

# Redes de Computadores

## Trabalho Prático 3

Carina Capelão de Oliveira  
Carolina Coimbra Vieira

Dezembro de 2017

### 1 Introdução

O presente trabalho objetiva apresentar e analisar a metodologia e os resultados da implementação de um par de programas que operam no modelo cliente-servidor e exercitam tanto a transmissão unidirecional quanto a comunicação do tipo requisição-resposta sobre o protocolo UDP, implementando um protocolo de janela deslizante. Para isso, faz-se necessário utilizar a biblioteca de *sockets* do Unix (Linux).

A tecnologia cliente-servidor é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes). Os processos cliente enviam pedidos para o processo servidor e este, por sua vez, processa e envia os resultados dos pedidos [2]. Nesse caso, em particular, o cliente será responsável por enviar o nome do arquivo que o servidor deverá enviar para ele, dividindo em *buffers* de tamanho determinado a priori.

Os *sockets* são uma abstração de endereços de comunicação que consistem de um nome de *host* e um número de porta. Os *sockets* surgiram no sistema operacional BSD Unix e, atualmente, são responsáveis pela interação e comunicação de programas ao longo da Internet. Além disso, os *sockets* são a interface padrão para a comunicação entre processos em redes TCP/IP [4], sobretudo por fornecer uma interface para a camada de aplicação se comunicar com a camada de transporte dessa arquitetura. Ou seja, os *sockets* UDP e TCP são a interface provida pelos respectivos protocolos na interface da camada de transporte [1] e, programar com *sockets* pode ser visto como desenvolver um protocolo de aplicação.

A comunicação entre cliente-servidor se dá sobre o protocolo UDP (*User Datagram Protocol*). Esse protocolo não é orientado para a conexão e é considerado muito simples, uma vez que não fornece controle de erros [3]. Dessa forma, caso garantias sejam necessárias, é preciso implementar uma série de estruturas de controle, tais como *timeouts*, retransmissões, reconhecimento ou

controle de fluxo. Nesse caso, a confirmação dos pacotes, inexistente no protocolo UDP, deverá ser implementada no trabalho utilizando o conceito de janelas deslizantes.

Nos protocolos *stop-and-wait*, por exemplo, para conseguir confiabilidade, o emissor aguarda a confirmação de cada pacote enviado e só então envia o próximo pacote. Nesse caso, a rede permanece ociosa durante este tempo. Ao utilizar o protocolo de janela deslizante, a transmissão de pacotes se torna mais eficiente, uma vez que vários pacotes podem ser enviados antes da confirmação. Isso torna o processo de comunicação ainda mais complexo. O protocolo de janelas deslizantes é usado para a entrega confiável e ordenada de mensagens, ou seja, ele garante que todas as mensagens enviadas são entregues aos destinatários integralmente e na ordem correta de envio.

Dessa forma, a implementação do programa deve levar em consideração todas as especificações acima. Primeiro dois programas principais são criados, bem como a criação dos *sockets* em cada um deles. Após isso, ocorre a troca de dados que foi implementada de forma confirmada sobre o protocolo UDP. Para isso, um cabeçalho foi criado a fim de definir o número de sequência e número de reconhecimento de cada um dos pacotes enviados. Além disso, um temporizador foi acionado a cada mensagem enviada para que ocorra a retransmissão dos pacotes quando ocorrer um evento de *timeout*. E, por fim, várias análises e testes são realizados para avaliar o desempenho dessa comunicação variando o tamanho do *buffer* e do arquivo que será enviado, o impacto de um canal de transmissão com erros sobre o tempo e o desempenho dessa comunicação e, principalmente o impacto do tamanho da janela deslizante utilizada no tempo de comunicação e *throughput*.

O trabalho se organiza da seguinte forma. Primeiro, na Seção 2 é apresentado o cabeçalho do protocolo, detalhes da implementação do protocolo de janela deslizante, decisões de projeto envolvidas e o tratamento de temporizações. Na Seção 3 é apresentada a metodologia contendo dados sobre os experimentos como a configuração da máquina utilizada, a localização da mesma na rede além da descrição de como foram realizadas as medições. Então, na Seção 4 são apresentados os resultados obtidos após a realização de diversos experimentos. A análise mais aprofundada sobre cada um dos experimentos é descrita na Seção 5. Finalmente, na Seção 6, é feita uma breve conclusão sobre o trabalho.

## 2 Implementação

A implementação consiste de um par de programas que operam no modelo cliente-servidor com comunicação via protocolo UDP, utilizando uma biblioteca de *sockets* do Unix. Entretanto, mesmo utilizando o UDP que, não garante a entrega das mensagens, essa garantia deverá ser implementada por meio de janelas deslizantes. Sendo assim, os programas cliente e servidor são implementados seguindo um padrão simples, conforme descrito a seguir:

#### **Cliente:**

```
processa argumentos da linha de comando:
host_do_servidor porto_servidor nome_arquivo tam_buffer
chama gettimeofday para tempo inicial
envia string com nome do arquivo (terminada em zero)
abre arquivo que vai ser gravado - pode ser fopen(nome,"w+")
loop recv buffer até que perceba que o arquivo acabou
escreve bytes do buffer no arquivo (fwrite)
atualiza contagem de bytes recebidos
fim loop
fecha arquivo
chama gettimeofday para tempo final e calcula tempo gasto
imprime resultado:
"Buffer = %5u byte(s), %10.2f kbps (%u bytes em %3u.%06u s)
fim cliente.
```

#### **Servidor:**

```
processa argumentos da linha de comando:
porto_servidor tam_buffer
faz abertura passiva e aguarda conexão
recebe o string com nome do arquivo
abre arquivo que vai ser lido - pode ser fopen(nome,"r")
se deu erro, fecha conexão e termina
loop lê o arquivo, um buffer por vez até fread retornar zero
envia o buffer lido
se quiser, contabiliza bytes enviados
fim loop
fecha arquivo
chama gettimeofday para tempo final e calcula tempo gasto
se quiser, imprime nome arquivo e no. de bytes enviados
fim servidor.
```

De forma resumida, o cliente deve enviar um *string* para o servidor com o nome do arquivo desejado, receber o arquivo um *buffer* de cada vez e salvar os dados no disco à medida que eles chegam. Quando não houver mais bytes para ler, o cliente fecha o arquivo e gera uma linha com os dados da execução. O servidor por sua vez deve operar de forma complementar, conforme exemplificado na Figura 1.

A comunicação entre esses processos está representada, de forma geral, na Figura 1. Nela, é possível perceber que, as estruturas são criadas e, ao contrário do TCP, não ocorre o estabelecimento da conexão. Após a criação dos *sockets*, os mesmos já podem iniciar a comunicação. Nesse caso, cada um dos processos deve especificar nas funções de envio e recebimento o interlocutor que desejam alcançar.

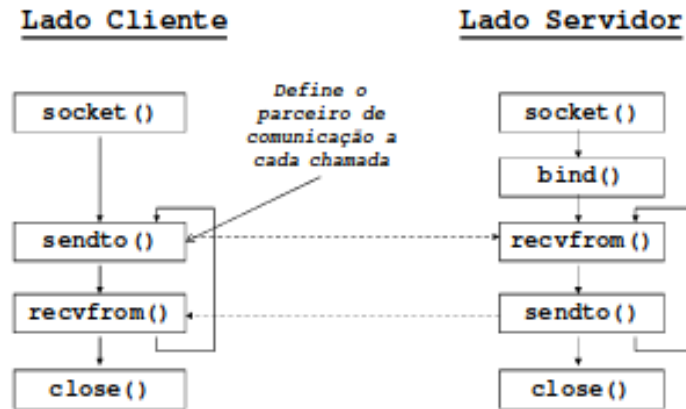


Figura 1: Diagrama cliente-servidor UDP

A maior dificuldade, no entanto, é garantir que as mensagens serão confirmadas mesmo utilizando o protocolo UDP. Dessa forma, foi necessário adicionar um cabeçalho extra em todas as mensagens trocadas entre cliente e servidor para adicionar o número de sequência da mensagem, bem como o número de reconhecimento. Essa alteração é necessária tendo em vista que o cabeçalho do protocolo UDP é bem simplificado, conforme mostrado na Figura 3, uma vez que não há garantias de entrega. Como referência a um protocolo que garante a entrega das mensagens foi analisado o TCP que, devido prover esse tipo de serviço, possui um cabeçalho mais completo. Sendo assim, analisando e comparando os cabeçalhos, chegou-se à conclusão de que adicionando os campos de número de sequência e número de reconhecimento seria suficiente para confirmar as mensagens trocadas.

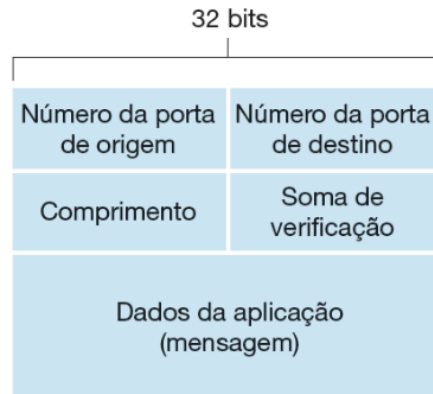


Figura 2: Cabeçalho do protocolo UDP

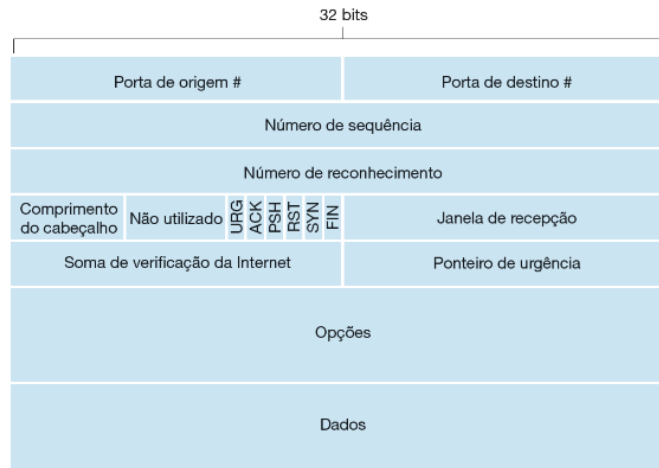


Figura 3: Cabeçalho do protocolo TCP

No que diz respeito ao cabeçalho das mensagens, o número de sequência e reconhecimento foram incorporados nas posições iniciais do vetor de dados. Esses valores são números inteiros e sequenciais, incrementados de um em um na medida em que os pacotes são confirmados pelo programa destino. Por se tratar de um valor que ocupa uma posição em um vetor de caracteres, cada número é representado por um byte (8 bits). Dessa forma, a cada vez que eram incrementados, foi necessário fazer o módulo desses valores por 127, a fim de se evitar um *overflow*. Sendo assim, a cada pacote recebido, o cliente, por exemplo, enviava um pacote de reconhecimento, incrementando sempre em uma unidade o campo que representa o número de sequência, bem como de reconhecimento. De forma complementar, a cada novo pacote enviado pelo servidor, o número de sequência do mesmo era incrementado. Além disso, por se tratar de um canal não confiável, é necessário tratar possíveis perdas de pacotes. Sendo assim, um temporizador é ativado pelo servidor com duração de 1 segundo para que ocorra um evento de *timeout* e o pacote seja reenviado para o cliente caso não tenha sido confirmado.

A implementação do cabeçalho com o número de sequência e número de reconhecimento como valores inteiros e sequenciais sobre o protocolo do tipo janela deslizante, baseou-se no princípio de janela deslizante. Esse protocolo é uma alternativa mais eficiente quando comparado ao protocolo *stop-and-wait*. O protocolo *stop-and-wait* possui um problema de desempenho, visto que o canal de comunicação fica ocioso enquanto não chegar uma confirmação do receptor. Para tal situação, o protocolo de de janela deslizante *go-back-n* possibilita o transmissor enviar  $n$  quadros antes que o primeiro seja confirmado.

O protocolo cria uma janela de tamanho fixo de forma a transmitir todos os pacotes que se encontram nesta janela antes de receber uma confirmação. Um pacote é dito não-confirmado se foi enviado e não recebeu nenhuma confirmação do cliente. Quando o remetente recebe uma confirmação para o primeiro pacote

da janela, ele desliza a janela e envia o próximo pacote. A janela continua deslizando de acordo com a chegada das confirmações, conforme representado na Figura 4.

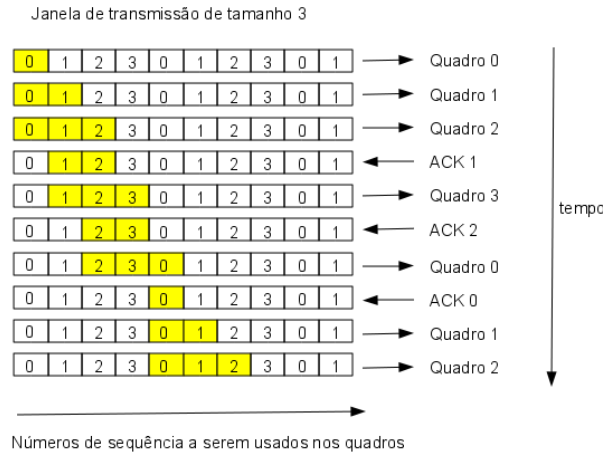


Figura 4: Exemplo de janela deslizante do tipo *Go-back-n*

Além disso, todos os quadros enviados após um quadro com erro, serão descartados pelo receptor. Ou seja, ele simplesmente descarta todos os quadros subsequentes ao erro e não envia qualquer confirmação desses quadros descartados. Em outras palavras, a camada de enlace de dados se recusa a aceitar qualquer quadro, exceto o próximo quadro que ela tem de entregar à camada de rede. Se a janela do transmissor for totalmente preenchida antes do *timer* encerrar a contagem, o *buffer* começará a se esvaziar. Consequentemente, o transmissor interromperá a transmissão e retransmitirá todos os quadros não confirmados em ordem, começando pelo quadro danificado ou perdido. Convém salientar que essa abordagem poderá desperdiçar uma grande quantidade de largura de banda se a taxa de erros do canal for alta. Essa situação está representada na Figura 5.

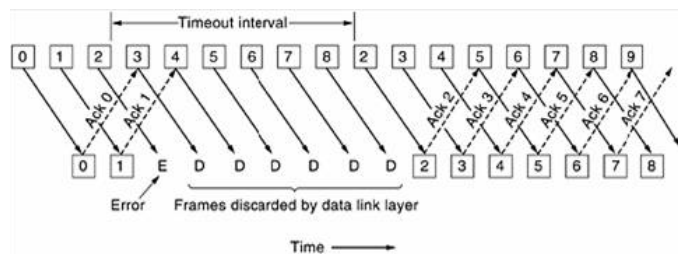


Figura 5: Exemplo de reenvio de pacotes usando o protocolo de janela deslizante do tipo *Go-back-n*

Na Figura 5 temos o caso em que a janela do receptor é grande. Os quadros 0 e 1 são corretamente recebidos e confirmados. Porém, o quadro 2 está danificado ou perdido. O transmissor, desavisado desse problema, continua a enviar quadros até expirar o temporizador correspondente ao quadro 2. Em seguida, ele volta até o quadro 2 e começa tudo de novo a partir dele, enviando mais uma vez os quadros 2, 3, 4 etc. Já na Figura 6 é possível ter uma noção da perspectiva do transmissor e receptor quanto ao protocolo de janela deslizante.

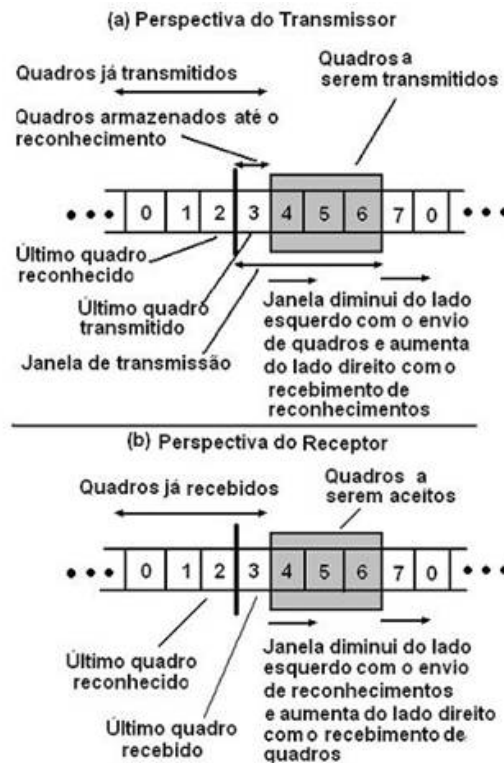


Figura 6: Exemplo de janela deslizante do tipo *Go-back-n*

Para simular a não entrega dos pacotes do protocolo UDP, a função "tp\_rcvfrom" disponibilizada para uso foi modificada, em que foi inserido na função um código que gera um número aleatório entre 0 e 50, e, com uma certa probabilidade definida manualmente, a função não recebe os dados. Ou seja, utilizando a função "tp\_rcvfrom", depois de modificada, o processo tem uma probabilidade  $p$  de receber ou não o pacote. Além disso, foi mantida uma versão original da função "tp\_rcvfrom" que garante o recebimento, chamada de "tp\_rcvfrom\_confivel".

Além disso, uma decisão tomada nesse projeto foi considerar que o envio do nome do arquivo pelo cliente se dará de forma segura, utilizando a função "tp\_rcvfrom\_confivel". Ou seja, uma função de recebimento seguro foi uti-

lizada a fim de garantir que, com certeza, o servidor irá receber o nome do arquivo enviado pelo cliente. Essa mensagem inicial representaria uma "falsa conexão" entre o par de programas. A troca de mensagens após esse envio do nome do arquivo se dá sobre um canal com probabilidade  $p$  de haver erros e os pacotes não serem recebidos. E, por fim, para encerrar a comunicação entre cliente e servidor, foi assumido que o recebimento do reconhecimento do fim da transmissão do arquivo por parte do servidor também será garantida, utilizando tal função. Sendo assim, o início e o fim da transmissão são assumidos como pacotes que, garantidamente, chegarão ao seu destino.

Em relação às decisões de projeto relacionadas à janela deslizante, deve-se definir o tamanho da janela deslizante utilizando a variável global TAMANHO\_JANELA. A janela foi implementada como uma matriz, em que o número de linhas da matriz é o tamanho da janela definida, e cada linha tem o tamanho de tam\_buffer passado como parâmetro. Depois de recebido o nome do arquivo, o servidor lê do arquivo requisitado pelo cliente uma certa quantidade de bytes definida por tam\_buffer, cria um pacote, insere o pacote na janela (caso tenha espaço) e envia para o cliente. O servidor repete esse processo até que não exista mais espaço na janela. Quando isso acontecer, o servidor fica aguardando o recebimento do ack enviado pelo cliente. Se o servidor não receber nenhum ack, ele reenvia todos os pacotes existentes na janela. Quando o servidor receber um ack, ele checa se o ack possui um ID igual ao ID que ele estava esperando, que no caso seria igual ao ID (número de sequência) do primeiro pacote inserido na janela. Se sim, ele libera um espaço na janela e insere o novo pacote que ele tinha criado para depois enviar. A janela utiliza deslocamento. Se ele recebe um ack que não estava esperando, como por exemplo o ack do terceiro pacote inserido, ele libera da janela todos os pacotes inseridos até o pacote que possui o ID igual ao ack recebido, insere o novo pacote na janela e envia o pacote. Na perspectiva do cliente, se ele recebe um pacote que possui um número de sequência (ID) que ele não estava esperando, ou seja, que não seja igual ao último recebido + 1, ele envia um ack com um número de sequência igual ao número que ele está esperando. Caso ele receba um pacote com um ID que ele está esperando, ele apenas envia um ack com um número de sequência igual ao do pacote recebido.

O temporizador, por sua vez, baseou-se no conceito de temporização por tratamento de sinal. Dessa forma, para cada nova mensagem enviada pelo servidor durante o envio dos dados do arquivo, um temporizador era inicializado de forma que, no caso de um *timeout* e um não recebimento de uma mensagem de confirmação do cliente, o servidor deveria reenviar o último pacote enviado ao cliente. Foi definido um tempo de espera igual a 1 segundo, ou seja, se em 1 segundo, o servidor não recebe uma confirmação do cliente, a partir da sua última mensagem enviada ao cliente ocorre o reenvio. Um evento de *timeout* pode estar associado a alguns fenômenos, sendo que as causas mais comuns são: perda do pacote enviado, perda do pacote de confirmação recebido ou temporização prematura e, em ambos os casos, o evento de *timeout* faz com que o servidor reenvie a partir do último pacote confirmado, quantas vezes for necessário.



Sendo assim, a implementação se resume em utilizar um cabeçalho com número de sequência e de reconhecimento de pacotes aliado ao conceito de temporização para garantir a confirmação durante a troca de mensagem entre cliente e servidor. Toda a implementação levou em consideração os conceitos de janela deslizante. As configurações dos testes e máquinas usadas para testar essa implementação são apresentadas na seção seguinte.

### 3 Metodologia

A metodologia apresentada se refere à implementação e testes de um par de programas que operam no modelo cliente-servidor com comunicação via protocolo UDP, utilizando uma biblioteca de *sockets* do Unix. Entretanto, mesmo utilizando o UDP que, não garante a entrega das mensagens, essa garantia foi implementada, conforme descrito na seção anterior.

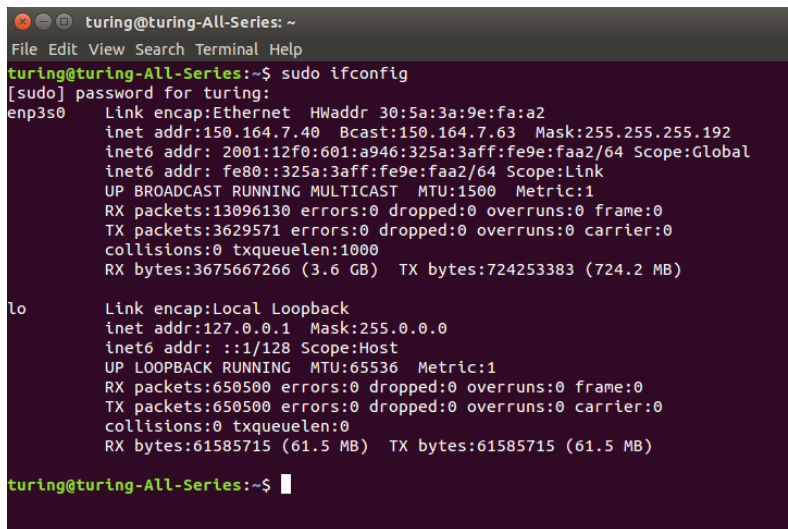
Todos os testes a serem descritos foram executados utilizando a mesma máquina, cujas informações principais referentes ao poder de processamento, memória e placa de rede são descritas a seguir:

Mémoria RAM: 16GB

Processador: Intel(R) Core(TM) i5-4440 CPU @ 3.10GHz

Placa de rede: 03:00:0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)

E, por fim, a máquina está localizada no laboratório *WISEMAP* do Departamento de Ciência da Computação na UFMG e possui o seguinte endereço IP: 150.164.7.40. Na Figura 7 é possível visualizar algumas configurações de rede da máquina.



```
turing@turing-All-Series: ~  
File Edit View Search Terminal Help  
turing@turing-All-Series:~$ sudo ifconfig  
[sudo] password for turing:  
enp3s0    Link encap:Ethernet  HWaddr 30:5a:3a:9e:fa:a2  
          inet addr:150.164.7.40  Bcast:150.164.7.63  Mask:255.255.255.192  
          inet6 addr: 2001:12f0:601:a946:325a:3aff:fe9e:faa2/64 Scope:Global  
          inet6 addr: fe80::325a:3aff:fe9e:faa2/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:13096130 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:3629571 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:3675667266 (3.6 GB)  TX bytes:724253383 (724.2 MB)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:650500 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:650500 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:61585715 (61.5 MB)  TX bytes:61585715 (61.5 MB)  
  
turing@turing-All-Series:~$
```

Figura 7: Especificações da rede do computador utilizado nos testes

O servidor irá rodar na máquina, ou seja, ele usa o IP da máquina e, é necessário definir a porta através da qual o cliente e o servidor irão se comunicar. Já para o cliente, é necessário especificar o IP onde o servidor está rodando e a porta correspondente. Dado essas especificações, a comunicação pode ser exemplificada conforme mostra a Figura 8.

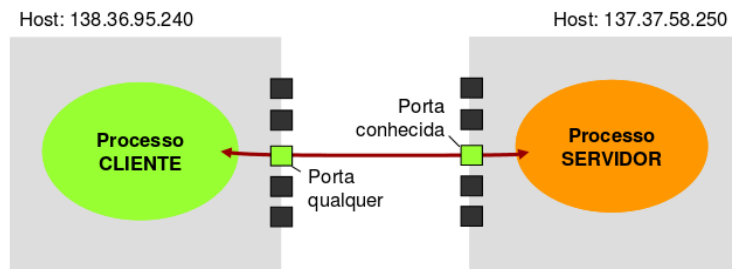


Figura 8: Diagrama representando como se dá a comunicação entre o cliente e servidor dado as especificações do IP e a porta de comunicação com o servidor

O servidor deve ser o primeiro a ser inicializado e, a partir de então, ele estará sempre "escutando" a porta. Ao contrário do protocolo TCP, o servidor UDP não aceita conexões assim como o cliente UDP não se conecta com o servidor. O que ocorre é que o servidor recebe diretamente as mensagens do cliente, assim como o servidor envia mensagens diretamente ao cliente. Ou seja, a comunicação do cliente com o servidor ocorre sem o estabelecimento de uma conexão a priori. Os parâmetros que o cliente recebe são: *host* do servidor (`argv[1]`), número da porta (`argv[2]`), nome do arquivo (`argv[3]`) e o tamanho do *buffer* (`argv[4]`) enquanto que o servidor recebe o número da porta (`argv[1]`) e o tamanho do *buffer* (`argv[2]`).

Vale ressaltar que, após o envio do nome do arquivo do cliente para o servidor, o servidor é responsável por abrir e enviar o mesmo para o cliente, *buffer* por *buffer*. Ao receber, o cliente salva esse conteúdo em um outro arquivo de extensão ".out". Dessa forma, ambos os arquivos, terão, ao fim da comunicação, o mesmo conteúdo, podendo ser verificado pela função *md5sum*.

Os testes sobre essa comunicação cliente-servidor, foram executados medindo o tempo decorrido desde o envio da primeira mensagem do cliente para o servidor até a finalização da comunicação, após o servidor enviar todo o arquivo solicitado pelo cliente. O *throughput* também foi medido nesses casos, ou seja, a taxa de transferência obtida entre cliente e servidor (calculado como o número total de bytes enviados dividido pelo tempo medido no cliente). Sendo assim, o mesmo código foi executado repetidas vezes variando alguns parâmetros como o tamanho do arquivo, tamanho do *buffer*, probabilidade de erros no canal de comunicação e, principalmente, o tamanho da janela deslizante. Os testes estão divididos em, basicamente, 4 tipos, conforme serão resumidos a seguir e melhor explicados e analisados nas Seções 4 e 5.

- **Teste 1:** Nesse teste, foi fixado um arquivo de 3MB, uma *probabilidade* de 2% de erro, uma janela deslizante de tamanho 16 e variando o tamanho do *buffer* entre 100 bytes, 1.000 bytes e 4.000 bytes. Esse teste foi feito com o intuito de analisar como o tempo e o *throughput* variam em relação ao tamanho do *buffer*.
- **Teste 2:** Nesse teste, foi fixado um *buffer* de 4.000 bytes, uma probabilidade de erro de 2%, uma janela deslizante de tamanho 16 e variando o tamanho dos arquivos de 3MB, 6MB e 9MB. Com esse teste, pode se analisar como o tamanho do arquivo influência no tempo de execução.
- **Teste 3:** Nesse teste, foi fixado um tamanho de *buffer* de 4.000 bytes, um arquivo de 3MB, uma janela deslizante de tamanho 16 e variando a probabilidade de ocorrer erro na entrega do pacote, em que essas probabilidades variaram entre 0%, 25%, 50% e 75%. Esse teste foi feito com o intuito de analisar como o tempo e o *throughput* variam em relação a mais ou menos erros na comunicação a serem tratados.
- **Teste 4:** Nesse teste, foi fixado um *buffer* de 4.000 bytes, uma probabilidade de erro de 2%, um arquivo de 3MB e variando o tamanho da janela deslizante entre 2, 4, 8 e 16. Com esse teste, pode se analisar como o tamanho da janela deslizante influência no tempo de execução.

## 4 Resultados

Os principais experimentos executados após a implementação do par cliente-servidor envolveram testes de desempenho medindo o tempo que leva a comunicação e completa transferência do arquivo. As hipóteses levantadas sugerem que esse desempenho é afetado principalmente com a variação do tamanho do arquivo que se deseja transferir, o tamanho do *buffer* através do qual o mesmo será transferido, a probabilidade de haver perdas de pacotes durante a transferência e, principalmente o tamanho da janela deslizante considerado.

No que diz respeito à relação entre o tamanho do *buffer* e o tempo de comunicação entre cliente-servidor, foram realizados testes variando o tamanho do *buffer* em: 100 bytes, 1.000 bytes e 4.000 bytes. Nesse teste, o tamanho do arquivo considerado era de 3MB, o tamanho da janela deslizante igual a 16 e para cada tamanho de *buffer* e foi assumida probabilidade de 2% de erro no canal. A Figura 9 representa o gráfico obtido com tempos encontrados para cada tamanho de *buffer* usado. O eixo y representa o tempo médio da troca de dados, medido em segundos, e o eixo x representa o tamanho do *buffer* utilizado.

A fim de facilitar a visualização, na Figura 10 apresenta o mesmo gráfico mostrado na Figura 9, porém, o eixo y representa o logaritmo do tempo médio da troca de dados. Dessa forma, é possível perceber que o aumento no tempo cresce de forma exponencial de acordo com a diminuição do tamanho do *buffer*. Isso porque, quanto menor o tamanho do *buffer*, mais envios deverão ser feitos pelo servidor e, conseqüentemente, maior será o tempo de execução.

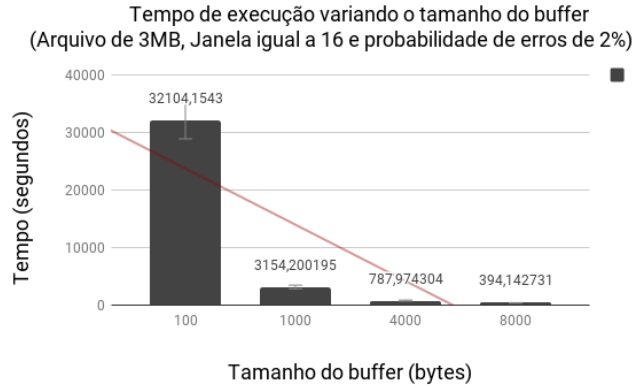


Figura 9: Gráfico representando o tempo médio de comunicação dado a variação no tamanho do *buffer* usado. Fixado um tamanho de arquivo igual a 3MB, um tamanho de janela deslizante igual a 16 e uma probabilidade de erros igual a 2% e variando o tamanho do *buffer* em: 100, 1.000 e 4.000 bytes.

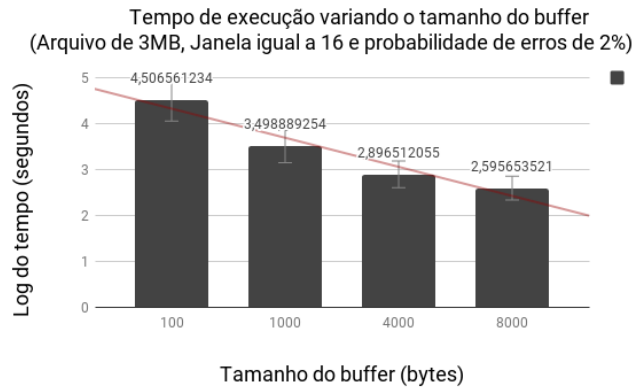


Figura 10: Gráfico representando o tempo médio de comunicação dado a variação no tamanho do *buffer* usado. Fixado um tamanho de arquivo igual a 3MB, um tamanho de janela deslizante igual a 16 e uma probabilidade de erros igual a 2% e variando o tamanho do *buffer* em: 100, 1.000 e 4.000 bytes.

Similarmente ao gráfico representado na Figura 9, a Figura 11 apresenta um gráfico com o *throughput* médio da comunicação, utilizando os dados do teste anterior. Porém, o eixo y representa agora a razão entre o número total de bytes enviados pelo tempo medido no cliente. Nesse gráfico, é possível visualizar o crescimento da taxa de envio com o aumento do tamanho do *buffer*, confirmando a influência do tamanho do *buffer* sobre a taxa de envio.

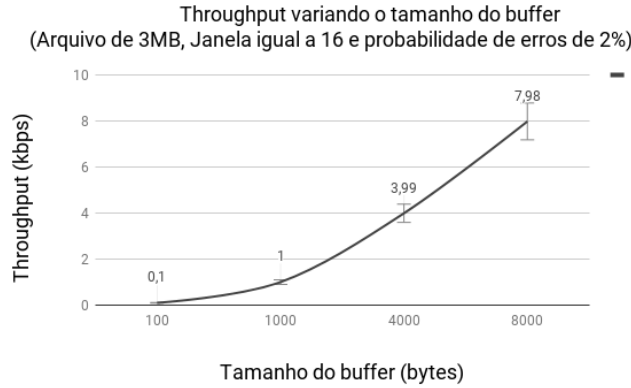


Figura 11: Gráfico representando o *throughput* dado a variação no tamanho do *buffer* usado. Fixado um tamanho de arquivo igual a 3MB, um tamanho de janela deslizante igual a 4 e uma probabilidade de erros igual a 2% e variando o tamanho do *buffer* em: 100, 1.000 e 4.000 bytes.

Outro fator que também influencia no tempo de comunicação é o tamanho do arquivo que se deseja enviar. Nesse caso, fixado um tamanho de *buffer* igual a 4.000 bytes, um tamanho de janela deslizante igual a 16 e uma probabilidade de erros igual a 2%, escolhidos de forma arbitrária, foram realizados testes para verificar a influência dos arquivos de tamanho 3MB, 6MB e 9MB no tempo de comunicação. A Figura 12 mostra o gráfico resultante desses testes realizados. Nela é possível se ter uma noção do quanto o tamanho do arquivo contribui para o aumento do tempo de troca de dados entre o cliente e o servidor. Nesse caso, percebe-se que essa relação se dá de forma linear. O eixo y representa o tempo médio da troca de dados, medido em segundos, e o eixo x representa o tamanho do arquivo enviado, em *megabytes* (MB).

Também é possível comparar a influência da probabilidade de erros, como por exemplo, perda de pacotes, sobre o tempo de comunicação, conforme mostra o gráfico da Figura 13. É possível perceber que, de fato, o aumento da probabilidade de erros influenciam muito no aumento do tempo de comunicação. O eixo y representa o tempo médio da troca de dados, medido em segundos, e o eixo x representa as probabilidade de erro consideradas, sendo elas: 0%, 25%, 50% e 75%.

Além da relação entre as probabilidades de erro e o tempo de comunicação entre cliente e servidor para a completa transferência do arquivo, a Figura 14 apresenta um gráfico com o *throughput* médio da comunicação, utilizando os dados do teste anterior. Porém, o eixo y representa agora a razão entre o número total de bytes enviados pelo tempo medido no cliente. Nesse gráfico, é possível visualizar o decréscimo da taxa de envio com o aumento das probabilidades de erro.

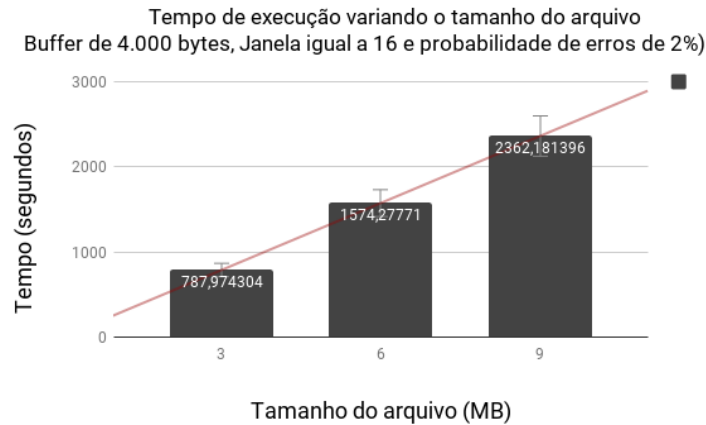


Figura 12: Gráfico representando o tempo médio de comunicação dado a variação no tamanho do arquivo a ser enviado. Fixado um tamanho de *buffer* igual a 4.000 bytes, um tamanho de janela deslizante igual a 16 e uma probabilidade de erros igual a 2% e variando o tamanho do arquivo em: 3, 6 e 9 MB.

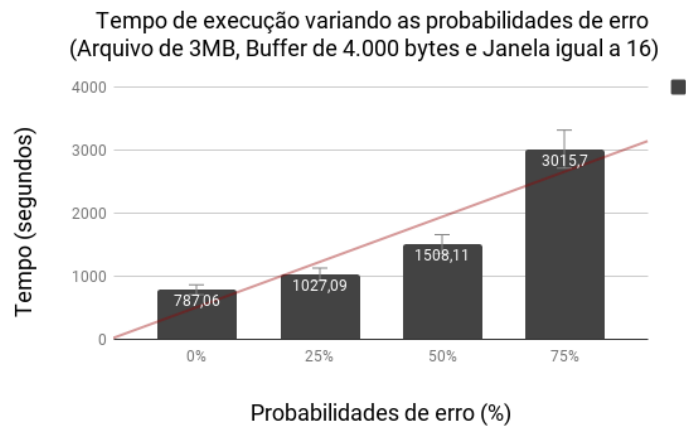


Figura 13: Gráfico representando o tempo médio de comunicação dado a variação na probabilidade de erros no canal. Fixado um tamanho de arquivo igual a 3MB, um *buffer* de 4.000 bytes, um tamanho de janela deslizante igual a 16 e variando as probabilidades de erro em: 0%, 25%, 50% e 75%.

Por fim, é possível comparar a influência do tamanho da janela deslizante sobre o tempo de comunicação, conforme mostra o gráfico da Figura 15. O eixo y representa o tempo médio da troca de dados, medido em segundos, e o eixo x representa os tamanhos de janela considerados, sendo eles: 2, 4, 8 e 16.

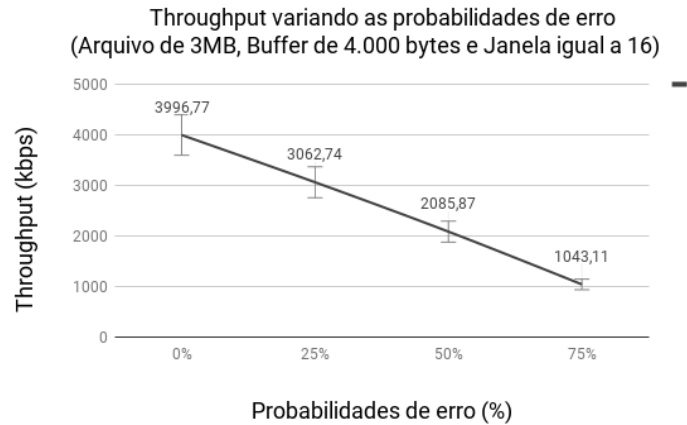


Figura 14: Gráfico representando o *throughput* dado a variação na probabilidade de erros no canal. Fixado um tamanho de arquivo igual a 3MB, um *buffer* de 4.000 bytes, um tamanho de janela deslizante igual a 16 e variando as probabilidades de erro em: 0%, 25%, 50% e 75%.

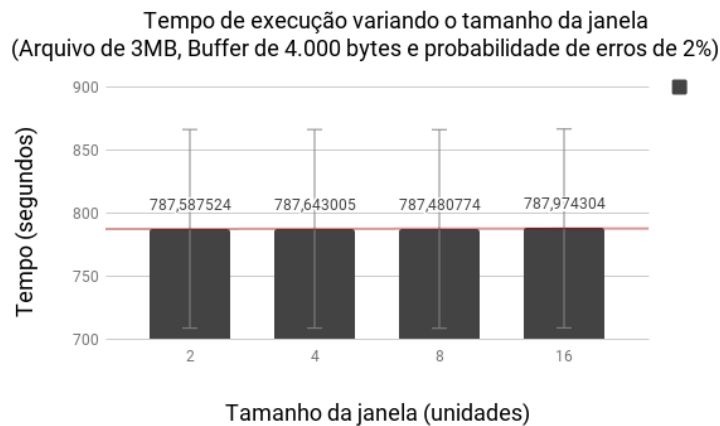


Figura 15: Gráfico representando o tempo médio de comunicação dado a variação no tamanho da janela deslizante. Fixado um tamanho de arquivo igual a 3MB, um *buffer* de 4.000 bytes, uma probabilidade de erros igual a 2% e variando o tamanho da janela deslizante em: 2, 4, 8, e 16.

A Figura 16 apresenta um gráfico com o *throughput* médio da comunicação, utilizando os dados do teste anterior. Porém, o eixo y representa agora a razão entre o número total de bytes enviados pelo tempo medido no cliente.

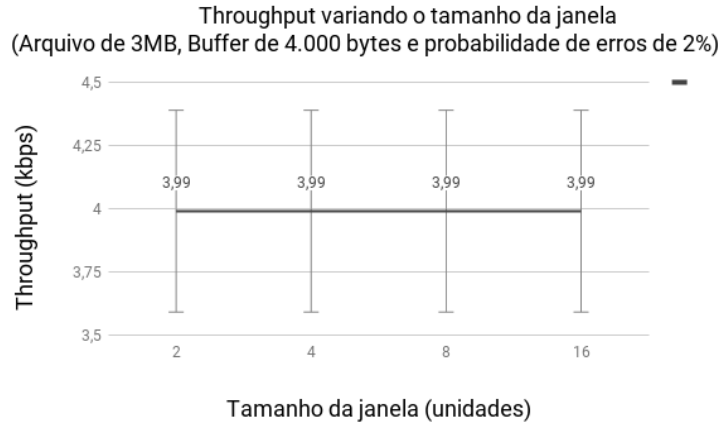


Figura 16: Gráfico representando o *throughput* dado a variação no tamanho da janela deslizante. Fixado um tamanho de arquivo igual a 3MB, um *buffer* de 4.000 bytes, uma probabilidade de erros igual a 2% e variando o tamanho da janela deslizante em: 2, 4, 8, e 16.

## 5 Análise

Após a execução dos experimentos apresentados na seção 4, é possível discutir um pouco sobre o que foi observado bem como comparar os resultados obtidos o aqueles que seriam esperados. Nessa seção, portanto, será feita uma análise referente a cada um dos quatro principais testes realizados.

Os resultados encontrados na Seção 4 refletem bem o que era esperado sobre essa comunicação. O fato do tempo médio de comunicação aumentar com a diminuição do tamanho do *buffer* mostra que, de fato, foi necessário fazer mais envios ao cliente, aumentando o tempo de comunicação. Esse resultado é sustentado também pelos testes em que mostram o *throughput* da comunicação, isto é, a taxa de transferência obtida entre cliente e servidor (basicamente, o número total de bytes enviados dividido pelo tempo medido no cliente) com a variação do tamanho do *buffer*.

Assim como o tamanho do *buffer*, o tamanho do arquivo que se deseja enviar ao cliente influencia no tempo de comunicação. Entretanto, essa relação é direta, uma vez que o aumento no tamanho do arquivo provoca um aumento no tempo que leva a fase de troca de dados. Esse resultado também foi comprovado pelos testes realizados.

Já os testes referentes à variação na probabilidade de erros no canal, comprovam que, de fato, quando maior a probabilidade de erros, maior será o tempo necessário para a completa transferência do arquivo. E, dessa forma, menor será o *throughput*. Esse resultado comprova a importância de um canal confiável para uma comunicação cliente-servidor, uma vez que a não garantia de um ca-



nal confiável aumenta consideravelmente o tempo de comunicação, reduzindo drasticamente o *throughput* ou desempenho da mesma. Entretanto, devido ao fato da maioria dos canais serem suscetíveis a erros, esses testes mostram que, para aplicações que são tolerantes a perdas, de fato, não vale a pena implementar um protocolo de confirmação de mensagens devido ao tempo que o mesmo acrescenta à comunicação. Porém, de forma inversa, é possível perceber que, para aplicações que exigem a entrega, é muito importante tratar essas perdas, mesmo que isso implique em um aumento nesse tempo.

Por fim, os testes variando o tamanho da janela foi surpreendente sob o ponto de vista de que as variações no tamanho da janela não impactaram da forma esperada no tempo de transmissão completa dos dados. O que esperávamos era uma diminuição nesse tempo na medida em que o tamanho da janela aumentava, uma vez que mais pacotes poderiam ser enviados de forma simultânea. Porém, essa hipótese não foi observada. Uma possível explicação, seria o fato de, por se tratar de um canal com erros, alguns pacotes foram perdidos e, consequentemente, o servidor teve que reenviá-los, aumentando assim o tempo de comunicação. Com isso, a diminuição do tempo com aumento no tamanho da janela era "anulada" pelo aumento dessa comunicação em casos de reenvio de pacotes.

## 6 Conclusão

O presente trabalho foi desenvolvido com o objetivo de implementar um par de cliente-servidor que se comunicam via *sockets* UDP, implementando uma comunicação usando o protocolo de janelas deslizantes. Todo o código foi desenvolvido em linguagem C, usando a biblioteca de *sockets*.

Os testes realizados comprovam hipóteses formuladas anteriormente, referentes à influência do tamanho do *buffer* e o tamanho do arquivo nessa comunicação. Nesse caso, conforme esperado, o aumento no tamanho do *buffer* reduz significativamente o tempo que demora a comunicação. Assim como o aumento no tamanho do arquivo está diretamente relacionado ao aumento no tempo necessário para a completa transferência do mesmo. E, além disso, o impacto da confirmação de pacotes, perdas ao longo da comunicação, *timeout*, retransmissão de mensagens e alterações no tamanho da janela deslizante.

De modo geral, todas as hipóteses levantadas foram comprovadas pelos testes realizados. E, o mais interessante é perceber o impacto negativo causado por um canal de comunicação com erros sobre o desempenho da comunicação cliente-servidor, bem como as influências do tamanho da janela deslizante nessa comunicação.

Por fim, o trabalho foi uma oportunidade de aprender e conviver com problemas práticos que são vividos no dia-a-dia e enfrentados no mercado de trabalho e que nos ensinam a ser autodidatas e transpor barreiras por necessidade e desejo próprio. E, de fato, após muita pesquisa e aprendizado, é possível desenvolver um trabalho bem estruturado e com resultados interessantes e consistentes.

## Referências

- [1] Cesar Augusto Tacla. Sockets udp, tcp e multicast. Slides de aula.
- [2] UFRGS. Protocolos cliente/servidor. Página Web.
- [3] USP. Socket udp. Slides de aula.
- [4] Marcos Augusto M. Vieira. Sockets. Slides de aula.