

kernel de versão 2.6.9 da distribuição *Slackware*, versão 9.0.

Os aplicativos que implementam as entidades do ambiente de simulação fazem uso da arquitetura de protocolos TCP/IP. O aplicativo NIST *Net* atua na camada de rede, e os aplicativos que implementam o “Nó 1” e o “Nó 2” atuam na camada de aplicação em dois momentos distintos: primeiramente utilizando o protocolo TCP na camada de transporte, e depois utilizando o protocolo SCTP. A Figura 4.12 mostra a pilha de protocolos utilizada por cada entidade, assim como o relacionamento entre as diversas camadas.

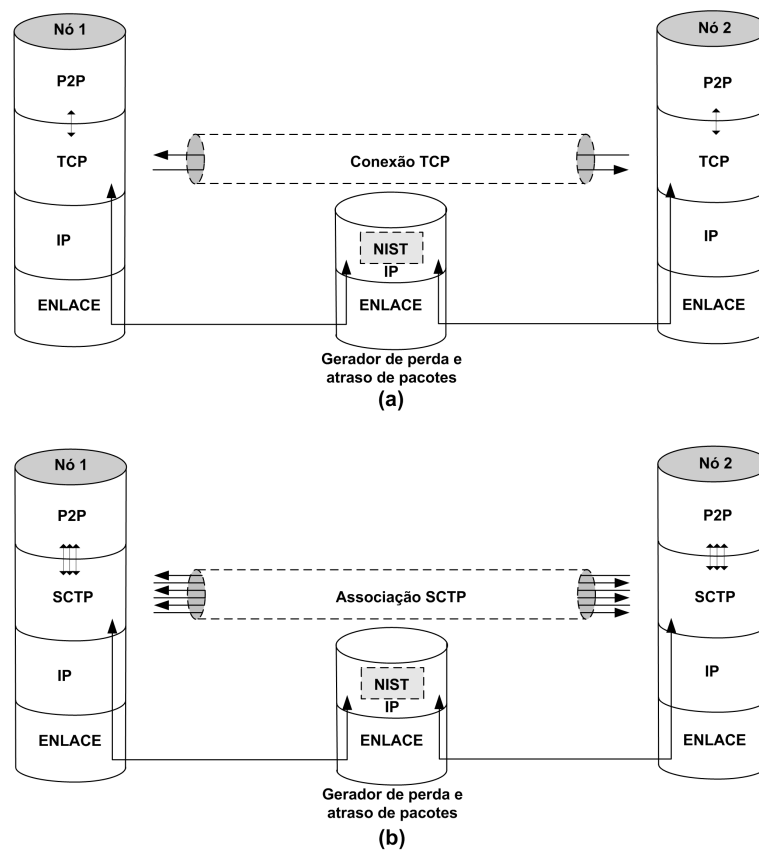


Figura 4.12: As pilhas de protocolos utilizadas pelos aplicativos do ambiente de simulação

A Figura 4.12 (a) representa a pilha de protocolos com a utilização do TCP. A conexão TCP é representada como a comunicação lógica entre dois processos rodando em dois computadores diferentes, e as setas, uma em cada sentido, representam a natureza *full-duplex* da conexão. O relacionamento entre a camada de aplicação, implementando um aplicativo *peer-to-peer*, e a camada de transporte, implementando o TCP, é representada por uma única seta, pois há uma única via, *full-duplex*, através da qual o processo se comunica com a camada de transporte.

Na Figura 4.12 (b), o destaque é dado pelo uso do SCTP na camada de transporte. A associação SCTP possui várias setas em cada sentido pois permite que diversos fluxos sejam utilizados para a comunicação entre processos em diferentes computadores, através de uma única associação. As setas indicando o relacionamento entre a camada de aplicação, implementando um aplicativo *peer-to-peer*, e a camada de transporte, implementando o SCTP, representam as múltiplas vias pelas quais o aplicativo *peer-to-peer* pode receber e enviar suas mensagens utilizando os multi-fluxos.

A aplicação que implementa o “Nó 1”, ou aplicação geradora e receptora de pacotes, cria duas *threads*, sendo que uma delas gera e transmite mensagens, e outra recebe mensagens. A aplicação que implementa o “Nó 2” atua simplesmente como *echo server*. O código fonte destas aplicações pode ser encontrado na íntegra no Apêndice A. Para essa implementação, fez-se uso da linguagem C, com utilização das bibliotecas *pthread lib*, e da *libsctp* [12, 13, 40].

O “Nó 1” atua como gerador e receptor de pacotes, simulando o tráfego de mensagens dentro de uma rede *overlay*. Os pacotes gerados por esse nó são enviados ao “Nó 2”. O “Nó 2” atua como um *echo server*, enviando as mensagens de volta ao “Nó 1”, assim que estiverem disponíveis. Desta maneira, é possível efetuar cálculos de desempenho individualmente para cada mensagem, em que o trânsito de uma mensagem do “Nó 1” ao “Nó 2”, e do “Nó 2” ao “Nó 1” representam dois *hops* da mensagem na rede *overlay*. O funcionamento destas aplicações é mostrado nos diagramas da Figura 4.13.

A Figura 4.13 (a) mostra o funcionamento do aplicativo gerador e receptor de pacotes, ou “Nó 1”, e a Figura 4.13 (b) mostra o funcionamento do aplicativo que atua como *echo server*, ou “Nó 2”.

O aplicativo que atua como *echo server*, mostrado na Figura 4.13 (b), simplesmente recebe o pacote e o devolve ao nó que o enviou. Caso esse pacote seja enviado fazendo o uso do protocolo SCTP, o *echo server* devolve esse pacote através do mesmo fluxo que o recebeu.

O aplicativo gerador e receptor de pacotes cria duas *threads*: uma que gera e envia os pacotes, e outra que recebe os pacotes que foram devolvidos pelo *echo server*. A primeira *thread* gera e envia os pacotes de acordo com parâmetros definidos pelo

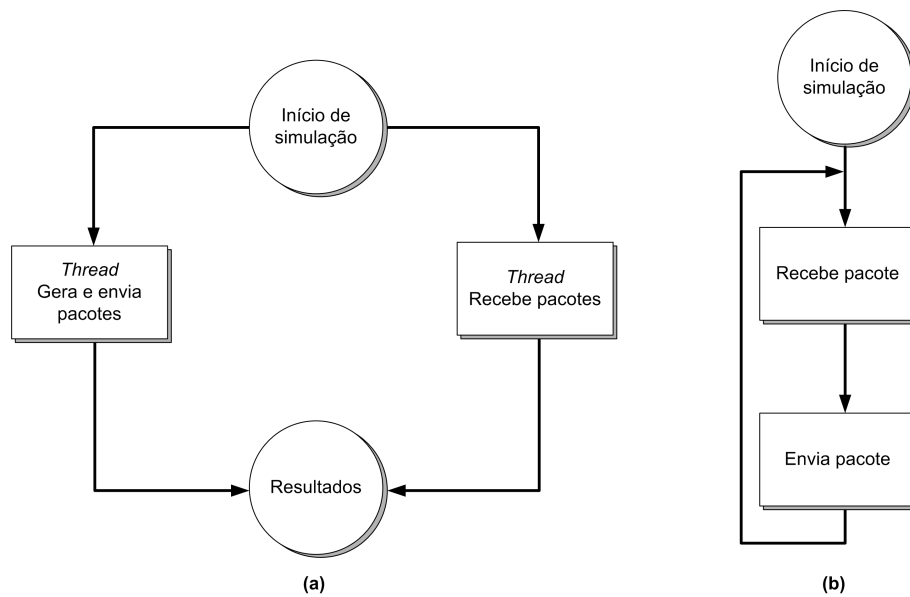


Figura 4.13: As fases de funcionamento dos aplicativos gerador e receptor de pacotes e *echo server*

usuário. Um desses parâmetros é o protocolo da camada de transporte, escolhido entre o TCP e o SCTP. Os parâmetros comuns a ambos os protocolos são:

- Endereço local e porta local: Um único endereço e uma única porta local devem ser especificados para iniciar o aplicativo gerador e receptor de pacotes;
- Endereço remoto e porta remota: Especifica o endereço e a porta do *echo server*;
- Número de mensagens a enviar: Para cada experimento, várias de mensagens são encaminhadas ao *echo server*, e essa quantidade de mensagens é definida pelo usuário por meio deste parâmetro;
- Intervalo de tempo entre as mensagens: Esse parâmetro, medido em milisegundos, representa o espaço de tempo entre cada nova mensagem gerada e enviada ao *echo server*. Por meio dele, é possível definir quantas mensagens o aplicativo vai enviar em um espaço de tempo. Pretende-se, com esse parâmetro, simular maiores e menores taxas de tráfego de mensagens entre os nós;
- Tamanho dos pacotes: As mensagens em uma rede *peer-to-peer* podem apresentar diferentes tamanhos. Esse parâmetro permite definir o tamanho, em *bytes*, das mensagens que são encaminhada ao outro nó;

Quando o SCTP é selecionado como protocolo da camada de transporte, outros parâmetros podem ser configurados, os quais fazem uso de funcionalidades não presentes no TCP:

- Número de fluxos: Esse parâmetro define a quantidade de fluxos que o aplicativo deve utilizar para enviar as mensagens. Cada uma delas vai ser enviada a um fluxo diferente, segundo a política *round-robin*;
- Envio desordenado de mensagens: Se este parâmetro for habilitado, as mensagens serão enviadas de forma desordenada, independentemente do número de fluxos utilizado.

Cada *thread* do aplicativo gerador e receptor de pacotes possui etapas de funcionamento distintas. A Figura 4.14 mostra o funcionamento da *thread* geradora de pacotes, e a Figura 4.15 mostra o funcionamento da *thread* de recebimento de pacotes.

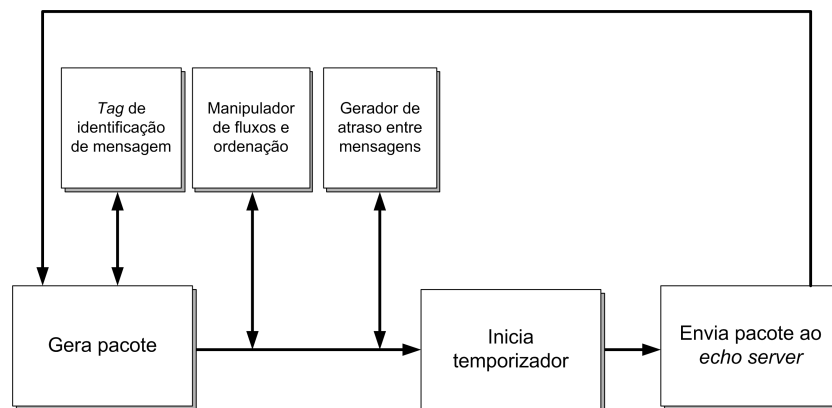


Figura 4.14: O funcionamento da *thread* geradora de pacotes

Quando uma mensagem é gerada, um “*tag* de identificação” é definido de forma que, no recebimento desta mensagem, seja possível interromper o seu temporizador específico. O “manipulador de fluxos e ordenação” é utilizado pelo protocolo SCTP a partir dos parâmetros “número de fluxos” e “envio desordenado de mensagens”. É nesta entidade que é definido através de qual fluxo uma mensagem é enviada e, também, se a mensagem vai ser enviada de forma desordenada.

O “gerador de atraso entre mensagens” suspende a execução da *thread* entre o envio de cada mensagem por um tempo definido pelo usuário. O objetivo desse atraso, é simular um número determinado de mensagens que são enviadas num espaço de

tempo. Assim que a mensagem é liberada para ser enviada, um temporizador, associado ao seu identificador, é iniciado. A mensagem é, então, enviada ao *echo server*, que, ao receber a mensagem, a envia de volta ao “Nó 1”.

Quando o SCTP é utilizado como protocolo da camada de transporte, as mensagens enviadas são devolvidas pelo *echo server* através do mesmo fluxo que as receberam, e se forem enviadas de forma desordenada, devem ser devolvidas da mesma maneira.

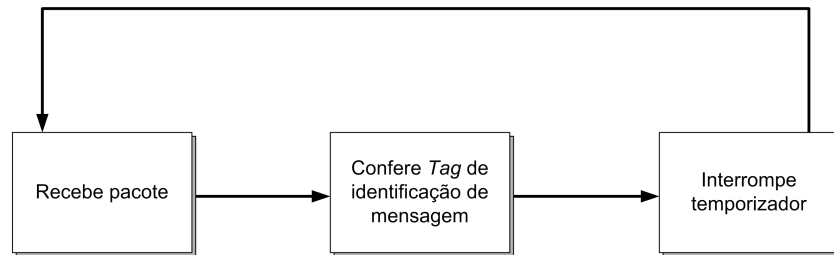


Figura 4.15: O funcionamento da *thread* receptora de pacotes

A *thread* receptora de pacotes, com suas etapas mostradas na Figura 4.15, confere o *tag* de identificação da mensagem e interrompe o temporizador ao qual está associada. Com isso, é possível obter a latência de envio e recebimento para cada mensagem, como exemplificado nas Figuras 4.16, 4.17 e 4.18.

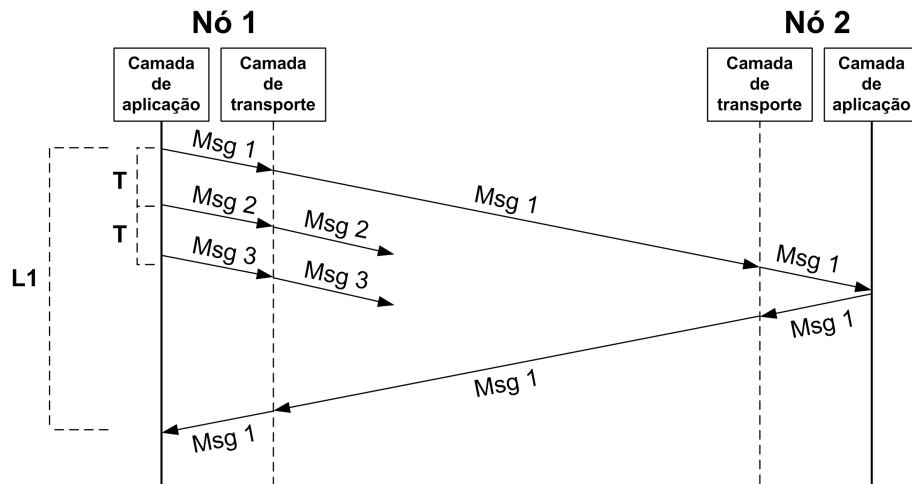


Figura 4.16: A cálculo da latência de uma mensagem - Exemplo 1

Na Figura 4.16 a mensagem é enviada do “Nó 1”, que gera e recebe os pacotes, para o “Nó 2”, que envia as mensagens de volta ao primeiro nó. A variável *L1* representa a latência da mensagem de número um, representada por *Msg 1*. Observa-se pela Figura 4.16, que o tempo de latência é calculado tomando como base o

momento em que a camada de aplicação envia a mensagem, e o momento em que essa mesma camada a recebe, ou seja, o temporizador só é interrompido quando a camada de aplicação recebe a mensagem.

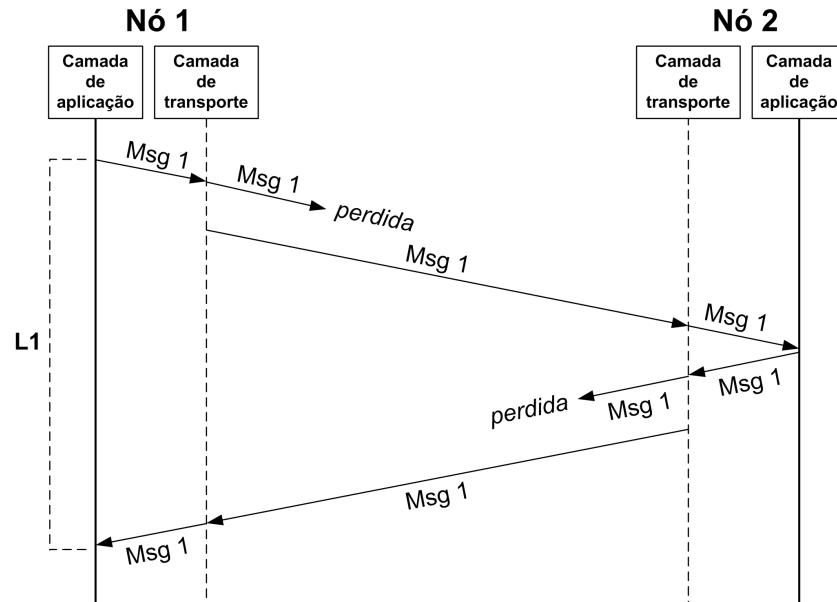


Figura 4.17: O cálculo da latência de uma mensagem - Exemplo 2

A Figura 4.17 mostra o comportamento de uma mensagem perdida no caminho de ida, do “Nó 1” ao “Nó 2”, e no caminho de volta, do “Nó 2” ao “Nó 1”. A latência final, representada pela variável $L1$, é computada a partir do momento em que a camada de aplicação envia os dados para a camada de transporte, e não recomeça a contagem caso haja uma retransmissão.

A Figura 4.18 mostra o envio de duas mensagens com o uso do TCP, em que a primeira é perdida no caminho de ida. Isso faz com que a mensagem de número dois sofra o HOL e fique armazenada na camada de transporte até que a primeira mensagem seja retransmitida e chegue ao “Nó 2”. Assim que isso acontece, as duas mensagens são enviadas de volta ao “Nó 1”. As variáveis $L1$ e $L2$ representam, respectivamente, a latência final da mensagem de número um e da mensagem de número dois.

Através deste ambiente, pretende-se simular uma rede *peer-to-peer* nó-a-nó, comparando os protocolos TCP e SCTP como protocolos da camada de transporte para aplicativos *peer-to-peer*. Esta comparação é atingida por meio da reprodução das características de transferência de mensagens em uma rede *peer-to-peer*, no ambi-

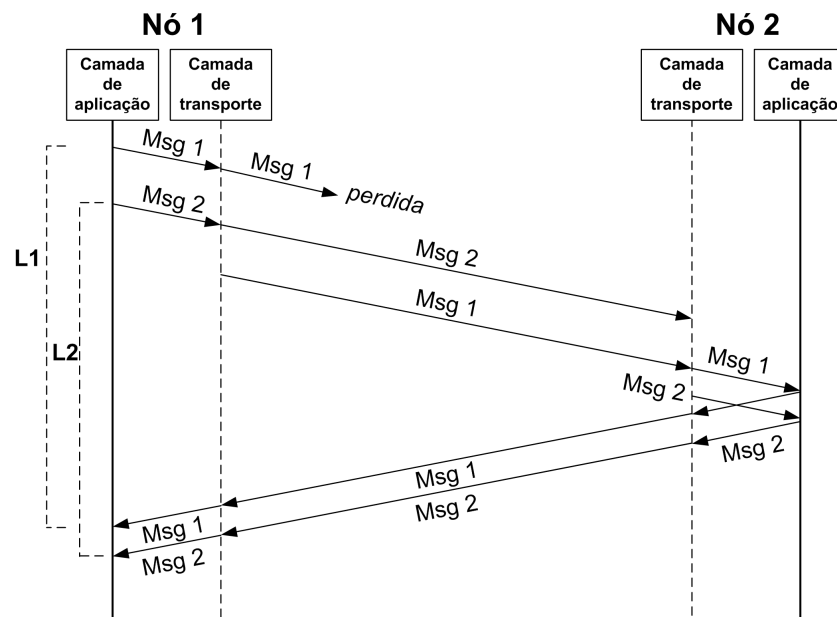


Figura 4.18: O cálculo da latência de uma mensagem - Exemplo 3

ente em que o comportamento e a dinâmica das mensagens são obtidas através dos parâmetros passados pelo usuário e, também, pela configuração do comportamento que a *Internet* provoca nas mensagens transmitidas. A métrica de desempenho é a média das latências de um dado número de mensagens.

4.3.2 A simulação da transferência de mensagens numa rede *peer-to-peer*

Para efetuar os experimentos que simulam a transferência de mensagens numa rede *peer-to-peer*, vários parâmetros devem ser configurados nos aplicativos que compõem o ambiente de simulação, não só naqueles que implementam o “Nó 1” e “Nó 2”, mas também naquele que atua como o gerador de atraso e perda de pacotes. Os parâmetros configurados neste último aplicativo são:

- RTT: Os valores de RTT (*Round Trip Time*) entre os nós de uma rede *peer-to-peer* são variados. Uma vez que eles podem se situar em diferentes localidades geográficas, um valor médio de escala mundial para este parâmetro é difícil de ser estimado. Para efetuar os experimentos aqui descritos, é utilizado o RTT médio registrado entre dois computadores, um localizado no Brasil e outro nos Estados Unidos, no mês de janeiro de 2006, que é de aproximadamente 200 milisegundos [4];

- Taxa de perda: Os valores de taxas de perda entre dois diferentes nós em uma rede *peer-to-peer* são variáveis, pois os pacotes que trafegam de um nó a outro podem atravessar caminhos que apresentam baixas ou altas taxas. Assim sendo, os experimentos mostram resultados para taxas de perda de 0% a 20%, permitindo, assim, uma análise geral do comportamento da transmissão de mensagens sob diferentes condições, referentes à perda de pacotes.

Os parâmetros configurados nos aplicativos que implementam o “Nó 1” e “Nó 2” são:

- Intervalo de tempo entre mensagens: Os dados descritos na Seção 4.1.1 são utilizados como referência para a configuração deste parâmetro. O valor médio de transmissão de QUERYs em uma rede *Gnutella* é de, aproximadamente, 40 mensagens por segundo e de QUERYHITs é de, aproximadamente, 25 mensagens por segundo;
- Tamanho das mensagens: Os dados descritos na Seção 4.1.2 são utilizados como referência para a configuração deste parâmetro. O tamanho médio das mensagens QUERY é de 50 *bytes* e das mensagens QUERYHIT é de 80 *bytes*;
- Número de mensagens: Em cada experimento são transmitidas 100 mensagens. Desta forma, é possível obter a latência média da transmissão de mensagens, calculada pela média da soma das latências de cada uma delas.

Os experimentos são executados em duas etapas; a primeira é executada fazendo uso dos parâmetros mostrados na Tabela 4.5.

Tabela 4.5: Parâmetros dos experimentos da primeira etapa

Parâmetro	Valor
Quantidade de mensagens	100
Tamanho das mensagens	80 <i>bytes</i>
Quantidade de mensagens por segundo	25

Na primeira etapa, cada experimento consiste no envio de 100 mensagens de 80 *bytes*, com um intervalo de 40 milissegundos entre cada uma delas, enviando, desta forma, 25 mensagens por segundo. Cada mensagem é enviada ao *echo server*, que responde imediatamente. São armazenadas as latências de cada uma das 100

mensagens, e o resultado final é a média aritmética das latências de todas elas, que representa o tempo médio que uma mensagem leva para ir e voltar do *echo server*.

São executados dez experimentos para cada valor de taxa de perda configurada, que variam de 0% a 20%. Foram escolhidos dez experimentos pois, desta forma, consegue-se mostrar a dinâmica da variação de desempenho entre as amostras sob uma mesma taxa de perda.

Em cada um dos experimentos é calculado o tempo médio de envio de mensagens com a utilização do protocolo TCP, com a utilização do protocolo SCTP enviando mensagens de forma desordenada, e com a utilização dos multi-fluxos do SCTP, sendo que três fluxos são utilizados. A utilização de três fluxos é meramente ilustrativa, para mostrar que o HOL pode ocorrer dentro de um mesmo fluxo.

A segunda etapa é executada com os parâmetros apresentados na Tabela 4.6.

Tabela 4.6: Parâmetros dos experimentos da segunda etapa

Parâmetro	Valor
Quantidade de mensagens	100
Tamanho das mensagens	50 <i>bytes</i>
Quantidade de mensagens por segundo	40

Na segunda etapa, cada experimento consiste no envio de 100 mensagens de 50 *bytes*, com um intervalo de 25 milissegundos entre cada uma delas, ou seja, são enviadas 40 mensagens por segundo. Da mesma forma que na primeira etapa, as taxas de perda variam de 0% a 20%, e para cada experimento são utilizados o TCP, o SCTP com envio de mensagens de forma desordenada, e o SCTP com a utilização de três fluxos.

4.3.3 O desempenho dos protocolos TCP e SCTP

Esta seção mostra os resultados comparativos entre o SCTP e o TCP. Os dez experimentos que são executados para cada taxa de perda configurada são indicados, nas figuras, pelas letras de *A* até *J*. As legendas das barras significam, no caso de “TCP”, a utilização do TCP na camada de transporte, no caso de “SCTP-D”, a utilização do SCTP com entrega de mensagens de forma desordenada e, no caso de “SCTP-F3”, a utilização dos multi-fluxos do SCTP com três fluxos. São mostrados os resultados para algumas taxas de perda e, também, um gráfico geral que consta

a média dos resultados de todos os experimentos, para a duas etapas.

As Figuras 4.19, 4.20 e 4.21 mostram os resultados de 30 experimentos da primeira etapa, sendo dez experimentos com a taxa de perda configurada em 0%, dez com a taxa de perda configurada em 10% e dez com a taxa de perda configurada em 20%.

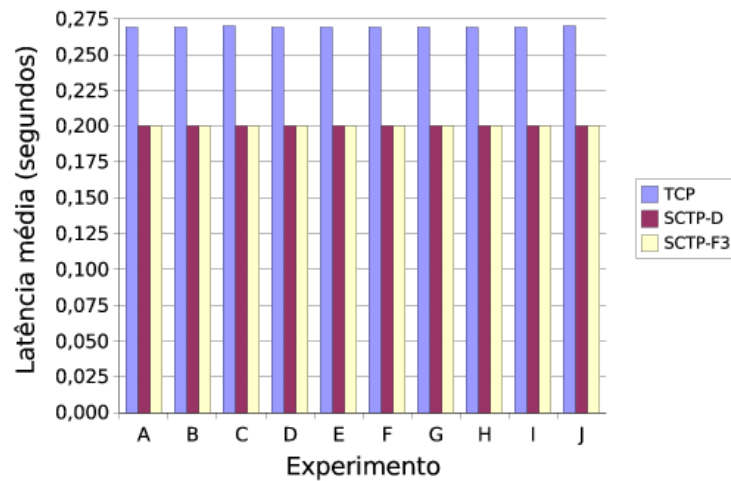


Figura 4.19: Resultados da primeira etapa com taxa de perda de 0%

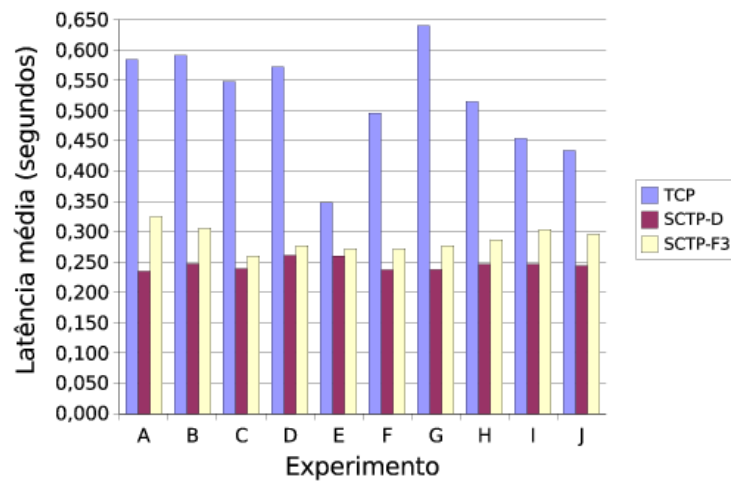


Figura 4.20: Resultados da primeira etapa com taxa de perda de 10%

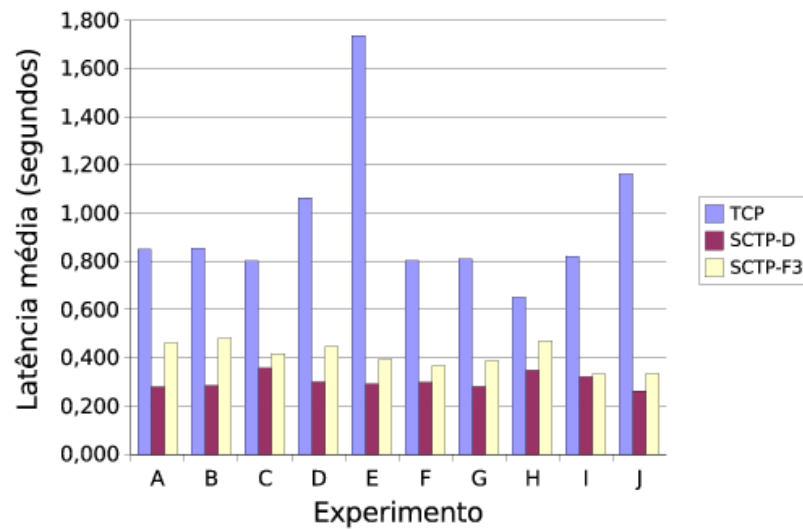


Figura 4.21: Resultados da primeira etapa com taxa de perda de 20%

Observa-se, pelos resultados dos experimentos, que o desempenho na transmissão de mensagens com a utilização do TCP é degradada, ou seja, o tempo médio da transmissão de mensagens nó-a-nó aumenta, quando se aumenta a taxa de perda. Observa-se, ainda, que o desempenho do SCTP com a utilização dos três fluxos é um pouco inferior ao SCTP com a utilização da entrega desordenada de mensagens pois, mesmo aliviando o HOL em relação ao TCP, ainda há casos onde ocorre o HOL intra-fluxo. Quando a taxa de perda é 0% o SCTP não demonstra ter ganhos significativos em relação ao TCP, no entanto, a diferença na latência média de aproximadamente 0,7 segundos deve-se às diferenças de implementação dos protocolos, assim como às funções adicionais da camada de aplicação que o TCP utiliza para atingir a confiabilidade completa.

Os tempos médios da transmissão de 100 mensagens em cada um dos experimentos, de acordo com o comportamento configurado nesta primeira etapa, mostram que o SCTP se adapta melhor ao comportamento de uma rede *peer-to-peer* que o TCP pois, alivia o HOL com a utilização dos multi-fluxos e evita completamente o HOL com a utilização da entrega desordenada de mensagens.

A Figura 4.22 mostra os resultados médios dos experimentos para taxas de perda de 0% a 20%. Cada ponto no gráfico representa a latência média do envio de 1000 mensagens.

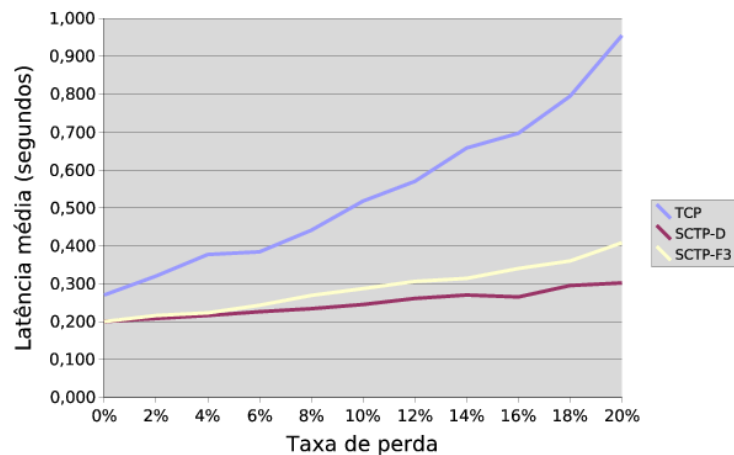


Figura 4.22: Média dos resultados da primeira etapa - variação da taxa de perda de 0% a 20%

Observa-se pelo gráfico geral que quanto maior a taxa de perda, mais degradado é o desempenho do TCP em relação ao SCTP. Isso ocorre por causa do problema do HOL, que faz com que as mensagens que chegam fora de ordem fiquem aguardando na camada de transporte e não sejam liberadas à camada de aplicação.

As Figuras 4.23, 4.24 e 4.25 mostram os resultados de 30 experimentos, desta vez com os parâmetros da segunda etapa, sendo dez experimentos com a taxa de perda configurada em 0%, dez com a taxa de perda configurada em 10% e dez com a taxa de perda configurada em 20%.

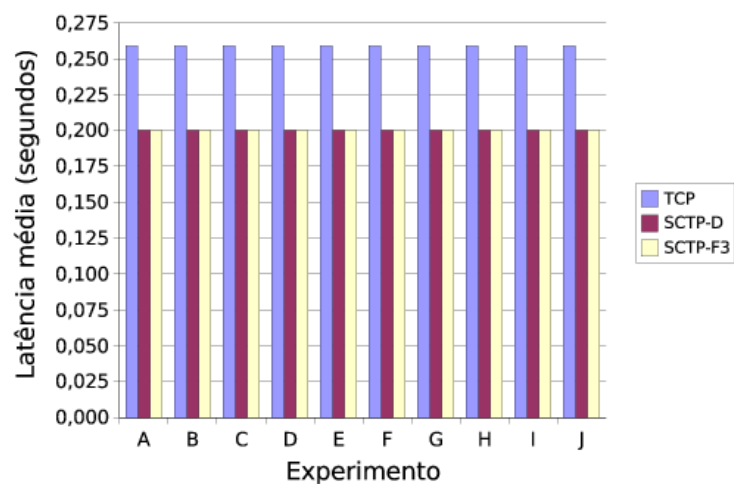


Figura 4.23: Resultados da segunda etapa com taxa de perda de 0%

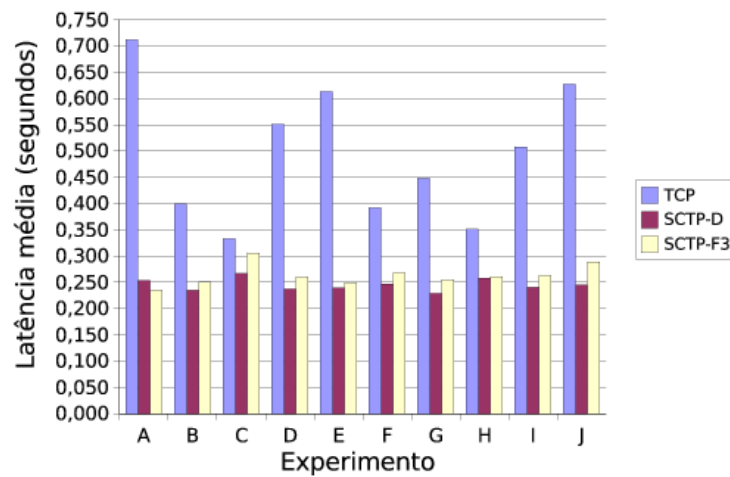


Figura 4.24: Resultados da segunda etapa com taxa de perda de 10%

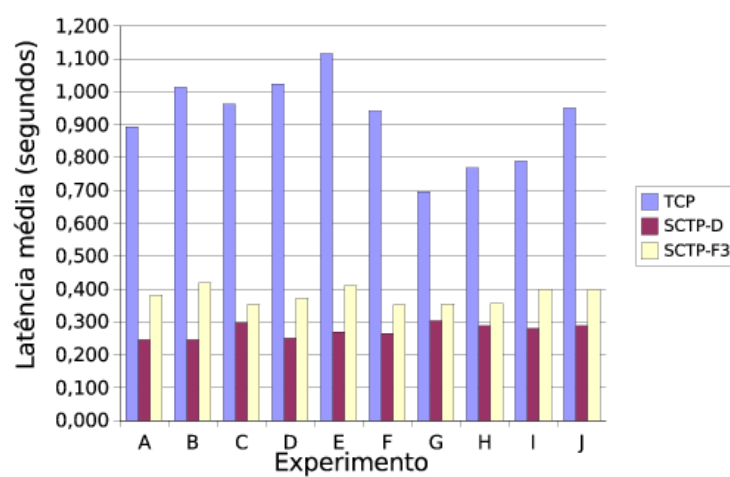


Figura 4.25: Resultados da segunda etapa com taxa de perda de 20%

Os resultados dos experimentos feitos nesta etapa não demonstraram ser significativamente diferentes daqueles executados na primeira etapa. Mesmo com o envio de maior quantidade de mensagens por segundo e, ainda, de mensagens de menor comprimento, o comportamento causado pelo HOL foi similar. A Figura 4.26 mostra os resultados médios de todos os experimentos da segunda etapa.

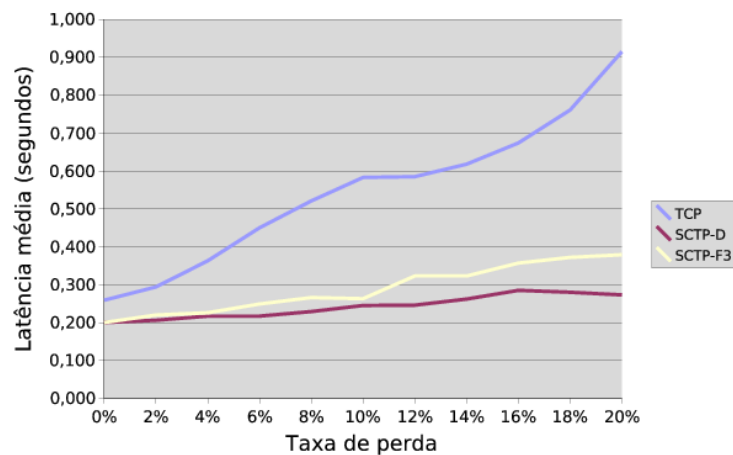


Figura 4.26: Média de resultados da segunda etapa - variação da taxa de perda de 0% a 20%

Observa-se na Figura 4.26, como na primeira etapa, que altas taxas de perda de pacotes tendem a degradar o desempenho da transmissão de mensagens nó-a-nó quando o protocolo TCP é utilizado na camada de transporte. Assim sendo, a utilização do SCTP pode ser mais adequada para esta tarefa. Ainda, como discutido na Seção 4.1.3, caso seja melhorado o tempo de transmissão de mensagens nó-a-nó, o tempo médio de resposta às consultas em uma rede *peer-to-peer* pode ser melhorado.

Foram mostradas duas utilizações para o SCTP, uma com os multi-fluxos, em que foram usados três fluxos, e outra com a entrega desordenada de mensagens. A forma de utilização destas funcionalidades em um aplicativo *peer-to-peer* depende da especificação do protocolo e dos serviços demandados pelo aplicativo.

O aplicativo *Octopus*, descrito na próxima seção, faz uso dessas funcionalidades, mais especificamente a entrega desordenada de mensagens para o transporte de mensagens *Gnutella*, e a entrega ordenada de mensagens para a transferência de arquivos entre dois nós, ambas através de uma única associação.

4.4 A aplicação *peer-to-peer* de compartilhamento de arquivos *Octopus*

Esta seção descreve o desenvolvimento do aplicativo *Octopus*. Este aplicativo permite que os usuários consultem os nós em uma rede *overlay* em busca de arquivos. Uma vez encontrado um arquivo, o usuário, de posse de sua localização, pode efetuar a sua transferência. O *Octopus* utiliza os conceitos do SCTP e do *Gnutella*, versão 0.4, descritos nos capítulos anteriores, em uma rede *peer-to-peer* cuja camada de transporte é implementada por meio do SCTP.

Esta seção tem como objetivo descrever um protótipo de aplicativo que mostra como algumas das funcionalidades do SCTP podem ser aproveitadas por um aplicativo *peer-to-peer* de compartilhamento de arquivos, sem a intenção de abranger completamente todas as formas de uso do SCTP para este, ou outros gêneros e arquiteturas de sistemas *peer-to-peer*.

O *Octopus* foi desenvolvido fazendo uso da linguagem C com as bibliotecas *pthread lib*, *libsctp* e a *GTK lib*, versão 2.0, para a *interface* gráfica, e seu código fonte pode ser encontrado na íntegra no Apêndice B.

4.4.1 O protocolo *Gnutella* com o uso do SCTP

O protocolo da camada de aplicação utilizado pelo aplicativo *Octopus* é o *Gnutella*, versão 0.4. No entanto, algumas modificações no formato do cabeçalho das mensagens *Gnutella* podem ser feitas de modo a melhor utilizar as funcionalidades oferecidas pelo protocolo da camada de transporte SCTP.

Dois campos são removidos do cabeçalho das mensagens do protocolo *Gnutella*: o campo de tamanho do *payload* e o de descritor de *payload*. A Figura 4.27 mostra a reestruturação do cabeçalho.

Campos	ID do descritor	Descritor de <i>payload</i>	TTL	Hops	Tamanho do <i>payload</i>
Byte Offset	0...15	16	17	18	19..22

(a)

Campos	ID do descritor	TTL	Hops
Byte Offset	0...15	16	17

(b)

Figura 4.27: Novo cabeçalho para o protocolo *Gnutella*

A Figura 4.27 (a) mostra o cabeçalho antigo, e a Figura 4.27 (b) mostra o

cabeçalho reestruturado:

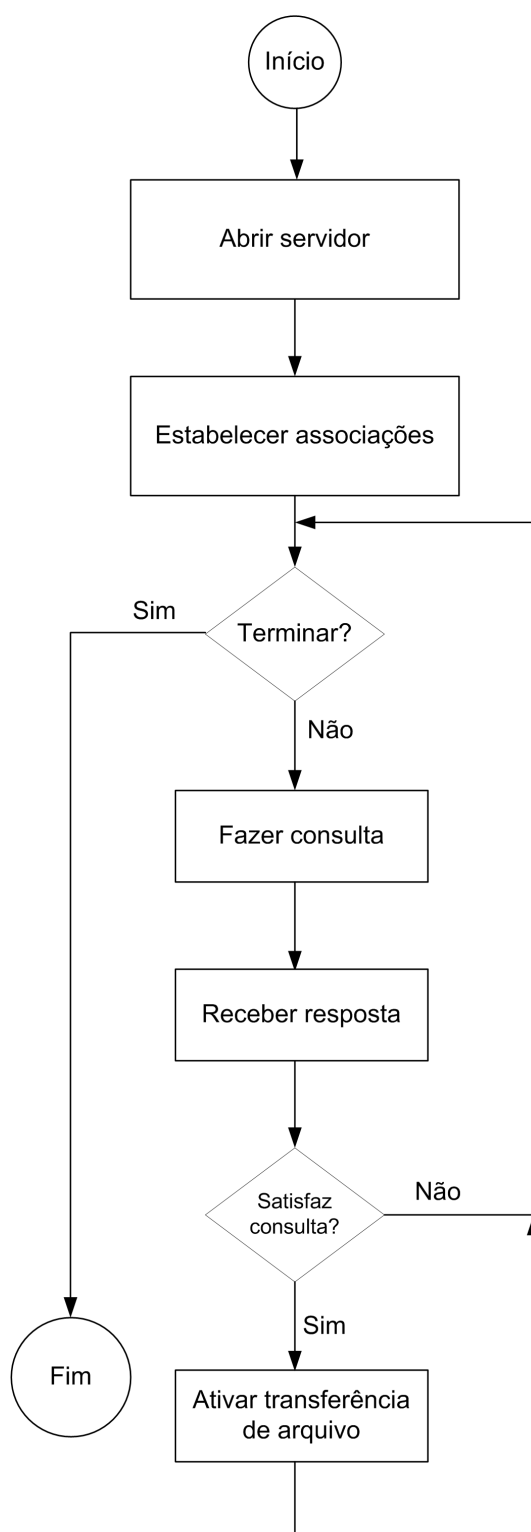
- O campo que indica o tamanho do *payload* é removido pois o SCTP é orientado a mensagens, não havendo a necessidade de se delimitar o fluxo de *bytes* com campos adicionais;
- O campo de descritor de *payload* é removido pois, para detectar qual tipo de mensagem está sendo entregue à camada de aplicação, utiliza-se o identificador do fluxo que a mensagem foi enviada.

4.4.2 O funcionamento do aplicativo *Octopus*

O aplicativo *Octopus* foi desenvolvido levando em consideração as três fases, descritas na introdução deste capítulo, referentes à conexão, à consulta e à transferência de arquivos em redes *peer-to-peer*. Como mostra o fluxograma da Figura 4.28, os estágios típicos³ de funcionamento do aplicativo *Octopus* são:

- Abrir servidor: Neste estágio, o aplicativo cria uma *thread* que inicia o servidor de associações. Por meio deste servidor, o *Octopus* pode aceitar pedidos de estabelecimento de associação de outros nós;
- Estabelecer associações: Neste estágio, o aplicativo solicita o estabelecimento de associação a nós cujo endereço IP o usuário forneceu, ou que o aplicativo buscou em uma base de endereços própria;
- Fazer consulta: O usuário pode efetuar uma consulta que vai ser propagada através da rede caso o aplicativo tenha estabelecido, ao menos, uma associação;
- Receber resposta: Caso o aplicativo receba uma mensagem de resposta a uma consulta feita pelo usuário, os dados desta resposta são mostrados na tela;
- Ativar transferência de arquivos: Se o usuário deseja obter um dos arquivos cujos dados foram apresentados na tela, este estágio é, então, ativado.

³Usa-se o termo “típico”, pois esses estágios podem ser intercalados e sobrepostos, diferindo da maneira pela qual é apresentada no fluxograma. Por exemplo, ao mesmo tempo que uma consulta está em andamento, o usuário pode decidir se associar a um outro nó e, ainda, pode já estar efetuando a transferência de um arquivo.

Figura 4.28: O fluxograma de funcionamento do *Octopus*

O modo com que as mensagens *Gnutella* e os arquivos são transmitidos no *Octopus* é diferente da forma pela qual essas tarefas são normalmente executadas por um aplicativo que faz uso do TCP na camada de transporte. O SCTP pode utilizar o mesmo canal de relacionamento para executar ambas as tarefas simultaneamente. As Figuras 4.29 e 4.30 mostram esses dois casos.

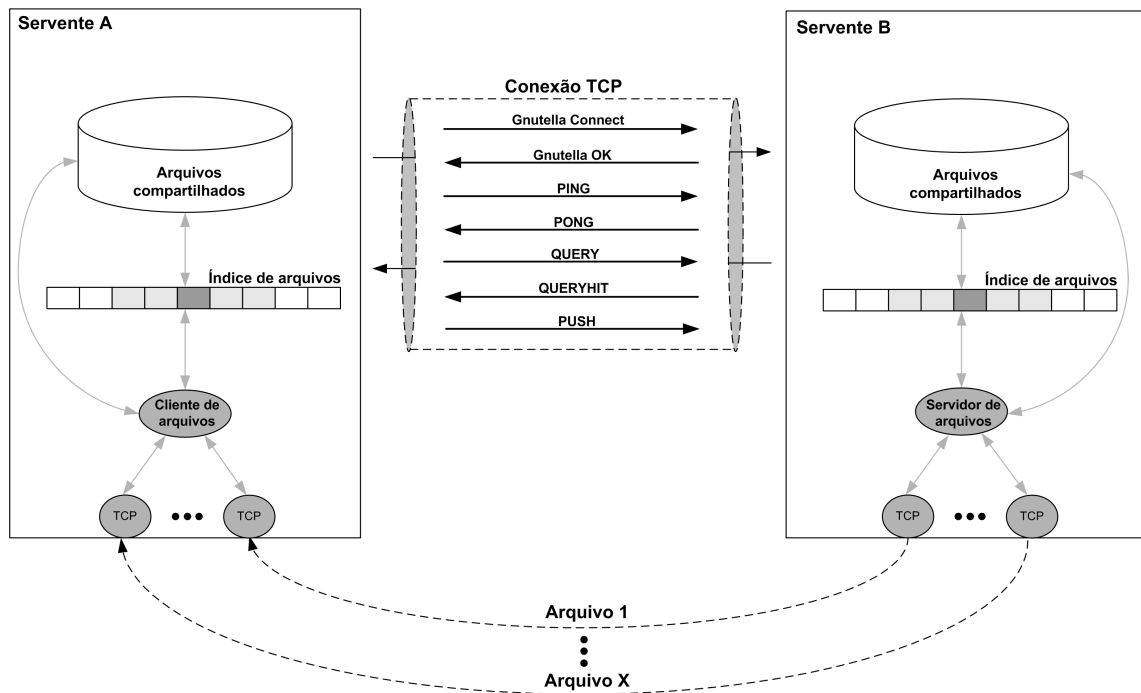


Figura 4.29: Um aplicativo *peer-to-peer* utilizando o TCP na camada de transporte

Um aplicativo que utiliza o TCP na camada de transporte normalmente utiliza uma conexão dedicada para a transmissão das mensagens *Gnutella* e, para cada arquivo que deseja transmitir, uma nova conexão é estabelecida, como mostrado na Figura 4.29.

Por sua vez, o aplicativo *Octopus* faz uso de uma única associação SCTP para a transferência das mensagens de negociação da “conexão” *Gnutella*, das mensagens PING, PONG, QUERY e QUERYHIT⁴, e que pode, ainda, ser utilizada para a negociação e transferência de mais de um arquivo simultaneamente⁵, como mostra a Figura 4.30.

⁴Nesta versão do *Octopus* não foi implementada a mensagem PUSH.

⁵Nesta versão do *Octopus* o número máximo de arquivos a ser transferidos simultaneamente de um mesmo nó é de três arquivos.

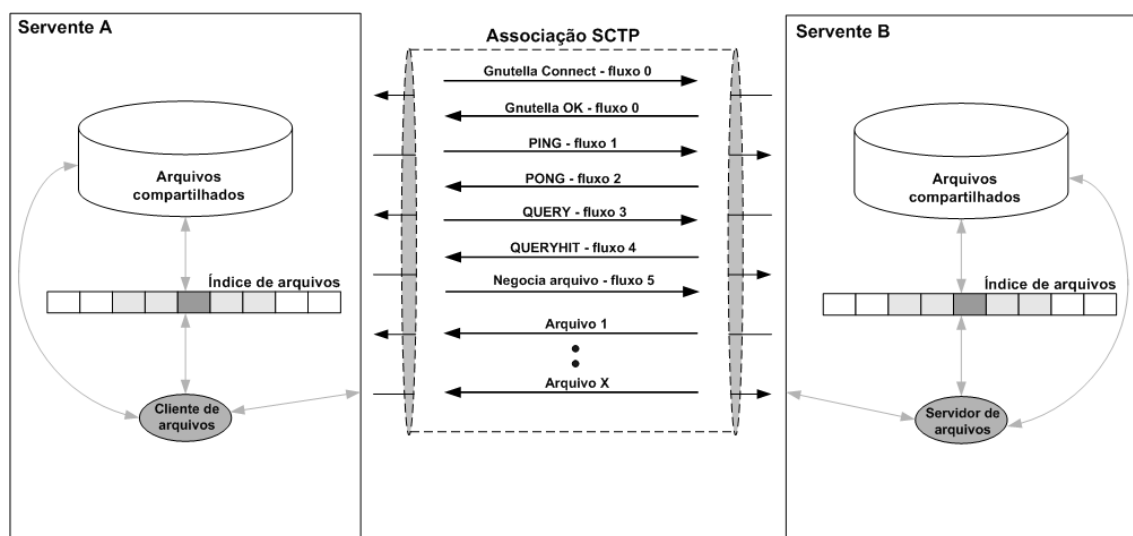


Figura 4.30: O aplicativo *Octopus* utilizando o Sctp na camada de transporte

Há dois cenários a serem considerados a respeito da utilização da associação Sctp para a transferência de arquivos: aquele em que o arquivo a ser transferido está em um nó que já é vizinho do nó requerente, ou seja, está localizado a um *hop* de distância, e aquele em que o arquivo a ser transferido está localizado em um nó que não é vizinho do nó requerente, ou seja, está localizado a mais de um *hop* de distância.

No caso de o arquivo estar localizado em um nó vizinho, a associação já estabelecida é utilizada para a negociação e transferência deste arquivo, sem a necessidade de se estabelecer outra associação entre esses nós. Há a possibilidade, ainda, de se transferir mais de um arquivo simultaneamente, junto com as mensagens *Gnutella*, através de uma única associação.

Por outro lado, caso o arquivo esteja localizado em um nó que está a mais de um *hop* de distância do nó requerente, uma associação deve ser estabelecida entre eles. A nova associação pode ser utilizada para transferir mais de um arquivo simultaneamente e, também, para transmitir mensagens *Gnutella*.

4.4.3 A arquitetura do aplicativo *Octopus*

O aplicativo *Octopus* é composto basicamente de quatro camadas: a camada de comunicação, a camada de tratamento de mensagens, a camada de armazenamento e a camada de *interface*. A camada de *interface* é aquela que implementa a *interface*

gráfica que o usuário pode utilizar para executar as funções do aplicativo.

Os módulos que compõem a camada de armazenamento auxiliam o funcionamento do aplicativo por meio da implementação do *cache* de associações, do *host cache*, do *cache* de identificadores de mensagens e do repositório de arquivos compartilhados.

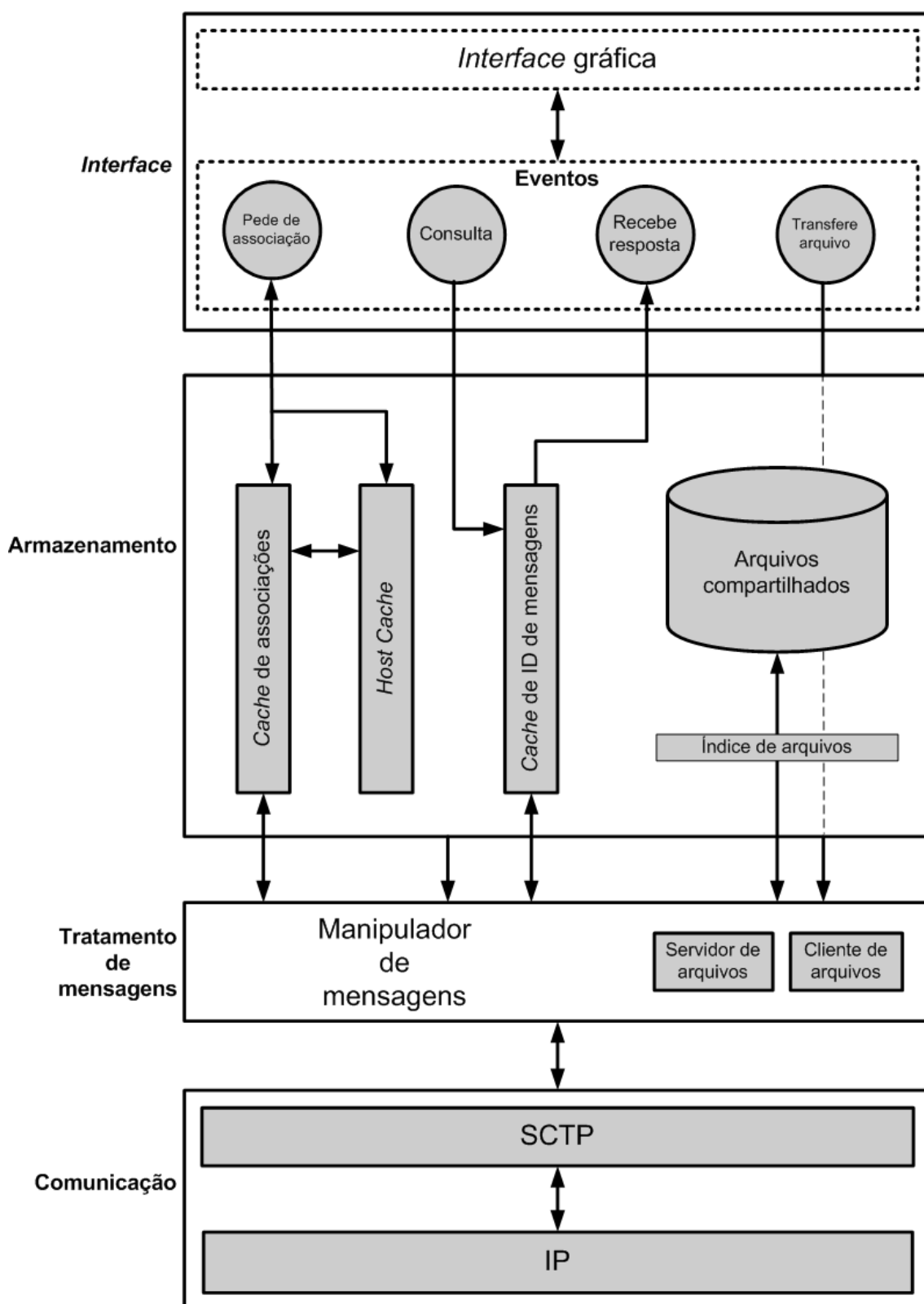
A camada de tratamento de mensagens implementa o manipulador de mensagens, que é a entidade que faz o reconhecimento das mensagens *Gnutella* de acordo com o fluxo através do qual foram enviadas, além de providenciar os seus tratamentos. Ainda, esta entidade trata a negociação e a transferência de arquivos, tanto como cliente, quanto como servidor, ou seja, o próprio manipulador de mensagens faz a negociação e o envio de arquivos que são requisitados por outros nós, ou que são transferidos de outros nós para o nó local.

A camada de comunicação é aquela que implementa os protocolos SCTP e IP, responsáveis, respectivamente pelas camadas de transporte e camada de rede. A Figura 4.31 mostra as camadas do aplicativo e os relacionamentos dos diferentes módulos, destacando, na camada de *interface*, os eventos que podem ocorrer durante o tempo de execução.

Na camada de armazenamento, o módulo de *cache* de associações armazena informações sobre as associações que o nó local estabeleceu com outros nós, tanto aquelas que o próprio usuário solicitou, quanto aquelas que outros nós estabeleceram com o nó local. Este módulo é sempre atualizado quando uma nova associação é estabelecida, ou quando há um encerramento em alguma associação do *cache*. A estrutura que armazena as informações do *cache* de associações é implementada através de uma estrutura cujo código é mostrado a seguir:

```
struct cacheAssociacoes{
    int numeroSocket;
    struct sockaddr_in endereco;
};
```

São armazenados o número do descritor do *socket* para o acesso a um determinado nó, assim como as informações do endereço, em uma estrutura do tipo *sockaddr_in*. É possível, por meio de uma chamada da *interface* gráfica, imprimir os dados contidos

Figura 4.31: A arquitetura do aplicativo *Octopus*

no *cache* de associações no *shell* no *Linux*, como mostrado na Figura 4.32.

```
CACHE DE ASSOCIACOES

Associacao numero:1
Descritor de socket de numero:6
Endereco:200.131.189.69
Porta:1000

Associacao numero:2
Descritor de socket de numero:8
Endereco:192.168.21.2
Porta:1000
```

Figura 4.32: A impressão do *cache* de associações no *shell* do *Linux*

O módulo de *host cache* armazena endereços de nós em que o usuário obteve sucesso no estabelecimento da associação, e também endereços que foram recebidos por meio das mensagens PONG, em resposta ao PING requerido pelo usuário.

O *host cache* é utilizado para facilitar o estabelecimento de associações, pois armazena endereços de possíveis nós participantes da rede *peer-to-peer* em um arquivo, que pode ser utilizado em outras sessões do aplicativo. É possível, por meio de uma chamada da *interface* gráfica, imprimir os dados contidos no *host cache* no *shell* no *Linux*, como mostrado na Figura 4.33.

```
Imprimindo HOST CACHE

Endereco:200.131.189.69
Porta:1000

Endereco:192.168.21.2
Porta:1000

Endereco:200.131.189.68
Porta:1010

Endereco:200.131.189.79
Porta:1013
```

Figura 4.33: A impressão do *host cache* no *shell* do *Linux*

O *cache* de identificadores de mensagens é utilizado para três propósitos: a detecção de mensagens que sofreram *loops* dentro da rede, a execução do *backtrack* de mensagens de resposta e a detecção de mensagens de resposta que pertencem

ao nó local, providenciando, assim, a exibição das informações na *interface* gráfica. A seguinte estrutura é utilizada para implementar o *cache* de identificadores de mensagens:

```
struct cacheIDMensagens
{
    int ID;
    int NumeroSocket;
    struct sockaddr_in endereco;
};
```

O repositório de arquivos compartilhados contém os arquivos que o usuário deseja disponibilizar para os nós da rede *overlay*. Para adicionar arquivos neste repositório, o usuário deve copiá-los no diretório “ArquivosCompartilhados”, tendo como referência o diretório que contém o executável do aplicativo *Octopus*. O “índice de arquivos” não apresenta nenhuma estrutura específica de armazenamento, representando somente uma chamada da aplicação que faz referência aos nomes dos arquivos.

O manipulador de mensagens é a entidade que faz o tratamento das mensagens *Gnutella*, e que implementa o cliente e o servidor de arquivos. Para detectar qual tipo de mensagem o nó recebeu e encaminhá-la para a função que faz o seu tratamento correto, assim como para efetuar a transferência de arquivos simultaneamente através de uma única associação, este módulo faz uso da funcionalidade dos multi-fluxos do SCTP, como mostrado na Figura 4.34.

Os fluxos numerados de um a quatro são utilizados para o envio das mensagens *Gnutella*⁶. O fluxo de número cinco é utilizado para a negociação da transferência de um arquivo. Todos estes fluxos utilizam o envio de mensagens de forma desordenada, pois não importa a sequência em que são entregues à camada de aplicação. As mensagens que compõem um arquivo são entregues a partir dos fluxos seis, sete ou oito.

Para cada mensagem que chega no servente, o conteúdo útil é armazenado em um

⁶O fluxo de número zero é utilizado para as mensagens de *handshake* do *Gnutella*, mas não foi ilustrado nesta figura.

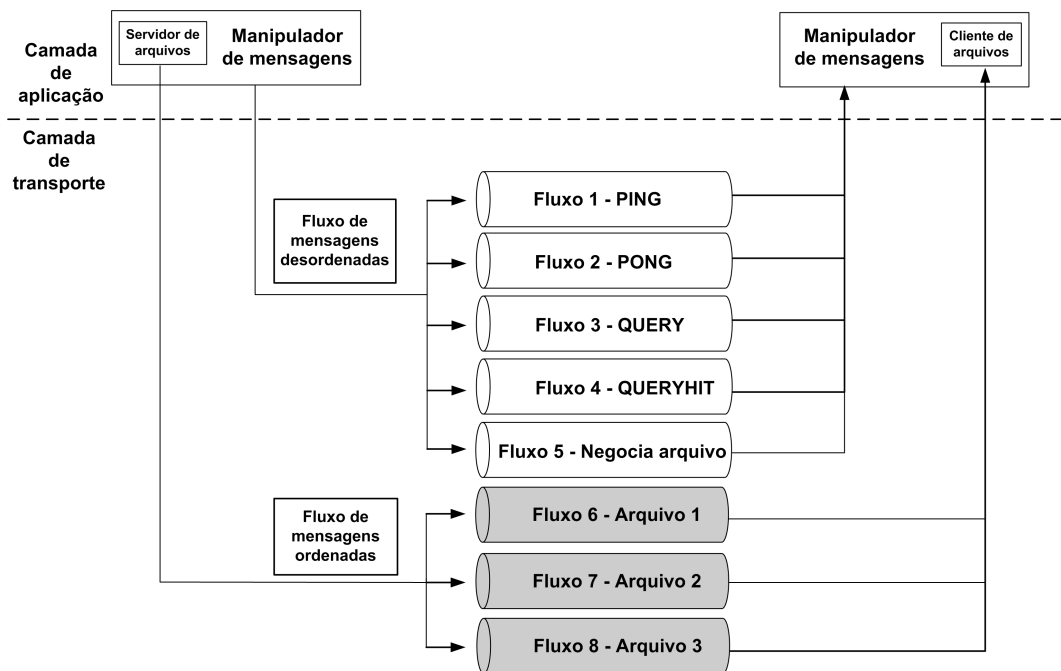


Figura 4.34: A utilização dos multi-fluxos pelo aplicativo *Octopus*

buffer. Em seguida, identifica-se através de qual fluxo a mensagem chegou e, então, o conteúdo do *buffer* é passado para a função que trata a mensagem apropriadamente. As mensagens *Gnutella* são identificadas da seguinte forma:

- PINGs: Devem ser enviadas através do fluxo 1. Ao receber uma mensagem, se o manipulador de mensagens detecta que chegou através do fluxo 1, ele a encaminha para a função que trata PINGs.
- PONGs: Devem ser enviadas através do fluxo 2. Ao receber uma mensagem, se o manipulador de mensagens detecta que chegou através do fluxo 2, ele a encaminha para a função que trata PONGs.
- QUERY: Devem ser enviadas através do fluxo 3. Ao receber uma mensagem, se o manipulador de mensagens detecta que chegou através do fluxo 3, ele a encaminha para a função que trata QUERYs.
- QUERYHIT: Devem ser enviadas através do fluxo 4. Ao receber uma mensagem, se o manipulador de mensagens detecta que chegou através do fluxo 4, ele a encaminha para a função que trata QUERYHITs.

Para identificar o número do fluxo através do qual a mensagem chegou, a API (*Application Programming Interface*) do SCTP preenche uma estrutura que contém as informações sobre o *chunk* que carregou essa mensagem, e confere o campo *sinfo_stream* da estrutura a seguir:

```
struct sctp_sndrcvinfo{
    uint16_t    sinfo_stream;
    uint16_t    sinfo_ssn;
    uint16_t    sinfo_flags;
    uint32_t    sinfo_ppid;
    uint32_t    sinfo_context;
    uint8_t     sinfo_dscp;
    sctp_assoc_t sinfo_assoc_id;
}
```

A seguinte função é utilizada para receber as mensagens através do *socket*:

```
struct sctp_sndrcvinfo sri;

quantidadeRecebida=sctp_rcvmsg(sock_fd,readbuf,200,
                               struct sockaddr *) &cliaddr,
                               &len,&sri,&msg_flags);
```

O campo *sinfo_stream* da estrutura *sctp_sndrcvinfo*, declarada no programa fonte como a variável *sri*, é utilizado para identificar em qual fluxo a mensagem chegou, e encaminhá-la para a função que faz o seu tratamento.

Os eventos que podem ocorrer neste aplicativo, os quais interagem com a *interface gráfica*, são: o usuário fazer o pedido de estabelecimento de associação, o usuário efetuar uma consulta, o usuário requerer a transferência de um arquivo, e o aplicativo receber uma mensagem de resposta a uma consulta do usuário. A *interface gráfica* do *Octopus* é mostrada na Figura 4.35:

- Associar: Ao pressionar este botão, o aplicativo tenta se associar ao nó cuja localização é dada pelas informações fornecidas nos campos “Endereço” e “Porta”. Caso haja sucesso na associação, o endereço em questão é adicionado ao *host cache*;

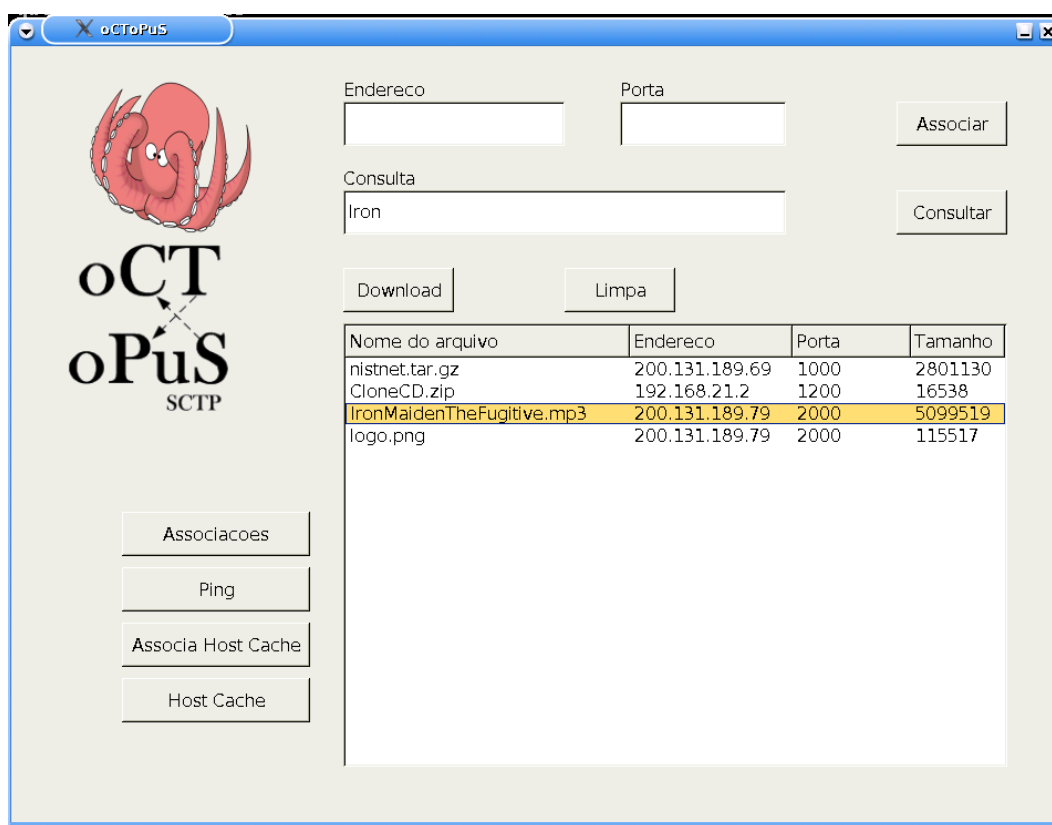


Figura 4.35: A interface gráfica do aplicativo *Octopus*

- **Consultar:** Caso uma associação tenha sido efetivada, o usuário pode consultar os nós da rede por arquivos, preenchendo o campo do texto e, em seguida, pressionando o botão “Consulta”. O identificador da mensagem QUERY gerada é armazenado no *cache* de identificadores de mensagem. Desta forma, se uma mensagem QUERYHIT for detectada, é possível identificar se a resposta pertence ao nó local;
- **Download:** Caso alguma resposta tenha chegado, as informações a respeito dos arquivos são adicionadas à lista de respostas. O usuário que decide fazer a transferência de algum arquivo deve indicar na lista a opção desejada e pressionar “Download”;
- **Associações:** Imprime no *shell* do *Linux* as informações sobre as associações que estão ativas;
- **Ping:** Envia uma mensagem PING a todos os nós diretamente associados,

como forma de atualizar o *host cache* local. Os nós respondem ao PING com um PONG para cada nó que tenham em seu *host cache*;

- Associa *Host Cache*: O nó local tenta associar-se a todos os nós cujas informações estão contidas no *host cache*;
- *Host Cache*: Imprime no *shell* do *Linux* as informações sobre todas os nós que estão registrados no *host cache*.

Há eventos que podem ocorrer em tempo de execução, que não têm interação com a *interface* gráfica:

- O recebimento de uma mensagem PONG: Mesmo sendo a mensagem PING gerada por uma chamada da *interface* gráfica, o recebimento da PONG não provoca nenhuma resposta visual ao usuário;
- O recebimento de uma mensagem QUERY: Quando uma mensagem QUERY é detectada, o nó local a processa, conferindo se possui um arquivo local que possa satisfazê-la, e a transfere a todos os nós diretamente conectados, exceto àquele que a enviou, caso o TTL não tenha expirado;
- O recebimento de uma mensagem QUERYHIT que carrega uma resposta que não pertence ao nó local: Neste caso, os dados do *cache* de identificadores de mensagens são utilizados para efetuar o processo do *backtrack*.

4.4.4 A negociação e a transferência de arquivos no *Octopus*

A associação estabelecida entre dois nós que fazem uso do aplicativo *Octopus* pode ser utilizada para o envio simultâneo arquivos. Para que esses nós possam negociar essa transferência, um protocolo simples, baseado no HTTP, foi especificado, e é mostrado nesta seção.

Quando o usuário solicita a transferência de um arquivo, uma mensagem de requisição é enviada ao nó que compartilha este arquivo. Esta mensagem é enviada através do fluxo cinco, de forma desordenada, e sua sintaxe é mostrada a seguir:

```
GET /get/<Nome do arquivo>/ OCTOPUS/1.0\r\n\r\n
```

Para implementar o envio simultâneo de arquivos através de uma única associação, faz-se uso da funcionalidade dos multi-fluxos. A versão do aplicativo desenvolvido limita-se ao envio de três arquivos, de maneira *full-duplex*, ou seja, três deles podem trafegar em um sentido, e três podem trafegar no sentido oposto, simultaneamente. Os fluxos de número seis, sete ou oito são utilizados para esta tarefa, e as mensagens enviadas através deles são entregues de forma ordenada.

Assim que um nó recebe o pedido de transferência de arquivo, ele seleciona um dos três fluxos que estiver disponível, de acordo com o algoritmo mostrado no fluxograma da Figura 4.36, e envia a seguinte mensagem, através do fluxo que selecionou:

```
OCTOPUS/1.0 001 OK\r\nArquivo:<Nome do Arquivo>\r\nTamanho:<Tamanho do arquivo>\r\n\r\n
```

O nó requerente, segundo as informações recebidas na mensagem de resposta do nó servidor, sabe o nome do arquivo que deve criar, assim como o fluxo através do qual vai receber as mensagens que compõem este arquivo. Um contador é configurado com o valor do tamanho do arquivo, obtido da mensagem de resposta, que é decrementado a cada porção de dados que é gravada. Quando este contador atinge o valor zero, o arquivo é fechado e o processo de transferência é encerrado.

A escolha dos fluxos é feita de maneira seqüencial, como mostrado no fluxograma da Figura 4.36. Primeiramente é avaliada a disponibilidade do fluxo de número seis. Caso ele esteja ocupado, o fluxo de número sete é conferido. Caso ambos estejam ocupados, o fluxo oito é utilizado.

Há, porém, a possibilidade de não haver mais fluxos disponíveis em um dado sentido da associação, pois três arquivos já estão sendo transmitidos simultaneamente. Neste caso, a seguinte mensagem é enviada ao nó requerente, através do fluxo cinco:

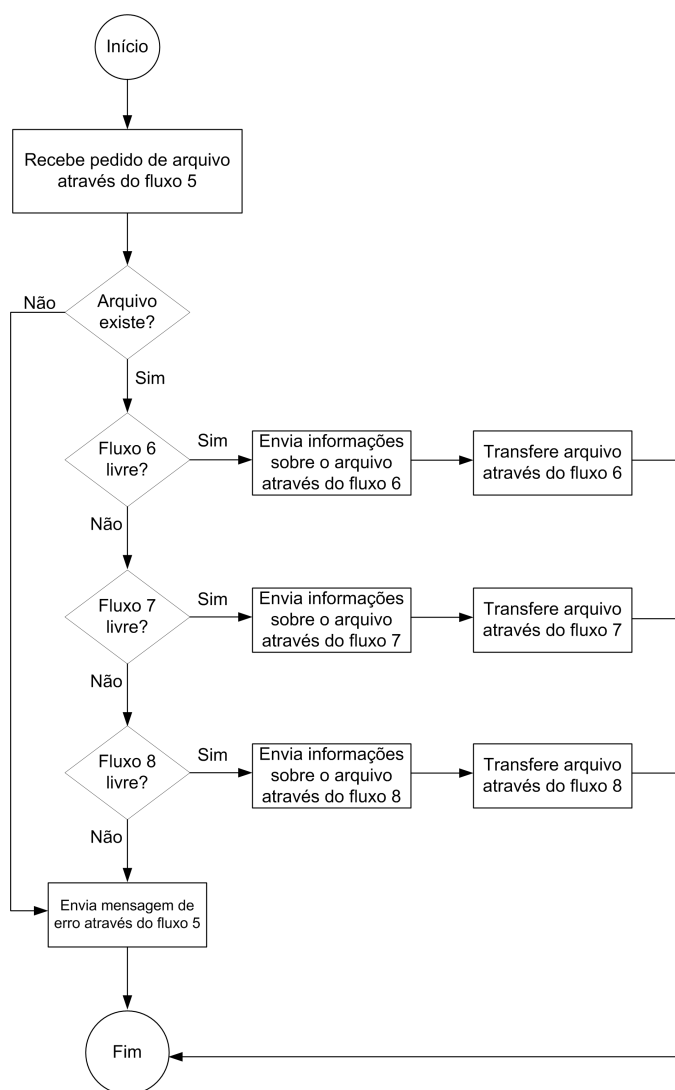


Figura 4.36: O fluxograma do servidor de arquivos do *Octopus*

OCTOPUS/1.0 002 ERRO\r\n

Arquivo:<Nome do Arquivo>\r\n

Motivo:Todos os fluxos ocupados\r\n

\r\n

Há, também, a possibilidade do arquivo não ser mais encontrado para o compartilhamento, e o servidor envia a seguinte mensagem, através do fluxo cinco:

OCTOPUS/1.0 003 ERRO\r\n

Arquivo:<Nome do Arquivo>\r\n

Motivo:Arquivo não encontrado\r\n

\r\n

O fluxograma da Figura 4.37 mostra os passos para a execução da transferência de arquivos, sob a perspectiva do cliente de arquivos, ou seja, aquele que requisita o arquivo.

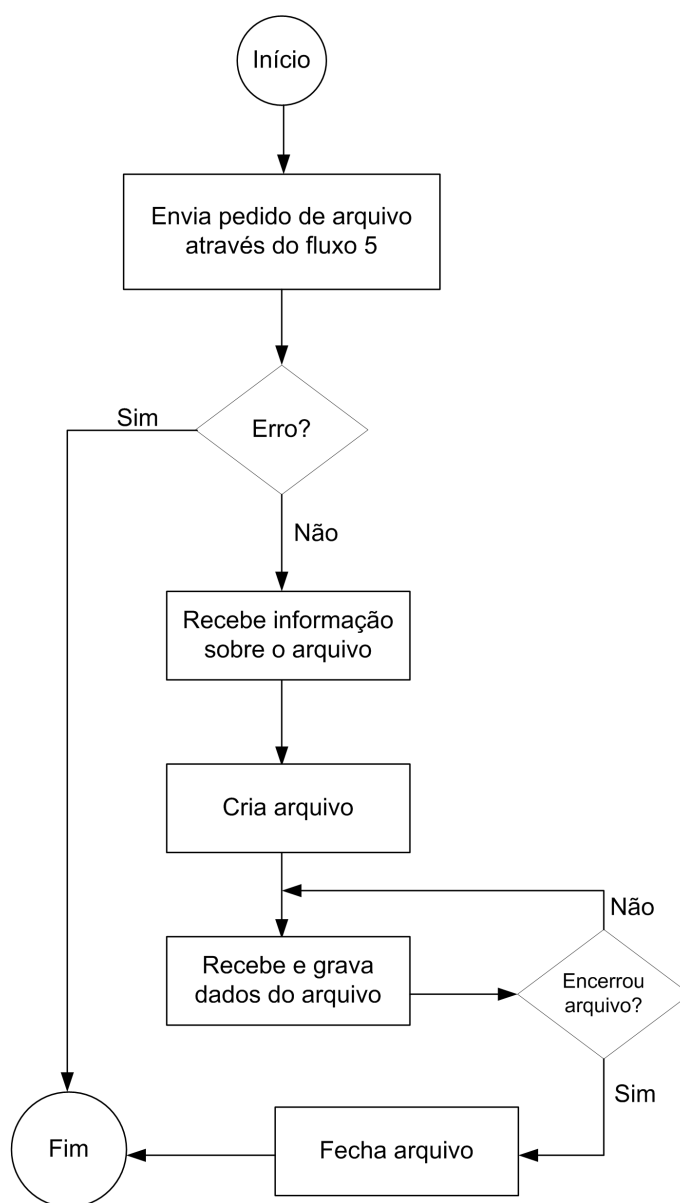


Figura 4.37: O fluxograma do cliente de arquivos do *Octopus*

Primeiramente o nó envia o pedido do arquivo. Caso não ocorra nenhum erro no servidor de arquivos, causados pela indisponibilidade dos fluxos, ou por algum erro na abertura do arquivo, o nó cliente recebe, através do fluxo selecionado pelo servidor, as informações sobre o arquivo que é enviado, podendo, desta forma, executar

a abertura deste arquivo e atualizar o contador que auxilia no seu fechamento.

Capítulo 5

Conclusão

Os aplicativos *peer-to-peer* têm sido cada vez mais utilizados na *Internet* para diversas finalidades. Aqueles aplicativos destinados ao compartilhamento de arquivos são os mais disseminados e empregados pelos usuários. Por este motivo, esse gênero de sistema foi o alvo de estudo e de desenvolvimento deste trabalho.

O protocolo de transporte é uma peça fundamental no desenvolvimento desses aplicativos e a sua escolha correta afeta diretamente o desempenho e as funcionalidades que esses sistemas oferecem. Foi mostrada, neste trabalho, a viabilidade da utilização do SCTP como protocolo da camada de transporte para os aplicativos *peer-to-peer* de compartilhamento de arquivos.

Primeiramente, foram apresentadas as definições e os gêneros de sistemas que podem ser desenvolvidos conforme o modelo *peer-to-peer*, assim como as características das redes que o implementam. Em seguida, foi apresentada uma classificação das diversas arquiteturas de redes *peer-to-peer* e exemplos de protocolos da camada de aplicação para aplicativos de compartilhamento de arquivos.

Uma das contribuições deste trabalho foi a proposta de uma nova classificação das diferentes arquiteturas de redes *peer-to-peer*. A partir do levantamento das características relacionadas à escalabilidade, aos papéis dos nós e aos algoritmos de roteamento, foi apresentada uma forma de classificar as arquiteturas de redes *peer-to-peer* dentro de um pólo mais centralizador a outro, descentralizador.

Em seguida, foram apresentados os conceitos sobre a camada de transporte da

arquitetura *Internet*, envolvendo os protocolos de transporte UDP, TCP e SCTP. Foram mostrados o formato de seus cabeçalhos e os detalhes do estabelecimento e do encerramento de conexões, assim como dos estados de funcionamento, para os protocolos orientados à conexão. Fez-se também um comparativo entre os três protocolos, com ênfase no TCP e no SCTP.

Os aplicativos *peer-to-peer* para compartilhamento de arquivos que apresentam arquiteturas descentralizadas precisam trocar mensagens entre si para encontrar arquivos entre os diversos computadores. Essas mensagens são caracterizadas por possuírem tamanhos variáveis e por não restringirem a ordenação de entrega.

A natureza do TCP, que restringe a ordem da entrega dos dados à camada de aplicação, pode fazer com que a transmissão de mensagens entre os nós de uma rede *peer-to-peer* tenha o seu desempenho degradado, causados pelo problema do HOL. O protocolo SCTP possui a funcionalidade da entrega desordenada de mensagens, que pode melhorar o desempenho desta tarefa.

Para mostrar esse ganho, foi utilizado um ambiente de simulação composto por dois aplicativos que foram desenvolvidos para simular o trânsito de mensagens entre dois nós de uma rede *overlay*, tomando como base a dinâmica de uma rede *Gnutella* real, e um gerador de perda e atraso de pacotes, implementado por meio do aplicativo NIST *Net*, que emula o comportamento que a *Internet* provoca nas mensagens transmitidas.

Comprovou-se, então, que o SCTP tem um melhor desempenho em relação ao TCP, em redes com altas taxas de perda de pacotes, proporcionando um menor tempo médio de transmissão de mensagens nó-a-nó. Isso se deve à natureza ordenada do TCP que, ao restringir a ordem de entrega das mensagens, degrada o desempenho geral, fato que não ocorre quando o SCTP utiliza a entrega desordenada de mensagens. A utilização dos multi-fluxos nesta simulação teve um caráter ilustrativo, para mostrar que o HOL intra-fluxo pode prejudicar de maneira sutil o desempenho da transmissão de mensagens.

Levando em consideração que muitos nós participantes nas redes *peer-to-peer*, para compartilhamento de arquivos, estão inter-conectados por meios que podem

apresentar taxas de perdas de pacotes variadas e que, muitas vezes, estão em regiões geográficas distintas, a utilização da entrega desordenada de mensagens, oferecida pelo SCTP, passa a ser uma opção em potencial para a transmissão das mensagens de busca, satisfazendo as necessidades dos aplicativos *peer-to-peer*.

Assim sendo, uma outra contribuição muito importante deste trabalho foi a análise comparativa de desempenho entre o SCTP e o TCP como protocolos da camada de transporte para aplicativos *peer-to-peer* de compartilhamento de arquivos. Com a utilização do SCTP, o tempo médio de transmissão de mensagens nó-a-nó é reduzido e, com isso, o tempo médio para a chegada das respostas da fase de consulta é diretamente beneficiado.

Para comprovar a viabilidade da utilização do SCTP, em vez do TCP, em um aplicativo *peer-to-peer* de compartilhamento de arquivos, o protótipo de aplicação *Octopus* foi desenvolvido. Este aplicativo utiliza os conceitos das redes *peer-to-peer*, a partir do uso do protocolo *Gnutella* na camada de aplicação, e do SCTP, que é utilizado na camada de transporte.

O objetivo deste desenvolvimento é, por meio deste protótipo, mostrar como algumas funcionalidades do SCTP podem ser aproveitadas em um aplicativo *peer-to-peer* de compartilhamento de arquivos. As funcionalidades exploradas foram a orientação à mensagem, os multi-fluxos e a entrega desordenada de mensagens. Comparativamente ao TCP, é de se esperar um melhor desempenho.

Foi implementada no *Octopus* uma reestruturação do cabeçalho das mensagens *Gnutella* para ser utilizado com o SCTP. Os delimitadores de mensagem, implementados pelos campos de tamanho de mensagem na versão *Gnutella* original, podem ser removidos, pois o SCTP oferece a orientação à mensagem. Os identificadores dos tipos de *payload* também podem ser removidos, pois os identificadores de fluxo podem ser utilizados para essa tarefa, sem necessariamente fazer uso da funcionalidade primária da transmissão de mensagem em um mesmo fluxo, que faz com que a ordenação seja mantida.

A funcionalidade da entrega de mensagens desordenadas foi utilizada para o tráfego de mensagens *Gnutella*. A principal motivação para este uso é a flexibilidade

na ordenação de entrega dessas mensagens. O ganho de desempenho no emprego desta funcionalidade foi mostrado na análise comparativa entre o TCP e o SCTP.

A funcionalidade dos multi-fluxos foi utilizada para realizar a transmissão simultânea de arquivos, através de uma única associação SCTP, que foi também utilizada para transmitir as mensagens *Gnutella*. Um protocolo, baseado no HTTP, foi especificado para a negociação da transferência dos arquivos entre os nós.

Este desenvolvimento teve como contribuição a apresentação da reestruturação do protocolo *Gnutella* que faz uso da orientação à mensagem e dos identificadores de fluxo, e a apresentação de um protótipo de aplicativo que mostra como as funcionalidades do SCTP podem ser utilizadas em um aplicativo *peer-to-peer* de compartilhamento de arquivos. Tem como contribuição, ainda, a especificação de um protocolo simplificado para a negociação da transferência de arquivos.

O presente trabalho teve também como contribuição a publicação de três artigos. Dois deles foram apresentados em congressos [9, 10] e referem-se ao uso do SCTP em aplicativos *peer-to-peer*. O outro refere-se à proposta de classificação das arquiteturas de redes *peer-to-peer*, e foi aceito em um periódico [11].

Como sugestões para a melhoria deste trabalho, tem-se a execução de testes exaustivos para a obtenção de estatísticas mais precisas referentes aos tamanhos e à frequência de transmissão das mensagens *Gnutella*, por meio da implementação de um monitor de rede. Com isso, novas execuções de testes comparativos entre o SCTP e o TCP podem ser feitas para validar os experimentos mostrados.

Uma sugestão para a melhoria do *Octopus* seria a atualização da *interface* do usuário de modo a apresentar outras funcionalidades, tais como a exibição das informações do *cache* de associações e do *host cache* diretamente na *interface* gráfica, além da implementação da mensagem do tipo PUSH. Outra melhoria seria a implementação de um *proxy* que integra a rede *peer-to-peer* do *Octopus* àquela já consolidada pelo protocolo *Gnutella* tradicional.

Como trabalho futuro, é sugerida a especificação e implementação de um protocolo de rede *peer-to-peer* descentralizada baseada em ultranós que faz uso do SCTP, tendo como referência as funcionalidades exploradas neste trabalho, e que, ainda,

faz uso da funcionalidade do *multi-homing* no nível dos ultranós, de modo a alcançar um melhor desempenho e a tolerância a falhas por meio da redundância de enlace. Sugere-se também a utilização do SCTP sob as redes de arquitetura estruturada.

Por fim, sugere-se a implementação de um aplicativo completo que integra, não só o compartilhamento de arquivos, mas também a comunicação entre usuários por meio de texto, áudio e vídeo, em que os multi-fluxos possam ser amplamente explorados.

Referências Bibliográficas

- [1] BACKX, P., WAUTERS, T., DHOEDT, B., AND DEMEESTER, P. A comparison of peer-to-peer architectures. In *Eurescom Summit* (Heidelberg, Germany, 2002), Broadband Communication Networks Group.
- [2] BROOKSHIER, D., GOVONI, D., KRISHNAN, N., AND SOTO, J. C. *JXTA: Java P2P Programming*. Sams, USA, Março 2002. 0672323664.
- [3] CARSON, M., AND SANTAY, D. *NIST Net - A Linux-based Network Emulation Tool*. <http://snad.ncsl.nist.gov/itg/nistnet/> acessado em 12/11/2005.
- [4] CENTER, S. L. A. Iepm. In *Internet End-to-end Performance Monitoring*. University of Stanford, 2006. <http://www-iepm.slac.stanford.edu/> em 09/02/2006.
- [5] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, (CA, USA, Julho 2000), pp. 311–320.
- [6] COENE, L. *RFC 3257 - Stream Control Transmission Protocol Applicability Statement*, 2002. <http://www.sctp.org> acessado em 06/08/2005.
- [7] CRESPO, A., AND GARCIA-MOLINA, H. Routing indices for peer-to-peer systems. In *Proceedings of the 22th International Conference on Distributed Computing Systems* (Washington, DC, 2002), 0-7695-1585-1, IEEE Computer Society, pp. 23–34.
- [8] DASWANI, S., AND FISK, A. A. Gnutella udp extension for scalable searches - guess v0.1. Tech. rep., Gnutella Development Forum, Agosto 2002.

- [9] DO CARMO, G. S. B., AND BARBAR, J. S. Peer-to-peer usando sctp. In *CONTECSI - Congresso Internacional de Gestão de Tecnologia e Sistemas de Informação* (São Paulo, SP, Brasil, 2004), TECSI/FEA/USP.
- [10] DO CARMO, G. S. B., AND BARBAR, J. S. A utilização do protocolo sctp em uma arquitetura peer-to-peer. In *WebMedia/LA-Web - Simpósio Brasileiro de Sistemas Multimídia e Web* (Ribeirão Preto, SP, Brasil, 2004), vol. 2, pp. 265–267.
- [11] DO CARMO, G. S. B., BARBAR, J. S., AND COELHO, F. B. Uma nova classificação para as arquiteturas de redes peer-to-peer. *Revista do IEEE América Latina* (2006). Artigo aceito com previsão de publicação em Julho de 2006.
- [12] ET AL, S. Internet draft. In *Sockets API Extensions for Stream Control Transmission Protocol* (Setembro 2004), Network Working Group, The Internet Society.
- [13] ET AL, S. Internet-draft. In *SCTP Implementer's Guide* (2004), Network Working Group, The Internet Society.
- [14] FANNING, S. *Napster*. <http://www.napster.com> acessado em 25/06/2005.
- [15] GE, Z., FIGUEIREDO, D. R., JAISWAL, S., KUROSE, J., AND TOWSLEY, D. Modeling peer-to-peer file sharing systems. In *Proceedings of INFOCOM* (San Francisco, California, USA, Abril 2003), Department of Computer Science, University of Massachusetts, Amherst.
- [16] HARGREAVES, T. The fasttrack protocol. <http://cvs.berlios.de/> - acessado em 24/10/1980, Julho 2004.
- [17] KIRK, P., Ed. *Gnutella a protocol for a revolution, RFC Gnutella 0.4* (2003). <http://rfc-gnutella.sourceforge.net> acessado em 08/12/2005.
- [18] KIRK, P., Ed. *Gnutella a protocol for a revolution, RFC Gnutella 0.6* (2003). <http://rfc-gnutella.sourceforge.net> acessado em 08/12/2005.

- [19] KULBAK, Y., AND BICKSON, D. Emule protocol guide. School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel, Janeiro 2005.
- [20] KUROSE, J. F., AND ROSS, K. W. *Rede de Computadores e a Internet*. Addison Wesley, São Paulo, 2003. 8588639181.
- [21] LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing* (New York, USA, 2002), A. Press, Ed., 1-58113-483-5, pp. 82–95.
- [22] MARKATOS, E. P. Tracing large-scale peer to peer system: an hour in the life of gnutella. In *2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid* (2002), IEEE Press, Piscataway, NJ, USA.
- [23] MATHIS, M., MAHDAVI, J., FLOYD, S., AND ROMANOW, A. *RFC 2018 - TCP Selective Acknowledgement Options*, Outubro 1996. <http://www.ietf.org/rfc/rfc2018.txt> acessado em 06/08/2005.
- [24] MAUMOUNKOV, P., AND MAZIÈRES, D. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems* (2004).
- [25] MCCALED, J. *eDonkeyProtocol*, 2005. <http://www.edonkey.com> acessado em 24/08/2005.
- [26] MENASCÉ, D. A., AND KANCHANAPALLI, L. Probabilistic scalable p2p resource location services. In *ACM SIGMETRICS Performance Evaluation Review* (New York, 2002), 0163-5999, ACM Press, pp. 48–58.
- [27] MERKUR. Emule project. <http://www.emule-project.net> acessado em 25/05/2005.
- [28] METRIX, J. M. Global napster usage plummets, but new file-sharing alternatives gaining ground. *Software Industry Report*, 14 (Julho 2001), 5.

- [29] MILOJICIC, D. S., KALOGERAKI, V., LUKOSE, R., NAGARAJA, K., PRUYNE, J., RICHARD, B., ROLLINS, S., AND XU, Z. *Peer-to-peer Computing*. HP Laboratories, 2002.
- [30] ONG, L., AND YOAKUM, J. *RFC 3286 - An Introduction to the Stream Control Transmission Protocol*, Outubro 2002. <http://www.sctp.org> acessado em 06/08/2005.
- [31] ORAM, A., Ed. *Peer-to-peer*, 1 ed. O'Reilly Media, Março 2001. 059600110X.
- [32] POSTEL, J. *RFC 768 - User Datagram Protocol*, Agosto 1980. <http://www.faqs.org/rfcs/rfc768.txt> acessado em 06/08/2005.
- [33] RAJAMANI, R., KUMAR, S., AND GUPTA, N. Sctp versus tcp: Comparing the performance of transport protocols for web traffic. Tech. rep., Computer Sciences Department, University of Wisconsin-Madison, Maio 2002.
- [34] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, architectures, and Protocols for Computer Communications* (New York, 2001), 1-58113-411-8, ACM Press, pp. 161–172.
- [35] RITTER, J. Why gnutella can't scale. no, really. Tech. rep., Darkridge, Fevereiro 2001. www.darkridge.com/jpr5/doc/gnutella.html acessado em 15/01/2006.
- [36] ROCHA, J., DOMINGUES, M., CALLADO, A., SOUTO, E., SILVESTRE, G., KAMIENSKI, C., AND SADOK, D. Peer-to-peer: Computação colaborativa na internet. Tech. rep., P2P Work Group of the Brazilian Research Network, 2004.
- [37] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP ACM International Conference on Distributed Systems Platforms* (2001), vol. 2218, Heidelberg, Alemanha, pp. 329–350.
- [38] SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the First In-*

- ternational Conference on Peer-to-Peer Computing* (2002), 0-7695-1503-7/02, IEEE Internet Computing.
- [39] SINGLA, A., AND ROHRS, C. Ultrapeers: Another step towards gnutella scalability. Tech. rep., Lime Wire LLC, 2002. <http://limewire.com>.
- [40] SOURCEFORGE.NET. *LKSCTP Project - Linux Implementation for SCTP*, 2005. <http://lksctp.sourceforge.net> acessado em 14/08/2005.
- [41] STEWART, R. R., AND XIE, Q. *RFC 2960 - Stream Control Transmission Protocol*, 2000. <http://www.sctp.org> acessado em 06/08/2005.
- [42] STEWART, R. R., AND XIE, Q. *Stream Control Transmission Protocol (SCTP) - A Reference Guide*. Addison-Wesley, 2002. 0201721864.
- [43] STOICA, I., MORRIS, R., LIBEN-NOWELL, D., KARGER, D. R., KAASHOEK, M. F., DABEK, F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *IEEE/ACM Transactions on Networking* (2003), vol. 11 of 1063-6692, IEEE Press, Piscataway, NJ, USA, pp. 17 – 32.
- [44] STROKES, M. Gnutella2 specifications. In *Gnutella2 Developer Network*. Gnutella Development Network, 2003. <http://www.gnutella2.com> acessado em 26/01/2006.
- [45] TANEMBAUM, A. S. *Redes de Computadores*. Editora Campus, São Paulo, 2003. 8535211853.
- [46] TEAM, D. S. *Peer-to-peer Application Development*. Hungry Minds, 2002.
- [47] TSOUMAKOS, D., AND ROUSSOPOULOS, N. Adaptive probabilistic search for peer-to-peer networks. In *Proc. 3rd International Conf. Peer-to-Peer Computing (P2P 2003)* (2003), IEEE CS Press, pp. 102–110.
- [48] TSOUMAKOS, D., AND ROUSSOPOULOS, N. A comparison of peer-to-peer search methods. In *International Workshop on the Web and Databases* (Junho 2003).

- [49] V. KALOGERAKI, D. G. E. D. Z.-Y. A local search mechanism for peer-to-peer networks. In *Proceedings of the 11th International Conference on Information and Knowledge Management* (New York, 2002), 1-58113-492-4, ACM Press, pp. 300–307.
- [50] YANG, B., AND GARCIA-MOLINA, H. Comparing hybrid peer-to-peer systems. In *Proceedings of the 27th Very Large Data Bases Conference, Roma, Itália* (2001), 1-55860-804-4, ACM Press, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, pp. 561 – 570.
- [51] YANG, B., AND GARCIA-MOLINA, H. Improving search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (Washington, DC, 2002), 0-7695-1585-1, IEEE Computer Society, pp. 5–14.
- [52] YANG, B., AND GARCIA-MOLINA, H. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India* (Março 2003), Computer Science Department, Stanford University.
- [53] ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications* (Janeiro 2004).

Apêndice A

Códigos fonte dos aplicativos do ambiente de simulação

Os códigos fonte apresentados neste apêndice referem-se aos aplicativos “Gerador e receptor de pacotes”, ou “Nó 1”, e *Echo server*, ou “Nó 2”, descritos no Capítulo 4.

```

/*****
  GeraPacotes.c - Aplicativo gerador e receptor de pacotes
  Gustavo Salvadori Baptista do Carmo
  10 de Fevereiro de 2006
  gustavocarmo@gmail.com
*****/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netinet/sctp.h>
#include <arpa/inet.h>
#include <errno.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>
#include " ./Misc.c"

//-----
//Estruturas de parâmetros para funções - threads
//-----
struct parametroT1
{
    int sock_fd;
    struct sockaddr_in endereco;
};

//-----
//Protótipos
//-----
//Funções para controle geral e exibição de resultados
void uso (void);
void cuidaDoLance (void);
```

```

void parser (int argc, char **argv);
void calculaTempo (void);
void geraScript (void);

//Funcoes para criação de relacionamentos
void criaConexaoTCP (void);
void criaAssociacaoSCTP (void);

//Funções para envio e recebimento de mensagens
void *enviaMensagensTCP (struct parametroT1 *);
void *enviaMensagensSCTP (struct parametroT1 *);
void *recebeMensagensTCP (struct parametroT1 *);
void *recebeMensagensSCTP (struct parametroT1 *);

//Funções específicas para o TCP
int writeN (int fd, const void *buffer, int n);
int readN (int fd, void *buffer, int n);

//-----
//Variáveis globais
//-----
int fluxos = 0;
int fila = 0;
char *enderecoLocal = NULL;
char *enderecoRemoto = NULL;
int portaLocal = 0;
int portaRemota = 0;
int tamanhoBuffer = 0;
int numeroMensagensEnviar = 30;
int tipoDeProtocolo = 0;
int tamanhoMensagem = 20;
int mensagemDesordenada = 0;
int intervaloEntreMensagens = 0;
int verbose = 1;
FILE *arquivo;
FILE *arquivo2;

//Representação de mensagens Gnutella
extern char *gnutella[5];

//estrutura enviada via socket
struct tipao
{
    int id;
    char sendline[100];
};

//Estruturas e variáveis de auxílio no cálculo de tempo
struct paraContagem
{
    int id;
    int tamanho;
    struct timeval cronometroIda;
    struct timeval cronometroVolta;
};

struct timeval tempoGlobalInicial;
struct timeval tempoGlobalFinal;
int quantidadeEnviadaGlobal = 0;
struct paraContagem matriz[NUMERO_MAXIMO_MENSAGENS];

//Miscelânea
double latenciaMedia;
double vazaoMedia;
double vazaoGeral = 0;
double latenciaGeral = 0;
double latenciaTotal;
double vazaoTotal;
double latenciaIndividual[NUMERO_MAXIMO_MENSAGENS];
double vazaoIndividual[NUMERO_MAXIMO_MENSAGENS];

```

```
//-----
//Programa principal
//-----
int
main (int argc, char **argv)
{
    parser (argc, argv);
    cuidaDoLance ();

    int i;

    //Executa o programa por um número definido de vezes
    if (tipoDeProtocolo == 1)
        for (i = 1; i <= verbose; i++)
        {
            //Protocolo escolhido foi o TCP
            criaConexaoTCP ();
        }
    else
        for (i = 1; i <= verbose; i++)
        {
            //Protocolo escolhido foi o SCTP
            criaAssociacaoSCTP ();
        }
    return 0;
}                                     //fim de main

//-----
//Função para preencher os valores da linha de comando
//-----
void
parser (int argc, char **argv)
{
    int getoptC;
    int bufferAux;
    int numAux;
    int tamanhoAux;
    int desordem;
    int intervalo;
    int verb;

    while (argc)
    {
        getoptC = getopt (argc, argv, "H:P:h:p:t:x:f:b:m:s:d:u:v:");

        if (getoptC < 0)
            break;

        switch (getoptC)
        {
            case 'H':
                enderecoLocal = optarg;
                break;

            case 'P':
                portaLocal = atoi (optarg);
                break;

            case 'h':
                enderecoRemoto = optarg;
                break;

            case 'p':
                portaRemota = atoi (optarg);
                break;

            case 't':
                tipoDeProtocolo = atoi (optarg);
                break;
        }
    }
}
```

```

    case 'x':
        fluxos = atoi (optarg);
        break;

    case 'f':
        fila = atoi (optarg);
        break;

    case 'b':
        bufferAux = atoi (optarg);
        if ((bufferAux > 0) && (bufferAux < 1001))
            tamanhoBuffer = bufferAux;
        break;

    case 'm':
        numAux = atoi (optarg);
        if ((numAux > 0) && (numAux < 101))
            numeroMensagensEnviar = numAux;
        break;

    case 's':
        tamanhoAux = atoi (optarg);
        if ((tamanhoAux > 1) && (tamanhoAux < 101))
            tamanhoMensagem = tamanhoAux;
        break;

    case 'd':
        desordem = atoi (optarg);
        if (desordem == 1)
            mensagemDesordenada = MSG_UNORDERED;
        break;

    case 'u':
        intervalo = atoi (optarg);
        if ((intervalo > 0) && (intervalo < 1001))
            intervaloEntreMensagens = intervalo;
        break;

    case 'v':
        verb = atoi (optarg);
        if ((verb > 1) && (verb < 101))
            verbose = verb;
        break;

    default:
        break;
}
//fim do switch
}
//fim while argc
}
//fim de parser

//-----
//Função para verificação dos parâmetros de entrada
//-----
void
cuidaDoLance (void)
{
    if ((enderecoLocal == NULL) ||
        (inet_aton (enderecoLocal, NULL) == 0))
    {
        printf ("\nFaltou colocar o endereço local\n");
        uso ();
        exit (0);
    }

    if (portaLocal == 0)
    {
        printf ("\nFaltou colocar a porta local\n");
        uso ();
        exit (0);
    }
}

```

```
if ((endereçoRemoto == NULL) ||
    (inet_aton (endereçoRemoto, NULL) == 0))
{
    printf ("\nFaltou colocar o endereço remoto\n");
    uso ();
    exit (0);
}

if (portaRemota == 0)
{
    printf ("\nFaltou colocar a porta remota\n");
    uso ();
    exit (0);
}

if ((tipoDeProtocolo != 1) && (tipoDeProtocolo != 2))
{
    printf ("\nEscolha de protocolo incorreta\n");
    printf ("\nTCP=1 e SCTP=2\n");
    uso ();
    exit (0);
}

if ((fluxos < 1) && (tipoDeProtocolo == 2))
{
    printf ("\nFluxo setado para 1\n");
    fluxos = 1;
}

if ((fila < 1) && (tipoDeProtocolo == 2))
{
    printf ("\nFila setada para 1\n");
    fila = 1;
}

printf ("\n");
printf ("Endereço do host local:%s\n", endereçoLocal);
printf ("Porta do host local:%d\n", portaLocal);
printf ("Endereço do host remoto:%s\n", endereçoRemoto);
printf ("Porta do host remoto:%d\n", portaRemota);

if ((tipoDeProtocolo == 1))
    printf ("Protocolo de transporte: TCP\n");
else
{
    //Caso do SCTP
    printf ("Protocolo de transporte: SCTP\n");
    printf ("Numero de fluxos:%d\n", fluxos);
    printf ("Fila de mensagens em cada fluxo:%d\n", fila);

    if (tamanhoBuffer == 0)
        printf ("Buffer de envio e recebimento setado para padrao\n");
    else
        printf ("Buffer de envio e recebimento (kBytes):%d\n",
            tamanhoBuffer);

    if (mensagemDesordenada == 0)
        printf ("Entrega das mensagens: ordenada\n");
    else
        printf ("Entrega das mensagens: desordenada\n");
}
//Fim do caso SCTP

printf ("Numero de mensagens a enviar:%d\n",
    numeroMensagensEnviar);
printf ("Tamanho das mensagens (bytes):%d\n", tamanhoMensagem);
printf ("Intervalo entre mensagens (milisegundos):%d\n",
    intervaloEntreMensagens);
printf ("\n");
}
//fim cuida do lance
```

```

//-----
//Função para imprimir o help
//-----
void
uso (void)
{
    printf ("\n");
    printf ("Uso: ./GeraPacotes -H endLoc -P ptLoc -h endRem -p ptRem -t tipProt\n");
    printf ("          -x numFlux -f fila -b buffer -m mensagens -s size\n");
    printf ("          -d ordem -u tempo -v verbose\n\n");
    printf ("          -H endLoc      - endereço do host local\n");
    printf ("          -P ptLoc       - porta do host local\n");
    printf ("          -h endRem      - endereço do host remoto\n");
    printf ("          -p ptRem       - porta do host remoto\n");
    printf ("          -t tipProt     - protocolo de transporte - TCP=1 - SCTP=2\n");
    printf ("          -x numFlux    - número de fluxos - min 1 - max 300 - SCTP\n");
    printf ("          -f fila       - fila de mensagens em cada fluxo - min 1 - max 5 - SCTP\n");
    printf ("          -b buffer     - buffer de envio e recebimento (kBytes) ");
    printf ("- min 1 - max 1000 - SCTP\n");
    printf ("          -m mensagens  - número de mensagens a enviar - min 1 - max 100\n");
    printf ("          -s size       - tamanho da mensagem (bytes) - min 2 - max 100\n");
    printf ("          -d ordem     - 1 - envia as mensagens de maneira desordenada\n");
    printf ("          -u tempo      - tempo entre mensagens - milisegundos - padrão 50\n");
    printf ("          -v verbose    - verbose 1 ou 2\n");
    printf ("\n");
}
//fim de uso

//-----
//-----
void
criaAssociacaoSCTP (void)
{
    int sock_fd;
    int erro;

    struct sockaddr_in servaddr, meuaddr, enderecoEnviante;
    struct sctp_event_subscribe evnts;

    //Criação do socket
    sock_fd = socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);

    if (sock_fd < 0)
    {
        printf ("Criação do socket falhou\n");
        exit (0);
    }

    //Configurando os endereços
    bzero (&servaddr, sizeof (servaddr));
    servaddr.sin_family = AF_INET;
    inet_aton (enderecoRemoto, &servaddr.sin_addr);
    servaddr.sin_port = htons (portaRemota);

    bzero (&meuaddr, sizeof (meuaddr));
    meuaddr.sin_family = AF_INET;
    inet_aton (enderecoLocal, &meuaddr.sin_addr);
    meuaddr.sin_port = htons (portaLocal);

    //Subscrevendo as notificações de eventos
    bzero (&evnts, sizeof (evnts));
    evnts.sctp_data_io_event = 1;

    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS,
                      &evnts, sizeof (evnts));

    if (erro != 0)
    {
        printf ("Erro a setar opcao do socket de initmessage\n");
        exit (0);
    }
}

```

```
int desliga = 1;

//Desligando o Nagle
erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_NODELAY,
                  &desliga, sizeof (desliga));

if (erro == (-1))
{
    printf ("Erro ao desligar o Nagle do SCTP\n");
    exit (0);
}

struct sctp_initmsg initmsg;

//Configurando o número de fluxos
initmsg.sinit_num_ostreams = fluxos;
initmsg.sinit_max_instreams = fluxos;
initmsg.sinit_max_attempts = 3;
initmsg.sinit_max_init_timeo = 30;

erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                  (char *) &initmsg,
                  sizeof (struct sctp_initmsg));

if (erro != 0)
{
    printf ("Erro a setar opção do socket de initmessage\n");
    exit (0);
}

//Configurando o tamanho do buffer
tamanhoBuffer = tamanhoBuffer * 1000;

if (tamanhoBuffer != 0)
{
    erro = setsockopt (sock_fd, IPPROTO_SCTP, SO_RCVBUF,
                      (char *) &tamanhoBuffer,
                      sizeof (tamanhoBuffer));

    if (erro != 0)
    {
        printf ("Erro a setar opção do socket de receive buffer\n");
        exit (0);
    }

    erro = setsockopt (sock_fd, IPPROTO_SCTP,
                      SO_SNDBUF, (char *) &tamanhoBuffer,
                      sizeof (tamanhoBuffer));

    if (erro != 0)
    {
        printf ("Erro a setar opção do socket de send buffer\n");
        exit (0);
    }
}

//fim tamanhobuffer!=0

//Efetuando o bind
erro = bind (sock_fd, (struct sockaddr *) &meuaddr,
            sizeof (meuaddr));

if (erro != 0)
{
    printf ("Erro ao fazer o bind\n");
    exit (0);
}

//esse trecho é para não contar o tempo do
//estabelecimento de associação do SCTP
struct sctp_sndrcvinfo sri;
struct tipao mensagem;
int quantidadeEnviada;
```

```
int quantidadeRecebida;
int msg_flags;

bzero (&sri, sizeof (sri));
sri.sinfo_stream = 0;

int out_sz;

socklen_t len;
len = sizeof (struct sockaddr_in);

char *sendline;
sendline = malloc (25);

bzero (mensagem.sendline, sizeof (100));
bzero (sendline, sizeof (25));
sprintf (mensagem.sendline, "Conectando");
out_sz = strlen (mensagem.sendline) + sizeof (int);
mensagem.id = 1000;

quantidadeEnviada =
    sctp_sendmsg (sock_fd, "Conectando", out_sz,
                  (struct sockaddr *) &servaddr,
                  sizeof (struct sockaddr), 0,
                  mensagemDesordenada, sri.sinfo_stream, 0,
                  0);

if (quantidadeEnviada == (-1))
{
    printf ("Falha ao enviar a mensagem\n");
}

quantidadeRecebida =
    sctp_rcvmsg (sock_fd, sendline, 25,
                 (struct sockaddr *) &enderecoEnviante,
                 &len, &sri, &msg_flags);

if (quantidadeRecebida == (-1))
{
    printf ("Falha ao receber a mensagem\n");
}

//fim do trecho de estabelecimento de associação SCTP

struct parametroT1 valores1;
struct parametroT1 valores2;

//Preenchendo as estruturas de parâmetros para as threads
valores1.sock_fd = sock_fd;
valores1.endereco = enderecoEnviante;
valores2.sock_fd = sock_fd;
valores2.endereco = servaddr;

pthread_t p_thread_envia;
pthread_t p_thread_recebe;

//Abre a thread que gera e envia mensagens
pthread_create (&p_thread_envia, NULL,
                (void *) enviaMensagensSCTP, &valores2);

//Abre a thread que recebe mensagens
pthread_create (&p_thread_recebe, NULL,
                (void *) recebeMensagensSCTP, &valores1);

//Aguarda os termos das duas threads
pthread_join (p_thread_envia, NULL);
pthread_join (p_thread_recebe, NULL);

//Calcula e exibe resultados
calculaTempo ();
geraScript ();
```

```

    return;
}                                     //fim de cria associação SCTP

//-----
//-----
void
geraScript (void)
{
    printf ("\n");
    printf ("#-----\n");
    printf ("#MANUAL DE LEITURA\n");
    printf ("#-----\n");

    printf ("\n");
    printf ("#-latência média: a média aritmética de todas as latências individuais\n");
    printf ("#-vazão média: a média aritmética de todas as vazões individuais\n");
    printf ("\n");
    printf ("#-latência geral: a soma de todas as latências, não considerando os intervalos\n");
    printf ("#-vazão geral: a soma de todas as vazoes, não considerando os intervalos\n");
    printf ("\n");
    printf ("#-latência total: desde a primeira mensagem até a ultima recebida\n");
    printf ("#-vazão total: desde da primeira mensagem enviada ateh a ultima recebida\n");

    printf ("\n");
    printf ("#-----\n");
    printf ("#INFORMACÕES GERAIS\n");
    printf ("#-----\n");

    printf ("\n");
    printf ("#Endereço do host local:%s\n", enderecoLocal);
    printf ("#Porta do host local:%d\n", portaLocal);
    printf ("#Endereço do host remoto:%s\n", enderecoRemoto);
    printf ("#Porta do host remoto:%d\n", portaRemota);

    if ((tipoDeProtocolo == 1))
        printf ("#Protocolo de transporte: TCP\n");
    else
    {
        //Caso do SCTP
        printf ("#Protocolo de transporte: STCP\n");
        printf ("#Número de fluxos:%d\n", fluxos);
        printf ("#Fila de mensagens em cada fluxo:%d\n", fila);

        if (tamanhoBuffer == 0)
            printf ("#Buffer de envio e recebimento setado para padrão\n");
        else
            printf ("#Buffer de envio e recebimento (kBytes):%d\n",
                    tamanhoBuffer);
    }
    //Fim do caso SCTP

    printf ("#Número de mensagens a enviar:%d\n",
            numeroMensagensEnviar);
    printf ("#Tamanho das mensagens (bytes):%d\n", tamanhoMensagem);
    printf ("\n");
    printf ("#Round Trip Time:\n");
    printf ("#Taxa de perda de pacotes:\n");

    printf ("\n");
    printf ("#-----\n");
    printf ("#IdMsg : Lat(s) : Vaz(b/s) : Lat Media(s) : Vazão Média(b/s) :\n");
    printf ("#-----\n");
    printf ("\n");

    printf ("#-----\n");
    printf ("#Lat Geral(s) : Vazão Geral(b/s) : Lat Total(s) : Vaz Total(b/s)\n");
    printf ("#-----\n");
    printf ("\n");

    int i;

```

```

for (i = 0; i < numeroMensagensEnviar; i++)
{
    printf ("%d          %0.6f          %0.6f          %0.6f          %0.6f          %0.6f\n", i, latenciaIndividual[i], vazaoIndividual[i],
        latenciaMedia, vazaoMedia, latenciaGeral, vazaoGeral, latenciaTotal, vazaoTotal);
}
} //fim de geraScript

//-----
//-----
void
calculaTempo (void)
{
    int i, j;
    double resultadoTempoSec;
    double resultadoTempoMicro;
    double resultadoFinal;
    double vazao;

    printf ("\n");
    printf ("-----\n");
    printf ("CÁLCULO DE LATÊNCIA E VAZÃO POR MENSAGEM ENVIADA\n");
    printf ("-----\n");

    for (i = 0; i < numeroMensagensEnviar; i++)
    {
        for (j = 0; j < numeroMensagensEnviar; j++)
        {
            if (i == matriz[j].id)
            {
                resultadoTempoSec =
                    (matriz[j].cronometroVolta.tv_sec) -
                    (matriz[j].cronometroIda.tv_sec);

                resultadoTempoMicro =
                    (double) (matriz[j].cronometroVolta.
                        tv_usec) -
                    (double) (matriz[j].cronometroIda.tv_usec);

                resultadoFinal =
                    resultadoTempoSec +
                    ((double) resultadoTempoMicro /
                    (double) 1000000);

                printf ("\n");
                printf ("Latência para mensagem %d = %0.6f segundos\n", i, resultadoFinal);

                latenciaIndividual[i] = resultadoFinal;

                vazao = (double) (matriz[j].tamanho) /
                    (double) resultadoFinal;

                printf ("Vazão para mensagem %d = %0.6f bytes/seg\n",
                    i, vazao);

                printf ("Quantidade de dados enviado: %d\n",
                    matriz[j].tamanho);

                vazaoIndividual[i] = vazao;

                j = numeroMensagensEnviar;
            } //fim if
        } //fim for 2
    } //fim for 1

    //Cálculo da latência e vazão globais
    resultadoTempoSec =
        (tempoGlobalFinal.tv_sec) - (tempoGlobalInicial.tv_sec);

```

```

    resultadoTempoMicro =
        (double) (tempoGlobalFinal.tv_usec) -
        (double) (tempoGlobalInicial.tv_usec);

    resultadoFinal =
        resultadoTempoSec +
        ((double) resultadoTempoMicro / (double) 1000000);

    printf ("\n");
    printf ("-----\n");
    printf ("\n");
    printf ("Latência Total Final = %0.6f segundos\n",
        resultadoFinal);

    latenciaTotal = resultadoFinal;

    vazao = (double) (quantidadeEnviadaGlobal) /
        (double) resultadoFinal;

    printf ("Quantidade de dados enviados: %d bytes\n",
        quantidadeEnviadaGlobal);

    printf ("Vazao Total Final = %0.6f bytes/seg\n\n", vazao);

    vazaoTotal = vazao;

    for (i = 0; i < numeroMensagensEnviar; i++)
    {
        latenciaGeral = latenciaIndividual[i] + latenciaGeral;

        vazaoGeral = vazaoIndividual[i] + vazaoGeral;
    }

    printf ("Latência Geral = %0.6f seg\n\n", latenciaGeral);

    latenciaMedia =
        (double) latenciaGeral / (double) numeroMensagensEnviar;

    printf ("Latência Média = %0.6f seg\n\n", latenciaMedia);

    printf ("Vazão Geral = %0.6f bytes/seg\n\n", vazaoGeral);

    vazaoMedia =
        (double) vazaoGeral / (double) numeroMensagensEnviar;

    printf ("Vazão Media = %0.6f bytes/seg\n\n", vazaoMedia);
} //fim de calculaTempo

//-----
//Função que gera e envia mensagens através do SCTP
//-----
void *
enviaMensagensSCTP (struct parametroT1 *variaveis)
{
    printf ("Entrei na thread de ENVIAR mensagens\n");

    struct tipao mensagem;

    int sock_fd = variaveis->sock_fd;
    struct sockaddr_in servaddr = variaveis->endereco;

    int loop;
    int indiceFluxos = 0;
    int out_sz;
    int indiceFila = fila;
    int indiceMensagem = 0;
    int quantidadeEnviada;
    struct sctp_sndrcvinfo sri;

```

```

//Essa estrutura é utilizada para saber em qual fluxo
//a mensagem veio
bzero (&sri, sizeof (sri));
sri.sinfo_stream = 0;

printf ("\n");
printf ("-----\n");
printf ("DADOS SOBRE O ENVIO DE MENSAGENS\n");
printf ("-----\n");

//loop de envio de mensagens
for (loop = 0; loop < numeroMensagensEnviar; loop++)
{
    sri.sinfo_stream = indiceFluxos;
    bzero (mensagem.sendline, sizeof (100));
    snprintf (mensagem.sendline, tamanhoMensagem,
              gnutella[loop % 5]);
    out_sz = strlen (mensagem.sendline) + 1;
    mensagem.id = loop;

    printf ("\n");
    printf ("ENVIEI\n");
    printf ("ID mensagem:%d\n", mensagem.id);
    printf ("Mensagem:%s\n", mensagem.sendline);
    printf ("Numero Fluxo:%d\n", indiceFluxos);
    printf ("Indice Fila:%d\n", indiceFila);
    printf ("Numero Loop:%d\n", loop);

    //Tempo entre as mensagens em milisegundos
    usleep (1000 * intervaloEntreMensagens);

    //Início do cálculo do tempo de ida da mensagem
    matriz[loop].id = mensagem.id;

    gettimeofday (&matriz[loop].cronometroIda, NULL);

    //Enviando a mensagem
    quantidadeEnviada =
        sctp_sendmsg (sock_fd, &mensagem, out_sz,
                     (struct sockaddr *) &servaddr,
                     sizeof (struct sockaddr), 0,
                     mensagemDesordenada,
                     sri.sinfo_stream, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar mensagem\n");
    }

    //Para cálculo do total de bytes enviados
    quantidadeEnviadaGlobal =
        quantidadeEnviadaGlobal + quantidadeEnviada;

    printf ("Bytes enviados:%d\n", quantidadeEnviada);
    matriz[loop].tamanho = quantidadeEnviada;

    //Controle de fila nos fluxos
    if (indiceFila == 1)
    {
        if (indiceFluxos == (fluxos - 1))
        {
            indiceFluxos = 0;
        }
        else
        {
            indiceFluxos++;
        }
        indiceFila = fila;
    }
    else
        indiceFila--;
}

```

```

        if (indiceMensagem == 4)
            indiceMensagem = 0;
        else
            indiceMensagem++;
    }
    //fim do loop for de envio de mensagens

return 0;
}
//fim de envioMensagens

```

```

//-----
//Função que recebe as mensagens através do SCTP
//-----
void *
recebeMensagensSCTP (struct parametroT1 *variaveis)
{
    printf ("Entrei na thread de RECEBER mensagens\n");

    //Utilizado para receber a mensagem
    struct tipao mensagem;

    //Preenchendo as variáveis locais com os dados
    //do parâmetro
    int sock_fd = variaveis->sock_fd;
    struct sockaddr_in enderecoEnviante = variaveis->endereco;

    int loop;
    int quantidadeRecebida;
    struct sctp_sndrcvinfo sri;

    char *sendline;
    sendline = malloc (105);

    socklen_t len;
    len = sizeof (struct sockaddr_in);
    int msg_flags;

    //Estrutura utilizada para saber em qual fluxo
    //a mensagem veio
    bzero (&sri, sizeof (sri));
    sri.sinfo_stream = 0;

    printf ("\n");
    printf ("-----\n");
    printf ("DADOS SOBRE A RECEPÇÃO DE MENSAGENS\n");
    printf ("-----\n");

    //loop de recepção de mensagens
    for (loop = 0; loop < numeroMensagensEnviar; loop++)
    {
        bzero (sendline, 105);

        //Recebendo a mensagem
        quantidadeRecebida =
            sctp_rcvmsg (sock_fd, sendline, 105,
                        (struct sockaddr *) &enderecoEnviante,
                        &len, &sri, &msg_flags);

        //Armazenando o momento da chegada da mensagem
        gettimeofday (&matriz[loop].cronometroVolta, NULL);

        bzero (mensagem.sendline, 100);
        memcpy (&mensagem, sendline, quantidadeRecebida);

        matriz[loop].id = mensagem.id;

        if (quantidadeRecebida == (-1))
        {
            printf ("Falha ao receber a mensagem\n");
            loop = loop - 1;
        }
    }
}

```

```

        continue;
    }

    if (loop == numeroMensagensEnviar - 1)
    {
        tempoGlobalInicial = matriz[0].cronometroIda;
        tempoGlobalFinal = matriz[loop].cronometroVolta;
    }

    //Impressão de dados para recebimento
    printf ("\n");
    printf ("RECEBI\n");
    printf ("ID=%d - Mensagem:%s\n",
            mensagem.id, mensagem.sendline);
    printf ("Numero Fluxo:%d\n", sri.sinfo_stream);
    printf ("Quantidade recebida:%d\n", quantidadeRecebida);

    }                                     //fim do loop de recepção de mensagens

    free (sendline);
    close (sock_fd);

    return 0;
}                                     //fim recebe mensagens

//-----
//Função para enviar N bytes através do TCP
//-----
int
writeN (int fd, const void *buffer, int n)
{
    int nRestante;
    int nEscritos;
    const char *ptr;

    ptr = buffer;
    nRestante = n;

    while (nRestante > 0)
    {
        if ((nEscritos = write (fd, ptr, nRestante)) <= 0)
        {
            if (nEscritos < 0 && errno == EINTR)
                nEscritos = 0;
            else
                return (-1);
        }

        nRestante -= nEscritos;
        ptr += nEscritos;
    }

    return n;
}                                     //fim do writeN

//-----
//Função para receber N bytes através do TCP
//-----
int
readN (int fd, void *buffer, int n)
{
    int nRestante;
    int nLidos;
    char *ptr;

    ptr = buffer;
    nRestante = n;

```

```
while (nRestante > 0)
{
    if ((nLidos = read (fd, ptr, nRestante)) < 0)
    {
        if (errno == EINTR)
            nLidos = 0;
        else
            return (-1);
    }
    else if (nLidos == 0)
        break;

    nRestante -= nLidos;
    ptr += nLidos;
}

return (n - nRestante);
}                                     //fim do readN

//-----
//Função para criar a conexão TCP com o echo server
//-----
void
criaConexaoTCP (void)
{
    int sock_fd;
    int erro;

    struct sockaddr_in servaddr, meuaddr;

    //Abre o socket
    sock_fd = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (sock_fd < 0)
    {
        printf ("Criação do socket falhou\n");
        exit (0);
    }

    int desliga = 1;

    //Desliga o Nagle
    erro = setsockopt (sock_fd, IPPROTO_TCP, TCP_NODELAY,
                       &desliga, sizeof (desliga));

    if (erro == (-1))
    {
        printf ("Falha ao desligar o Nagle do TCP\n");
    }

    //Definindo os endereços
    bzero (&servaddr, sizeof (servaddr));
    servaddr.sin_family = AF_INET;
    inet_aton (enderecoRemoto, &servaddr.sin_addr);
    servaddr.sin_port = htons (portaRemota);

    bzero (&meuaddr, sizeof (meuaddr));
    meuaddr.sin_family = AF_INET;
    inet_aton (enderecoLocal, &meuaddr.sin_addr);
    meuaddr.sin_port = htons (portaLocal);

    //Tentando conectar ao echo server
    erro = connect (sock_fd, (struct sockaddr *) &servaddr,
                   sizeof (servaddr));

    if (erro < 0)
    {
        printf ("Erro ao efetuar o connect\n");
        exit (0);
    }
}
```

```

    }

    struct parametroT1 valores1;
    struct parametroT1 valores2;

    valores1.sock_fd = sock_fd;
    valores2.sock_fd = sock_fd;

    pthread_t p_thread_envia;
    pthread_t p_thread_recebe;

    //Cria a thread que envia mensagens
    pthread_create (&p_thread_envia, NULL,
                   (void *) enviaMensagensTCP, &valores2);

    //Cria a thread que recebe mensagens
    pthread_create (&p_thread_recebe, NULL,
                   (void *) recebeMensagensTCP, &valores1);

    //Aguarda as duas threads terminarem
    pthread_join (p_thread_envia, NULL);
    pthread_join (p_thread_recebe, NULL);

    //Calcula resultados e os exibe
    calculaTempo ();
    geraScript ();

    return;
} //fim de cria conexao TCP

//-----
//Função para o envio de mensagens através do TCP
//-----
void *
enviaMensagensTCP (struct parametroT1 *variaveis)
{
    struct tipao mensagem;

    int sock_fd = variaveis->sock_fd;
    int out_sz;
    int loop;
    int quantidadeEnviada;

    printf ("\n");
    printf ("-----\n");
    printf ("DADOS DE ENVIO DE MENSAGENS\n");
    printf ("-----\n");

    //loop de envio de mensagens
    for (loop = 0; loop <= numeroMensagensEnviar; loop++)
    {
        bzero (mensagem.sendline, sizeof (100));
        snprintf (mensagem.sendline, tamanhoMensagem,
                  gnutella[loop % 5]);
        out_sz = strlen (mensagem.sendline) + 1;
        mensagem.id = loop;

        //Impressão de dados sobre o envio
        printf ("\n");
        printf ("ENVIEI\n");
        printf ("ID mensagem:%d\n", mensagem.id);
        printf ("Mensagem:%s\n", mensagem.sendline);
        printf ("Número Loop:%d\n", loop);

        //Início do cálculo do tempo de ida da mensagem
        matriz[loop].id = mensagem.id;
        gettimeofday (&matriz[loop].cronometroIda, NULL);

        usleep (1000 * intervaloEntreMensagens);
    }
}

```

```

quantidadeEnviada = writeN (sock_fd, &mensagem, out_sz);

if (quantidadeEnviada == (-1))
{
    printf ("Falha ao enviar mensagem\n");
}

//Para cálculo do total de bytes enviados
quantidadeEnviadaGlobal =
    quantidadeEnviadaGlobal + quantidadeEnviada;

printf ("Bytes enviados:%d\n", quantidadeEnviada);
matriz[loop].tamanho = quantidadeEnviada;

}
return 0;
}

//fim do loop for de envio de mensagens
//fim de envia mensagens

//-----
//Função que recebe mensagens através do TCP
//-----
void *
recebeMensagensTCP (struct parametroT1 *variaveis)
{
    struct tipao mensagem;

    int sock_fd = variaveis->sock_fd;
    int loop;
    socklen_t len;
    int quantidadeRecebida;

    len = sizeof (struct sockaddr_in);

    char *sendline;
    sendline = malloc (105);

    printf ("\n");
    printf ("-----\n");
    printf ("DADOS DE RECEPCAO DE MENSAGENS\n");
    printf ("-----\n");

    //loop de recepção de mensagens
    for (loop = 0; loop <= numeroMensagensEnviar; loop++)
    {
        bzero (sendline, 105);
        quantidadeRecebida =
            readN (sock_fd, sendline, matriz[loop].tamanho);

        gettimeofday (&matriz[loop].cronometroVolta, NULL);

        if (quantidadeRecebida == (-1))
        {
            printf ("Falha ao receber a mensagem\n");
            loop = loop - 1;
            continue;
        }

        if (quantidadeRecebida == (0))
        {
            loop = loop - 1;
            continue;
        }

        bzero (mensagem.sendline, 100);
        memcpy (&mensagem, sendline, quantidadeRecebida);

        if (loop == numeroMensagensEnviar - 1)
        {
            tempoGlobalInicial = matriz[0].cronometroIda;

```

```
        tempoGlobalFinal = matriz[loop].cronometroVolta;
    }

    //Impressão de dados para recebimento
    printf ("\n");
    printf ("RECEBI\n");
    printf ("ID=%d - Mensagem:%s\n", mensagem.id,
            mensagem.sendline);
    printf ("Quantidade Recebida: %d\n", quantidadeRecebida);
    printf ("Quantidade que era pra receber: %d\n",
            matriz[loop].tamanho);

    }                                     //fim do loop de recepção de mensagens

    free (sendline);
    close (sock_fd);

    return 0;
}                                     //fim de recebe mensagens

//-----
//Fim de GeraPacotes.c
//-----
```

```

/*****
    EchoServer.c - Echo Server
    Gustavo Salvadori Baptista do Carmo
    10 de Fevereiro de 2006
    gustavocarmo@gmail.com
*****/

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/sctp.h>
#include <netinet/tcp.h>
#include <errno.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

//-----
//Protótipos
//-----
void uso (void);
void parser (int argc, char **argv);
void cuidaDoLance (void);
void echoSCTP (void);
void echoTCP (void);
int writeN (int fd, const void *buffer, int n);

//-----
//Variáveis globais
//-----
int tamanhoBuffer = 0;
int portaLocal = 0;
char *enderecoLocal = NULL;
int mensagemDesordenada = 0;
int tipoProtocolo = 1;

//Estrutura usada para receber mensagens
struct tipao
{
    int id;
    char sendline[100];
} mensagem;

//-----
//Programa principal
//-----
int
main (int argc, char **argv)
{
    parser (argc, argv);
    cuidaDoLance ();

    if (tipoProtocolo == 1)
        echoTCP ();
    else if (tipoProtocolo == 2)
        echoSCTP ();
    else
        uso ();

    return 0;
}

```

```
//-----  
//EchoServer com o TCP  
//-----  
void  
echoTCP ()  
{  
    int sock_fd, sock_novo;  
    int erro;  
  
    struct sockaddr_in servaddr, cliaddr;  
  
    sock_fd = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);  
  
    if (sock_fd < 0)  
    {  
        printf ("Criação do socket falhou\n");  
        exit (0);  
    }  
  
    bzero (&servaddr, sizeof (servaddr));  
    servaddr.sin_family = AF_INET;  
    inet_aton (enderecoLocal, &servaddr.sin_addr);  
    servaddr.sin_port = htons (portaLocal);  
  
    int desliga = 1;  
  
    erro = setsockopt (sock_fd, IPPROTO_TCP, TCP_NODELAY,  
                      &desliga, sizeof (desliga));  
  
    if (erro == (-1))  
    {  
        printf ("Falha ao desligar o Nagle do TCP\n");  
    }  
  
    erro = bind (sock_fd, (struct sockaddr *) &servaddr,  
                sizeof (servaddr));  
  
    if (erro != 0)  
    {  
        printf ("Erro ao dar o bind\n");  
        exit (0);  
    }  
  
    erro = listen (sock_fd, 1);  
  
    if (erro != 0)  
    {  
        printf ("Erro ao entrar no listen\n");  
        exit (0);  
    }  
  
    socklen_t len;  
  
    len = sizeof (struct sockaddr_in);  
  
    while (1)  
    {  
        sock_novo =  
            accept (sock_fd, (struct sockaddr *) &cliaddr, &len);  
  
        if (sock_novo < 0)  
        {  
            printf ("Erro no accept\n");  
            exit (0);  
        }  
  
        char *readbuf;  
        int quantidadeEnviada;  
        int quantidadeRecebida;  
  
        readbuf = malloc (100);
```

```

printf ("-----\n");
printf ("DADOS DA RECEPÇÃO\n");
printf ("-----\n");

while (1)
{
    bzero (readbuf, 100);

    //Recebendo o segmento do cliente
    quantidadeRecebida = read (sock_novo, readbuf, 100);

    if (quantidadeRecebida == (-1))
    {
        printf ("Falha ao receber os bytes\n");
    }

    //O cliente encerra a comunicação
    if (quantidadeRecebida == 0)
    {
        printf ("\nCliente encerrou a comunicação\n\n");
        close (sock_novo);
        free (readbuf);
        break;
    }

    printf ("\n");
    printf ("Número de bytes recebidos:%d\n",
            quantidadeRecebida);
    printf ("Endereço remoto:%s\n",
            inet_ntoa (cliaddr.sin_addr));

    //Enviando o segmento devolta para o cliente
    quantidadeEnviada =
        writeN (sock_novo, readbuf, quantidadeRecebida);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar os bytes\n");
    }
} //fim do while 1
}
return;
} //Fim de Echo Server TCP

//-----
//EchoServer com o SCTP
//-----
void
echoSCTP ()
{
    int sock_fd;
    int erro;

    struct sockaddr_in servaddr, cliaddr;
    struct sctp_sndrcvinfo sri;
    struct sctp_event_subscribe evnts;

    sock_fd = socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);

    if (sock_fd < 0)
    {
        printf ("Criação do socket falhou\n");
        exit (0);
    }

    bzero (&servaddr, sizeof (servaddr));
    servaddr.sin_family = AF_INET;
    inet_aton (endereçoLocal, &servaddr.sin_addr);
    servaddr.sin_port = htons (portaLocal);

```

```
bzero (&evnts, sizeof (evnts));
evnts.sctp_data_io_event = 1;

erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS,
                  &evnts, sizeof (evnts));

if (erro != 0)
{
    printf ("Erro ao setar a opção setar eventos\n");
    exit (0);
}

struct sctp_initmsg initmsg;

initmsg.sinit_num_ostreams = 30;
initmsg.sinit_max_instreams = 30;
initmsg.sinit_max_attempts = 3;
initmsg.sinit_max_init_timeo = 30;

int desliga = 1;

erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_NODELAY,
                  &desliga, sizeof (desliga));

if (erro == (-1))
{
    printf ("Erro ao desligar o Nagle do SCTP\n");
    exit (0);
}

erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                  (char *) &initmsg,
                  sizeof (struct sctp_initmsg));

if (erro != 0)
{
    printf ("Erro ao setar a opção de initmessage\n");
    exit (0);
}

tamanhoBuffer = tamanhoBuffer * 1000;

if (tamanhoBuffer != 0)
{
    erro = setsockopt (sock_fd, IPPROTO_SCTP, SO_RCVBUF,
                      (char *) &tamanhoBuffer,
                      sizeof (tamanhoBuffer));

    if (erro != 0)
    {
        printf ("Erro a setar opção do socket de receive buffer\n");
        exit (0);
    }

    erro = setsockopt (sock_fd, IPPROTO_SCTP, SO_SNDBUF,
                      (char *) &tamanhoBuffer,
                      sizeof (tamanhoBuffer));

    if (erro != 0)
    {
        printf ("Erro a setar opção do socket de send buffer\n");
        exit (0);
    }
}

//fim tamanho buffer

erro = bind (sock_fd, (struct sockaddr *) &servaddr,
            sizeof (servaddr));

if (erro != 0)
{
    printf ("Erro ao dar o bind\n");
}
```

```

        exit (0);
    }

    erro = listen (sock_fd, 1);

    if (erro != 0)
    {
        printf ("Erro ao setar a entrar no listen\n");
        exit (0);
    }

    socklen_t len;
    int msg_flags;
    char *readbuf;
    int quantidadeEnviada;
    int quantidadeRecebida;

    len = sizeof (struct sockaddr_in);
    readbuf = malloc (105);

    while (1)
    {
        bzero (readbuf, 105);

        //Recebendo a mensagem do cliente
        quantidadeRecebida = sctp_recvmmsg (sock_fd, readbuf, 105,
                                           (struct sockaddr *)
                                           &cliaddr, &len, &sri,
                                           &msg_flags);

        if (quantidadeRecebida == (-1))
        {
            printf ("Falha ao receber a mensagem\n");
        }

        bzero (mensagem.sendline, 100);
        memcpy (&mensagem, readbuf, quantidadeRecebida);

        //DADOS PARA CONFERÊNCIA
        printf ("\n");
        printf ("Id:%d\n", mensagem.id);
        printf ("buffer:%s\n", mensagem.sendline);
        printf ("Numero da stream:%d\n", sri.sinfo_stream);
        printf ("Numero de bytes recebidos:%d\n",
                quantidadeRecebida);
        printf ("Endereco remoto:%s\n",
                inet_ntoa (cliaddr.sin_addr));

        //enviando a mensagem devolta para o cliente
        quantidadeEnviada =
            sctp_sendmsg (sock_fd, readbuf, quantidadeRecebida,
                        (struct sockaddr *) &cliaddr,
                        sizeof (struct sockaddr),
                        sri.sinfo_ppid, mensagemDesordenada,
                        sri.sinfo_stream, 0, 0);

        if (quantidadeEnviada == (-1))
        {
            printf ("Falha ao enviar a mensagem\n");
        }
    }
    //fim do while 1
    free (readbuf);
    return;
}
//fim de Echo Server SCTP

//-----
//Conferindo os dados da linha de comando
//-----
void
cuidaDoLance (void)
{
    if ((endereçoLocal == NULL) ||

```

```
(inet_aton (enderecoLocal, NULL) == 0))
{
    printf ("\nFaltou colocar o endereco local\n");
    uso ();
    exit (0);
}

if (portaLocal == 0)
{
    printf ("\nFaltou colocar a porta local\n");
    uso ();
    exit (0);
}

printf ("\n");
printf ("Endereco do host local:%s\n", enderecoLocal);
printf ("Porta do host local:%d\n", portaLocal);
printf ("Protocolo de transporte = ");

if (tipoProtocolo == 1)
    printf ("TCP\n");
else
    printf ("SCTP\n");

if (tamanhoBuffer == 0)
    printf ("Buffer de envio e recebimento setado para padrao\n");
else
    printf ("Buffer de envio e recebimento (Kbytes):%d\n",
            tamanhoBuffer);

if (mensagemDesordenada == 0)
    printf ("Entrega das mensagens: ordenada\n");
else
    printf ("Entrega das mensagens: desordenada\n");

printf ("\n");
}                                     //termina cuida do lance

//-----
//Preenchendo com os dados da linha de comando
//-----
void
parser (int argc, char **argv)
{
    int getoptC;
    int bufferAux;
    int desordem;
    int protocolo;

    while (argc)
    {
        getoptC = getopt (argc, argv, "H:P:t:b:d:");

        if (getoptC < 0)
            break;

        switch (getoptC)
        {
            case 'H':
                enderecoLocal = optarg;
                break;

            case 'P':
                portaLocal = atoi (optarg);
                break;

            case 't':
                protocolo = atoi (optarg);
                if ((protocolo == 1) || (protocolo == 2))
                    tipoProtocolo = protocolo;
```

```

        break;

    case 'b':
        bufferAux = atoi (optarg);
        if ((bufferAux > 0) && (bufferAux < 1001))
            tamanhoBuffer = bufferAux;
        break;

    case 'd':
        desordem = atoi (optarg);
        if (desordem == 1)
            mensagemDesordenada = MSG_UNORDERED;
        break;

    default:
        break;
    }
    //fim do switch
}
//fim while argc
}
//fim de parser

//-----
//Imprime uso do aplicativo
//-----
void
uso (void)
{
    printf ("\nUso:\n");
    printf (". /EchoServer -H endLoc -P ptLoc -t tipProt -b buffer -d ordem\n");
    printf ("        -H endLoc        - endereco do host local\n");
    printf ("        -P ptLoc            - porta do host local\n");
    printf ("        -t tipProt          - protocolo de transporte - TCP=1 - SCTP=2\n");
    printf ("        -b buffer           - buffer de envio e recebimento (Kbytes) ");
    printf (" - min 1 - max 1000\n");
    printf ("        -d ordem            - 1 - envia as mensagens de forma desordenada\n");
    printf ("\n");
}
//fim de uso

//-----
//Função para forçar a escrita de todos os bytes TCP
//-----
int
writeN (int fd, const void *buffer, int n)
{
    int nRestante;
    int nEscritos;
    const char *ptr;

    ptr = buffer;
    nRestante = n;

    while (nRestante > 0)
    {
        if ((nEscritos = write (fd, ptr, nRestante)) <= 0)
        {
            if (nEscritos < 0 && errno == EINTR)
                nEscritos = 0;
            else
                return (-1);
        }

        nRestante -= nEscritos;
        ptr += nEscritos;
    }
    return n;
}
//fim de writeN

//-----
//Fim de EchoServer.c
//-----

```

Apêndice B

Código fonte do aplicativo *Octopus*

O código fonte apresentado neste apêndice refere-se ao aplicativo *Octopus*, descrito no Capítulo 4.

```

/*****
    octopus.c - Interface grafica do aplicativo Octopus
    Versao 1.0
    Gustavo Salvadori Baptista do Carmo
    27 de Fevereiro de 2006
    gustavocarmo@gmail.com
*****/

#include </usr/include/gtk-2.0/gtk/gtk.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <time.h>
#include "SERVENTE.c"

//-----
//Headers das funcoes locais
//-----

void *fazGraficos (void);
void delete_event (void);
static void associa (GtkWidget * widget, GtkWidget * entry);
static void enviaDownload (GtkWidget * widget, GtkWidget * entry);
void selectionMade (GtkWidget * clist, gint row, gint column,gpointer data);
GtkWidget *image_new_from_file_at_size (const char *filename,
                                         gint width, gint height);

//-----
//Headers das funcoes do SERVENTE.c
//-----

void *abreServidor (struct sockaddr_in *servaddr);
void imprimeAssociacoesAbertas (void);
void enviaQUERY (char *consulta);
void enviaPING (void);
void pedeArquivo (char *nomeArquivo, char *endereco, char *porta,char *tamanho);
int enderecoIgual (struct sockaddr_in endereco1,struct sockaddr_in endereco2);
int associaPeer (struct sockaddr_in);
void associaHostCache (void);
void imprimeHostCache (void);
int incluiHostCache (struct sockaddr_in);
```

```
//-----
//Variáveis globais
//-----
struct sockaddr_in enderecoConectar;
struct sockaddr_in meuaddr;
GtkWidget *porta;
GtkWidget *listaRemota;
gint linhaListaLocal = 500;
gint colunaListaLocal = 0;
extern int emDownload;

//-----
//Programa principal
//-----
int
main (int argc, char *argv[])
{
    int porta;
    if (argc != 3)
    {
        printf ("./octopus Endereco Porta\n");
        exit (0);
    }

    porta = atoi (argv[2]);

    //Endereco local do aplicativo
    bzero (&meuaddr, sizeof (meuaddr));
    meuaddr.sin_family = AF_INET;
    inet_aton (argv[1], &meuaddr.sin_addr);
    meuaddr.sin_port = htons (porta);

    //Criando a thread do servidor
    pthread_t p_thread_servidor;
    pthread_create (&p_thread_servidor, NULL, (void *) abreServidor,
                   &meuaddr);

    //Criando a thread da interface grafica
    pthread_t p_thread_graficos;
    pthread_create (&p_thread_graficos, NULL, fazGraficos (), NULL);

    return 0;
}                                     //fim de main

//-----
//Funcao de evento do botao associar
//-----
static void
associa (GtkWidget * widget, GtkWidget * entry)
{
    const gchar *enderecoTexto;
    const gchar *portaTexto;
    struct sockaddr_in servaddr;
    int erro;

    enderecoTexto = gtk_entry_get_text (GTK_ENTRY (entry));
    portaTexto = gtk_entry_get_text (GTK_ENTRY (porta));

    if ((strcmp (portaTexto, "") == 0) || (strcmp (enderecoTexto, "") == 0))
    {
        printf ("Nenhum endereco digitado para a associacao\n");
        return;
    }

    bzero (&servaddr, sizeof (servaddr));
    servaddr.sin_family = AF_INET;
    inet_aton (enderecoTexto, &servaddr.sin_addr);
    servaddr.sin_port = htons (atoi (portaTexto));
}
```

```

    if (widget == widget)
    {
    };

    if (strcmp (enderecoTexto, inet_ntoa (meuaddr.sin_addr)) == 0)
        printf ("Associar a si mesmo nao pode\n");
    else
    {
        erro = associaPeer (servaddr);
        if (erro == 1)
            incluiHostCache (servaddr);
        gtk_entry_set_text (GTK_ENTRY (entry), "");
        gtk_entry_set_text (GTK_ENTRY (porta), "");
    }
} //Fim de associa

//-----
//Funcao para limpar a lista de respostas
//-----
void
limpaLista (GtkWidget * widget, GtkWidget * entry)
{
    if (widget == widget)
    {
    };
    gtk_clist_clear (GTK_CLIST (entry));
    linhaListaLocal = 500;
}

//-----
//Funcao de controle da lista de respostas
//-----
void
selection_made (GtkWidget * clist, gint row, gint column,
                gpointer data)
{
    if (clist == clist)
    {
    };
    if (data == data)
    {
    };

    linhaListaLocal = row;
    colunaListaLocal = column;

    return;
}

//-----
//Funcao que envia o pedido de transferencia de arquivo
//-----
static void
enviaDownload (GtkWidget * widget, GtkWidget * entry)
{
    if (widget == widget)
    {
    };

    gchar *nomeArquivo;
    gchar *endereco;
    gchar *porta;
    gchar *tamanho;

    gtk_clist_get_text (GTK_CLIST (entry), linhaListaLocal, 0, &nomeArquivo);
    gtk_clist_get_text (GTK_CLIST (entry), linhaListaLocal, 1, &endereco);
    gtk_clist_get_text (GTK_CLIST (entry), linhaListaLocal, 2, &porta);

```

```

gtk_clist_get_text (GTK_CLIST (entry), linhaListaLocal, 3, &tamanho);

if (linhaListaLocal != 500)
    pedeArquivo (nomeArquivo, endereco, porta, tamanho);

}                                     //fim de enviaDownload

//-----
//Funcao que envia uma consulta para todos os nos associados
//-----
static void
enviaConsulta (GtkWidget * widget, GtkWidget * entry)
{
    if (widget == widget)
    {
        };

    const gchar *consultaTexto;

    bzero (&consultaTexto, sizeof (gchar));
    consultaTexto = gtk_entry_get_text (GTK_ENTRY (entry));
    printf ("A string de consulta:%s\n", consultaTexto);

    if (strlen (consultaTexto) < 4)
        printf ("Consulta muito curta; use mais de 3 letras\n");
    else
    {
        enviaQUERY ((char *) consultaTexto);
    }
}                                     //fim de enviaConsulta

//-----
//Função evento de clique no X da janela
//-----
void
delete_event ()
{
    exit (0);
}

//-----
//Funcao para a exibicao da interface grafica
//-----
void *
fazGraficos ()
{
    GtkWidget *janelaGeral;
    GtkWidget *logo;
    GtkWidget *letreiro;
    GtkWidget *fixed;
    GtkWidget *botaoAssocia;
    GtkWidget *botaoDownload;
    GtkWidget *botaoConsulta;
    GtkWidget *botaoAssociacao;
    GtkWidget *botaoPing;
    GtkWidget *botaoLimpa;
    GtkWidget *endereco;
    GtkWidget *consulta;
    GtkWidget *labelEndereco;
    GtkWidget *labelPorta;
    GtkWidget *labelConsulta;
    GtkWidget *botaoAssociaHostCache;
    GtkWidget *botaoImprimeHostCache;

    gtk_init (NULL, NULL);

    gchar *titles[] = { "Nome do arquivo", "Endereco", "Porta", "Tamanho" };

    //Criando os widgets
    janelaGeral = gtk_window_new (GTK_WINDOW_TOPLEVEL);

```

```
logo = image_new_from_file_at_size ("./logo.png", 150, 150);
letreiro = image_new_from_file_at_size ("./letreiro.png", 150, 180);
fixed = gtk_fixed_new ();
botaoAssocia = gtk_button_new_with_label ("Associar");
botaoDownload = gtk_button_new_with_label ("Download");
botaoConsulta = gtk_button_new_with_label ("Consultar");
botaoAssociaHostCache = gtk_button_new_with_label ("Associa Host Cache");
botaoImprimeHostCache = gtk_button_new_with_label ("Host Cache");
botaoPing = gtk_button_new_with_label ("Ping");
botaoLimpa = gtk_button_new_with_label ("Limpa");
botaoAssociacao = gtk_button_new_with_label ("Associacoes");
listaRemota = gtk_clist_new_with_titles (4, titles);
consulta = gtk_entry_new ();
endereco = gtk_entry_new ();
porta = gtk_entry_new ();
labelEndereco = gtk_label_new ("Endereco");
labelPorta = gtk_label_new ("Porta");
labelConsulta = gtk_label_new ("Consulta");

//Configurando
gtk_window_set_title (GTK_WINDOW (janelaGeral), "oCToPuS");
gtk_window_set_position (GTK_WINDOW (janelaGeral), GTK_WIN_POS_CENTER);
gtk_window_set_resizable (GTK_WINDOW (janelaGeral), FALSE);
gtk_widget_set_size_request (janelaGeral, 950, 700);
gtk_clist_set_selection_mode (GTK_CLIST (listaRemota), GTK_SELECTION_BROWSE);
gtk_clist_set_shadow_type (GTK_CLIST (listaRemota), GTK_SHADOW_IN);

//Posicionando
gtk_fixed_put (GTK_FIXED (fixed), labelEndereco, 300, 30);
gtk_fixed_put (GTK_FIXED (fixed), labelPorta, 550, 30);

gtk_widget_set_size_request (endereco, 200, 40);
gtk_fixed_put (GTK_FIXED (fixed), endereco, 300, 50);

gtk_widget_set_size_request (porta, 150, 40);
gtk_fixed_put (GTK_FIXED (fixed), porta, 550, 50);

gtk_widget_set_size_request (botaoAssocia, 100, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoAssocia, 800, 50);

gtk_widget_set_size_request (botaoAssociacao, 170, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoAssociacao, 100, 420);

gtk_widget_set_size_request (botaoPing, 170, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoPing, 100, 470);

gtk_widget_set_size_request (botaoAssociaHostCache, 170, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoAssociaHostCache, 100, 520);

gtk_widget_set_size_request (botaoImprimeHostCache, 170, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoImprimeHostCache, 100,
570);

gtk_fixed_put (GTK_FIXED (fixed), labelConsulta, 300, 110);

gtk_widget_set_size_request (consulta, 400, 40);
gtk_fixed_put (GTK_FIXED (fixed), consulta, 300, 130);

gtk_widget_set_size_request (botaoConsulta, 100, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoConsulta, 800, 130);

gtk_widget_set_size_request (botaoDownload, 100, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoDownload, 300, 200);

gtk_widget_set_size_request (botaoLimpa, 100, 40);
gtk_fixed_put (GTK_FIXED (fixed), botaoLimpa, 500, 200);

gtk_widget_set_size_request (listaRemota, 600, 400);
```

```
gtk_fixed_put (GTK_FIXED (fixed), listaRemota, 300, 250);
gtk_clist_set_column_width (GTK_CLIST (listaRemota), 0, 250);
gtk_clist_set_column_width (GTK_CLIST (listaRemota), 1, 140);
gtk_clist_set_column_width (GTK_CLIST (listaRemota), 2, 40);
gtk_clist_set_column_width (GTK_CLIST (listaRemota), 2, 100);

gtk_fixed_put (GTK_FIXED (fixed), logo, 70, 30);
gtk_fixed_put (GTK_FIXED (fixed), letreiro, 50, 155);

//Montando
gtk_container_add (GTK_CONTAINER (janelaGeral), fixed);

//Atribuido os sinais
g_signal_connect (G_OBJECT (janelaGeral), "delete_event",
                  G_CALLBACK (delete_event), NULL);

g_signal_connect (G_OBJECT (botaoAssociacao), "clicked",
                  G_CALLBACK (imprimeAssociacoesAbertas), NULL);

g_signal_connect (G_OBJECT (botaoAssocia), "clicked",
                  G_CALLBACK (associa), (gpointer) endereco);

g_signal_connect (G_OBJECT (botaoConsulta), "clicked",
                  G_CALLBACK (enviaConsulta), (gpointer) consulta);

g_signal_connect (G_OBJECT (botaoDownload), "clicked",
                  G_CALLBACK (enviaDownload), (gpointer) listaRemota);

g_signal_connect (G_OBJECT (listaRemota), "select_row",
                  GTK_SIGNAL_FUNC (selection_made), NULL);

g_signal_connect (G_OBJECT (botaoLimpa), "clicked",
                  G_CALLBACK (limpaLista), (gpointer) listaRemota);

g_signal_connect (G_OBJECT (botaoPing), "clicked",
                  G_CALLBACK (enviaPING), (gpointer) listaRemota);

g_signal_connect (G_OBJECT (botaoImprimeHostCache), "clicked",
                  G_CALLBACK (imprimeHostCache), NULL);

g_signal_connect (G_OBJECT (botaoAssociaHostCache), "clicked",
                  G_CALLBACK (associaHostCache), NULL);

//Mostrando
gtk_widget_show (labelEndereco);
gtk_widget_show (labelPorta);
gtk_widget_show (labelConsulta);
gtk_widget_show (endereco);
gtk_widget_show (porta);
gtk_widget_show (consulta);
gtk_widget_show (listaRemota);
gtk_widget_show (botaoDownload);
gtk_widget_show (botaoAssociacao);
gtk_widget_show (botaoLimpa);
gtk_widget_show (botaoPing);
gtk_widget_show (botaoConsulta);
gtk_widget_show (botaoAssocia);
gtk_widget_show (botaoAssociaHostCache);
gtk_widget_show (botaoImprimeHostCache);
gtk_widget_show (logo);
gtk_widget_show (letreiro);
gtk_widget_show (fixed);
gtk_widget_show (janelaGeral);

gtk_main ();
//Término dos gráficos

return 0;
```

```

}                                     //Fim de fazGraficos

//-----
//Funcao que coloca uma figura na interface grafica
//-----

GtkWidget *
image_new_from_file_at_size (const char *filename, gint width, gint height)
{
    GtkWidget *image = NULL;
    GdkPixbuf *pixbuf;
    GError *error;
    const gchar *st = "icon_size";

    g_return_val_if_fail (filename != NULL, NULL);

    error = NULL;

    pixbuf = gdk_pixbuf_new_from_file (filename, &error);

    if (error != NULL)
    {
        g_warning (G_STRLOC ": cannot open %s: %s", filename,
                    error->message);
        g_error_free (error);
    }

    if (pixbuf != NULL)
    {
        #if DEBUG
        debug (D_STATUS, "pixbuf new from file %s\n", filename);
        #endif
        GdkPixbuf *scaled =
            gdk_pixbuf_scale_simple (pixbuf, width, height,
                                    GDK_INTERP_BILINEAR);

        image = gtk_image_new ();
        gtk_image_set_from_pixbuf (GTK_IMAGE (image), scaled);
        g_object_unref (G_OBJECT (scaled));
        g_object_unref (G_OBJECT (pixbuf));
    }
    else
    {
        #if DEBUG
        debug (D_STATUS, "pixbuf new from file %s failed !\n",
                filename);
        #endif
        GdkIconSize icon_size =
            gtk_icon_size_register (st, width, height);
        image = gtk_image_new_from_stock (GTK_STOCK_MISSING_IMAGE, icon_size);
    }

    return image;
}                                     //Fim da funcao que insere uma figura na interface grafica

//-----
//Fim de octopus.c
//-----

```

```

/*****
    SERVENTE.c - Nucleo do aplicativo Octopus
    Versao 1.0
    Gustavo Salvadori Baptista do Carmo
    27 de Fevereiro de 2006
    gustavocarmo@gmail.com
*****/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/sctp.h>
#include <errno.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <fcntl.h>
#include <signal.h>

//-----
//Estruturas das mensagens Gnutella
//-----

struct CABECALHO
{
    unsigned int DescriptorID;
    unsigned char TTL;
    unsigned char Hops;
};

struct PONG
{
    short porta;
    int endereco;
    int numeroArquivos;
    float numeroKilobytes;
};

struct QUERY
{
    int DescriptorID;
    int TTL;
    int Hops;
    char consulta[100];
};

struct QUERYHIT
{
    int DescriptorID;
    int TTL;
    int Hops;
    struct sockaddr_in endereco;
    long tamanhoArquivo;
    char nomeArquivo[100];
};

struct dadosPedido
{
    struct sockaddr_in endereco;
    int fluxo;
    int sock_fd;
    char nomeArquivo[100];
    int valorVetor;
};

```

```
int tamanhoCabecalho = sizeof (int) + sizeof (int) + sizeof (int);
int tamanhoQUERYHIT = 3 * sizeof (int) + sizeof (struct sockaddr_in) +
    sizeof (long);

//-----
//Variaveis e estruturas para a manipulacao do cache de associacoes
//-----
int numeroDeAssociacoesAbertas = 0;

struct cacheAssociacoes
{
    int numeroSocket;
    struct sockaddr_in endereco;
};

struct cacheAssociacoes associacoesAbertas[20];

//-----
//Variaveis e estruturas que implementam o cache de ID de mensagens
//-----
int indiceListaLoop = 0;

struct cacheIDMensagens
{
    int ID;
    int NumeroSocket;
    struct sockaddr_in endereco;
} listaDeIDs[200];

//-----
//Prototipos das funcoes locais
//-----

//Manipulacao de associacoes
void *abreServidor (struct sockaddr_in *servaddr);
int associaPeer (struct sockaddr_in servaddr);

//Manipulacao de mensagens
void *manipuladorMensagens (int *);
void enviaPING (void);
void enviaPONG (struct sockaddr_in, struct CABECALHO, int);
void enviaQUERY (char *);
void enviaQUERYHIT (char nomeArquivo[], struct QUERY, struct sockaddr_in, int);
void trataPING (struct CABECALHO mensagem, struct sockaddr_in cliaddr);
void trataQUERY (struct QUERY, struct sockaddr_in, int);
void trataQUERYHIT (struct QUERYHIT);
void pedeArquivo (char *, char *, char *, char *);
void *enviaArquivo (struct dadosPedido *);

//Tratamento de loops e backtrack
int trataLoop (int);
void iniciaListaLoop (void);
void incluiListaLoop (int, int, struct sockaddr_in);

//Manipulacao do Host Cache
int existeHostCache (struct sockaddr_in);
int incluiHostCache (struct sockaddr_in);
void imprimeHostCache (void);
void associaHostCache (void);

//Funcoes Miscelania
void imprimeAssociacoesAbertas (void);
int enderecoIgual (struct sockaddr_in, struct sockaddr_in);
int existeEnderecoLista (struct sockaddr_in);
int removeEnderecoLista (int);
int aRespostaMinha (int);
int retornaIndiceLista (int ID);
int retornaIndiceListaEndereco (struct sockaddr_in);
void adicionaResposta (char *texto[]);
```

```

int calculaTamanhoArquivo (char *);
int contaArquivosDiretorio (void);
int tamanhoDiretorio (void);
void retornaNomeArquivoGET (char *, char *);
void retornaSeguinte (char *, int, char *);
int existeSocketLista (int);

//Variaveis Miscelania
int socketNovoParaAssociacoes[20];
int indiceSocketNovoAssociacoes = 0;
pthread_t threadCliente[20];
FILE *hostCacheFile;

//Variaveis do octopus.c
extern struct sockaddr_in meuaddr;
extern GtkWidget *listaRemota;
extern gint linhaListaLocal;

//-----
//Funcao para enviar o arquivo para alguem que pediu
//Envia informacoes sobre o arquivo, em seguida os dados
//-----
void *
enviaArquivo (struct dadosPedido *buffer)
{
    int quantidadeEnviada;
    int out_sz;
    int o = 0;
    char *stringPedido;
    char bufferArquivo[200];
    char nomeArquivoCompleto[100];
    struct dadosPedido dados;
    FILE *arquivo;

    //Preenchendo estrutura local - informacoes da transferencia
    dados.endereco = buffer->endereco;
    strcpy (dados.nomeArquivo, buffer->nomeArquivo);
    dados.sock_fd = buffer->sock_fd;
    dados.fluxo = buffer->fluxo;
    dados.valorVetor = buffer->valorVetor;

    printf ("\nEstou prestes a enviar um arquivo\n");
    printf ("Nome do arquivo:%s\n", dados.nomeArquivo);
    printf ("Numero socket:%d\n", dados.sock_fd);
    printf ("Pedido por:%s\n", inet_ntoa (dados.endereco.sin_addr));
    printf ("Porta:%d\n", ntohs (dados.endereco.sin_port));
    printf ("Fluxo de envio:%d\n\n", dados.fluxo);

    bzero (nomeArquivoCompleto, 100);
    strcat (nomeArquivoCompleto, "./ArquivosCompartilhados/");
    strcat (nomeArquivoCompleto, dados.nomeArquivo);

    stringPedido = malloc (100);
    bzero (stringPedido, 100);

    arquivo = fopen (nomeArquivoCompleto, "rb");

    //Envia mensagem de erro se der problema ao abrir o arquivo
    if (arquivo == NULL)
    {
        printf ("Problema ao abrir o arquivo\n");

        sprintf (stringPedido,
            "OCTOPUS/1.0 003 ERRO\r\nArquivo:%s\r\nMotivo:Arquivo nao encontrado\r\n\r\n",
            dados.nomeArquivo);

        printf ("Enviando mensagem de erro:%s\n", stringPedido);

        out_sz = strlen (stringPedido);
    }
}

```

```

    quantidadeEnviada =
        sctp_sendmsg (dados.sock_fd, stringPedido, out_sz,
                      (struct sockaddr *) &dados.endereco,
                      sizeof (struct sockaddr), 0,
                      MSG_UNORDERED, 5, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
        free (stringPedido);
        memcpy ((void *) dados.valorVetor, &o, sizeof (int));
        pthread_exit (0);
    }

    free (stringPedido);
    memcpy ((void *) dados.valorVetor, &o, sizeof (int));
    pthread_exit (0);
} //Fim de erro de arquivo

//Mensagem de confirmacao que esta tudo ok
sprintf (stringPedido,
"OCTOPUS/1.0 001 OK\r\nArquivo:%s\r\nTamanho:%d\r\n\r\n",
dados.nomeArquivo, calculaTamanhoArquivo (dados.nomeArquivo));

printf ("Mensagem sobre as informacoes:%s\n", stringPedido);

out_sz = strlen (stringPedido);

quantidadeEnviada =
    sctp_sendmsg (dados.sock_fd, stringPedido, out_sz,
                  (struct sockaddr *) &dados.endereco,
                  sizeof (struct sockaddr), 0, 0,
                  dados.fluxo, 0, 0);

if (quantidadeEnviada == (-1))
{
    printf ("Falha ao enviar a mensagem\n");
    free (stringPedido);
    pthread_exit (0);
}

free (stringPedido);

//Tudo ok, enviando o arquivo para o cliente de arquivos
while (!(feof (arquivo)))
{
    bzero (bufferArquivo, 200);
    out_sz = fread (bufferArquivo, sizeof (char), 200, arquivo);

    quantidadeEnviada =
        sctp_sendmsg (dados.sock_fd, &bufferArquivo, out_sz,
                      (struct sockaddr *) &dados.endereco,
                      sizeof (struct sockaddr), 0, 0,
                      dados.fluxo, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
        free (bufferArquivo);
        fclose (arquivo);
        pthread_exit (0);
    }
} //End while de envio do arquivo

fclose (arquivo);

//Liberando o fluxo que foi utilizado para enviar o arquivo
memcpy ((void *) dados.valorVetor, &o, sizeof (int));
pthread_exit (0);

```

```

} //Fim enviaArquivo

//-----
//Funcao para enviar os dados de solicitacao
//do arquivo a ser transferido
//-----
void
pedeArquivo (char *nomeArquivo, char *endereco, char *porta, char *tamanho)
{
    int numeroSocket;
    int quantidadeEnviada;
    int out_sz;
    char *stringPedido;
    struct sockaddr_in enderecoPedido;

    stringPedido = malloc (100);
    bzero (stringPedido, 100);

    sprintf (stringPedido, "GET /get/%s/ OCTOPUS/1.0\r\n\r\n", nomeArquivo);

    printf ("A string de pedido eh:%s\n", stringPedido);

    //Configurando o endereco para quem vai pedir
    bzero (&enderecoPedido, sizeof (enderecoPedido));
    enderecoPedido.sin_family = AF_INET;
    inet_aton (endereco, &enderecoPedido.sin_addr);
    enderecoPedido.sin_port = htons (atoi (porta));

    printf ("\nEstou solicitando um arquivo\n");
    printf ("Nome do arquivo:%s\n", nomeArquivo);
    printf ("Endereco de contato:%s\n", inet_ntoa (enderecoPedido.sin_addr));
    printf ("Porta de contato:%d\n", ntohs (enderecoPedido.sin_port));
    printf ("Tamanho do arquivo:%s\n", tamanho);

    //Tratar aqui a associacao com um no que jah estou associado
    if (existeEnderecoLista (enderecoPedido) == 0)
    {
        associaPeer (enderecoPedido);
        printf
            ("Me associei ao no que estava longe para transferir arquivo\n");
    }

    printf ("Jah estou associado\n");
    numeroSocket =
        associacoesAbertas[retornaIndiceListaEndereco
            (enderecoPedido)].numeroSocket;

    printf ("Numero do socket local do contato:%d\n", numeroSocket);

    out_sz = strlen (stringPedido);

    quantidadeEnviada =
        sctp_sendmsg (numeroSocket, stringPedido, out_sz,
            (struct sockaddr *) &enderecoPedido,
            sizeof (struct sockaddr), 0, MSG_UNORDERED,
            5, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem de pedido de arquivo\n");
    }

    printf ("Quantidade enviada para pedido de arquivo:%d\n",
        quantidadeEnviada);

    free (stringPedido);
    return;
} //fim de pedeArquivo

```

```

//-----
//- Tratamento de mensagens -
//As mensagens que chegam ao no sao recebidas por esta funcao
//-----
void *
manipuladorMensagens (int *socketNumber)
{
    int sock_fd = *socketNumber;
    socklen_t len;
    int msg_flags;
    char *readbuf;
    char *mensagemErro;
    char nomeArquivo[100];
    int out_sz;
    int quantidadeRecebida;
    int quantidadeEnviada;
    int quantidadeEscrita;
    struct sockaddr_in cliaddr;
    struct sctp_sndrcvinfo sri;

    len = sizeof (struct sockaddr_in);
    readbuf = malloc (200);
    mensagemErro = malloc (150);
    printf ("\n0 manipulador de mensagens foi aberto com o socket numero: %d\n",
            sock_fd);

    struct PONG dadosPong;
    struct CABECALHO ping;
    struct QUERY corpo;
    struct QUERYHIT resposta;
    struct sockaddr_in END;

    FILE *arquivo1;
    FILE *arquivo2;
    FILE *arquivo3;

    pthread_t thread_servidor1;
    pthread_t thread_servidor2;
    pthread_t thread_servidor3;

    int vetorEstadosTransferencia[3] = { 0, 0, 0 };
    int vetorEstadosRecebimento[3] = { 0, 0, 0 };
    int fluxo;
    struct dadosPedido bufferEnvio;

    //Aqui e o loop de direcionamento do tratamento das mensagens
    while (1)
    {
        bzero (&sri, sizeof (sri));
        bzero (readbuf, 200);

        quantidadeRecebida = sctp_recvmmsg (sock_fd, readbuf, 200,
                                           (struct sockaddr *)
                                           &cliaddr, &len, &sri,
                                           &msg_flags);

        if (quantidadeRecebida == (-1))
        {
            printf ("Falha ao receber a mensagem - encerrando thread\n");
            removeEnderecoLista (sock_fd);
            if (vetorEstadosRecebimento[0] != 0)
                fclose (arquivo1);
            if (vetorEstadosRecebimento[1] != 0)
                fclose (arquivo2);
            if (vetorEstadosRecebimento[2] != 0)
                fclose (arquivo3);
            free (readbuf);
            pthread_exit (0);
        }
    }
}

```

```
//Tratamento das mensagens PING
if ((sri.sinfo_stream) == 1)
{
    bzero (&ping, sizeof (struct CABECALHO));
    memcpy (&ping, readbuf, quantidadeRecebida);

    //Se nao houve loop
    if (trataLoop (ping.DescriptorID) == 0)
    {
        printf ("\nRecebi um PING\n");
        printf ("Descriptor ID:%d\n", ping.DescriptorID);
        printf ("TTL=%d\n", ping.TTL);
        printf ("Hops=%d\n", ping.Hops);
        printf ("Recebi do endereco:%s\n",
                inet_ntoa (cliaddr.sin_addr));

        incluiListaLoop (ping.DescriptorID, sock_fd,
                        cliaddr);
        enviaPONG (cliaddr, ping, sock_fd);
        trataPING (ping, cliaddr);
    }
    else
        continue; //senao joga fora o descritor
} //fim de PING

//Tratamento das mensagens PONG
if ((sri.sinfo_stream) == 2)
{
    bzero (&ping, sizeof (struct CABECALHO));
    memcpy (&ping, readbuf, sizeof (struct CABECALHO));

    bzero (&dadosPong, sizeof (dadosPong));
    memcpy (&dadosPong,
            (readbuf + sizeof (struct CABECALHO)),
            sizeof (dadosPong));
    bzero (&END, sizeof (struct sockaddr_in));
    memcpy (&END.sin_addr, &dadosPong.endereco,
            sizeof (int));
    END.sin_port = htons (dadosPong.porta);

    //Atualizando host cache
    incluiHostCache (END);

    printf ("\nDados do PONG recebido\n");
    printf ("Quantidade:%d\n", quantidadeRecebida);
    printf ("ID do descritor:%d\n", ping.DescriptorID);
    printf ("Endereco IP do servente:%s\n",
            inet_ntoa (END.sin_addr));
    printf ("Porta do servente:%d\n", dadosPong.porta);
    printf ("Quantidade de arquivos:%d\n",
            dadosPong.numeroArquivos);
    printf ("Quantidade de dados compartilhados:%f\n",
            dadosPong.numeroKilobytes);
} //fim de PONG

//Tratamento das mensagens QUERY
if ((sri.sinfo_stream) == 3)
{
    bzero (&corpo, sizeof (struct QUERY));
    memcpy (&corpo, readbuf, quantidadeRecebida);

    //Se nao houve loop
    if (trataLoop (corpo.DescriptorID) == 0)
    {
        incluiListaLoop (corpo.DescriptorID, sock_fd,
                        cliaddr);
        trataQUERY (corpo, cliaddr, sock_fd);
    }
}
```

```

        else
            continue;                //senao joga fora o descritor
    }                                //fim de QUERY

//Tratamento das mensagens QUERYHIT
if ((sri.sinfo_stream) == 4)
{
    bzero (&resposta, sizeof (resposta));
    memcpy (&resposta, readbuf, quantidadeRecebida);
    trataQUERYHIT (resposta);
}                                //fim de QUERYHIT

//Tratamento da negociacao da transferencia de arquivo
if ((sri.sinfo_stream) == 5)
{
    //Recebi um pedido
    if (strstr (readbuf, "GET /get/") != NULL)
    {
        printf ("0 outro no estah pedindo um arquivo\n");

        bzero (&bufferEnvio, sizeof (struct dadosPedido));

        retornaNomeArquivoGET (readbuf,
                                bufferEnvio.nomeArquivo);
        bufferEnvio.endereco = cliaddr;
        bufferEnvio.sock_fd = sock_fd;

        if (vetorEstadosTransferencia[0] == 0)
        {
            fluxo = 6;
            vetorEstadosTransferencia[0] = 1;
            bufferEnvio.fluxo = fluxo;
            bufferEnvio.valorVetor =
                &vetorEstadosTransferencia[0];
            pthread_create (&thread_servidor1, NULL,
                            (void *) enviaArquivo,
                            &bufferEnvio);
            pthread_detach (thread_servidor1);
        }
        else if (vetorEstadosTransferencia[1] == 0)
        {
            fluxo = 7;
            vetorEstadosTransferencia[1] = 1;
            bufferEnvio.fluxo = fluxo;
            bufferEnvio.valorVetor =
                &vetorEstadosTransferencia[1];
            pthread_create (&thread_servidor2, NULL,
                            (void *) enviaArquivo,
                            &bufferEnvio);
            pthread_detach (thread_servidor2);
        }
        else if (vetorEstadosTransferencia[2] == 0)
        {
            fluxo = 8;
            vetorEstadosTransferencia[2] = 1;
            bufferEnvio.fluxo = fluxo;
            bufferEnvio.valorVetor =
                &vetorEstadosTransferencia[2];
            pthread_create (&thread_servidor3, NULL,
                            (void *) enviaArquivo,
                            &bufferEnvio);
            pthread_detach (thread_servidor3);
        }
        else
        {
            //Envia mensagem de erro - fluxos insuficientes
            retornaNomeArquivoGET (readbuf, nomeArquivo);
            bzero (mensagemErro, 150);
            sprintf (mensagemErro,

```

```

"OCTOPUS/1.0 002 ERRO\r\n/Arquivo:%s\r\n/Tamanho:%d\r\n
Motivo:Todos os fluxos ocupados\r\n\r\n",
nomeArquivo, calculaTamanhoArquivo (nomeArquivo));

printf ("Enviando mensagem de erro:%s\n",
        mensagemErro);

out_sz = strlen (mensagemErro);

quantidadeEnviada =
    sctp_sendmsg (sock_fd, mensagemErro,
                  out_sz,
                  (struct sockaddr *)
                  &cliaddr,
                  sizeof (struct sockaddr),
                  0, MSG_UNORDERED, 5, 0, 0);

if (quantidadeEnviada == (-1))
{
    printf ("Falha ao enviar a mensagem de erro 002\n");
}

printf ("Quantidade enviada como erro:%d\n",
        quantidadeEnviada);
}
//Fim de envio de mensagem de erro
}
//Fim de IF GET /get/
else if ((strstr (readbuf, "OCTOPUS/1.0 002 ERRO") != NULL)
|| (strstr (readbuf, "OCTOPUS/1.0 003 ERRO") != NULL))
{
    printf ("\n%s", readbuf);
    continue;
}
//Fim do recebimento de erros
}
//Fim de tratamento de transferencia de arquivos

//Tratamento dos fluxos usados para transferir arquivos
if ((sri.sinfo_stream) > 5)
{
    //O outro no vai enviar o arquivo
    //Enviou mensagem de OK
    if (strstr (readbuf, "OCTOPUS/1.0 001 OK") != NULL)
    {
        char *nomeArquivo = malloc (100);
        char *tamanhoArquivo = malloc (20);

        retornaSeguinte (readbuf, 1, nomeArquivo);
        retornaSeguinte (readbuf, 2, tamanhoArquivo);

        printf ("Nome do arquivo:%s\n", nomeArquivo);
        printf ("Tamanho do arquivo:%s\n", tamanhoArquivo);

        char nomeArquivoCompleto[100];

        bzero (nomeArquivoCompleto, 100);
        strcat (nomeArquivoCompleto,
                "./ArquivosCompartilhados/");
        strcat (nomeArquivoCompleto, nomeArquivo);
        strcat (nomeArquivoCompleto, ".novo");

        if (sri.sinfo_stream == 6)
        {
            arquivo1 = fopen (nomeArquivoCompleto, "wb");
            vetorEstadosRecebimento[0] =
                atoi (tamanhoArquivo);
        }

        if (arquivo1 == NULL)
        {
            printf ("Deu problema ao abrir o arquivo\n");
            vetorEstadosRecebimento[0] = 0;
        }
    }
}

```

```

    }

    if (sri.sinfo_stream == 7)
    {
        arquivo2 = fopen (nomeArquivoCompleto, "wb");
        vetorEstadosRecebimento[1] =
            atoi (tamanhoArquivo);
    }

    if (arquivo2 == NULL)
    {
        printf ("Deu problema ao abrir o arquivo\n");
        vetorEstadosRecebimento[1] = 0;
    }

    if (sri.sinfo_stream == 8)
    {
        arquivo3 = fopen (nomeArquivoCompleto, "wb");
        vetorEstadosRecebimento[2] =
            atoi (tamanhoArquivo);
    }

    if (arquivo3 == NULL)
    {
        printf ("Deu problema ao abrir o arquivo\n");
        vetorEstadosRecebimento[2] = 0;
    }

    printf ("Arquivo %s aberto\n", nomeArquivoCompleto);
    free (nomeArquivo);
    free (tamanhoArquivo);
} //fim de enviou mensagem de OK
else //Escrevendo os dados no arquivo
{
    printf ("Escrevi\n");
    printf ("Quantidade de bytes que veio:%d\n",
        quantidadeRecebida);
    printf ("Fluxo:%d\n", sri.sinfo_stream);

    if (sri.sinfo_stream == 6)
    {
        quantidadeEscrita =
            fwrite (readbuf, sizeof (char),
                quantidadeRecebida, arquivo1);
        vetorEstadosRecebimento[0] -= quantidadeEscrita;
        if (vetorEstadosRecebimento[0] == 0)
            fclose (arquivo1);
    }

    if (sri.sinfo_stream == 7)
    {
        quantidadeEscrita =
            fwrite (readbuf, sizeof (char),
                quantidadeRecebida, arquivo2);
        vetorEstadosRecebimento[1] -= quantidadeEscrita;
        if (vetorEstadosRecebimento[1] == 0)
            fclose (arquivo2);
    }

    if (sri.sinfo_stream == 8)
    {
        quantidadeEscrita =
            fwrite (readbuf, sizeof (char),
                quantidadeRecebida, arquivo3);
        vetorEstadosRecebimento[2] -= quantidadeEscrita;
        if (vetorEstadosRecebimento[2] == 0)
            fclose (arquivo3);
    }
} //Fim de escrevendo os dados no arquivo
} //Fim de tratamento de fluxos
} //fim do while(1)

```

```

    return 0;
}                                     //fim do manipulador de mensagens

//-----
//Funcao que abre o servidor que aguarda novas associacoes
//-----
void *
abreServidor (struct sockaddr_in *serv)
{
    int sock_fd;
    int erro;

    struct sockaddr_in servaddr;
    struct sockaddr_in cliaddr;
    struct sctp_sndrcvinfo sri;
    struct sctp_event_subscribe evnts;

    bzero (&servaddr, sizeof (servaddr));
    servaddr = *serv;

    printf ("\nAbrindo o Servidor\n");
    printf ("No endereco:%s\n", inet_ntoa (servaddr.sin_addr));
    printf ("Na porta:%d\n", ntohs (servaddr.sin_port));

    //zerando o cache de ID de associacoes
    inicializaListaLoop ();

    //Cria o socket SCTP
    sock_fd = socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);
    if (sock_fd < 0)
    {
        printf ("Criacao do socket falhou\n");
        exit (0);
    }

    //Inscrevendo nos eventos que devem ser notificados
    bzero (&evnts, sizeof (evnts));
    evnts.sctp_data_io_event = 1;

    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS, &evnts,
                       sizeof (evnts));
    if (erro != 0)
    {
        printf ("Erro ao setar eventos\n");
        exit (0);
    }

    //Definindo o tempo maximo de idle time de uma associacao para encerramento automatico
    int close_time = 240;
    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_AUTOCLOSE,
                       &close_time, sizeof (close_time));
    if (erro != 0)
    {
        printf ("Erro ao setar a opcao de close time\n");
        exit (0);
    }

    //Definindo o numero de fluxos que entram e saem, assim como timeouts
    struct sctp_initmsg initmsg;

    initmsg.sinit_num_ostreams = 10;
    initmsg.sinit_max_istreams = 10;
    initmsg.sinit_max_attempts = 3;
    initmsg.sinit_max_init_timeo = 30;

    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                       (char *) &initmsg,
                       sizeof (struct sctp_initmsg));
    if (erro != 0)
    {

```

```
    printf ("Erro ao setar a opcao de initmessage\n");
    exit (0);
}

//Binding
erro = bind (sock_fd, (struct sockaddr *) &servaddr,
             sizeof (servaddr));
if (erro != 0)
{
    printf ("Erro ao dar o bind\n");
    exit (0);
}

erro = listen (sock_fd, 10);
if (erro != 0)
{
    printf ("Erro ao setar a entrar no listen\n");
    exit (0);
}

socklen_t len;
pthread_t thread_servidor[20];
sctp_assoc_t IDassociacao;
int quantidadeEnviada;
int quantidadeRecebida;
int msg_flags;
int i = 0;
int SocketNovo[10];
char *readbuf;

len = sizeof (struct sockaddr_in);
readbuf = malloc (105);

while (1)
{
    bzero (readbuf, 105);
    bzero (&cliaddr, sizeof (cliaddr));

    //recebendo a mensagem do cliente
    //A partir daqui se estabelece uma associacao
    quantidadeRecebida = sctp_rcvmsg (sock_fd, readbuf, 105,
                                     (struct sockaddr *)
                                     &cliaddr, &len, &sri,
                                     &msg_flags);

    if (quantidadeRecebida == (-1))
    {
        printf ("Falha ao receber a mensagem\n");
        continue;
    }

    //Dados para conferencia
    printf ("Informacoes sobre o no requerente:\n");
    printf ("Socket:%d\n", sock_fd);
    printf ("Numero da stream:%d\n", sri.sinfo_stream);
    printf ("Numero de bytes recebidos:%d\n", quantidadeRecebida);
    printf ("Endereco remoto:%s\n", inet_ntoa (cliaddr.sin_addr));
    printf ("Porta remota:%d\n", ntohs (cliaddr.sin_port));
    printf ("Mensagem recebida:%s\n", readbuf);

    //nao ha mais slots de associacao disponiveis
    if (i == 21)
    {
        bzero (readbuf, 105);
        sprintf (readbuf,
                "500 ERROR - NO MORE AVAILABLE SLOTS\n\n");

        quantidadeEnviada = sctp_sendmsg (sock_fd, readbuf,
                                          quantidadeRecebida, (struct sockaddr *) &cliaddr,
                                          sizeof (struct sockaddr), sri.sinfo_ppid, sri.sinfo_flags,
```

```

        sri.sinfo_stream, 0, 0);    //enviando devolta a mensagem

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
    }

    continue;
}    //fim de tratamento de casos de fim de slots disponiveis

//efetivando a associacao com outro no - handshake Gnutella
if (strcmp (readbuf, "GNUTELLA CONNECT/0.4\n\n") == 0)
{
    printf ("GNUTELLA CONNECT/0.4 - Aceito\n");

    //Sim, houve, envia a proxima leg
    bzero (readbuf, 105);
    sprintf (readbuf, "GNUTELLA OK\n\n");

    quantidadeEnviada = sctp_sendmsg (sock_fd, readbuf,
        quantidadeRecebida, (struct sockaddr *) &cliaddr,
        sizeof (struct sockaddr), sri.sinfo_ppid,
        sri.sinfo_flags, sri.sinfo_stream, 0, 0);
    //enviando devolta a mensagem

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
        continue;
    }

    printf ("\nEnviado o GNUTELLA OK\n");
    printf ("Quantidade enviada:%d\n\n", quantidadeEnviada);

    IDassociacao = sri.sinfo_assoc_id;
    SocketNovo[i] = sctp_peeloff (sock_fd, IDassociacao);

    if (SocketNovo[i] == (-1))
    {
        printf ("Falha ao receber no peeloff\n");
        continue;
    }

    printf ("SOCKET NOVO:%d \n", SocketNovo[i]);

    //Atualizando o cache de associacoes com a associacao vindoura
    numeroDeAssociacoesAbertas++;
    associacoesAbertas[numeroDeAssociacoesAbertas].
        numeroSocket = SocketNovo[i];
    associacoesAbertas[numeroDeAssociacoesAbertas].endereco =
        cliaddr;
    printf ("Associacao vindoura ok, e inclusa no cache de associacoes\n\n");

    pthread_create (&thread_servidor[i], NULL,
        (void *) manipuladorMensagens,
        &SocketNovo[i]);
    if (pthread_detach (thread_servidor[i]) == 0)
        printf ("Detach da thread ok\n");
    i++;    //Atualiza numero suportavel de slots de associacao
}

//Fim do if do GNUTELLA CONNECT
//fim do while 1

free (readbuf);
pthread_exit (0);
}    //Fim de abreServidor

//-----
//Funcao para fazer um pedido de associacao a outro no
//Retorna 1 em sucesso e 0 em falha

```

```
//-----
int
associaPeer (struct sockaddr_in servaddr)
{
    printf ("\nTentativa de associar peer\n");
    printf ("O endereco a associar:%s\n",
            inet_ntoa (servaddr.sin_addr));
    printf ("A porta a associar:%d\n\n", ntohs (servaddr.sin_port));

    int sock_fd;
    int erro;
    struct sockaddr_in enderecoEnviante;
    struct sctp_event_subscribe evnts;

    if (existeEnderecoLista (servaddr) == 1)
    {
        printf ("Voce ja possui uma associacao com esse no\n\n");
        return 0;
    }

    if (numeroDeAssociacoesAbertas == 20)
    {
        printf ("Nao e possivel abrir mais associacoes\n");
        return 0;
    }

    //Criando o socket
    sock_fd = socket (AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);
    if (sock_fd < 0)
    {
        printf ("Criacao do socket falhou\n");
        return 0;
    }

    //Inscricao de eventos para receber notificacoes
    bzero (&evnts, sizeof (evnts));
    evnts.sctp_data_io_event = 1;

    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_EVENTS, &evnts,
                      sizeof (evnts));
    if (erro != 0)
    {
        printf ("Erro a setar opcao do socket de initmessage\n");
        return 0;
    }

    //Definindo numero de fluxos a utilizar
    struct sctp_initmsg initmsg;

    initmsg.sinit_num_ostreams = 10;
    initmsg.sinit_max_instreams = 10;
    initmsg.sinit_max_attempts = 3;
    initmsg.sinit_max_init_timeo = 30;

    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_INITMSG,
                      (char *) &initmsg,
                      sizeof (struct sctp_initmsg));
    if (erro != 0)
    {
        printf ("Erro a setar opcao do socket de initmessage\n");
        return 0;
    }

    //Definindo close time automatico
    int close_time = 240;
    erro = setsockopt (sock_fd, IPPROTO_SCTP, SCTP_AUTOCLOSE,
                      &close_time, sizeof (close_time));
    if (erro != 0)
    {
        printf ("Erro ao setar a opcao de close time\n");
        return 0;
    }
}
```

```
}

struct sctp_sndrcvinfo sri;
int quantidadeEnviada;
int quantidadeRecebida;
int msg_flags;
int out_sz;
char *sendline;
socklen_t len;

len = sizeof (struct sockaddr_in);

sendline = malloc (50);

//Definindo o fluxo zero para envio de mensagens Gnutella Connect
bzero (&sri, sizeof (sri));
sri.sinfo_stream = 0;

bzero (sendline, sizeof (50));
sprintf (sendline, "GNUTELLA CONNECT/0.4\n\n");
out_sz = strlen (sendline);

printf ("Enviando o GNUTELLA CONNECT/0.4\n");

quantidadeEnviada = sctp_sendmsg (sock_fd, sendline, out_sz,
                                   (struct sockaddr *) &servaddr,
                                   sizeof (struct sockaddr), 0, 0,
                                   sri.sinfo_stream, 0, 0);

if (quantidadeEnviada == (-1))
{
    printf ("Falha ao enviar a mensagem de GNUTELLA CONNECT\n");
    return 0;
}

bzero (sendline, sizeof (50));

quantidadeRecebida = sctp_rcvmsg (sock_fd, sendline, 50,
                                   (struct sockaddr *)
                                   &enderecoEnviante, &len, &sri,
                                   &msg_flags);

if (quantidadeRecebida == (-1))
{
    printf ("Falha ao receber a mensagem de GNUTELLA OK\n");
    return 0;
}

//aqui eu vou preencher a lista de descritores de sockets
if (strcmp (sendline, "GNUTELLA OK\n\n") == 0)
{
    numeroDeAssociacoesAbertas++;
    associacoesAbertas[numeroDeAssociacoesAbertas].numeroSocket = sock_fd;
    associacoesAbertas[numeroDeAssociacoesAbertas].endereco = servaddr;
    printf ("Recebi o %s\n\n", sendline);
    printf ("Entrando na thread com o socket de numero %d\n", sock_fd);
    socketNovoParaAssociacoes[indiceSocketNovoAssociacoes] = sock_fd;

    pthread_create (&threadCliente[indiceSocketNovoAssociacoes],
                   NULL, (void *) manipuladorMensagens,
                   &socketNovoParaAssociacoes
                   [indiceSocketNovoAssociacoes]);

    if (pthread_detach(threadCliente[indiceSocketNovoAssociacoes]) == 0)
        printf ("Detach da thread ok\n");

    indiceSocketNovoAssociacoes++;
    free (sendline);
    return 1;
}
```

```

else
{
    close (sock_fd);
    printf ("Associacao nao foi aceita\n");
    free (sendline);
    return 0;
}
} //fim de associaPeer

//-----
//Tratamento das mensagens PING
//-----
void
trataPING (struct CABECALHO mensagem, struct sockaddr_in cliaddr)
{
    printf ("\nEstou tratando um PING\n");

    //Esse trecho eh para propagar os PINGS para outros nos
    int out_sz;
    int quantidadeEnviada;
    struct sockaddr_in servaddr;
    int i;

    out_sz = sizeof (mensagem);

    //Descarta o pacote se o TTL esgotou
    if ((mensagem.TTL) == 0)
        return;
    else
    {
        (mensagem.TTL)--;
        (mensagem.Hops)++;
    }

    //para cada associacao que foi feita, envia um QUERY,
    //menos para aquela que enviou
    for (i = 1; i <= numeroDeAssociacoesAbertas; i++)
    {
        servaddr = associacoesAbertas[i].endereco;

        if (enderecoIgual (servaddr, cliaddr) != 1)
        {
            quantidadeEnviada =
                sctp_sendmsg (associacoesAbertas[i].numeroSocket,
                             &mensagem, out_sz,
                             (struct sockaddr *) &servaddr,
                             sizeof (struct sockaddr), 0,
                             MSG_UNORDERED, 1, 0, 0);

            if (quantidadeEnviada == (-1))
            {
                printf ("Falha ao enviar a mensagem\n");
            }

            printf ("Quantidade de bytes de PING enviada:%d\n",
                    quantidadeEnviada);
            printf ("Para o endereco:%s\n",
                    inet_ntoa (servaddr.sin_addr));
        }
        //fim do If
    }
    //fim do for de propagacao de PINGS
}
//Fim trataPING

//-----
//Funcao para o envio de PINGS aos nos diretamente associados
//-----
void
enviaPING (void)

```

```

{
    int out_sz;
    int i;
    int quantidadeEnviada;
    struct sockaddr_in servaddr;
    struct CABECALHO mensagem;

    //Nao pode fazer enviar PING se nao esta associado a ninguem
    if (numeroDeAssociacoesAbertas == 0)
    {
        printf ("Nenhuma associacao efetivada\n");
        return;
    }

    //Preenchendo o payload
    //gerando um numero randomico para o identificador da mensagem
    bzero (&mensagem, sizeof (mensagem));
    srand (time (0));
    mensagem.DescriptorID = rand ();
    mensagem.TTL = 0;
    mensagem.Hops = 0;

    printf ("\nEnviando PING\n");
    printf ("DescriptorID:%d\n", mensagem.DescriptorID);
    printf ("TTL:%d\n", mensagem.TTL);
    printf ("Hops:%d\n", mensagem.Hops);

    //quantidade de dados que vai enviar atraves do socket
    out_sz = sizeof (mensagem);

    //para cada associacao aberta, envia um PING
    for (i = 1; i <= numeroDeAssociacoesAbertas; i++)
    {
        servaddr = associacoesAbertas[i].endereco;
        quantidadeEnviada =
            sctp_sendmsg (associacoesAbertas[i].numeroSocket,
                          &mensagem, out_sz,
                          (struct sockaddr *) &servaddr,
                          sizeof (struct sockaddr), 0,
                          MSG_UNORDERED, 1, 0, 0);

        if (quantidadeEnviada == (-1))
        {
            printf ("Falha ao enviar PING\n");
        }

        printf ("Quantidade Enviada:%d\n", quantidadeEnviada);
        printf ("Para o endereco:%s\n",
                inet_ntoa (servaddr.sin_addr));

        //Para identificacao de quando a mensagem voltar, para exibir os resultados
        incluiListaLoop (mensagem.DescriptorID, 0, meuaddr);
    }
    //fim do for de envio de mensagens PING
}
//Fim do enviaPING

//-----
//Envia uma mensagem de PONG em resposta ao PING
//-----
void
enviaPONG (struct sockaddr_in enderecoCliente,
           struct CABECALHO cabeca, int sock_fd)
{
    int quantidadeEnviada;
    int out_sz;
    struct PONG resposta;
    struct CABECALHO mensagemResp;
    char *buffer;

    buffer = malloc (200);

```

```
bzero (buffer, 200);

//Setar os dados do cabeçalho de resposta
bzero (&mensagemResp, sizeof (mensagemResp));
mensagemResp.DescriptorID = cabeca.DescriptorID;
mensagemResp.TTL = cabeca.Hops + 1;
mensagemResp.Hops = 0;

//Setar os dados da resposta
bzero (&resposta, sizeof (struct PONG));
resposta.porta = (short) ntohs (meuaddr.sin_port);
memcpy (&resposta.endereco, &meuaddr.sin_addr, sizeof (int));
resposta.numeroArquivos = contaArquivosDiretorio ();
resposta.numeroKilobytes = (float) (tamanhoDiretorio () / 1000);

//Passando pro buffer o cabeçalho + resposta
memcpy (buffer, &mensagemResp, sizeof (struct CABECALHO));
memcpy (buffer + sizeof (struct CABECALHO), &resposta,
        sizeof (struct PONG));

printf ("\nEnviando PONG\n");
printf ("0 ID do descritor:%d\n", mensagemResp.DescriptorID);
printf ("0 TTL:%d\n", mensagemResp.TTL);
printf ("0 Hop:%d\n", mensagemResp.Hops);
printf ("A quantidade de arquivos:%d\n", resposta.numeroArquivos);
printf ("Tamanho do diretorio:%f\n", resposta.numeroKilobytes);

out_sz = sizeof (struct CABECALHO) + sizeof (struct PONG);

//Enviar os dados da mensagem PONG
quantidadeEnviada = sctp_sendmsg (sock_fd, buffer, out_sz,
                                  (struct sockaddr *)
                                  &enderecoCliente,
                                  sizeof (struct sockaddr), 0,
                                  MSG_UNORDERED, 2, 0, 0);

if (quantidadeEnviada == (-1))
{
    printf ("Falha ao enviar a mensagem\n");
}

printf ("Quantidade enviada como PONG:%d\n", quantidadeEnviada);

//Enviando os PONGs com os enderecos do host cache
struct sockaddr_in endereco;
hostCacheFile = fopen (". /hostCacheFile.dat", "rb");

if (hostCacheFile == NULL)
{
    printf ("Erro ao abrir o arquivo do Host Cache\n\n");
    free (buffer);
    return;
}

while (!feof (hostCacheFile))
{
    bzero (&endereco, sizeof (endereco));
    fread (&endereco, sizeof (endereco), 1, hostCacheFile);
    if (strcmp (inet_ntoa (endereco.sin_addr), "0.0.0.0") != 0)
    {

        //Setar os dados da resposta
        bzero (&resposta, sizeof (struct PONG));
        resposta.porta = (short) ntohs (endereco.sin_port);
        memcpy (&resposta.endereco, &endereco.sin_addr,
                sizeof (int));
        resposta.numeroArquivos = 0;
        resposta.numeroKilobytes = 0;

        //Passando pro buffer cabeçalho+resposta
        memcpy (buffer + sizeof (struct CABECALHO), &resposta,
```

```

        sizeof (struct PONG));

    printf ("\nEnviando PONG\n");
    printf ("0 ID do descritor:%d\n", mensagemResp.DescriptorID);
    printf ("0 TTL:%d\n", mensagemResp.TTL);
    printf ("0 Hop:%d\n", mensagemResp.Hops);
    printf ("A quantidade de arquivos:%d\n", resposta.numeroArquivos);
    printf ("Tamanho do diretorio:%f\n", resposta.numeroKilobytes);
    printf ("Endereco:%s\n", inet_ntoa (endereco.sin_addr));
    printf ("Porta:%d\n", ntohs (endereco.sin_port));

    quantidadeEnviada =
        sctp_sendmsg (sock_fd, buffer, out_sz,
                      (struct sockaddr *)
                        &enderecoCliente,
                      sizeof (struct sockaddr), 0,
                      MSG_UNORDERED, 2, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
    }

    printf ("Quantidade enviada como PONG:%d\n",
            quantidadeEnviada);
} //Fim do if
//fim do While(!feof)

fclose (hostCacheFile);
free (buffer);
return;
} //Fim de enviaPONG

//-----
//Envia a QUERY solicitada pelo usuario
//-----
void
enviaQUERY (char *consulta)
{
    int out_sz;
    int i;
    int quantidadeEnviada;
    struct sockaddr_in servaddr;
    struct QUERY cabeca;

    //Nao pode fazer consulta se nao esta associado a ninguem
    if (numeroDeAssociacoesAbertas == 0)
    {
        printf ("Nenhuma associacao efetivada\n");
        return;
    }

    //Preenchendo o payload
    //Gerando um numero randomico para o identificador do descritor
    bzero (&cabeca, sizeof (cabeca));
    srand (time (0));
    cabeca.DescriptorID = rand ();
    cabeca.TTL = 4;
    cabeca.Hops = 0;
    strcpy (cabeca.consulta, consulta);

    printf ("Enviando QUERY do usuario\n");
    printf ("DescriptorID:%d\n", cabeca.DescriptorID);
    printf ("TTL:%d\n", cabeca.TTL);
    printf ("Hops:%d\n", cabeca.Hops);
    printf ("Consulta da cabeca %s\n", cabeca.consulta);
    printf ("Consulta sem cabeca %s\n", consulta);

    //quantidade de dados que vai enviar atraves do socket
    out_sz = tamanhoCabecalho + strlen (consulta);

```

[illegible]

```

                                sizeof (struct sockaddr), 0,
                                MSG_UNORDERED, 4, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
    }
    printf ("Quantidade enviada como QUERYHIT:%d\n",
            quantidadeEnviada);
} //Fim de enviaQUERYHIT

//-----
//Funcao que trata as mensagens QUERYHIT
//Ou exibe a resposta na tela ou efetua o backtrack
//-----
void
trataQUERYHIT (struct QUERYHIT resposta)
{
    int indiceLista;
    int out_sz;
    int quantidadeEnviada;
    char endereco[30];
    char porta[10];
    char tamanho[50];
    char *texto[] =
        { resposta.nomeArquivo, endereco, porta, tamanho };

    printf ("\nRECEBI UM QUERYHIT\n");
    printf ("Nome do arquivo que encontrei:%s\n", resposta.nomeArquivo);
    printf ("O ID do descritor que veio:%d\n", resposta.DescriptorID);
    printf ("O TTL da mensagem:%d\n", (resposta.TTL - 1));
    printf ("O Hop do descritor que veio:%d\n", (resposta.Hops + 1));
    printf ("O tamanho do arquivo:%d\n", (int) resposta.tamanhoArquivo);
    printf ("O endereco do sujeito que tem o arquivo:%s\n",
            inet_ntoa (resposta.endereco.sin_addr));
    printf ("A porta do sujeito:%d\n\n",
            ntohs (resposta.endereco.sin_port));

    strcpy (endereco, inet_ntoa (resposta.endereco.sin_addr));
    sprintf (porta, "%d", ntohs (resposta.endereco.sin_port));
    sprintf (tamanho, "%d", (int) resposta.tamanhoArquivo);

    //Caso a resposta seja deste no mesmo
    //Exibe as informacoes na lista da interface grafica
    if (aRespostaMinha (resposta.DescriptorID) == 1)
    {
        adicionaResposta (texto);
        linhaListaLocal = 0;
        printf ("Essa resposta pertence a este no\n");
    }
    else //resposta de outro no
    {
        printf ("A resposta e de outro computador - Backtrack\n");
        indiceLista = retornaIndiceLista (resposta.DescriptorID);
        resposta.TTL--;
        resposta.Hops++;

        out_sz = tamanhoQUERYHIT + strlen (resposta.nomeArquivo);

        //Enviar os dados do backtrack
        quantidadeEnviada =
            sctp_sendmsg (listaDeIDs[indiceLista].NumeroSocket,
                        &resposta, out_sz,
                        (struct sockaddr *)
                        &(listaDeIDs[indiceLista].endereco),
                        sizeof (struct sockaddr), 0,
                        MSG_UNORDERED, 4, 0, 0);

        if (quantidadeEnviada == (-1))
        {

```

[illegible]

```

        sizeof (struct sockaddr), 0,
        MSG_UNORDERED, 3, 0, 0);

    if (quantidadeEnviada == (-1))
    {
        printf ("Falha ao enviar a mensagem\n");
    }

    printf ("Quantidade da QUERY enviada:%d\n",
            quantidadeEnviada);
    printf ("Para o endereco:%s\n",
            inet_ntoa (servaddr.sin_addr));
}
//fim do If
//fim do for de envio de mensagens QUERY
}
//Fim trataQUERY

//-----
//Compara se dois enderecos IP sao iguais
//Retorna 1 se for igual, retorna 0 caso contrario
//-----
int
enderecoIgual (struct sockaddr_in endereco1,
               struct sockaddr_in endereco2)
{
    char endereco1Texto[16];
    char endereco2Texto[16];

    bzero (endereco1Texto, 16);
    bzero (endereco2Texto, 16);

    strcpy (endereco1Texto, inet_ntoa (endereco1.sin_addr));
    strcpy (endereco2Texto, inet_ntoa (endereco2.sin_addr));

    if (strcmp (endereco1Texto, endereco2Texto) == 0)
        return 1;
    else
        return 0;
}
//Fim de enderecoIgual

//-----
//Checa se jah existe uma associacao dessa no cache de associacoes
//Retorna 1 se existe, e 0 caso contrario
//-----
int
existeEnderecoLista (struct sockaddr_in endereco)
{
    int i;

    for (i = 1; i <= numeroDeAssociacoesAbertas; i++)
    {
        if (enderecoIgual (associacoesAbertas[i].endereco, endereco)
            == 1)
            return 1;
    }
    return 0;
}
//fim existeEnderecoLista

//-----
//Checa se jah existe um socket no cache de associacoes
//Retorna 1 se existe, e 0 caso contrario
//-----
int
existeSocketLista (int numSocket)
{
    int i;

    for (i = 1; i <= numeroDeAssociacoesAbertas; i++)
    {

```

```

        if (numSocket == (associacoesAbertas[i].numeroSocket))
            return 1;
    }
    return 0;
} //fim de existeSocketLista

//-----
//Remove um endereco do cache de associacoes
//Se sucesso, retorna 1, senao 0
//-----
int
removeEnderecoLista (int sock_fd)
{
    if (numeroDeAssociacoesAbertas == 0)
        return 0;

    if (numeroDeAssociacoesAbertas == 1)
    {
        numeroDeAssociacoesAbertas--;
        printf ("Removi no do cache de associacoes\n");
        return 1;
    }

    int i;

    for (i = 1; i <= numeroDeAssociacoesAbertas; i++)
    {
        if (sock_fd == (associacoesAbertas[i].numeroSocket))
        {
            if (i == numeroDeAssociacoesAbertas)
            {
                numeroDeAssociacoesAbertas--;
                printf ("Removi no do cache de associacoes\n");
                return 1;
            }
            else
            {
                associacoesAbertas[i] =
                    associacoesAbertas[numeroDeAssociacoesAbertas];
                numeroDeAssociacoesAbertas--;
                printf ("Removi\n");
                return 1;
            }
        }
    } //fim do if
} //fim do for

return 0;
} //fim removeEnderecoLista

//-----
//Imprime as associacoes estabelecidas e ativas
//-----
void
imprimeAssociacoesAbertas (void)
{
    int i;

    printf ("CACHE DE ASSOCIACOES\n");

    for (i = 1; i <= numeroDeAssociacoesAbertas; i++)
    {
        printf ("\nAssociacao numero:%d\n", i);
        printf ("Descritor de socket de numero:%d\n",
            associacoesAbertas[i].numeroSocket);
        printf ("Endereco:%s\n",
            inet_ntoa (associacoesAbertas[i].endereco.sin_addr));
        printf ("Porta:%d\n\n",
            ntohs (associacoesAbertas[i].endereco.sin_port));
    }
}

```

```

    return;
}                                     //fim de imprimeAssociacoesAbertas

//-----
//Iniciacao do cache de ID de mensagens
//-----
void
iniciaListaLoop ()
{
    int i;
    for (i = 0; i <= 200; i++)
    {
        listaDeIDs[i].ID = 0;
    }
}                                     //fim iniciaListaLoop

//-----
//Funcao para conferir se uma mensagem efetuou loop dado o ID
//Retorna 0 caso nao efetuou um loop e 1 caso tenha efetuado
//-----
int
trataLoop (int ID)
{
    int i;

    for (i = 0; i <= 200; i++)
    {
        if (listaDeIDs[i].ID == ID)
            return 1;
    }

    return 0;
}                                     //fim trataLoop

//-----
//Retorna 1 caso conste o ID na lista e ainda tenha saído daqui
//Encerra o processo de backtrack
//-----
int
aRespostaMinha (int ID)
{
    int i;

    for (i = 0; i <= 200; i++)
    {
        if ((listaDeIDs[i].ID == ID)
            && (enderecoIgual (listaDeIDs[i].endereco, meuaddr) ==
                1))
            return 1;
    }

    return 0;
}                                     //fim aRespostaMinha

//-----
//Caso o DescriptorID esteja na lista
//retorna a posicao dele dentro da lista, senao retorna 500
//-----
int
retornaIndiceLista (int ID)
{
    int i;

    for (i = 0; i <= 200; i++)
    {
        if (listaDeIDs[i].ID == ID)
            return i;
    }
}

```

```

    return 500;
}                                     //fim retornaIndiceLista

//-----
//Caso o endereco exista no cache de associacoes
//retorna o indice dele dentro do cache
//-----
int
retornaIndiceListaEndereco (struct sockaddr_in endereco)
{
    int i;

    for (i = 0; i <= 200; i++)
    {
        if (enderecoIgual (endereco, associacoesAbertas[i].endereco)
            == 1)
            return i;
    }

    return 500;
}                                     //fim retornaIndiceListaEndereco

//-----
//Adiciona um numero no cache de ID de associacoes
//-----
void
incluiListaLoop (int ID, int NumeroSocket,
                 struct sockaddr_in endereco)
{
    listaDeIDs[indiceListaLoop].ID = ID;
    listaDeIDs[indiceListaLoop].NumeroSocket = NumeroSocket;
    listaDeIDs[indiceListaLoop].endereco = endereco;
    indiceListaLoop++;
    if (indiceListaLoop == 201)
        indiceListaLoop = 0;
}                                     //fim incluiListaLoop

//-----
//Adiciona as respostas QUERYHIT na interface grafica
//-----
void
adicionaResposta (char *texto[])
{
    gtk_clist_prepend (GTK_CLIST (listaRemota), texto);
    gtk_widget_show (listaRemota);
    return;
}                                     //fim adicionaResposta

//-----
//Calcula o tamanho, em bytes, do repositório de arquivos
//-----
int
tamanhoDiretorio (void)
{
    struct direct **arquivos;
    int count;
    int tamanhoFinal = 0;
    int i;
    char nomeArquivo[100];

    count = scandir ("./ArquivosCompartilhados", &arquivos, NULL,
                    NULL);

    for (i = 2; i < count; i++)
    {
        bzero (nomeArquivo, 100);
        strcpy (nomeArquivo, "");
    }
}

```

```

        strcat (nomeArquivo, arquivos[i]->d_name);
        tamanhoFinal = tamanhoFinal + calculaTamanhoArquivo (nomeArquivo);
    }
    return tamanhoFinal;
}                                     //fim tamanho Diretorio

```

```

//-----
//Checa se um determinado endereco existe no host cache
//Retorna 1 se existe e 0 se naum existe
//-----
int
existeHostCache (struct sockaddr_in enderecoComparar)
{
    struct sockaddr_in endereco;

    hostCacheFile = fopen (".hostCacheFile.dat", "rb");

    if (hostCacheFile == NULL)
    {
        printf ("Erro ao abrir o arquivo do Host Cache\n\n");
        return 0;
    }

    while (!feof (hostCacheFile))
    {
        bzero (&endereco, sizeof (endereco));
        fread (&endereco, sizeof (endereco), 1, hostCacheFile);
        if (enderecoIgual (endereco, enderecoComparar) == 1)
        {
            fclose (hostCacheFile);
            return 1;
        }
    }

    fclose (hostCacheFile);
    return 0;
}                                     //fim de existeHostCache

```

```

//-----
//Inclui um endereco no host cache
//retorna 1 em sucesso ou 0 em falha
//-----
int
incluiHostCache (struct sockaddr_in enderecoIncluir)
{
    struct sockaddr_in endereco = enderecoIncluir;
    if (existeHostCache (endereco) == 1)
        return 0;

    int erro;

    hostCacheFile = fopen (".hostCacheFile.dat", "ab");

    if (hostCacheFile == NULL)
    {
        printf ("Erro ao abrir o arquivo do Host Cache\n\n");
        return 0;
    }

    erro = fwrite (&endereco, sizeof (endereco), 1, hostCacheFile);
    if (erro == 0)
    {
        printf ("Erro ao inserir um arquivo no host cache\n");
        fclose (hostCacheFile);
        return 0;
    }

    fclose (hostCacheFile);
    return 1;
}

```

```

} //fim incluiHostCache

//-----
//Funcao para imprimir os enderecos do Host Cache
//-----
void
imprimeHostCache (void)
{
    struct sockaddr_in endereco;
    hostCacheFile = fopen (".hostCacheFile.dat", "rb");

    if (hostCacheFile == NULL)
    {
        printf ("Erro ao abrir o arquivo do Host Cache\n\n");
        return;
    }

    printf ("\nImprimindo HOST CACHE\n\n");

    while (!feof (hostCacheFile))
    {
        bzero (&endereco, sizeof (endereco));
        fread (&endereco, sizeof (endereco), 1, hostCacheFile);
        if (strcmp (inet_ntoa (endereco.sin_addr), "0.0.0.0") != 0)
        {
            printf ("Endereco:%s\n", inet_ntoa (endereco.sin_addr));
            printf ("Porta:%d\n\n", ntohs (endereco.sin_port));
        }
    }

    fclose (hostCacheFile);
} //fim de imprimeHostCache

//-----
//Funcao para ativar a associacao com todos os enderecos do
//host cache
//-----
void
associaHostCache (void)
{
    int erro;
    struct sockaddr_in endereco;
    hostCacheFile = fopen (".hostCacheFile.dat", "rb");

    if (hostCacheFile == NULL)
    {
        printf ("Erro ao abrir o arquivo do Host Cache\n\n");
        return;
    }

    while (!feof (hostCacheFile))
    {
        bzero (&endereco, sizeof (endereco));
        fread (&endereco, sizeof (endereco), 1, hostCacheFile);

        if (strcmp (inet_ntoa (endereco.sin_addr), "0.0.0.0") != 0)
        {
            printf ("Tentando se associar ao no do host cache:\n");
            printf ("Endereco:%s\n", inet_ntoa (endereco.sin_addr));
            printf ("Porta:%d\n", ntohs (endereco.sin_port));

            erro = associaPeer (endereco);

            if (erro == 0)
                printf ("Erro ao associar a este no\n\n");
            else
                printf ("Sucesso ao associar a este no\n\n");
        }
    } //fim do while feof
}

```

```

    fclose (hostCacheFile);
}                                     //Fim de associaHostCache

//-----
//Calcula o tamanho de um determinado arquivo
//-----
int
calculaTamanhoArquivo (char *nomeArquivo)
{
    char nomeArquivoCompleto[120];
    int tamanhoArquivo = 0;
    int n;

    FILE *arquivo;
    char buffer[200];

    bzero (nomeArquivoCompleto, 120);
    strcat (nomeArquivoCompleto, "../ArquivosCompartilhados/");
    strcat (nomeArquivoCompleto, nomeArquivo);

    arquivo = fopen (nomeArquivoCompleto, "rb");

    if (arquivo == NULL)
    {
        printf ("Deu problema ao abrir o arquivo\n");
        pthread_exit (0);
    }

    while (!(feof (arquivo)))
    {
        bzero (buffer, 200);
        n = fread (buffer, sizeof (char), 200, arquivo);
        tamanhoArquivo = tamanhoArquivo + n;

    }                                     //End while

    fclose (arquivo);
    return tamanhoArquivo;
}                                     //Fim calculaTamanhoArquivo

//-----
//Funcao para separar os tokens do servidor HTTP
//-----
//Retorna o nome do arquivo de pedido
void
retornaNomeArquivoGET (char *entrada, char *saida)
{
    int i = 0;
    int j = 0;
    int barra = 0;

    while (entrada[i] != '\0')
    {
        {
            if (entrada[i] == '/')
            {
                barra++;
                i++;
                continue;
            }

            if (barra == 2)
            {
                saida[j] = entrada[i];
                j++;
            }
            i++;
        }
        saida[j] = '\0';
    }
}

```

```

}                                     //fim de retornaNomeArquivoGET

//Retorna o nome do arquivo da segunda mensagem de negociacao
void
retornaSeguinte (char *entrada, int doisPontos, char *saida)
{
    int i = 0;
    int j = 0;
    int doisPontosInternos = 0;

    while (entrada[i] != '\0')
    {
        if (entrada[i] == ':')
        {
            doisPontosInternos++;
            i++;
        }

        if (doisPontosInternos == doisPontos)
        {
            saida[j] = entrada[i];
            j++;
            if (entrada[i + 1] == '\r')
                break;
        }
        i++;
    }
    saida[j] = '\0';
}                                     //fim de retornaSeguinte

//-----
//Funcao para contar o numero de arquivos num diretorio
//-----

int
contaArquivosDiretorio (void)
{
    struct direct **arquivos;
    int count;
    count = scandir ("./ArquivosCompartilhados", &arquivos, NULL,
                    NULL);
    count = count - 2;
    return count;
}                                     //Fim de contaArquivosDiretorio

//-----
//Fim de SERVENTE.c
//-----

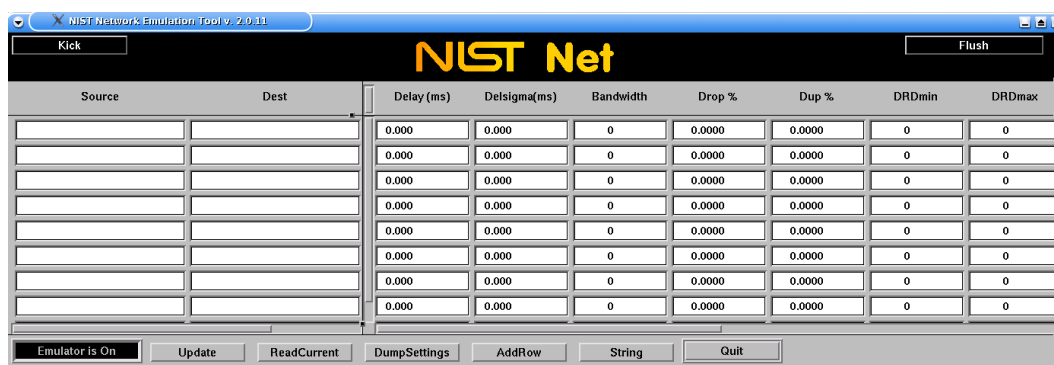
```

Anexo A

A *interface* gráfica do aplicativo NIST Net

O aplicativo NIST Net consiste de duas partes: um módulo de *kernel* carregável, que se conecta ao código de *kernel* que implementa as funções de rede e do relógio do sistema, além de exportar uma série de APIs; um conjunto de *interfaces* que utilizam as APIs para a configuração e o controle das operações do emulador.

Estas *interfaces* podem ser usadas em modo texto, ou em modo gráfico. A *interface* gráfica, como mostrada na figura a seguir, torna simples a configuração dos parâmetros do aplicativo.



Os campos “Source” e “Dest” são configurados com os endereços IP de origem e destino, respectivamente, que identificam os pacotes que devem ser interceptados e tratados pelo aplicativo. Podem ser configurados os parâmetros de atraso e variação de atraso de pacotes, limitação de largura de banda, taxa e variação da taxa de perda, dentre outros.