DESENVOLVIMENTO DIDÁTICO DE UMA APLICAÇÃO CLIENTE/ SERVIDOR PARA O ENSINO DE PROTOCOLOS DE COMUNICAçãO

Elisa Mannes [1]
Eduardo da Silva [2]

Resumo. A cadeira de Protocolos de Comunicação, sem exercícios práticos que ilustrem o funcionamento dos protocolos, torna-se sem atrativos para a maioria dos alunos. Por meio da análise do funcionamento dos sockets e de suas características, firma-se uma relação entre eles e os protocolos de transporte, criando um ambiente integrado que facilita a compreensão teórica dos protocolos de comunicação. Para aplicar essa integração, optou-se pelo desenvolvimento de uma aplicaçãoc liente/servidord esenvolvidan al inguagemP ython,u tilizando sockets. A utilização da programação de sockets em uma aplicação cliente/servidor possibilitou o uso de conceitos de protocolos de comunicação como o TCP e o UDP e suas características de funcionamento de uma forma prática, atingindo o escopo da disciplina de uma forma menos teórica e conseqüentemente mais compreensível.

Palavras-chave: Sockets, Redes, Protocolos

INTRODUçãO

As cadeiras que tratam de protocolos de comunicação não conseguem cativar a maioria dos alunos do curso de graduação da área de computação, por serem matérias teóricas e abstratas. De forma geral, essas cadeiras contam com pouco desenvolvimento prático. No caso do Instituto Superior Tupy (IST), e, em especial, do curso de Bacharelado em Sistemas de Informação (BSI), as disciplinas práticas ficam destinadas a cadeiras específicas, tais como Administração de Sistemas Operacionais e Hardware para Redes. Essa separação dificulta o entendimento e, conseqüentemente, o índice de motivação torna-se baixo. Percebe-se que a programação voltada para redes oferece um modo diferente de estudar o funcionamento dos protocolos, dos métodos de comunicação cliente/servidor, e do tráfego de rede em geral e, por este motivo, o desenvolvimento de uma aplicação cliente/servidor foi a forma encontrada para tornar mais atraentes os conceitos que devem ser aprendidos na cadeira de protocolos de comunicação.

Semelhante ao problema de ensino da cadeira de protocolos de comunicação encontra-se, num outro extremo, os alunos que possuem maior afinidade com as cadeiras da área de redes de computadores que, por sua vez, não se motivam a estudar programação. Isso acontece pelo fato de não verem uma ligação direta entre a utilização da programação com a área de redes de computadores. Da mesma forma, o desenvolvimento da aplicação cliente/servidor vem auxiliar tais alunos a compreenderem a importância do estudo de programação. Para serem consideradas linguagens próprias para o ensino, as linguagens de programação devem conter algumas características especiais como, por exemplo, ter uma sintaxe simples e um feedback imediato (MANNILA; RAADT, 2006). Por esse motivo, optou-se por desenvolver a aplicação cliente/ servidor na linguagem de programação Python que oferece, além de tais características, a vantagem do desenvolvimento rápido.

O objeto deste trabalho é o desenvolvimento de uma aplicação de bate-papo, no qual diversos clientes podem se conectar a um servidor central e trocar mensagens. Para o desenvolvimento de aplicações cliente/servidor, foram utilizados sockets, que são interfaces entre a camada de aplicação e a camada de transporte dentro de um computador (KUROSE; ROOS, 2006). Entende-se por aplicação cliente/servidor, uma comunicação onde os papéis das aplicações participantes são bem definidos. Nesse caso, o cliente assume um papel ativo iniciando uma comunicação com um servidor, que, por sua vez, aguarda passivamente por um contato de um ou mais clientes (COMER, 2001). A aplicação desenvolvida é composta de duas partes: um módulo servidor e um módulo cliente. Em ambos os módulos foram aplicados os conceitos de protocolos de comunicação.

O desenvolvimento desse trabalho mostrou que a programação para redes é um método eficaz para o ensino de protocolos de comunicação, pois emprega os conceitos de uma forma prática e fácil para o entendimento dos alunos.

O restante desse artigo está organizado da seguinte forma: a seção 2 trata do conceito de sockets, definindo para que eles servem e a sua importância na comunicação entre computadores de uma rede; a seção 3 refere-se às características da programação de sockets em Python e à seqüência lógica do desenvolvimento do módulo cliente e do módulo servidor; a seção 4 detalha o código fonte do programa servidor e do programa cliente com o intuito de que seja facilmente entendido e aplicado pelos alunos, explicando detalhadamente a construção das funções e a aplicação dos conceitos de sockets; a seção 5 apresenta os resultados desse trabalho, que recomendam o uso da programação para redes e a integração das cadeiras de redese p rogramação,p romovendou mm elhore ntendimentod e ambas as disciplinas.

1 DEFINIçãO DE SOCKETS E THREADS

Para o desenvolvimento de aplicações que se comunicam em redes, é necessária a utilização de sockets. Socket é uma Interface de Programação de Aplicação (API – Application Programming Interface) utilizada principalmente para a criação de aplicações cliente/servidor, na qual são definidos os mecanismos de comunicação utilizados entre os processos em execução em um ambiente de rede, em especial, os protocolos de Internet (SHELDON, 1997).

As threads permitem que sejam executados processos em quase-paralelo, ou seja, várias atividades ao mesmo tempo (TANEMBAUM, 2003). A utilização de threads em uma aplicação é justificada pela aceleração que as threads oferecem ao aplicativo e pela possibilidade de utilizar o poder das arquiteturas de computadores atuais.

1.1 SOCKETS

(1) Sociedade Educacional de Santa Catarina – SOCIESC (elisamannes@gmail.com)
(2) Sociedade Educacional de Santa Catarina – SOCIESC (eduardo@sociesc.org.br)

Os processos em máquinas diferentes comunicam-se utilizando sockets. Para a aplicação fazer a comunicação com o protocolo de transporte, é necessária a criação do socket, que é responsável por essa ligação. O cliente inicia o contato com o servidor, que precisa previamente estar preparado para o cliente iniciar o contato, tendo um socket para aceitar a comunicação do cliente (KUROSE; ROOS, 2006). O socket guarda informações sobre o endereço IP (Internet Protocol) e a porta do processo, além de especificar qual protocolo de transporte é utilizado.

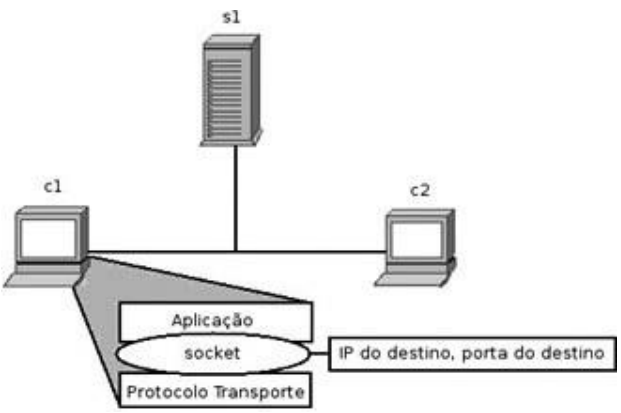

Figura 1 – Relação entre os sockets e o protocolo de transporte

Dependendo do protocolo de transporte utilizado na aplicação, dois tipos diferente de sockets podem ser utilizados (COMER, 2001). Para

protocolos orientados à conexão, como o Protocolo de Controle deT ransmissão (TCP –T ransmission Control Protocol) (POSTEL, 1981), que fornece transporte confiável de dados, um socket do tipo stream deve ser utilizado. O TCP é um protocolo orientado à conexão e, desse modo, garante à aplicação a entrega dos pacotes transmitidos. Para garantir que uma conexão utilizandoT CP seja confiável, o lado cliente doT CP faz uma apresentação de três vias (3-way handshake) entre ele e o servidor. Durante essa apresentação, o cliente conecta-se ao servidor, que por sua vez cria um socket para essa comunicação. O TCP então cria um túnel virtual, por onde os dados podem trafegar, em modo full-duplex (KUROSE; ROOS, 2006). O socket do tipo stream cria uma ligação entre o cliente e o servidor, na qual é possível iniciar, manter e encerrar a conexão. No caso da aplicação utilizar um tipo de protocolo de transporte que não seja orientado à conexão, como o UDP (POSTEL, 1980), o tipo de socket a ser utilizado é o socket datagrama. Nesse caso, o comportamento do socket muda. Como o UDP é um protocolo não orientado à conexão, ele não garante a entrega dos pacotes ao destino. O UDP não precisa de um socket de entrada e, diferente do TCP, não cria um túnel por onde trafegam os dados. Em vez disso, ele anexa ao pacote de dados uma tupla contendo o endereço IP e a porta do processo do destinatário. Contudo, não garante que o pacote chegará ao destino.

O socket stream garante a entrega dos dados, mas é mais devagar que o socket do tipo datagrama. Em contrapartida, o socket datagrama, apesar de não garantir a entrega dos pacotes, apresenta-se mais rápido na comunicação, o que pode ser vantajoso no caso de aplicações multimídia.

Quando o pacote chega ao destino, o destinatário deve desmontar esse pacote para obter os dados enviados (KUROSE; ROOS, 2006).

1.2 THREADS

Utilizou-se os conceitos de threads na aplicação para que fosse possível um mesmo servidor atender pedidos de váriosc lientess imultaneamente.N ormalmenteu mc omputador, rodando uma aplicação servidora, fica em modo de espera até que receba uma requisição de algum cliente, para então fornecer um serviço que seja solicitado. Com a aplicação de threads, os computadores podem atender vários processos simultaneamente, pois mantêm os processos revezando a capacidade de processamento do processador (MORAIS; PIRES, 2004). As threads são processos individuais que trabalham simultaneamente em um sistema operacional multitarefa. Elas também são chamadas de processos leves, pois podem compartilhar diversos recursos entre si, que fazem parte de um processo maior, também chamados de processos pesados (DEITEL; DEITEL; CHOFFNES, 2005).

Essas threads mantém várias partes de um processo rodando simultaneamente, permitindo, por exemplo, que o servidor fique esperando por conexões em uma thread e, em outra, trate da comunicação com outro cliente. Cada conexão de um cliente ao servidor será uma nova thread, e cada thread é identificada no servidor pelo socket.

2 PROGRAMAçãO DOS SOCKETS EM PYTHON

Parac ompora a plicaçãoc liente/servidor,f oramc riados dois módulos, um primeiro que gerencia a aplicação servidor chamado“ servidor”, e um segundo módulo que gerencia a aplicação cliente, chamado“ cliente”. Para fazer a

programação dos sockets e realizar a comunicação entre o cliente e o servidor é importante conhecer os sockets e os protocolos de comunicação que serão utilizados. Esse conhecimento será importante para a decisão de qual tipo de socket utilizar na aplicação, levando em consideração o serviço oferecido, para garantir a qualidade do desenvolvimento e da aplicação.

Cada uma das aplicações conta com sockets para as conexões e threads para fornecer agilidade na execução. Como protocolo de comunicação foi empregado o TCP, pela característica da garantia de entrega dos pacotes.

2.1 O MóDULO SERVIDOR

Um servidor deve possuir, entre outras, a habilidade de (i) fornecer aos seus diversos clientes um serviço especifico;
(ii) esperar passivamente pelo contato de clientes (COMER, 2001).

Primeiramente, devido à natureza desta aplicação, foi optado por utilizar o TCP como protocolo de transporte e, conseqüentemente, sockets do tipo stream. Para criação do socket, em linguagem Python, utilizou-se a seguinte linha de código:

```
socketServidor = socket(AF_INET,SOCK_STREAM)
```

Após a criação do socket, é necessário fazer a ligação do socket a um IP e uma porta, e fazer com que esse socket fique esperando por uma conexão entrante. Isso é conseguido com as seguintes linhas, na qual o endereço de IP 10.0.0.1 corresponde ao host e o número 36000 corresponde à porta de comunicação:

```
socketServidor.bind((10.0.0.1,36000))
socketServidor.listen(numeroConexoes)
```

A partir desse ponto, o servidor está pronto e aguardando por conexões dos clientes. Foi criado então, um laço que fica esperando pelas conexões do cliente e as aceite quando solicitado. A linha a seguir mostra o comando para esse procedimento: try: conn, addr = socketServidor.accept()

Deste ponto em diante, o servidor espera receber dados dos clientes, com o seguinte código:

```
nick = socketCliente.recv(qtdeDados)
```

Quando solicitado, o servidor desconecta o socket, e termina a comunicação com o cliente, conforme o comando abaixo:

```
conCliente.close(0)
```

As linhas acima descritas formam um programa servidor simples, que fica esperando por conexões de clientes, recebe as informações e, em seguida, desconecta-se. Esses comandos representam a base para a criação de comunicação entrec omputadorese mu mar ede.A p artird essai mplementação, pode-se incrementar várias outras funcionalidades que serão detalhadas na seção seguinte.

2.2 O MóDULO CLIENTE

O cliente é definido por um programa que (i) inicia o contato com o servidor; (ii) torna-se cliente temporariamente quando for necessário o acesso remoto, além de executar o processamento localmente (COMER, 2001). Do mesmo modo que o servidor, o lado cliente inicia um socket e tenta se conectar ao servidor, com um IP e porta específicos:

```
socketCliente = socket(AF_INET,SOCK_STREAM)
socketCliente.connect((ip,int(porta)))
```

Realizada a conexão, o cliente começa a enviar dados para o servidor que, conforme mostrado anteriormente, já está aguardando pelos dados do cliente. O comando abaixo envia dados para o socket, chamado socketCliente, passando como parâmetro a mensagem, que deve ser uma variável do tipo string.

```
socketCliente.send('Olá!')
```

Em seguida, o cliente desconecta-se do servidor; a comunicação é finalizada, e o cliente desconecta seu socket do servidor. O comando abaixo foi utilizado para finalizar a comunicação:

```
socketCliente.close()
```

Assim, implementa-se um cliente simples para o servidor, no qual pode-se mandar mensagens para o servidor e para outros clientes, através do servidor. Outras funcionalidades do programa cliente são descritas na seção 4. Pelo diagrama de seqüência, é possível visualizar os passos para a inicialização do servidor e do cliente e as conexões com seus respectivos sockets.

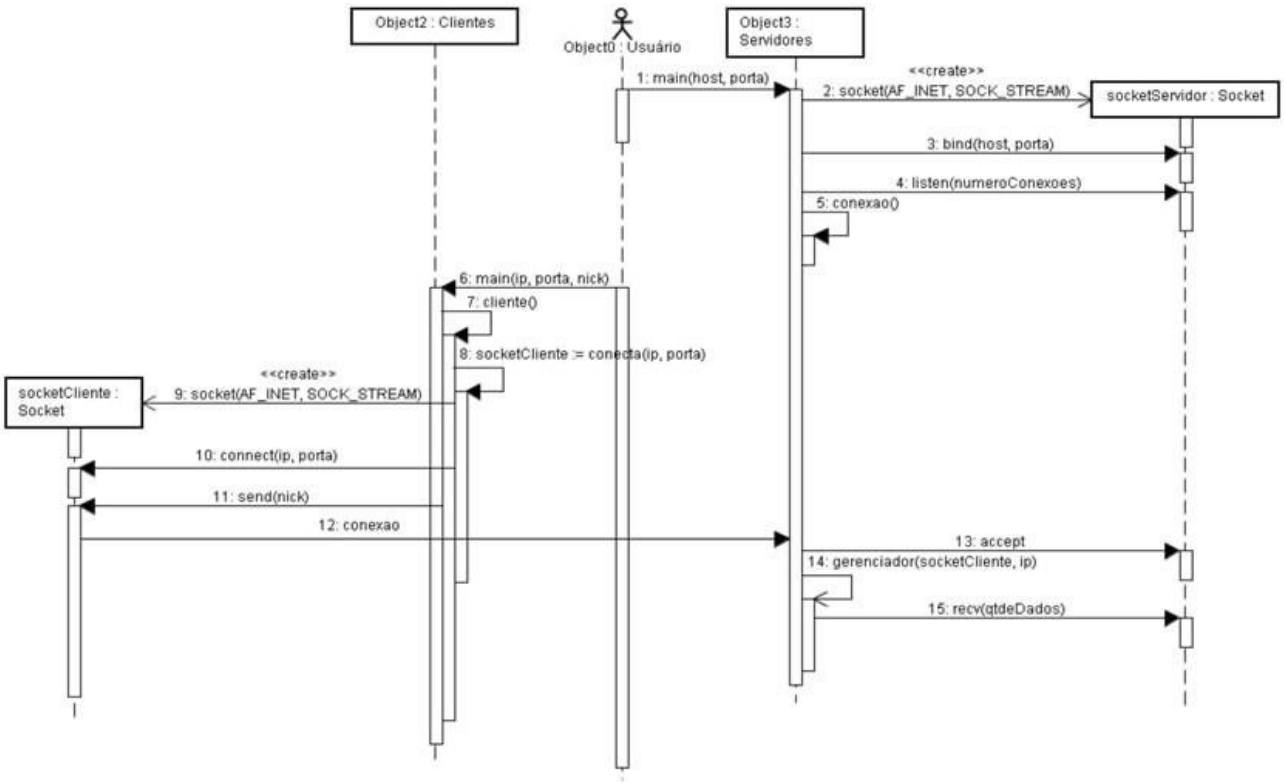

Figura 2 – Diagrama de seqüência de conexão do cliente no servidor

3 DESENVOLVIMENTO DO PROGRAMA

O desenvolvimento da aplicação foi dividido em dois programas: o servidor.py e o cliente.py, que realizam a função do servidor escutando em uma determinada porta, e a função do cliente iniciando uma conexão com o servidor, respectivamente. Em ambos os programas é necessária a importação das bibliotecas socket, threading e sys, assim como a criação de sockets threads.

Nas subseções seguintes, é apresentado o código fonte dos programas, com suas respectivas funções e características.

3.1 SERVIDOR.PY

Para iniciar o programa, define-se algumas variáveis globais, como é o exemplo da lista listaCliente[], que será responsável por armazenar o socket de todos os clientes que se conectarem ao servidor; uma variável host e uma variável porta, que levam consigo o IP do servidor e a porta de escuta do serviço de chat, respectivamente.

```
listaCliente = []
host = ‘’ porta =
36000
```

Cria-se também uma função chamada gerenciador(), que recebe como parâmetro o socket da conexão com o cliente e o IP do cliente. De posse dessas informações, a função gerenciador() insere o socket do cliente na lista e entra em um laço infinito para receber os dados enviados pelo cliente. Se houver dados provenientes do cliente, a função mostra na tela os dados e chama a função broadcast(), que será explicada adiante. Se a mensagem que o servidor receber do cliente for uma requisição para saída do programa (nesse caso, a string ‘sair’), o servidor retira o socket desse cliente da lista e termina a comunicação, chamando a função desconecta().

```
def gerenciador(socketCliente,ip):
listaCliente.append(socketCliente)      qtdeDados =
1024      nick = socketCliente.recv(qtdeDados)
print nick, “entrou na sala.”      broadcast(“%s entrou
na sala.\r\n” %
(nick))      while 1:                  msg = socketCliente.
recv(qtdeDados) if msg.strip() == “sair”:
 print nick, “ saiu da sala.”                desconecta(socketCliente, nick)
break       elif msg <> “”:
                                                broadcast(“%s diz -> %s
\r\n” % (nick,msg))
                                                  print “%s diz -> %s” %
(nick,msg)                else:                  print nick, “ saiu da sala.”
desconecta(socketCliente, nick)          break
```

Af unçãod esconectar ecebe,c omop arâmetro,o socket do cliente que está sendo desconectado e o apelido (nick) desse usuário. Ela remove o socket da lista de usuário listaCliente e em seguida finaliza o socket do usuário, com o método shutdown(), informando os demais usuários que esse usuário deixou o programa.

```
def desconecta(conCliente, nick): listaCliente.remove(conCliente)
conCliente.shutdown(1)      broadcast(“%s saiu da sala. \r\n” % (nick))
```

A fim de realizar a entrega da mensagem enviada pelos clientes para todos os clientes conectados no servidor, foi implementada uma função chamada broadcast(), que recebe como parâmetro o texto a ser enviado aos clientes. Essa função percorre toda a lista de clientes armazenada na lista listaCliente[] e envia a mensagem para cada um dos clientes, utilizando o método send( ).

```
def broadcast(mensagem): for usuario in
listaCliente:       usuario.send(mensagem)
```

A função conexão contém em si toda a estrutura de criação de sockets. Ela aceita as conexões que são endereçadas a esse servidor, e abre uma nova thread da função gerenciador. def conexao():

```
        try:   conn, addr = socketServidor. accept()
        threading._start_new_ thread(gerenciador, (conn, addr))
except:
```

```
return None
```

Como parte da função principal do programa, cria-se uma variável contendo o número de conexões simultâneas que desejamos ter em nosso servidor, e então coloca-se o socket para ouvir na porta especificada anteriormente. Cria-se um laço para que o servidor aguarde pelo contato de algum cliente, que executa a função conexão().

```
if __name__ == "__main__":              print
"Servidor Ativo."

        numeroConexoes = 10         socketServidor =
socket(AF_ INET,SOCK_STREAM)
        socketServidor.bind((host,int
(porta)))
        socketServidor. listen(numeroConexoes)
        print "Aguardando conexões"             while 1:
                conexao()
```

3.2. CLIENTE.PY

Por ser uma linguagem com tipagem dinâmica (AGARWAL; AGARWAL, 2005), o Python não exige que sejam declaradas as variáveis antes de utilizá-las, então o programa cliente não tem a necessidade de declarar variáveis globais, que precisem ser definidas antes das funções. Primeiramente, criou-se uma função de conexão, que contém o início dos sockets com o servidor.

```
def conecta(ip,porta):
        socketCliente = socket(AF_INET,SOCK_ STREAM)
socketCliente. connect((ip,int(porta)))     return socketCliente
```

A função cliente() conecta efetivamente o cliente com o servidor, chamando a função conecta() e enviando para o servidor o nick escolhido pelo usuário ao iniciar o cliente.

```
def cliente():
socketCliente = conecta(argv[1],argv[2])    nick = sys.argv[3]
socketCliente.send(nick)     print "Você entrou na sala."
conexao(socketCliente, nick)
```

Para gerenciar as entradas do usuário, criou-se a função conexao(), que tem como tarefa iniciar uma nova thread para a função que escuta o servidor, esperar pela entrada de dados do usuário, e enviar para o servidor o conteúdo da variável msg.

```
def conexao(socketCliente,nick): try:
                threading._start_
new_thread(escutaServidor, (socketCliente,nick))
        while 1:
                msg = raw_input()
        if msg.strip() == 'sair':
        socketCliente.
send(msg)
        print "Saindo da sala...."
socketCliente.shutdown(1)               exit(0)
        socketCliente.send(msg)
except KeyboardInterrupt:
        socketCliente.shutdown(1)       print
"Saindo da sala..."      exit(0)
```

A função escutaServidor(), que é chamada na função conexão é responsável por ouvir o servidor, e, quando houver mensagens oriundas do servidor, essa função exibe a mensagem que o usuário recebeu de outros clientes através do servidor.

```
def escutaServidor(socketCliente,nick): while 1:
        msg = socketCliente.recv(1024)      if
msg <> '':
                print
        print msg       print
print nick,": " ,
```

Na parte principal do programa, coloca-se a chamada para a função cliente(), que por sua vez dá início ao processo de criação do socket e conexão com o servidor.

```
if __name__ == "__main__": print
"Executando o cliente.\n"
cliente()  exit(0)
```

CONCLUSãO

Neste trabalho, apontou-se a dificuldade da compreensão do conteúdo das disciplinas de redes de computadores, geralmente teóricas. Sugere-se, através do desenvolvimento de programas cliente/servidor, no qual são amplamente empregados conceitos de protocolos de comunicação, uma proposta acadêmica para a

aplicação do conteúdo de cadeiras de redes de computadores. A programação de sockets reproduz a idéia do que realmente constitui uma comunicação de rede, do nível da aplicação ao nível de transporte de dados e, deste modo, pode-se (tornar prática uma disciplina normalmente teórica.

As principais contribuições desse trabalho são (i) aplicação prática dos conceitos de protocolos de comunicação; ii) a interligação de programação com conceitos de redes; (iii) a facilidade do desenvolvimento da aplicação cliente/servidor na linguagem Python.

Com este tipo de abordagem, é possível introduzir o aluno em novas descobertas, como a melhora do desempenho de comunicação em sistemas operacionais e a melhor utilização do poder de processamento do processador. Desta forma, a programação para redes apresenta um caminho ideal para realizar o ensino de protocolos de comunicação, formando profissionais com melhor base para a construção, administração e manutenção de um ambiente de rede.

Em trabalhos futuros, planeja-se (i) melhorar a funcionalidade dos módulos cliente e servidor, permitindo a criação de comunicações privadas entre clientes; (ii) agregar novos conceitosd ep rotocolosd ec omunicaçãon od esenvolvimentod e aplicações em Python; (iii) o desenvolvimento de aplicações que utilizem outros protocolos de transportes, com características diferentes doT CP, como o Stream ControlT ransmission Protocol (SCTP) (STEWART, 2000).

REFERÊNCIAS

Abstract. Without practical exercises to illustrate how the protocols could be implemented, the protocol classes are not attractive to most students. Studying the characteristics of sockets and its implementation, we could trace a relationship between the sockets and transport protocols, creating an integrated environment that facilitates the theoretical understanding of protocols. To apply this integration, a client/server application in Python was developed using sockets.T he application of sockets in a client/server application allows the students to use protocols concepts like TCP and UDP and their functionality in a practical way, making it more comprehensible.

Keywords: Sockets, Network, Protocols

AGARWAL, K.K.; AGARWAL, A. Python for CS1 CS2 and beyond. South Center Conference, 2005.

COMER, D.E. Redes de Computadores e Internet. 2. ed. Porto Alegre: Bookman, 2001.

DEITEL, H.M.; DEITEL P.J.; CHOFFNES, D.R. Sistemas Operacionais. 3. ed. São Paulo: Pearson, 2005.

KUROSE, J.F.; ROOS, K. W. Redes de Computadores e a Internet. 3 ed. São Paulo: Addison Wesley, 2006.

MANNILA,L.;RAADT,M.D.AnObjectiveComparisonofLanguagesforTeachingIntroductoryProgramming.ProceedingsKoliCalling,2006.

MORAIS, P.; PIRES, J.N. Python - Curso Completo. Lisboa: FCA, 2004.

POSTEL, J. RFC 793 Transmission Control Protocol. 1981.

POSTEL, J. RFC 768 User Datagram Protocol. 1980.

SHELDON, T. Encyclopedia of Networking and Telecommunications. 2. ed. New York: McGraw Hill, 1997.

STEWART, R. RFC 2960 Stream Control Transmission Protocol. IETF Network Working Group, 2000.

TANENBAUM, A.S. Redes de Computadores. 4. ed. Rio de Janeiro: Campus, 2003.