

FCT - Faculdade de Ciências e Tecnologia  
UNESP - Câmpus de Presidente Prudente  
Rede de Computadores I  
Prof. Dr. Milton Hirokazu Shimabukuro

**João Vítor Antunes Ribeiro  
Rafael Silva Santos**

**Relatório do TP Final  
Simulação de fragmentação/remontagem de  
datagramas**

Presidente Prudente - SP  
Junho de 2013

---

## Sumário

---

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Propósito do trabalho . . . . .	1
1.2	Descrição geral . . . . .	1
1.3	Visão geral do restante do documento . . . . .	1
<b>2</b>	<b>Aplicações</b>	<b>3</b>
2.1	Cliente . . . . .	4
2.2	Rede . . . . .	5
2.3	Servidor . . . . .	6
<b>3</b>	<b>Testes</b>	<b>8</b>
3.1	Rede Interna . . . . .	8
3.2	Rede Externa . . . . .	9
<b>4</b>	<b>Conclusões</b>	<b>12</b>

---

## Lista de Figuras

---

2.1	Modelo de datagrama utilizado pelo protocolo IP; extraído de [3]. . . . .	3
2.2	Modelo de datagrama baseado no protocolo IP. . . . .	4
2.3	Método de criação do campo de dados de um datagrama. . . . .	5
2.4	Método de envio de pacotes. . . . .	5
2.5	Método de fragmentação e envio de pacotes. . . . .	6
2.6	Método de desfragmentação e restauração de pacotes. . . . .	7
3.1	Teste local, simulando a transferência de arquivos entre máquinas na mesma rede. . . . .	9
3.2	Cliente sendo executado em uma rede sem fio. . . . .	10
3.3	Informações da máquina rodando Cliente. . . . .	10
3.4	Rede sendo executada em uma rede sem fio. . . . .	10
3.5	Informações da máquina rodando Rede. . . . .	11
3.6	Teste local, simulando a transferência de arquivos entre máquinas na mesma rede. . . . .	11
3.7	Informações da máquina rodando Servidor. . . . .	11

# CAPÍTULO 1

---

## Introdução

---

### 1.1 Propósito do trabalho

O objetivo deste relatório é discorrer sobre as fases de desenvolvimento de aplicações que simulam o funcionamento da transferência de informações entre sistemas finais através de pacotes, salientando o processo de fragmentação e remontagem dos datagramas.

### 1.2 Descrição geral

Inicialmente, foi necessário desenvolver um ambiente de rede com três aplicações independentes, Cliente, Rede e Servidor. A primeira aplicação, o Cliente, representa um sistema típico de cliente, que envia arquivos de texto para outra aplicação, a Rede; a Rede recebe os arquivos enviados pelo Cliente e os fragmenta, redirecionando esses fragmentos para uma terceira aplicação, o Servidor; o Servidor recebe e desfragmenta os fragmentos recebidos da Rede, restaurando o arquivo original enviado pelo Cliente.

### 1.3 Visão geral do restante do documento

A partir desse capítulo introdutório, o restante do relatório está organizado da seguinte forma:

- **Aplicações:** apresenta o projeto a ser desenvolvido, incluindo relato das fases de implementação das aplicações;
- **Teste:** apresenta os resultados obtidos na fase de testes das aplicações;
- **Conclusões:** apresenta as conclusões quanto do funcionamento do sistema de fragmentação e remontagem de datagramas sobre o protocolo TCP/IP.

# CAPÍTULO 2

---

## Aplicações

---

As aplicações desse projeto foram desenvolvidas na linguagem de programação Java, pois além de possuir maior portabilidade entre sistemas operacionais, também possui um conjunto amplo de funcionalidades para manipular componentes de redes de computadores.

Para realizar a comunicação entre as aplicações, que desde já serão definidas como Cliente, Rede e Servidor, foi utilizado o protocolo de transmissão TCP (*Transmission Control Protocol*), devido ao tipo de informações que serão transmitidas entre as aplicações, isto é, arquivos de texto puro (formato “txt”), onde é exigida grande confiabilidade de transferência e não é tolerada a perda de dados. Na Figura 2.1 é exibido o modelo de datagrama utilizado pelo protocolo IP.

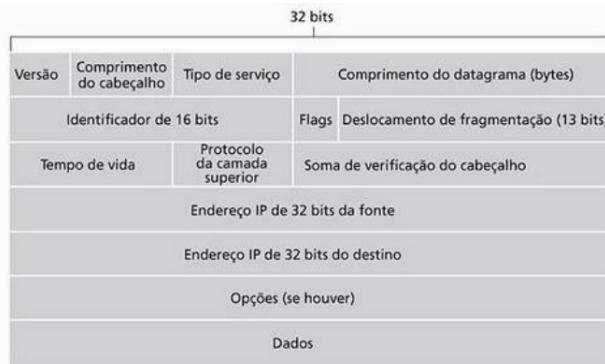


Figura 2.1: Modelo de datagrama utilizado pelo protocolo IP; extraído de [3].

Para trocar informações entre as aplicações, foi criado um modelo de pacote ou datagrama baseado no protocolo IP. Na Figura 2.2 é apresentado o modelo do datagrama desenvolvido para o simulador.

```
public class Datagrama implements Serializable{  
    //Cabecalho: tamanho total = 20 bytes  
    private byte [] ipOrigem, ipDestino;           //Campos IP Origem e IP Destino  
    private byte tipoServiço, tempoDeVida, protocolo; //Tipo de Serviço, Tempo de Vida e Protocolo  
    private char tamanho, identificador, checksum, offset; //Tamanho Datagrama, ID, Checksum e OFSET  
    private BitSet flag, versão, tamCabecalho;      //Flag, versão e Tamanho do Cabecalho do datagrama  
    //Dados:  
    private byte[] dados;
```

Figura 2.2: Modelo de datagrama baseado no protocolo IP.

Quanto à transmissão de informações entre as aplicações, utilizou-se o conjunto de funcionalidade providas pelas classes *Socket* e *ServerSocket*. De acordo com [1], *Socket* é uma classe de estabelecimento de conexão para envio de dados entre duas máquinas em uma rede de computadores. Já a classe *ServerSocket*, conforme descrito em [2], atua como um servidor de computadores, esperando pelo recebimento de dados enviados por outras máquinas em uma rede de computadores.

## 2.1 Cliente

A aplicação Cliente simula um sistema de computador pessoal, em que o usuário envia dados de sua máquina para outra, isto é, transfere arquivos para outro computador. Nesse caso, os arquivos que podem ser transferidos estão limitados ao formato de texto puro, o *txt*. Sendo assim, essa aplicação conta com funcionalidades de ajuda de utilização, seleção de arquivo para envio, envio de arquivo, visualização dos parâmetros do datagrama, definição do IP e da porta da rede, limpeza da tela de log e encerramento da aplicação.

Após selecionar um arquivo de texto, a aplicação Cliente cria um novo datagrama com as especificações da Rede introduzidas e transforma o conteúdo do arquivo na região de dados serializados desse datagrama, representados por um vetor de *bytes*. A Figura 2.3 mostra o método de criação do campo de dados do datagrama.

```

public void setDados(byte [] dados){           //Setando os dados no arquivo, array de bytes
    this.dados = dados;
}

public void setDados(File arquivo) throws IOException {      //Setando Dados do Arquivo, arquivo
    String todoArquivo = "";
    @SuppressWarnings("resource")
    BufferedReader entrada = new BufferedReader(new FileReader(arquivo));
    while (entrada.ready()) {                      //Lendo o arquivo de upload.
        todoArquivo += entrada.readLine()+"\n";
    }
    dados = todoArquivo.getBytes();           //Convertendo string para bytes
    setTamanho(getTamanho()+dados.length);
}

```

Figura 2.3: Método de criação do campo de dados de um datagrama.

Após configurar o Cliente e selecionar o arquivo de *upload*, o usuário deve inserir o comando de envio do arquivo que, assim como todos os outros comandos de todas as aplicações, podem ser consultados através do comando *ajuda* ou *help*; dessa forma, o arquivo deverá ser enviado sem fragmentação para a Rede, que simula o funcionamento do núcleo de uma rede de computadores. A Figura 2.4 mostra o código de envio do datagrama.

```

public void enviarPacote() throws Exception{      //Enviar o datagrama pelo socket c
    try{
        cliente = new Socket(ip, porta);
        ObjectOutputStream saida = new ObjectOutputStream(cliente.getOutputStream());
        saida.writeObject(datagrama);
        saida.close();
        cliente.close();
        System.out.println("Objeto enviado com sucesso...");
    }catch(Exception erro){
        System.out.println("Falha no envio do objeto...\nErro: "+erro);
    }
}

```

Figura 2.4: Método de envio de pacotes.

## 2.2 Rede

Essencial no contexto da transmissão de pacotes entre o Cliente e o Servidor, o papel da Rede é receber a informação enviada pelo Cliente através de um pacote não fragmentado e direcionar tal informação para a aplicação Servidor, realizando sua fragmentação em pacotes. As funcionalidades dessa aplicação são a troca da porta do Servidor, visualização das informações do Servidor, definição do MTU (*Maximum Transmission Unit*), estabelecimento de conexão, limpeza da tela de log e encerramento da aplicação.

A Rede age como um roteador, redirecionando pacotes que chegam do Cliente para o Servidor. Após configurar a Rede, o usuário deve conectá-la com o comando *conectar* ou *connect*; a partir desse momento, a Rede entrará em um estado permanente de conexão, permitindo o fluxo de dados de entrada e saída. Baseado no MTU definido, o objeto recebido do Cliente é fragmentado e enviado para o Servidor, responsável pela restauração do mesmo. O método de fragmentação desenvolvido está exposto na Figura 2.5.

```

public void fragmentar(Datagrama datagrama) throws Exception
{
    Transferencia transferencia = new Transferencia(); //Necessário para enviar os datagramas
    int areaDados = mtu-20, posIniDatagrama, numFragmento = 0;
    Datagrama novoDatagrama; //Necessário para enviar os dados fragmentados.

    byte[] dados = datagrama.getDados(); //Resgatando a área de dados do datagrama.

    /* Fragmentando o datagrama original: */
    for (posIniDatagrama = 0; posIniDatagrama < dados.length; posIniDatagrama += areaDados) {
        /* Dividindo os dados do datagrama em porções e agrupando em novos datagramas: */
        novoDatagrama = new Datagrama();
        novoDatagrama.setIPOrigem(datagrama.getipOrigem());
        novoDatagrama.setIPDestino(datagrama.getipDestino());
        novoDatagrama.setChecksum(datagrama.getChecksum());
        novoDatagrama.setTamanho(mtu);
        offset = areaDados/8; //Calculo do Offset
        novoDatagrama.setOffset(posIniDatagrama*offset);
        novoDatagrama.setIdentificador(numFragmento);

        //Identificando o último datagrama a ser enviado:
        if (posIniDatagrama+areaDados >= dados.length) novoDatagrama.setFlag(0);
        else novoDatagrama.setFlag(1);

        //FAZENDO O TRATAMENTO PARA O CASO DA ÁREA DE DADOS DO NOVO PACOTE EXCEDER O Q AINDA RESTA
        //SE NÃO TRATAR, SÃO ESCRITOS BYTES NULOS NO ARQUIVO FINAL, NO SERVIDOR, POR CAUSA DO ULT.
        if (posIniDatagrama+areaDados > dados.length) areaDados = dados.length-posIniDatagrama;
        byte[] parcelaDados = new byte[areaDados];

        for (int pos = 0; pos < areaDados && posIniDatagrama+pos < dados.length; pos++) { //Pr
os do datagrama original.
            parcelaDados[pos] = dados[posIniDatagrama+pos];
        }

        novoDatagrama.setDados(parcelaDados);

        /* Enviando um novo fragmento do datagrama original: */
        transferencia.setDatagrama(novoDatagrama);
        transferencia.enviarPacote();
        ++numFragmento;
    }
}

```

Figura 2.5: Método de fragmentação e envio de pacotes.

## 2.3 Servidor

A aplicação Servidor apenas recebe pacotes enviados pela Rede e os desfragmenta com base no campo *flag*. Após a definição da porta de escuta e o estabelecimento da conexão, o Servidor recebe e armazena os dados dos pacotes transmitidos pela Rede e, quando do recebimento de um pacote com o campo *flag* igual a 0, restaura o objeto enviado pelo Cliente juntando todos os dados acumulados. Ao final desse processo, uma janela de seleção é mostrada permitindo ao usuário salvar o arquivo recebido. Abaixo, a Figura 2.6 mostra o código de desfragmentação e restauração de pacotes.

```

public boolean desfragmentar(Datagrama datagrama)
{
    dados.add(datagrama.getDados());      //Retirando a porção de dados do pacote
    tamanho += datagrama.getDados().length; //Calculando o tamanho do datagrama
    if (datagrama.getFlag() == 0) {        //Último pacote detectado.
        this.datagrama = new Datagrama();
        this.datagrama.setIPOrigem(datagrama.getipOrigem());
        this.datagrama.setIPDestino(datagrama.getipDestino());
        this.datagrama.setTamanho(tamanho+20); //Adicionando o tamanho do cabeçalho
        int pos = 0, i;
        byte[] dadosRestaurados = new byte[tamanho]; //Necessário para restaurar os dados
        for (byte[] b : dados) { //Percorrendo todos os fragmentos de dados,
            for (i = 0; i < b.length; i++) { //Percorrendo todos os bytes de cada fragmento
                dadosRestaurados[pos+i] = b[i]; //Restaurando os dados originais
            }
            pos += i;
        }
        this.datagrama.setDados(dadosRestaurados);
        return(false); //Último pacote recebido: fim da restauração.
    }
    return(true); //Faltam pacotes a serem restaurados.
}

```

Figura 2.6: Método de desfragmentação e restauração de pacotes.

# CAPÍTULO 3

---

## Testes

---

O simulador de transferência de pacotes criado pode ser testado em dois ambientes de rede distintos, interno e externo. Por isso, esse capítulo foi dividido em duas seções que relatam os testes realizados em ambos os cenários.

### 3.1 Rede Interna

No teste da rede local, as aplicações são executadas dentro de uma mesma rede de computadores, com mesmo IP, chamada de rede interna. Desta forma, as aplicações executadas na mesma máquina são diferenciadas sómente através da porta de escuta, pois contém o mesmo endereço de IP, 127.0.0.1 ou *localhost*. Cliente não tem a necessidade de seleção de porta, já que isso depende exclusivamente do sistema operacional em que está sendo executado.

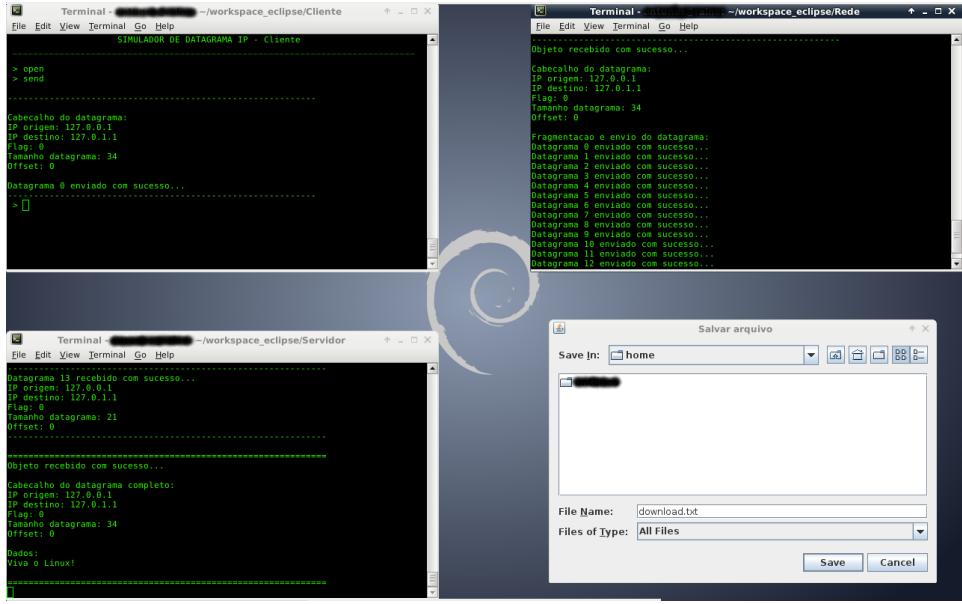


Figura 3.1: Teste local, simulando a transferência de arquivos entre máquinas na mesma rede.

Na Figura 3.1 podemos observar o cenário do teste realizado, em que as três aplicações são executadas em uma mesma máquina, com o SO Debian 7. No Cliente é selecionado o arquivo de texto a ser enviado para a Rede, que após ter sido devidamente configurada, recebe o arquivo como um objeto e o fragmenta em datagramas relativos ao MTU escolhido pelo usuário. A Rede então envia esse conjunto de datagramas fragmentados para o Servidor, que desfragmenta os datagramas e restaura o objeto original enviado, permitindo ao usuário salvá-lo em formato de texto.

## 3.2 Rede Externa

Assim como no teste da Rede Interna, as aplicações mantém o mesmo princípio de funcionamento; o que muda neste caso são as configurações de endereçamento do Servidor e da Rede, variáveis somente na aplicação Cliente. Para esse teste, foi utilizado uma rede de computadores sem fio, com o prefixo 186.217.0.0.

A Figura 3.2 mostra o funcionamento de Cliente sendo executado em uma máquina com o IP local 192.168.10.190 e IP externo (sem fio) 187.217.105.156, utilizando o SO Windows 7 Pro. As configurações da máquina podem ser visualizadas na Figura 3.3.

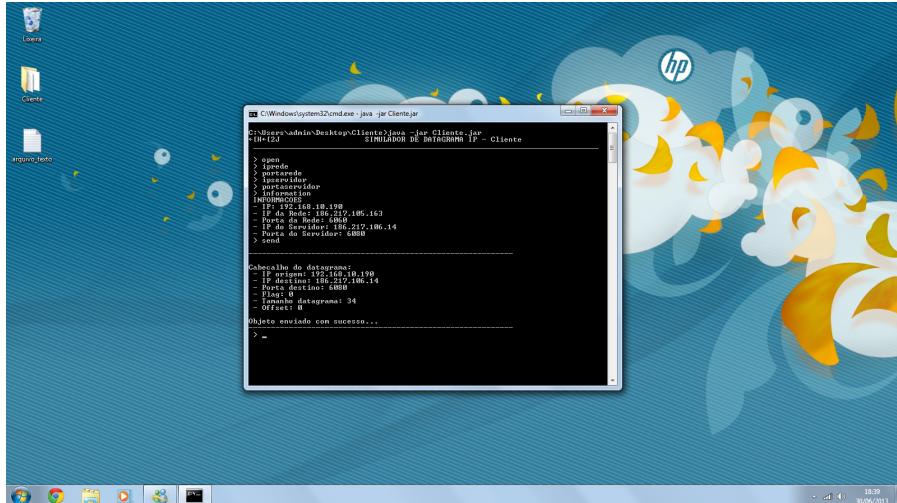


Figura 3.2: Cliente sendo executado em uma rede sem fio.

```
Configuração de IP do Windows
Adaptador de Rede sem Fio Conexão de Rede sem Fio:
Sufixo DNS específico de conexão . . . . . : 186.217.105.156
Endereço IPv4 . . . . . : 192.168.10.190
Máscara de Sub-rede . . . . . : 255.255.255.0
Gateway Padre . . . . . : 186.217.105.10
Adaptador Ethernet Conexão local:
Sufixo DNS específico de conexão . . . . . : 192.168.10.190
Endereço IPv4 . . . . . : 192.168.10.190
Máscara de Sub-rede . . . . . : 255.255.255.0
Gateway Padre . . . . . : 192.168.10.254
```

Figura 3.3: Informações da máquina rodando Cliente.

Abaixo, na Figura 3.4, é exibido o funcionamento de Rede sobre o IP local 127.0.0.1 e IP externo (sem fio) 186.217.105.163, utilizando o SO Debian 7. As configurações da máquina podem ser visualizadas na Figura 3.5.

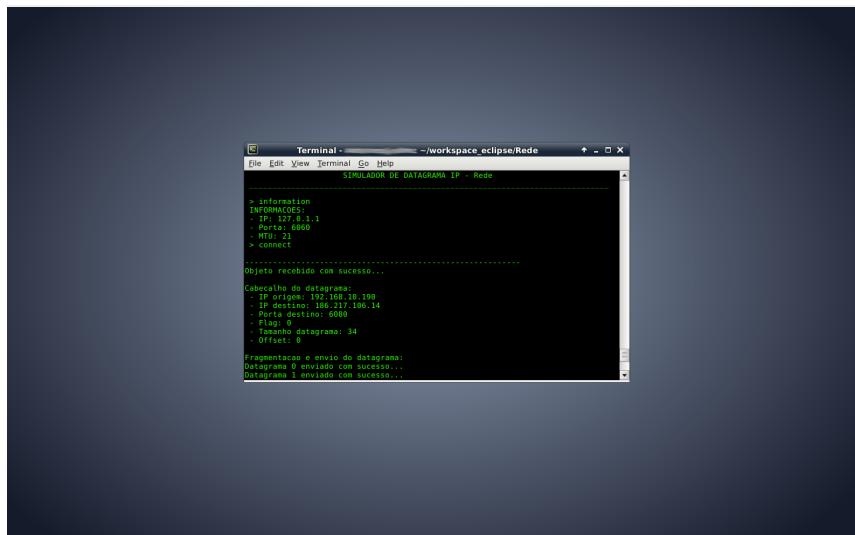


Figura 3.4: Rede sendo executada em uma rede sem fio.

```

lo      Link encap:UNSPEC brd:0x000000000000
        inet addr:127.0.0.1  Mask:255.0.0.0
              inet6 addr: fe80::1%lo1  brd:255.255.255.255 Scope:Link
                    UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
                    RX packets:30 errors:0 dropped:0 overruns:0 frame:0
                    TX packets:30 errors:0 dropped:0 overruns:0 carrier:0
                    collisions:0 txqueuelen:1
                      RX bytes:1860 (1.8 Kib)
                         TX bytes:1860 (1.8 Kib)

wlan0   Link encap:UNSPEC brd:0x000000000000
        inet6 addr: fe80::217:106%1  brd:255.255.255.255 Scope:Link
              inet  addr:192.168.106.14  Bcast:192.168.106.14  Mask:255.255.255.0
                    UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
                    RX packets:18542 errors:0 dropped:0 overruns:0 frame:0
                    TX packets:18542 errors:0 dropped:0 overruns:0 carrier:0
                    collisions:0 txqueuelen:1000
                      RX bytes:21703453 (20.6 Mib)
                         TX bytes:3722701 (3.5 Mib)

```

Figura 3.5: Informações da máquina rodando Rede.

Figura 3.6 mostra o funcionamento de Servidor sendo executado em uma máquina com o IP local 192.168.10.223 e IP externo (sem fio) 187.217.106.14, utilizando o SO Windows 7 Pro. As configurações da máquina podem ser visualizadas na Figura 3.7.

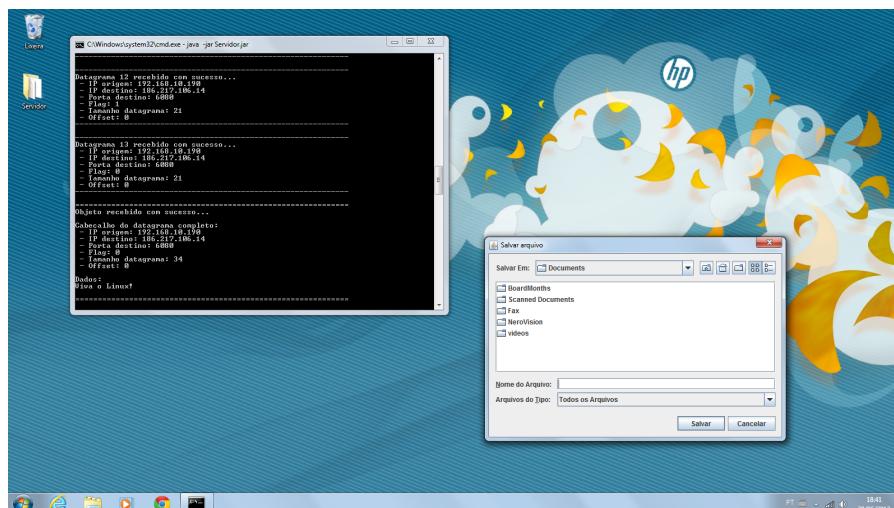


Figura 3.6: Teste local, simulando a transferência de arquivos entre máquinas na mesma rede.

```

Configuração de IP do Windows
Adaptador de Rede sem Fio Conexão de Rede sem Fio 2:
  Estado da mídia ..... : mídia desconectada
  Sufixo DNS específico de conexão..... :
Adaptador de Rede sem Fio Conexão de Rede sem Fio:
  Sufixo DNS específico de conexão..... : fe80::6cf2:1423:6910:9c16%13
  Endereço IPv6 de link local ..... : fe80::217:106%14
  Endereço IPv6 de link global ..... : 187.217.106.14
  Máscara de Sub-rede ..... : ::ffff:ffff:ffff:ffff::0
  Gateway Padrão ..... : 187.217.106.10
Adaptador Ethernet Conexão local:
  Sufixo DNS específico de conexão..... : ncc.fct.unesp.br
  Endereço IPv6 de link local ..... : fe80::1%12
  Endereço IPv6 de link global ..... : 192.168.10.223
  Máscara de Sub-rede ..... : ::ffff:ffff:ffff::0
  Gateway Padrão ..... : 192.168.10.254

```

Figura 3.7: Informações da máquina rodando Servidor.

# CAPÍTULO 4

---

## Conclusões

---

Existem muitas consequências negativas se a transmissão de datagramas for feita de forma incorreta, como congestionamento na rede, perda de pacotes e sobrecarga de serviços. Assim, o uso da fragmentação e remontagem de datagramas IP é fundamental para uma transmissão efetiva e otimizada de dados em uma rede de computadores.

Através deste trabalho concluímos que, embora em uma rede de computadores existam muitos componentes de otimização de serviços, além de arquiteturas e mecanismos complexos que visam oferecer maior qualidade na estrutura da rede, se separados, a maioria desses componentes são simples e intuitivos quanto à compreensão, podendo ser simulados por pequenas aplicações, como é o caso da fragmentação e remontagem de datagramas.

---

## Referências Bibliográficas

---

- [1] *Class Socket*, Disponível em: <http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html>, Acessado em: 25/06/2013.
- [2] *Class ServerSocket*, Disponível em: <http://docs.oracle.com/javase/1.4.2/docs/api/java/net/ServerSocket.html>, Acessado em: 25/06/2013.
- [3] KUROSE, J. F., ROSS, K. W., *Redes de computadores e a Internet: uma abordagem top-down*, e. 5 - São Paulo: Addison Wesley, 2010.