Pedro Paulo Vezzá Campos

Trabalho Prático 4: Implementação de um Protocolo da Camada de Transporte

Santa Catarina - SC, Brasil

21 de novembro de 2010

Pedro Paulo Vezzá Campos

Trabalho Prático 4: Implementação de um Protocolo da Camada de Transporte

Trabalho apresentado para avaliação na disciplina INE5414, do curso de Bacharelado em Ciências da Computação, turma 04208, da Universidade Federal de Santa Catarina, ministrada pelo professor Carlos Becker Westphall

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

Santa Catarina - SC, Brasil

21 de novembro de 2010

Resumo

Este trabalho apresenta uma implementação do protocolos das camadas de rede, transporte e aplicação, com ênfase na segunda. Para demonstrar o funcionamento de tais protocolos também foi desenvolvida uma aplicação orientada a conexão para apresentar o funcionamento de tais entidades de rede. Como forma de fundamentação, também é apresentada uma introdução teórica, mostrando os principais conceitos na área dos protocolos e camadas da pilha de protocolos do Modelo OSI utilizados durante o trabalho.

Introdução

Este quarto trabalho prático possui como objetivo sedimentar os conhecimentos apreendidos durante o curso da disciplina, apresentando o desenvolvimento de um protocolo de camada de transporte, baseado no exemplo fornecido pela bibliografia da disciplina [1]. Como forma de mostrar seu funcionamento foram desenvolvidos adicionalmente um protocolo orientado a conexão, além das camadas de aplicação e de rede para prover a infraestrutura necessária para o funcionamento correto do aplicativo desenvolvido.

Este relatório possui a seguinte organização: Primeiramente serão apresentados os conceitos principais das camadas de rede, transporte e aplicação da pilha de protocolos de redes OSI utilizados durante a implementação da aplicação apresentada anteriormente. Posteriormente há uma explanação detalhada dos procedimentos adotados para o desenvolvimento da aplicação. Ainda, são apresentados alguns comentários e conclusões sobre este trabalho. Por fim é apresentado o código fonte comentado da aplicação desenvolvida.

Camada de Rede

No modelo OSI, a camada de rede é a terceira, já no modelo TCP/IP é a segunda. Sua tarefa principal é prover serviços à camada de transporte. Para isso, ela faz uso das primitivas fornecidas pela camada de enlace de dados.

A camada de rede e a sua camada inferior, a camada de enlace de dados, possuem propósitos diferentes. A camada de enlade de dados, tem a responsabilidade de transmitir dados entre duas pontas de um canal físico, tendo como preocupações questões como a correção de erros, detecção de colisões e perdas de datagramas. Já a camada de rede é a responsável pela entrega de pacotes da origem ao destino, havendo a possibilidade de ser necessário o roteamento entre

hosts intermediários. É na camada de rede que estão imbutidos os protocolos e algoritmos de roteamento, detecção de congestionamento, dentre outros. Ainda, ela é responsável por propiciar uma compatibilidade entre redes diferentes, cada uma com configurações variadas. Ela possui participação no controle da qualidade de serviço (QoS), além de executar funções de controle de erros.

A camada de rede necessita conhecer diversos parâmetros de configuração para poder fornecer um serviço adequado à camada de transporte. Algumas dessas informações estão descritas abaixo:

- O protocolo de rede é orientado a conexão?
- Como controlar o congestionamento da rede?
- Quais são os endereços únicos (globais) da rede?
- Quais são minhas máquinas vizinhas?
- Como devo rotear dados com destino outros computadores?

Há duas famílias principais de protocolos de rede, os orientados a conexão, funcionando por circuitos virtuais e os baseados em datagramas. Cada uma dessas abordagens possui vantagens e desvantagens, como é possível ver a seguir:

Protocolos de rede orientados a conexão são mais simples de implementar devido a características da própria abordagem, não há a necessidade de controlar congestionamento ou utilizar algoritmos sofisticados para oferecer qualidade de serviço. Ainda, em tais redes não é necessário que os endereços de origem e destino sejam transmitidos em cada pacote. Outra vantagem de se usar protocolos de rede baseados em circuitos virtuais é que a possibilidade de garantir parâmetros de atraso máximo, garantia essa muito importante para a transmissão de dados multimídia, por exemplo. [2]

Como desvantagem, há um custo ao usar essa escolha: A partir do momento que é estabelecido um circuito virtual há a separação de uma porção dos recursos de rede exclusivamente para um único usuário, esteja ele usando a rede ou não. Isso incorre em custos extra aos provedores de serviços de Internet, sendo repassados aos clientes finais, de maneira similar como o que ocorre na telefonia convencional.

Já uma camada de rede não orientada a conexão funciona por multiplexação de pacotes, ou seja, os pacotes não possuem uma rota pré-estabelecida como ocorre no caso anterior após o

estabelecimento do circuito virtual. Dessa forma, cada pacote deve conter o endereço completo de origem e destino para poder ser roteado. Em uma transmissão entre dois computadores os pacotes podem adotar diversas rotas diferentes, passando por vários países, por exemplo, dependendo da carga da rede no momento do envio do pacote.

Uma grande vantagem de seusar uma camada de rede baseada em datagramas é a sua capacidade de distribuir melhor a carga no canal físico, colaborando com menor congestionamento em momentos de grande utilização. Além disso, uma rede baseada em comutação de pacotes é mais tolerante a falhas uma vez que quando um nodo da rede falha os dados podem adotar uma rota alternativa. Um exemplo de protocolo de redes não orientado a conexão é o IP [3], utilizado na Internet.

Camada de Transporte

A camada de transporte é a terceira camada do modelo TCP/IP e a quarta camada segundo o modelo OSI. Ela utiliza os serviços da camada de rede (roteamento de pacotes entre hosts, por exemplo) para atender as requisições dos protocolos de aplicação.

A camada de transporte tem um papel muito importante dentre as camadas da pilha de protocolos OSI. Ela é a primeira camada que estabelece uma conexão fim-a-fim, confiável ou não, entre dois processos de aplicação em dois hosts. Para um a camada de aplicação (ou sessão, dependendo do modelo), que usa a interface da camada de transporte, a conexão de rede pode ser vista como uma conexão direta, em que bytes são inseridos em um lado da conexão e após algum tempo aparecem no outro lado. O protocolo de transporte é responsável por iniciar e terminar as conexões entre processos que desejam estabelecer comunicação, fazendo a multiplexação dessas conexões.

É nessa camada que os dados que chegam da camada de sessão são quebrados em unidades menores, os pacotes, para que possam ser mais facilmente enviados pela camada de rede. Faz parte da responsabilidade de uma camada de transporte que forneça uma conexão confiável, por exemplo, garantir que os pacotes cheguem ao destino corretamente e em ordem.

O código binário do protocolo de transporte geralmente é o último que encontra-se no núcleo (*kernel*) do sistema operacional. As camadas de sessão, apresentação e aplicação, quando existentes, geralmente encontram-se em espaço de usuário, implementadas como bibliotecas, por exemplo. Programas que queriam fazer uso de comunicações HTTP, por exemplo, como um *browser web*, utilizam as bibliotecas HTTP para transmitir e receber os dados que por ventura devam trafegar pela rede.

Como forma de suportar várias conexões de transporte ao mesmo tempo, o software responsável por essa camada de rede possui diversas possibilidades:

Multiplexação no tempo Caso haja apenas uma conexão de rede disponível, a camada de transporte envia dados de diferentes conexões em diferentes períodos de tempo.

Exclusividade de conexão Caso haja alguma aplicação que demande maiores recursos de redes, há a possibilidade que ela monopolize a conexão, de forma a fornecer a melhor qualidade de serviço possível a essa aplicação específica.

Escalonamento Caso haja mais de uma conexão de rede disponível, é possível ainda escalonar a utilização, utilizando algoritmos como o round-robin com as conexões de rede disponíveis, diminuindo assim o congestionamento nas sub-redes que o host está conectado.

Camada de Aplicação

A camada de aplicação é sétima camada no modelo OSI e a quarta no modelo TCP/IP. Em ambas é a camada de mais alto nível. Sua característica principal é que ela apresenta uma interface direta com os aplicativos. Seu serviço é prover a eles uma abstração da comunicação em rede, escondendo os detalhes de implementação das camadas mais inferiores. Para isso, ela faz uso da camada de apresentação ou de transporte dependendo do modelo de redes adotado.

Exemplos de protocolos da camada de aplicação:

- BitTorrent
- SSH - *Secure Shell*
- BGP - *Border Gateway Protocol*
- DHCP - *Dynamic Host Configuration Protocol*
- HTTP - *HyperText Transfer Protocol*
- DNS - *Domain Name System*
- NFS - *Network File System*
- FTP - *File Transfer Protocol*
- SNMP - *Simple Network Management Protocol*

- IMAP - *Internet Message Acces Protocol*

A camada de aplicação é responsável transformar os pacotes recebidos da camada inferior em informação reconhecível pela aplicação. Os dados passam a ter um significado maior que apenas um fluxo de bits. Os programas aplicativos, dentre os quais podemos exemplificar os navegadores, players multimídia, clientes de IM, dentre outros geralmente usam de bibliotecas para apresentar ao usuário sua funcionalidade.

A interface entre os programas de aplicação e os protocolos da camada de transporte é feita usando-se *sockets*. Um socket pode ser visto como um "tubo" com um fluxo bidirecional de dados, podendo ter características como criptografia, garantia de entrega de informações, etc. As operações disponíveis em um *socket* são similares às possíveis de realizar em arquivos. Exemplos de operações são o *BIND*, *LISTEN*, *CONNECT*, *READ* e *WRITE*. Essas operações são a base do funcionamento de um socket.

Desenvolvimento dos protocolos

Como informa a bibliografia, que forneceu o código base para a implementação do protocolo de transporte:

> "Você deve ter em mente a simplicidade da Figura 6.20. Uma entidade de transporte real normalmente verificaria a validade de todos os parâmetros fornecidos, cuidaria da recuperação do funcionamento após uma falha na camada de rede, trataria das colisões de chamadas e seria compatível com um serviço de transporte mais genérico, com interrupções, datagramas e versões sem bloqueio das primitivas SEND e RECEIVE." [1]

Dessa forma, a camada de transporte desenvolvida nesse trabalho tem características incomuns a uma camada de transporte usual, não implementando o TCP nem o UDP, por exemplo. Dessa forma, não é possível usar aplicativos normais, tais como *browsers*, ou clientes de e-mail para testar o funcionamento da camada de transporte desenvolvida.

Sendo assim, foi desenvolvida uma aplicação orientada a conexão como prova de conceito. Tal programa consiste em um programa de bate-papo simples com dois clientes distintos em uma mesma janela para melhor visualização. Toda mensagem enviada por um cliente é enviada ao outro utilizando a infraestrutura de camadas desenvolvida. Como forma de apresentar o funcionamento interno dos protocolos, cada cliente pode ver as primitivas invocadas por cada uma das camadas desde o envio da mensagem por um lado até a chegada dela no outro lado.

Recursos utilizados

Ubuntu Linux Versão 10.10

Python 2.7 Linguagem de programação adotada

PyQt Biblioteca utilizada no desenvolvimento da interface gráfica com o usuário.

Camada de Rede

Esta camada possui uma implementação simples neste trabalho. Sua tarefa é de realizar o redirecionamento de pacotes enviado por uma camada de transporte para a outra. Para isso, a camada de rede fornece a primitiva `enviarPacote` às camadas de transporte. Por outro lado, sempre que um novo pacote chega, após a decisão de qual camada de transporte corresponde o pacote, é invocado o método `daRede` correspondente, enviando o dado à camada superior.

Camada de Transporte

Como descrito anteriormente, a linguagem de programação adotada foi o Python. Como o código fornecido pela bibliografia estava programado em C, o primeiro passo foi realizar uma tradução para a linguagem destino, realizando diversas adaptações necessárias diante da diferença de paradigmas adotados: Estruturado vs. Orientado a Objetos, variáveis globais, definições em tempo de compilação, etc.

Ainda, o código fornecido teve algumas implementações faltantes suplantadas:

- `sleep`, e `wakeUp` que no programa final tornaram-se `dormir` e `acordar`, cumprem a função de uma barreira da programação concorrente, bloqueando e reiniciando o funcionamento da camada de transporte à medida que haja novos eventos para serem tratados.
- `fromNet`, e `toNet` que no programa final tornaram-se `daRede` e `paraRede`, que são métodos de interface com a camada de rede para o recebimento e envio de pacotes.

Para permitir a interação entre camadas vizinhas houve a necessidade que cada camada tivesse referências às suas vizinhas para permitir um funcionamento síncrono.

A camada de transporte possui um vetor que armazena e organiza as conexões atuais, representadas como objetos do tipo `Conexao`. Cada pacote é representado por objeto da classe

`Pacote`, além disso e eles são enviados à camada de rede através do método interno `paraRede`, são roteados pela camada de rede, e depois ressurgem na camada de transporte homóloga através do método `daRede`.

Camada de Aplicação

A camada de apresentação provê as primitivas básicas ao programa de bate-papo a ser apresentado posteriormente. Seus métodos principais são `conectar`, `escutar`, `enviarMensagem`, `receberMensagem` e `fecharConexao`. Tais métodos são responsáveis por estabelecer uma conexão, permanecer aguardando uma nova conexão, enviar uma mensagem, receber uma mensagem que será posteriormente entregue à aplicação e encerrar uma conexão estabelecida, respectivamente. Todos esses serviços prestados pela camada de aplicação utilizam primitivas fornecidas pela camada de transporte.

Aplicação Prova de Conceito

Como explicado anteriormente, foi desenvolvida um programa de bate-papo simples com dois clientes distintos em uma mesma janela para melhor visualização. Toda mensagem enviada por um cliente é enviada ao outro utilizando a infraestrutura de camadas desenvolvida.

Adicionalmente, como forma de apresentar o funcionamento interno dos protocolos, cada cliente pode ver as primitivas invocadas por cada uma das camadas desde o envio da mensagem por um lado até a chegada dela no outro lado.

Validação, Verificação e Testes

A validação e a verificação foram feitas através de máquinas de estados representando o protocolo implementado. Posteriormente à implementação, procedeu-se com a codificação de testes unitários através daa biblioteca PyUnit, embutida na linguagem Python. Através destes testes unitários verificou-se o funcionamento adequado dos protocolos programados.

Comentários sobre os resultados obtidos

O quarto trabalho prático de INE5414 foi de grande importância para a sedimentação dos conceitos vistos durante o semestre, contribuindo para o objetivo de relacionar teoria e prática

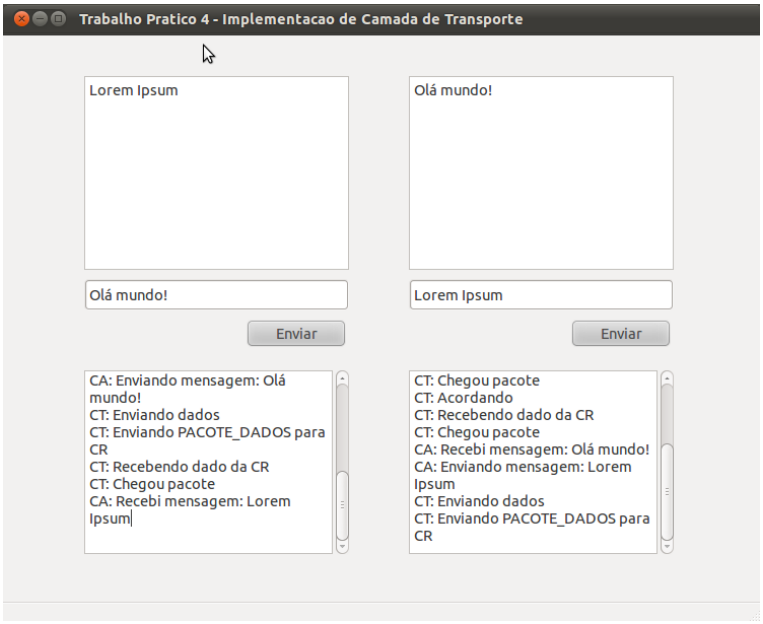

Figura 1: Imagem do funcionamento do aplicativo de prova de conceito

em Redes de Computadores. Ainda, permitiu elucidar vários conceitos, principalmente os relacionados ao modelo de redes em camada, o desenvolvimento de protocolos e suas relações com a Engenharia de Software.

Apesar dos protocolos e programa produzidos para esse trabalho possuirem apenas caráter didático e não de uso estritamente prático, ainda assim, são importantes ferramentas no aprendizado prático dos conceitos vistos em aula.

Conclusões

Neste trabalho, após uma breve introdução ao problema proposto, foi apresentado inicialmente uma fundamentação teórica em três importantes camadas do modelo OSI e do modelo TCP/IP.

Posteriormente foram apresentadas as principais características do desenvolvimento de um protocolo de transporte orientado a conexão, além de suas camadas vizinhas e de uma aplicação de bate-papo, como prova de conceito. Além disso foram explicados os passos de verificação, validação e testes empregados durante o desenvolvimento do programa.

Por fim, foram apresentados comentários a respeito do trabalho e o código fonte comentado resultante.

Código fonte

Inicialização do programa

```
#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''
from camadaRede import CamadaRede
from camadaTransporte import CamadaTransporte
from camadaAplicacao import CamadaAplicacao
from interface.interface import Interface

if __name__ == '__main__':
    #Criacao da camada de rede
    camRede = CamadaRede()

    #Criacao das camadas de transporte utilizadas no programa.
    camTransOrigem = CamadaTransporte(1, camRede)
    camTransDestino = CamadaTransporte(2, camRede)

    #Associacao da camada de rede com as camadas de transporte.
    camRede.origem = camTransOrigem
    camRede.destino = camTransDestino

    #Criacao das camadas de aplicacao utilizadas no programa.
    camAplicOrigem = CamadaAplicacao(1, camTransOrigem)
    camAplicDestino = CamadaAplicacao(2, camTransDestino)

    #Associacao das camadas de transporte com as suas camadas de aplicacao
    #correspondentes.
    camTransOrigem.camadaAplicacao = camAplicOrigem
    camTransDestino.camadaAplicacao = camAplicDestino

    #Inicializacao das camadas de aplizacao
    camAplicDestino.start()
    camAplicOrigem.start()

    #Criacao da interface grafica do programa.
    interface = Interface(camAplicOrigem, camAplicDestino)

    #Inicializacao da interface grafica.
    interface.iniciar()
```

Camada de Aplicação

```
#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
```

```
'''
from threading import Thread
import logging

class CamadaAplicacao(Thread):
    def __init__(self, id, camadaTransporte):
        Thread.__init__(self)
        self.camadaTransporte = camadaTransporte
        self.aplicacao = None
        self.id = id
        self.cid = None

        self.log = logging.getLogger("computador{0}".format(self.id))

    def run(self):
        self.escutar(self.id)

    def conectar(self, remoto):
        '''
        Tentar obter uma conexao com o computador identificado por "remoto".
        '''
        self.log.info("CA: Conectando com {0}".format(remoto))
        self.cid = self.camadaTransporte.conectar(self.id, remoto)

        if self.cid > 0:
            self.log.info("CA: Conexao entre {0} e {1} estabelecida".format(self.id, remoto))
        else:
            self.log.error("CA: Conexao falhou")

    def escutar(self, t):
        '''
        Envia mensagem a camada de transporte para permanecer aguardando por
        uma nova conexao no endereco "t"
        '''
        self.cid = self.camadaTransporte.escutar(t)

    def fecharConexao(self):
        '''
        Encerra a conexao atual.
        '''
        return self.camadaTransporte.desconectar(self.cid);

    def enviarMensagem(self, s):
        '''
        Informa a camada de transporte para enviar a mensagem "s" ao computador remoto.
        '''
        self.log.info("CA: Enviando mensagem: {0}".format(s))
        bytes = [s]
        self.camadaTransporte.enviar(self.cid, bytes, 1)

    def receberMensagem(self, origem, s):
        '''
        Metodo invocado pela camada de transporte sempre que um novo pacote chega.
        '''
```

```
        self.log.info("CA: Recebi mensagem: {0}".format(s))
        if self.aplicacao != None:
            self.aplicacao.receberMensagem(s, self.id)
```

Camada de Transporte

```
#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''
from pacote import Pacote
import logging

class CamadaTransporte(object):

    def __init__(self, id, camadaRede):
        '''
        Inicializacao de configuracoes da camada de transporte. O endereco
        da camada e definido por "id" e a sua camada de rede correspondente
        e definida por "camadaRede".
        '''
        self.maxConexoes = 32
        self.tamMaxMsg = 8129
        self.tamMaxPacote = 512
        self.timeout = 20
        self.cred = 1
        self.OK = 0
        self.erroCheio = -1
        self.erroRejeitado = -2
        self.erroFechado = -3
        self.erroMenor = -3

        self.endTransporte = 0
        self.endEscuta = 0
        self.conexaoEscuta = 0
        self.acordado = True
        self.dados = []
        self.conexoes = []

        self.pacote = None

        self.camadaRede = camadaRede
        self.camadaAplicacao = None
        self.id = id

        for _ in xrange(self.maxConexoes):
            self.conexoes.append(Conexao())
            self.dados.append("")

        self.__configurarLog()
```

```
def dormir(self):
    '''
    Suspende a execucao da camada de transporte ate que seja acordada novamente
    por algum evento.
    '''
    self.log.info("CT: Indo dormir")

    self.acordado = False
    while(not self.acordado):
        pass

def acordar(self):
    '''
    Reinicia a execucao da camada de transporte.
    '''
    self.log.info("CT: Acordando")
    self.acordado = True

def paraRede(self, cid, q, m, pt, data, bytes):
    '''
    Metodo invocado pela camada de transporte para montar um pacote e
    envia-lo a camada de rede.
    '''
    self.log.info("CT: Enviando {0} para CR".format(pt))
    pacote = Pacote(cid, q, m, pt, data, bytes)
    self.camadaRede.enviarPacote(self, pacote)

def daRede(self, pacote):
    '''
    Metodo invocado pela camada de rede para informar a camada de transporte
    que um novo pacote esta disponivel no buffer da camada de rede.
    '''
    self.log.info("CT: Recebendo dado da CR")
    self.chegadaPacote(pacote, 1)

def escutar(self, t):
    '''
    Metodo para permanecer aguardando por uma nova conexao no endereco "t"
    '''
    self.log.info("CT: Escutando a rede")
    i = 1
    encontrado = 0

    for i in xrange(self.maxConexoes):
        if self.conexoes[i].estado != 'ENFILEIRADO' or self.conexoes[i].enderecoLocal != t:
            continue
        encontrado = i
        break

    if encontrado == 0:
        self.endEscuta = t
        self.dormir()
        i = self.conexaoEscuta
```

```
        self.conexoes[i].estado = 'ESTABELECIDO'
        self.conexoes[i].timer = 0
        self.conexaoEscuta = 0
        self.paraRede(i, 0, 0, 'CHAMADA_ACEITA', self.dados, 0)
        return i

    def conectar(self, local, remoto):
        '''
        Metodo para tentar estabelecer uma conexao entre "local" e "remoto"
        '''
        self.log.info("CT: Iniciando conexao")
        self.dados[0] = str(remoto)
        self.dados[1] = str(local)

        i = 31
        while self.conexoes[i].estado != 'INATIVO' and i > 0:
            i -= 1

        if self.conexoes[i].estado == 'INATIVO':
            cptr = self.conexoes[i]
            cptr.enderecoLocal = local
            cptr.enderecoRemoto = remoto
            cptr.estado = 'AGUARDANDO'
            cptr.clearRequestRecebido = 0
            cptr.creditos = 0
            cptr.timer = 0
            self.paraRede(i, 0, 0, 'REQ_CHAMADA', self.dados, 2)
            self.dormir()

            if cptr.estado == 'ESTABELECIDO':
                return i

            if cptr.clearRequestRecebido != 0:
                cptr.estado = 'AGUARDANDO'
                self.paraRede(i, 0, 0, 'CONF_LIVRE', self.dados, 0)
                return -2
            else:
                return 0
        else:
            return -1

    def enviar(self, cid, pontBuf, bytes):
        '''
        Metodo para enviar um pacote com dados armazenados no endereco
        "pontBuf" e de tamanho "bytes" bytes a conexao estabelecida previamente "cid".
        '''
        self.log.info("CT: Enviando dados")
        cptr = self.conexoes[cid]
        if cptr.estado == 'AGUARDANDO':
            return -3
        cptr.estado = 'ENVIANDO'
        cptr.numBytes = 0
        if cptr.clearRequestRecebido == 0:
```

```
            while cptr.numBytes < bytes:
                if bytes - cptr.numBytes > 512:
                    contagem = 512
                    m = 1
                else:
                    contagem = bytes - cptr.numBytes
                    m = 0
                for i in xrange(contagem):
                    self.dados[i] = pontBuf[cptr.numBytes + i]
                    self.paraRede(cid, 0, m, 'PACOTE_DADOS', self.dados, contagem)
                    cptr.numBytes += contagem
            cptr.estado = 'ESTABELECIDO'
            return 0
        else:
            cptr.estado = 'ESTABELECIDO'
            return -3

    def receber(self, cid, pontBuf, bytes):
        '''
        O usuario esta preparado para receber uma mensagem.
        '''
        self.log.info("CT: Recebendo dados")
        cptr = self.conexoes[cid]
        if cptr.clearRequestRecebido == 0:
            cptr.estado = 'RECEBENDO'
            cptr.enderecoBufferUsuario = pontBuf
            cptr.numBytes = 0
            self.dados[0] = 'CRED'
            self.dados[1] = '1'
            self.paraRede(cid, 1, 0, 'CREDITO', self.dados, 2)

        cptr.estado = 'ESTABELECIDO'
        if cptr.clearRequestRecebido != 1:
            return 0
        return -3

    def desconectar(self, cid):
        '''
        Metodo invocado para desconectar a conexao estabelecida "cid"
        '''
        self.log.info("CT: Desconectando")
        cptr = self.conexoes[cid]
        if cptr.clearRequestRecebido == 1:
            cptr.estado = 'INATIVO'
            self.paraRede(cid, 0, 0, 'CONF_LIVRE', self.dados, 0)
        else:
            cptr.estado = 'DESCONECTADO'
            self.paraRede(cid, 0, 0, 'REQ_LIVRE', self.dados, 0)
        return 0

    def chegadaPacote(self, pacote, contagem):
        '''
        Metodo invocado pela propria camada de transporte para analizar
        um pacote recebido da camada de rede e fornecer um destino adequado.
```

```
        '''
        self.log.info("CT: Chegou pacote")
        cid = int(pacote.cid)
        tipo = pacote.pt
        dados = pacote.p
        cptr = self.conexoes[cid]
        if tipo == 'REQ_CHAMADA':
            cptr.enderecoLocal = int(dados[0])
            cptr.enderecoRemoto = int(dados[1])
            if cptr.enderecoLocal == self.endEscuta:
                self.conexaoEscuta = cid
                cptr.estado = 'ESTABELECIDO'
                self.acordar()
            else:
                cptr.estado = 'ENFILEIRADO'
                cptr.timer = 20
            cptr.clearRequestRecebido = 0
            cptr.creditos = 0
        elif tipo == 'CHAMADA_ACEITA' :
            cptr.estado = 'ESTABELECIDO'
            self.acordar()
        elif tipo == 'REQ_LIVRE':
            cptr.clearRequestRecebido = 1
            if cptr.estado == 'ESTABELECIDO':
                cptr.estado = 'INATIVO'
                self.desconectar(cid)
            if cptr.estado == 'ESPERANDO' or cptr.estado == 'RECEBENDO' or \
                cptr.estado == 'ENVIANDO':
                self.acordar()
        elif tipo == 'CONF_LIVRE':
            cptr.estado = 'INATIVO'
            print "Desconectado"
        elif tipo == 'CREDITO':
            cptr.creditos += int(dados[1])
            if cptr.estado == 'ENVIANDO':
                self.acordar()
        elif tipo == 'PACOTE_DADOS':
            for i in xrange(contagem):
                cptr.enderecoBufferUsuario[0] = dados[i]

            cptr.numBytes += contagem
            self.camadaAplicacao.receberMensagem(cptr.enderecoRemoto, cptr.enderecoBufferUsuario[0])

    def relogio(self):
        '''
        Houve um pulso de relogio. Verifica timeouts de solicitacoes de conexoes
        enfileiradas.
        '''
        for i in range(1,32):
            cptr = self.conexoes[i]
            if cptr.timer > 0:
                cptr.timer -= 1
                if cptr.timer == 0:
                    cptr.estado = 'INATIVO'
```

```
                self.paraRede(i, 0, 0, 'REQ_LIVRE', self.dados, 0)

    def __configurarLog(self):
        '''
        Configura a infraestrutura de log para analise do funcionamento da
        camada de transporte.
        '''
        self.log = logging.getLogger("computador{0}".format(self.id))
        self.log.setLevel(logging.INFO)
        ch = logging.StreamHandler()
        ch.setLevel(logging.INFO)
        self.log.addHandler(ch)


class Conexao(object):

    def __init__(self):
        '''
        Definicao de uma conexao utilizada por essa implementacao de
        camada de transporte.
        '''
        self.enderecoLocal = 0
        self.enderecoRemoto = 0
        self.estado = 'INATIVO'
        self.enderecoBufferUsuario = []
        self.numBytes = 0
        self.clearRequestRecebido = 0
        self.timer = 0
        self.creditos = 0
        self.tamBuffer = 100

        for _ in xrange(100):
            self.enderecoBufferUsuario.append("")
```

Pacote

```
#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''

class Pacote(object):

    def __init__(self, cid, q, m, pt, p, bytes):
        '''
        Definicao de um pacote utilizada por essa implementacao de
        camada de transporte.
        '''
        self.cid = cid;
        self.q = q;
        self.m = m;
        self.pt = pt;
```

```
        self.p = p;
        self.bytes = bytes;
```

Camada de Rede

```
#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''

class CamadaRede(object):

    def __init__(self):
        self.origem = None
        self.destino = None

    def enviarPacote(self, emissor, pacote):
        '''
        Redireciona um pacote de uma camada de transporte para a outra.
        '''
        if emissor == self.origem:
            receptor = self.destino
        else:
            receptor = self.origem

        receptor.daRede(pacote)
```

Testes

```
#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''
import unittest
from camadaRede import CamadaRede
from camadaTransporte import CamadaTransporte
from camadaAplicacao import CamadaAplicacao


class Test(unittest.TestCase):

    def setUp(self):
        self.camRede = CamadaRede()
        self.camTransOrigem = CamadaTransporte(1, self.camRede)
        self.camTransDestino = CamadaTransporte(2, self.camRede)
        self.camRede.origem = self.camTransOrigem
        self.camRede.destino = self.camTransDestino
        self.camAplicOrigem = CamadaAplicacao(1, self.camTransOrigem)
        self.camAplicDestino = CamadaAplicacao(2, self.camTransDestino)
```

```
        self.camAplicOrigem.aplicacao= self
        self.camAplicDestino.aplicacao = self
        self.camTransOrigem.camadaAplicacao = self.camAplicOrigem
        self.camTransDestino.camadaAplicacao = self.camAplicDestino
        self.camAplicDestino.start()
        self.camAplicOrigem.start()


    def tearDown(self):
        self.camRede = None
        self.camTransOrigem = None
        self.camTransDestino = None
        self.camAplicOrigem = None
        self.camAplicDestino = None


    def testeConexaoOrigemDestino(self):
        self.camAplicOrigem.conectar(2)
        self.assertTrue(self.camAplicOrigem.cid > 0)
        self.assertTrue(self.camAplicDestino.cid > 0)

    def testeConexaoDestinoOrigem(self):
        self.camAplicDestino.conectar(1)
        self.assertTrue(self.camAplicOrigem.cid > 0)
        self.assertTrue(self.camAplicDestino.cid > 0)

    def testeEnviarPacoteOrigemDestino(self):
        self.camAplicOrigem.conectar(2)
        self.camAplicOrigem.enviarMensagem("Teste")
        self.assertEquals(("Teste",2), self.msgRecebida)

    def testeEnviarPacoteDestinoOrigem(self):
        self.camAplicDestino.conectar(1)
        self.camAplicDestino.enviarMensagem("Teste")
        self.assertEquals(("Teste",1), self.msgRecebida)

    def testeDesconectarOrigem(self):
        self.camAplicOrigem.conectar(2)
        self.assertEquals(0, self.camAplicOrigem.fecharConexao())

    def testeDesconectarDestino(self):
        self.camAplicDestino.conectar(1)
        self.assertEquals(0, self.camAplicOrigem.fecharConexao())

    def receberMensagem(self, msg, id):
        self.msgRecebida = (msg, id)

if __name__ == "__main__":
    #import sys;sys.argv = ['', 'Test.testName']
    unittest.main()
```

Interface Gráfica

```
#-*- coding: utf-8 -*-
```

```
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''

from PyQt4 import QtGui
from layoutInterface import Ui_MainWindow
import sys

class Interface(object):
    def __init__(self, camAplicOrigem, camAplicDestino):
        self.app = QtGui.QApplication(sys.argv)
        self.janela = QtGui.QMainWindow()
        self.ui = Ui_MainWindow(camAplicOrigem, camAplicDestino)
        self.ui.setupUi(self.janela)
        self.janela.show()


    def iniciar(self):
        self.ui.iniciarConexao()
        self.app.exec_()

#-*- coding: utf-8 -*-
'''
INE5414 - REDES DE COMPUTADORES I
TRABALHO PRATICO 4 - IMPLEMENTACAO DE UM PROTOCOLO DA CAMADA DE TRANSPORTE
ALUNO: PEDRO PAULO VEZZA CAMPOS
'''
from PyQt4 import QtCore, QtGui
import logging

class Ui_MainWindow(object):
    def __init__(self, camAplicOrigem, camAplicDestino):
        self.camAplicOrigem = camAplicOrigem
        self.camAplicDestino = camAplicDestino
        self.idOrigem = camAplicOrigem.id
        self.idDestino = camAplicDestino.id


    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(748, 582)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton = QtGui.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(240, 280, 98, 27))
        self.pushButton.setObjectName("pushButton")
        self.lineEdit = QtGui.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(80, 240, 261, 31))
        self.lineEdit.setObjectName("lineEdit")
        self.pushButton_2 = QtGui.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(560, 280, 98, 27))
        self.pushButton_2.setObjectName("pushButton_2")
```

```
        self.lineEdit_2 = QtGui.QLineEdit(self.centralwidget)
        self.lineEdit_2.setGeometry(QtCore.QRect(400, 240, 261, 31))
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.plainTextEdit = QtGui.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit.setEnabled(True)
        self.plainTextEdit.setGeometry(QtCore.QRect(80, 40, 261, 191))
        self.plainTextEdit.setPlainText("")
        self.plainTextEdit.setObjectName("plainTextEdit")
        self.plainTextEdit_2 = QtGui.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_2.setGeometry(QtCore.QRect(80, 330, 261, 181))
        self.plainTextEdit_2.setObjectName("plainTextEdit_2")
        self.plainTextEdit_3 = QtGui.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_3.setGeometry(QtCore.QRect(400, 40, 261, 191))
        self.plainTextEdit_3.setObjectName("plainTextEdit_3")
        self.plainTextEdit_4 = QtGui.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_4.setGeometry(QtCore.QRect(400, 330, 261, 181))
        self.plainTextEdit_4.setObjectName("plainTextEdit_4")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 748, 25))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtGui.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QObject.connect(self.pushButton, QtCore.SIGNAL("clicked()"), lambda o=1, d=2: self.__enviarP
        QtCore.QObject.connect(self.pushButton_2, QtCore.SIGNAL("clicked()"),lambda o=2, d=1: self.__enviar
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

        self.__configurarLog()

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow", "Trabalho Pratico 4 - Implemer
        self.pushButton.setText(QtGui.QApplication.translate("MainWindow", "Enviar", None, QtGui.QApplicati
        self.pushButton_2.setText(QtGui.QApplication.translate("MainWindow", "Enviar", None, QtGui.QApplica

    def iniciarConexao(self):
        self.camAplicOrigem.aplicacao = self
        self.camAplicDestino.aplicacao = self
        self.camAplicOrigem.conectar(self.camAplicDestino.id)

    def __enviarPacote(self, origem, destino):
        if origem == self.idOrigem and destino == self.idDestino:
            self.camAplicOrigem.enviarMensagem(str(self.lineEdit.text()))
        elif origem == self.idDestino and destino == self.idOrigem:
            self.camAplicDestino.enviarMensagem(str(self.lineEdit_2.text()))

    def receberMensagem(self, s, id):
        if id == self.idOrigem:
            self.plainTextEdit.appendPlainText(unicode(s))
        elif id == self.idDestino:
            self.plainTextEdit_3.appendPlainText(unicode(s))
```

```
    def __configurarLog(self):
        logOrigem = logging.StreamHandler(AdaptadorStreamLogQt(self.plainTextEdit_2))
        logOrigem.setLevel(logging.INFO)
        logDestino = logging.StreamHandler(AdaptadorStreamLogQt(self.plainTextEdit_4))
        logDestino.setLevel(logging.INFO)

        self.camAplicOrigem.log.addHandler(logOrigem)
        self.camAplicDestino.log.addHandler(logDestino)

class AdaptadorStreamLogQt(object):
    def __init__(self, campo):
        self.campo = campo

    def write(self, entrada):
        self.campo.appendPlainText(unicode(entrada[:-1]))

    def flush(self):
        pass
```

Referências Bibliográficas

[1] TANENBAUM, A. *Redes de Computadores*. 4a. ed. [S.l.]: Elsevier, 2002. ISBN 0130661023.

[2] WIKIPEDIA. *Network Layer — Wikipedia, The Free Encyclopedia*. 2010. [Online; acessado em 21 de novembro de 2010]. Disponível em: <http://en.wikipedia.org/w/index.php?title=Network_Layer&oldid=397520581>.

[3] WIKIPEDIA. *Internet Layer — Wikipedia, The Free Encyclopedia*. 2010. [Online; acessado em 21 de novembro de 2010]. Disponível em: <http://en.wikipedia.org/w/index.php?title=Internet_Layer&oldid=377971057>.