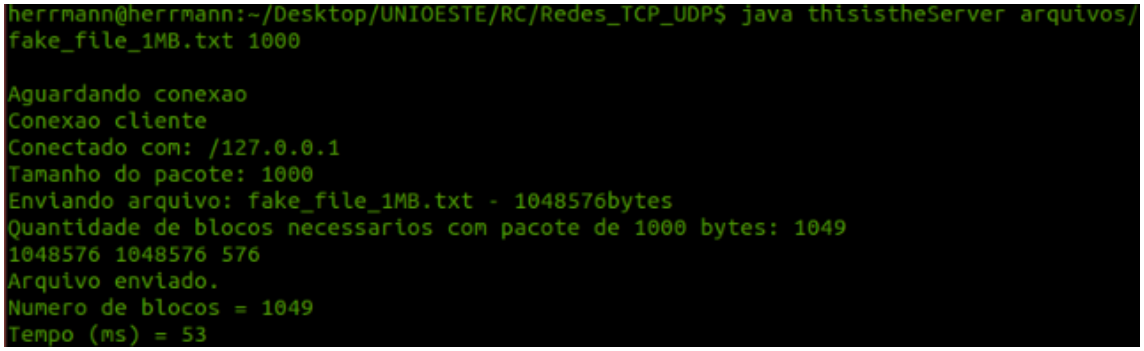

Figura 3. Execução Servidor UDP

Figura 4. Execução Cliente UDP

4. Resultados e Discussão

Para a análise foram tomados quatro arquivos de diferentes tamanhos: 1MB, 10MB, 100MB e 1GB. Para cada um, as transferências são feitas em pacotes de 100, 500, 1000 e 1400 bytes, correspondendo a um total de 16 execuções para cada protocolo. Cada uma destas execuções é repetida 30 vezes a fim de diminuir a porcentagem de erro máximo entre as saídas.

As amostras possuem intervalo de confiança de 95% e devem atingir um erro máximo de 10%. Caso esse valor seja superior, será realizado o cálculo do número de repetições necessárias para que seja atingido o erro máximo exigido. Os cálculos foram feitos de acordo com [Casas 2011].

São apresentados nas Tabelas 2 a 5 os valores da média, desvio padrão e o erro máximo de tempo, em milissegundos, de cada cenário (amostra) de execução do protocolo TCP.

Tabela 2. Resultados do Cliente-Servidor TCP – Arquivo 1MB

Pacote (bytes)	100	500	1000	1400
Média	31,7	15,3667	14,0334	8,4334
Desvio Padrão	4,3004	2,7852	3,2322	1,7357
Erro Máximo	4,21%	5,62%	7,14%	6,38%

Tabela 3. Resultados do Cliente-Servidor TCP – Arquivo 10MB

Pacote	100	500	1000	1400
Média	178,3667	65,8334	48,1	38,9
Desvio Padrão	14,7799	4,6984	5,6038	3,2520
Erro Máximo	2,57%	2,21%	3,61%	2,59%

Todas as tabelas ilustram a diminuição do tempo da média do tempo de transferência conforme o aumento do tamanho do pacote. Isso ocorre, entre outros motivos, porque o tempo que o

Tabela 4. Resultados do Cliente-Servidor TCP – Arquivo 100MB

Pacote	100	500	1000	1400
Média	1369,9334	469,4667	357,8667	333
Desvio Padrão	72,3235	17,8475	7,8772	9,1199
Erro Máximo	1,64%	1,18%	0,68%	0,85%

Tabela 5. Resultados do Cliente-Servidor TCP – Arquivo 1GB

Pacote	100	500	1000	1400
Média	16712,3333	12244,9667	12207,3333	11932,6667
Desvio Padrão	774,0053	854,1280	837,7103	760,8635
Erro Máximo	1,44%	2,16%	2,13%	1,98%

protocolo TCP leva pra executar suas funções tem certa relevância no tempo total da transferência, assim, quanto mais pacotes são enviados, mais tempo é gasto nestas execuções. Além disso, o erro máximo tem uma diminuição significativa entre a transferência dos arquivos de 1MB e 1GB, isso porque o desvio padrão se torna insignificante conforme o tamanho dos arquivos aumenta. Por exemplo, no Arquivo 1MB com pacotes de 500 bytes, o desvio padrão é um número proporcionalmente próximo da média. Já o Arquivo 100MB com pacotes de 1000 bytes possui um desvio padrão insignificante se comparado à média, ou seja, os valores de tempo se encontram próximos um do outro se analisar um desvio padrão de 7,8ms a uma média de 357,8667ms.

Não foi possível obter resultados confiáveis com o protocolo UDP. O servidor UDP obteve sucesso ao enviar todos os pacotes ao cliente, porém o cliente para de responder. Algumas possíveis causas encontradas para isso são em relação (i) às entradas na tabela NAT, que podem ter sido perdidas no momento em que o servidor termina de enviar os pacotes e; (ii) perda de conexão do cliente com a porta do servidor. Testou-se aplicar um valor de TimeOut de 5 segundos, mas nenhuma das vezes o envio foi realizado.

Em algumas execuções com arquivos de tamanhos pequenos (195 bytes, 1MB, 10MB) o cliente foi capaz de receber todos os pacotes, porém em execuções seguintes o mesmo não ocorreu, fazendo com que os resultados não se tornassem válidos. Em arquivos grandes (100MB, 1GB) o número de bytes transferidos ficou próximo ao tamanho do arquivo. Mesmo em uma das execuções de 10MB (10.485.760 bytes) o cliente recebeu 10.468.000 bytes.

5. Conclusão

O protocolo TCP oferece muita facilidade na transferência de arquivos, pois lida com garantia de entrega fazendo com que o usuário saiba que todos os pacotes serão entregues, mesmo que a transferência não seja tão rápida. Seu desempenho melhora ao aumentar o tamanho dos pacotes para transferência, pois menos tempo é gasto para executar suas funções devido ao menor número de blocos. Como não foi possível comparar o desempenho entre os protocolos devido aos problemas encontrados nas execuções do UDP, não pode-se inferir através da análise de dados práticos se o UDP continuaria ou não tendo mais velocidade dada a garantia de entrega.

Referências

Casas, P. H. B. L. (2011). Métodos quantitativos para ciência da computação experimental. `http://homepages.dcc.ufmg.br/~pedro.lascasas/metq2.pdf`.

Kurose, J. F. and Ross, K. W. (2013). *Redes de Computadores e a Internet: uma abordagem top-down*. Pearson Education do Brasil, 6 edition.

RFC 739 (2017). Transmission control protocol – tcp. `https://tools.ietf.org/html/rfc793`. Acesso: 14 de setembro de 2017.

RFC 768 (2017). User datagram protocol – udp. `https://www.ietf.org/rfc/rfc768.txt`. Acesso: 14 de setembro de 2017.