

Monografia para o projeto de Redes de Computadores (SSC0641)

Elisa J. Marcatto¹, Victor M. Nunes¹

¹Departamento de Sistemas de Computação
Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)

1. Introdução

A comunicação entre pessoas é algo essencial em nosso cotidiano, seja para fins estritamente sociais, seja para fins empresariais. Assim, os esforços para torná-la cada vez mais eficaz são constantes, o que leva ao desenvolvimento de aplicações que satisfaçam esta condição e permitam que pessoas se comuniquem de forma eficiente.

É neste cenário que este projeto se insere, sendo, resumidamente, caracterizado como uma aplicação para troca de mensagens em uma arquitetura P2P, permitindo a comunicação entre usuários e o aprimoramento de conceitos aprendidos na disciplina de Redes de Computadores. A ferramenta abrange todas as funcionalidades necessárias para que a comunicação seja realizada com bom desempenho, que são adicionar contato, listar contatos, enviar mensagem, mensagem em grupo e sair.

2. Fundamentos teóricos

Para o desenvolvimento deste trabalho foram utilizados conceitos adquiridos na disciplina de Redes de Computadores, envolvendo aspectos da arquitetura *peer-to-peer* (P2P), da implementação de *sockets* e do uso *threads*.

2.1. Arquitetura P2P

Segundo [Brito and Moura 2005] Uma arquitetura de rede é caracterizada por ser o conjunto de camadas e regras de comunicação que devem ser estabelecidos numa rede. Uma das arquiteturas mais usadas atualmente é a arquitetura P2P, que é definido pelo compartilhamento de recursos computacionais e serviços através da comunicação direta e descentralizada entre os sistemas envolvidos [Ooi et al. 2003]. Estes recursos e serviços incluem, dentre outras coisas, a troca de informações, ciclos de processamento e espaço de armazenamento em disco. Os sistemas P2P se aproveitam do poder computacional e da conectividade de computadores convencionais para, de forma barata, tornar estes recursos acessíveis entre os nós do sistema, denominados *peers*. Sistemas P2P podem ser classificados quanto a sua arquitetura em 3 tipos básicos:

- parcialmente centralizada: contém um servidor central, responsável pelo mecanismo de busca e manutenção da infra-estrutura, deixando a cargo dos *peers* participantes, o compartilhamento de recursos e serviços de forma distribuída;
- descentralizada: não possui um *peer* central e os mecanismos de busca e manutenção da infra-estrutura, assim como o compartilhamento de recursos, serviços e conteúdos estão distribuídos pela rede, em cada *peer* participante;

- *super-peer*: é formada por um subconjunto de *peers* de maior poder computacional interligados entre si, denominados *super-peers*. Estes são responsáveis pelo gerenciamento e compartilhamento de recursos, tendo cada um deles outros *peers* conectados a si.

Devido a estas características, tem-se que, para uma aplicação de troca de mensagens, a arquitetura P2P é a mais eficiente, já que permite um envio distribuído das informações, proporcionando um melhor desempenho para este tipo ferramenta. Dessa forma, a especificação proposta para o uso da arquitetura P2P condiz perfeitamente com sua estruturação, fazendo do projeto uma aplicação bem formulada.

2.2. Sockets

Socket é uma interface de software que permite o envio e o recebimento de mensagens de um processo de aplicação, ou seja, representa um ponto de conexão para uma rede TCP/IP, que é o conjunto de protocolos usado pela Internet. Um *socket* é uma conexão de dados transparente entre dois computadores numa rede e é identificado pelo endereço de rede dos computadores e por seus pontos finais e uma porta em cada computador. Os computadores em rede direcionam os *streams* de dados recebidos da rede para programas receptores específicos, associando cada programa a um número diferente (porta do programa).

Conforme dito por [da Silva 2000], os sockets têm dois modos principais de operação: o modo baseado em conexões e o modo sem conexão, que estão relacionados, respectivamente, com o protocolo TCP (Transport Control Protocol) e o UDP (User Datagram Protocol). No primeiro modo, o socket precisa se conectar ao destino antes de transmitir os dados. Uma vez conectados, os *sockets* são acessados pelo uso de uma interface de fluxos: abertura-leitura-escrita-fechamento. Tudo que é enviado por um socket é recebido pela outra extremidade da conexão, exatamente na mesma ordem em que foi transmitido. Já no segundo modo, o *socket* não precisa se conectar a um outro de destino, ele simplesmente envia o datagrama. O protocolo UDP só promete fazer o melhor esforço possível para tentar entregar o datagrama.

Assim, os *sockets* são essenciais no estabelecimento de uma conexão, é através dele que se configuram o protocolo de transporte usado e que se definem certas especificações de nível de uma camada, podendo ser descrito como o elo entre a aplicação e a rede.

2.3. Threads

Threads podem ser descritas como processos "leves", os quais nada mais são do que um programa em execução. Elas acrescentam no modelo de processo o fato de múltiplas execuções poderem ser realizadas no mesmo ambiente com grande grau de independência, já que compartilham o espaço de endereçamento e os recursos, assim como a "leveza", já mencionada, ao gerenciar suas funcionalidades. O suporte à *thread* é fornecido pelo próprio sistema operacional, podendo ser implementada ao nível do núcleo, ou através de uma biblioteca de uma determinada linguagem. Uma *thread* permite, por exemplo, que o usuário de um programa utilize uma funcionalidade do ambiente enquanto outras linhas de execução realizam outros cálculos e operações.

Ao utilizar *threads* no contexto de gerência de conexões, tem-se a possibilidade de receber várias solicitações ao mesmo tempo e gerenciá-las de forma eficiente, permitindo que a operação de receber e enviar mensagens simultaneamente apareçam para o usuário como algo transparente.

3. Funcionamento da Aplicação

A aplicação se baseia nas funcionalidades de adicionar contato, listar contatos, enviar mensagem, mensagem em grupo e sair, sendo que cada uma delas está relacionada a uma especificidade de gerenciamento de conexões. Ao iniciar a execução, a primeira atividade realizada pelo programa é a criação de um *socket* para que possa ficar "ouvindo" quando se deseja estabelecer uma conexão com *host*, além disso, é criada uma *thread* que manipula todas estas conexões de forma paralela, permitindo que várias delas possam ser aceitas de forma eficiente. A Figura 1 é um fluxograma que representa a sequência das atividades realizadas pelo programa nesta etapa, envolvendo todas as verificações e os estabelecimentos, realizadas de maneira transparente ao usuário.

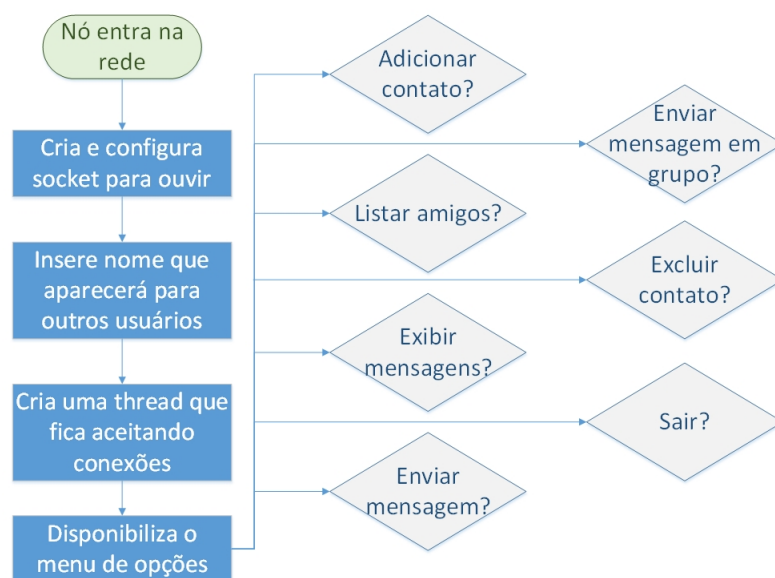


Figura 1. Fluxograma do menu

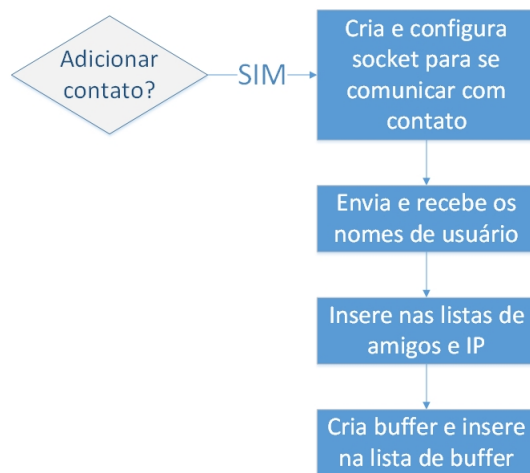


Figura 2. Adicionar contato

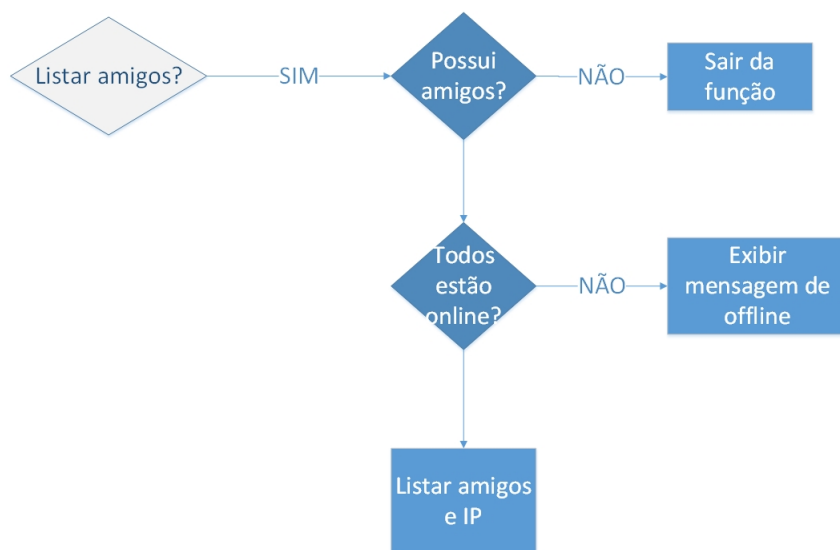


Figura 3. Listar amigos

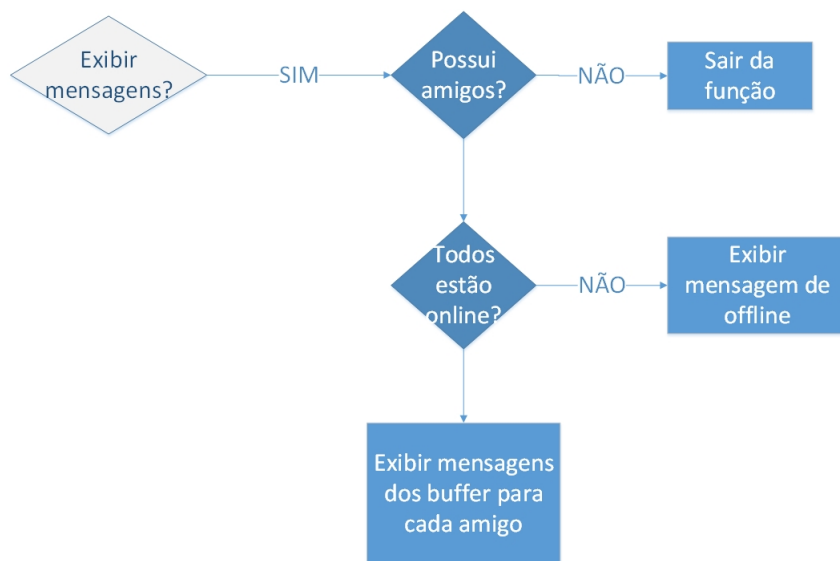


Figura 4. Exibir mensagens

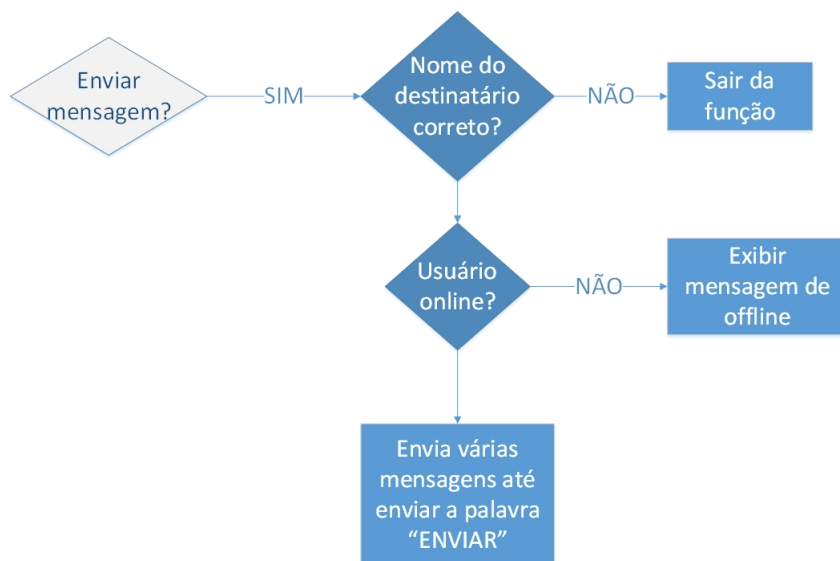


Figura 5. Enviar mensagem

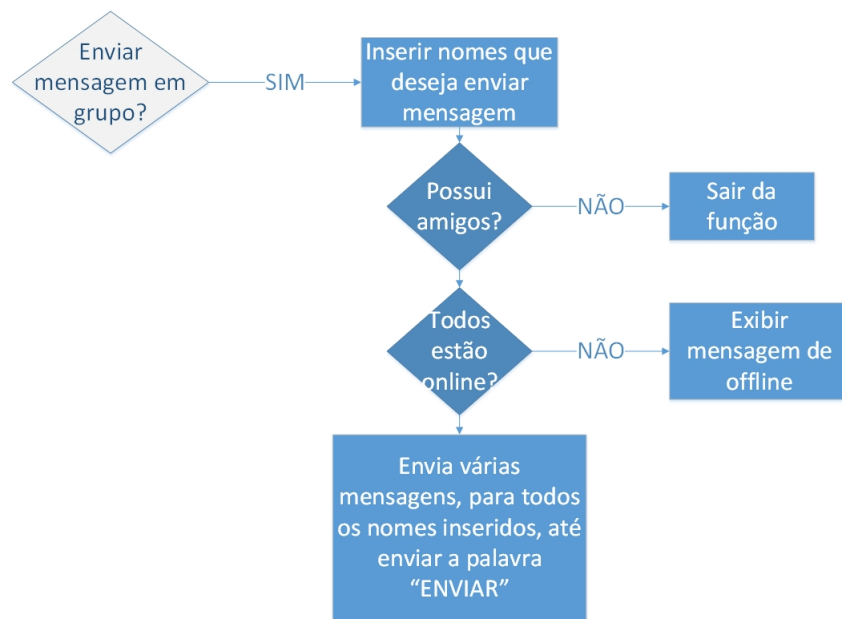


Figura 6. Enviar mensagem em grupo

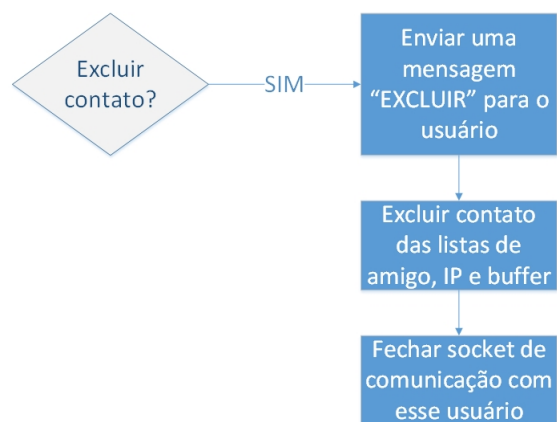


Figura 7. Excluir contato

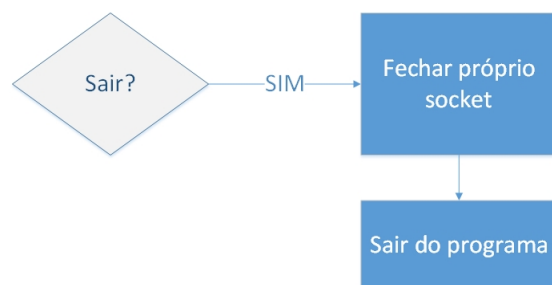


Figura 8. Sair

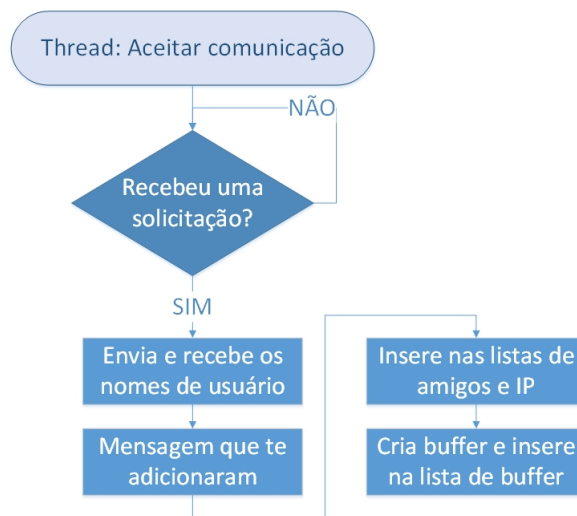


Figura 9. Thread de aceitar solicitações

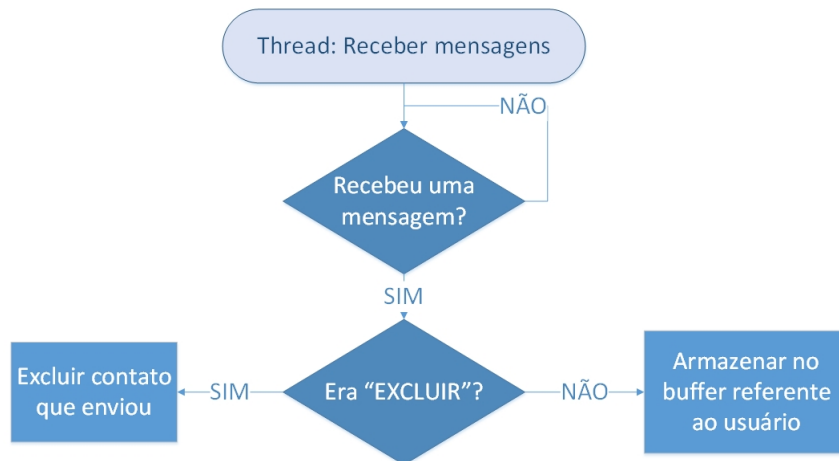


Figura 10. Thread de receber mensagens

Assim, é exibido na primeira tela Figura 11 apenas para o usuário inserir seu nome, o qual será usado referenciá-lo nas conversas de outras pessoas. Após o usuário inserir seu nome, a segunda tela Figura 12 é exibida, que é o menu para que ele possa selecionar qual das funcionalidade deseja selecionar. O menu organiza cada comando disponível através das teclas de 1 a 7, que serão descritos a seguir.

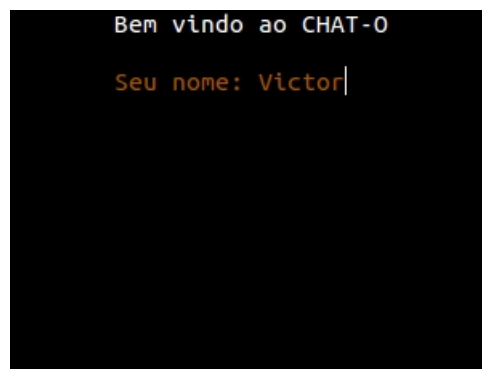


Figura 11. Primeira tela: usuário entrando na rede



Figura 12. Segunda tela: menu de opções para o usuário

3.1. Adicionar contato

Ao adicionar um amigo, o usuário está estabelecendo uma conexão com outro *host*, sendo possível o fluxo de mensagens entre eles. Para isso, é necessário que aquele que adiciona insira o IP do contato a ser adicionado. Assim que um contato é adicionado, ele recebe uma mensagem de quem acabou de se tornar seu amigo, conforme explicitado na Figura 13. Também é importante ressaltar que, adicionando um contato, cria-se uma nova *thread* (diferente daquela já mencionada) para gerenciar o recebimento e envio de mensagens, tratando todo fluxo de informações que passarão pela conexão estabelecida.

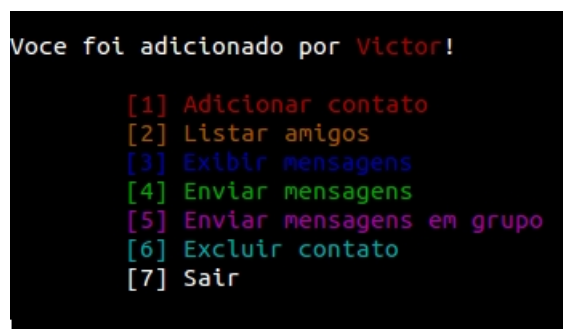


Figura 13. Adicionar: mensagem mostrando que o usuário foi adicionado por outro

3.2. Excluir contato

Excluindo um contato, o usuário estará fechando uma conexão, o que implica na necessidade de gerenciar também o contato excluído, inspecionando toda sua estrutura de contato e conexão. Ao ser excluído, a mensagem, mostrada na Figura ??, é mostrada na tela.

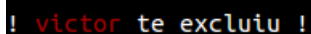


Figura 14. Excluir: mensagem mostrando que o usuário te excluiu

3.3. Enviar mensagens

Esta funcionalidade é gerenciada por uma *thead*, como mencionada anteriormente, e é exibida conforme Figura 15. Para enviar uma mensagem, é verificado,

internamente ao programa, se o outro *host* está *online* e se ele está adicionado na lista de contatos.

```
- Amigos:
# Victor

Nome do contato: Victor

!Digite ENVIAR para enviar a mensagem e voltar ao menu!

Para Victor:

Ola
ENVIAR|
```

Figura 15. Enviar mensagem: modo de envio de mensagem para outro usuário

3.4. Enviar mensagens em grupo

Enviar mensagens em grupo é semelhante à funcionalidade anterior, porém difere que vários contatos recebem uma mesma mensagem e são inspecionados quanto aos mesmos aspectos (existência na lista e conexão disponível). A tela mostrada ao usuário é Figura 16.

```
- Amigos:
# Victor

Insira os contatos para enviar a mensagem em grupo e digite 'FIM':
Contato 0: Victor

!Digite ENVIAR para enviar a mensagem e voltar ao menu!
Mensagem em grupo:
Olá grupo!
ENVIAR|
```

Figura 16. Enviar mensagens em grupo: modo de enviar mensagens para um grupo de pessoas

3.5. Listar amigos

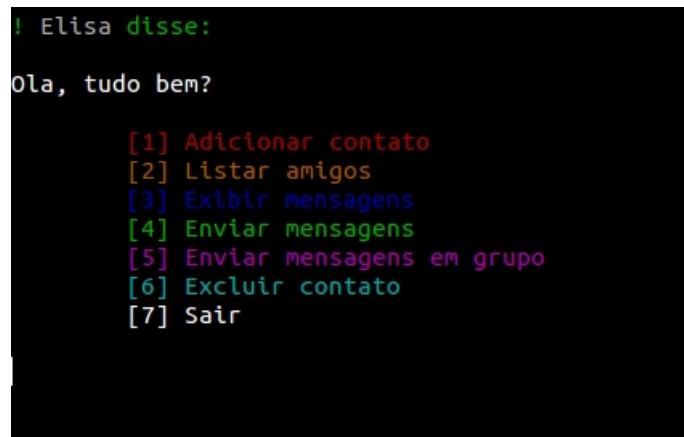
Esta funcionalidade busca na estrutura de um usuário as conexões ainda ativas, listando todos os contatos correspondentes. A tela exibido é a Figura 17.

```
2
- Amigos:
# elisa          node02
# henrique       192.168.0.30
```

Figura 17. Listar amigos: modo que aparece os amigos adicionados e seu IP ou nó

3.6. Exibir mensagens

Esta funcionalidade verifica em um *buffer* existente relacionado a cada contato, as mensagens trocadas, o que é exibido como mostra Figura 18.

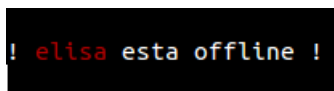


```
! Elisa disse:  
ola, tudo bem?  
  
[1] Adicionar contato  
[2] Listar amigos  
[3] Exibir mensagens  
[4] Enviar mensagens  
[5] Enviar mensagens em grupo  
[6] Excluir contato  
[7] Sair
```

Figura 18. Exibir mensagens: modo de receber mensagens de outros usuários

3.7. Sair

Ao sair da aplicação, o usuário estará fechando sua própria conexão, assim é necessário que os outros usuários tenham essa informação. Dessa forma, foi implementado um esquema de gerenciamento que avisa aos demais contatos que algum usuário da lista de amigos saiu do estado disponível, como mostra a Figura 19.



```
! elisa esta offline !
```

Figura 19. Sair: mensagem mostrando que o usuário está offline

4. Considerações finais

Na elaboração do projeto, foi possível aprimorar o aprendizado na disciplina de Rede de Computadores, envolvendo aspectos práticos associados à comunicação entre *hosts* e à troca de mensagens. Foram encontradas dificuldades relacionadas à implementação da comunicação entre os usuários, já que não tivemos nenhuma experiência prévia com *sockets*, e em sua sincronização, atrelada ao uso de *threads*, para permitir o recebimento de mensagens de múltiplos contatos. O gerenciamento de queda de conexão adicionou complexidade ao tratar as funcionalidades, já que antes de realizar qualquer comando, é necessário verificar se um *host* está *online* ou não. Superados estes obstáculos, o trabalho foi implementado sem mais problemas, sendo testado tanto na rede local (entre dois usuários), quanto no cluster do LaSDPC (entre mais de dois usuários).

Referências

Brito, G. A. D. D. and Moura, A. M. d. C. (2005). Sistema rosa-p2p: uma arquitetura distribuída para integração de objetos de aprendizagem. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 1, pages 190–200.

da Silva, F. A. (2000). Comunicação de computadores utilizando sockets.

Ooi, B. C., Shu, Y., and Tan, K.-L. (2003). Relational data sharing in peer-based data management systems. *SIGMOD Rec.*, 32(3):59–64.