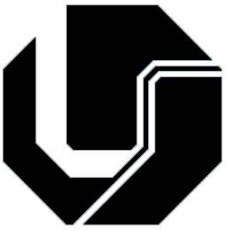
Peer-to-peer com a utilização do SCTP para aplicativos de compartilhamento de arquivos

Gustavo Salvadori Baptista do Carmo

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Ciência da Computação da Universidade Federal de Uberlândia



Programa de Pós–graduação em Ciência da Computação
Faculdade de Computação
Universidade Federal de Uberlândia
Minas Gerais – Brasil

Março de 2006

Gustavo Salvadori Baptista do Carmo

PEER-TO-PEER COM A UTILIZAÇÃO DO SCTP PARA APLICATIVOS DE COMPARTILHAMENTO DE ARQUIVOS

Dissertação apresentada ao programa de Pós-graduação em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de pesquisa: Redes de Computadores.

Orientador: Prof. Dr. Jamil Salem Barbar

UBERLÂNDIA – Março de 2006

Gustavo Salvadori Baptista do Carmo

Peer-to-peer com a utilização do SCTP para aplicativos de compartilhamento de arquivos

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Uberlândia, para obtenção do título de Mestre.

Área de pesquisa: Redes de Computadores.

Banca Examinadora:

Uberlândia, 24 de Março de 2006.

Prof. Dr. Jamil Salem Barbar – UFU (orientador)

Prof. Dr. João Cândido Lima Dovicchi – UFSC

Prof. Dr. Pedro Frosi Rosa – UFU

Dedicado ao

Meu irmão.

Peer-to-peer com a utilização do SCTP para aplicativos de compartilhamento de arquivos

Gustavo Salvadori Baptista do Carmo

Resumo

As redes *peer-to-peer* oferecem uma forma eficaz para o compartilhamento de recursos dos computadores. Sob essas redes está a camada de transporte. Na *Internet*, os protocolos mais utilizados nessa camada são o TCP e o UDP. Outro protocolo da camada de transporte é o SCTP, que oferece funcionalidades adicionais em relação ao TCP e ao UDP que o tornam mais apropriado a determinados tipos de aplicações.

Este trabalho tem como objetivo mostrar a viabilidade da utilização do SCTP como protocolo da camada de transporte em aplicativos *peer-to-peer* de compartilhamento de arquivos, fazendo uso das funcionalidades peculiares a esse protocolo para satisfazer de forma mais eficaz as necessidades destes aplicativos.

Palavras-chave: Redes de computadores, Redes *Peer-to-peer*, SCTP, Camada de transporte.

Gustavo Salvadori Baptista do Carmo

Abstract

Peer-to-peer networks provide an efficient way for sharing computer resources. The transport layer underlies these networks. On the Internet, TCP and UDP are the mostly used protocols in this layer. SCTP is another transport layer protocol: related to TCP and UDP, SCTP offers additional functionalities that makes it more suitable to certain kinds of applications.

This work aims to show the viability of the use of SCTP as the transport layer protocol for peer-to-peer file sharing applications, using the peculiar functionalities of this protocol to satisfy the needs of these applications in a more efficient way.

Keywords: Computer networks, Peer-to-peer Networks, SCTP, Transport Layer.

Copyright

Agradecimentos

Primeiramente a Deus, que me deu esta oportunidade e esteve ao meu lado durante todo o percurso.

Ao meu orientador Prof. Dr. Jamil Salem Barbar, pela dedicação e paciência.

Ao meu pai José Carlos, à minha mãe Bernadete, ao meu irmão Rodrigo, e à minha namorada Graziela, pelo imensurável apoio.

Aos meus amigos do mestrado, pela força, incentivo e, muitas vezes, pelo suporte técnico.

Aos amigos do pensionato e da república, pelos momentos de descontração e companhia.

Aos professores e funcionários do mestrado.

À CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, pelo apoio financeiro concedido.

Sumário

Resumo **v**

Abstract **vi**

Agradecimentos **viii**

Lista de Figuras **xii**

Lista de Tabelas **xv**

Lista de Acrônimos **xvi**

1 Introdução **1**

2 O modelo *peer-to-peer* **6**

2.1 Definição 6

2.2 Aplicabilidade das redes *peer-to-peer* 7

2.3 Características das redes *peer-to-peer* 10

2.4 A classificação das arquiteturas de redes *peer-to-peer* 12

2.4.1 Arquitetura semi-centralizada intermediada 13

2.4.2 Arquitetura semi-centralizada não intermediada 15

2.4.3 Arquitetura descentralizada não estruturada pura 16

2.4.4 Arquitetura descentralizada estruturada 20

2.4.5 Arquitetura descentralizada não estruturada baseada em ultranós 22

2.5 Protocolos para sistemas de compartilhamento de arquivos 23

2.5.1 *Gnutella* 24

2.5.2 *Napster* 37
2.5.3 *Gnutella2* 38
2.5.4 *eDonkey* 39
2.5.5 *FastTrack* 40
2.5.6 *Kademlia* 41

3 Os protocolos da camada de transporte 44
3.1 O protocolo UDP 45
3.2 O protocolo TCP 46
3.3 O protocolo SCTP 51
3.3.1 Histórico do SCTP 51
3.3.2 Características do SCTP 51
3.4 A comparação entre o UDP, o TCP e o SCTP 58
3.4.1 Limitações do UDP 58
3.4.2 Limitações do protocolo TCP 59
3.4.3 SCTP versus TCP versus UDP 60
3.4.4 As similaridades entre o SCTP e o TCP 61
3.4.5 As diferenças entre o SCTP e o TCP 62

4 A utilização do protocolo SCTP por aplicativos de compartilhamento de arquivos em *peer-to-peer* 64
4.1 O comportamento de uma rede *Gnutella* 67
4.1.1 O tráfego gerado na fase de consulta em uma rede *Gnutella* . 68
4.1.2 Os tamanhos das mensagens de uma rede *Gnutella* 72
4.1.3 Discussão a respeito do tempo médio da fase de consulta . . . 73
4.2 O problema de HOL em redes *peer-to-peer* 75
4.2.1 O HOL causado pelo TCP 76
4.2.2 A utilização dos multi-fluxos 78
4.2.3 A utilização da entrega desordenada de mensagens 80
4.3 Análise comparativa entre o SCTP e o TCP para aplicativos de compartilhamento de arquivos em *peer-to-peer* 81
4.3.1 Descrição do ambiente de simulação 81

4.3.2 A simulação da transferência de mensagens numa rede *peer-to-peer* 89
4.3.3 O desempenho dos protocolos TCP e SCTP 91
4.4 A aplicação *peer-to-peer* de compartilhamento de arquivos *Octopus* . 97
4.4.1 O protocolo *Gnutella* com o uso do SCTP 97
4.4.2 O funcionamento do aplicativo *Octopus* 98
4.4.3 A arquitetura do aplicativo *Octopus* 101
4.4.4 A negociação e a transferência de arquivos no *Octopus* 109

5 Conclusão **114**

Referências Bibliográficas **119**

Apêndice A **125**

Apêndice B **150**

Anexo A **187**

Lista de Figuras

2.1 Representação das redes *peer-to-peer* e cliente-servidor 7
2.2 Classificação dos aplicativos *peer-to-peer* sob a perspectiva do gênero de sistema 9
2.3 Classificação dos aplicativos *peer-to-peer* sob a perspectiva de mercado 9
2.4 Arquiteturas de redes *peer-to-peer* 13
2.5 Arquitetura semi-centralizada intermediada 14
2.6 Arquitetura semi-centralizada não intermediada 16
2.7 Arquitetura descentralizada não estruturada pura 17
2.8 Arquitetura descentralizada estruturada 21
2.9 Arquitetura descentralizada não estruturada baseada em ultranós . . 22
2.10 Cabeçalho das mensagens *Gnutella* 26
2.11 A mensagem PING 28
2.12 A mensagem PONG 28
2.13 A mensagem QUERY 29
2.14 A mensagem QUERYHIT 30
2.15 A estrutura de um resultado da QUERYHIT 30
2.16 A mensagem PUSH 31
2.17 Exemplo de Roteamento de PINGs e PONGs 34
2.18 Exemplo de Roteamento de QUERYs, QUERYHITs e PUSHs 35

3.1 Comunicação lógica entre dois processos, em dois *hosts* diferentes . . 45
3.2 Cabeçalho do protocolo UDP 45
3.3 Cabeçalho do protocolo TCP 47
3.4 Iniciação de conexão TCP 48
3.5 Estados de uma conexão TCP 49

3.6 Encerramento de conexão TCP 50
3.7 Formato do pacote SCTP 52
3.8 *Chunk* de dados do SCTP 53
3.9 Iniciação de uma associação SCTP 55
3.10 Estados de uma associação SCTP 56
3.11 Encerramento de conexão SCTP 57
3.12 Característica de *multi-homing* do SCTP 57
3.13 A conexão TCP 62
3.14 A associação SCTP 62

4.1 As fases de funcionamento de um aplicativo *peer-to-peer* para compartilhamento de arquivos 65
4.2 A fase de consulta em uma rede *Gnutella* 68
4.3 Número de QUERYs por segundo em três diferentes monitores 69
4.4 Número de QUERYs em um monitor 70
4.5 Número de QUERYHITs por segundo em três diferentes monitores . . 70
4.6 Número de QUERYHITs em um monitor 71
4.7 Tempo de resposta a uma consulta 74
4.8 Transmissão de mensagens utilizando o TCP 77
4.9 Transmissão de mensagens utilizando os multi-fluxos do SCTP 79
4.10 Transmissão de mensagens utilizando a entrega desordenada do SCTP 80
4.11 A emulação do comportamento esperado de uma rede de computadores por meio do NIST *Net* 82
4.12 As pilhas de protocolos utilizadas pelos aplicativos do ambiente de simulação 83
4.13 As fases de funcionamento dos aplicativos gerador e receptor de pacotes e *echo server* 85
4.14 O funcionamento da *thread* geradora de pacotes 86
4.15 O funcionamento da *thread* receptora de pacotes 87
4.16 A cálculo da latência de uma mensagem - Exemplo 1 87
4.17 O cálculo da latência de uma mensagem - Exemplo 2 88
4.18 O cálculo da latência de uma mensagem - Exemplo 3 89
4.19 Resultados da primeira etapa com taxa de perda de 0% 92

4.20 Resultados da primeira etapa com taxa de perda de 10% 92
4.21 Resultados da primeira etapa com taxa de perda de 20% 93
4.22 Média dos resultados da primeira etapa - variação da taxa de perda de 0% a 20% 94
4.23 Resultados da segunda etapa com taxa de perda de 0% 94
4.24 Resultados da segunda etapa com taxa de perda de 10% 95
4.25 Resultados da segunda etapa com taxa de perda de 20% 95
4.26 Média de resultados da segunda etapa - variação da taxa de perda de 0% a 20% 96
4.27 Novo cabeçalho para o protocolo *Gnutella* 97
4.28 O fluxograma de funcionamento do *Octopus* 99
4.29 Um aplicativo *peer-to-peer* utilizando o TCP na camada de transporte 100
4.30 O aplicativo *Octopus* utilizando o SCTP na camada de transporte . . 101
4.31 A arquitetura do aplicativo *Octopus* 103
4.32 A impressão do *cache* de associações no *shell* do *Linux* 104
4.33 A impressão do *host cache* no *shell* do *Linux* 104
4.34 A utilização dos multi-fluxos pelo aplicativo *Octopus* 106
4.35 A *interface* gráfica do aplicativo *Octopus* 108
4.36 O fluxograma do servidor de arquivos do *Octopus* 111
4.37 O fluxograma do cliente de arquivos do *Octopus* 112

Lista de Tabelas

2.1 Descritores de *payload* do *Gnutella* 26

2.2 Exemplos de protocolos de compartilhamento de arquivos e suas arquiteturas 43

3.1 Os *chunks* do SCTP 54

3.2 Comparação entre o SCTP, o TCP e o UDP 60

4.1 Número de QUERYs por segundo - média em uma hora 70

4.2 Número de QUERYHITs por segundo - média em uma hora 71

4.3 Uma mensagem QUERY na rede *Gnutella* 72

4.4 Uma mensagem QUERYHIT na rede *Gnutella* 72

4.5 Parâmetros dos experimentos da primeira etapa 90

4.6 Parâmetros dos experimentos da segunda etapa 91

Lista de Acrônimos

ADSL - *Asymmetric Digital Subscriber Line*

AIMD - *Additive-increase-multiplicative-decrease*

AOL - *America Online*

API - *Application Programming Interface*

APS - *Adaptive Probabilistic Search*

ASCII - *American Standard Code for Information Interchange*

BFS - *Breadth-first Search*

CAD - *Computer-aided Design*

CAE - *Computer-aided Engineering*

CAN - *Content Addressable Network*

DFS - *Depth-first Search*

DHT - *Distributed Hash Table*

DNS - *Domain Name Server*

DoS - *Denial of Service*

DRLP - *Distributed Resource Location Protocol*

FTP - *File Transfer Protocol*

GPL - *GNU General Public Licence*

GNU - *GNU's not Unix*

HOL - *Head-of-line Blocking*

HTL - *Hops to Live*

HTTP - *Hypertext Transfer Protocol*

ICQ - *"I seek you"*

IETF - *Internet Engineering Task Force*

IP - *Internet Protocol*

IRC - *Internet Relay Chat*

ISO - *International Organization for Standardization*

JXTA - *Juxtapose*

MDTP - *Multi-Network Datagram Transmission Protocol*

MP3 - *MPEG Audio Layer 3*

MSN - *The Microsoft Network*

MSS - *Maximum Segment Size*

MTU - *Maximum Transmission Unit*

NIST - *National Institute of Standards and Technology*

OSI - *Open Systems Interconnection*

PDA - *Personal Digital Assistant*

PPP - *Point-to-point Protocol*

PSNT - *Public Switched Telephone Network*

P2P - *Peer-to-peer*

QoS - *Quality of Service*

RAM - *Random-access Memory*

RFC - *Request for Comments*

RTT - *Round-trip Time*

SACK - *Selective Acknoledgement*

SCTP - *Stream Control Transmission Protocol*

SETI - *Search for Extra-terrestrial Intelligence*

SIGTRAN - *Signaling Transport Working Group*

SMTP - *Simple Mail Transfer Protocol*

TCP - *Transmission Control Protocol*

TSN - *Transmission Sequence Number*

TTL - *Time to Live*

UDP - *User Datagram Protocol*

UTF - *Unicode Transformation Format*

VoIP - *Voice over Internet Protocol*

XML - *Extensible Markup Language*

Capítulo 1

Introdução

Atualmente, os computadores tornaram-se fundamentais como ferramentas de desenvolvimento, e até mesmo de diversão. Permitem a execução de diferentes tipos de aplicativos, tais como CAD (*Computer Aided Design*), CAE (*Computed Aided Engineering*), além daqueles destinados ao entretenimento.

Esses computadores são caracterizados por seus recursos disponíveis, que podem ser físicos ou lógicos, de *hardware* ou de *software*, tais como microfones, *webcams*, impressoras, arquivos, capacidade de processamento e capacidade de armazenamento. Possibilitam a comunicação e o estreitamento das relações humanas por meio de aplicativos que permitem a troca de mensagens e de informações, por exemplo, através de *e-mails* ou VoIP (*Voice over IP*).

Para que os computadores troquem informações entre si, é necessário que estejam, de alguma forma, conectados fisicamente através de uma rede de computadores. As redes de computadores podem ser categorizadas por suas topologias, tais como: topologia estrela, ou centralizada, topologia em barra, ou *Ethernet*, e topologia em anel, ou *Token Ring*, dentre outras. Uma rede é composta por nós, que podem ser qualquer tipo de dispositivo conectado a ela. Os nós podem ser computadores, PDAs (*Personal Digital Assistants*), telefones celulares, ou diversos utensílios com algum tipo de processamento interno. Em uma rede IP (*Internet Protocol*), um nó é qualquer dispositivo que contenha um endereço IP[1].

As redes de computadores podem ser implementadas com o uso de uma variedade

[1]Endereço IP é uma localização lógica, composto por 4 *bytes* ou *16 bytes*, que designam onde um determinado dispositivo se encontra na rede de computadores.

de arquiteturas de pilhas de protocolos e topologias de redes de computadores. A arquitetura de pilha de protocolos é uma abstração da divisão dos diferentes tipos de serviços disponíveis, em um sistema computacional, na forma de camadas.

O protocolo é a descrição formal dos formatos de mensagens e de regras que dois nós obedecem para a troca de informações. Cada mensagem é composta basicamente pelos campos de cabeçalho, corpo e rodapé, onde o corpo acomoda a informação, que por sua vez pode ser tratada também como uma mensagem, a ser encaminhada. O protocolo é baseado em um conjunto de serviços que são normalmente oferecidos à camada superior, ou a uma aplicação, utilizando os serviços das camadas inferiores.

A pilha de protocolos utilizada na *Internet*, denominada TCP/IP (*Transmission Control Protocol/Internet Protocol*), inclui o IP e outros protocolos de camadas superiores que utilizam seus serviços, tais como TCP e UDP (*User Datagram Protocol*) na camada de transporte, e HTTP (*Hyper Text Transfer Protocol*) e FTP (*File Transfer Protocol*) na camada de aplicação.

As redes *peer-to-peer*, ou redes *overlay*, são aquelas que atuam na camada de aplicação e são definidas por um protocolo. Sua finalidade é o compartilhamento dos recursos dos computadores, também denominados nós. Exemplos destes recursos são os discos rígidos para armazenamento de dados, o poder de processamento, a largura de banda ou até mesmo a presença humana. Essas redes são também tipicamente utilizadas para conectar nós de maneira descentralizada, sem a necessidade de um controle central.

A maior diferença, quando se compara o modelo cliente-servidor com o modelo *peer-to-peer*, é que este último possui apenas um elemento atuante, denominado de servente, enquanto que o modelo cliente-servidor possui dois elementos, denominados cliente e servidor. O servente tem as mesmas funções do cliente e do servidor do modelo cliente-servidor.

Clientes são entidades que demandam serviços dos servidores, ao passo que servidores recebem as requisições dos clientes e retornam com os dados ou resultados obtidos. A comunicação entre cliente e servidor é feita por meio de protocolos específicos. Os serventes são entidades que atuam como cliente e servidor ao mesmo tempo para um mesmo propósito [29].

Os nós de uma rede *peer-to-peer* podem apresentar, também, características do

modelo cliente-servidor. Dependendo do grau qualitativo de envolvimento dos nós com este modelo, essas redes podem ser implementadas sob diferentes abordagens, resultando em diferentes arquiteturas, podendo ser classificadas dentro de um espectro que varia de um pólo mais centralizador até um pólo descentralizador.

Um dos objetivos deste trabalho é apresentar uma nova classificação para as diversas arquiteturas de redes *peer-to-peer* existentes [11]. A classificação das arquiteturas é feita a partir do levantamento de suas características, considerando as funcionalidades intrínsecas de cada nó, a organização dos nós, e o algoritmo de roteamento utilizado, definindo, desta maneira, o grau de centralização da arquitetura.

Um dos grandes desafios das redes *peer-to-peer* é garantir o bom desempenho do roteamento de mensagens entre os nós participantes. Uma vez que os recursos estão dinamicamente distribuídos pela rede, não se pode ter noções exatas sobre em que local esses recursos se encontram; assim sendo, cada nó conta com um mecanismo de busca para executar a tarefa de obter as informações referentes à localização desses recursos na rede virtual. Esse mecanismo é implementado pelos protocolos da camada de aplicação do modelo de referência OSI (*Open Systems Interconnection*), que definem os algoritmos de roteamento das mensagens.

As redes *peer-to-peer* atuantes na *Internet* utilizam, sob a camada de aplicação, a pilha de protocolos TCP/IP. A escolha de qual protocolo deve ser utilizado na camada de transporte é um fator determinante para prover e melhorar as funcionalidades específicas a uma determinada aplicação *peer-to-peer*.

Os protocolos da camada de transporte mais utilizados na *Internet* são o UDP e o TCP. O UDP é um protocolo orientado à mensagem, não orientado à conexão e não confiável, isto é, não garante que os dados sejam entregues ao destinatário e nem que cheguem na ordem correta. O UDP oferece os serviços de multiplexação e demultiplexação dos datagramas e carrega em seu cabeçalho uma "soma de verificação", utilizada no controle de integridade dos dados. Já o TCP oferece outros serviços. Este protocolo garante que o segmento será entregue ao destinatário, garante a ordem correta de entrega e ainda implementa os controles de fluxo e de congestionamento. Diferentemente do UDP, o TCP é um protocolo orientado à conexão.

Um outro protocolo da camada de transporte é o SCTP (*Stream Control Trans-*

mission Protocol). O SCTP é um protocolo confiável, orientado à conexão, e orientado à mensagem. Pode ser utilizado concorrentemente com o TCP pois implementa, de forma similar, os controles de fluxo e de congestionamento.

A camada de transporte pode oferecer à camada de aplicação um canal de relacionamento, criado entre dois sistemas finais, que conecta os processos de maneira virtual, isto é, eles não estão conectados fisicamente, mas do ponto de vista da aplicação, é como se estivessem. No TCP, este relacionamento é chamado de conexão, e possui duas vias, uma em cada sentido, através das quais informações podem ser enviadas e recebidas de maneira *full-duplex*. No SCTP, o relacionamento criado entre dois sistemas finais é chamado de associação. Esta denominação, diferentemente do TCP, é utilizada pois um número arbitrário de vias, chamadas de fluxos, pode ser criado para ambos os sentidos.

O SCTP apresenta funcionalidades adicionais, em relação ao TCP e ao UDP, que o tornam mais adequado para determinados tipos de aplicações, como, por exemplo, para as aplicações *peer-to-peer*, que podem fazer uso da funcionalidade que provê múltiplos fluxos para a transmissão de dados, e da entrega desordenada de mensagens.

Assim, este trabalho tem como objetivo principal mostrar a viabilidade da utilização do protocolo SCTP como protocolo da camada de transporte para aplicativos *peer-to-peer* de compartilhamento de arquivos [9, 10]. Primeiramente, é feita uma análise sobre o ganho de desempenho na utilização do SCTP, em vez do TCP, na transferência das mensagens entre os nós de uma rede *peer-to-peer*.

Para efetuar esta análise, um ambiente de simulação é desenvolvido. Este ambiente é composto basicamente por dois nós e um gerador de perda e atraso de pacotes. Os nós representam entidades que fazem parte de uma rede *peer-to-peer* e que trocam mensagens entre si, enquanto que o gerador de perda e atraso de pacotes, implementado por meio do aplicativo NIST *Net* [3], emula o comportamento que uma rede real provoca nas mensagens trafegadas de um nó a outro, em que parâmetros, tais como perda e atraso de pacotes, podem ser configurados.

A partir deste ambiente usa-se o TCP num dado momento, e o SCTP em outro, como protocolos da camada de transporte, em que experimentos práticos são executados baseados na dinâmica de uma rede *peer-to-peer* real em funcionamento. Com

o SCTP, as funcionalidades dos multi-fluxos e da entrega desordenada de mensagens são exploradas, a fim de se efetuar a comparação com o protocolo TCP, que não apresenta essas funcionalidades.

Este trabalho tem, ainda, como objetivo, mostrar um protótipo de aplicação *peer-to-peer* que faz uso das funcionalidades do SCTP. Para isso, foi desenvolvido o aplicativo de compartilhamento de arquivos *Octopus*, que utiliza o SCTP na camada de transporte e o protocolo *Gnutella* reestruturado na camada de aplicação. São descritas as suas camadas e os seus algoritmos, mostrando a forma que este aplicativo faz uso de algumas funcionalidades do protocolo SCTP em um aplicativo *peer-to-peer* de compartilhamento de arquivos.

Esta dissertação, além deste capítulo introdutório, organiza-se como mostrado a seguir.

No Capítulo 2 são apresentadas informações sobre o modelo *peer-to-peer*, tais como definições, taxonomias, arquiteturas e protocolos existentes para aplicativos de compartilhamento de arquivos.

No Capítulo 3, faz-se um levantamento das características dos protocolos da camada de transporte UDP, TCP e SCTP, e, em seguida, é apresentada uma comparação entre eles.

O Capítulo 4 apresenta, inicialmente, um estudo sobre o comportamento de uma rede *peer-to-peer* real em funcionamento. Em seguida, é feita uma discussão sobre a transmissão de mensagens com o uso dos protocolos TCP e SCTP. É descrito, então, o ambiente de simulação utilizado para a análise comparativa entre o TCP e o SCTP. Por fim, apresenta-se os detalhes sobre o desenvolvimento do aplicativo de compartilhamento de arquivos *Octopus*.

O Capítulo 5 encerra o trabalho com as considerações finais, incluindo o levantamento da contribuição deste trabalho à comunidade científica e as propostas para trabalhos futuros.

O Apêndice A apresenta os códigos fonte dos aplicativos utilizados no ambiente de simulação e o Apêndice B apresenta o código fonte do aplicativo *Octopus*. O Anexo A mostra a *interface* gráfica do aplicativo NIST *Net*.

Capítulo 2

O modelo *peer-to-peer*

Este capítulo apresenta as definições, as características e alguns detalhes sobre o modelo *peer-to-peer* e as redes que o implementam. É apresentada também uma nova classificação das diferentes arquiteturas de redes *peer-to-peer*. Em seguida, o protocolo *Gnutella* é mostrado em detalhes, por ser o protocolo de referência para as implementações descritas no Capítulo 4. Por fim, são citadas as características de outros protocolos de redes *peer-to-peer* mais utilizados na *Internet*.

2.1 Definição

O modelo *peer-to-peer* define regras para o desenvolvimento de redes descentralizadas, que funcionam sem a necessidade de uma entidade central. O nó, neste modelo, se comparado ao modelo cliente-servidor, deixa de ser somente um cliente ou servidor e passa a ser cliente e servidor simultaneamente; além disso, em vez de apenas consumir recursos ou oferecer recursos, realiza as duas tarefas simultaneamente [46].

As redes *peer-to-peer*, ou redes *overlay*, implementam o modelo *peer-to-peer*. São redes virtuais que têm como finalidade principal o compartilhamento dos recursos dos nós. Estes recursos podem ser os discos rígidos para armazenamento de arquivos, o poder de processamento, a largura de banda ou, ainda, a presença humana [36].

Na *Internet*, as funcionalidades dessas redes residem na camada de aplicação, atuando, assim, sobre as camadas de rede, implementada pelo protocolo IP, e de transporte, implementada, em sua maioria, pelos protocolos UDP ou TCP, da arquitetura de protocolos TCP/IP [20].

As redes *peer-to-peer* são compostas por nós, também denominados *peers*, que podem estar localizados nas bordas ou no centro da rede, possuindo conectividade variável e temporária. Seus endereços, utilizados para definir a localização de um nó na rede virtual, são também variáveis, pois os nós podem conectar-se ou desconectar-se da rede conforme necessidade ou falha, sendo assim atribuído um novo endereço, quando de sua nova conexão. A Figura 2.1 *(a)* representa uma topologia de rede que implementa o modelo *peer-to-peer*, e a Figura 2.1 *(b)* representa uma topologia de rede que implementa o modelo cliente-servidor.

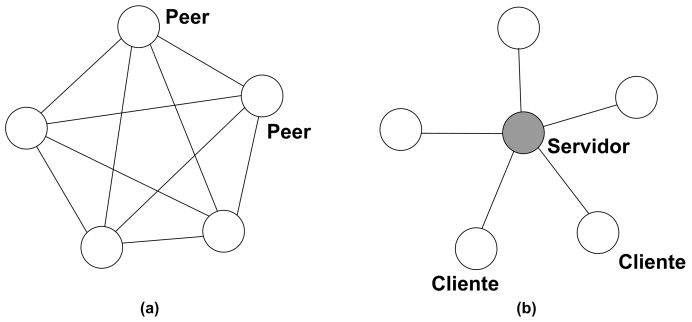


Figura 2.1: Representação das redes *peer-to-peer* e cliente-servidor

Para efetuar algumas de suas tarefas, as redes *peer-to-peer* podem implementar funcionalidades que apresentam características do modelo cliente-servidor. Assim sendo, à medida que diferentes características são incorporadas, diferentes arquiteturas de redes *peer-to-peer* são abordadas [1]. Essas características são definidas pelos protocolos de redes *peer-to-peer* que, além da arquitetura, definem os formatos de mensagens e as regras que dois nós devem obedecer para a troca de informações.

2.2 Aplicabilidade das redes *peer-to-peer*

Os aplicativos *peer-to-peer* podem ser utilizados para diversas finalidades. Assim sendo, há como classificá-los em diferentes sistemas: sistemas de comunicação, sistemas de compartilhamento de arquivos, sistemas colaborativos e sistemas de tarefas distribuídas. As plataformas de desenvolvimento dão suporte à implementação destes sistemas, oferecendo *frameworks*, protocolos e documentações de auxílio ao desenvolvimento [29].

Os sistemas de comunicação são os aplicativos *peer-to-peer* mais populares na *Internet*. Eles permitem que usuários se comuniquem por texto, áudio ou vídeo, e

ainda identifiquem a disponibilidade de outros pares para comunicação. Aplicativos mais elaborados oferecem o serviço de conferência, em que mais de dois usuários podem, simultaneamente, compartilhar uma sessão de comunicação. Exemplos de tais aplicativos têm-se o MSN, o ICQ e o *Skype*.

Os sistemas de compartilhamento de arquivos afetaram com grande impacto a indústria fonográfica, cinematográfica, e todas as outras que trabalham com conteúdos digitalizáveis protegidos por lei. Eles permitem que usuários utilizem e forneçam espaço em disco para o compartilhamento de arquivos de música, áudio, livros digitais ou vídeos, acessíveis a todos os nós. Nessa classe de sistemas, os usuários têm acesso irrestrito ao disco rígido dos nós compartilhantes, que disponibilizam arquivos para *download*. Uma busca deve ser feita de modo a encontrar o arquivo requerido. Exemplos de tais aplicativos têm-se o *Morpheus*, o *Shareaza* e o *Kazaa*.

Os sistemas colaborativos permitem que usuários troquem informações a respeito de uma determinada tarefa comum, em tempo real, sem a necessidade de um servidor central. Aplicações que implementam esta forma de cooperação organizam as informações de maneira inteligente e permitem que pessoas utilizem seus próprios dados e recuperem dados dos demais. Exemplos de tais aplicativos têm-se o *Groove* e o *Microsoft Netmeeting*.

Os sistemas de tarefas distribuídas têm como principal objetivo fazer com que atividades complexas sejam divididas em outras menos complexas e distribuídas pelos nós que compõem a rede, para que sejam executadas em paralelo. Assim que os resultados de cada tarefa menos complexa estiverem disponíveis, são retornados à entidade responsável por combiná-los, para resolver o problema da tarefa mais complexa. Exemplos de tais aplicativos têm-se o SETI@*Home* e o *Condor*.

As plataformas de desenvolvimento oferecem ferramentas para facilitar a construção de sistemas *peer-to-peer*, e especificam não só protocolos para a troca de mensagens, mas também funções que podem ser utilizadas para a elaboração do código-fonte. A utilização destas plataformas para o desenvolvimento dos sistemas *peer-to-peer* é opcional.

Além dos exemplos citados, há muitos outros *softwares* disponíveis na *Internet*, *freeware* ou não, que se enquadram em algum gênero desses sistemas. A Figura 2.2 posiciona alguns destes aplicativos em três eixos [29]: o eixo dos sistemas de

comunicação e colaboração, que possuem funcionalidades similares, o dos sistemas de tarefas distribuídas e o dos sistemas de compartilhamento de arquivos. No ponto de encontro destes três eixos estão representadas as plataformas de desenvolvimento, que podem ser usadas para a implementação de qualquer gênero de sistema. A Figura 2.2 traz como exemplos de plataformas o *.NET* e o *JXTA*.

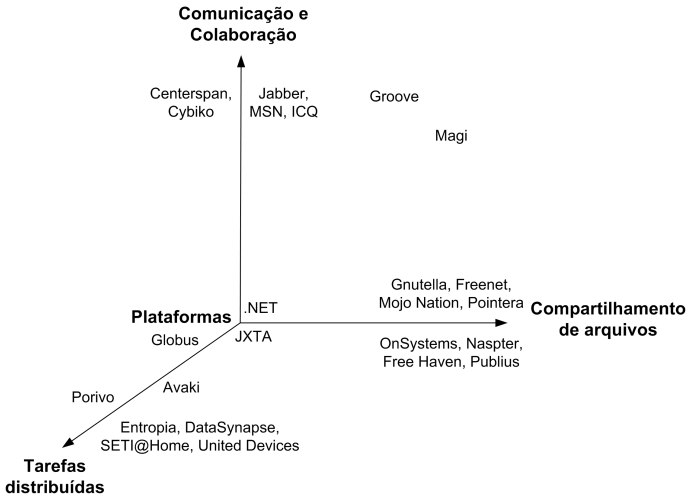


Figura 2.2: Classificação dos aplicativos *peer-to-peer* sob a perspectiva do gênero de sistema

A classificação dos aplicativos *peer-to-peer* pode ser feita, ainda, sob outra perspectiva [29], além daquela apresentada na Figura 2.2. Esta classificação é mostrada na Figura 2.3.

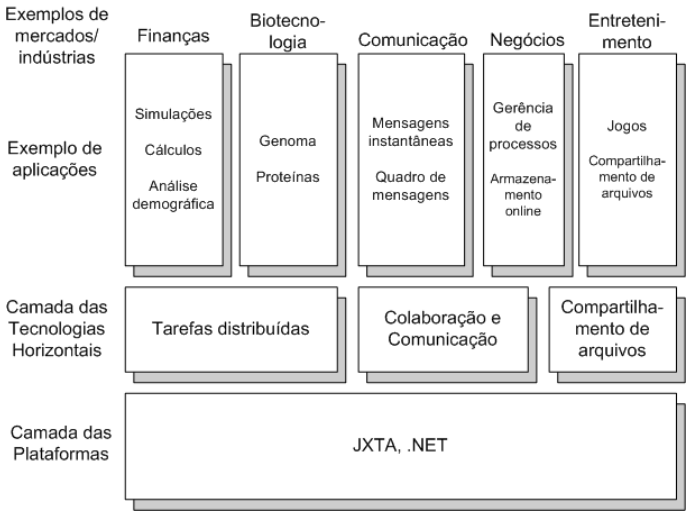


Figura 2.3: Classificação dos aplicativos *peer-to-peer* sob a perspectiva de mercado

Em termos de desenvolvimento, a plataforma JXTA [2] provê infra-estrutura para

a criação de vários sistemas *peer-to-peer*. Logo acima da camada das plataformas, está a camada de aplicações horizontais, que são aquelas que podem ser empregadas em mais de um ramo de mercado, em contraste com as verticais, que são mais específicas.

Logo acima da camada de aplicações horizontais são citadas exemplos de aplicativos que podem ser criadas a partir de cada classe de sistema, e logo acima, as indústrias que fariam uso destes aplicativos. Para exemplificar o diagrama, de acordo com a Figura 2.3, pode-se usar a plataforma JXTA para auxiliar a criação de um sistema de comunicação, tal como o de mensagem instantânea, para uso no mercado de comunicação.

2.3 Características das redes *peer-to-peer*

Os aplicativos *peer-to-peer* causaram impactos estruturais na *Internet*. Um destes impactos refere-se ao consumo da largura de banda. Estes aplicativos requerem mais banda de *upload* do que os aplicativos desenvolvidos sob o modelo cliente-servidor, que vinham sendo os mais utilizados pela grande massa de usuários da *Internet*.

O modelo cliente-servidor é utilizado para implementar redes em que os nós são divididos em dois níveis: um nível contendo os clientes, que demandam recursos, e outro, contendo os servidores, que oferecem recursos. Nesse modelo, um mesmo computador pode ser, ao mesmo tempo, cliente e servidor, porém não para um mesmo propósito ou objetivo. Por sua vez, o modelo *peer-to-peer* implementa redes em que os nós estão em um mesmo nível, que contém os serventes, que atuam simultaneamente como cliente e servidor para um mesmo propósito.

Por exemplo, um computador pode ser servidor FTP, e ser cliente HTTP simultaneamente, não implementando o modelo *peer-to-peer*, pois é cliente e servidor ao mesmo tempo, porém, para propósitos diferentes. No entanto, um computador que atua como cliente, procurando por arquivos e, ao mesmo tempo, oferece os arquivos que possui a outros usuários, atuando assim como servidor, implementa o modelo *peer-to-peer*, pois é cliente e servidor ao mesmo tempo, para um mesmo propósito. Cada um dos modelos apresenta características distintas, como descritas a seguir:

- Tolerância a falha: Quanto maior a descentralização, maior é a tolerância a

falhas. No modelo cliente-servidor, se o servidor falha, o serviço e os recursos oferecidos são gravemente prejudicados. No modelo *peer-to-peer*, a falha de um nó pouco afeta o bem-estar geral da rede, resultando em um ambiente mais tolerante a falhas, ao implementar a redundância dos recursos;

- Segurança: Os usuários das redes *peer-to-peer* deixam de somente consumir recursos passando, também, a disponibilizar recursos. Isso faz com que os nós estejam mais expostos a falhas de segurança do que no modelo cliente-servidor, em que a exposição é mais restrita ao servidor de recursos.

- Redução de custos: Os custos de compra e de manutenção de um servidor de grande porte, que oferece, por exemplo, alto poder de processamento e grande espaço de armazenamento, são muito elevados. O modelo *peer-to-peer* faz com que esses custos sejam distribuídos através dos nós, ao agregar os recursos dos usuários;

- Escalabilidade: As redes implementadas segundo o modelo cliente-servidor possuem escalabilidade limitada pois, caso o número de clientes cresça de maneira descontrolada, um determinado servidor pode vir a não suportar a demanda pelos recursos. No modelo *peer-to-peer*, existe uma melhora em relação à escalabilidade, pois o aumento do número de nós afeta o desempenho do sistema de forma positiva;

- Autonomia e privacidade: A presença de servidores pode ser detectada com maior facilidade do que a de usuários de um sistema *peer-to-peer*. Assim sendo, os usuários podem escolher quais arquivos compartilhar, mesmo que estes arquivos sejam legalmente restritos. Além disso, o usuário escolhe o momento de sua entrada e saída da rede, garantindo, assim, a sua autonomia.

- Gerenciabilidade: O gerenciamento é o planejamento sistematizado da implementação, monitoramento, e utilização dos serviços oferecidos por intermédio de um sistema computacional. Os sistemas baseados no modelo cliente-servidor são gerenciáveis, enquanto os sistemas baseados no modelo *peer-to-peer* são auto-organizáveis. Uma rede é dita auto-organizável quando, enquanto o número de conexões cresce ou diminui dentro da malha de nós, a organização

desses nós se altera de forma independente de uma entidade externa. Essa auto-organização depende de vários fatores tais como interesses comuns dos participantes, estabilidade da rede ou implementação do protocolo da camada de aplicação;

- Dinamismo: Os sistemas baseados no modelo cliente-servidor permitem que as buscas por recursos sejam feitas de forma centralizada. Os sistemas baseados no modelo *peer-to-peer*, devido à sua natureza dinâmica, devem buscar recursos que estão distribuídos na rede e que estão sob constantes modificações;

- Componentes: Diferentes componentes são utilizados em cada um dos modelos. O DNS (*Domain Name Server*), por exemplo, é utilizado como recurso para a descoberta de servidores na rede, substituindo os endereços IP por seqüência de *strings* mais simples para o usuário. No ambiente *peer-to-peer*, outros componentes, tais como o *web caching* e o *host caching*, são utilizados para o descobrimento de nós;

- Protocolos: O HTTP, o SMTP (*Simple Mail Transfer Protocol*), e outros protocolos, são utilizados para implementar os serviços oferecidos pelos sistemas baseados no modelo cliente-servidor. Os sistemas baseados no modelo *peer-to-peer* utilizam protocolos específicos, tais como *Gnutella*, *FastTrack*, entre outros.

As características de um ou de outro modelo podem fundir-se em um mesmo aplicativo, dependendo de como ele é implementado. Essa conjunção de características permite que diferentes arquiteturas de redes *peer-to-peer* sejam desenvolvidas, dependendo do grau qualitativo de envolvimento dos nós em cada um dos modelos *peer-to-peer* e cliente-servidor [29].

2.4 A classificação das arquiteturas de redes *peer-to-peer*

As arquiteturas de redes *peer-to-peer* são divididas em semi-centralizadas e descentralizadas[1]. As arquiteturas semi-centralizadas são subclassificadas em semi-

[1]Essa classificação foi elaborada tendo-se a arquitetura centralizada como implementação do modelo cliente-servidor.

centralizadas intermediadas e semi-centralizadas não intermediadas. As arquiteturas descentralizadas são subclassificadas em descentralizadas estruturadas e descentralizadas não estruturadas.

As arquiteturas descentralizadas não estruturadas são divididas em descentralizadas não estruturadas puras e descentralizadas não estruturadas baseadas em ultranós. Ambas podem usufruir de uma abordagem pura ou guiada [15, 21, 38, 46]. A classificação proposta pode ser melhor representada pela Figura 2.4.

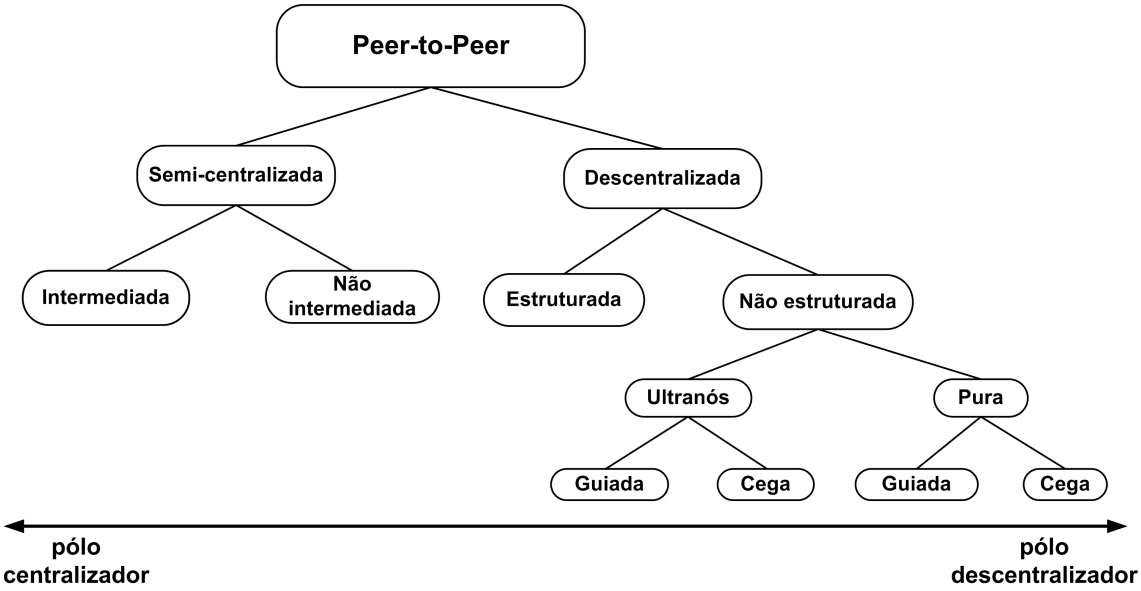


Figura 2.4: Arquiteturas de redes *peer-to-peer*

A Figura 2.4 posiciona as arquiteturas dentro de um pólo que representa as centralizadas, e outro pólo que representa as descentralizadas. Arquiteturas centralizadas são aquelas que dependem de uma entidade central para seu funcionamento. A arquitetura cliente-servidor, por exemplo, é totalmente centralizada, pois depende do funcionamento do servidor, como entidade central, para que suas tarefas sejam executadas e seus serviços oferecidos aos clientes. As arquiteturas descentralizadas, por sua vez, não dependem de uma entidade central e organizam-se de maneira independente de uma única entidade.

2.4.1 Arquitetura semi-centralizada intermediada

A arquitetura semi-centralizada intermediada é a que mais se aproxima, em termos de papéis e responsabilidades dos nós que a compõem, de uma arquitetura

centralizada. Os nós que desejam enviar mensagens, transferir arquivos, ou trocar qualquer fluxo de dados com outros nós devem, antes de iniciar sua comunicação, autenticar-se em uma entidade central, registrando, assim, sua presença.

Toda comunicação vai ser controlada pela entidade central, ou seja, ao enviar dados para o outro nó, estes dados passam por esta entidade central, que os encaminha para o destinatário. A resposta, de maneira análoga, é enviada primeiramente à entidade central, que a encaminha para o nó de origem.

Esta arquitetura possui como vantagem a possibilidade de filtrar pacotes, armazenar informações sobre a troca de dados, tais como horário, datas e textos. No entanto, é uma arquitetura pouco escalável, uma vez que o aumento no número de clientes pode facilmente levar à sobrecarga na entidade central.

A diferença básica desta arquitetura para uma totalmente centralizada, é que os recursos não estão localizados na entidade central, mas nos nós serventes que o cercam. Esta arquitetura pode, ainda, ser utilizada como solução para o compartilhamento de recursos entre dois nós que estão atrás de *firewalls*. A arquitetura semi-centralizada intermediada pode ser vista na Figura 2.5.

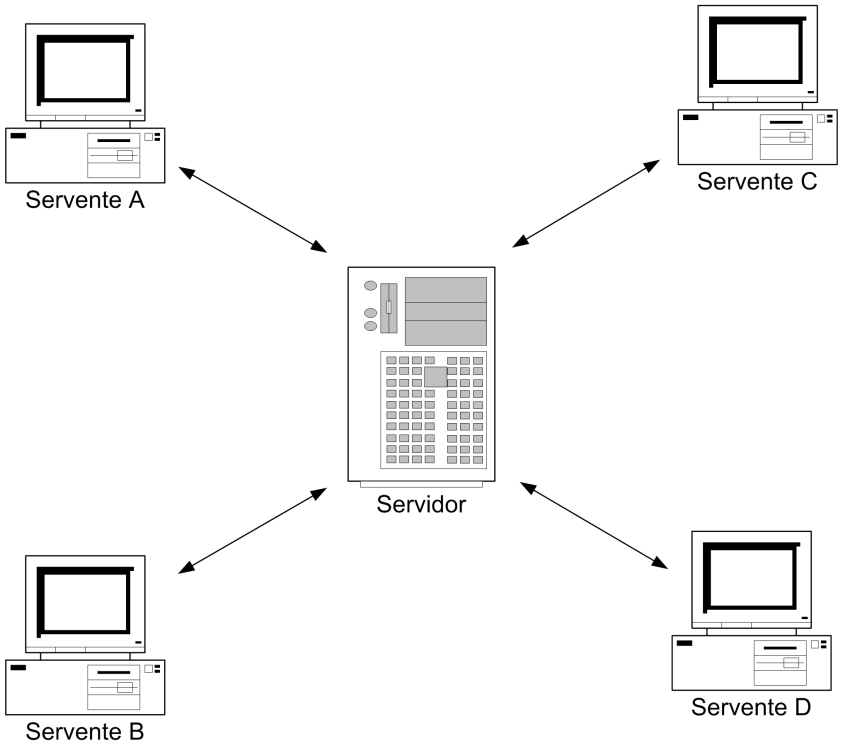


Figura 2.5: Arquitetura semi-centralizada intermediada

A utilização de uma entidade central mostra que o modelo cliente-servidor é utilizado em conjunção com o modelo *peer-to-peer* para executar a tarefa de mediar a comunicação entre dois serventes. Isso faz com que esta arquitetura tenda ao pólo centralizador dentro do espectro de classificação, sob a perspectiva do grau de centralização.

2.4.2 Arquitetura semi-centralizada não intermediada

Essa arquitetura caracteriza-se pela presença de um servidor central, onde são disponibilizadas informações sobre os recursos que os nós compartilham, mais comumente arquivos, além da localização destes recursos na rede [50].

Todas as consultas, que têm como objetivo buscar recursos, são direcionadas ao servidor central que, ao recebê-las, escolhe os nós mais adequados para satisfazê-las. Em aplicativos de compartilhamento de arquivos, por exemplo, o nó requerente envia uma consulta por um determinado arquivo ao servidor central. Assim que este nó recebe a resposta do servidor, ele se conecta diretamente ao nó que oferece o arquivo, para efetuar a transferência, sem a intervenção do servidor central, caracterizando, então, nesta etapa, o modelo *peer-to-peer*.

É necessário, portanto, que os usuários da rede se autentiquem antes de começar a requisitar ou servir arquivos. Como na arquitetura semi-centralizada intermediada, a escalabilidade desta arquitetura é limitada pois, quando o número de nós aumenta, e mais espaço de armazenamento para os índices dos recursos são necessários, exige-se mais carga de trabalho por parte dos servidores centrais. Contudo, a experiência do *Napster* mostrou que essa arquitetura é bastante robusta e eficiente [14, 36]. A Figura 2.6 representa a arquitetura em questão.

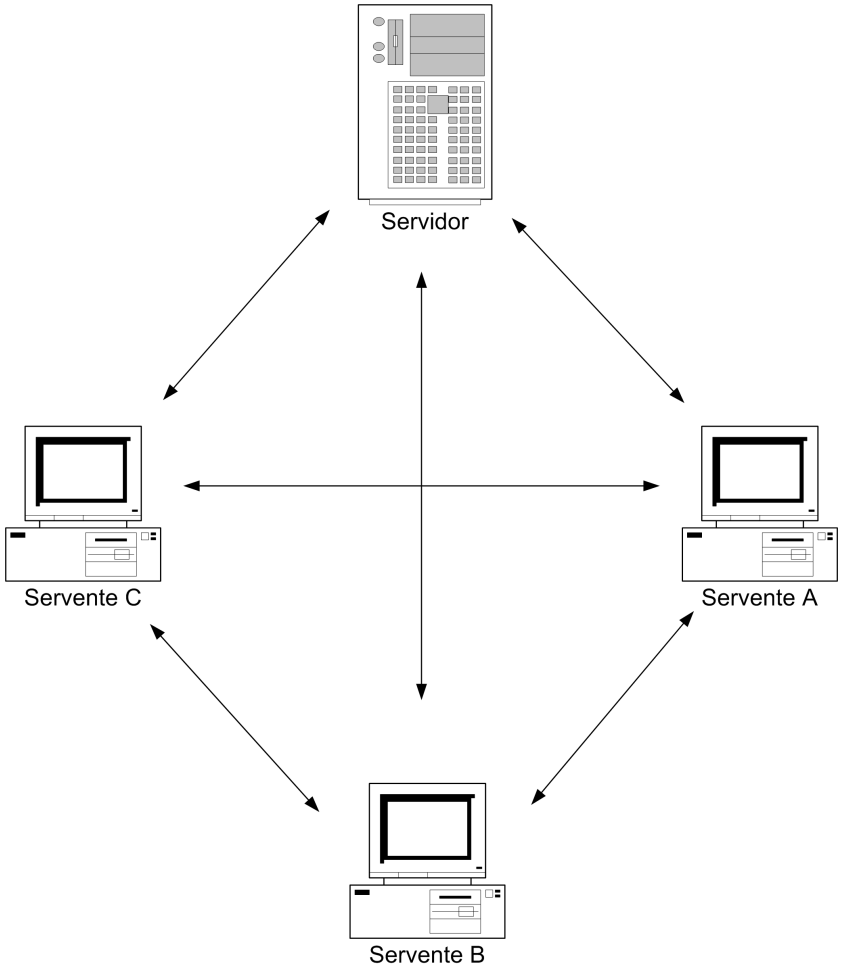


Figura 2.6: Arquitetura semi-centralizada não intermediada

Uma vantagem peculiar dessa arquitetura é a qualidade dos resultados. As respostas são fiéis, pois o servidor central armazena informações de arquivos em nós atualmente participantes na rede. Além disso, caso o servidor não esteja sobrecarregado, a resposta é imediata e não é necessário o envio de muitas mensagens para atingir a um número razoável de respostas. Se o servidor falha, o sistema todo é interrompido. No entanto, esta arquitetura pode ser organizada de várias formas, a fim de melhorar a tolerância a falhas e o desempenho [50].

2.4.3 Arquitetura descentralizada não estruturada pura

A arquitetura descentralizada não estruturada pura é caracterizada por não possuir um servidor central para autenticar usuários e auxiliar na busca de arquivos, como mostrado na Figura 2.7, assim como não apresenta nenhum tipo de estrutura de organização da rede. Com isso, dois problemas devem ser tratados: como encontrar

nós para se conectar à rede e como encontrar recursos, por exemplo, arquivos, na malha de nós.

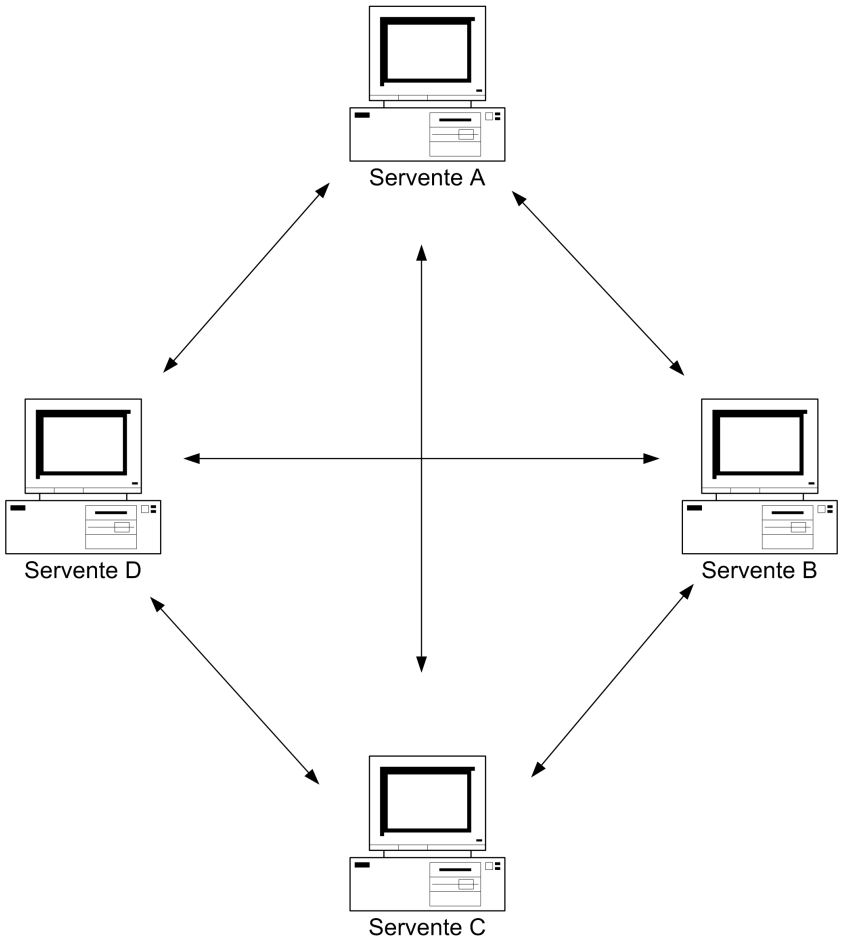


Figura 2.7: Arquitetura descentralizada não estruturada pura

O primeiro problema é causado pela característica dinâmica das redes *peer-to-peer*. Nós se conectam e desconectam de uma maneira não previsível. Com isso, diferentes técnicas devem ser utilizadas para encontrar os nós disponíveis na rede, tais como *web caching* e *pong caching* [17, 18, 44].

Para a solução do segundo problema, pode-se utilizar algoritmos sob a estratégia cega, ou sob a estratégia guiada [48]. A estratégia cega propaga uma consulta para um número suficiente de nós, sem a utilização de informação de consultas prévias. Já a guiada utiliza informações sobre consultas passadas para escolher a melhor rota para as mensagens.

O exemplo tradicional de uma estratégia cega é o algoritmo utilizado pela rede *Gnutella* [17,18]. Esse algoritmo executa inundação baseado em BFS (*Breadth First*

Search), ou seja, cada consulta é enviada a todos os nós diretamente conectados ao nó requerente. O envio de mensagens é feito por cada nó que recebe a consulta, até que ocorra um número máximo de encaminhamentos, definido pelo parâmetro TTL (*Time To Live*), decrementado a cada iteração.

Esse algoritmo apresenta um alto grau de consumo de largura de banda. A partir da boa manipulação do número de conexões de cada nó e da configuração apropriada do valor do parâmetro TTL das mensagens, esta arquitetura pode ser utilizada por centenas de milhares de nós, requerendo alta capacidade dos enlaces de comunicação para proporcionar desempenho razoável, apresentando problemas de escalabilidade, quando o objetivo é alcançar todos os nós em uma rede [35].

O DFS (*Depth First Search*) processa a consulta localmente e, aleatoriamente, escolhe um de seus nós vizinhos para encaminhar a mensagem de busca. Este algoritmo também pode ser referenciado como *chain mode*, em contraste com o *broadcast mode* do BFS. Há, também, a possibilidade de se implementar esse algoritmo sob a abordagem guiada [5].

O BFS modificado [49] é uma variação do algoritmo de inundação, em que os nós escolhem aleatoriamente uma parcela de seus vizinhos para encaminhar as mensagens de requisições. Esse algoritmo reduz a produção média de mensagens em comparação com o BFS tradicional, mas, mesmo assim, causa um alto consumo de largura de banda.

O algoritmo de aprofundamento iterativo [21, 51] usa buscas BFS consecutivas com profundidades cada vez maiores. Alcança melhores resultados quando a condição de término de propagação de consulta está relacionada com o número de resultados definido pelo usuário, sendo possível que poucas iterações satisfaça a consulta. Em casos diferentes deste, o desempenho do algoritmo pode ser ainda pior do que o BFS tradicional.

No algoritmo de caminhos aleatórios [51], o nó requerente envia k mensagens para um número k de nós vizinhos escolhidos aleatoriamente. Cada uma destas mensagens segue seu próprio caminho; os nós intermediários encaminham-as para nós aleatórios a cada iteração. Essas consultas podem também ser chamadas de "andarilhos". Cada "andarilho" pode encerrar seu trajeto a partir de uma falha, ou de um sucesso. A falha pode ser dada por esgotamento de TTL ou pelo método

de checagem, quando o "andarilho" confere com o nó raiz se a sua consulta já foi satisfeita, ou um número definido de respostas já foi atingido.

A grande vantagem desse algoritmo é a redução significativa no número de mensagens propagadas. Ele produz k * TTL[2] mensagens no pior caso, e ainda alcança um bom balanceamento de carga, uma vez que nenhum nó é favorecido no processo de encaminhamento de mensagens. A desvantagem é que apresenta um desempenho variável, uma vez que depende da topologia da rede e das decisões aleatórias feitas pelos "andarilhos".

Alguns exemplos de algoritmos de estratégia guiada são o BFS inteligente, o APS (*Adaptative Probabilistic Search*), o algoritmo baseado em índices locais, baseado em índices de roteamento e o que se fundamenta no protocolo de localização de conteúdo distribuído, ou DRLP (*Distributed Resource Location Protocol*).

O BFS inteligente [49] é um modelo "informado" do BFS modificado, pois armazena informações sobre consultas já efetuadas pelos seus vizinhos, ou através de seus vizinhos. Primeiramente, um nó identifica uma consulta similar àquela que está sendo processada, de acordo com alguma métrica de similaridade. Então o nó encaminha a consulta ao nó que retornou resultados mais "parecidos" com a consulta atual.

No algoritmo APS [47], o nó armazena uma entrada para cada arquivo que ele solicitou a seus vizinhos. O valor desta entrada reflete a probabilidade de um determinado vizinho ser escolhido como próximo salto em uma consulta futura por um arquivo específico. A busca é baseada em k "andarilhos" independentes e com encaminhamento probabilístico. Cada nó intermediário encaminha a consulta para um nó vizinho, com a probabilidade dada por sua entrada. Os valores dos índices são atualizados por *feedback* dos "andarilhos". Se um deles falha, a probabilidade relativa do nó diminui, enquanto se for bem sucedido, a probabilidade relativa do nó aumenta.

No algoritmo de índices locais [51], cada nó indexa todos os arquivos armazenados em nós localizados a um raio de r saltos do nó considerado, e pode responder a consultas em nome destes nós. A consulta é efetuada de uma maneira similar ao

[2]O TTL é definido como o número de saltos, ou *hops*, como normalmente utilizado em redes *peer-to-peer*, e não como um espaço de tempo.

BFS, no entanto, os nós acessíveis ao raio definido podem processar a consulta. Com isso, diminui-se o número de saltos necessários para se atingir um dado nó, reduzindo o número de mensagens propagadas na rede.

O algoritmo de índices de roteamento [7] faz com que os arquivos, ou documentos, sejam divididos em categorias. Cada nó sabe um número aproximado de documentos de cada categoria que podem ser recuperados através de cada conexão. A condição de parada está sempre relacionada com um número definido de respostas. Um nó que não pode satisfazer à condição de parada com seu repositório local encaminha a mensagem para o seu vizinho que possa "melhor" satisfazer à consulta.

Por fim, o algoritmo DRLP [26] estabelece que nós sem informação sobre a localização de um arquivo encaminhem a consulta para vizinhos com alguma probabilidade de resolver essa consulta. Se o arquivo é encontrado, a consulta é revertida para o requerente, armazenando a localização deste documento nos nós do caminho de volta. Em consultas futuras, nós com informações sobre a localização do recurso entram em contato diretamente com o nó indicado, sem encaminhar a mensagem pelos nós intermediários.

De maneira geral, a arquitetura descentralizada não estruturada pura resulta em um sistema extremamente tolerante a falhas, pois, pouco ou nada será modificado referente ao bem-estar da rede caso um nó sofra uma falha. Uma outra característica que essa arquitetura possui é a boa escalabilidade, em que o funcionamento do sistema não se degrada quando mais computadores são adicionados à rede.

Alguns aplicativos *peer-to-peer* implementam essa descentralização pura, permitindo que arquivos legalmente restritos sejam compartilhados e transferidos. Em uma arquitetura descentralizada é mais difícil de se identificar um usuário específico do que um servidor em uma arquitetura mais centralizada, o que dificulta o bloqueio da transmissão de arquivos com conteúdos protegidos e garante privacidade ao usuário, mantendo-o anônimo por não precisar se autenticar em uma entidade central.

2.4.4 Arquitetura descentralizada estruturada

A arquitetura descentralizada estruturada, também referenciada por arquitetura de tabelas de *hash* distribuídas, ou simplesmente DHT (*Distributed Hash Table*), é a

mais recente proposta para a busca de recursos em sistemas *peer-to-peer*. Nesta arquitetura, um número identificador é associado a cada nó da rede, e cada um deles conhece uma determinada quantia de outros nós.

Quando um documento, ou arquivo, é publicado, um número identificador é associado a ele baseado em um *hash* do seu conteúdo, ou do seu nome. Cada nó, então, o encaminha ao nó cujo identificador é mais próximo do identificador do documento, até que se atinja o local mais adequado para o seu armazenamento.

Quando um nó efetua uma consulta, a requisição é transferida até o nó com identificador mais semelhante ao do documento buscado. Esse processo continua até que uma cópia do documento seja encontrada para que seja, então, transferido ao nó que originou a requisição. Cópias desse documento podem ser feitas em cada nó que participou do roteamento, para melhorar o desempenho da busca.

Apesar de ser eficiente para comunidades grandes e globais, a arquitetura baseada em DHT apresenta um problema relacionado ao identificador do documento: este identificador precisa ser conhecido antes que uma consulta seja realizada. Assim sendo, é mais difícil implementar um esquema de busca nesta arquitetura do que nas demais. Outra desvantagem é que, quando um documento é publicado, o algoritmo força a replicação deste documento em um nó específico. Pode não ser do gosto do usuário ter de armazenar esse documento em seu espaço compartilhado. A Figura 2.8 representa esta arquitetura.

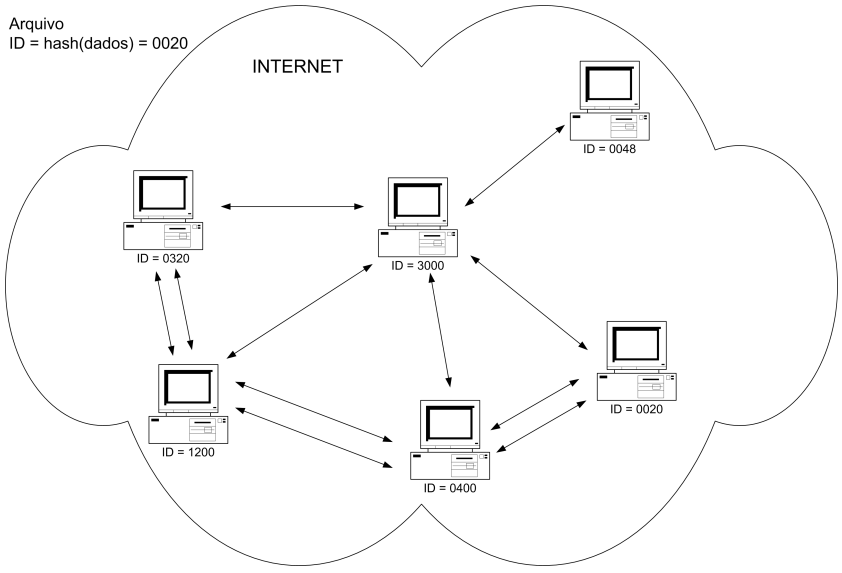


Figura 2.8: Arquitetura descentralizada estruturada

Quatro principais algoritmos foram desenvolvidos para esta arquitetura: *Chord* [43], CAN (*Content Addressable Network*) [34], *Tapestry* [53] e *Pastry* [37]. Os objetivos desses algoritmos são similares: reduzir o número de saltos necessários para encontrar um determinado arquivo e minimizar a quantidade de informação armazenada em cada nó para auxílio de roteamento das mensagens.

2.4.5 Arquitetura descentralizada não estruturada baseada em ultranós

A arquitetura baseada em ultranós [39], ou supernós, ou, ainda, nós *hub*, divide a rede *overlay* em dois níveis: o nível de ultranós, e o nível de nós folha, como apresentado na Figura 2.9.

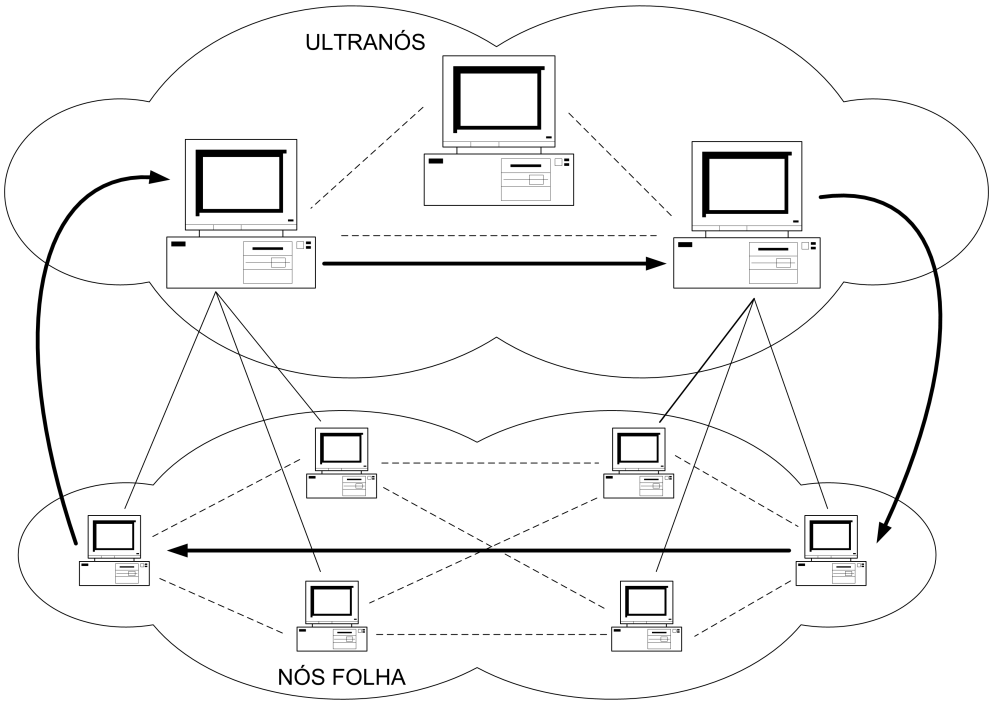


Figura 2.9: Arquitetura descentralizada não estruturada baseada em ultranós

Os ultranós vão ter a responsabilidade de atuar como servidores *proxy* para os demais nós, os nós folha, que estão abaixo na hierarquia da rede *overlay*. Estes ultranós vão "proteger" os nós folha do fluxo de mensagens, evitando, assim, a sobrecarga em nós com enlace, com memória ou com poder de processamento limitados.

A eleição dos nós que devem fazer parte do nível de ultranós é dependente da implementação do protocolo de camada de aplicação utilizado. O tempo de resposta, a média de tempo conectado na rede *overlay*, a capacidade do enlace, o poder de processamento e a quantidade de memória disponível podem ser alguns dos parâmetros

utilizados para a eleição [52].

O algoritmo GUESS [8] foi uma das primeiras propostas de implementação desta arquitetura. Nesse algoritmo uma busca é feita por meio do contato interativo com os diferentes ultranós, não necessariamente vizinhos. Como os ultranós armazenam informações de arquivos compartilhados por nós folha a ele conectados, uma única consulta apresenta alto grau de abrangência. A ordem que os ultranós são contatados não é especificada.

Outro algoritmo que implementa a noção de ultranós é aplicado às redes *Gnutella2* [44]. Quando um ultranó recebe uma consulta, aquele processa localmente a consulta em nome de seus nós folha e a encaminha para seus nós *hub* vizinhos relevantes. Os nós *hub* repetem o processo até que o TTL da mensagem se esgote. O nós *hub* que são vizinhos trocam tabelas de informações para filtrar tráfego desnecessário entre eles.

A grande vantagem do uso desta arquitetura é que, por um lado, ela alcança a robustez da arquitetura cliente-servidor e, por outro, adquire a característica descentralizada, dinâmica e anônima das arquiteturas puramente descentralizadas. A grande desvantagem é a sobrecarga que se pode causar em um nó eleito *hub*. No entanto, com boas políticas de eleição de ultranós e de escolha do número de nós folha permitidos por nó *hub*, pode-se atingir um bom balanceamento de carga [52].

As redes descentralizadas não estruturadas baseadas em ultranós também podem ser implementadas sob diferentes abordagens. A abordagem cega, tal como GUESS, faz consultas aleatoriamente aos ultranós, ou seja, não usa nenhuma informação de consultas passadas para efetuar as suas. Com o uso de informações de consultas prévias, como feito pelo protocolo *Gnutella2*, as mensagens são enviadas a nós que possuem uma maior probabilidade de responder a uma dada consulta, evitando, assim, o envio de mensagens desnecessárias a nós com baixa probabilidade.

2.5 Protocolos para sistemas de compartilhamento de arquivos

Para que os nós em uma rede *overlay* possam comunicar-se, um protocolo deve ser especificado. O protocolo define as regras que governam as sintaxes, semânticas

e a sincronização da troca de informação entre os sistemas finais, e permite a conexão, comunicação e transferência de dados entre os nós na rede. Nesta seção são apresentados alguns dos protocolos mais utilizados na *Internet* em sistemas de compartilhamento de arquivos.

2.5.1 *Gnutella*

O protocolo *Gnutella* utiliza, para a propagação das mensagens, o algoritmo BFS que utiliza a arquitetura descentralizada não estruturada pura e cega. Esse protocolo provê uma *interface* sob a qual os usuários podem executar consultas e ver seus resultados, e ao mesmo tempo permite que os aplicativos respondam às consultas de outros usuários. As versões 0.4 e 0.6 foram definidas e consideradas estáveis, e sua documentação é aberta [17, 18].

Pode-se dizer que o *Gnutella* é um dos primeiros protocolos a ser considerado puramente *peer-to-peer*. Foi desenvolvido pela *NullSoft* e foi disponibilizado sob a GPL (*GNU Public License*) por volta de março de 2000. A AOL (*America Online*) adquiriu a *NullSoft* após o surgimento do *WinAmp* e, devido a problemas com gravadoras a partir da experiência do *Napster*, a AOL suspendeu qualquer desenvolvimento formal do *Gnutella* [31].

No entanto, uma reação ocorreu por parte de usuários da *Internet*, que uniram esforços individuais e corporativos para inovar o *Gnutella* e mantê-lo funcionando. Várias versões do protocolo foram implementadas e há diversos aplicativos que o utilizam em diferentes sistemas operacionais, tais como *Windows*, *Macintosh* e *Linux*.

A versão 0.6 do protocolo adota a eleição de alguns nós potenciais para fazer função de ultranós, ou nós *hub*, atuando como *proxy* para outros nós, chamados de nós folha, com o objetivo de diminuir o tráfego de mensagens de consulta entre nós com deficiência em termos de largura de banda, processamento e memória. Dessa maneira, o balanceamento de carga faz com que a maior parte do trabalho seja feita por nós com mais capacidade.

O protocolo *Gnutella* define a maneira pela qual os serventes se comunicam através da rede. Ele consiste de mensagens, ou descritores, e um conjunto de regras que governam a troca de informação entre os nós, ou serventes. A versão 0.4 já

define as cinco principais mensagens:

- PING: Utilizado para descobrir nós *Gnutella* na rede. Um servente que recebe uma mensagem do tipo PING deve responder com uma ou mais mensagens do tipo PONG;
- PONG: É a mensagem de resposta ao PING. Inclui o endereço de um servente *Gnutella* e informações sobre a quantidade de dados que ele disponibiliza à rede;
- QUERY: É a mensagem que implementa o mecanismo de busca na rede. Um servente que recebe a mensagem do tipo QUERY deve responder com uma mensagem do tipo QUERYHIT, caso o arquivo procurado seja encontrado;
- QUERYHIT: Utilizada como resposta à mensagem do tipo QUERY. Esta mensagem provê informações suficientes para que o arquivo procurado possa ser recuperado pelo servente que originou a consulta;
- PUSH: Esta mensagem implementa o mecanismo que permite que serventes atrás de *firewalls* contribuam com seus arquivos aos demais nós.

Um nó se conecta à rede *Gnutella* ao estabelecer uma conexão com outro já participante. A maneira pela qual os endereços de outros serventes são adquiridos não faz parte da definição do protocolo.

Estabelecimento de conexão e negociação do protocolo

Uma vez que o endereço IP de outro servente é obtido, uma conexão TCP é criada, e a seguinte *string* ASCII (*American Standard Code for Information Interchange*) deve ser enviada:

```
GNUTELLA CONNECT\<versão do protocolo>\n\n
```

O indicador de "versão do protocolo" é definido como a *string* (ASCII) "0.4", nessa versão, seguidos de dois caracteres *line-feed* (0xA). Um servente que deseja aceitar a conexão deve responder com a seguinte mensagem:

```
GNUTELLA OK\n\n
```

Qualquer outra resposta indica que o servente não deseja efetivar a conexão. Vários motivos podem justificar esta situação, entre eles o número de conexões máximo permitido foi atingido, ou a versão do protocolo que não é suportada.

As mensagens *Gnutella*

Uma vez que dois serventes obtiveram sucesso na conexão, eles se comunicam enviando e recebendo mensagens *Gnutella*. Cada mensagem é precedida por um cabeçalho com o formato mostrado pela Figura 2.10[3] [4].

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

Figura 2.10: Cabeçalho das mensagens *Gnutella*

- ID da mensagem: *String* de 16 *bytes* que identifica unicamente a mensagem na rede. Este valor deve ser preservado ao se encaminhar as mensagens entre os serventes. Permite a detecção de ciclos e auxilia na redução de tráfego desnecessário;
- Descritor de *payload*: Indica o tipo de carga que a mensagem carrega. Os valores são definidos como mostra a Tabela 2.1:

Tabela 2.1: Descritores de *payload* do *Gnutella*

0x00	PING
0x01	PONG
0x80	QUERY
0x81	QUERYHIT
0x40	PUSH

[3]Todas as estruturas são utilizadas com ordenação de *bytes little-endian*, a não ser nas exceções especificadas.

[4]Todos os endereços IP das estruturas são do formato IPv4, definidos como *big-endian*.

Este protocolo não define meios para o rastreamento de padrões dentro da *string* recebida, e como os dados são transferidos através de uma conexão TCP, que é um fluxo contínuo de *bytes*, este campo deve ser rigorosamente validado para manter o sincronismo. Uma desincronização pode ser detectada pela presença de um descritor de *payload* inválido;

- TTL (*Time to live*): Indica o número máximo de encaminhamentos da mensagem permitido até que seja retirada da rede. Cada servente deve decrementar o TTL antes de a transferir para outro servente. Quando o TTL alcança o valor zero, a mensagem não pode mais ser encaminhada. Os serventes devem cuidadosamente averiguar o campo TTL e atualizá-lo adequadamente. Os abusos na utilização deste campo podem levar ao excesso de tráfego na rede;

- *Hops*: Indica o número de vezes que uma mensagem foi encaminhada. Enquanto as mensagens são encaminhadas de servente a servente através da rede, os valores de TTL e *Hops* devem satisfazer os seguintes critérios:

  - $TTL_{(i)} + Hops_{(i)} = TTL_{(0)}$;
  - $TTL_{(i+1)} < TTL_{(i)}$;
  - $Hops_{(i+1)} > Hops_{(i)}$.

  Os valores de $TTL_{(i)}$ e $Hops_{(i)}$ são os valores dos campos TTL e *Hops* no cabeçalho das mensagens no i-ésimo encaminhamento, para $i >= 0$;

- Tamanho do *payload*: Representa o tamanho da mensagem imediatamente após o cabeçalho. O cabeçalho da próxima mensagem é encontrado exatamente após o número de *bytes* indicado por este campo.

Imediatamente após o cabeçalho das mensagens há o *payload* específico, cujo conteúdo e estrutura dependem do valor do "Descritor de *payload*". Esses descritores de *payload* e suas respectivas estruturas são apresentados a seguir.

PING

As mensagens do tipo PING não costumam possuir nenhum *payload* adicional e apresentam tamanho nulo, sendo simplesmente um cabeçalho com valor 0x00 no

campo de "Descritor de *payload*". No entanto, caso houver dados após o cabeçalho, o seu *payload* pode ser como mostrado na Figura 2.11.

Campos	Dados opcionais
Byte Offset	0...L-1

Figura 2.11: A mensagem PING

Um servente utiliza a mensagem PING para conseguir dados de outros nós disponíveis na rede virtual. Ao receber esta mensagem, o servente pode responder com uma mensagem PONG, que contém o endereço de um nó *Gnutella* participante, podendo ser ele próprio, enviando também os dados sobre os arquivos que compartilha na rede. O campo "Dados opcionais" consiste de um número variável de *bytes* que são reservados para futuras extensões do protocolo.

Não há a necessidade de se encaminhar mensagens PING para outros serventes, ou, ainda, encaminhá-los com altos valores de TTL e *Hops*. A maioria das implementações incluem uma política para limitar o tráfego de PINGs.

PONG

O *payload* da mensagem de tipo PONG é como mostrado na Figura 2.12.

Campos	Porta	Endereço IP	Número de arquivos compartilhados	*Kbytes* compartilhados	Dados opcionais
Byte Offset	0...1	2...5	6...9	10...13	14...L-1

Figura 2.12: A mensagem PONG

- Porta: Indica a porta TCP através da qual o nó pode receber conexões *Gnutella*;
- Endereço IP: Indica o endereço IP do *host*;
- Número de arquivos compartilhados: Representa o número de arquivos que o *host* disponibiliza na rede;

- *Kbytes* compartilhados: Representa a quantidade, medida em *Kbytes*, de dados que o *host* disponibiliza na rede;

- Dados opcionais: Um campo opcional, de tamanho variável, reservado para extensões do protocolo.

As mensagens do tipo PONG são somente enviadas em resposta a uma mensagem PING. Múltiplas PONGs podem ser enviadas, permitindo, assim, que dados em *cache* sobre outros serventes sejam transmitidos através dos nós.

QUERY

A mensagem do tipo QUERY possui o *payload* representado pela Figura 2.13.

Campos	Velocidade mínima	*String* de busca	NULL	Dados opcionais
Byte Offset	0...1	2...N	N+1	N+2...L-1

Figura 2.13: A mensagem QUERY

- Velocidade mínima: A velocidade mínima, em *kbits*/segundo, de serventes que respondem à essa mensagem. Um servente que recebe uma mensagem QUERY deve responder com uma QUERYHIT somente se for possível comunicar-se a uma velocidade maior ou igual à indicada.

- *String* de busca: A *string* de busca é encerrada por um NULL, sendo o tamanho máximo restringido pelo campo "Tamanho do *payload*" presente no cabeçalho. Deve-se usar uma codificação compatível com ASCII. Nenhuma delas foi especificada, mas a maioria dos serventes utilizam a ISO-8859-1 ou a UTF-8. Essa consulta deve consistir de uma lista de palavras separadas por um espaço ASCII (0x20=32) que podem, opcionalmente, carregar a extensão do formato do arquivo buscado, depois do caracter "ponto" em ASCII (0x2e=46);

- NULL: O campo que encerra a *string* de busca;

- Dados de consulta opcionais: Este campo é opcional e de tamanho variável, e é reservado para extensões do protocolo. Alguns serventes utilizam este campo para consultas baseadas em metadados, tal como o XML *Extensible Markup Language*.

QUERYHIT

A mensagem do tipo QUERYHIT possui o *payload* representado pela Figura 2.14.

Campos	Número de *hits*	Porta	Endereço IP	Velocidade	Conjunto de resultados	Dados opcionais	ID do servente
Byte Offset	0	1...2	3...6	7...10	11...10+N	11+N...L-17	L-16...L-1

Figura 2.14: A mensagem QUERYHIT

- Número de *hits*: Número de arquivos que satisfazem à consulta;
- Porta: Indica a porta em que o *host* que responde pode receber conexões para a transferência dos arquivos;
- Endereço IP: Indica o endereço IP do *host*;
- Velocidade: Indica a velocidade máxima, em kbits/s do *host*;
- Conjunto de resultados: Um conjunto de respostas para a QUERY recebida, contendo "Número de *hits*" resultados, cada uma contendo uma estrutura como representada pela Figura 2.15

Campos	Índice do arquivo	Tamanho do arquivo	Nome do arquivo	NULL	Dados opcionais	NULL
Byte Offset	0...3	4..7	8...7+K	8+K	9+K...R-2	R-1

Figura 2.15: A estrutura de um resultado da QUERYHIT

O campo "Índice de arquivo" é um valor associado ao arquivo que o unicamente identifica dentro do conjunto de respostas enviado. O "Tamanho do arquivo" indica o tamanho do arquivo em *bytes*. O "Nome do arquivo" indica o nome

do arquivo compartilhado, terminado em NULL. Os "Dados opcionais" podem conter metadados sobre o arquivo compartilhado.

- Dados opcionais: Campo opcional para dados de QUERYHITs estendidos;
- Identificador do servente: Este campo de 16 *bytes* identifica unicamente o servente na rede. Tipicamente é calculado por alguma função sobre seu endereço IP. Este identificador é instrumento para a operação efetuada pela mensagem PUSH.

As mensagens do tipo QUERYHIT são enviadas somente em resposta a uma QUERY. Um servente deve responder somente se contém arquivos que podem satisfazer o critério de busca. Devem ser gerados inicialmente com o valor de *Hops* em zero, e o valor TTL igual ao número de *Hops* efetuados pela mensagem QUERY correspondente. O "ID do descritor" deve conter o mesmo valor associado à QUERY, permitindo, assim, que um servente possa rotear as QUERYHITs adequadamente.

A mensagem QUERYHIT, com sua estrutura complexa, é aquela que pode apresentar o *payload* de maior tamanho. Para a melhor eficiência, um servente que recebe uma QUERY deve limitar a quantidade de dados que envia como resposta em uma QUERYHIT. Quando muitos *hits* são detectados, os serventes devem dividi-los em um subconjunto de resultados, e enviá-los separadamente com atrasos. O processo de roteamento dos QUERYHITs de volta ao servente que originou a consulta é chamado de *backtrack*.

PUSH

A mensagem do tipo PUSH possui o *payload* representado pela Figura 2.16.

Campos	ID do servente	Índice do arquivo	Endereço IP	Porta	Dados opcionais
Byte Offset	0...15	16...19	20...23	24...25	26...L-1

Figura 2.16: A mensagem PUSH

- ID do servente: *String* de 16 *bytes* que identifica unicamente o servente na rede. Este servente está sendo requisitado para "empurrar" o arquivo indicado por "Índice do arquivo". O servente que inicia o PUSH deve atribuir a este campo o mesmo valor dado ao campo "ID de servente" da mensagem QUERYHIT;
- Índice do arquivo: Esse valor identifica unicamente o arquivo a ser "empurrado" pelo servente. O nó que inicia o PUSH deve atribuir a este campo o valor dado em "Índice do arquivo" de um arquivo contido no conjunto de resultados do QUERYHIT correspondente;
- Endereço IP: Indica o endereço IP do *host* para o qual o arquivo deve ser "empurrado";
- Porta: Indica a porta do *host* para a qual o arquivo deve ser "empurrado";
- Dados opcionais: Este campo é reservado para futuras extensões do protocolo.

Um servente deve enviar uma mensagem PUSH caso receba uma QUERYHIT de um servente que não pode receber pedidos de conexão. Isso pode ocorrer em casos de nós que estão atrás de *firewalls*. Quando um servente recebe uma PUSH, ele deve dar continuidade ao processo somente se o seu identificador de servente é igual ao valor contido em "ID do servente" da mensagem recebida.

Roteamento de mensagens

A natureza descentralizada das redes *Gnutella* requer que mensagens sejam roteadas através dos nós de maneira apropriada. Um servente *Gnutella* deve rotear as mensagens segundo os seguintes critérios:

- PONGs só podem ser enviadas através do mesmo caminho que a PING correspondente atravessou. Um servente que recebe uma PONG cujo identificador não está armazenado no seu *cache* de identificadores de PINGs enviadas ou roteadas deve remover a PONG da rede;
- QUERYHITs só podem ser enviadas através da mesma rota que a QUERY correspondente atravessou. Um servente que recebe uma QUERYHIT cujo

identificador não está armazenado no seu *cache* de identificadores de QUERYs roteadas ou enviadas deve remover a QUERYHIT da rede;

- PUSHs só podem ser enviadas através da mesma rota que a QUERYHIT correspondente atravessou. Um servente que recebe uma PUSH cujo identificador de servidor não está armazenado no seu *cache* de identificadores de servidor enviados ou roteados deve remover a PUSH da rede;

- Um servente deve encaminhar PINGs e QUERYs a todos os seus nós vizinhos, exceto àquele que enviou a mensagem;

- Um servente deve decrementar o campo TTL e incrementar o *Hops* antes de encaminhar qualquer mensagem aos nós vizinhos. Se, após decrementado, o TTL apresentar o valor zero, a mensagem não deve ser mais roteada através das conexões, sendo removida da rede;

- Um servente que recebe uma mensagem com "Descritor de *payload*" e "ID de descritor" iguais a algum já antes recebido, deve descartar a mensagem;

As Figuras 2.17 e 2.18 apresentam exemplos de roteamento em uma pequena rede *Gnutella*.

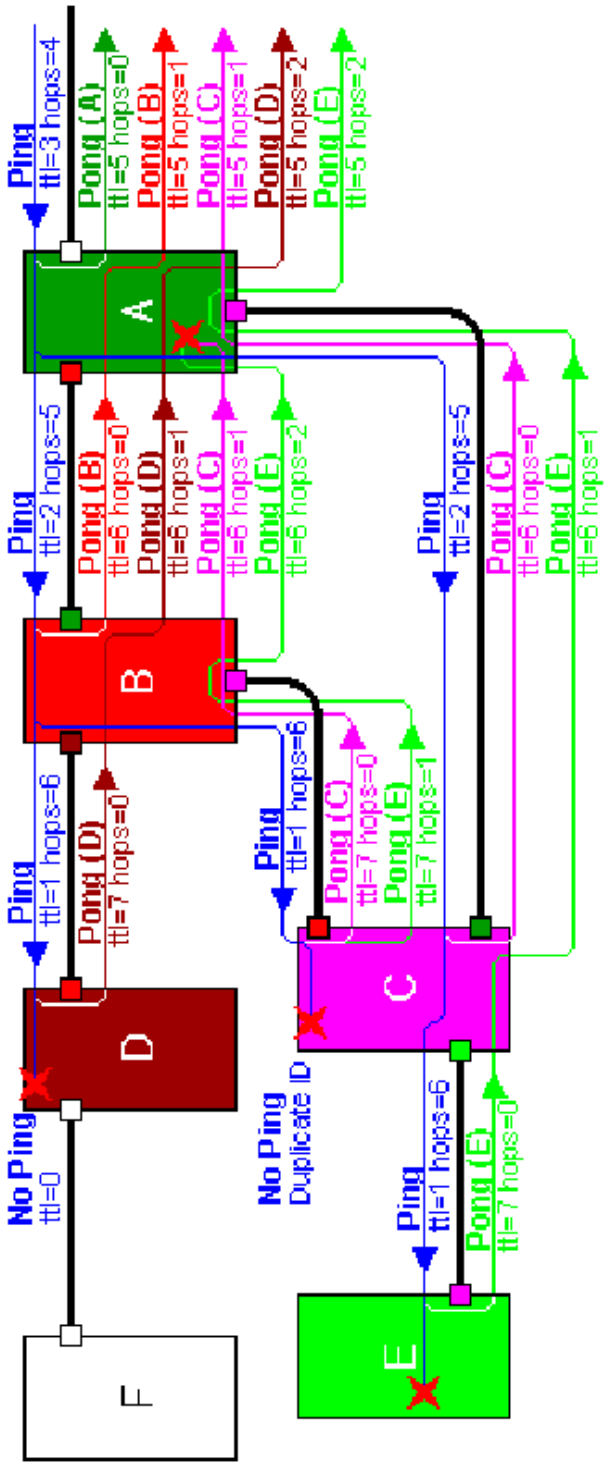


Figura 2.17: Exemplo de Roteamento de PINGs e PONGs

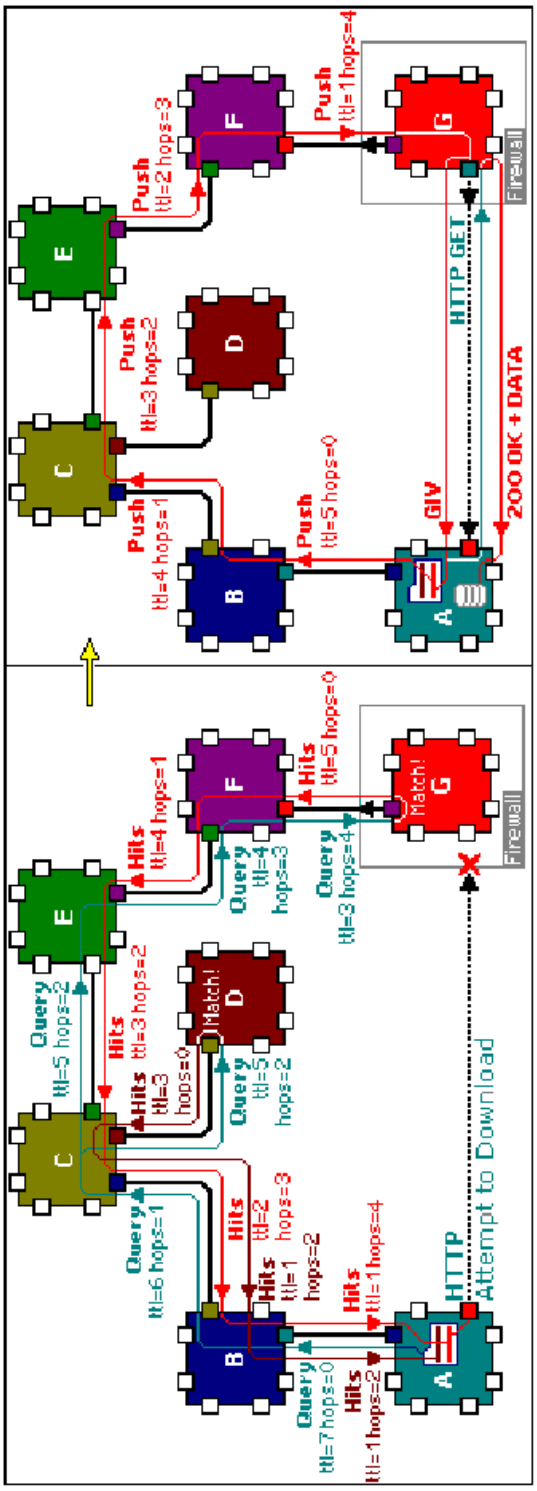


Figura 2.18: Exemplo de Roteamento de QUERYs, QUERYHITs e PUSHs

Para visualizar como o *Gnutella* trabalha, pode-se imaginar uma grande malha de nós, representando vértices de um grafo, e suas conexões representadas por arestas. O nó precisa, no início, encontrar pelo menos um outro nó já participante na rede, para poder propagar suas mensagens e rotear mensagens de outros nós.

O grande problema da rede *Gnutella* é a maneira como seu protocolo implementa o roteamento de mensagens. Dada uma consulta, o nó envia uma mensagem do tipo QUERY a todos os nós diretamente a ele conectados. Estes nós, em seguida, processam a consulta localmente, enviando um QUERYHIT em caso de sucesso, e propagam a consulta a todos os nós a eles diretamente conectados, com exceção do nó que a enviou a.

A consulta vai ser propagada de nó-a-nó até que o valor de TTL no cabeçalho dos pacotes atinja zero. Este mecanismo é chamado de *query flooding* e, devido à natureza *broadcast* da consulta, o sistema não possui uma boa escalabilidade [35].

A demanda de largura de banda cresce exponencialmente com um acréscimo linear do número de nós, portanto, aumentar o número de nós pode causar uma rápida saturação na rede. A manipulação do valor de TTL pode diminuir o número de mensagens, no entanto, implica em uma menor eficiência na busca, por atingir um número menor de nós.

Na prática, a busca numa rede *Gnutella* é lenta e não confiável. Cada nó é um computador comum, e, portanto, estão constantemente conectando e desconectando, fazendo com que a rede nunca esteja completamente estável. Uma vez que conexões de usuários individuais são quase sempre lentas, pode-se tomar muito tempo para a consulta se transferir por grande parte da rede.

Esse protocolo oferece flexibilidade no processamento das consultas pois, cada nó pode determinar como processar uma consulta e responder de acordo com ela. Por outro lado, sistemas *Gnutella* são muito suscetíveis a atividades maléficas. Os nós com más intenções podem enviar muitas consultas, o que produz uma carga significantemente grande na rede.

Download de arquivos

Assim que um servente recebe um QUERYHIT, o usuário pode decidir iniciar o *download* de um dos arquivos presentes no "conjunto de resultados". Os arquivos são transferidos por fora da rede *Gnutella*, através de uma conexão direta entre os nós. Arquivos de dados nunca são transferidos dentro da rede *Gnutella*. O protocolo utilizado para negociar a transferência é o HTTP. Na iniciação do *download* o servente que requer o arquivo deve enviar a seguinte *string*:

```
GET /get/<Índice do arquivo>/<Nome do arquivo>/ HTTP/1.0\r\n
User-Agent: Gnutella/0.4\r\n
Range: bytes=<Offset de início>-\r\n
Connection: Keep-Alive\r\n
\r\n
```

O servidor do arquivo, então, envia uma resposta também compatível com o HTTP. O arquivo de dados segue esta resposta e deve ser lido por meio do *socket* até que se esgote o tamanho especificado na resposta do servidor. O protocolo define ainda um mecanismo para efetuar o *resume* de arquivos, caso haja algum problema, permitindo que a transferência seja reiniciada por algum ponto do meio do arquivo.

2.5.2 *Napster*

O *Napster* é um aplicativo para o compartilhamento de arquivos, e seu protocolo, proprietário, aqui também referenciado simplesmente como *Napster*, é o representante mais importante das arquiteturas semi-centralizadas não intermediadas, pois utiliza um servidor central para o armazenamento da informação referente à localização dos arquivos na rede [14].

Sua tecnologia permitiu que usuários compartilhassem arquivos no formato MP3 com facilidade, e isso levou a indústria da música a impetrar massivas acusações sobre violação de direitos autorais por parte dos usuários e dos proprietários do aplicativo. Mesmo com a desativação do seu serviço original, o *Napster* pavimentou o caminho às redes *peer-to-peer* descentralizadas, que são muito mais difíceis de controlar.

Shawn Fanning foi quem lançou o *Napster* em 1999. Sua motivação era criar um sistema para busca de arquivos de música mais fáceis de utilizar do que os disponíveis na época, tal como o IRC (*Internet Relay Chat*). Isso causou um alvoroço grande o bastante para ícones da cena musical mundial e grandes empresas entrarem com ações judiciais contra os utilizadores e criadores do sistema [31]. Com essa nova ferramenta, a maneira pela qual as pessoas utilizavam a *Internet* mudou, e redes de alta velocidade em muitas universidades passaram a ser sobre-utilizadas: 80% do tráfego era para transferência de arquivos MP3 [31].

O formato de suas mensagens é proprietário, no entanto é possível observar algumas operações que são utilizadas. No momento da conexão de um nó na rede, ele deve autenticar-se a um servidor central e enviar os metadados dos arquivos que disponibiliza. Os dados ficam armazenados no servidor que responde às consultas dos nós [1].

Um nó que procura um arquivo direciona-se ao servidor central enviando a consulta. O servidor responde, em caso positivo, com os endereços IP de nós conectados que podem atender àquela consulta. De posse deste endereço, o nó efetua uma conexão direta com o outro que disponibiliza o arquivo e inicia a transmissão sem a necessidade do servidor central, caracterizando aqui o modelo *peer-to-peer*.

Sistemas que são construídos de acordo com essa abordagem possuem um desempenho variável, dependendo de como são implementados [50]. Apesar de apresentarem um grande nível de centralização, mostram ser uma opção bastante robusta, como provado pelo *Napster*, registrando um pico de 26.4 milhões de usuários em 2001 [28].

2.5.3 *Gnutella2*

O protocolo *Gnutella2* possui documentação aberta [44], sendo um retrabalho do protocolo *Gnutella*, porém não reconhecido oficialmente como uma extensão do protocolo oficial. Ele elimina muitos dos aspectos do antigo protocolo *Gnutella* a não ser o *handshake* de conexão, adotando um sistema novo e completo.

O protocolo *Gnutella2* não é interoperável com nenhuma versão do protocolo *Gnutella*, no entanto utiliza a mesma arquitetura da versão 0.6: a arquitetura descentralizada não estruturada baseada em ultranós.

A eleição de quais nós vão atuar como *hub* é feita com base em vários critérios: o sistema operacional que permite a abertura de vários *sockets* simultaneamente, a quantidade de memória RAM (*Random-access memory*), a velocidade de processamento, o número de horas conectados à rede, a largura de banda adequada, e a habilidade de aceitar conexões TCP e UDP sem intervenção de *firewalls*.

As conexões TCP são utilizadas entre os nós quando eles são eleitos para formarem uma conexão permanente, em uma rede com topologia baseada em ultranós, ou *hubs*, fortemente conectados, servindo um denso *cluster* de nós folha.

Estabelecer uma conexão TCP para entregar um simples pacote de informação é perda de volume de dados e tempo, especialmente quando considera-se uma grande quantidade de nós. O UDP oferece uma solução para esse caso, por ser um protocolo que apresenta um menor *overhead*.

Porém, como o UDP é um protocolo não confiável, os pacotes podem ser perdidos ao longo da rota. O protocolo *Gnutella2* resolve isto implementando uma camada de confiabilidade sobre o protocolo básico UDP. Esta camada de confiabilidade divide funcionalidades comuns ao TCP, mas não provê nenhum estado de conexão e, assim sendo, aproveita o desempenho implícito ao UDP.

O protocolo *Gnutella2*, portanto, permite que haja comunicação através de três vias distintas: para um volume significante de dados, ou para o caso de uma comunicação em que dados futuros vão ser trafegados, usa-se o TCP, neste último com conexão persistente; para volumes pequenos de dados importantes, usa-se o UDP confiável; para volumes de dados pequenos de dados não muito importantes, usa-se o UDP não confiável.

O protocolo *Gnutella2* permite, ainda, que seja feito o *swarming download*, em que pedaços de arquivos podem ser obtidos, paralelamente, de diferentes *hosts*. Algumas mensagens que são definidas no protocolo *Gnutella2* são parecidas com as mensagens do protocolo *Gnutella*, gerando confusão ao se levantar aspectos de interoperabilidade.

2.5.4 *eDonkey*

Desenvolvido pela *MetaMachine*, o *eDonkey*, também chamado de *eDonkey2000* ou *ed2k*, é um protocolo desenvolvido para o compartilhamento de arquivos, especial-

mente de músicas, filmes e *software* [25].

Assim como a maioria dos protocolos de compartilhamento de arquivos, utiliza a arquitetura descentralizada: arquivos não são armazenados em um servidor central, mas são trocados diretamente pelos pares, assim como não há indexação centralizada de arquivos, e sim a utilização de ultranós. A utilização do algoritmo de hash MD4 sobre o conteúdo dos arquivos, permite uma identificação de arquivos idênticos com nomes diferentes dentro do espaço de armazenamento.

O *eMule* foi um dos aplicativos mais populares na *Internet* [27]. O protocolo utilizado por esse aplicativo é baseado no *eDonkey*. Cada nó folha é pré-configurado com uma lista de nós *hub* e uma lista de arquivos compartilhados em seu sistema local, que é transmitida ao nó *hub* no momento de sua conexão.

Uma fila é criada para cada arquivo de modo a controlar os *downloads* e *uploads*, e é possível fazer *downloads* simultâneos de diferentes localizações para acelerar o processo e melhorar o balanceamento de carga. É possível também disponibilizar pedaços de arquivos que sequer tenham terminado seu *download* completamente.

O protocolo utilizado pelo *eMule* estende as capacidades do protocolo *eDonkey* permitindo que os nós folha troquem informações sobre os nós *hub*, sobre outros nós folha e sobre arquivos [19].

Os nós *hub*, usualmente, não armazenam arquivos, mas, sim, informações sobre os arquivos localizados nos nós folha. Outra funcionalidade destes nós é intermediar conexões entre nós folha que estão atrás de *firewalls* e não podem receber pedidos de conexão. Essa intermediação de conexões acrescenta uma grande carga nos nós *hub*.

Faz parte do protocolo um sistema de créditos que estimulam o usuário a compartilhar mais arquivos, aumentando, assim, a largura de banda disponível a um determinado nó para fazer *downloads*. Esse recurso é útil para eliminar os chamados *freeloaders*, que utilizam recursos da rede virtual sem prover recursos [19].

2.5.5 *FastTrack*

O *FastTrack* é um protocolo proprietário. No entanto, algumas partes de sua implementação são conhecidas [16]. Em sua organização, os nós são divididos em nós *folha* e ultranós. Ultranós são responsáveis por indexar os arquivos que os nós folha

compartilham e também fazem trabalhos de estatísticas.

Os mecanismos de conexão e de distribuição de responsabilidades são similares ao protocolo *Gnutella2*. A fase de transferência de arquivos é negociada utilizando o protocolo HTTP. Os nós folha não se comunicam uns com os outros, a não ser para a transferência de arquivos.

O aplicativo *Kazaa*, muito utilizado na *Internet*, utiliza a porta 1214 para efetuar as transferências das mensagens. No entanto, versões recentes utilizam portas aleatórias, e algumas escutam na porta 80. Alguns formatos de pacotes e seus tipos são conhecidos, mas pouco é sabido sobre a comunicação entre ultranós.

Alguns aplicativos que implementam esse protocolo utilizam um sistema de "reputação", que visa incentivar os usuários a compartilhar arquivos e permitir que *uploads* sejam efetuados. Inicia-se com nível de participação de valor dez, e pode-se atingir o valor 1000. Um maior nível de participação significa que o usuário está conectado por grandes períodos de tempo e permitiu que usuário tirassem proveito disso, obtendo seus arquivos. Usuários com maiores níveis de participação são favorecidos em filas de espera de *download* e recebem melhor QoS (*Quality of Service*).

2.5.6 *Kademlia*

O protocolo *Kademlia* [24] implementa a arquitetura descentralizada estruturada. Um identificador é associado a cada nó da rede, e um valor a cada documento. Cada participante possui uma chave de identificação de 160 *bits*, e cada documento será roteado até o nó de melhor posição dentro do sistema. Os pares (chave, documento) são armazenados em nós com identificadores mais "próximos" à chave, para se ter a noção de proximidade.

O *Kademlia* visa a minimizar o número de mensagens que os nós têm que enviar uns aos outros para adquirir informações sobre a rede. A informação de configuração é espalhada automaticamente como efeito das buscas, e os nós possuem conhecimento e flexibilidade para rotear consultas através de caminhos de menor latência.

Um nó que deseja entrar na rede deve antes passar por um processo de iniciação. Nesta fase, o nó deve descobrir o endereço IP de outro nó, obtido diretamente pelo usuário, ou por uma lista armazenada, que já está participando da rede *Kademlia*.

Um identificador randômico ainda não utilizado é calculado e atribuído ao novo nó.

O algoritmo do *Kademlia* é baseado no cálculo da "distância" entre dois nós. Essa distância é calculada como um "ou exclusivo" de dois identificadores, tendo como resultado um número inteiro. A "distância" não possui relações com condições geográficas, mas designa a distância dentro da faixa de identificadores. Com isso, pode ocorrer o caso de um nó na Alemanha e um nó na Austrália serem vizinhos dentro do sistema.

O número de nós contactados durante uma busca é dependente do tamanho da rede. Se o número de participantes na rede dobra, então um nó requerente deve consultar somente um nó a mais por busca, e não duas vezes mais, provando a boa escalabilidade do sistema.

Algumas vantagens ainda podem ser encontradas na redes descentralizadas estruturadas, que claramente aumentam a resistência contra ataques DoS (*Denial of Service*). Mesmo que um conjunto de nós forem atacados, isso implicará um pequeno impacto na disponibilidade da rede, que se recuperará, pois o algoritmo faz com que estes "buracos" sejam supridos por outros nós.

Uma vez que não há uma instância central para armazenar os índices dos arquivos existentes, essa tarefa é dividida de maneira igual através de todas as entidades. O nó que deseja compartilhar um arquivo efetua um processamento, aplicando um *hash* no conteúdo do arquivo que, como resultado, vai identificá-lo dentro da rede de compartilhamento. Por isso, o resultado dos *hashes* e os identificadores dos nós devem possuir o mesmo tamanho, garantindo a compatibilidade entre os valores.

Para efetuar uma consulta, busca-se na rede o nó cujo identificador tem a menor distância do *hash* do arquivo, com a utilização da lista de contatos que está armazenada no nó local. Os contatos armazenados na rede estão sob constante mudança pois os nós conectam e desconectam aleatoriamente. A replicação das informações a respeito da localização de nós faz com que o desempenho da busca aumente. Com isso, quando um contato não está na lista local, o nó consulta o nó mais próximo daquele que ele procura afim de recuperar o endereço final.

O *Kademlia* é utilizado pelo programa de compartilhamento *Morpheus*. Outros protocolos possuem a mesma natureza, com variações em desempenho quando analisados sob a perspectiva do número de mensagens que requerem para uma busca,

tempo de configuração de um novo nó, ou quantidade de dados que os nós devem armazenar para efetuar o roteamento das mensagens.

A Tabela 2.2 mostra uma tabela que resume alguns protocolos utilizados por aplicativos de compartilhamento de arquivos na *Internet*, mostrando a arquitetura de rede *peer-to-peer* utilizada por cada um deles.

Tabela 2.2: Exemplos de protocolos de compartilhamento de arquivos e suas arquiteturas

Protocolo	Arquitetura
Gnutella	Descentralizada pura
Gnutella2	Baseada em ultranós
Freenet	Estruturada
Napster	Semi-centralizada não intermediada
Kademlia	Estruturada
eDonkey	Baseada em ultranós
FastTrack	Baseada em ultranós
BitTorrent	Descentralizada pura
Chord	Estruturada
Pastry	Estruturada

A escolha de quais arquiteturas e protocolos devem ser utilizados depende das características do aplicativo e do ambiente em que será executado. Deve-se considerar todos os requisitos necessários de um aplicativo *peer-to-peer*, desde o seu desenvolvimento, até a sua implantação, levando-se sempre em consideração o seu desempenho para a sua especificidade.

Capítulo 3

Os protocolos da camada de transporte

Os procedimentos utilizados para a implementação dos mecanismos que permitem a troca de dados entre os elementos de uma rede de computadores são complexos. Para reduzir esta complexidade, tarefas são divididas em camadas; cada camada executa um subconjunto de funções, utilizando serviços da camada inferior, e provendo serviços à camada superior.

Os protocolos implementam os serviços que uma determinada camada oferece, definindo os tipos e a sintaxe das várias mensagens, a semântica de seus campos e as regras que determinam quando essas mensagens são enviadas e respondidas.

A camada de transporte é situada entre a camada de aplicação e a camada de rede da arquitetura *Internet*, e tem como função principal oferecer uma "comunicação lógica" entre processos de aplicação em diferentes hospedeiros. Isso significa que, embora os processos não estejam fisicamente conectados, do ponto de vista da aplicação, é como se o estivessem. A Figura 3.1 ilustra essa "comunicação lógica", destacando as pilhas de protocolos utilizadas em uma comunicação entre dois processos, em dois *hosts* diferentes.

Os protocolos da camada de transporte mais utilizados na *Internet* são o UDP (*User Datagram Protocol*) e o TCP [45]. Um outro procolo da camada de transporte é o SCTP. Este capítulo tem como objetivo mostrar as características e os serviços oferecidos por esses protocolos, e efetuar uma comparação entre eles.

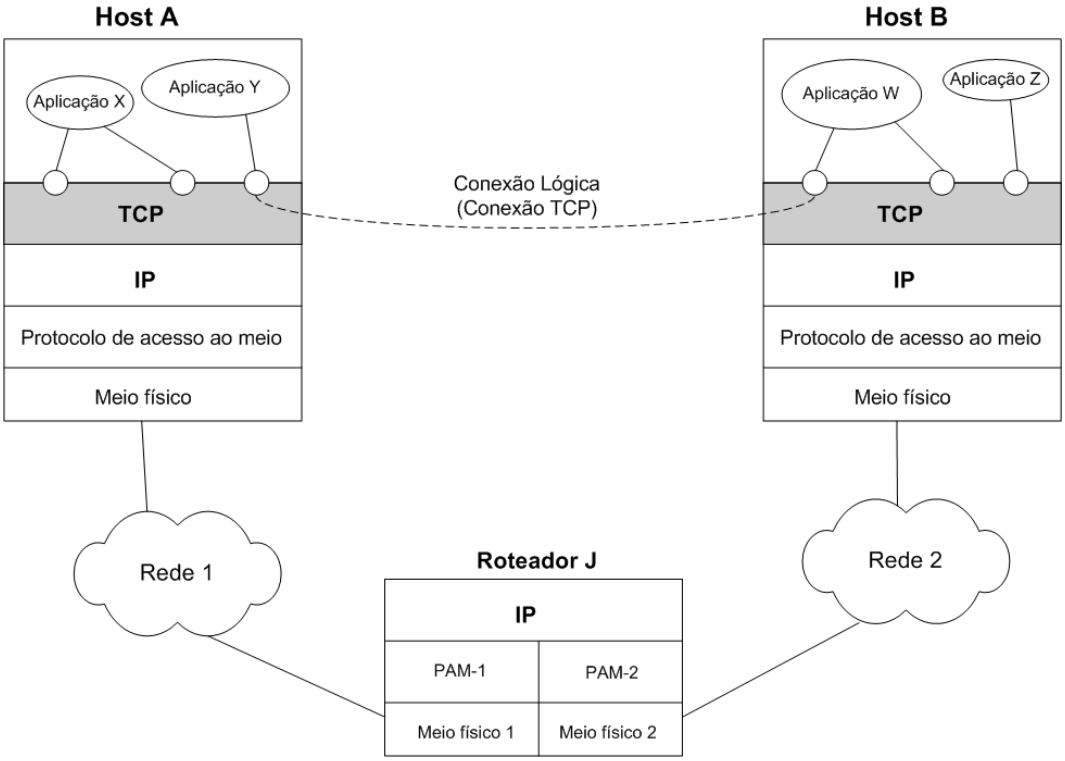


Figura 3.1: Comunicação lógica entre dois processos, em dois *hosts* diferentes

3.1 O protocolo UDP

O UDP [32] é um protocolo de transporte simples, leve, com um modelo de serviço que apresenta as funções mínimas da camada de transporte: multiplexação, demultiplexação e verificação de erros. Os dados da camada de aplicação, quando empacotados pelo UDP, são referenciados por datagrama. O seu cabeçalho é como mostra a Figura 3.2.

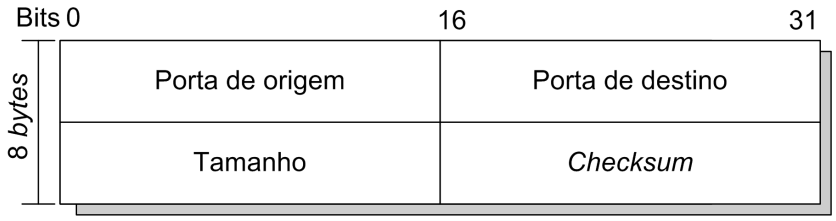


Figura 3.2: Cabeçalho do protocolo UDP

Os campos "porta de origem" e "porta de destino", de 16 *bits* cada um, identificam os processos nos sistemas finais que enviam e recebem os datagramas. Uma vez que o UDP é um protocolo que não armazena estados, a "porta de origem" pode ser opcional, pois um computador que envia dados pode não solicitar respostas. Se

não utilizado, o valor deste campo deve ser configurado com o valor zero.

O campo "tamanho" designa o comprimento do datagrama em *bytes*, incluindo ambos o cabeçalho e o campo de dados. O tamanho mínimo de um datagrama é oito *bytes*, sendo, neste caso, o campo de dados vazio. O campo "*checksum*" carrega a soma de verificação utilizada na detecção de erros no cabeçalho e nos dados transmitidos.

É um protocolo orientado à mensagem, não orientado à conexão e não confiável, isto é, não garante que os pacotes sejam entregues ao destinatário e nem que cheguem de forma ordenada. No entanto, se o pacote chegar ao destinatário, o *checksum* permite a verificação da existência de erros.

As aplicações construídas sobre o UDP devem tratar perdas, erros, duplicação e desordenação na entrega de datagramas. No entanto, muitas destas aplicações se adaptam melhor ao UDP pois:

- Não há estabelecimento de conexão;
- Não há armazenamento de estados de conexão;
- Possui pouco *overhead* no cabeçalho do pacote;
- Possui taxa de envio não regulada.

O UDP não necessita de estabelecimento de conexão, pois nenhuma variável necessita ser configurada no remetente, ou no destinatário. Assim sendo, é um protocolo que não armazena estado de conexão, ou seja, não necessita de variáveis para armazenar detalhes sobre a troca de dados entre sistemas finais.

O seu cabeçalho é pequeno, sendo considerado *lightweigth*, ou seja, adiciona pouco *overhead* em relação ao campo de dados. A sua taxa de envio não é regulada, não havendo mecanismos para a detecção de transbordamento de *buffer* do destinatário ou de congestionamento na rede.

3.2 O protocolo TCP

O TCP oferece outros serviços além da multiplexação, demultiplexação e verificação de erros que o UDP oferece. Esse protocolo oferece o serviço de entrega de dados

confiável e ordenado. É um protocolo orientado à conexão e a *bytes*, e ainda implementa o controle de fluxo e de congestionamento. Os dados da camada de aplicação empacotados com o cabeçalho do TCP é denominado segmento. O seu cabeçalho pode ser representado pela Figura 3.3.

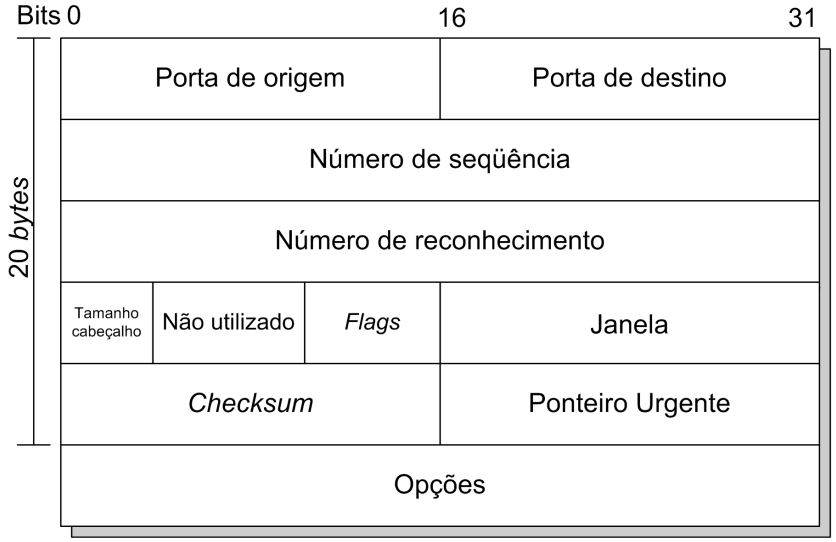


Figura 3.3: Cabeçalho do protocolo TCP

Os campos "porta de origem" e "porta de destino", assim como no UDP, determinam a quais processos encaminhar os dados nos sistemas finais. Os "número de seqüência" e "número de reconhecimento" são utilizados para a implementação do serviço confiável de transferência de dados.

O campo "tamanho do cabeçalho" designa o tamanho do cabeçalho do pacote em palavras de 32 *bits*. O campo "*flags*" contém 6 *bits*, que são utilizados para identificar início e fim de conexão, assim como entrega de dados urgentes, entre outras finalidades. O campo "janela" é utilizado para o controle de fluxo.

O campo "*checksum*", como no protocolo UDP, é utilizado para a verificação de erros no cabeçalho e nos dados do usuário, e o campo "ponteiro urgente" auxilia na entrega de dados urgentes à camada de aplicação. O campo "opções" é facultativo, e dentre outras funções, pode ser utilizado para definir o MSS (*Maximum Segment Size*) entre os sistemas finais.

Para efetuar uma conexão, o TCP implementa o chamado *3-way handshake*. Este mecanismo executa a troca de três mensagens entre os hospedeiros, que faz com que *buffers* sejam criados para armazenar dados durante o tempo de vida da conexão.

São definidos, nesta fase, os número de seqüência iniciais, sendo J o do cliente, e K o do servidor, como mostra a Figura 3.4.

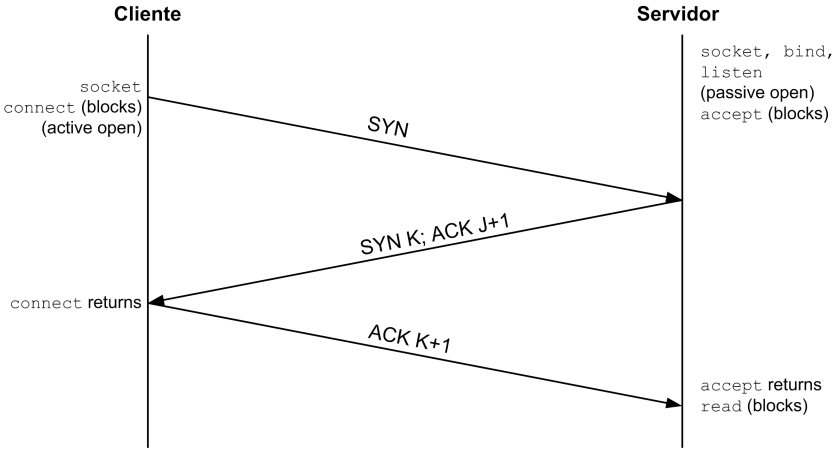


Figura 3.4: Iniciação de conexão TCP

O cliente efetua um pedido de conexão (*active open*) enviando um segmento TCP do tipo SYN, e define o número de seqüência inicial para os dados que enviar. Normalmente não há dados do usuário com este segmento; somente o cabeçalho IP e TCP, e possíveis opções do TCP. O servidor responde então com um ACK, enviando também seu número de seqüência inicial. O cliente, então, responde com um SYN, e a conexão está estabelecida.

Durante o tempo de vida de uma conexão TCP, o protocolo que roda em cada um dos hospedeiros faz transições por vários estados TCP. A Figura 3.5 mostra estes vários estados, desde o estabelecimento até o encerramento da conexão[1].

Há 11 diferentes estados para uma conexão TCP, e as regras do protocolo ditam como as transições acontecem. Por exemplo, se uma aplicação está no estado CLOSED e decide conectar-se a um servidor (*active open*), o TCP envia um SYN e passa para o estado SYN_SENT. Se recebe em seguida um SYN conjugado com um ACK, envia um ACK, e o novo estado passa a ser ESTABLISHED. É neste estado que a maioria das transferência dos dados ocorrem.

Enquanto o TCP troca três mensagens para estabelecer uma conexão, ele troca quatro para encerrá-la, como mostra a Figura 3.6. O lado que recebe um FIN efetua

[1]Na figura apresentada, duas transições são omitidas: iniciação simultânea, em que SYNs percorrem a rede em sentidos opostos ao mesmo tempo, e encerramento simultâneo de conexão, em que FINs percorrem a rede em sentidos opostos ao mesmo tempo.

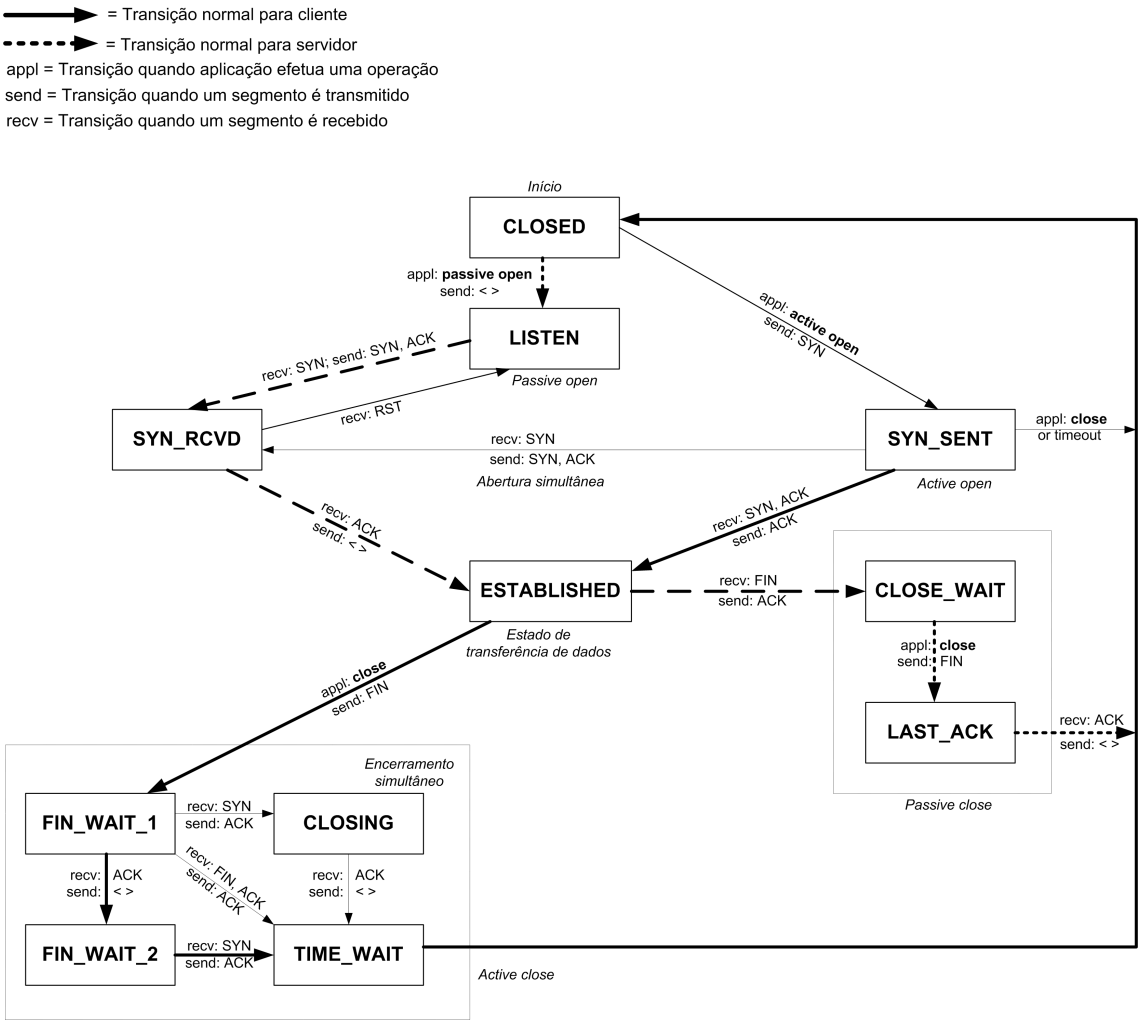


Figura 3.5: Estados de uma conexão TCP

o encerramento passivo (*passive close*), sendo, então, enviado um reconhecimento ao FIN. O recebimento de um FIN quer dizer que dados não serão mais recebidos através da conexão. Algum tempo depois, a aplicação que recebeu o FIN também decide encerrar a sua conexão. Isso faz com que o TCP envie um FIN ao outro par, que envia o respectivo reconhecimento.

O TCP vê os dados como uma cadeia de *bytes* desestruturada, mas ordenada. O uso que o TCP faz dos número de seqüência reflete essa visão, pelo fato de que esses números são aplicados sobre a cadeia de *bytes* transmitidos, e não sobre a série de segmentos. Do ponto de vista da aplicação, um fluxo de *bytes* é "empurrado" através do *socket*, no qual uma determinada quantidade de *bytes* de dados vai ser enviada ao destinatário.

O controle de fluxo é o mecanismo que os sistemas finais utilizam para evitar que

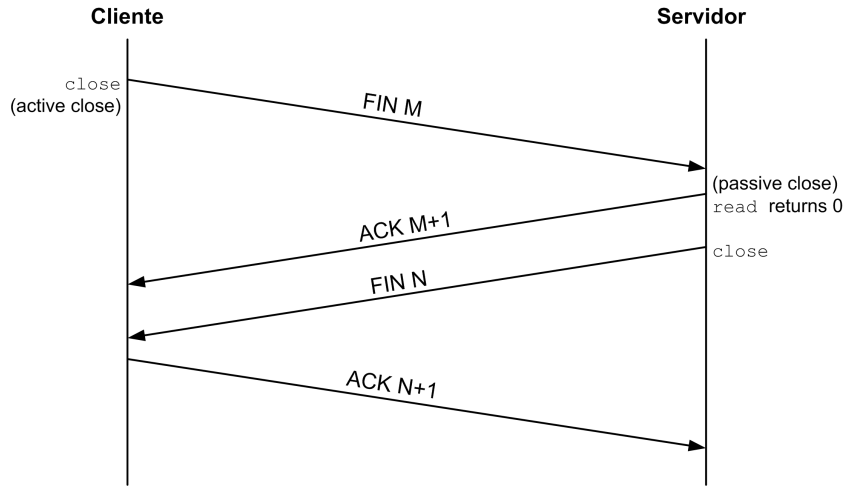


Figura 3.6: Encerramento de conexão TCP

os dados enviados transbordem a capacidade do *buffer* de seus destinatários. Para isso, o campo do cabeçalho "janela" é utilizado. A janela de recepção é utilizada para dar ao remetente uma idéia de quanto espaço livre de *buffer* está disponível no destinatário.

Na conexão do TCP, que é *full-duplex*, cada remetente mantém uma janela de recepção distinta, que varia dinamicamente durante o tempo de vida útil da conexão. A cada segmento que cada remetente envia ao outro, o campo "janela" é atualizado, informando o outro par sobre quantos *bytes* estão disponíveis a receber, sem encontrar problemas na utilização de seu *buffer*.

O controle de congestionamento, diferentemente do controle de fluxo, visa ao bem-estar geral da rede, e não somente ao do destinatário. O congestionamento é causado, principalmente, pelo estouro na capacidade dos *buffers* dos roteadores da rede, em que demasiadas fontes estão tentando enviar dados a uma taxa muito alta. O TCP utiliza para o controle de congestionamento um algoritmo chamado "aumento-aditivo-diminuição-multiplicativa", ou AIMD (*Additive-increase-multiplicative-decrease*) [20].

O TCP é utilizado quando a aplicação necessita de confiabilidade sobre o serviço não confiável da rede IP. No entanto, em alguns casos, o TCP não provê as exatas funcionalidades necessárias à aplicação, ou provê mais que o necessário. No primeiro caso a aplicação necessita fazer um trabalho extra para utilizar o TCP, enquanto no último, a funcionalidade extra do TCP pode ser um entrave [33].

3.3 O protocolo SCTP

3.3.1 Histórico do SCTP

O desenvolvimento do SCTP foi motivado pela necessidade de sanar as deficiências do TCP e do UDP ao lidar com algumas aplicações, em especial aquelas de telefonia sobre a rede IP. Em princípio, seu desenvolvimento visava a um protocolo que operava sobre o UDP, oferecendo confiabilidade sobre um protocolo da camada de transporte não confiável [41].

Muitas implementações foram feitas até que se chegasse ao precursor do SCTP, chamado MDTP (*Multi-Network Datagram Transmission Protocol*). Assim que a primeira implementação utilizável foi concluída, seus autores a submeteram para a IETF (*Internet Engineering Task Force*).

Uma iniciativa da IETF relativa à telefonia sobre IP estava sendo desenvolvida paralelamente a essa submissão pelo grupo SIGTRAN (*Signaling Transport Working Group*). Ao analisar o protocolo MDTP, o SIGTRAN considerou seus conceitos interessantes e propôs várias modificações no protocolo, a fim de melhorar e refinar seu desenvolvimento original.

Durante esse refinamento, o nome do protocolo mudou de MDTP para SCTP, como sinal de expansão de escopo e de funcionalidade do protocolo. Essa expansão permitiu que o SCTP passasse a ser considerado um protocolo da camada de transporte, ao invés de um protocolo atuando sobre o UDP, sendo assim utilizado diretamente sobre a camada de rede.

3.3.2 Características do SCTP

O SCTP é um protocolo que, como o TCP e o UDP, pertence à camada de transporte da pilha de protocolos da arquitetura TCP/IP [30, 41]. É um protocolo confiável, orientado à conexão e à mensagem. Implementa o controle de fluxo e de congestionamento, e pode ser usado concorrentemente com o TCP, sem falhas na justiça de utilização do canal de dados. Apresenta funcionalidades adicionais em relação ao TCP e ao UDP, tais como o *multi-homing*, os multi-fluxos e entrega desordenada[2]

[2]O termo "desordenada" aqui proposto, e doravante usado no decorrer do texto, está relacionado com mensagens do SCTP, e refere-se ao tratamento de entrega das mensagens sem considerar a ordem ou

das mensagens, que torna este protocolo mais leve e, portanto, mais adequado para determinados tipos de aplicações.

Os dados da camada de aplicação empacotados com o cabeçalho do protocolo SCTP são denominados simplesmente pacotes. Os campos desse cabeçalho são mostrados na Figura 3.7. Nas redes IP, esses pacotes são encapsulados no *payload* do pacote IP. Um pacote SCTP é formado por um cabeçalho comum e *chunks*. Múltiplos *chunks* podem ser multiplexados em um só pacote, até o limite definido pelo MTU (*Maximum Transfer Unit*). Um *chunk* pode conter dados de controle ou dados do usuário.

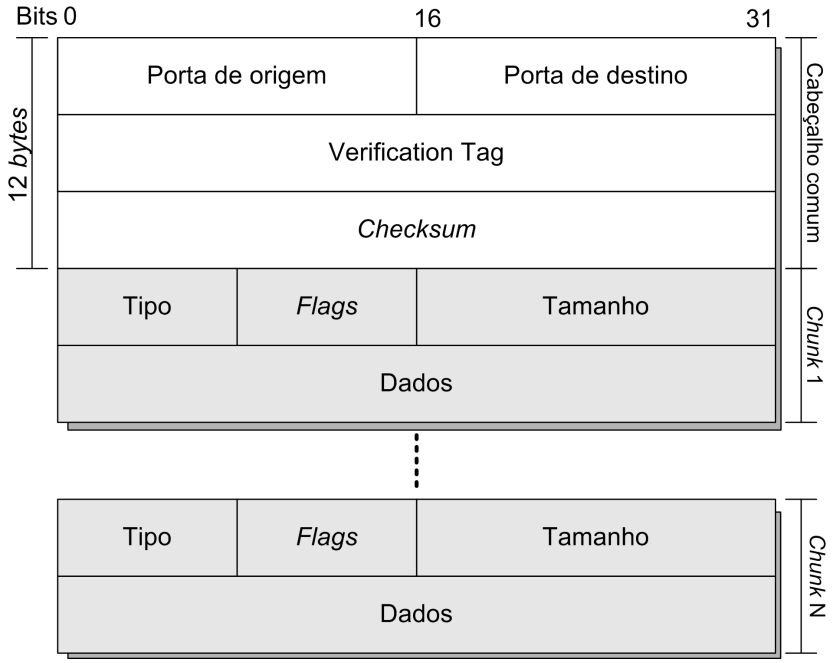


Figura 3.7: Formato do pacote SCTP

O cabeçalho comum consiste de 12 *bytes*. Para a multiplexação e demultiplexação o SCTP utiliza os mesmos mecanismos do TCP e UDP, a partir dos campos "porta de origem" e "porta de destino". Para a detecção de erros de transmissão, cada pacote SCTP é protegido com um valor de soma de verificação de 32 *bits* (*Adler-32 algorithm*).

O cabeçalho comum possui ainda um campo chamado "*Verification tag*". Esse campo é específico para cada associação SCTP[3], e eles são trocados entre os sistemas

seqüência.

[3]No SCTP o relacionamento entre os pares é denominado associação, em vez de conexão, como no TCP,

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo