

Universidade Federal de São João del-Rei

Implementação de uma aplicação cliente-servidor sobre o protocolo UDP

Isabella Vieira Ferreira
Lucivânia Ester da Costa
Mônica Neli de Resende

15 de dezembro de 2015

Sumário

1	Introdução	3
2	Funcionamento de uma aplicação cliente-servidor sobre UDP	4
3	Implementação do Cliente	5
3.1	Rotinas do Cliente	6
4	Implementação do Servidor	7
4.1	Rotinas do Servidor	8
5	Resultados	10
5.1	Envio normal de dados	10
5.2	Envio com perda de dados	11
5.3	Enviado de dados duplicados	12
5.4	Enviado de dados embaralhados	13
6	Conclusão	14
7	Referências Bibliográficas	15

Lista de Figuras

1	Fluxo de Dados com Socket	4
2	ACK	4
3	Fluxograma do Cliente	5
4	Fluxograma do Servidor	8
5	Cliente para envio normal de dados	10
6	Servidor para envio normal de dados	11
7	Cliente para envio com perda de dados	12
8	Servidor para envio com perda de dados	12
9	Cliente para envio de dados duplicados	13
10	Servidor para envio de dados duplicados	13
11	Cliente para envio de dados embaralhados	14
12	Servidor para envio de dados embaralhados	14

1 Introdução

O UDP (*User Datagram Protocol*) é um protocolo da camada de transporte. Com este protocolo, não se pode garantir que o pacote irá chegar ou não. Para fazer o controle dos dados, pode-se implementar estruturas de controle tais como *timeouts*, retransmissões, *acknowledgments*, controle de fluxo etc. O UDP é um serviço sem conexão, pois não há necessidade de manter um relacionamento entre o cliente e o servidor. Sendo assim, um cliente UDP pode criar um socket, enviar um datagrama para um servidor e imediatamente enviar outro datagrama com o mesmo socket para um servidor diferente. Da mesma forma, um servidor poderia ler datagramas vindos de diversos clientes, usando um único socket.

Este trabalho tem como objetivo desenvolver uma aplicação cliente-servidor sobre UDP. O cliente tem uma interface que permite que o usuário escolha se quer perder, duplicar ou embaralhar os dados. Se a opção é embaralhar ou duplicar dados, deve ser adotada uma estratégia aleatória para escolher quais dados serão embaralhados ou duplicados. Se a opção é perder dados, o servidor deve fazer um pedido explícito de retransmissão. Além disso, o servidor deve imprimir os dados que recebeu (com dados perdidos, duplicados e fora de ordem) bem como os dados corretos, com todos os problemas resolvidos. A Seção 2 apresenta a implementação de tais estratégias e a Seção 5 apresenta os resultados obtidos.

2 Funcionamento de uma aplicação cliente-servidor sobre UDP

Uma aplicação stop-and-wait orientada a datagramas funciona como a Figura 6. Quando o aplicativo inicia, um socket é aberto tanto para o cliente quanto para o servidor. O cliente então faz a requisição dos dados para o servidor. Quando o servidor recebe tal requisição, ele envia a resposta para o cliente com os dados solicitados. O cliente recebe então os dados e envia um ACK (*Acknowledgment*) confirmando o recebimento do pacote solicitado. Quando todos os pacotes requisitados pelo cliente são enviados pelo servidor, os sockets são fechados e a aplicação encerra.

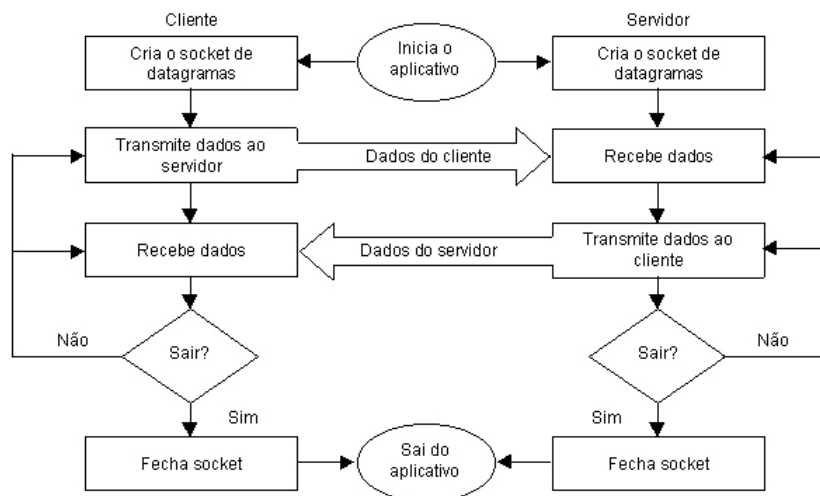


Figura 1: Fluxo de Dados com Socket

A Figura 2 apresenta como os ACKs são enviados assim que um pacote requisitado pelo cliente é enviado pelo servidor.

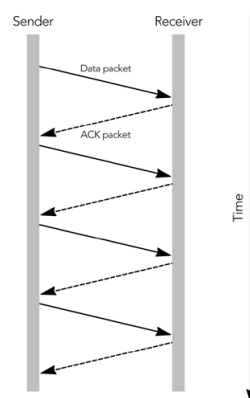


Figura 2: ACK

3 Implementação do Cliente

A Figura 3 apresenta um fluxograma com a ordem da execução do Cliente.

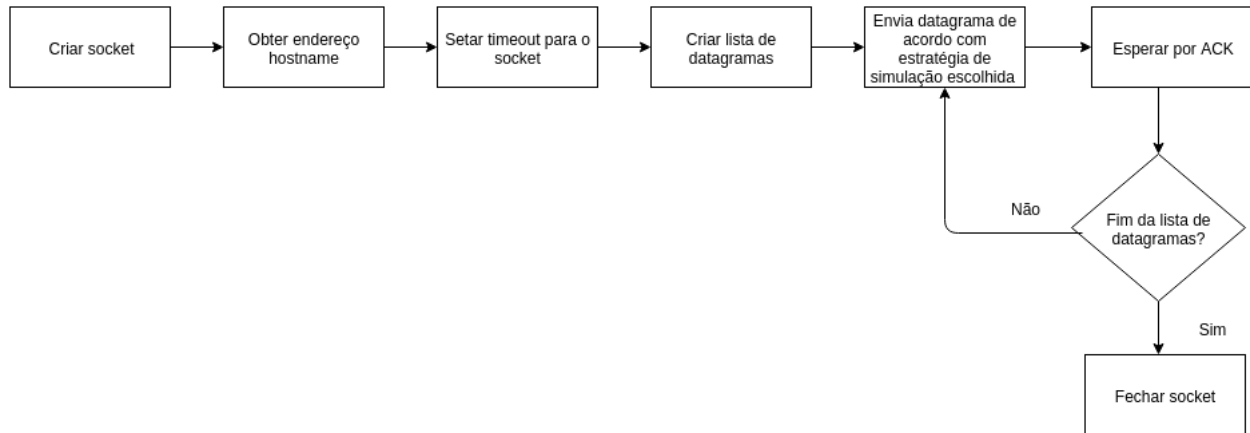


Figura 3: Fluxograma do Cliente

O nosso cliente UDP tem a opção de escolher três tipos de envio de dados: com perda de dados, duplicados ou embaralhados.

Quando o cliente escolhe uma das opções disponíveis, um socket é criado possuindo um *timeout* de 1000 milissegundos para o envio de um arquivo texto (pré-definido no código-fonte). Então, os datagramas são criados.

Para a criação de datagramas, dividimos a mensagem a ser enviada em blocos de 5 bytes cada. Cada datagrama possui um cabeçalho, contendo: o identificador do pacote de envio, o identificador do pacote esperado, o número de datagramas total e o ACK (*acknowledgment*). O ACK pode ser *true* ou *false*, confirmando ou não o recebimento do pacote pelo servidor. A fim de separar os dados do cabeçalho e identificar o que é cada dado, utilizamos o separador “-@-” para dar o *split* nos dados. Com o cabeçalho criado, concatenamos então o dados do datagrama *i* com o cabeçalho. Para a criação do datagrama, utilizamos a classe *DatagramPacket* de Java, onde há um método responsável por criar o datagrama, que recebe como parâmetro o endereço de IP e o dado concatenado com o cabeçalho. O datagrama criado é então adicionado à uma lista.

Com os datagramas da mensagem a ser enviada em uma lista, simulamos então os dados de acordo com a escolha do usuário: perda de dados, dados duplicados ou dados embaralhados. Para a perda de dados, damos um *sleep* com *timeout* de 1 segundo. Sendo assim, um datagrama será perdido a cada vez que ultrapassar o *timeout*. Para simular os dados duplicados, enviamos o datagrama anterior ao que está sendo enviado atualmente, duplicando assim os dados. Para simular dados embaralhados, escolhemos um número aleatório entre a quantidade de datagramas. Então, escolhemos o datagrama com tal identificador sorteado para ser enviado.

Ressaltamos que a fim de garantir que todos os pacotes cheguem, foi utilizado uma *flag* para

alternar entre o datagrama que será perdido, o datagrama que será duplicado ou os datagramas embaralhados com datagrama que será enviado corretamente.

Com o datagrama escolhido para envio de acordo com a opção do usuário, o mesmo é então enviado para o socket. Quando todos os pacotes forem enviados, o socket é fechado.

3.1 Rotinas do Cliente

criarDatagramas(int max, InetAddress IPaddress):

A função criar datagramas recebe por parâmetro duas variáveis importantes para a sua execução, que são a variável max que é representada por um inteiro e a variável IPaddress que guarda o endereço de IP da máquina destino. O retorno é nulo.

DatagramPacket novoDatagrama(InetAddress IPaddress, byte[] dados):

A função novo datagrama recebe por parâmetro duas variáveis importantes para a sua execução, que o vetor de bytes que contém os dados em bytes já convertidos e a variável IPaddress que guarda o endereço de IP da máquina destino. Essa função retorna um pacote no formato do datagrama especificado em nosso código.

DatagramPacket getNextDatagrama(int index):

A função get Next Datagrama recebe por parâmetro uma variável importante para a sua execução, que é a variável index que possui um valor em inteiro guardando o índice do próximo datagrama esperado para o envio. Essa função retorna um pacote no formato do datagrama especificado em nosso código.

DatagramPacket simulacaoDados(int flag, int nextSequence, int tamanho):

A função simulação dados recebe por parâmetro tres variáveis importantes para a sua execução, que são a variável flag que é representada por um inteiro a variável nextSequence o numero de sequência do próximo segmento de dados, e a variável tamanho que diz o tamanho dos dados daquele segmento. Essa função também retorna um pacote no formato do datagrama especificado em nosso código.

void enviarDatagrama(DatagramSocket clientSocket, DatagramPacket pacote):

A função criar datagramas recebe por parâmetro duas variáveis importantes para a sua execução, que são a variável max que é representada por um inteiro e a variável IPaddress que guarda o endereço de IP da máquina destino. Essa função retorna nulo.

String extrairDados(DatagramPacket pacote, MutableInteger numeroDeSequen-

ciaEsperado):

A função extrair dados recebe por parâmetro duas variáveis importantes para a sua execução, que são a variável pacote que é representada pelo tipo datagram packet, que é o pacote com dados e cabeçalho propriamente montados e o número de sequência esperado, que é responsável pela verificação se aquele pacote deve de fato ser extraído ou não. Caso o pacote não seja o esperado, ou seja, se o número de sequência esperado não bater com os dados do pacote ele então é descartado (para as escolhas de recebimento de dados que demandam recebimento em ordem dos dados. O retorno é a string do conteúdo final (texto puro). Um detalhe que deve ser falado é que a classe MutableInteger foi desenvolvida para tratar problemas de compatibilidade de tipos int e integer que não estavam sendo referenciadas corretamente dentro dos métodos. Esse problema trouxe transtornos para passar inteiros por parâmetro e depois recuperá-los posteriormente no código. Com essa implementação o problema foi solucionado.

void receberACK(DatagramSocket clientSocket, MutableInteger indexPacoteEnviado):

A função receber ACK recebe por parâmetro duas variáveis importantes para a sua execução, que são a variável clientSocket que é socket do cliente, que por ele é esperado o pacote recebido. Também é passado o índice do pacote enviado que é um valor inteiro. Essa função é responsável pela detecção do ACK correto para o preparo e envio dos dados para o servidor posteriormente. a função retorna valor nulo.

void run(int flag):

A função run é uma função para executar todos os comandos corretamente na ordem correta, para que o pacote consiga ser enviado no modelo correto, na sequência correta (ou desejada, de acordo com a especificação)

static void main(String args[]):

A função main da parte do cliente solicita ao usuário as opções de simulação com perda de dados, de envio de dados duplicados e envio de dados embaralhados, além de dar início a execução das demais funções (função run, descrita acima).

4 Implementação do Servidor

A Figura 4 apresenta um fluxograma com a ordem da execução do Servidor.

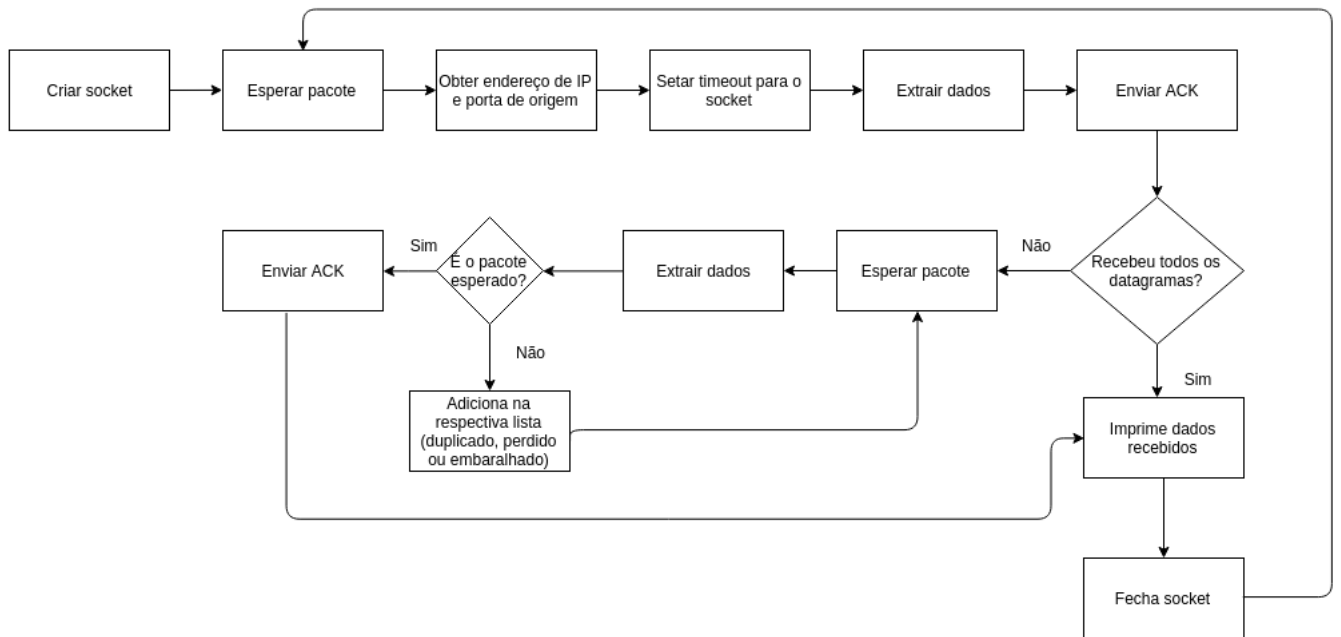


Figura 4: Fluxograma do Servidor

Na implementação do servidor, os dados do socket do cliente são recebidos de 5 em 5 bytes mais o cabeçalho de cada datagrama. Os dados então são decodificados (dando *split* através do separador “-@-”), a fim de separar o cabeçalho do dado.

Se o número do pacote recebido for igual ao número do pacote esperado, significa que o pacote certo foi recebido e um ACK é enviado para confirmar o recebimento do pacote. Se o número do pacote recebido for menor que o número do pacote esperado, então os dados foram duplicados e o ACK do pacote anterior ao esperado é enviado. Se o número do pacote recebido for maior que o número do pacote esperado, então os dados foram embaralhados e o ACK do pacote anterior ao esperado é enviado.

Guardamos em uma lista todos os pacotes enviados normalmente, perdidos, duplicados e embaralhados, para que no final possamos fazer uma estatística de todos os pacotes recebidos.

4.1 Rotinas do Servidor

DatagramPacket novoDatagrama(InetAddress IPaddress, byte[] dados):

A função novo datagrama é exatamente a mesma implementada no lado do cliente, para realizar as mesmas verificações quando os dados estão no lado do servidor.

void enviarDatagrama(DatagramSocket serverSocket, DatagramPacket pacote):

A função novo datagrama é exatamente a mesma implementada no lado do cliente, para

realizar as mesmas verificações quando os dados estão no lado do servidor.

void enviarACK (int indexDoPacoteRecebido)

A função enviar ACK é responsável por enviar o ACK do pacote recebido para o cliente para posterior averiguação.

String extrairDados(DatagramPacket pacote, MutableInteger numeroDeSequenciaRecebido, MutableInteger totalDatagramas):

A função extrair dados é exatamente a mesma implementada no lado do cliente, para realizar as mesmas verificações quando os dados estão no lado do servidor.

int analisarPacote(DatagramPacket pacoteRecebido, Integer numPacoteEsperado, MutableInteger data):

A função analisar pacote verifica se é um pacote com perda de dados, com dados duplicados ou com dados embaralhados. O retorno é o número do pacote esperado.

int run():

A função run é exatamente a mesma implementada no lado do cliente, para realizar as mesmas verificações quando os dados estão no lado do servidor.

static void main(String args[])):

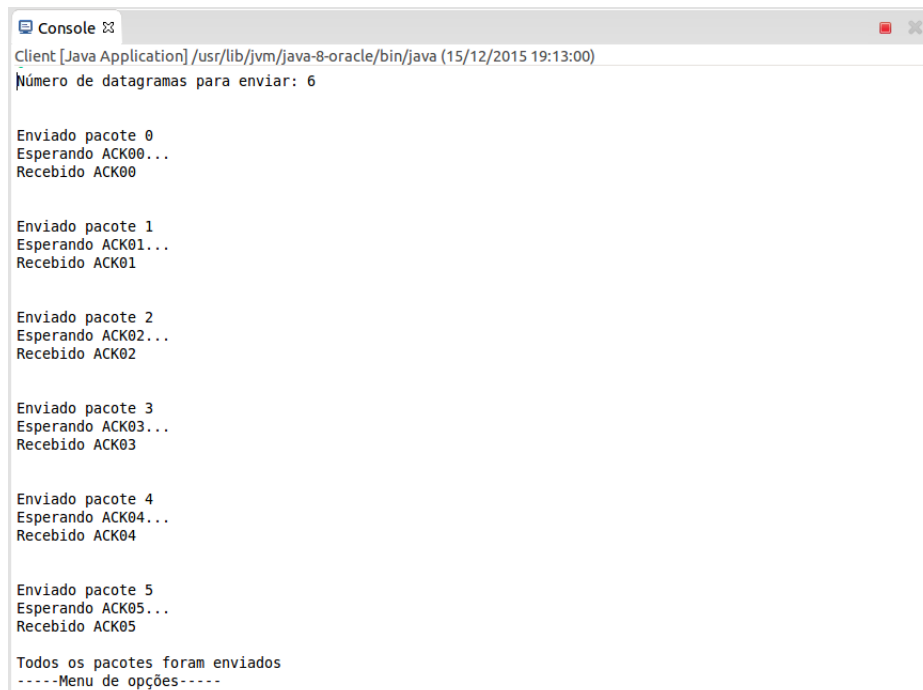
A função main é quase a mesma função implementada no lado do cliente, para realizar as mesmas verificações quando os dados estão no lado do servidor, porém, as diferenças são que na main do cliente é feita a escolha de simulação de dados.

5 Resultados

Abaixo apresentamos os resultados obtidos do cliente e servidor para o envio normal de dados, envio com perda de dados, envio de dados duplicados e envio de dados embaralhados. Para todos os testes utilizamos como entrada o texto *“Testando envio de uma mensagem”*.

5.1 Envio normal de dados

Foram enviados 6 pacotes de 5 bytes cada um. Além disso, todos os ACKs dos pacotes enviados foram recebidos com sucesso na ordem certa e com isso, 100% dos pacotes foram recebidos com sucesso.



```
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:13:00)
Número de datagramas para enviar: 6

Enviado pacote 0
Esperando ACK00...
Recebido ACK00

Enviado pacote 1
Esperando ACK01...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK02

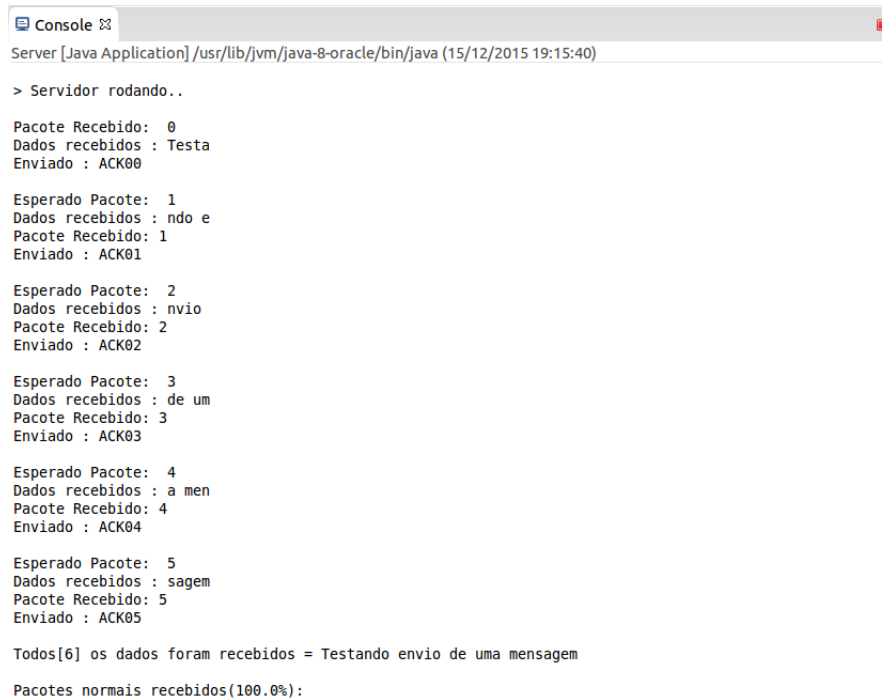
Enviado pacote 3
Esperando ACK03...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK05

Todos os pacotes foram enviados
-----Menu de opções-----
```

Figura 5: Cliente para envio normal de dados



```
Console
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:15:40)

> Servidor rodando..

Pacote Recebido: 0
Dados recebidos : Testa
Enviado : ACK00

Esperado Pacote: 1
Dados recebidos : ndo e
Pacote Recebido: 1
Enviado : ACK01

Esperado Pacote: 2
Dados recebidos : nvio
Pacote Recebido: 2
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03

Esperado Pacote: 4
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : sagem
Pacote Recebido: 5
Enviado : ACK05

Todos[6] os dados foram recebidos = Testando envio de uma mensagem
Pacotes normais recebidos(100.0%):
```

Figura 6: Servidor para envio normal de dados

5.2 Envio com perda de dados

No envio com perda de dados, foram enviados 6 pacotes de 5 bytes cada um. Para a simulação de perda de dados, podemos observar no Cliente (Figura 7) que os pacotes 0 e 1 receberam seus respectivos ACKs. Entretanto para o pacote 2, foi recebido o ACK do pacote 1, ocasionando na retransmissão do dado. O pacote 2 foi então reenviado e seu ACK foi recebido corretamente. O mesmo aconteceu para os pacotes 4 e 5. Ao observar a Figura 8, podemos observar que 33,33% dos pacotes foram perdidos.

```
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:18)
Número de datagramas para enviar: 6

Enviado pacote 0
Esperando ACK00...
Recebido ACK00

Enviado pacote 1
Esperando ACK01...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK03
```

(a) Cliente

```
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:18)
Esperando ACK04...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK05

Todos os pacotes foram enviados
-----Menu de opções-----
[0] Envio normal
[1] Perder Dados
[2] Duplicar Dados
[3] Embaralhar Dados
[4] Sair
Digite sua opção: |
```

(b) Cliente

Figura 7: Cliente para envio com perda de dados

```
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:21)
> Servidor rodando..

Pacote Recebido: 0
Dados recebidos : Testa
Enviado : ACK00

Esperado Pacote: 1
Dados recebidos : ndo e
Pacote Recebido: 1
Enviado : ACK01

Timeout atingido! 2
Enviado : ACK01

Esperado Pacote: 2
Dados recebidos : nvio
Pacote Recebido: 2
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : nvio
Pacote Recebido: 2
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03

Esperado Pacote: 4
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03
```

(a) Servidor

```
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:21)
Timeout atingido! 4
Enviado : ACK03

Esperado Pacote: 4
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : sagem
Pacote Recebido: 5
Enviado : ACK05

Todos[6] os dados foram recebidos = Testando envio de uma mensagem
Pacotes normais recebidos(100.0%):
0 1 2 3 4 5
Pacotes duplicados recebidos(66.66666666666667%):
2 3 4 4
Pacotes perdidos(33.33333333333333%):
2 4 |
Pacotes embaralhados recebidos(0.0%):
/
```

(b) Servidor

Figura 8: Servidor para envio com perda de dados

5.3 Enviado de dados duplicados

Para o envio de pacotes duplicados, podemos observar na Figura 9 que o pacote 1 recebeu o ACK do pacote 0 e logo após o pacote foi retransmitido, recebendo então seu respectivo ACK1. Da mesma forma, os pacotes 2, 3, 4, 5 foram duplicados até que seu respectivo ACK fosse recebido. Na Figura 10 podemos observar que 83,33% dos dados foram duplicados.

```
Console 11
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:18)
2
Número de datagramas para enviar: 6

Enviado pacote 0
Esperando ACK00...
Recebido ACK00

Enviado pacote 1
Esperando ACK01...
Recebido ACK00

Enviado pacote 1
Esperando ACK01...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK03
```

(a) Cliente

```
Console 11
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:18)
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK05

Todos os pacotes foram enviados
-----Menu de opções-----

[0] Envio normal
[1] Perder Dados
[2] Duplicar Dados
[3] Embaralhar Dados
[4] Sair

Digite sua opção: |
```

(b) Cliente

Figura 9: Cliente para envio de dados duplicados

```
Console 11
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:21)
Pacote Recebido: 0
Dados recebidos : Testa
Enviado : ACK00

Esperado Pacote: 1
Dados recebidos : Testa
Pacote Recebido: 0
Enviado : ACK00

Esperado Pacote: 1
Dados recebidos : ndo e
Pacote Recebido: 1
Enviado : ACK01

Esperado Pacote: 2
Dados recebidos : ndo e
Pacote Recebido: 1
Enviado : ACK01

Esperado Pacote: 2
Dados recebidos : nvio
Pacote Recebido: 2
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : nvio
Pacote Recebido: 2
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03

Esperado Pacote: 4
```

(a) Servidor

```
Console 11
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:21)
Esperado Pacote: 4
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03

Esperado Pacote: 4
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : sagem
Pacote Recebido: 5
Enviado : ACK05

Todos[6] os dados foram recebidos = Testando envio de uma mensagem
Pacotes normais recebidos(100.0%):
0 1 2 3 4 5

Pacotes duplicados recebidos(83.33333333333333%):
0 1 2 3 4

Pacotes perdidos(0.0%):

Pacotes embaralhados recebidos(0.0%):

> Servidor rodando..
```

(b) Servidor

Figura 10: Servidor para envio de dados duplicados

5.4 Enviado de dados embaralhados

Para enviar os dados de forma embaralhada, podemos observar na Figura 12 que 33,33% dos pacotes foram embaralhados.

```
Console [1]
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:18)
3
Número de datagramas para enviar: 6

Enviado pacote 0
Esperando ACK00...
Recebido ACK00

Enviado pacote 1
Esperando ACK01...
Recebido ACK00

Enviado pacote 1
Esperando ACK01...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK01

Enviado pacote 2
Esperando ACK02...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK03
```

(a) Cliente

```
Console [2]
Client [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:18)

Enviado pacote 3
Esperando ACK03...
Recebido ACK02

Enviado pacote 3
Esperando ACK03...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK03

Enviado pacote 4
Esperando ACK04...
Recebido ACK04

Enviado pacote 5
Esperando ACK05...
Recebido ACK05

Todos os pacotes foram enviados
----Menu de opções-----

[0] Envio normal
[1] Perder Dados
[2] Duplicar Dados
[3] Embaralhar Dados
[4] Sair

Digite sua opção:
```

(b) Cliente

Figura 11: Cliente para envio de dados embaralhados

```
Console [1]
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:21)

Pacote Recebido: 0
Dados recebidos : Testa
Enviado : ACK00

Esperado Pacote: 1
Dados recebidos : Testa
Pacote Recebido: 0
Enviado : ACK00

Esperado Pacote: 1
Dados recebidos : ndo e
Pacote Recebido: 1
Enviado : ACK01

Esperado Pacote: 2
Dados recebidos : sagem
Pacote Recebido: 5
Enviado : ACK01

Esperado Pacote: 2
Dados recebidos : nvio
Pacote Recebido: 2
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : sagem
Pacote Recebido: 5
Enviado : ACK02

Esperado Pacote: 3
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03

Esperado Pacote: 4
```

(a) Servidor

```
Console [2]
Server [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (15/12/2015 19:29:21)

Esperado Pacote: 3
Dados recebidos : de um
Pacote Recebido: 3
Enviado : ACK03

Esperado Pacote: 4
Dados recebidos : a men
Pacote Recebido: 4
Enviado : ACK04

Esperado Pacote: 5
Dados recebidos : sagem
Pacote Recebido: 5
Enviado : ACK05

Todos[6] os dados foram recebidos = Testando envio de uma mensagem
Pacotes normais recebidos(100.0%):
0 1 2 3 4 5

Pacotes duplicados recebidos(33.333333333333336%):
0 0

Pacotes perdidos(0.0%):
5 5
> Servidor rodando..
```

(b) Servidor

Figura 12: Servidor para envio de dados embaralhados

6 Conclusão

Com este trabalho pudemos aprender como criar uma aplicação cliente-servidor UDP com diversas estratégias de simulação de dados: envio de dados normais, envio com perda de dados, envio de dados duplicados, envio de dados embaralhados. Os desafios encontrados na realização deste trabalho foram a temporização e criar as estratégias para simular os dados de acordo com a opção do usuário.

7 Referências Bibliográficas

[1] <http://code-worm.blogspot.com.br/2012/10/83-stop-wait-protocol-using-sockets-in.html>

[2] <http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html>

[3] Kurose, James F., and Keith W. Ross. “Redes de Computadores e a Internet” São Paulo: Person (2011)