

Apostila de Rede de Computadores

Universidade Federal de Itajubá

**Instituto de Engenharia de Sistemas e Tecnologias da
Informação**

Grupo de Pesquisas em Engenharia de Sistemas e Computação

Prof. Isaías Lima

Prefácio

Esta apostila dedica-se aos cursos de Redes de Computadores I e II ministrados na Universidade Federal de Itajubá, cujos capítulos e assuntos tratados nas seções seguem exatamente o livro texto referenciado no plano de ensino, e que serviu de base para uma tradução livre.

Portanto, dentro das expectativas de um curso introdutório ou de nível mais avançado, algumas incorreções são meramente erros de compreensão ou entendimento do autor. Assim, numa tentativa de apresentar um conteúdo abrangente e necessariamente direto, procurou-se descrever da melhor forma possível, os termos que por ora são correntes em Rede de Computadores em geral.

De outra forma, apesar de um nível relativamente satisfatório, uma tradução livre nem sempre se prende rigorosamente ao conteúdo explícito da obra; cabendo a ressalva de alguns questionamentos em relação à estrutura das seções, os termos técnicos e as descrições, bem como ao próprio texto traduzido dos parágrafos e seções.

Fica ainda a necessidade de recorrência ao livro original em inglês, cujo acompanhamento é recomendado desde o início do curso, e consta das referências bibliográficas da ementa de Redes de Computadores I e II.

Sumário

Capítulo 1 - Meios de Transmissão	10
1.1 Introdução.....	10
1.2 Os Fios de Cobre	10
1.3 Fibras Ópticas.....	11
1.4 Rádio-Frequência	11
1.5 Satélites.....	12
1.5.1 Satélites Geo-sincronizados (GEOS).....	12
1.5.2 Satélites de órbita baixa (LEOS).....	12
1.6 Conjuntos de satélites de órbita baixa	13
1.7 Microondas	13
1.8 Infravermelho	13
1.9 Laser	13
Capítulo 2 – Comunicação Local Assíncrona	14
2.1 Introdução.....	14
2.2 A necessidade para comunicação assíncrona	14
2.3 Usando a corrente elétrica para enviar dados.....	14
2.4 Comunicações Padrões.....	15
2.5 Taxa de Envio, Framing, e Erros	16
2.6 Comunicação assíncrona Full Duplex.....	16
2.7 Limitações de um Hardware real.....	17
2.8 Largura de banda de um Hardware e transmissão de bits	17
2.9 O efeito dos ruídos na comunicação.....	18
2.10 Importância para Redes de Dados	18
Capítulo 3 – Tecnologia de LANs e Topologias de Rede	19
3.1 Introdução.....	19
3.2 Conexão direta Ponto a Ponto	19
3.3 Canais de Comunicação Compartilhados.....	20
3.4 Importância das LANs e Localização de Referência	20
3.5 Topologia de LANs.....	21
3.5.1 Topologia de Estrela	21
3.5.2 Topologia de Anel.....	21
3.5.3 Topologia de Barramento	22
3.5.4 A Razão para Várias Topologias	22
3.6 Exemplo da rede em barramento: Ethernet.....	23
3.6.1 História da Ethernet	23
3.6.2 Compartilhamento em uma Ethernet.....	23
3.7 Portador de Lógica em uma Rede de Múltiplos Acessos (CSMA).....	23
3.8 Detecção de Colisões e Impedimento com CSMA/CD	24
3.9 LANs Wireless e CSMA/CA	24
3.10 Outro Exemplo de Rede de Barramento: LocalTalk.....	25
3.11 Exemplo de Rede em Anel: IBM Token Ring	26
3.12 Outro Exemplo de Rede em Anel: FDDI	26
3.13 Exemplo de Rede em Estrela: ATM.....	27
Capítulo 4 – Endereçamentos de Hardware e Identificação do Tipo de Frame	29
4.1 Introdução.....	29
4.2 Especificando um Receptor	29
4.3 Como o Hardware de uma LAN usa o Endereço para Filtrar os pacotes.....	29
4.4 Formato de um Endereço Físico.....	30
4.5 Broadcasting	31
4.6 Multicasting	31
4.7 Multicasting Addressing	32
4.8 Identificando o Conteúdo dos Pacotes	32
4.9 Cabeçalho de Frames e Formato de Frames	33
4.10 Um exemplo da Formatação de Frames	33
4.11 Usando Redes que não Auto-Identificam Frames.....	34

4.12 Análise de Rede, Endereços Físicos, Tipo de Frames	35
Capítulo 5 – Cabeamentos de LAN, Topologia Física e Interface de Hardware	37
5.1 Introdução.....	37
5.2 Velocidades de LANs e Computadores.....	37
5.3 Interface de Rede.....	37
5.4 Conexão entre NIC e Rede.....	38
5.5 Thick Ethernet.....	38
5.6 Multiplexador de Conexão.....	39
5.7 Thin Ethernet.....	40
5.8 Par Trançado	41
5.9 Vantagens e Desvantagens dos Sistemas de Cabeamento	41
5.10 Placas de Rede e Esquemas de Cabeamento	43
Capítulo 6 – Extensão das LANs	44
6.1 Introdução.....	44
6.2 Limitação pela Distância e Design de LAN	44
6.3 Extensões de Fibra Óptica	44
6.4 Repetidores	45
6.5 Bridges.....	45
6.6 Filtro de Frames	46
6.7 Startup e Steady State de Redes com Bridge.....	46
6.8 Bridge entre Construções.....	47
6.9 Bridges em Redes de Longa Distância	47
6.10 Bridges em Círculos	48
6.11 Switching.....	49
Capítulo 7 – WAN e Roteadores.....	50
7.1 Introdução.....	50
7.2 Grandes Redes e Grandes Áreas.....	50
7.3 Packet Switch.....	50
7.4 Formando uma WAN.....	51
7.5 Store e Forward	51
7.6 O Endereço Físico.....	52
7.7 Next-Hop Forward	52
7.8 Relação de Endereçamento Hierárquico para o Roteador.....	53
7.9 Roteamento numa WAN.....	53
7.10 Uso de Rotas Padrão	54
7.11 Formando a Routing Table.....	55
7.12 Calculando o Caminho mais Curto.....	55
7.13 Rota de Distância Vetorial.....	56
7.14 Link-State Routing (SPF)	56
7.15 Exemplos de Tecnologias WAN.....	56
Capítulo 8 – Proprietário da Rede, Padrão de Serviços, e Desempenho	58
8.1 Introdução.....	58
8.2 Proprietário da Rede.....	58
8.3 Virtual Private Networks.....	59
8.4 Padrão de Serviço.....	59
8.5 Duração de Conexões e Persistência	60
8.6 Endereços e Identificadores de Conexão	61
8.7 Características do Desempenho de Redes	62
8.7.1 Delay.....	62
8.7.2 Ritmo de Transferência	62
8.7.3 A relação entre Delay e Ritmo de Transferência.....	63
8.7.4 Produto do Delay - Ritmo de Transferência	63
Capítulo 9 – Protocolos e Divisão em Camadas	64
9.1 Introdução.....	64
9.2 A Necessidade dos Protocolos.....	64
9.3 Coleções de Protocolos	64
9.4 Um Plano para o Projeto de Protocolos.....	64
9.5 As sete Camadas	65
9.6 Pilhas: Software em Camadas	66
9.7 Como Softwares de Camadas Funcionam	67
9.8 Cabeçalhos Abrigados Multiplamente.....	67
9.9 A Base Científica para a Divisão em Camadas	68

9.10 Técnicas Usadas nos Protocolos	69
9.10.1 Sequência de Entrega Fora da Ordem	69
9.10.2 Sequência para Eliminar Pacotes Duplicados.....	70
9.10.3 Retransmissão de Pacotes Perdidos	70
9.10.4 Evitando Replay Causado pelo Delay Excessivo.....	70
9.10.5 Controle de Fluxo para Evitar o Excesso de Dados.....	70
9.10.6 Mecanismos para Evitar Congestionamento da Rede.....	73
Capítulo 10 – Internet: Conceitos, Arquitetura e Protocolos.....	74
10.1 Introdução.....	74
10.2 A Motivação para Estudar Internet.....	74
10.3 O Conceito de Universal Service	74
10.4 Serviço Universal em um Mundo Heterogêneo.....	74
10.5 Internet (entre Redes)	74
10.6 Redes Conectadas com Roteadores.....	75
10.7 Arquitetura de Internet	75
10.8 Realizando o Universal Service	75
10.9 A Rede Virtual.....	76
10.10 Protocolos para Internet	77
10.11 Camadas e Protocolos TCP/IP	77
Capítulo 11 – IP: Internet Protocol Addresses	79
11.1 Introdução.....	79
11.2 Endereços para a Internet Virtual	79
11.3 O Esquema de Endereçamento do IP	79
11.4 Hierarquia no IP.....	79
11.5 Classes dos Endereços de IP	80
11.6 Computando a Classe de um Endereço	80
11.7 Notação Decimal de Ponto	81
11.8 Endereços Especiais de IP.....	83
11.8.1 Endereços de Rede	83
11.8.2 Endereço de Transferência Direta	83
11.8.3 Endereço de Transferência Limitada	83
11.8.4 This Computer Address.....	83
11.8.5 Endereço de Loopback.....	83
11.9 Resumo dos Endereços Especiais	84
11.10 Roteadores e o Princípio do Endereçamento de IP	84
11.11 Multi-Homed Hosts	85
Capítulo 12 – Binding Protocol Address.....	86
12.1 Introdução.....	86
12.2 Protocolo de Endereçamento e Entrega de Pacotes	86
12.3 Determinação dos Endereços.....	86
12.4 Técnicas de Determinação de Endereços.....	87
12.5 Determinação de Endereços com Busca em Tabela.....	87
12.6 Determinação de Endereços com Computação de forma fixa	88
12.7 Determinação de Endereços com Troca de Mensagens.....	88
12.8 Protocolo de Determinação de Endereços	89
12.9 Entrega da Mensagem ARP.....	89
12.10 Formato da mensagem ARP	90
12.11 Enviando a mensagem ARP.....	91
12.12 Identificando a mensagem ARP	91
Capítulo 13 – Datagramas IP e Encaminhamento de Datagramas.....	92
13.1 Introdução.....	92
13.2 Serviço sem conexão.....	92
13.3 Pacotes virtuais	92
13.4 O Datagrama IP.....	93
13.5 Encaminhando um datagrama IP	93
13.6 Endereços IP e entradas de tabelas de roteamento.....	94
13.7 O campo Máscara e Encaminhamento de datagramas	94
13.8 Destino e Endereços Próximo salto	95
13.9 Entrega a qualquer custo.....	95
13.10 Formato do cabeçalho IP	96
Capítulo 14 – Encapsulamento IP, Fragmentação e Reunião	97

14.1	Introdução.....	97
14.2	Transmissão de Datagramas e Frames	97
14.3	Encapsulamento	97
14.4	Transmissão através da Internet.....	98
14.5	MTU, Tamanho do Datagrama, e Encapsulamento	98
14.6	Reunião.....	100
14.7	Identificando um Datagrama	100
14.8	Perda de Fragmento.....	100
14.9	Fragmentando um Fragmento	101
Capítulo 15 – O IP do futuro (Ipv6)		102
15.1	Introdução.....	102
15.2	O sucesso do IP	102
15.3	A motivação da mudança.....	102
15.4	Nome e Número de Versão	102
15.5	Caracterização dos atributos no IPv6.....	103
15.6	Formato do Datagrama do IPv6.....	103
15.7	Formato do Cabeçalho Base do IPv6	104
15.8	Como o IPv6 lida com múltiplos Cabeçalhos	105
15.9	Fragmentação e <i>path MTU</i>	106
15.10	O Propósito de Múltiplos Cabeçalhos.....	107
15.11	Endereçamento no IPv6.....	107
15.12	Notação Hexadecimal do IPv6.....	108
Capítulo 16 – Capítulo 21 – Mecanismo de informação de erro (ICMP).....		109
16.1	Introdução.....	109
16.2	Semânticas “A Qualquer Custo” e Detecção de Erro.....	109
16.3	Protocolo de Mensagem de Controle de Erro	109
16.4	Transporte da mensagem ICMP	111
16.5	Utilizando as mensagens ICMP para testar o alcance	112
16.6	Utilizando ICMP para traçar uma Rota.....	112
16.7	Utilizando ICMP para descobrir o MTU do Caminho.....	113
Capítulo 17 – TCP: Serviço de Transporte Confiável		114
17.1	Introdução.....	114
17.2	A necessidade de um transporte confiável.....	114
17.3	O Protocolo de Controle de Transmissão (TCP).....	114
17.4	O Serviço que o TCP fornece aos Aplicativos	114
17.5	Serviço Ponto a Ponto e Datagramas	115
17.6	Alcançando Confiança	116
17.7	Perca de Pacotes e Retransmissão	116
17.8	Retransmissão Adaptativa	117
17.9	Comparação entre Tempos de Retransmissão	117
17.10	Buffers e Controle de Fluxo	118
17.11	Three-Way Handshake	119
17.12	Controle de Congestionamento	120
17.13	Formato do Segmento TCP	120
Capítulo 18 – Interação Cliente-Servidor.....		122
18.1	Introdução.....	122
18.2	As Funcionalidades que o Software de Aplicação Fornece	122
18.3	As Funcionalidades Providas pela Internet.....	122
18.4	Fazendo Contato.....	123
18.5	O Paradigma Cliente-Servidor	123
18.6	Características de Clientes e de Servidores	123
18.7	Programas de Servidores e Computadores da classe de Servidores.....	124
18.8	Pedidos, Respostas e Direção do Fluxo de Dados.....	124
18.9	Protocolos de Transporte e Interação Cliente-Servidor	124
18.10	Múltiplos Serviços em um Computador	125
18.11	Identificando um Serviço Particular	126
18.12	Múltiplas Cópias de um Servidor para um só Serviço	126
18.13	Criação Dinâmica de Servidor	126
18.14	Protocolos de Transporte e Comunicação Não Ambígua.....	127
18.15	Conexão-Orientada e Transporte Sem Conexão.....	127
18.16	Um Serviço Através de Vários Protocolos	128
18.17	Interações Complexas Cliente-Servidor	128
18.18	Interações e Dependências Circulares	128
Capítulo 19 – A Interface Socket.....		130

19.1 Introdução.....	130
19.2 Application Program Interface	130
19.3 O Socket API.....	130
19.4 Sockets and Socket Libraries.....	131
19.5 Comunicação Socket e UNIX I/O.....	131
19.6 Sockets, Descriptors, e Network I/O	131
19.7 Parâmetros e o Socket API	132
19.8 Procedimentos do Socket API	132
19.8.1 O Procedimento Socket.....	132
19.8.2 O Procedimento Close.....	132
19.8.3 O Procedimento Bind.....	133
19.8.4 O Procedimento Listen	134
19.8.5 O Procedimento Accept	134
19.8.6 O Procedimento Connect.....	135
19.8.7 Os Procedimentos Send, Sendto e Sendmsg	135
19.8.8 Os Procedimentos Recv, Recvfrom e Recvmsg	136
19.9 Lendo e Escrevendo com Sockets.....	137
19.10 Sockets, Processos e Heranças.	137
Capítulo 20 – Exemplo de um Cliente e Servidor	138
20.1 Introdução.....	138
20.2 Comunicação com Conexão Orientada.....	138
20.3 Um exemplo de Serviço.....	138
20.4 Argumentos das Linhas de Comandos deste Exemplo de Programa	138
20.5 Sequência de Processos no Socket	139
20.6 Exemplo de Código para Cliente.....	140
20.7 Exemplo de Código para o Servidor	142
20.8 Serviço Stream e Múltiplas Chamadas Recv.....	143
20.9 Processos do Socket e Bloqueios.....	143
20.10 Tamanho do Código e Relatório de Erros	144
20.11 Usando o Exemplo de Cliente com outro Serviço	144
20.12 Utilizando Outro Cliente para Testar o Servidor	145
Capítulo 21 – Nomeando com o Domain Name System.....	146
21.1 Introdução.....	146
21.2 Estrutura de Nomes de Computadores.....	146
21.3 Estrutura Geográfica	147
21.4 Nomes de Domínios dentro de uma Organização	148
21.5 O Modelo Cliente-Servidor do DNS.....	148
21.6 A Hierarquia de Servidores DNS	149
21.7 Arquiteturas de Servidores.....	149
21.8 Localidade de Referência e Múltiplos Servidores.....	150
21.9 Links entre Servidores	150
21.10 Decidindo um Nome	150
21.11 Otimização do Desempenho do DNS.....	151
21.12 Tipos de Entradas do DNS.....	151
21.13 Apelidos Utilizando o Tipo CNAME.....	152
21.14 Uma Consequência Importante de Tipos Múltiplos	152
21.15 Abreviações e o DNS	153
Capítulo 22 – Descrição de Correio Eletrônico e Transferências	154
22.1 Introdução.....	154
22.2 O Paradigma do E-mail	154
22.3 Caixas de E-mail e Endereços.....	154
22.4 Formato da Mensagem de E-mail	155
22.5 Extensões do E-mail com Múltiplos Propósitos	155
22.6 E-mail e Aplicativos.....	156
22.7 Transferência de E-mail.....	157
22.8 Simple Mail Transfer Protocol.....	157
22.9 Otimização de Múltiplos Destinatário em um Computador	157
22.10 Listas e Remetentes de E-mails	158
22.11 Mail Gateways	158
22.12 Transmissão de E-mail e Endereços de E-mail	159
22.13 Acesso a Caixa de Mensagens.....	160
22.14 Conexão Dial-up e POP.....	161
Capítulo 23 – Transferência de arquivos e Acesso Remoto de arquivos	162

23.1 Introdução.....	162
23.2 Transferência de Dados e Processamento Distribuído	162
23.3 Salvando Resultados Intermediários	162
23.4 Generalização da Transferência de Arquivos	163
23.5 Interatividade e Conjunto de Paradigmas da Transferência.....	163
23.6 O Protocolo de Transferência de Arquivos (FTP)	164
23.7 Modelo Geral do FTP e Interface com o Usuário	164
23.8 Comando FTP.....	164
23.9 Conexões, Autorizações e Permissões de Arquivos.....	165
23.10 Acesso Anônimo de Arquivos	166
23.11 Transferência de Arquivos em Ambas as Direções.....	166
23.12 Expansão por Caractere Curinga em Nomes de Arquivos	167
23.13 Tradução de Nomes de Arquivos	167
23.14 Mudando de Diretório e Listando o Conteúdo.....	167
23.15 Tipos de Arquivos e Modos de Transferência	168
23.16 Exemplos Utilizando FTP	168
23.17 Verbose Output.....	168
23.18 Interações Cliente-Servidor no FTP	169
23.19 Conexões de Controle e de Dados.....	169
23.20 Conexões de Dados e Fim do Arquivo	169
23.21 Protocolo de Transferência de Arquivos Trivial.....	170
23.22 Sistema de Arquivos de Rede	170
Capítulo 24 – Páginas da World Wild Web e Navegação.....	172
24.1 Introdução.....	172
24.2 Interface de Browser	172
24.3 Hipertexto e Hipermídia	172
24.4 Representação de Documentos	173
24.5 Formato e Representação HTML	173
24.6 Exemplo de Sinalizadores de Formatação HTML.....	174
24.7 Títulos.....	175
24.8 Listas	175
24.9 Imagens Gráficas Colocadas em uma Página da Web	175
24.10 Identificando uma Página	176
24.11 Links em Hipertextos de um Documento para Outro	177
24.12 Interação Cliente-Servidor	177
24.13 Transporte de Documentos Web e HTTP	178
24.14 Arquitetura do Browser.....	178
24.15 Clientes Opcionais	179
24.16 Armazenamento em Cache nos Browsers Web.....	180
Capítulo 25 – RPC e Middleware.....	182
25.1 Introdução.....	182
25.2 Programando Clientes e Servidores.....	182
25.3 Paradigma de Chamada de Rotinas Remotas	182
25.4 Paradigma RPC.....	184
25.5 Stubs da Comunicação	185
25.6 Representação de Dados Externos	186
25.7 Middleware e Middleware com Orientação a Objeto.....	186
25.7.1 ONC RPC	186
25.7.2 DCE RPC	187
25.7.3 MSRPC	187
25.7.4 CORBA	187
25.7.5 MSRPC2	187
25.7.6 COM/DCOM	188

Capítulo 26 – Tecnologia CGI para Documentos Dinâmicos Web	189
26.1 Introdução.....	189
26.2 Três Tipos Básicos de Documentos Web	189
26.3 As Vantagens e Desvantagens de Cada Tipo de Documento.....	190
26.4 Implementação de Documentos Dinâmicos	191
26.5 O Padrão CGI.....	191
26.6 Resposta de um Programa CGI	191
26.7 Exemplo de um Programa CGI.....	192
26.8 Parâmetros e Variáveis Globais	193
26.10 O Script CGI com Informação de Estado de Longo Termo	194
26.11 O script CGI com Informação de Estado de Curto Termo.....	195
26.12 Formulário e Interações.....	197
Capítulo 27 – Tecnologia Java para Documentos Ativos Web.....	198
27.1 Introdução.....	198
27.2 Uma maneira precoce de Atualizações Contínuas	198
27.3 Documentos Ativos e Sobrecarga no Servidor	199
27.4 Representação e Tradução de um Documento Ativo.....	199
27.5 Tecnologia Java	200
27.6 A Linguagem de Programação Java	201
27.6.1 Características da Linguagem.....	201
27.6.2 Similaridades ao C++.....	202
27.7 O Ambiente do Tempo de Execução no Java	202
27.8 A Biblioteca Java.....	203
27.9 O Conjunto de Ferramentas Gráficas	204
27.10 Usando os Gráficos Java em um Computador Particular.....	205
27.11 Interpretes Java e Navegadores	206
27.12 Compilando um Programa Java	206
27.13 Um Exemplo de Applet	207
27.14 Chamando um Applet.....	209
27.15 Exemplo da Interação com o Navegador	209
27.16 Erros e Lidar com Exceções	211
27.17 Alternativas e Variações	212
Capítulo 28 – Administração de Rede (SNMP)	213
28.1 Introdução.....	213
28.2 Administrando uma Internet.....	213
28.3 O Perigo de Falhas Ocultas.....	213
28.4 Software de Administração de Rede.....	214
28.5 Clientes, Servidores, Administradores e Agentes	214
28.6 Protocolo de Administração de Rede Simples.....	214
28.7 Paradigma Fetch-Store	215
28.8 O MIB e Nomes de Objetos	216
28.9 A Variedade de Variáveis MIB	216
28.10 Variáveis MIB Que Correspondem a Vetores.....	216
Capítulo 29 – Segurança de Rede.....	218
29.1 Introdução.....	218
29.2 Redes Seguras e Políticas de Segurança	218
29.3 Aspectos da Segurança.....	219
29.4 Responsabilidade e Controle	219
29.5 Mecanismos de Integridade	219
29.6 Controle de Acesso e Senhas.....	220
29.7 Encriptação e Privacidade	220
29.8 Encriptação por Chave Pública.....	220
29.9 Autenticação com Assinaturas Digitais.....	221
29.10 Filtragem de Pacotes	222
29.11 Conceito de Firewall de Internet	223

Capítulo 1 - Meios de Transmissão

1.1 Introdução

Pode-se dizer que toda comunicação entre computadores envolve codificação de informações em uma troca de energia e transferir essa energia através de um meio. Assim, neste capítulo será discutido o conceito básico da transmissão de dados, examinando os meios de transmissão que são utilizados em modernos sistemas de redes de computadores.

1.2 Os Fios de Cobre

As redes de computadores tradicionais usam fios como meio principal de conexão entre computadores, devido ao baixo custo e a facilidade de instalação. Além disso, os fios podem ser feitos de diferentes metais, e muitas redes utilizam o cobre por ter baixa resistência elétrica, o que significa que um sinal pode ser enviado a uma distância muito maior.

O tipo de fiação utilizado em redes de computadores é escolhido de maneira que a interferência seja a mínima possível. A interferência tem origem quando um sinal elétrico é transmitido em um cabo, e este atua como uma pequena estação de rádio (o fio emite uma pequena quantidade de energia eletromagnética que se propaga no ar). Essa energia ao encontrar outro fio, produz uma pequena corrente elétrica. O somatório das correntes geradas depende da potência das ondas eletromagnéticas e da posição do fio. Normalmente, essa interferência só passa a ser detectada quando dois fios são colocados lado a lado e em paralelo, assim um sinal elétrico enviado em um fio, pode gerar um sinal similar em outro. Os computadores não conseguem distinguir sinais gerados acidentalmente, assim são tomadas algumas iniciativas para reduzir a interferência.

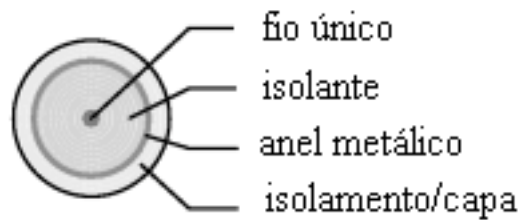
Na tentativa de minimizar a interferência são usados dois tipos básicos de fiação: o par trançado e o cabo coaxial. O par trançado também é usado em sistemas de telefonia.

O simples trançar dos cabos altera as propriedades elétricas dos fios, ajudando a torná-lo adequado ao uso em uma rede. Primeiro, porque limita a energia eletromagnética que o próprio fio emite. Segundo, porque os torna menos susceptível à energia eletromagnética emitida por outros fios. A figura a seguir ilustra um fio tipo par trançado.



O fio coaxial apresenta uma maior proteção que o par trançado, em vez de trançar os fios, é utilizado um anel metálico em volta de um núcleo, um único fio.

Este anel de metal forma um cilindro flexível em volta do fio isolado, o que gera uma barreira à radiação eletromagnética. Esta barreira protege o fio de duas maneiras, evitando que o mesmo gere radiação eletromagnética, e evitando que a radiação eletromagnética gerada por outros fios interfira no mesmo. Sua eficácia se deve ao fato de o anel cercar o fio por todos os lados. Assim o cabo coaxial pode ser posto em paralelo com outros cabos. A figura abaixo ilustra o escudo e o cabo coaxial.



A idéia de um escudo para proteger a fiação também foi aplicada ao par trançado. Um *par trançado protegido* consiste em um par de fios blindados por uma metálica. Cada fio é protegido por um material isolante, evitando que um toque no outro. Este tipo adicional de proteção é utilizado em cabos que estejam perto de equipamentos que tenham campo magnético ou elétrico intensos.

1.3 Fibras Ópticas

As redes de computadores também utilizam as fibras ópticas para transmitirem dados, o meio de transporte do dado é a luz. O transmissor em uma ponta do cabo é um LED (light emitting diode) ou um laser, que envia pulsos de luz; já na recepção de dados é usado um transistor foto-sensível.

As fibras ópticas têm quatro vantagens em relação aos fios. Primeiro, a luz não produz, nem sofre interferência elétrica. Segundo, devido a sua fabricação, os feixes de luz podem ser levados muito mais longe do que um sinal em um fio de cobre. Terceiro, a luz pode codificar mais informações do que sinais elétricos. Quarto, não necessitam de mais fios para completar um circuito elétrico, pois apenas um fio de fibra óptica é necessário para interligar dois computadores.

Porém, essas mesmas fibras têm algumas desvantagens, como a dificuldade de instalação, pois é necessário um equipamento especial para polir a extremidade do cabo; permitindo que a luz passe por ele. E, depois, se um cabo de fibra óptica se rompe, é difícil de difícil localização, além disso, para reparar um fio rompido é necessário também um equipamento especial para unir as duas fibras.

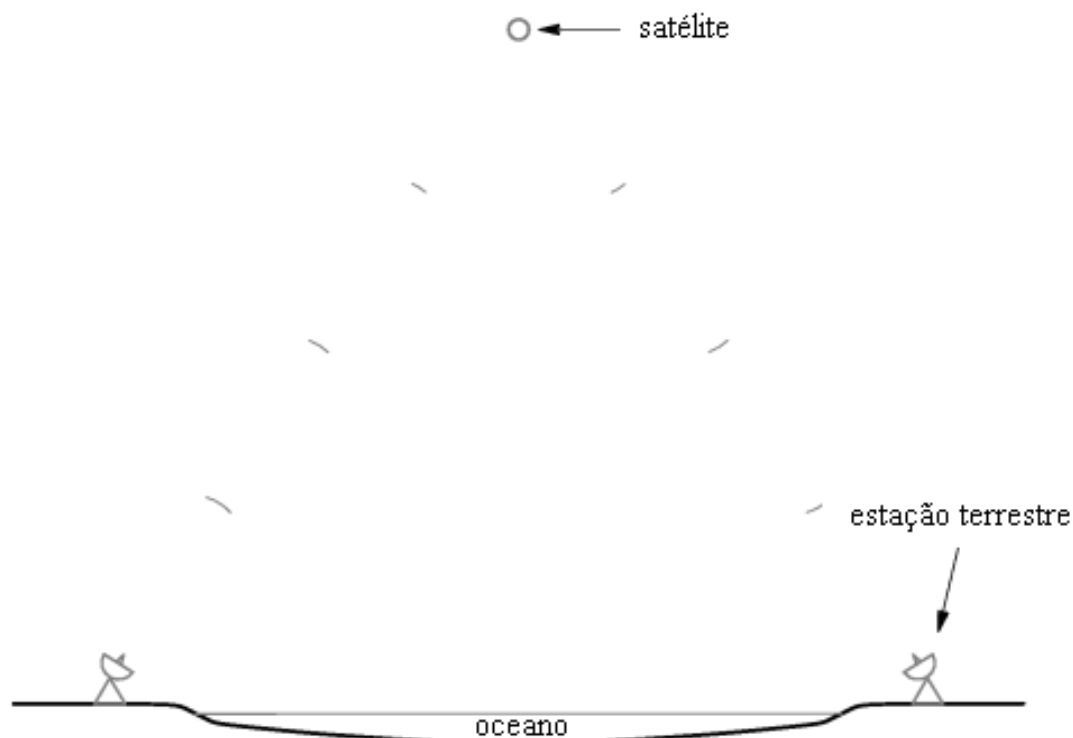
1.4 Rádio-Freqüência

Além de ser usada na transmissão de radio, televisão e comunicação privada entre walk-talks, a radiação eletromagnética pode ser usada para transmitir dados. As redes que usam ondas de radio são feitas para trabalharem em uma determinada radio-freqüência, e as transmissões são referidas como transmissões RF. Diferente de redes que usam fibra óptica, redes usando transmissões em RF não necessitam de um meio físico que conecte os computadores. Assim cada computador é conectado a uma antena, que pode mutuamente transmitir e receber RF.

Fisicamente, as antenas usadas para transmissão de RF podem ser grandes ou pequenas, dependendo do alcance necessário. Por exemplo, uma antena projetada para transmitir sinais a muitos quilômetros deve ter uma altura de um pólo metálico com dois metros de altura, que é montado no topo de um edifício. Já uma antena para permitir comunicação entre dois prédios é tão pequena que cabe em um computador portátil.

1.5 Satélites

Transmissões de RF nem sempre alcançam todo território na superfície da terra, assim podem ser combinadas com satélites, que proverão uma comunicação a grandes distâncias. Como mostrado na figura abaixo, a transmissão de dados entre um oceano:



O satélite contém um aparelho chamado transponder, que consiste em um transmissor e receptor de RF. Esse transponder aceita a transmissão de RF, amplifica a mesma, depois transmite em um ângulo diferente do de chegada. Assim podem ser feitas comunicações a grandes distâncias, como entre um continente e outro. Uma das desvantagens do uso de satélites é o custo elevado para se colocar um satélite em órbita, assim quando se coloca um satélite, normalmente este tem 6 ou 12 transponders, que trabalham em múltiplas conexões, pois estão em frequências diferentes.

1.5.1 Satélites Geo-sincronizados (GEOS)

A comunicação entre satélites pode ser classificada a partir da altura da órbita de cada um. Os satélites chamados de geo-sincronizados, ou estacionários, são colocados em uma determinada órbita que quando vistos do chão, permanecem no mesmo local, ou seja, sua rotação está sincronizada com a da terra.

As leis de Kepler determinam a altura necessária para que um satélite permaneça sincronizado com a órbita terrestre. Essa distância é de aproximadamente de 60.000 quilômetros. Em termos técnicos, essa distância é chamada de órbita terrestre elevada.

Porém há um espaço limitado no espaço, pois deve ser evitada a interferência entre satélites. Geralmente o espaço tomado entre eles, no equador, deve ser de 4° a 8°, assim, como a terra tem 360°, cabem aproximadamente 45 a 90 satélites.

1.5.2 Satélites de órbita baixa (LEOS)

Uma segunda categoria de satélites são aqueles chamados de satélites de órbita baixa, ou seja, que estão entre 320 e 640 milhas da Terra, assim seu período de rotação é maior do que

o da Terra, não permanecendo fixos em um determinado ponto do céu. De fato um satélite pode completar uma órbita inteira em aproximadamente 1,5 horas.

Suas desvantagens é que só pode ser usado durante o tempo que permanece entre duas estações, e depois para controlar um sistema que está em contínuo movimento, um complexo sistema de controle na terra é requerido.

1.6 Conjuntos de satélites de órbita baixa

Este esquema de comunicação consiste em permitir comunicação constante através de satélites em baixa órbita. Ao invés de um satélite ser focado, o esquema exige uma determinada quantidade de satélites, de modo que um ponto nunca fique sem nenhum satélite sobre ele.

Além de ter equipamentos para fazer a transmissão com as estações terrestres, estes satélites também têm um equipamento que permite a conexão entre um satélite e outro.

1.7 Microondas

Radiação eletromagnética que tem frequência além das usadas em radio e televisão chamada microondas, e também é utilizada para transmitir informações.

Ondas de microondas além de ter a frequência maior do que as ondas de rádio, também têm um comportamento diferente. Diferentemente das ondas de rádio que são transmitidas em todas as direções, as microondas podem ser atiradas em uma única direção, dificultando a intercessão do sinal. Porém as microondas não podem penetrar em metais, assim tem funcionamento melhor em locais limpos, sem a presença de vegetação alta nem edifícios. Como resultado, as torres de microondas são maiores que qualquer obstáculo e cada transmissor de microondas está direcionado um para o outro.

1.8 Infravermelho

Os controles remotos utilizados em aparelhos de som e televisão são um exemplo do uso de infravermelho para comunicação entre dispositivos eletrônicos. Porém ele é limitado a pequenas distâncias, e usualmente requer que seja apontado diretamente para o receptor. Infravermelho não é caro comparado a outros dispositivos, além do que não necessita de antena.

Redes de computador podem usar infravermelho para transmissão de dados. É possível, por exemplo, uma equipa em uma sala enorme com conexões de infravermelho conectarem se ao mesmo provedor e compartilhar dados. Infravermelho é conveniente para pequenos aparelhos porque oferece as vantagens das redes sem fio, sem necessidade de antenas.

1.9 Laser

Já foi mencionado que a luz pode ser um meio de comunicação via fibras

Ópticas. Como a comunicação de microondas, o laser consiste em duas localidades com um transmissor e um receptor. Este equipamento é fixo normalmente em uma torre e o transmissor de um está mirado no receptor do outro. O transmissor usa um laser para gerar o feixe de luz, por que um laser se manterá no foco durante a longa distância.

Como a transmissão de microondas, o feixe de laser deve viajar em uma linha reta e não deve ser bloqueado. Infelizmente o raio laser não consegue ultrapassar vegetação ou condições climáticas adversas, como fumaça ou neve, assim tem seu uso limitado.

Capítulo 2 – Comunicação Local Assíncrona

2.1 Introdução

Por serem equipamentos digitais, computadores utilizam dígitos binários para representar os dados. Assim, transmitir dados através de uma rede, de um computador para o outro, consiste em enviar bits através de um meio de comunicação.

Fisicamente, o meio para se enviar bits é o meio elétrico, ondas de radio ou feixes de luz. Neste capítulo será explicada uma dessas formas, corrente elétrica, que pode ser usada para transmitir dados à curta distância.

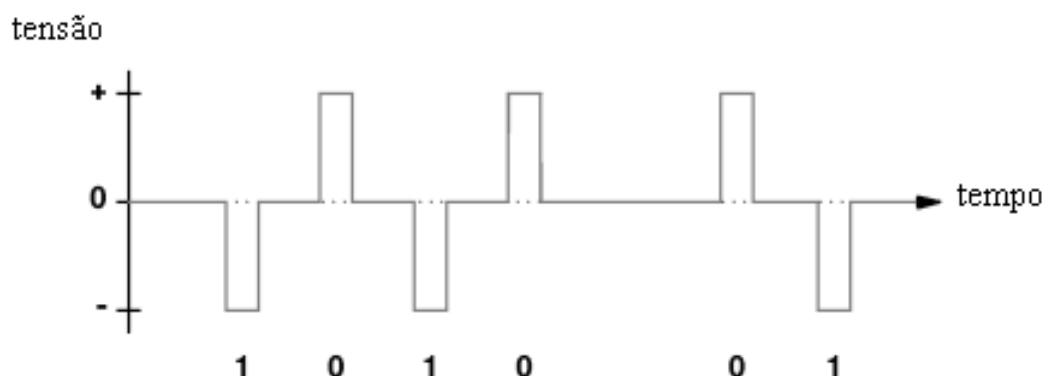
Além de discutir o básico da comunicação, serão introduzidas duas propriedades importantes de uma rede, a largura de banda e delay. Ambas podem ser quantizadas.

2.2 A necessidade para comunicação assíncrona

Uma comunicação é considerada assíncrona quando emissor e receptor não precisam se coordenar antes de dados serem transmitidos. Quando se usa comunicação assíncrona, o emissor pode esperar arbitrariamente para enviar dados, enquanto o receptor deve aceitá-los a qualquer momento que estes cheguem. Sistemas assíncronos são utilizados especialmente em equipamentos como teclado, onde os dados são gerados pelo toque humano.

2.3 Usando a corrente elétrica para enviar dados

O mais simples dos equipamentos eletrônicos utiliza uma pequena corrente para codificar dados. Para entender como a eletricidade pode codificar dados, imagine um fio que conecta dois equipamentos eletrônicos. Voltagens negativas são representadas por 1, e voltagens positivas por 0. Assim para enviar o bit 1, o equipamento emissor seta uma voltagem positiva, e depois volta para zero volt. Já para se enviar 0, o equipamento fornece uma tensão negativa, depois volta para o zero volt. A figura abaixo ilustra como a voltagem no fio varia com o tempo enquanto um aparelho envia uma sequência de bits.



2.4 Comunicações Padrões

Muitas questões sobre o envio de dados por meio de tensões elétricas ainda ficam confusas, como o tempo que deve durar uma voltagem para um simples bit? Qual é o tempo que o receptor se torna sensível à voltagem? Qual é a taxa máxima que o hardware consegue mudar a voltagem? Como o consumidor final irá saber que o hardware de uma marca se comunicará com o de outra marca?

Para certificar que a comunicação entre hardware de diferentes marcas não se torne inoperante, as especificações para comunicação foram padronizadas. Organizações como a União Internacional de Telecomunicações (ITU), Associação das Indústrias Eletrônicas (EIA), e Instituto de Elétrica e Engenharia Eletrônica (IEEE) publicam especificações para comunicação em documentos chamados padronizações.

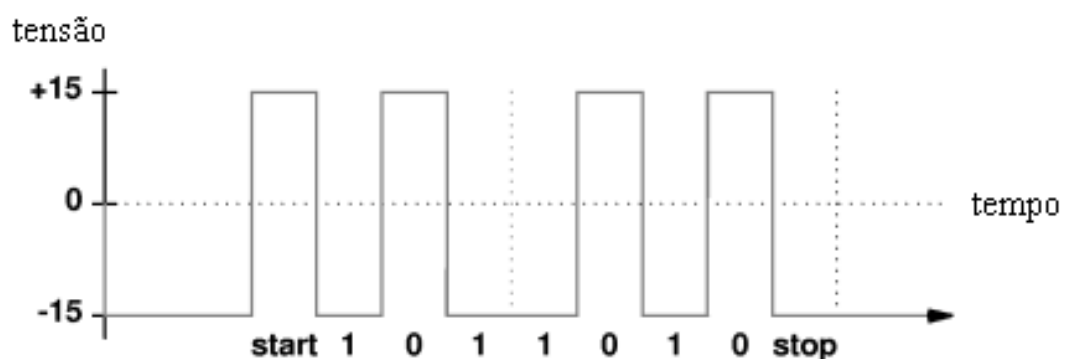
Documentos de Padronização respondem as questões sobre comunicação de tecnologias particulares. O padrão especifica tanto o tempo de sinal e os detalhes da voltagem e a corrente. Se duas marcas seguirem um mesmo padrão, os seus equipamentos irão comunicar entre si.

Um padrão particular produzido pela EIA emergiu como o mais usado e aceito na transmissão entre um computador e um teclado, modem ou terminal. O padrão RS-23-C, abreviado como RS-232 especifica os detalhes físicos da conexão, como os detalhes elétricos. Como o RS-232 é projetado para equipamentos como modem, e terminais, este especifica a transmissão de caracteres. Além do que este pode ser usado para transmitir caracteres de oito bits. RS 232 é configurado de tal maneira que cada caractere consiste em sete bits.

RS 232 define uma comunicação serial assíncrona, serial porque os bits são enviados em um único fio, um bit atrás do outro. RS-232 permite também que o emissor transmita um caractere a qualquer tempo, e que tenha tempo arbitrário para enviar outro caractere. Porém ao começar o envio de um caractere, o hardware envia um bit atrás do outro, sem nenhum delay entre eles. E o mais importante é que nesse padrão, a tensão no fio nunca será de zero volt, assim quando não há nada para se transmitir, o hardware deixa uma tensão negativa no fio que corresponde ao bit de valor 1.

Por causa de que o fio nunca tem a tensão de zero volt, o receptor não pode usar esta tensão para marcar o fim de um bit e o começo do próximo. Assim, tanto o emissor, quanto o receptor devem entrar em acordo sobre o tempo exato que cada voltagem será mantida para cada bit. Quando o primeiro bit chega, o receptor começa um timer e usa este tempo para saber qual será o tempo de todos os bits sucessivos. Assim RS-232 requer que o emissor envie o bit extra, 0 no caso, antes de começar a transmitir os bits de um caractere. Esse bit extra é chamado de start bit.

O período entre o fim de um caractere e o início de outro seria arbitrário, porém o RS-232 especifica que o emissor deve deixar o fio com a tensão negativa por um tempo mínimo. Esse tempo escolhido como mínimo é o tempo necessário para enviar um único bit. Assim podemos pensar que o bit 1 é um fantasma, e é fundido com cada caractere. Na terminologia do RS-232, o bit fantasma é chamado de bit de parada. Assim para transmitir um dado, são necessários 9 bits. A figura abaixo ilustra uma comunicação no padrão RS-232.



2.5 Taxa de Envio, Framing, e Erros

Ao invés de especificar o tempo necessário para cada bit, o que é uma pequena fração de segundo, os sistemas de comunicação especificam o número de bits que podem ser transferidos em um segundo. Nas conexões RS-232 são operadas comumente as velocidades de 19200 bits por segundo e 33600 bits por segundo.

Tecnicamente, os hardwares são medidos em taxas de envio (baud rate), que são o número de mudanças no sinal por segundo que o hardware gera. Assim uma taxa de 9600 significa 9600 bits por segundo.

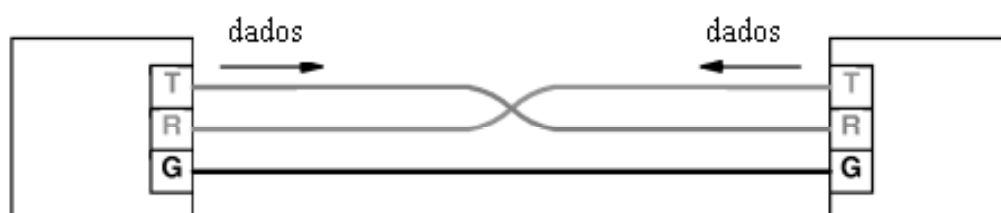
Para tornar o RS-232 um padrão mais geral, os fabricantes usualmente projetam cada peça do hardware para operar em uma variedade de taxas de envio (baud rate). A “baud rate” pode ser configurada manualmente ou automaticamente. Se o emissor e receptor não estiverem configurados na mesma taxa de envio, erros ocorrerão, porque o tempo do receptor não irá esperar o suficiente para ler cada bit, causando assim uma defasagem. Para detectar erros, um receptor mede diversas vezes a tensão em cada bit, e compara as medições. Se a voltagem não for unânime ou o bit de parada não ocorrer no instante esperado, o receptor acusa erro. Esses erros são chamados erros de **framing**, pois um caractere é como uma figura que não se encaixa em tamanho padrão de figuras.

O hardware do RS-232 pode fazer uso dos erros de framing. Em particular um teclado tem a tecla BREAK, essa tecla não gera nenhum caractere ASCII. Ao invés disso quando o usuário pressiona a tecla BREAK, o teclado coloca o bit 0 em muito mais tempo que seria necessário para enviar um único bit. Assim ao ser detectado que a voltagem tenha movido para o estado de 0 bit, o receptor assume que um caractere está chegando e fica esperando pelo bit de parada 1. Se este bit não chega, o receptor reporta um erro, o que pode abortar uma aplicação.

2.6 Comunicação assíncrona Full Duplex

Foi descrito que a corrente elétrica flui por um único cabo, porém todo circuito elétrico necessita de no mínimo de dois fios – a corrente flui em um e volta no outro. Este segundo fio é chamado de terra. Assim ao se usar o RS-232 em um par trançado, um fio carrega o sinal e o outro é o terra. Nos cabos coaxiais o sinal vai pelo cabo do centro, e o escudo é o terra.

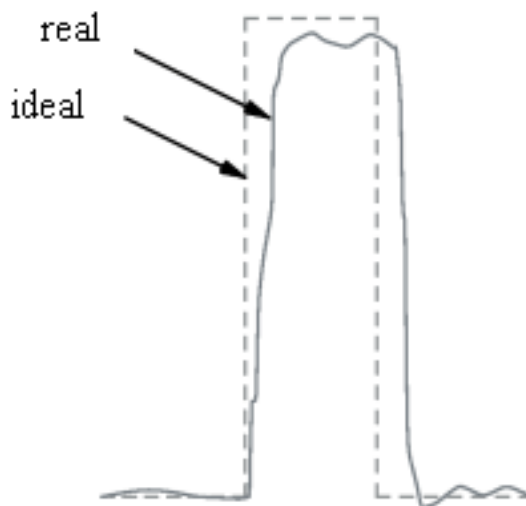
Em muitas aplicações do RS-232 os dados devem viajar em duas direções ao mesmo tempo. Esse tipo de transferência é conhecida como Full Duplex e são distinguidas das transmissões em única direção, conhecidas como Half Duplex. Para acomodar uma transmissão Full Duplex em RS-232, serão necessários três fios, uma para os dados enviados, um para os dados recebidos e o terra que funcionará em ambas as direções. De fato, o RS-232 define um conector DB-25 e especifica como o hardware irá usar os 25 fios para controlar dados. Por exemplo, enquanto o receptor está apto a receber dados, este receptor fornece uma voltagem em um fio de controle, que o emissor interpreta como livre para enviar. Para reduzir custos, os hardwares podem ser configurados para que ignorem os fios de controles e assumam que a outra parte está funcionando corretamente. A figura abaixo mostra um circuito de três fios.



Como a figura mostra, o fio terra conecta diretamente o terra de um com o do outro. Os outros dois fios estão cruzados porque o transmissor de um está ligado no receptor do outro. Para tornar os cabos mais simples, os projetistas decidiram que os computadores e modems devem usar pinos opostos do conector DB25. O computador transmite no fio 2 e recebe no fio 3, enquanto o modem transmite no fio 3 e recebe no fio 2.

2.7 Limitações de um Hardware real

Na prática, nenhum equipamento eletrônico consegue produzir uma voltagem exata ou mudar de uma voltagem para outra instantaneamente. Além do que, nenhum fio conduz eletricidade perfeitamente – ao passo que a corrente passa no fio, há perdas de energia. Com resultado, há um pequeno tempo para voltagem subir ou cair, e o sinal recebido não é perfeito. A figura abaixo ilustra como um bit realmente é e como ele deveria ser.



Como maioria das tecnologias de comunicação, RS-232 reconhece que os hardwares reais não são perfeitos, assim especifica a proximidade que uma forma emitida deve ser do padrão, e a tolerância sobre as imperfeições. Por exemplo, a padronização não especifica que o receptor deve medir a voltagem exatamente no começo de cada bit, apenas recomenda que se tirem amostras durante o meio do tempo de alocação de cada bit.

2.8 Largura de banda de um Hardware e transmissão de bits

Sabendo que um hardware não consegue mudar tensões instantaneamente, é explicado uma propriedade fundamental dos sistemas de transmissão, que é relacionada com a velocidade de bits que podem ser enviados. Cada sistema tem uma largura de banda limitada, que é a taxa máxima que o hardware pode modificar o sinal. Se o emissor tentar transmitir dados mais rápido do que a largura de banda, o hardware não conseguirá continuar porque não terá tempo suficiente para completar uma mudança de tensão antes de o emissor tentar fazer outra. Assim algumas das mudanças serão perdidas.

A largura de banda é medida em Hertz (Hz). Pode se pensar que a largura de banda é a oscilação de um sinal que foi enviado através de um hardware.

Na década de 1920 um pesquisador descobriu uma relação fundamental entre largura de banda de um sistema de comunicação e o máximo número de bits por segundo que podem ser transferidos. Conhecido como Teorema da Amostragem de Nyquist, a relação provém uma fronteira teórica para a máxima velocidade que dados podem ser enviados. O teorema mostra

que a máxima quantidade de dados a serem enviados é duas vezes a largura de banda do logaritmo na base de 2 do número de tensões trabalhadas. Matematicamente temos:

$$D = 2 \cdot B \cdot \log_2 K$$

Onde D = taxa de bits por segundo,

B = largura de banda

K = número de voltagens trabalhadas.

2.9 O efeito dos ruídos na comunicação

O teorema de Nyquist prove um máximo absoluto que pode ser alcançada na prática, engenheiros observaram que sistemas reais de comunicação estão sujeitos a uma pequena quantia de interferências, chamadas de ruídos, e estes ruídos fazem impossível alcançar a máxima taxa de transmissão teórica. Em 1948, Claude Shannon estendeu o teorema de Nyquist para especificar a máxima taxa de dados que podem ser alcançadas em um sistema de transmissão que introduz ruídos. O resultado é chamado de Teorema de Shannon, e é definido por:

$$C = B \log_2(1 + S/N)$$

Onde C é o limite efetivo da capacidade de um canal em bits por segundo, B é a largura de banda do hardware, S o valor médio do poder do sinal, N o valor médio do poder do sinal de ruídos.

Usualmente, S/N é conhecido como taxa de ruídos do sinal, e não é representado diretamente. Ao invés disso, engenheiros citam a quantidade $10 \log_{10} S/N$, que é medida em decibéis (abreviação dB).

2.10 Importância para Redes de Dados

Os teoremas de Nyquist e de Shannon têm conseqüências para engenheiros que projetam redes. O trabalho de Nyquist incentiva explorar maneiras complexas para codificar bits em sinais.

“O teorema de Nyquist encoraja engenheiros a explorar maneiras de codificar bits em um sinal porque uma codificação permite mais bits serem transmitidos por unidade de tempo.”

No mesmo senso, o Teorema de Shannon é mais fundamental, pois representa uma limitação absoluta, que é derivada das leis da física. Muito dos ruídos em uma linha de transmissão pode ser atribuído a termodinâmica. Assim:

“O Teorema de Shannon informa aos engenheiros que nenhuma solução inteligente para codificar dados pode sobrepor as leis da física, que colocam um limite fundamental no número de bits por segundo que podem ser transmitidos em um sistema real de comunicação”.

No senso pratico, o teorema de Shannon ajuda a explicar qual velocidade dados podem ser enviados através de uma ligação telefônica. A voz no telefone tem uma taxa de ruídos de sinal de aproximadamente 30dB e uma largura de banda de aproximadamente 3000 Hz. Assim, de acordo com o Teorema de Shannon, o número máximo de bits que podem ser transmitidos é de aproximadamente 30.000 bits por segundo. Engenheiros reconhecem este limite fundamental, transmissões mais rápidas irão apenas ser possíveis se a taxa de ruído seja aprimorada.

Capítulo 3 – Tecnologia de LANs e Topologias de Rede

3.1 Introdução

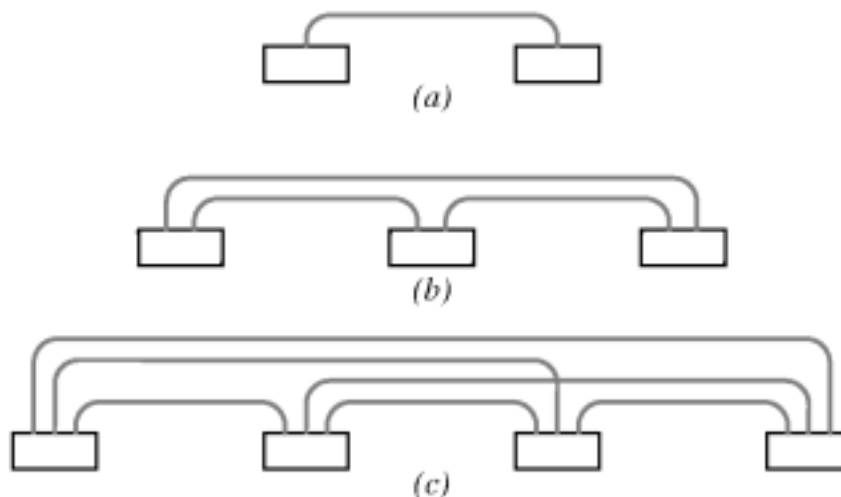
Pequenas redes são frequentemente projetados para permitir que múltiplos computadores compartilhem recursos como impressoras, arquivos e até mesmo conexões com a internet.

A tecnologia de hardware usada para redes locais não consiste em modems e cabos separados, ao invés disso a tecnologia é projetada para compartilhamento. Eles permitem que múltiplos computadores e equipamentos, como impressoras, sejam ligados diretamente como um só. Este capítulo irá descrever os conceitos que dão suporte as tecnologias de uma rede local, e explica porque as redes locais se tornaram tão populares.

3.2 Conexão direta Ponto a Ponto

Cada canal de comunicação conecta exatamente dois computadores, e só pode ser usado por esses dois computadores, exclusivamente. Conhecido como rede ponto a ponto, e tem três propriedades bem proveitosas. A primeira é que cada conexão é instalada independentemente das outras. Assim o equipamento de suporte não precisa ser o mesmo para todas as conexões. Segundo porque cada conexão tem conexão exclusiva, ou seja, os computadores decidem exatamente como enviar os dados pela conexão. Eles podem escolher o formato dos frames, um mecanismo para detectar erros, e o tamanho máximo de um frame. Por causa de cada conexão é independente das outras, os detalhes podem ser mudados sem que o administrador da rede concorde em fazer a mudança. Terceiro, porque apenas dois computadores terem acesso ao canal, é fácil reforçar a segurança e privacidade.

É claro que conexões ponto a ponto apresentam desvantagens, como há mais de dois computadores precisando se comunicar. A figura abaixo representa como o aumento de computadores compromete a conexão ponto a ponto.

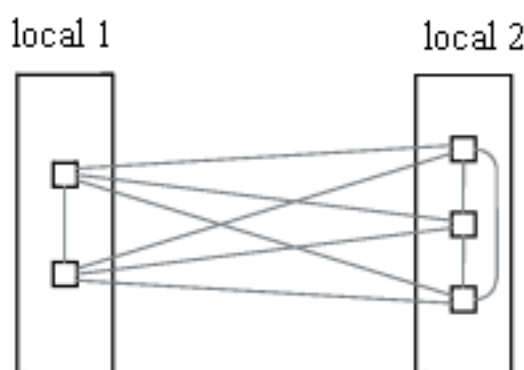


Matematicamente, o número de conexões é que é preciso para conectar N computadores é:

$$\text{Conexões necessárias} = (N^2 - N) / 2$$

Na prática, o custo é alto, pois muitas conexões seguem o mesmo caminho no chão, parede e cabos condutores. Por exemplo, suponha uma empresa com cinco computadores, com dois em um andar do prédio e outros três computadores em outro andar do prédio. A figura

abaixo ilustra cada computador e suas respectivas conexões, seis conexões passaram entre duas localizações - em muitos casos essas conexões seguem o mesmo caminho, aumentando custos.



O número de conexões acima passando entre as duas localidades excede o número total de computadores. Se outro computador é adicionado à localização 1, a situação se torna bem pior, e o número de conexões totais cresce para 9.

3.3 Canais de Comunicação Compartilhados

A história das redes de computadores mudou drasticamente entre 1960 e 1970, quando pesquisadores desenvolveram uma forma de comunicação entre computadores chamada de Local Area Networks (LANs). Planejadas como alternativas para fugir do alto custo das conexões ponto a ponto, os pesquisadores diferenciaram fundamentalmente das redes a longas distâncias. Cada LAN consiste em um único meio compartilhado, usualmente um cabo, para os computadores se atarem. Os computadores têm turnos para usar o meio ao enviar pacotes de dados.

Muitos projetos de LANs surgiram da pesquisa e esses eliminaram a duplicação de cabos e custos, trazendo um importante impacto econômico nas redes. Consequentemente as LANs se tornaram populares.

“Redes que permitem múltiplos computadores compartilhar um meio de comunicação são usados para comunicação local. Conexões ponto a ponto são usadas para redes a longas distâncias e outros casos especiais.”

Se compartilhar diminui custos, porque as LANs são usadas somente para comunicação local? Porque os computadores em uma rede precisam coordenar o uso da rede, assim essa coordenação requer tempo, e a longa distância esse tempo seria maior que o próprio envio de dados, assim a rede passaria maior parte do tempo coordenando, e não enviando dados. Além do que, engenheiros descobriram que disponibilizando um canal com grande largura de banda, a longa distância é muito mais caro do que provendo esse mesmo canal a curta distância.

3.4 Importância das LANs e Localização de Referência

A importância das LANs pode ser resumida em:

“As LANs se tornaram a forma mais popular de comunicação entre computadores. LANs agora conectam mais computadores que qualquer outro meio de rede.”

Uma das razões das LANs serem instaladas é a economia. As tecnologias de uma LAN são tanto baratas quanto disponíveis. Todavia, a principal razão para que a demanda de redes seja tão alta pode ser atribuída ao princípio fundamental conhecido como **localização de referência**. O princípio da localização de referência diz que a comunicação entre uma quantidade de computadores não é randômica, mas ao invés disso segue dois modelos. Primeiro,

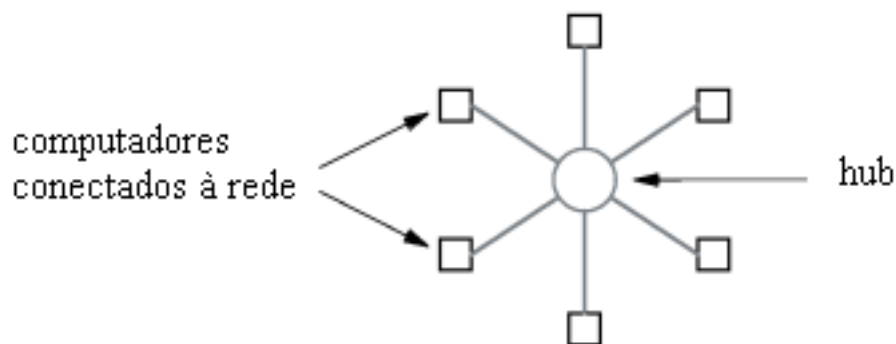
um par de computadores comunica uma vez, esse par irá comunicar novamente em um tempo próximo, e frequentemente. Essa corrente é chamada de localização de referência temporal, e implica um relacionamento no tempo. Segundo, um computador tende a comunicar com mais frequência com computadores que estão mais próximos dele. O segundo modelo é chamado de localização de referência física e enfatiza o relacionamento geográfico.

3.5 Topologia de LANs

Devido a várias tecnologias de LANs terem sido inventadas, é importante saber que tecnologias são similares e como elas diferem. Para compreender as similaridades, cada rede é classificada em uma categoria de acordo com a sua “topologia” ou forma geral. Esta seção descreve as três topologias mais usadas em LANs.

3.5.1 Topologia de Estrela

Uma rede que usa topologia de estrela tem todos os computadores conectados a um único ponto. A figura abaixo ilustra o conceito:

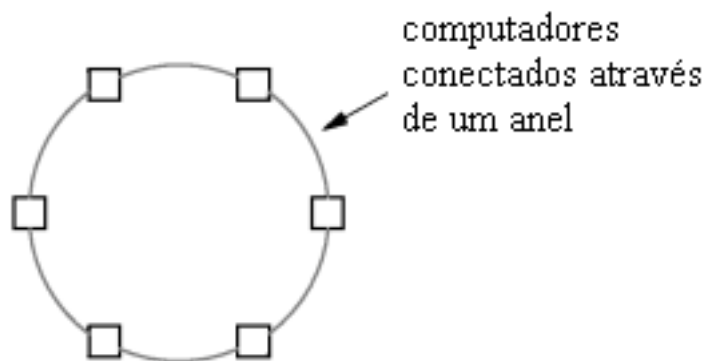


Como uma topologia de estrela lembra uma roda, o centro da estrela é comumente chamado de hub. Um hub típico consiste em um equipamento eletrônico que aceita dados de computadores e entrega no destino certo.

A figura acima ilustra uma rede de estrela idealizada. Na prática, redes em estrela raramente têm um formato simétrico no qual o hub é localizado numa distância igual de todos os computadores. Ao invés disso, frequentemente o hub está localizado em um local diferente do centro da estrela.

3.5.2 Topologia de Anel

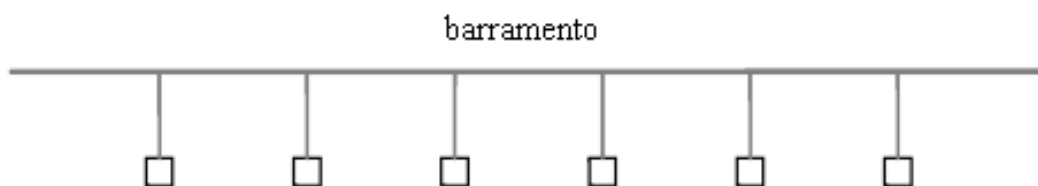
Uma rede que usa uma topologia de anel para agrupar computadores conectados em uma sequência fechada- um cabo conecta o primeiro computador ao segundo computador, outro cabo conecta o segundo ao terceiro e assim por diante. O nome anel porque pode se imaginar os computadores se conectando como um círculo como a figura abaixo ilustra:



É importante entender que o anel, como a topologia de estrela, refere-se para conexões lógicas entre computadores, não como uma orientação física - os computadores conectados em uma topologia de anel não precisam estar arranjados em um círculo. Ao invés disso, o cabo entre dois computadores deve seguir um corredor ou subir verticalmente de um andar de um prédio a outro.

3.5.3 Topologia de Barramento

Uma rede que usa a topologia de barramento usualmente consiste em um único e longo cabo, no qual os computadores são conectados. Qualquer computador conectado ao barramento pode enviar um sinal através do cabo e todos os computadores receberão o sinal. A figura abaixo ilustra essa topologia. Devido todos os computadores estão conectados ao cabo, todos podem receber sinais elétricos, e qualquer um pode enviar dados para outro computador. É claro que um computador conectado ao barramento deve estar coordenado com o resto da rede, evitando que mais de um computador envie dados ao mesmo tempo, ou o resultado será o caos.



3.5.4 A Razão para Várias Topologias

Cada topologia tem vantagens e desvantagens. A topologia de anel torna fácil a coordenação do acesso dos computadores e a detecção de mau funcionamento na rede, porém é desabilitada se um único cabo é rompido. A topologia de estrela protege no caso de danos em um único cabo, porque cada cabo só conecta um computador. A de barramento requer menos fios, mas tem a mesma desvantagem da topologia de anel, se o cabo principal é rompido ou desabilitado, a rede para de funcionar.

3.6 Exemplo da rede em barramento: Ethernet

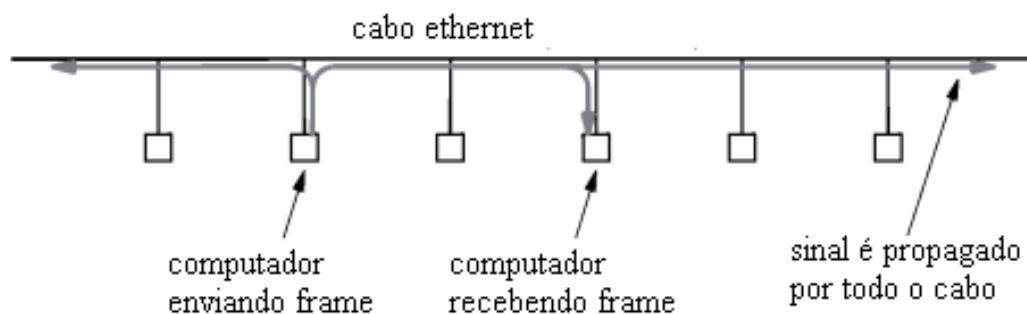
3.6.1 História da Ethernet

A Ethernet foi inventada pela Xerox Corporation na década de 1970. Na sua versão original a Ethernet consistia em um único cabo coaxial, chamado ether, no qual múltiplos computadores se conectam. Engenheiros usam o termo segmento para referir-se ao cabo coaxial da Ethernet. Um segmento de Ethernet é limitado em 500 metros de distância, e o padrão requer uma separação mínima de 3 metros entre cada par de conexões. O hardware original da Ethernet operava em uma largura de banda de 10 Mbps, uma versão posterior conhecida como Fast Ethernet operava em 100 Mbps e a versão mais recente, conhecida por Gigabyte Ethernet opera em 1000 Mbps, ou 1 Gigabyte por segundo.

3.6.2 Compartilhamento em uma Ethernet

O padrão para a Ethernet especifica todos os detalhes, incluindo o formato dos frames que o computador envia através do ether, a voltagem a ser usada, e o método para modular o sinal.

Por usar a topologia de barramento, a Ethernet requer múltiplos computadores compartilhando um mesmo meio. Um emissor transmite um sinal, que propaga do emissor para as duas pontas do cabo. A figura abaixo ilustra como dados fluem pela Ethernet.



Como a figura mostra, o sinal propaga de um computador para as duas pontas do cabo. É importante entender que compartilhar um mesmo meio não significa que múltiplos frames serão enviados ao mesmo tempo. Ao invés disso, o computador emissor tem o uso exclusivo do cabo durante a transmissão de um determinado frame – outros computadores têm que esperar. Após um computador terminar a transmissão de um frame, o cabo se torna disponível para os outros computadores utilizá-lo.

3.7 Portador de Lógica em uma Rede de Múltiplos Acessos (CSMA)

O aspecto mais interessante na Ethernet é o mecanismo utilizado para coordenar a transmissão. Uma rede Ethernet não tem um controle centralizado que diz para cada computador qual é a hora de usar o cabo comum. Ao invés disso, todos os computadores conectados a Ethernet participam de um esquema coordenado chamado Portador de lógica para acesso múltiplo (CSMA). Quando nenhum computador está transmitindo frames, o ether não tem nenhum sinal elétrico. Porém durante a transmissão o emissor transmite sinais elétricos usados

para codificar os bits. Ainda que o sinal difira um pouco das ondas descritas no capítulo 5, elas são chamadas normalmente de portadoras (carrier). Assim, para verificar quando o cabo está disponível, o computador procura por uma portadora no cabo. Se a portadora não está presente, o computador transmite o frame. Se a portadora está presente, o computador deveria esperar para o fim da transmissão antes de fazer a transmissão. Tecnicamente, o processo de checar por uma portadora é chamado de senso de portador (CSMA).

3.8 Detecção de Colisões e Impedimento com CSMA/CD

O CSMA permite que cada computador cheque o cabo, a fim de verificar sua disponibilidade. Porém o CSMA não pode prevenir todos os conflitos possíveis. Imagine dois computadores nas pontas opostas do cabo, e ambos tem um frame pronto para ser enviado. Quando checam por um carrier (portador), e ambos vêem que o cabo está livre, então começam a enviar simultaneamente os frames. Os sinais viajam a uma velocidade de 70% da velocidade da luz, e quando atingem o mesmo ponto do cabo, eles se interferem.

A interferência entre os dois sinais é chamada de colisão. Essa colisão cria um sinal distorcido evitando que os dois frames sejam recebidos corretamente. O padrão da Ethernet requer que estação emissora monitore os sinais no cabo. Se o sinal no cabo difere do sinal enviado pela estação emissora, é detectada a colisão e é parado imediatamente o envio de dados por ela. Tecnicamente, monitorar o cabo durante a transmissão é conhecido como Collision Detect (CD) e o mecanismo de Ethernet é conhecido como Carrier Sense Multiple Access with Collision Detect (CSMA/CD).

O CSMA/CD, além de prevenir as colisões, evita que após ocorrer uma colisão, ocorra outra, pois ambos os computadores irão parar a transmissão e refazê-la quando o cabo estiver livre, assim o padrão especifica que deve haver um delay no computador após a colisão e antes que aja a retransmissão. O padrão especifica um delay máximo d , e força o computador a escolher randomicamente um delay menor que d . Assim o computador que escolher o delay menor ira transmitir primeiro.

Se os dois ou mais computadores escolhem delays próximos, eles irão começar a transmissão e uma nova colisão ira ocorrer, produzindo assim a segunda colisão. A Ethernet então requer que cada computador dobre o limite que foi escolhido na colisão anterior. Assim a probabilidade de uma sequência de colisões se torna bem pequena.

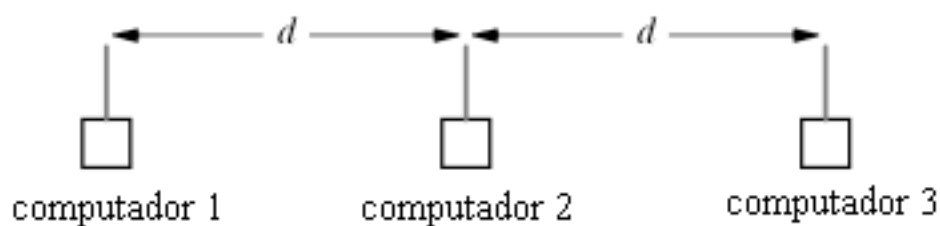
Tecnicamente, dobrar o limite do delay após cada colisão é conhecido como binary exponential backoff. Na essência, exponencial backoff significa que a Ethernet pode recuperar-se rapidamente após uma colisão porque cada computador aceita esperar tempos maiores enquanto o cabo está com dados. Assim é garantido que as colisões serão mínimas.

3.9 LANs Wireless e CSMA/CA

Ao invés de transmitir sinais através de um cabo, LAN wireless usam antenas para transmitir sinais de RF através do ar, que outros computadores irão receber. Os dispositivos usam a frequência de 900 MHz para enviar dados a 2 Mbps. Como as redes wireless são usadas para compartilhamento, todos os computadores devem estar configurados na mesma frequência.

Uma diferença entre a maneira de transmissão de dados em fios e em wireless é o gerenciamento de dados devido à maneira de propagação da transmissão. Embora radiação eletromagnética seja transmitida em todas as direções, os transmissores de LAN wireless usam pouca energia, significando quem a transmissão tem poder de viajar apenas em uma curta distância. Além do que, obstruções metálicas bloqueiam o sinal.

O caráter da LAN wireless não permite que esta use o mesmo mecanismo CSMA/CD que a Ethernet usa. Para entender o porquê, considere três computadores com hardware LAN wireless posicionados na máxima distância d , como a figura abaixo mostra.



Na figura, os dois computadores externos estão separados a uma distância que não conseguem receber transmissões um do outro. Nessa situação, o carrier sense e collision detection não basta. Por exemplo, suponha que o computador 1 está enviando dados para o computador 2. Por causa do computador 3 não receber essa transmissão, ele pode também transmitir para o computador 2, resultando em uma colisão. Similarmente, se os computadores 1 e 3 transmitirem frames ao mesmo tempo, apenas o computador 2 irá detectar a colisão.

Para certificar-se que a transmissão de dados está correta, LANs wireless usam um esquema modificado, conhecido como Carrier Sense Multiple Access With Collision Avoidance (CSMA/CA). Ao invés de dependerem de todos outros computadores receberem toda transmissão, o CSMA/CA é usado com gatilhos curtos de transmissão para o receptor antes de transmitir o pacote de dados. Por exemplo, suponha que o computador 1 na figura acima precise transmitir dados para o computador 2. Assim, antes de transmitir o frame, o computador 1 transmite uma mensagem curta de controle. Quando o computador 2 recebe a mensagem de controle, ele responde enviando outra mensagem de controle, indicando que está pronto para receber a transmissão. Quando o computador 1 recebe esta mensagem, ele começa o envio dos frames.

A vantagem de esperar uma resposta do receptor se torna clara se lembrarmos que a transmissão é assimétrica. Na figura, embora o computador 3 não receba a transmissão do computador 1, ele recebe a transmissão do computador 2. Assim, uma vez que o computador 2 envie a mensagem de resposta, todos os computadores na área de alcance da antena irão esperar pela transmissão de um pacote (mesmo que não consigam receber a transmissão).

Colisões de mensagens de controle podem ocorrer quando se usa CSMA/CA, mas podem ser facilmente contornadas. Na figura, por exemplo, se o computador 1 e 3 gerarem um pacote para o computador 2 no mesmo instante, ambos enviam uma mensagem de controle. As mensagens de controle irão chegar simultaneamente ao computador 2, causando uma colisão. Quando este tipo de colisão ocorre, a estação emissora aplica um random de impedimento antes de re-enviar a mensagem de controle. Como as mensagens de controle são menores que os pacotes de frames, a probabilidade de uma segunda colisão ocorrer é muito menor que na Ethernet convencional. Eventualmente, uma das duas mensagens de controle chega intacta e o computador 2 transmite a resposta.

3.10 Outro Exemplo de Rede de Barramento: LocalTalk

A tecnologia de LAN LocalTalk foi desenvolvida pela Apple, e é usada principalmente em seus computadores pessoais.

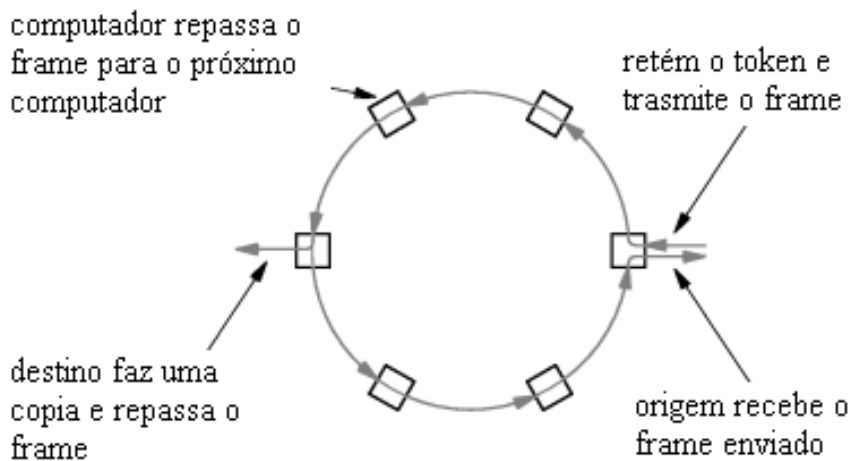
A rede LocalTalk é um barramento no qual os computadores conectados utilizam uma versão do CSMA/CA para acessar dados. Como a mensagem de controle utilizada neste tipo de comunicação é extremamente pequena comparada com uma mensagem de dados, o tempo destinado para a reserva do envio se torna insignificante.

Uma das desvantagens da LocalTalk é a largura de banda, que é 2.3% da largura de banda disponível em uma rede Ethernet de 10 Mbps. Uma largura de banda menor significa que levará um tempo maior para transmitir grandes volumes de dados. A rede LocalTalk também tem limitações de distância, e não é disponível em todos os computadores.

Sua vantagem principal é o baixo custo, pois o hardware necessário para fazer a conexão já está incluso no computador. Por exemplo, um único cabo é necessário para conectar dois computadores Macintosh.

3.11 Exemplo de Rede em Anel: IBM Token Ring

A maioria das redes em anel utiliza um mecanismo de acesso chamado de Token passing (passagem do anel). Quando um computador precisa enviar dados, ele deve esperar a permissão antes de acessar a rede. Uma vez que obtém a permissão, o computador tem total controle sobre o anel - não há duas permissões simultâneas. Assim que o computador transmite um frame, os bits passam para o próximo computador e assim em diante, até completar o anel e o emissor receber de volta os dados que ele mesmo envia. A figura abaixo ilustra o conceito.



Assim, para verificar que não ocorreu erros na transmissão, o emissor compara os dados que ele recebeu com os que foram enviados. Se um frame é destinado a um determinado computador, este computador faz uma cópia dos bits que estão passando pelo anel.

Para coordenação do envio de dados, é usado uma mensagem especial chamada de Token. O Token é um bit padrão que difere do formato normal de dados. Para assegurar que dados normais não sejam interpretados como Token, algumas tecnologias usam bits de preenchimento. E o mais importante, há apenas um Token em toda rede.

Na essência, o Token dá ao computador a permissão para enviar um frame. Assim, antes de enviar um frame, o computador deve esperar o Token chegar. Quando o Token chega, o computador remove o Token temporariamente do anel e usa o anel para transmitir um frame, logo após transmite o Token. Assim o Token viaja de um computador para o adjacente, que pode usar a rede para enviar um frame.

Note que o esquema garante um acesso justo: assim que o Token passa no anel, cada computador tem a oportunidade de usar a rede. Se um computador não tem nenhum frame para enviar, ele repassa o Token sem nenhum delay. Além do que, o tempo necessário para que um Token passe por todo anel, com todos os computadores sem enviar nenhum dado é muito pequeno, na casa de milissegundos.

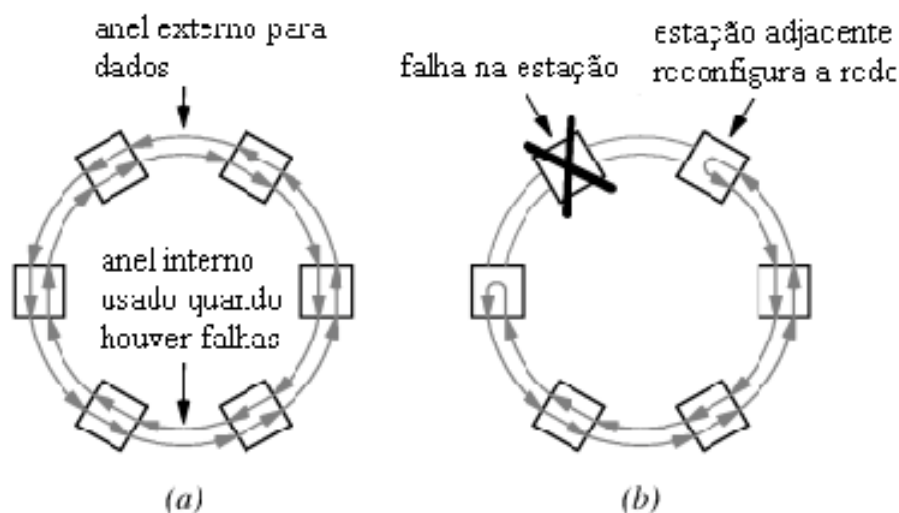
3.12 Outro Exemplo de Rede em Anel: FDDI

Uma das desvantagens principais do rede em anel é que a rede é muito suscetível a falhas. Como cada computador está conectado ao anel, e os bits devem passar por eles, uma falha em um único computador desabilita toda rede.

Fiber Distributed Data Interconnect (FDDI) é uma tecnologia que transmite dados na taxa de 100 milhões de bits por segundo, oito vezes mais rápido que a tecnologia IBM Token Ring e dez vezes maior que a Ethernet original. Para prover essa taxas tão altas, FDDI utiliza fibra óptica ao invés de fios de cobre.

FDDI também utiliza redundância para evitar que a rede falhe. Uma rede em FDDI contém dois anéis completos – um para enviar dados quando tudo está funcionando corretamente, e outro que é usado quando o primeiro anel falha.

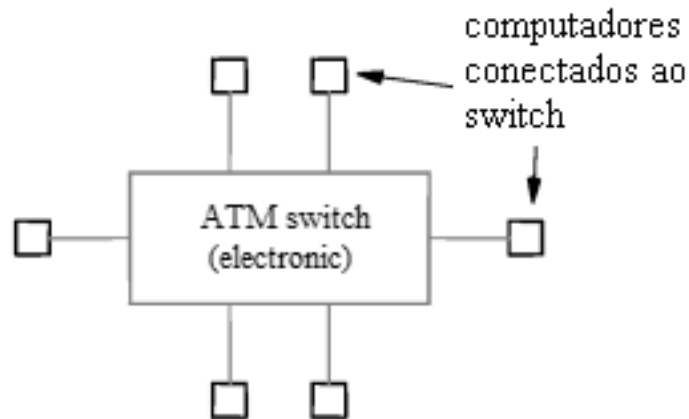
Os anéis em uma rede FDDI são chamados de counter rotation porque os dados escorrem no segundo anel em sentido diferente da direção do anel principal. Para entender a motivação da rotação contrária, considere que uma falha catastrófica ocorra Primeiro porque um par de fibras ópticas que conectam dois locais usualmente seguem o mesmo caminho, e uma quebra acidental iria quebrar ambos. Segundo, que dados sempre passam na mesma direção em ambos os cabos, desconectando uma estação ira desconectar ambos os cabos. Porém se os dados estiverem em sentidos contrários, as estações remanescentes podem re-configurar a rede, a figura abaixo ilustra o conceito.



A figura acima ilustra o caminho dados com uma falha. O hardware nas estações adjacentes detecta a desconexão e re-configura a rede de modo que os dados que chegam voltam pelo caminho reverso. Esse processo de re-configuração é chamado de self healing, ou cura automática, e FDDI é conhecido como self healing network.

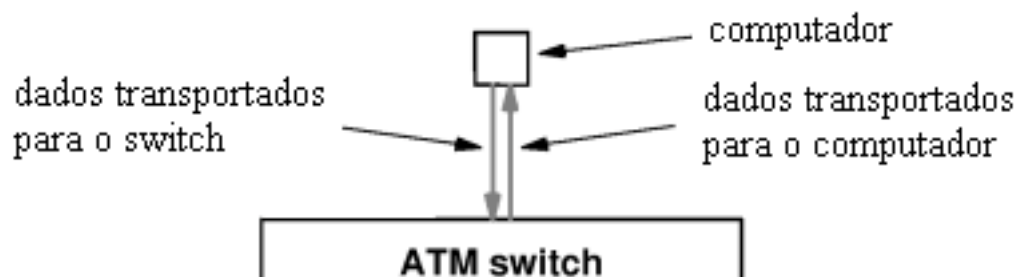
3.13 Exemplo de Rede em Estrela: ATM

Companhias telefônicas desenvolveram a tecnologia conhecida como Asynchronous Transfer Mode (ATM). O elemento básico de uma rede ATM é um switch eletrônico, no qual diversos computadores podem se conectar. Por exemplo, a figura abaixo ilustra 6 computadores conectados ao switch ATM.



Como a figura mostra, a rede ATM tem topologia de estrela. Diferentemente das topologias de barramento ou anel, uma rede em estrela não propaga dados para qualquer computador ao não ser o par que está se comunicando – o hub recebe os dados diretamente do emissor, e transmite diretamente para o receptor. Assim a rede em ATM tem menor dependência das conexões individuais de cada computador, do que a rede em anel. Assim se a conexão entre um computador e o switch falha, apenas esse computador é afetado.

Como a rede ATM é projetada para ter uma alta largura de banda, uma conexão típica entre um computador e um switch ATM opera na velocidade de 155 Mbps ou mais rápido que isso. Normalmente é usada fibra óptica, ao invés de fio de cobre. De fato, um único cabo de fibra óptica não consegue levar dados nas duas direções simultaneamente, assim cada conexão utiliza um par de fibras ópticas, como a figura abaixo ilustra.



Capítulo 4 — Endereçamentos de Hardware e Identificação do Tipo de Frame

4.1 Introdução

Este capítulo considerará a transmissão em uma LAN com mais detalhes, e explicará como um par de computadores comunica-se através da LAN sem forçar outros computadores receber e processar uma cópia de cada mensagem. Assim, este capítulo descreve o endereçamento de hardware, e mostra como um computador emissor usa o endereçamento de hardware para identificar qual computador deverá receber a cópia de um determinado frame. Esse capítulo também explica como a interface do hardware de rede usa o endereçamento para filtrar os frames recebidos.

Também é examinado um mecanismo que permite ao emissor identificar o tipo de dados em cada frame. Isso explica o propósito conceitual de tipo de identificação de frame e o usa como exemplo para ilustrar o conceito os frames da Ethernet.

4.2 Especificando um Receptor

Normalmente a comunicação em um meio comum não envolve todas as estações. Assim, a maioria das comunicações ocorre porque um aplicativo em um computador envia dados diretamente para o aplicativo em outro computador.

A maioria das tecnologias de LANs usa um esquema de endereçamento para prover uma comunicação direta. Cada estação em uma LAN tem um valor numérico único, chamado endereço físico, endereço de hardware, ou Média Access Address (MAC address). Quando o emissor transmite um frame através da LAN, o emissor inclui o endereço de hardware do receptor daquele frame. Assim o hardware da LAN em cada estação checa o endereço de cada frame que chega, para determinar quando ela deverá aceitar o frame.

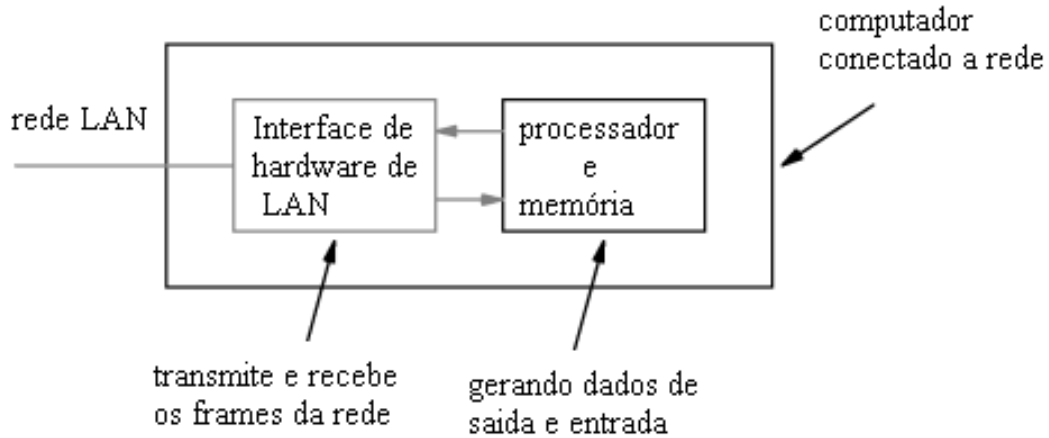
De fato, cada frame transmitido através uma LAN inclui dois endereços – um que especifica o receptor e outro que especifica o emissor. Cada frame começa com um *header* fixo, que contém o espaço para os dois endereços. Antes de transmitir um frame, o emissor deve colocar o endereço do receptor no campo chamado *destination address field* e seu próprio endereço no campo *source address field*. Incluir o endereço do emissor em cada frame faz com que a resposta do receptor seja gerada mais facilmente. A interface do hardware é projetada para examinar os campos de endereço em cada frame que passa pela rede, e aceitar apenas aqueles que o endereço do destinatário é igual ao endereço da estação.

4.3 Como o Hardware de uma LAN usa o Endereço para Filtrar os pacotes

Para avaliar como os sistemas usam o endereçamento para prover uma conexão direta, é necessário compreender a organização básica do hardware de LAN. A figura abaixo ilustra o conceito, o hardware de uma LAN que é completamente separado da CPU e da memória do computador.

A interface do hardware de LAN lida com todos os detalhes de enviar e receber frames do meio compartilhado. Mais importante, o hardware da LAN pode realizar as funções de envio e recepção de frames sem utilizar a CPU do computador. Assim, depois de passar os dados para

a interface da LAN e requisitar a transmissão, a CPU pode continuar executando um programa, enquanto a interface espera o acesso ao meio e transmite o frame.



Conceitualmente, a interface do hardware de LAN, que opera independentemente do processador do computador, usa o endereçamento físico para prevenir o computador de receber todos os pacotes que estão na LAN. Relembrando que um frame viaja através do meio compartilhado, e uma cópia dele é passada a cada estação. Uma vez que tenha capturado completamente um frame, o hardware de interface compara o endereço do destinatário com o endereço da estação. Se o endereço for o mesmo, o hardware aceita o frame e o passa para o sistema. Se o endereço não for o mesmo, o frame é descartado e o hardware volta a esperar por mais um frame e um frame endereçado com uma estação que não existe, sempre será ignorado.

4.4 Formato de um Endereço Físico

Os vários tipos de formas de endereços podem ser agrupados em 3 categorias:

- **Static:** Um endereço static conta com o fabricante do hardware, que irá determinar um único endereço físico para cada interface de rede. Um endereço físico static não muda ao menos que o hardware seja substituído.
- **Configurable:** Um endereço configurável provém de um mecanismo que o consumidor pode usar para determinar o endereço físico. O mecanismo pode ser manual ou eletrônico (memórias EPROM). A maioria dos hardwares precisa ser configurado uma vez – usualmente feita quando ele é instalado.
- **Dynamic:** Um endereço dinâmico provém de um mecanismo que automaticamente determina o endereço físico de uma estação quando esta é ligada. A maioria dos esquemas de endereçamento dinâmicos requer que a estação tente números randômicos até encontrar um valor que nenhum computador está usando. Por exemplo, uma estação escolhe o número de dias como valor inicial. Para cada número random gerado, a estação envia uma mensagem na rede para o endereço especificado. Se um computador está usando o endereço, ele irá responder a mensagem. Se não houver resposta, o emissor pode utilizar esse endereço como endereço físico. Assim, o endereço de um

computador depende dos endereços dos outros computadores quando este é ligado – um computador poderá obter diferentes endereços a cada vez que é reiniciado.

A maior vantagem do endereço estático é sua facilidade de uso e permanência. O esquema é fácil de usar porque os fabricantes determinam o endereço e confirmam que aquele hardware terá aquele endereço único em todo mundo. Assim hardwares de vários fabricantes podem ser conectados sem nenhum conflito de endereço.

Endereço dinâmico tem duas vantagens: elimina a necessidade de o fabricante ter que coordenar o endereçamento e permite que cada endereço seja menor. A razão para um endereço ser menor chega devido à exclusividade de ter que ser única apenas na LAN que ele está. A maior desvantagem é a falta de permanência e conflitos em potencial. Por exemplo, se dois computadores reiniciam enquanto a rede está desabilitada, estes podem escolher os mesmos endereços, ocorrendo conflitos.

O endereço configurável é uma mescla do estático e do dinâmico. Como no dinâmico, o configurável não precisa usar endereços longos, pois ele deve ser único apenas na rede que ele está conectado. E a interface configurável pode ser trocada sem mudar o endereço físico do computador.

4.5 Broadcasting

O termo, broadcasting (transmissão) que foi originalmente aplicado em transmissões de rádio e televisão e se refere à transmissão para uma larga audiência. Quando um aplicativo realiza uma transmissão de broadcast, todos os outros computadores da rede recebem uma cópia dos dados enviados.

Broadcast tem muitos usos. Por exemplo, suponha que um computador precise encontrar uma impressora na rede. O computador pode formar uma mensagem que especifique uma impressora e aí transmite a mensagem para todas as estações na rede. Mesmo que todas recebam a mensagem, apenas as que contem impressora irão responder.

Como a maioria das tecnologias usa um meio compartilhado, elas podem fazer do broadcasting uma técnica extremamente eficiente. De fato, não é necessário nenhum hardware adicional para ocorrer um broadcast na LAN, pois todas as estações estão conectadas ao meio comum. Tudo que é preciso para fazer um broadcast é um mecanismo que todas as estações irão extrair e processar uma cópia dos frames.

Para fazer o broadcast eficiente, a maioria das LAN estende o esquema de endereços. O projetista da LAN define um endereço especial e reservado, conhecido como *broadcast address*. A interface em um computador foi construída para reconhecer o *broadcast address* como reconhece os endereços das estações.

4.6 Multicasting

Diferentemente da aparente facilidade do uso do broadcast, poucos aplicativos em redes foram projetados como o descrito acima. A razão é simples – broadcasting é extremamente ineficiente. Mesmo que cada estação na rede pode ser configurada para descartar frames desnecessários, processar e descartar um frame requer recursos computacionais. Quando um frame chega, o hardware da interface de rede coloca o frame na memória, interrompe a CPU, e permite o software do sistema determinar se o frame deve ser ignorado. Assim, descartar frames envolve uma decisão da CPU. Temos então um grande desperdício de tempo de processamento em toda rede.

Como computadores em uma LAN podem tirar vantagem do broadcasting, sem desperdiçar recursos da CPU? A resposta está na forma restrita de broadcasting, o multicasting. Ele se parece muito com o broadcasting, porém quando os frames chegam, não são repassados diretamente para a CPU. A própria interface de rede faz a decisão de aceitar ou não o frame.

Para que isso ocorra, o hardware deve ser programado com determinadas especificações, e só serão aceitos frames que checam essas especificações.

4.7 Multicasting Addressing

Multicasting estende o esquema de endereços reservando alguns endereços para multicast, e estende a interface de rede permitindo que a interface reconheça um conjunto adicional de endereços. Quando o computador é ligado, a interface de rede é programada só para reconhecer os endereços dos computadores e os de broadcasting. Se um aplicativo em um computador necessitar receber frames de multicasting, o aplicativo deve informar a interface de rede qual endereço de multicasting deve ser usado. A interface adiciona o endereço ao conjunto que ela irá reconhecer, e começa aceitar os frames enviados para aquele endereço.

Suponha que dois aplicativos tenham sido construídos para enviar sons pela rede. Um aplicativo aceita e digitaliza o áudio e então envia o resultado para o outro aplicativo. O segundo aplicativo recebe o áudio digitalizado da rede, converte novamente para sinais de áudio e toca o resultado em um alto-falante. Agora suponha que outros computadores na rede desejem escutar o áudio. Ao menos que dois aplicativos usem broadcast para enviar frames, nenhum outro computador irá receber uma cópia dos frames. Assim broadcasting terá a desvantagem de consumir recursos da CPU de todos os computadores, até mesmo aqueles que não querem ouvir o áudio.

Multicasting provém uma excelente solução para o problema de permitir alguns computadores de participarem da transmissão de áudio, enquanto não incomoda o outro. Multicasting oferece a vantagem de enviar apenas uma única cópia de cada frame na rede, e permitir computadores arbitrários receberem a transmissão. Para usar multicasting, um endereço de multicasting deve ser escolhido pelo aplicativo de áudio. Assim o software de áudio deve ser configurado para usar o endereço. O programa emissor deve arranjar um lugar para colocar o endereço no campo de destino do frame, e o programa receptor deve ser configurado para usar o endereço dos frames que chegam. Mais especificamente, quando o programa receptor é ligado, ele passa o endereço de multicasting para a interface de rede. A interface adiciona o endereço e começa a aceitar os pacotes enviados para aquele endereço. Devido à interface de rede checar o endereço, computadores que não rodem o aplicativo de áudio não irão desperdiçar tempo da CPU para descartar os frames.

4.8 Identificando o Conteúdo dos Pacotes

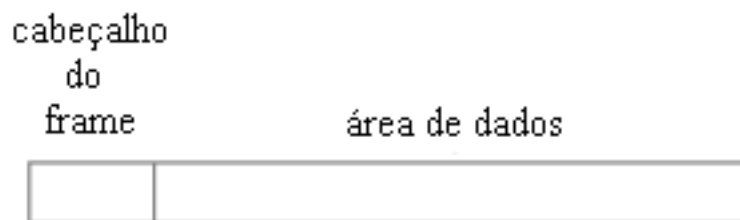
Embora o esquema de endereços descrito acima permita o emissor identificar o receptor de um pacote, o endereço não especifica o que os pacotes contêm. Mais importante, porque muitos dados usam representações, um receptor não pode utilizar dados no pacote para determinar o que os pacotes contêm. Por exemplo, pacotes que levam mensagens de e-mail, texto e páginas da web todos usam ASCII para representar os dados. Para informar ao receptor o que contém no pacote, cada frame contém informações adicionais para especificar o tipo do conteúdo, dois métodos são usados:

- **Tipo de frame explícito** - Neste método, os projetistas dos hardware de rede especificam qual tipo de informação é incluído no frame e os valores usados para identificar os vários tipos de frames. Os bits de um frame que são usados para identificar o conteúdo são chamados *frame type field*, e o frame é chamado de *self-identifying*.
- **Tipo de frame implícito** - Neste método, o hardware de rede não inclui o tipo de frame em cada frame. Ao invés disso, o frame leva apenas dados. Assim, o emissor e o

receptor devem concordar sobre o tipo de frame ou concordar em usar uma porção dos dados para ser usado como campo de tipo.

4.9 Cabeçalho de Frames e Formato de Frames

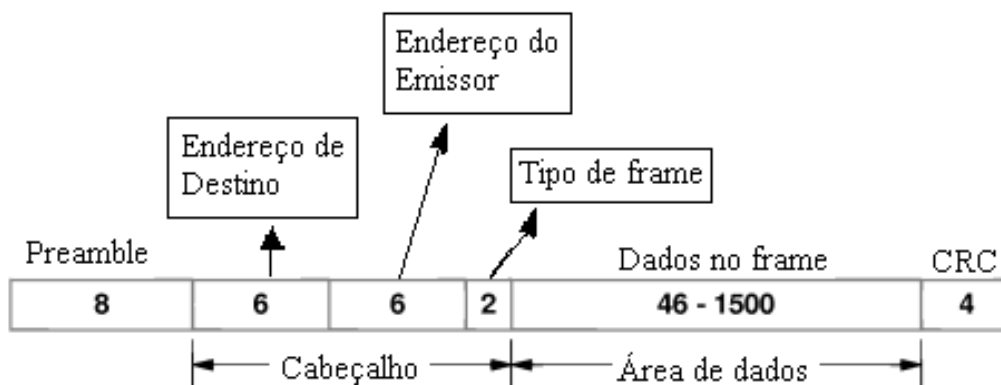
Cada tecnologia de LAN define o formato exato de frames a ser usado. E a maioria das LANs define o frame com duas partes, o cabeçalho do frame, que contém informações como a fonte e o destinatário. E é seguido por uma área de dados, que contém a informação que é enviada. A figura abaixo ilustra o formato geral:



Cada campo no cabeçalho do frame tem tamanho e localização fixos. Como resultado, todos os frames, usados com a mesma tecnologia, têm o mesmo tamanho de cabeçalho. Em contraste, a área dos dados não tem tamanho fixo – a quantidade de dados que será enviada que determina o tamanho da área de dados.

4.10 Um exemplo da Formatação de Frames

A figura abaixo ilustra o formato de frame utilizado na Ethernet. Como a figura mostra, um frame da Ethernet começa com um cabeçalho que contém três itens. O *preamble* de 64 bits que precede o frame contém 1 e 0 alternados, para que o hardware do receptor se sincronize com o sinal que chega. Os dois primeiros campos contêm os endereços físicos. A Ethernet usa endereços estáticos de 48 bits em que cada equipamento que é feito com endereço único pelo fabricante. O campo chamado de *Dest. Address* contém o endereço físico do destinatário. O campo chamado *Source Address* contém o endereço físico da estação que está enviando o frame. O terceiro campo do cabeçalho consiste no tipo do frame, tendo 16 bits.



A Digital-Intel-Xerox Ethernet define os valores padrões que devem ser usados no cabeçalho e seus significados.

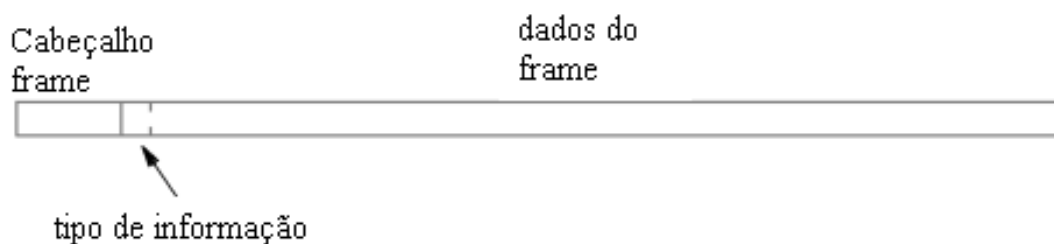
4.11 Usando Redes que não Auto-Identificam Frames

Algumas tecnologias de rede não incluem o formato dos frames no cabeçalho, isto é, os frames não são auto-identificados. Como podem os computadores conectados nessas redes saber o tipo de dados em cada frame? Existem duas possibilidades:

1. Antes que qualquer dado seja enviado, o emissor e receptor concordam em usar um único formato de frames. O software no computador emissor é programado para colocar os dados no formato escolhido, e o software no receptor é programado para esperar os dados no formato escolhido;
2. Antes que qualquer dado seja enviado, o emissor e o receptor concordam em usar os primeiros octetos do campo de dados para guarda a informação do tipo de dados. O software no computador emissor adiciona a informação do tipo de dado antes de colocar os dados no frame a ser enviado. O software no receptor extrai a informação do tipo dos dados, usando essa informação para saber como proceder com os dados.

A primeira técnica é raramente usada, pois limita um par de computadores a um único formato de dados. Assim, os donos dos dois computadores não podem instalar novos aplicativos, a não ser que estes usem o formato de dados que foi selecionado. Se a rede comporta broadcast, todos os computadores ligados na rede deverão concordar em um único tipo de formatação de dados.

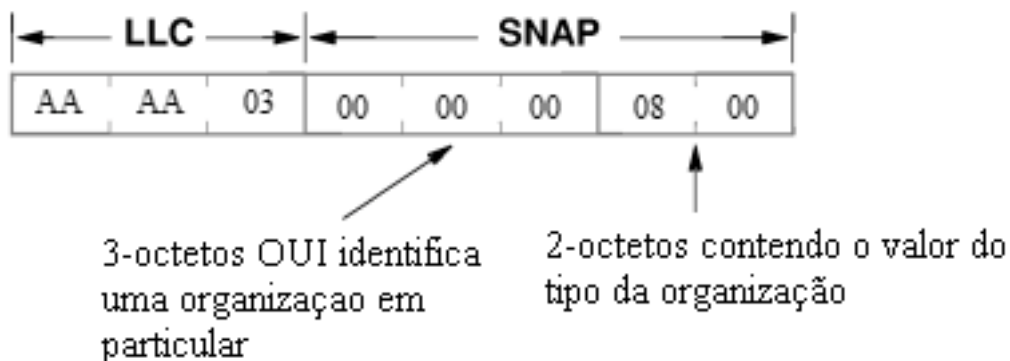
A figura abaixo mostra a segunda técnica, que usa parte da área de dados para armazenar a informação sobre o tipo do frame.



Usar parte da área de dados para informar o tipo do frame levanta duas questões. Primeiro qual será o tamanho exato do tipo da informação? Segundo, quem irá especificar os valores que são permitidos e os seus significados? Essas decisões são tomadas pelos projetistas do hardware de rede. Mesmo assim, se o hardware de rede não incluir o tipo dos frames, o software do receptor é livre para interpretar o tipo da informação. Infelizmente, permitir que cada programador escolha valores para o campo do tipo não funciona bem, pois dois programadores podem escolher acidentalmente os mesmos valores para diferentes tipos. O problema se torna complexo se o aplicativo realizar broadcasts.

Para assegurar que todos os softwares usam os mesmos valores para especificar os tipos, as organizações padrões definiram os significados para cada valor. Infelizmente, múltiplas organizações determinaram valores diferentes para o mesmo tipo. Como resultado, duas organizações pode acidentalmente escolher o mesmo valor para diferentes tipos, IEEE definiu um padrão que inclui um campo para identificar a organização padrão bem como o tipo do frame. Essa especificação é conhecida com *Logical Link Control (LLC) SubNetwork Attachment Point (SNAP)*. E esse padrão é largamente aceito.

A figura abaixo mostra um exemplo de um cabeçalho LLC/SNAP que contém oito octetos.



A porção SNAP é dividida em dois campos, o primeiro é chamado de *Organizationally Unique Identifier (OUI)*, que é usado para identificar uma organização em particular. O segundo contém o valor do tipo, definido por aquela organização.

Como um tipo codificado no cabeçalho, o tipo LLC/SNAP torna possível todos os computadores fazerem broadcast de frames na rede. Quando um frame chega a determinado computador, o computador procura pela informação LLC/SNAP no começo da área de dados. Se o computador não reconhecer o OUI ou não ter o software que lida com o tipo de dados que estão sendo enviados, o receptor descarta o frame. Assim, se apenas três computadores de uma rede entendem um tipo de frame, o broadcast desse frame será ignorado por todos, menos os três.

4.12 Análise de Rede, Endereços Físicos, Tipo de Frames

Um *network analyzer* ou um *network monitor* é um aparelho usado para determinar o desempenho da rede. A maioria deles é portátil, permitindo ser movidos facilmente. Um analisador pode monitorar um evento em específico como o número de frames por segundo ou o tamanho médio dos frames. Por exemplo, um analisador para uma rede CSMA/CD deveria reportar o número de colisões que ocorrerão. Mais importante, os analisadores são flexíveis, um gerente pode configurar o analisador para observar os frames enviados por uma determinada máquina, observar o tráfego de um tipo específico, ou computar a porcentagem de cada tipo de frame.

O hardware necessário para um analisador é surpreendentemente simples – muitos analisadores consistem em um computador portátil com uma interface de LAN. Quando é usado como analisador, o computador é dedicado completamente a essa função. Um software também é necessário para realizar a análise. O programa de análise começa permitindo ao usuário configurar os parâmetros, e utiliza esses parâmetros para analisar os pacotes.

Para ler os pacotes, o software coloca a interface de rede do computador no modo *promiscuous* o qual cancela o reconhecimento convencional de endereço. Assim o software configura o computador para receber todos os frames. Uma vez no modo *promiscuous*, a interface de rede não checa o endereço do destinatário, assim não rejeita nenhum frame. A interface simplesmente faz uma cópia de cada frame e armazena na memória do computador interrompendo a CPU para informar que um frame chegou.

Usuários mal informados podem assumir que um cartão de interface especial é requerido para o modo *promiscuous*. Surpreendentemente, não é esse o caso. Quase todas as interfaces de redes comercializadas suportam o modo *promiscuous*. Além do que, mudar uma interface para o modo *promiscuous* é trivial – apenas uma quantidade boa de instruções da CPU

precisa ser executada. Como consequência, qualquer usuário com um computador conectado a rede pode ler todos os pacotes que passam pela rede. Assim pode-se notar que mensagens em uma LAN não são totalmente privadas.

Para entender como isso é possível, considere como um analisador de rede trabalha. O analisador pode examinar os campos no cabeçalho do frame para determinar o emissor, o destinatário, o tipo, ou olhar na área dos dados.

Um analisador de rede é configurável: a exata configuração que um usuário seleciona determina que campo o analisador examine e qual informação ele mantém.

Assim um analisador pode tanto gerenciar o tráfego, ou checar todos os frames que um determinado computador envia.

Capítulo 5 – Cabeamentos de LAN, Topologia Física e Interface de Hardware

5.1 Introdução

Este capítulo continua a discussão das redes locais examinando o esquema de fios com mais detalhes. O capítulo começa considerando a interface de redes de cartão que conecta um computador a uma rede e trata dos detalhes da transmissão e recepção de pacotes (packets).

Depois de discutir interfaces, o capítulo descreve LANs convencionais e identifica os vários componentes usados em quase todos os esquemas de rede com fios, incluindo a descrição do hub.

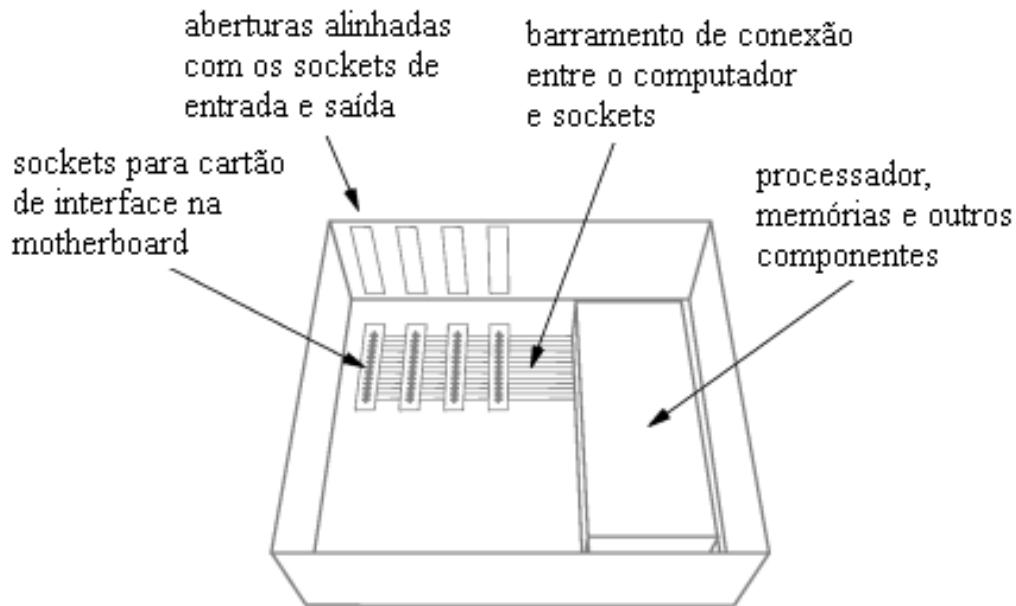
Por fim o capítulo discute a diferença entre topologia lógica e topologia física e mostra como os conceitos discutidos no capítulo sete são implementados na prática.

5.2 Velocidades de LANs e Computadores

Cada tecnologia de rede especifica uma taxa na qual os dados devem ser enviados. Surpreendentemente, muitas redes locais operam em taxas tão rápidas que a CPU é incapaz de processar os bits na velocidade da rede. A diferença entre a velocidade da rede e a do computador é um problema grave, se a rede acompanhar a velocidade do computador mais lento da rede ela irá prejudicar a velocidade de transferência entre dois computadores mais rápidos. As redes são feitas de modo a operar sempre utilizando a taxa de transferência máxima que o hardware pode suportar.

5.3 Interface de Rede

Numa transmissão via rede a CPU não lida com um bit individualmente, ele usa uma placa que lida com isso. Fisicamente, isso é um circuito impresso que contém vários componentes eletrônicos que interpretam os sinais elétricos e os converte de modo que a CPU possa ler os dados. Esse adaptador leva o nome de *network interface card* (NIC), essa placa é conectada ao processador, assim os dados que chegam via rede pode ser processado. A figura abaixo mostra os sockets, nos quais as placas de rede podem ser conectadas no computador.



Um NIC contém um circuito eletrônico capaz de operar independentemente do processador, ele é capaz de enviar e receber bits sem usar o processador do computador. A vantagem disso é que o processador do computador pode fazer outras coisas enquanto a placa de rede processa os bits e quando um pacote termina de ser enviado, a placa avisa o processador por meio de uma interrupção.

O mais importante disso tudo é saber que a maioria das redes transfere dados por diferentes meios a uma taxa fixa, normalmente essa taxa é superior à taxa que o computador é capaz de processar. Para evitar problemas no envio e no recebimento de dados, cada computador tem um hardware especial para lidar com isso, é a placa de rede (NIC), O NIC funciona como um dispositivo I/O: ele é construído para uma determinada tecnologia de rede e lida com os detalhes da transmissão de frames sem precisar do processador do computador.

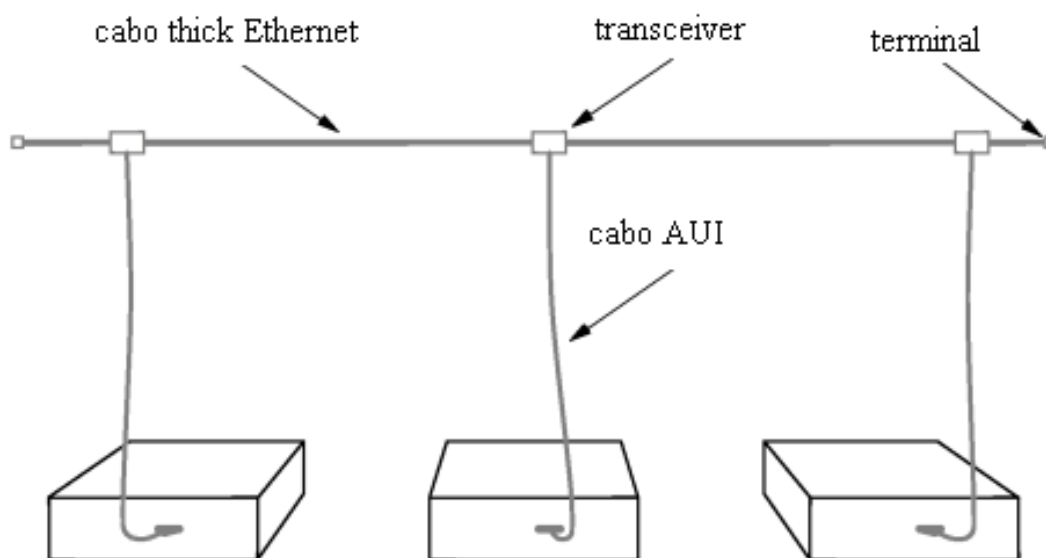
5.4 Conexão entre NIC e Rede

A conexão entre uma rede e uma placa depende da tecnologia da rede, em algumas tecnologias o NIC contém todo o hardware necessário para que um cabo de rede seja conectado diretamente nela, mas em muitas outras tecnologias o NIC não contém todo o hardware necessário, quando isso ocorre é preciso usar alguns componentes eletrônicos adicionais. Surpreendentemente, os detalhes de conexão entre rede e NIC não são determinados pela tecnologia, uma mesma tecnologia pode suportar vários esquemas de fiação. Aqui vamos analisar uma tecnologia que suporta 3 esquemas de fios: Ethernet.

5.5 Thick Ethernet

O Thick ethernet é um sistema de cabeamento de rede também conhecido informalmente pelo nome de Thickenet, formalmente recebe o nome 10Base5 e o meio de comunicação desse sistema é um cabo coaxial longo. Quanto ao hardware que lida com esse tipo de rede podemos dizer que se divide em duas partes principais: um circuito que lida com os aspectos digitais da comunicação (incluindo detecção de erros e reconhecimento de endereços) e um circuito que lida com os dados analógicos, conhecidos pelo nome de transceiver. O hardware que lida com os dados digitais não é capaz de converter ondas em sinais digitais, por essa razão faz-se necessário o uso do transceiver. O transceiver é um dispositivo de funcionamento simples, ele converte um sinal analógico em um digital. Em sistemas de rede

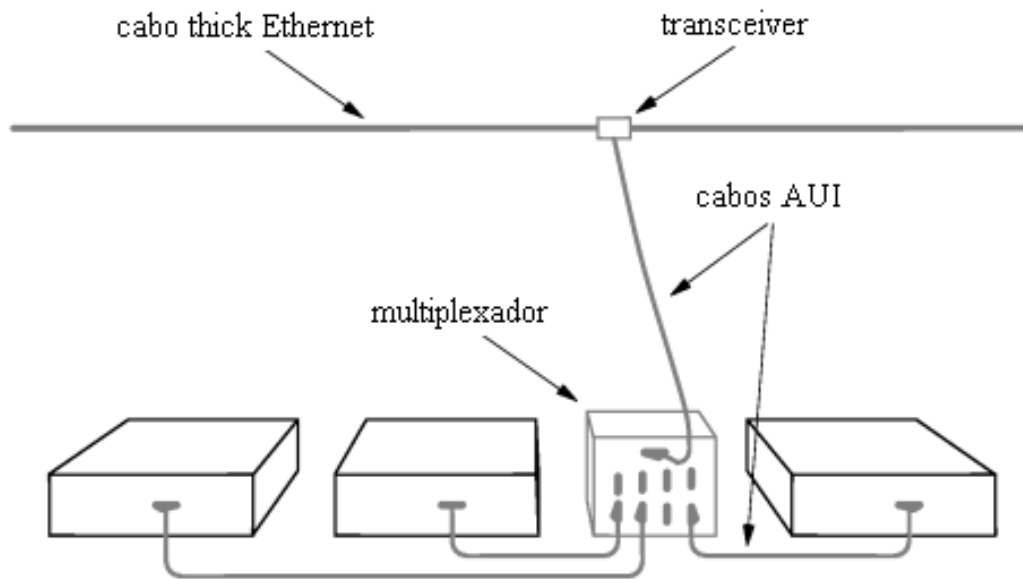
que usam thicknet cada computador precisa ter um transceiver, tal dispositivo fica acoplado no cabo principal (como a figura mostra) e é conectado ao computador através de um cabo que recebe o nome de AUI (Attachment Unit Interface), a placa de rede o conector que recebe esse cabo se chama AUI conector. Um AUI contém vários fios para transportar dados e alguns fios que são destinados a controlar o transceiver e levar energia para o circuito.



Prestando atenção na figura nota-se outro detalhe: um terminal na ponta do cabo principal. Esse terminal tem o objetivo de evitar que um sinal, ao chegar ao fim do cabo principal seja “refletido”. A ponta de um cabo funciona para um sinal elétrico como um espelho funciona para a luz, ou seja, reflete o sinal. Isso é terrível para o sistema de transmissão, o sinal que está voltando pode causar uma interferência no sinal que deveria ser enviado, modificando a informação original.

5.6 Multiplexador de Conexão

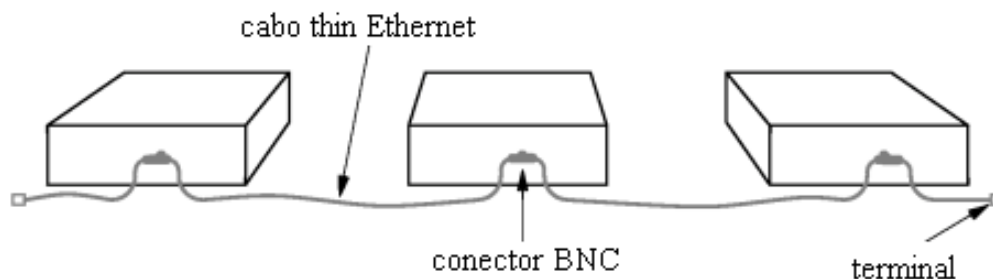
Em alguns casos a thicknet pode ser inconveniente por causa da distância entre o cabo principal e o computador. Pensando nisso engenheiros desenvolveram um dispositivo conhecido por multiplexador de conexão. Um multiplexador de conexão permite que vários computadores da rede utilizem um único transceiver, diminuindo a quantidade de AUI's. Ao invés de usar um AUI para cada computador da rede, esse esquema utiliza apenas um que é ligado ao multiplexador. A figura abaixo ilustra a idéia.



Cada computador tem sua própria placa de rede, mas ao invés de se conectar a rede através de um transceiver, ele faz isso se conectando ao multiplexador. Os AUI que convencionalmente se conectariam a um transceiver, se conectam ao multiplexador. O multiplexador faz o papel do transceiver para todos os computadores da rede.

5.7 Thin Ethernet

Os hardwares de rede disponíveis atualmente no mercado também aceitam outro tipo de cabeamento, o 10Base2, conhecido informalmente por *Thinnet*. Esse esquema utiliza um fio mais fino e mais flexível que a Thicknet, mas as diferenças não param por aí, existem pelo menos mais três pontos importantes nos quais os dois esquemas diferem. Primeiro: Thinnet é geralmente mais barato de se implantar que Thicknet. Segundo: a função do transceiver já está embutida na placa de rede, por isso é desnecessária a utilização de dispositivos externos. Terceiro: Thinnet não usam AUI's para conectar o computador ao meio de transmissão, nesse caso o cabo de rede é conectado diretamente na placa de rede através de um conector que recebe o nome de BCN.



Como se pode ver na figura, esse esquema também utiliza terminais nas pontas do cabo, no entanto o cabo coaxial é esticado entre pares de computadores.

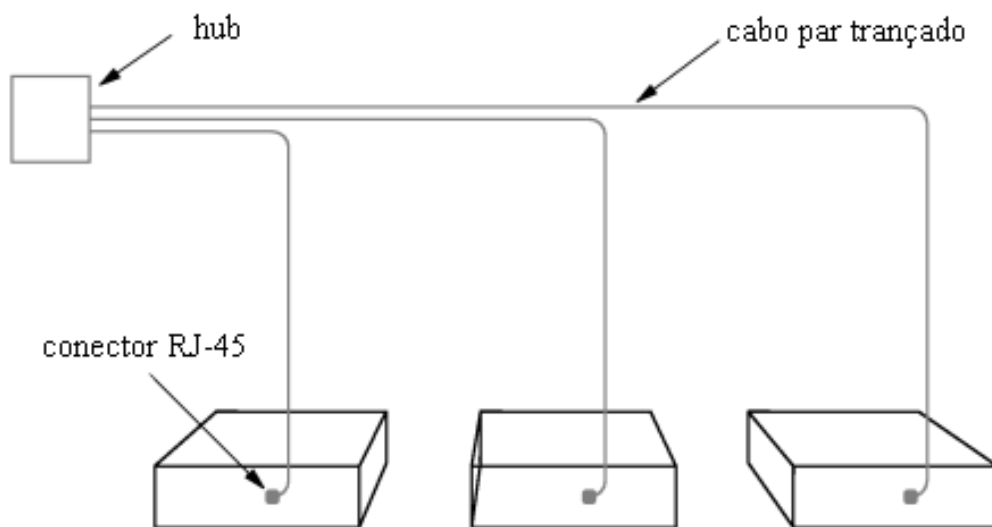
Apesar de Thinnet e Thicknet parecerem muito diferente eles tem muita coisa em comum; ambos usam cabo coaxial para proteger o sinal de interferências externas, ambos usam terminais nas pontas do cabo principal, ambos usam *bus topology* (topologia de barramento), e mais importante, os dois esquemas tem características elétricas semelhantes (resistência e capacitância), por isso o sinal transita ao longo do cabo da mesma forma nos dois sistemas.

5.8 Par Trançado

O terceiro tipo de cabeamento de rede Ethernet é o par trançado. Esse sistema difere drasticamente dos outros dois estudados até aqui. Ele é formalmente conhecido pelo nome 10Base-T e informalmente conhecido pelo nome de Par Trançado. Esse sistema não utiliza um cabo coaxial como os outros dois vistos acima, ou seja, esse esquema não tem um meio físico compartilhado como nos outros dois casos. Ao invés disso fez-se uso da idéia de multiplexagem, um dispositivo eletrônico é o centro da rede. Tal dispositivo recebe o nome de Ethernet hub, ou simplesmente hub.

O 10Base-T também requer que cada computador da rede tenha uma placa de rede, agora a conexão com a rede é feita diretamente na placa, sem utilização de transceiver. O conector recebe o nome de RJ-45. Em cada uma das pontas do cabo existe um conector RJ-45 uma ponta é conectada ao hub e a outra é conectada à placa de rede.

O hub simula um cabo coaxial principal, fazendo com que o sistema se comporte exatamente como rede Ethernet convencional. A diferença entre o sistema de cabeamento usado por uma rede não pode ser percebida por softwares, as placas de rede lidam com os detalhes e escondem as diferenças. Os hubs podem ser encontrados com diferentes quantidades de portas (a cada porta se conecta um computador), há hubs de 4 portas e há hubs que suportam redes grandes que chegam a ter centenas de portas.



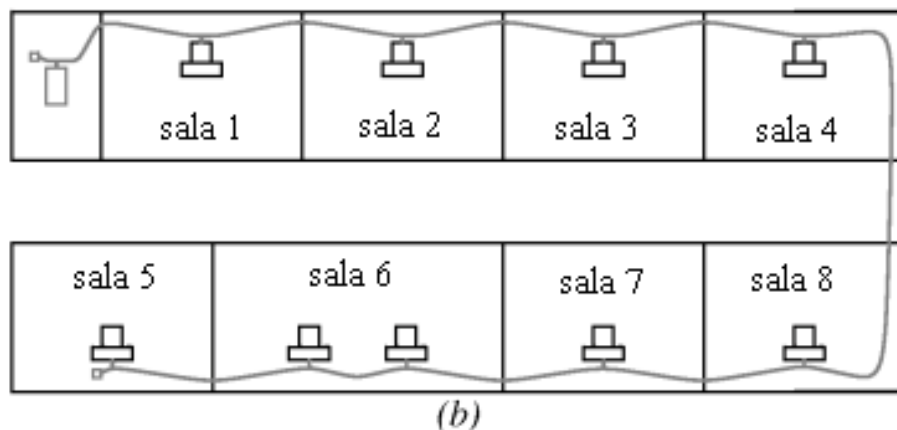
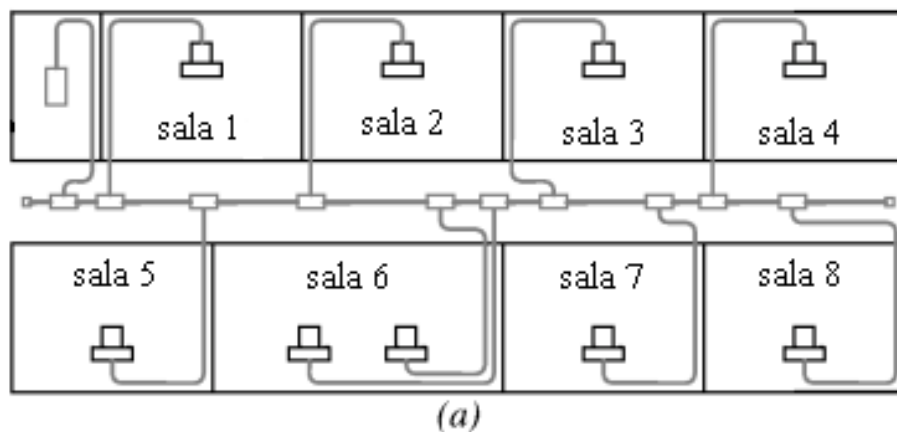
A figura acima ilustra um sistema de par trançado.

5.9 Vantagens e Desvantagens dos Sistemas de Cabeamento

Cada um dos três esquemas de cabeamento discutido acima apresenta vantagens e desvantagens. Cabeamentos que usam um transceiver para cada máquina permitem que um computador seja retirado da rede sem problemas, já no sistema Thinnet isso não é possível por causa da conexão ponto-a-ponto. Os transceiver muitas vezes são colocados em locais de difícil acesso, se um transceiver falhar, a localização e até mesmo a substituição do dispositivo pode se tornar complicada. No sistema Thinnet, se um computador for desconectado da rede todo o resto também será, no entanto para desconectar um cabo não é preciso de nenhuma ferramenta

especial, o conector BCN é muito fácil de ser desconectado. Redes que usam hub são mais imunes a desconexões acidentais, se isso acontece apenas uma máquina da rede vai ser afetada e todo o restante permanece funcionando.

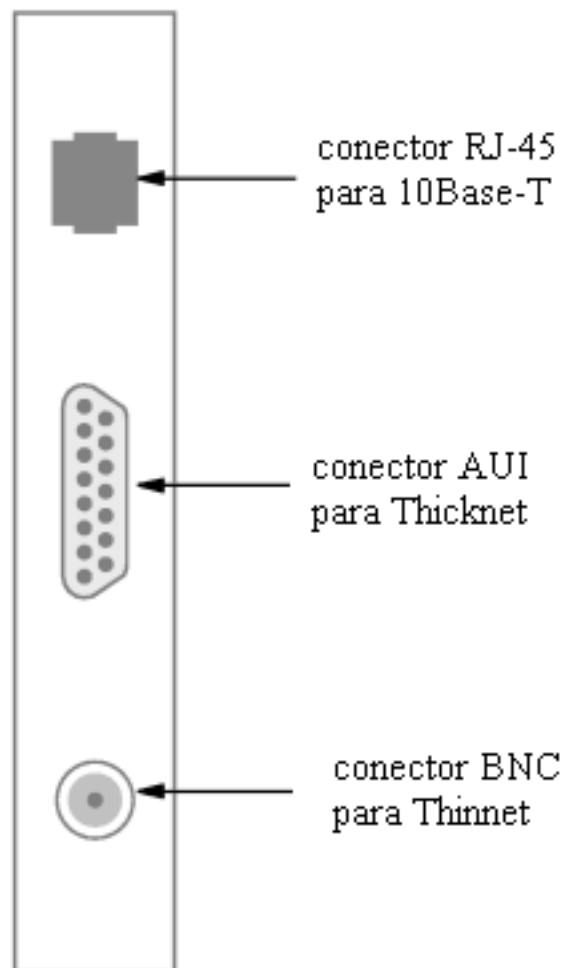
Hoje em dia, há um fator muito importante na escolha de qual tipo de cabeamento usar: o preço. Atualmente o sistema que vem dominando o mercado é o 10Base-T, por causa de seu baixo custo (comparando com os outros dois). Mas o preço depende de vários fatores, como, por exemplo, distância entre as máquinas, preço da placa de rede, da quantidade de micros da rede, o custo dos reparos, e da frequência com que novos computadores serão adicionados ou retirados dessa rede.



A figura compara uma rede de nove computadores usando diferentes sistemas de cabeamento. Repare que para este caso, por exemplo, a quantidade de fio poderia ser um fator importante na hora de escolher um esquema.

5.10 Placas de Rede e Esquemas de Cabeamento

Como existem vários tipos de cabeamento possíveis de se aplicar numa rede, é conveniente que uma mesma placa de rede aceite vários esquemas.



A figura ilustra a situação: uma mesma placa de rede pode ser usada no esquema Thicknet, Thinnet ou de Par Trançado.

Apesar de isso acontecer, apenas um conector pode estar funcionando por vez, não se pode ligar um computador a duas redes de dois esquemas diferentes ao mesmo tempo. Um software deve decidir qual dos três conectores está habilitado. Isso torna possível usar um computador com vários esquemas cabeamento, ou seja, a placa de rede não precisará ser trocada caso a rede seja substituída por outra que usa outro esquema de cabeamento.

Mas o melhor de tudo isso é que o endereço do computador na rede é determinado não pelo sistema de cabeamento, mas sim pela placa de rede. Por isso o endereço físico de uma máquina continua inalterado quando se substitui o sistema de cabeamento por outro.

Capítulo 6 – Extensão das LANs

6.1 Introdução

Cada tecnologia de rede se aplica a uma situação diferente, e os fatores, que especificam qual tipo usar, são geralmente velocidade, distância e principalmente o custo. Quando se “desenha” uma LAN deve-se especificar a distância máxima que ela poderá se expandir (normalmente algumas centenas de metros). Por isso a grande maioria das LANs se restringe a um prédio apenas.

Muitas vezes computadores de uma rede precisam estar a uma distância maior do que aquela que a rede pode se estender. Por isso foram desenvolvidos mecanismos capazes de entender uma rede a uma distância maior. Este capítulo trata desse assunto. As palavras-chave do capítulo são: extensão de LANs em longas distâncias, uso dos modem de fibra, repetidores e bridges.

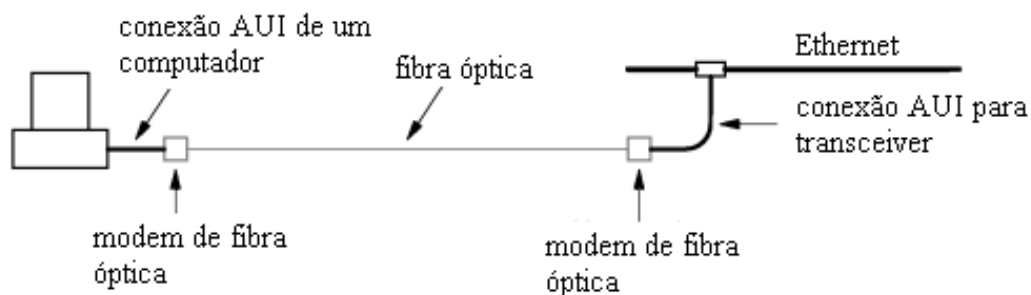
6.2 Limitação pela Distância e Design de LAN

O fato de uma rede só poder se estender a uma determinada distância é um fator importantíssimo na hora de desenhar uma rede. Quando se está projetando uma rede, deve-se fazer uma combinação de capacidade, máxima delay, e máxima distância para economizar.

Para garantir que o delay não comece a ser significativo na hora da transmissão de dados, as LANs são desenhadas para trabalharem com um tamanho máximo de cabo, ultrapassar esse comprimento significa incluir delays na transmissão. Outro problema que limita a distância é o hardware. Uma placa de rede não pode emitir pulsos elétricos arbitrariamente, então, acontece que se a distância for demasiadamente longa o sinal começa a se perder no fio de cobre. A seguir serão discutidos alguns artifícios que podem ser usados para driblar o problema das limitações trazidas pela distância.

6.3 Extensões de Fibra Óptica

Já que a distância é uma limitação no comprimento de uma rede, a solução encontrada por engenheiros da área foi fazer uma extensão. O tipo mais simples de extensão é o que usa uma fibra óptica e um par de modems de fibra óptica. A idéia desse mecanismo pode ser vista na figura abaixo.



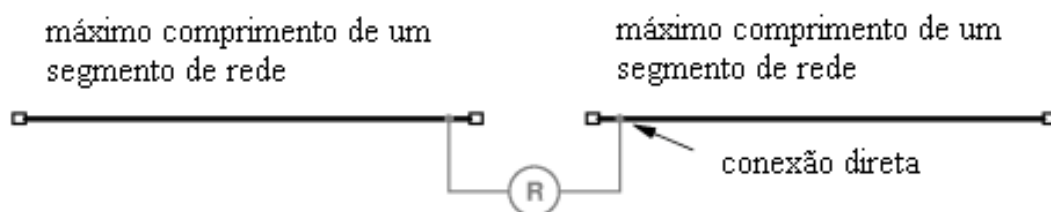
Devido ao baixo delay e a alta condutividade da fibra óptica, alguns mecanismos podem aceitar que um computador seja conectado a uma rede remota. Para se usar uma extensão dessas

deve-se proceder como na figura: os modems servem para transformar o sinal analógico (pulso elétrico) em luz, para que o dado possa ser transportado pela fibra. A recíproca também é verdadeira.

A principal vantagem dessa extensão é poder conectar um computador à rede a uma distância absurdamente grande (vários quilômetros), a desvantagem é que cada computador precisa de um par de modems, o que encarece esse tipo de extensão. É importante lembrar que essa extensão se aplica entre uma rede e um computador.

6.4 Repetidores

Para ultrapassar as limitações de distância algumas tecnologias de rede permitem que dois cabos sejam colocados juntos em um dispositivo conhecido pelo nome de *repetidor*. Um repetidor é um dispositivo analógico que monitora constantemente o sinal elétrico nas pontas dos cabos. Quando um sinal chega ao dispositivo, ele (o repetidor) emite uma cópia ampliada no outro cabo. A figura mostra um esquema usando repetidor.



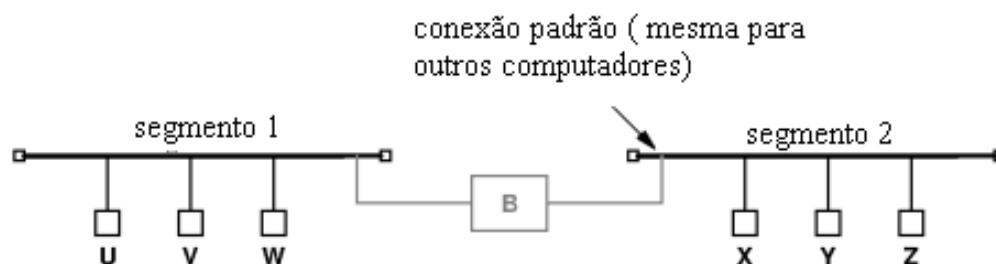
Um repetidor conecta-se em dois cabos cada um com terminais nas pontas (isso pode ser observado na figura acima). Os repetidores não conseguem identificar um frame por isso eles apenas emitem cópias idênticas do sinal que recebem. O máximo comprimento que um segmento de rede pode ter é 500 metros, no entanto, se usarmos um repetidor esse comprimento pode dobrar, mas numa LAN estendida podemos ter no máximo quatro repetidores separando duas estações. Isso acontece porque cada repetidor acrescenta um delay, se o delay começa a ficar significativo o esquema começa a falhar.

Os repetidores apresentam alguns problemas, o mais grave deles é o fato de um repetidor não conseguir identificar um frame. Quando esse dispositivo recebe um sinal, ele não faz nenhuma verificação de erros, por isso se um sinal sofrer uma interferência eletromagnética antes de chegar ao repetidor, esse sinal continuará a se propagar.

6.5 Bridges

Uma bridge é o dispositivo eletrônico que conecta dois segmentos de rede, mas diferentemente dos repetidores, uma bridge consegue identificar frames completos. Quando uma bridge recebe um frame de um dos segmentos conectados a ele, verifica se o frame está chegando intacto e só depois envia uma cópia para o outro segmento se necessário. Um bridge unindo dois segmentos transforma-os em uma única rede. A figura abaixo ilustra esse conceito.

O uso de bridge se tornou mais popular do que o uso de repetidores porque as bridges evitam a propagação de erros e faz isso comparando o frame que foi enviado pelo computador com o frame que ele recebeu. Se esses dois frames forem iguais, a bridge transmite o frame para o computador de destino.



6.6 Filtro de Frames

A maioria dos bridges faz mais do que simplesmente transmitir uma cópia de frame de um segmento para outro. A função mais valiosa de uma bridge é filtrar os frames – uma bridge não transmite um frame desnecessariamente para um segmento.

Particularmente, se um computador que está num segmento envia um frame para um computador que este no mesmo segmento, a bridge não precisará enviar esse frame para outro segmento. Para determinar para onde enviará o frame, a bridge usará o endereço físico encontrado no cabeçalho do frame. Para identificar quais os computadores estão conectados a cada um dos segmentos a bridge executa duas tarefas. Primeiro: extrai o endereço do cabeçalho do frame e adiciona este endereço na lista do segmento que o computador está. Segundo: extrai o endereço de destino do frame e procede da mesma maneira que no primeiro passo. Portanto, a bridge só aprende a localização de um computador depois que ele envia seu primeiro frame. Para ilustrar, observe a figura abaixo.

Evento	Lista segmento 1	Lista segmento 2
Inicialização	-	-
U envia para V	U	-
V envia para U	U,V	-
Z broadcasts	U,V	Z
Y envia para V	U,V	Z,Y
Y envia para X	U,V	Z,Y
X envia para W	U,V	Z,Y,X
W envia para Z	U,V,W	Z,Y,X

6.7 Startup e Steady State de Redes com Bridge

Depois que todos os computadores que estão conectados a uma rede com um bridge enviam um frame, o bridge aprende a localização de cada um deles e usa essa informação para fazer o filtro de frames. Como resultado disso, uma rede com bridge só envia frames para outro segmento se isso for necessário.

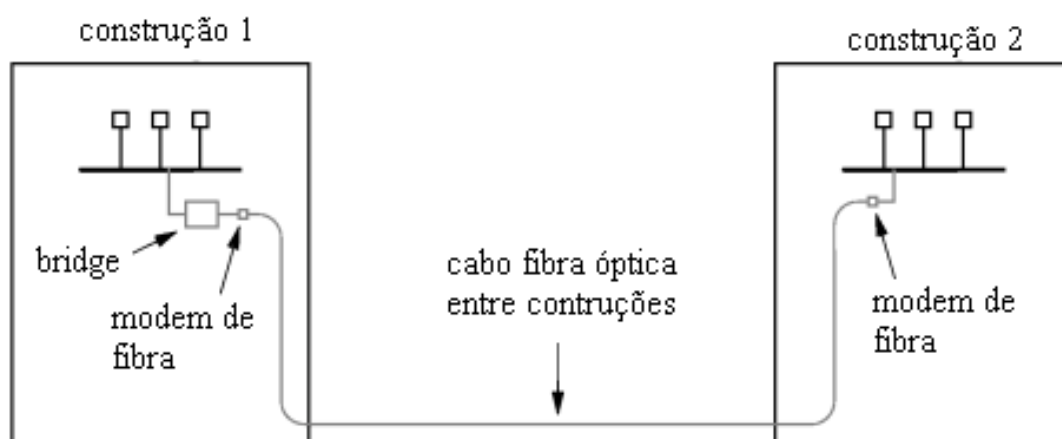
O princípio de propagação de redes com bridges nos diz que no [steady state](#) o bridge envia cada frame apenas a distância necessária. No primeiro boot o bridge ainda não sabe a localização de cada computador e se um computador não enviar nenhum frame sua localização não será detectada pelo bridge. Felizmente, a comunicação entre computadores é normalmente bidirecional, ou seja, quando num computador recebe um frame normalmente ele envia uma

resposta. Por esta razão o bridge consegue identificar rapidamente a posição de todos os computadores ligados à rede.

Quando se deseja construir uma rede com bridge, deve-se colocar os computadores que se comunicam com mais frequência em um mesmo segmento. Isto porque devido ao princípio de propagação, se um computador for enviar um frame para um computador do mesmo segmento, os computadores do outro segmento poderão se comunicar ao mesmo tempo.

6.8 Bridge entre Construções

Do mesmo modo que os repetidores, os bridges também podem ser usados para expandir uma rede a uma determinada distância. Se, por exemplo, dois prédios estão separados por uma distância significativa uma LAN convencional não será capaz de fazer a comunicação entre elas. Se para solucionar este problema for usada uma extensão de fibra ótica, o custo da extensão será altíssimo, isto porque para cada computador adicional será necessário um par de modems de fibra e a fibra ótica propriamente dita. Uma solução que diminuiria bastante o preço seria usar apenas um bridge, um par de modems de fibra e uma única fibra.

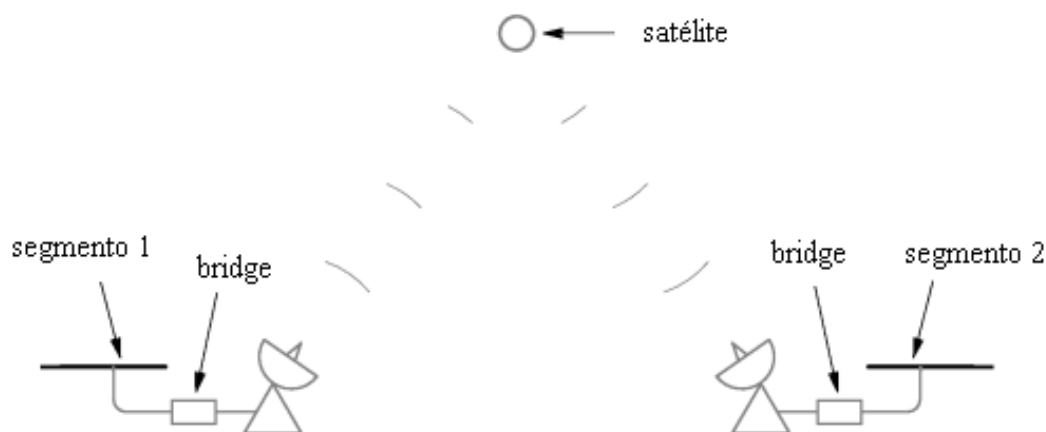


Neste caso a extensão conecta dois segmentos ao invés de conectar um computador ao segmento. Essa solução traz três vantagens principais:

Primeira: requer apenas uma fibra para conexão o que diminui o custo; Segunda: pelo fato de estar conectando dois segmentos, um computador pode ser facilmente adicionado ou removido da rede; Terceira: devido ao fato de uma rede com bridge aceitar comunicação simultânea nos dois segmentos, mesmo que um dos segmentos esteja congestionado o outro pode trabalhar sem problemas, desde que a comunicação ocorra no próprio segmento.

6.9 Bridges em Redes de Longa Distância

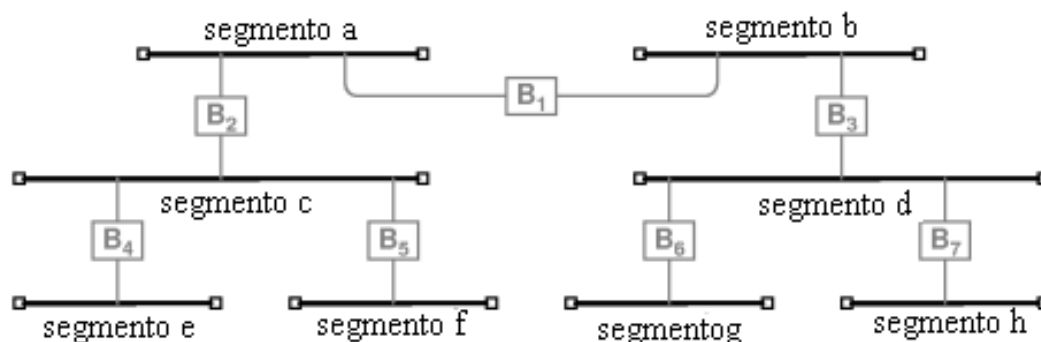
Uma rede com bridge pode se expandir por uma distância muito longa de dois modos diferentes. O primeiro usa uma linha para conectar os dois lados, o segundo usa satélites. Normalmente o uso de linhas é o mais comum porque é mais barato. Tanto a linha quanto o satélite podem ser alugados. Neste tópico analisaremos a comunicação via satélite, ela permite comunicação a qualquer distância.



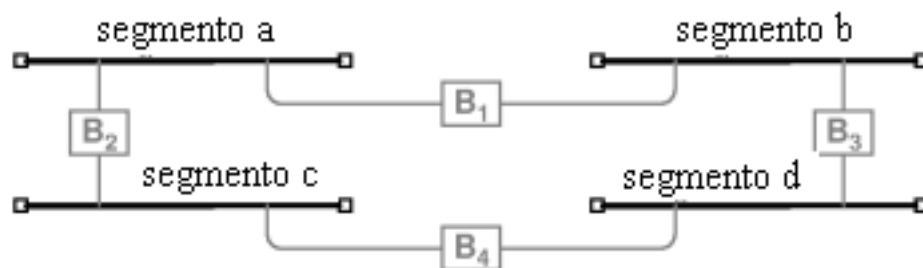
Como pode ser visto na figura, o hardware usado nesta situação é completamente diferente do usado na comunicação local, note que cada segmento tem um bridge. Neste tipo de comunicação é comum usar o filtro de frame em ambos os lados, isto porque os satélites são alugados e quanto menos frames forem enviados menor será o custo do aluguel. Conseqüentemente, um canal de satélite opera a uma capacidade menor do que um segmento. O resultado é que o canal do satélite não é capaz de enviar todos os frames de um segmento para outro na mesma velocidade que um fio ou uma fibra o faria. Para evitar problemas de comunicação o bridge, usado nesses casos, faz um *buffer* dos frames excedentes que chegam ai. Fazer um buffer significa salvar uma cópia dos frames que estão sendo enviados na memória até que eles possam ser enviados. “Bufferizar” não resolve todos os problemas deste tipo de transmissão, se os frames continuarem chegando a uma velocidade maior do que os satélites podem enviar, o bridge começará a descartar os frames que chegarem por último.

6.10 Bridges em Círculos

Uma rede pode ter vários bridges para interligar os segmentos e apesar de cada bridge acrescentar um pequeno delay, ele não é suficientemente grande para que a rede para de funcionar corretamente. Abaixo está um exemplo de construção de rede com vários bridges.



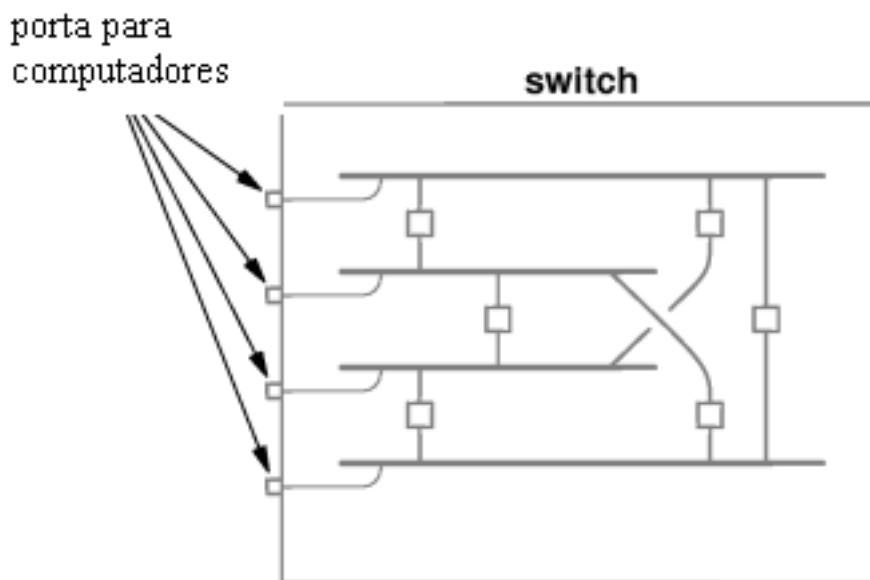
Se uma rede com bridges fizer um círculo, como na figura abaixo, podem acontecer problemas na hora da transmissão dos dados. Considere o que acontece se um computador envia um frame. Com uma rede aberta, a informação chegaria ao computador de destino e depois seria descartada. No entanto, com o sistema em círculo, o frame fica passando de um segmento para o outro infinitamente, isso significa que a rede fatalmente ficará congestionada.



Para evitar esse problema de looping infinito, uma rede com bridge não pode permitir que todos os bridges enviem todos os frames simultaneamente e, é claro, evitar que a rede tenha um círculo de bridges. Isso na prática pode se tornar difícil numa rede extensa, mas por sorte a prevenção pode ser automática.

6.11 Switching

O conceito de bridge é fundamental para entender um switch. Um switch é um dispositivo que pode transferir frames, como uma bridge. Um switch lembra um hub quando visto externamente, mas do ponto de vista de funcionamento eles são diferentes: um hub funciona como um cabo coaxial, ou seja, como um meio. Já o switch funciona simulando uma rede de bridges com um computador por segmento (a figura abaixo ilustra essa idéia).



A principal vantagem de usar um switch no lugar de um hub é a mesma de se usar uma bridge no lugar de um único segmento: o paralelismo. Como um hub simula um segmento apenas dois computadores podem trocar informações simultaneamente, o switch simula uma rede com bridge, e devido ao princípio da propagação computadores em segmentos diferentes podem se comunicar ao mesmo tempo. Como o switch simula uma rede com um computador por segmento, a rede dificilmente ficará congestionada e a velocidade do tráfego de dados aumenta quando comparada com a de uma rede com hub.

O problema do switch é seu preço, que é maior que o preço de um hub. Muitas empresas que tem redes de computadores com muitos computadores utilizam combinações de

hubs e switches para montar suas redes a um custo menor. A idéia principal é ligar um hub em cada uma das portas do switch, assim a quantidade de entradas do switch fica multiplicada pelo número de entradas de cada hub. Apesar da velocidade do tráfego de frames diminuir, o custo é bem menos. Esses sistemas também funcionam como redes com bridge convencionais.

Capítulo 7 – WAN e Roteadores

7.1 Introdução

O capítulo anterior tratou de tecnologias que ofereciam comunicação digital por longas distâncias e que entregavam comunicação digital aos assinantes. Esse capítulo vai mostrar como aquelas tecnologias podem ser usadas para construir redes que se expandem por grandes áreas. O capítulo vai descrever os componentes básicos necessários para se montar uma rede com packet switch. Como parte do assunto será discutido o conceito fundamental do roteador e será mostrado como ele é usado nas WANs.

7.2 Grandes Redes e Grandes Áreas

As redes são, em geral, classificadas dentro de três categorias. A classificação depende do tamanho que a rede pode atingir assim os três grupos são explicados abaixo:

- Local Área Network (LAN). Pode se estender por um simples prédio ou campus;
- Metropolitan Area Network (MAN). Pode se estender por toda uma cidade;
- Wide Área Network (WAN). Pode se estender por grandes áreas, cidades, países e até mesmo continentes.

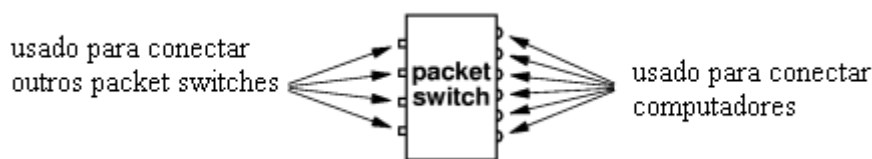
Temos visto ao longo dos capítulos que apesar de uma LAN ser desenvolvida para ser usada apenas em um prédio, existem maneiras de estendê-la (satélite com uma bridge pode estendê-la por qualquer distância). Apesar disso, uma LAN estendida não pode ser considerada uma WAN porque não é possível conectar muitos computadores e não é possível que se tenham vários “lados”.

A questão básica que separa WAN de LAN é a escala. Uma WAN deve ser capaz de crescer o quanto for necessário para conectar quantos lados forem necessários e com quantos computadores forem necessários em cada lado. Felizmente uma tecnologia não é classificada como WAN a menos que ela seja capaz de permitir a comunicação entre vários computadores simultaneamente. É por esse motivo que se usa um packet switch.

7.3 Packet Switch

É interessante pensar em como uma WAN consegue lidar com tantos computadores. Para entender isso é preciso primeiro entender que a rede deve ser capaz de crescer. No entanto, usando uma conexão ponto-a-ponto com circuitos alugados (como no capítulo anterior) que conecta um computador diretamente a outro, é possível construir uma WAN, para isso basta adicionar switches para conectar computadores individuais. Depois que uma WAN foi montada é possível aumentar seu tamanho facilmente apenas adicionando outros switches.

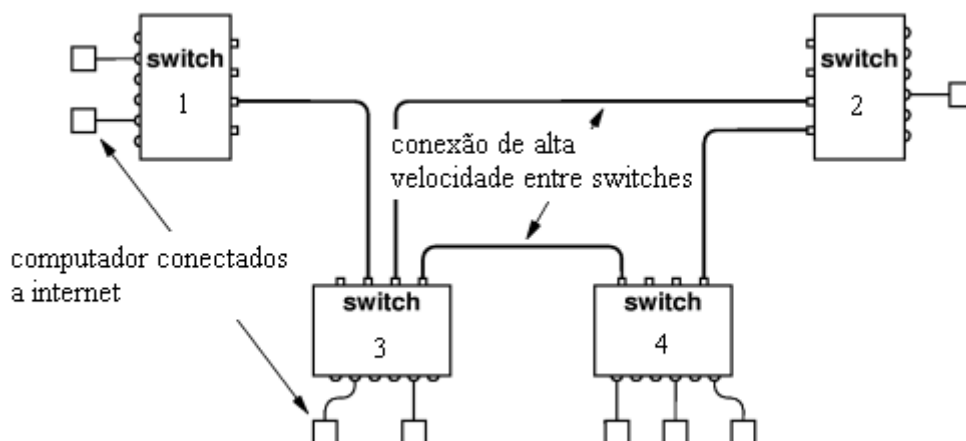
O switch usado nas WANs recebe o nome especial de packet switch, isto graças a sua habilidade de mover pacotes inteiros de computador para computador. Conceitualmente, cada packet switch é um **peque** no computador com processador, memória, e dispositivos I/O usados para enviar e receber os pacotes. A figura está mostrando um packet switch esquematicamente.



Uma parte do dispositivo I/O é usada para operar a altas velocidades e conecta o switch à linha alugada, que se conecta ao outro switch. Assim sendo a transmissão de dados entre os packet switch se processa em alta velocidade. A outra parte do dispositivo I/O opera em baixas velocidades e é usada para conectar o computador ao switch. A transmissão de dados entre computadores “pendurados” em um mesmo packet switch acontece à baixa velocidade. É importante ressaltar que todas as formas de comunicação ponto-a-ponto são usadas para montar as WANs, incluindo circuitos alugados, fibras ópticas, micro ondas e canais de satélites.

7.4 Formando uma WAN

Um grupo de packet switches deve ser interconectado para formar uma WAN. Cada switch tem, normalmente, vários conectores de I/O, por isso é possível formar várias topologias diferentes e conectar vários computadores. A figura logo abaixo ilustra um modo possível de se montar uma WAN interconectando quatro packet switches e oito computadores.



Como a figura mostra a WAN não precisa ser simétrica; a quantidade de interconexões e a capacidade de cada conexão são escolhidas para acomodar o tráfego esperado e promover redundâncias em caso de falhas em uma das interconexões.

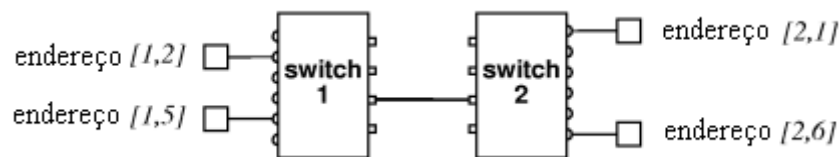
7.5 Store e Forward

Diferentemente de uma LAN, uma WAN permite que vários computadores se comuniquem ao mesmo tempo enviando e recebendo frames. Para fazer isso existe um sistema chamado de *store and forward*. Nesse sistema a packet switch cria um buffer dos pacotes em sua memória. O procedimento de store (estocar) acontece quando o frame chega ao switch. Uma cópia do frame fica guardada na memória do packet switch. O procedimento de forward acontece em seguida. O processador examina o cabeçalho do frame e determina qual interface deve ser usada para enviá-lo e então inicia o envio. Esse sistema é bom porque permite a transmissão em alta velocidade (a velocidade é a máxima que o hardware suportar), e o mais importante é que se vários pacotes precisarem ser enviados pela mesma saída do dispositivo, o

packet switch pode salvar os pacotes em memória até que a saída esteja livre para poder ser usada.

7.6 O Endereço Físico

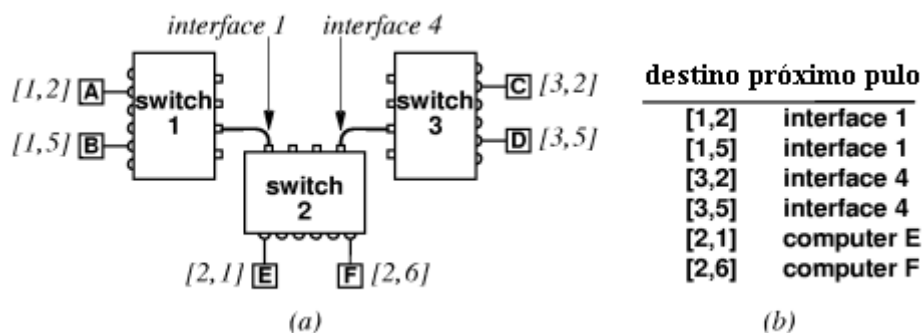
Do ponto de vista do computador, individualmente falando, uma WAN se comporta do mesmo modo que uma LAN. Mas na verdade não é bem assim. Numa WAN cada computador deve ter um endereço físico que deve determinar em qual porta de qual packet switch o computador está ligado. Isso fica óbvio de se imaginar, pois quando um computador mandar um frame pela WAN, este frame precisará ir para um destino. Muitas WANs usam um sistema hierárquico de endereçamento para fazer o forward com maior eficiência. No sistema hierárquico o primeiro número indica qual o packet switch no qual o computador está conectado, e o segundo número indica qual a porta do switch está sendo usada. A figura ajuda a imaginar a situação descrita acima.



7.7 Next-Hop Forward

O packet switch deve escolher uma de suas saídas através da qual enviará o pacote, e essa escolha é feita conforme o cabeçalho do próprio pacote. Se o destino for um dos computadores conectado ao próprio packet switch em questão, ele faz o envio direto (à baixa velocidade), mas se o pacote tem como destino um computador conectado a outro packet switch da WAN, a transmissão será feita por meios alugados e à alta velocidade.

O packet switch não tem informações sobre todos os destinos possíveis da WAN, mas ele tem informações sobre o “próximo pulo” (next-hop forwarding). O conceito é um pouco abstrato, mas para compreendê-lo imagine que na figura abaixo haja um frame no switch 2. A tabela mostra o next-hop para todas as possibilidades.



7.8 Relação de Endereçamento Hierárquico para o Roteador

A tabela usada para estocar o next-hop (tabela do item 12.7 acima) é chamada de *routing table*, e o processo de forward do pacote é conhecido pelo nome de *routing*. A vantagem do endereçamento hierárquico fica evidente agora. Quando um frame chegar a um packet switch para ser enviado, o packet switch precisará identificar apenas o packet switch de destino, ou seja, precisará ler apenas o primeiro número do cabeçalho do frame. Depois que o frame chegar ao seu destino, o outro packet switch terá que identificar apenas o segundo número do endereço que está no cabeçalho para direcionar ao computador certo. Isso trás algumas conseqüências práticas, sendo que a principal é tornar o tempo do processo menor do que aquele que seria gasto para se ler todo o endereço.

Destino	Próximo pulo
(1, indiferente)	Interface 1
(3, indiferente)	Interface 4
(2, indiferente)	Computador local

A tabela acima é uma tabela abreviada da anterior. Todos os frames, que possuem o switch 1 como destino, são enviados através da interface 1, o processo é igual para os outros destinos. Cabe ao switch de destino analisar e direcionar o frame para um dos computadores conectados a ele.

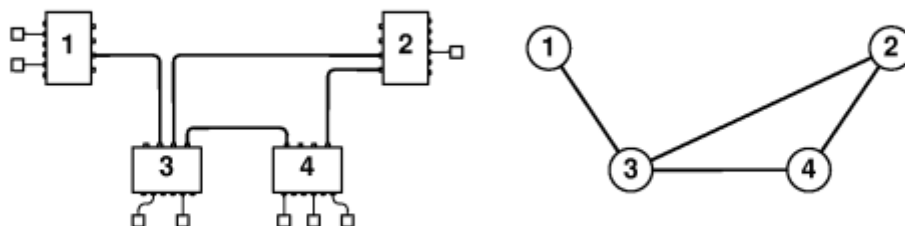
7.9 Roteamento numa WAN

A capacidade de uma WAN deve ser aumentar com a adição de computadores conectados à rede. Para lidar com a adição de poucos computadores, a capacidade de um switch deve ser aumentada adicionando portas de I/O ou fazendo a substituição por um switch mais rápido. Tais mudanças podem acomodar pequenos aumentos no tamanho da rede. Para fazer mudanças significativas no tamanho da rede é preciso adicionar novos packet switches.

Para que uma WAN trabalhe corretamente é preciso que os valores da routing table garantam o seguinte:

- Roteamento Universal - A tabela de roteamento do switch deve conter o next-hop para todos os destinos possíveis;
- Rotas Ótimas - Num switch, o next-hop da tabela de roteamento para um dado destino deve ser o ponto mais próximo do destino.

Uma maneira fácil de visualizar a situação descrita é imaginar um modelo representativo para a WAN. Na figura abaixo temos dois esquemas para entender melhor o roteamento.



Agora vejamos a routing table para esse esquema.

Destino	Próximo pulo	Destino	Próximo pulo	Destino	Próximo pulo	Destino	Próximo pulo
1	-	1	(2,3)	1	(3,1)	1	(4,3)
2	(1,3)	2	-	2	(3,2)	2	(4,2)
3	(1,3)	3	(2,3)	3	-	3	(4,3)
4	(1,3)	4	(2,4)	4	(3,4)	4	-
Nó 1		Nó 2		Nó 3		Nó 4	

Onde se lê (1,3), por exemplo, entenda como interface entre 1 e 3. repare que a ordem é importante.

7.10 Uso de Rotas Padrão

A routing table do nó 1 ilustra uma idéia importante: apesar de o endereçamento hierárquico reduzir o tamanho da routing table retirando as rotas repetidas, ainda existem muitas entradas com o mesmo next-hop. Em exemplos pequenos essa lista de coisas repetidas pode até parecer pequena, mas quando a WAN tem tamanho consideravelmente grande isso fica um pouco complicado. Para resolver esse problema a maioria das WANs tem um mecanismo que elimina as duplicatas. O sistema é conhecido pelo nome de *default route*. O mecanismo faz uma simplificação da routing table. A Tabela acima simplificada fica da seguinte maneira:

Destino	Próximo pulo	Destino	Próximo pulo	Destino	Próximo pulo	Destino	Próximo pulo
1	-	2	-	1	(3,1)	2	(4,2)
*	(1,3)	4	(2,4)	2	(3,2)	4	-
		*	(2,3)	3	-	*	(4,3)
				4	(4,3)		
Nó 1		Nó 2		Nó 3		Nó 4	

Note no nó 1 que para qualquer destino diferente do próprio nó 1 o next-hop é o mesmo.

7.11 Formando a Routing Table

Apesar de ser fácil montar manualmente uma routing table para uma rede pequena o processo não é nada simples para redes grandes. Por esse motivo são usados softwares para fazer isso. Tais softwares seguem dois princípios básicos:

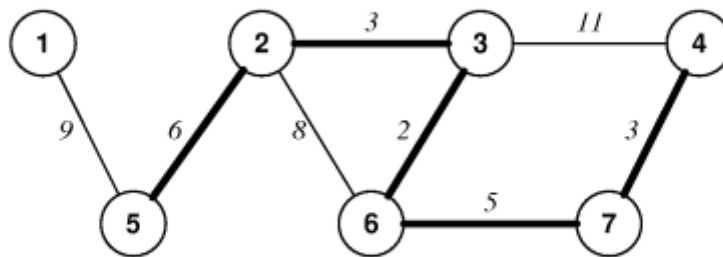
- Static Routing - O programa calcula e instala as rotas quando o pacote switch é iniciado; as rotas não mudam;
- Dynamic Routing - O programa constrói uma routing table inicial quando o packet switch é iniciado, mas a routing table pode ser alterada quando a rede sofrer alterações.

Cada tipo de roteamento tem suas vantagens e desvantagens. A principal vantagem do static routing é sua simplicidade, e a principal desvantagem é a inflexibilidade. A maioria das WANs usa um sistema de dynamic routing porque ele aceita que a rede trate de problemas automaticamente. Se uma interface apresentar um problema e ficar inutilizável o roteador modifica a routing table de modo a corrigir o problema.

7.12 Calculando o Caminho mais Curto

O software que calcula toda a routing table representa a rede como um gráfico. Para calcular as distâncias entre os computadores ligados à rede o software usa o chamado *Algoritmo de Dijkstra*. Este algoritmo procura o caminho mais curto até cada um dos computadores a partir de um mesmo nó, o procedimento é repetido para cada um dos nós. Assim, a routing table é construída, com os caminhos mais curtos. O algoritmo deve rodar uma vez por cada routing table. O algoritmo de Dijkstra se popularizou pelo fato de poder ser usado com várias definições de caminho mais curto, de modo particular, o algoritmo não precisa de bordas no gráfico para representar distâncias geográficas. No entanto o algoritmo aceita que cada borda seja marcada com um valor não negativo chamado *peso*, e define a distância entre dois nós com sendo a soma dos pesos ao longo do caminho entre os nós. A figura abaixo ilustra um caso: a idéia é calcular a menor distância entre os nós 4 e 5.

A distância entre dois nós é marcada com um peso. A menor distância é aquela na qual a soma dos pesos é a menor possível.



Abaixo se tem a idéia básica do algoritmo de Dijkstra.

O algoritmo acima mostra como a routing table pode ser computada depois que as informações da rede são codificadas em forma de gráfico. Uma técnica alternativa para se descobrir o caminho mais curto é a técnica *distributed route computation* na qual cada packet switch calcula uma routing table local e então envia sua routing table pela rede para os packet switches vizinhos.

Redes que usam essa técnica fazem com que cada packet switch envie informação sobre as rotas para a vizinhança periodicamente (poucos segundos). Depois de um período inicial, cada packet switch aprende o caminho mais curto para todos os destinos possíveis da rede, e a routing table gerada é a mesma que seria gerada pelo outro método. Se por acaso um packet switch falhar, ou seja, parar de enviar informação sobre as rotas periodicamente, os outros packet switch da rede aprendem as rotas que ele fazia e continuam a enviar informações para os computadores, se isso for possível.

7.13 Rota de Distância Vetorial

Um dos melhores algoritmos conhecidos para calcular a *distributed route computation* é o *distance vector*. Como no algoritmo mostrado acima, cada link é marcado com um peso, e a distância até o destino é definida como sendo a soma dos pesos ao longo do caminho. Por causa do valor da distância precisar ficar registrado, um campo adicional é requerido em cada entrada da routing table. O campo adicional contém a distância para os destinos ao longo do caminho que corresponde ao próximo destino. [O algoritmo abaixo mostra a idéia desse algoritmo.](#)

7.14 Link-State Routing (SPF)

Uma solução para a distribuição de rotas que usa uma versão do algoritmo de Dijkstra misturada com a rota de distância vetorial é chamada formalmente de Link-State Routing e informalmente recebe o nome de Shortest Path First (SPF).

Apesar dos packet switches que usam SPF enviarem mensagens através da rede, as mensagens não contêm informações sobre as tabelas de roteamento. Apesar disso cada mensagem carrega o status do link entre dois packet switches, e a mensagem é transmitida para todos os outros switches. Cada switch coleta o status da mensagem que está chegando e o usa para construir o gráfico da rede. A partir daí o switch pode usar os dados no algoritmo de Dijkstra para produzir a tabela de roteamento tomando ele mesmo como fonte.

O SPF também é capaz de se adaptar a situações de erro além de ter a vantagem de que todos os cálculos de distância podem ser feitos ao mesmo tempo.

7.15 Exemplos de Tecnologias WAN

Muitas tecnologias têm sido criadas para experiências e acabam sendo usadas em WANs. As tecnologias de WAN serão apenas citadas, em capítulos posteriores elas serão abordadas com a devida importância: ARPANET, X.25, Frame Relay, SMDS, ATM entre outras.

Capítulo 8 – Proprietário da Rede, Padrão de Serviços, e Desempenho

8.1 Introdução

Cada tecnologia de rede é classificada em uma dessas três categorias: LAN, MAN, ou WAN, com a maioria sendo classificada como LAN ou WAN. Essa classificação é útil porque ajuda a comparar a arquitetura das redes.

Este capítulo irá discutir três características adicionais de redes: proprietário da Rede, tipo de serviços providos por uma rede, e o desempenho do sistema resultante.

8.2 Proprietário da Rede

O fato de uma rede só poder se estender a uma determinada distância é um fator importantíssimo na hora de desenhar uma rede. Quando se está projetando uma rede, deve-se fazer uma combinação de capacidade, máximo delay, e máxima distância para economizar.

Para garantir que o delay não comece a ser significativo na hora da transmissão de dados, as LANs são desenhadas para trabalharem com um tamanho máximo de cabo, ultrapassar esse comprimento significa incluir delays na transmissão. Outro problema que limita a distância é o hardware. Uma placa de rede não pode emitir pulsos elétricos arbitrariamente, então, acontece que se a distância for demasiadamente longa o sinal começa a se perder no fio de cobre. A seguir serão discutidos alguns artifícios que podem ser usados para driblar o problema das limitações trazidas pela distância.

Uma rede de propriedade que é usada por uma única companhia ou indivíduo é chamada de privada, e uma rede cujos proprietários são os portadores comuns, é chamada de pública.

A maioria das redes é privada, pois as companhias usualmente têm uma ou mais LANs que conectam os computadores de um único prédio ou local. A companhia compra o hardware e o software necessários para cada LAN, instala os cabos, e opera a rede.

Uma grande companhia também pode ter uma WAN privada, que é usada para conectar computadores em múltiplas localidades. A companhia compra o hardware para WAN, como packet switches e opera a rede.

Para formar uma WAN privada, a empresa deve alugar as conexões entre os múltiplos locais das companhias públicas, como as de telefonia. A WAN é ainda considerada privada, pois mesmo alugando as conexões nenhuma outra corporação terá acesso aos dados que passam nos fios.

Uma rede pública é análoga ao sistema de telefonia, qualquer um pode se inscrever no serviço e conectar um computador. O sistema de rede pública permite que um computador se comunique com qualquer outro que esteja inscrito na rede. Para fazer o serviço de rede pública atrativo, este deve ser disponível para vários computadores em várias localidades. Consequentemente, a maioria das redes públicas é WANs.

O termo público refere-se à disponibilidade do serviço, não à transferência de dados. Em particular, uma rede pública oferece uma comunicação privada, ou seja, apenas os computadores que estão participando recebem uma cópia dos dados. Além disso, algumas redes públicas permitem que um grupo de computadores se comunique, como numa conferência telefônica, e uma rede pública não transmite pacotes de broadcast. Assim, essas redes são chamadas de públicas no mesmo sentido que o sistema telefônico.

A maior vantagem de uma rede privada é que o proprietário tem controle total sobre ela. O proprietário ajusta as configurações e quais os computadores poderão se conectar. Além disso, o proprietário pode garantir que a rede está isolada dos computadores que não pertencem

à organização, e que um computador nunca conseguirá conectar-se a um computador de outra organização.

Uma grande rede privada pode ser extremamente cara de ser instalada e mantida. Além de comprar o hardware de rede, a companhia deverá contratar e treinar um pessoal para instalar, gerenciar e operar a rede. E o mais importante, as tecnologias de rede mudam rapidamente, manter-se com essas mudanças pode ser caro.

As maiores vantagens de uma rede pública são a flexibilidade e a habilidade para utilizar a rede sem altos custos de manutenção. A rede pública é flexível porque computadores arbitrários e em locais arbitrários podem se conectar a ela.

8.3 Virtual Private Networks

A tecnologia virtual private network (VPN) surgiu da junção das vantagens da rede pública e privada. A rede VPN restringe o tráfego, assim os pacotes podem ser transmitidos apenas entre os locais de uma mesma companhia, assim, utiliza o meio público como meio de transmissão.

Para construir uma VPN é necessário um hardware e um software especial para cada uma das localidades. O sistema é colocado entre a rede privada da companhia e a rede pública, sendo que cada um dos sistemas deve ser configurado com os endereços dos outros sistemas VPN da companhia. Assim, o software só irá trocar pacotes com os sistemas da mesma companhia, garantindo privacidade na comunicação. Além do que o sistema VPN codifica cada pacote antes da transmissão.

O sistema VPN examina o destinatário, codifica o pacote e envia o resultado através da rede pública para o sistema VPN em outra localidade. Assim, quando o pacote chega, o sistema verifica se ele é válido, decodifica o conteúdo e transmite o pacote para o destinatário.

8.4 Padrão de Serviço

Para ajudar a explicar as similaridades e diferenças entre os padrões de serviço, todas as interfaces de redes são classificadas em duas categorias de transmissão:

- Serviço de conexão orientada;
- Serviço sem conexão.

O serviço de conexão orientada opera de modo análogo ao sistema de telefonia. Antes de dois computadores poderem se comunicar, eles devem estabelecer uma conexão através da rede. Um computador pede à rede uma forma de conectar-se ao outro computador, e o segundo computador aceita a conexão. Após ambos os computadores concordarem em conectar, o sistema forma um caminho digital, conhecido por conexão. E informa aos computadores.

A comunicação através dessa conexão não precisa ser contínua. Ela continua até que um computador decida que não será mais necessária. Para fazer isso o computador informa à rede que a conexão não é mais necessária, análogo a desligar o telefone.

Redes sem conexão operam analogamente ao serviço de e-mails. Quando um computador for enviar dados, deverá colocar esses dados em um formato de frame apropriado, adicionando o endereço do computador que receberá os dados, depois passa o frame para a rede e ela o entrega ao computador de destino.

Cada um desses serviços tem vantagens e desvantagens. A vantagem do serviço de conexão orientada é a habilidade de informar a um computador imediatamente se a conexão foi cessada, já o serviço sem conexão não pode reportar um erro imediatamente, assim, o computador continua enviando pacotes mesmo depois que a falha acontece.

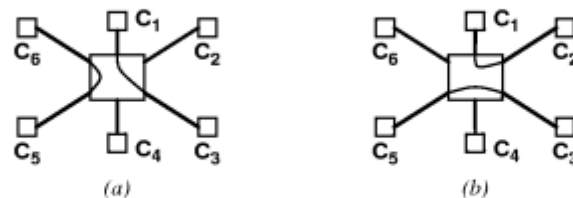
A vantagem do serviço sem conexão é permitir que um computador envie dados imediatamente, sem ter que esperar por uma conexão, que pode ser formada em milissegundos, um tempo considerado alto para as atuais velocidades de processamento.

8.5 Duração de Conexões e Persistência

Os termos *conexão switched* e *conexão permanente* distinguem dois tipos de conexão.

Conexões permanentes são usadas com hardware dedicado, ou seja, eram usadas em redes antigas. Nas redes modernas, conexões permanentes são concebidas configurando a rede para formar um caminho de comunicação eletrônico. Por exemplo, um conjunto de switches pode ser configurado se um par de computadores requisitou uma conexão. Uma vez que uma conexão permanente é configurada, ela é armazenada em uma memória não volátil, permitindo aos switches restabelecerem a conexão após uma queda de energia.

Assim cada computador mantém uma conexão com a rede durante todo o tempo, e o switch conecta um computador ao outro. A figura abaixo ilustra a idéia.



A maioria das conexões feitas por um switch tem curta duração. Se um computador trava enquanto está usando uma conexão com switch, tal conexão deverá ser restabelecida após o computador ser reiniciado.

O termo switch é aplicado hoje a equipamentos eletrônicos usados em redes de dados, e essas redes são chamadas de *switched data networks*.

Muitas redes que provem de conexões switched usam uma idéia derivada do sistema de telefonia: múltiplas ligações ao mesmo tempo. A rede permite ao computador estabelecer mais de uma conexão ao mesmo tempo e especificar qual conexão é usada em determinado instante.

A vantagem de conexões permanentes é a persistência e a disponibilidade garantida. Assim, requer pouca manutenção, além do que uma conexão permanente está sempre pronta para receber dados.

As vantagens da conexão switched são a flexibilidade e a generalidade. Como são formadas eletronicamente, não é necessário instalar ou mudar fisicamente os cabos. Além do que uma rede switched necessita apenas de uma conexão física por computador.

O quadro abaixo sumariza as características de várias tecnologias de rede.

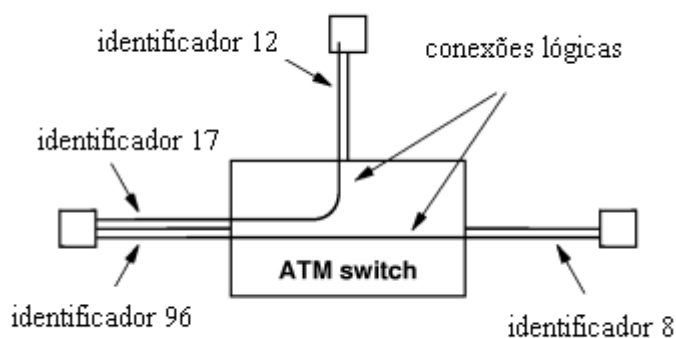
Tecnologia	Orientada para conexão	Sem conexão	Usado pela LAN	Usado pela WAN
Ethernet		•	•	
Token Ring		•	•	
FDDI		•	•	
Frame Relay	•			•
SMDS		•		•
ATM	•		•	•
Local Talk		•	•	

8.6 Endereços e Identificadores de Conexão

Em uma rede sem conexão, cada pacote deve conter o endereço do destinatário. Já uma rede de conexão orientada usa abreviações. Quando um computador requer uma nova conexão, ele envia uma mensagem para a rede, especificando o endereço da destinação remota. . Depois de estabelecida a conexão, a rede responde com uma mensagem que verifica a conexão e especifica um *identificador da conexão* para ser usado. Usualmente, o identificador da conexão é um inteiro pequeno, bem menor que o endereço total do destinatário.

Usar um identificador de conexão reduz o *overhead* (código extra) tornando o cabeçalho de dados bem menor.

Um identificador de conexão é determinado independentemente a cada fim de conexão ATM; eles não são valores globais. Assim um fim de conexão deve determinar com identificador 17, por exemplo, enquanto outro fim da mesma conexão pode determinar o valor 12. O switch é quem cuida das conexões é que determina os identificadores e os traduz na rede. A figura abaixo ilustra a idéia:



Na prática, em uma rede ATM, os identificadores são divididos em duas partes, que são o *Virtual Path Identifier* (VPI) e o *virtual circuit identifier* (VCI). O VPI é usado para o caminho do destinatário do switch, e o VCI é usado para a entrega ao computador ligado aquele switch.

8.7 Características do Desempenho de Redes

As redes são classificadas como de baixa ou alta velocidade. Porém essas definições são inadequadas porque as tecnologias de rede mudam rapidamente, e uma rede classificada como de alta velocidade pode se tornar de média ou baixa velocidade em um período de 4 anos. Assim foi especificada a velocidade da rede precisamente, usando termos quantitativos. Essa seção define duas quantidades fundamentais para uma rede, e explica como são relacionadas à capacidade da rede.

8.7.1 Delay

O delay de uma rede especifica quanto tempo irá demorar um bit de dados para ir de um computador a outro, o delay é medido em segundos ou frações de segundo. O delay pode ser diferente dependendo da localização específica de um par de computadores que se comunica. Os usuários só se preocupam com o delay total de uma rede. Como são necessárias medições, é reportados o delay máximo e o delay médio, e os dividem em várias partes.

O delay criado pela demora de tempo de um sinal viajar em um fio ou fibra óptica é chamado de delay de propagação. Esse delay é da casa de milissegundos em LANs, e pode ser alto, quando, por exemplo, uma rede utiliza um satélite que órbita a terra e precisa levar dados de um continente ao outro.

O delay, introduzido pelos equipamentos eletrônicos, é conhecido como switching delay. Um equipamento eletrônico deve esperar até que todos os bits de um pacote tenham chegado, e leva um pequeno tempo para escolher para onde enviar o pacote. Equipamentos altamente especializados tornaram este delay insignificante.

Como a maioria das LANs utiliza um meio comum, o computador deve esperar para até que o meio esteja livre. Esse delay é conhecido como delay de acesso.

Uma forma final de delay acontece em um pacote switched na WAN. Relembre que cada pacote é colocado na fila, fazendo parte do processo de store-and-forward. Assim o pacote na fila deve esperar até que o pacote que chegou primeiro seja enviado. Esse delay é conhecido como delay de enfileiramento.

8.7.2 Ritmo de Transferência

A segunda propriedade fundamental de redes que pode ser medida é a taxa de transferência (throughput). O ritmo de transferência é uma taxa sobre os dados que são enviados na rede, e é especificada em bits por segundo (bps). Muitas redes têm um ritmo de transferência de alguns milhões de bits por segundo (Mbps).

Como o ritmo de transferências pode ser medido de várias maneiras, deve ser bem cuidadoso para especificar exatamente qual método foi utilizado. Por exemplo, no capítulo 4, nós aprendemos que a capacidade do ritmo de transferência dos hardwares é chamada de *largura de banda*. De fato, o termo *largura de banda* é às vezes sinônimo de ritmo de transferência. Em particular, na maioria das tecnologias cada frame contém um cabeçalho, que significa que o ritmo de transferências efetivo é menor que a largura de banda oferecida pelo hardware. Nunca a largura de banda do hardware é usada como uma aproximação do ritmo de transferência da rede, pois a largura de banda é um limite bem maior no ritmo de transferências – é impossível ao usuário enviar dados mais rápido que o hardware pode transferir os bits.

Os profissionais de rede normalmente utilizam o termo velocidade como sinônimo do ritmo de transferência. Porém pode haver confusão com o delay. De fato, o ritmo de transferências é a medida da capacidade, e não velocidade.

O delay é medido em segundos, e especifica quanto tempo um bit permanece em trânsito na rede. O ritmo de transferência, que é medido em bits por segundo, especifica quantos bits podem entrar na rede por unidade de tempo. O ritmo de transferências mede a capacidade da rede.

8.7.3 A relação entre Delay e Ritmo de Transferência

Na teoria, delay e ritmo de transferências de uma rede são independentes. Na prática, eles podem ser relacionados. Se um packet switch está na fila esperando enquanto um novo pacote chega, o novo pacote será colocado na calda da fila e irá precisar esperar enquanto o switch envia o pacote anterior. Análogo ao trânsito pesado nas rodovias, tráfego excessivo na rede é chamado de congestionamento.

Cientistas da computação estudaram a relação entre delay e congestionamento, e chegaram a vários casos, que o delay esperado pode ser estimado pela porcentagem da capacidade da rede que é utilizada. Se D_0 denota o delay quando a rede está vazia, e U é um valor entre 1 e 0 que denota a utilização no momento, o delay efetivo D é dado por esta simples fórmula:

$$D = D_0 / (1-U)$$

Assim pode-se notar pela fórmula que o delay e o ritmo de transferências não são completamente independentes.

8.7.4 Produto do Delay - Ritmo de Transferência

Uma vez que o delay e o ritmo de transferências de uma rede são conhecidos, é possível computar uma quantidade interessante, o produto do delay e ritmo de transferência.

O produto dessas duas quantidades mede o volume de dados que estão presentes na rede. Este produto é importante para qualquer rede, especialmente para redes com delay alto ou grande ritmo de transferência, porque significa ao computador que está enviando dados na rede pode gerar um grande volume, antes mesmo de o destinatário receber o primeiro bit.

Capítulo 9 – Protocolos e Divisão em Camadas

9.1 Introdução

Esse capítulo examinará a estrutura de um software utilizado no sistema de redes. O capítulo explica porque o hardware não consegue resolver sozinho todos os problemas de comunicação, e mostra o porquê da necessidade dos protocolos. Esse capítulo também irá descrever o conceito das camadas, que provem uma base conceitual para a compreensão do conjunto completo de protocolos que trabalham para prover um rápido e eficaz sistema de comunicação.

9.2 A Necessidade dos Protocolos

O software irá lidar nos detalhes e problemas de comunicação no nível mais baixo automaticamente, fazendo possíveis os aplicativos se comunicarem mais facilmente. Assim a maioria dos programas conta com o software de rede para se comunicar: eles não interagem diretamente com o hardware de rede.

Todas as partes envolvidas na comunicação devem concordar em um conjunto de regras para ser usada na troca de mensagens. Esse tipo de acordo é chamado de *protocolo*. O termo é aplicado bem à comunicação entre computadores: um conjunto de regras que especifica o formato das mensagens e as ações apropriadas requeridas para cada mensagem é conhecido como protocolo de rede. O software que implementa este protocolo é conhecido como Software de Protocolo. Esses protocolos podem ser muitas vezes simples ou complexos, dependendo da complexidade das mensagens que serão enviadas.

9.3 Coleções de Protocolos

Ao invés de haver um único e gigante protocolo que especifique completamente os detalhes para todas as formas de comunicação, os designers decidiram dividi-lo em peças, e projetar separadamente cada protocolo para cada peça. Fazendo isso, a análise e teste dos protocolos se tornaram mais fácil. Assim dividindo o software de comunicação em vários protocolos aumenta a flexibilidade porque permite subgrupos de protocolos serem utilizados, de acordo com a necessidade.

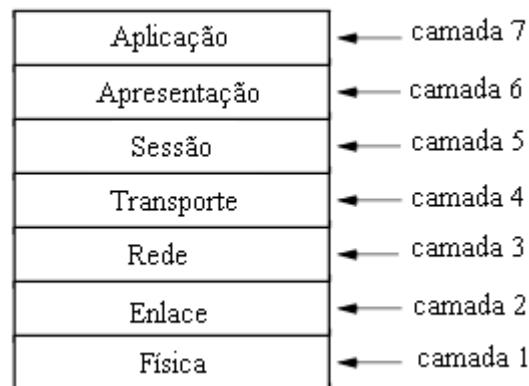
A divisão entre protocolos separados deve ser escolhida cuidadosamente para que o sistema de comunicação resultante seja eficiente e efetivo. Para tornar essa implementação possível, os protocolos devem ser projetados de maneira que possam compartilhar estruturas de dados e informações. Finalmente, a combinação dos protocolos devesse lidar com todas as possíveis falhas e condições excepcionais que ocorram no hardware.

A garantia dos protocolos trabalharem bem está contida no desenvolvimento, que ao invés de ser feito isoladamente, são projetados em completo, em conjuntos, chamados *suites* ou *famílias*. Cada protocolo em um switch resolve uma parte dos problemas de comunicação; juntos, eles resolvem o problema de comunicação por inteiro. Além do que, o switch inteiro é projetado para fazer interações dentro do próprio protocolo.

9.4 Um Plano para o Projeto de Protocolos

Diversas ferramentas foram desenvolvidas para ajudar os projetistas de protocolos entenderem as subpartes dos problemas da comunicação e planejar uma família toda de protocolos. Uma das ferramentas mais importantes é chamada de *layering model* (modelagem

por camadas). Na essência, o modelo de camadas descreve uma maneira de dividir o problema da comunicação em pequenos pedaços, chamados de *layer* (camadas). A figura abaixo ilustra a idéia (ISO layering model).



9.5 As sete Camadas

O modelo ISO foi bastante popular porque provia uma explicação simples das relações entre um hardware complexo e os componentes de um protocolo na rede. No modelo ISO, o nível mais baixo corresponde ao hardware, e enquanto sobe o nível, as camadas vão correspondendo sucessivamente ao [firmware](#) ou software que usam o hardware. Essa seção explica cada camada.

- Camada 1: Física

Essa camada corresponde à base do hardware de rede. Por exemplo, a especificação do RS-232 pertence à camada 1, e da explicação detalhada do hardware de LAN.

- Camada 2: Enlace

Essa camada do protocolo especifica como organizar os dados em frames, e como transmiti-los através da rede.

- Camada 3: Rede

Essa camada do protocolo especifica quais endereços serão marcados e como os pacotes são enviados de um até o outro computador na rede.

- Camada 4: Transporte

A camada 4, que especifica como lidar com os detalhes de uma transferência confiável, está entre um dos protocolos mais complexos.

- Camada 5: Sessão

A camada 5 especifica como estabelecer uma seção de comunicação com um sistema remoto. Especificações de segurança, como autenticação utilizando senha secreta, pertencem a esta camada.

- Camada 6: Apresentação

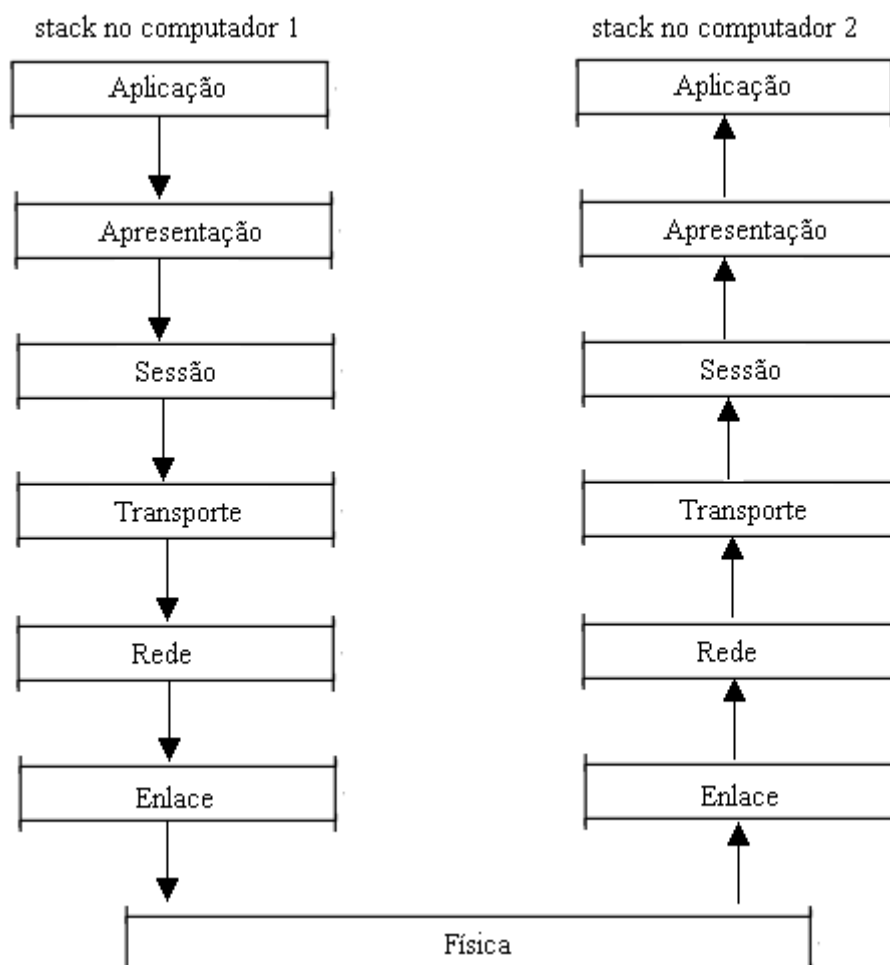
A camada 6 especifica como representar os dados. Esse protocolo é necessário porque diferentes marcas de computadores usam representações internas diferentes para inteiros e caracteres.

- Camada 7: Aplicação

Cada camada 7 de um protocolo especifica como um aplicativo usa a rede. Por exemplo, a especificação de um aplicativo que transfere arquivos de um computador a outro. Esse protocolo especifica os detalhes de como um aplicativo em um computador requer um arquivo em outro computador, e como este responde.

9.6 Pilhas: Software em Camadas

O software de protocolos em cada computador é dividido em módulos, com cada módulo correspondendo a uma camada. Mais importante, a divisão em camadas determina a interação entre os módulos: na teoria, quando um software de protocolos envia e recebe dados, cada módulo se comunica apenas com o módulo da camada mais alta e o módulo com a camada mais baixa. Assim os dados que estão saindo passam descendo dentro de cada camada, e os dados recebidos passam subindo em cada camada. A figura abaixo ilustra o conceito.



Os fornecedores usam o termo pilhas (stack) para se referirem ao software de protocolos, pois o modelo de camadas sobre o qual o software é feito é sempre visualizado como um conjunto de retângulos como as figuras anteriores mostram.

Diversas pilhas são disponíveis comercialmente. A tabela abaixo mostra a lista das seis mais populares:

Vendor	Stack
Novell Corporation	Netware
Banyan System Corporation	VINES
Apple Computer Corporation	AppleTalk
Digital Equipment Corporation	DECNET
IBM	SNA
(many vendors)	TCP/IP

Como cada pilha foi projetada independentemente, os protocolos de uma pilha não podem se interagir com os protocolos de outro.

Para resolver esse problema, se necessário um computador pode utilizar mais de uma pilha ao mesmo tempo. Duas pilhas podem ser executados no mesmo computador e podem transmitir em uma única rede física sem interferência, [porque o campo tipo em cada frame identifica qual pilha deveria lidar com a mensagem.](#)

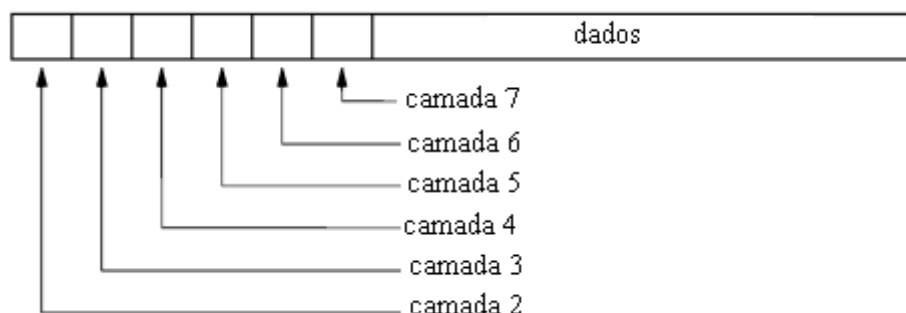
9.7 Como Softwares de Camadas Funcionam

O software em uma determinada camada no computador emissor adiciona a informação para os dados que estão saindo, e o software do computador receptor, na mesma camada, utiliza a informação adicional para processar os dados.

Assim, cada camada que adiciona uma informação, e essa mesma camada no receptor irá extrair as informações e dados.

9.8 Cabeçalhos Abrigados Multiplamente

Usualmente, cada camada coloca informações adicionais no cabeçalho, antes de enviar os dados para uma camada mais baixa. Assim, um frame viajando através da rede contém uma série de cabeçalhos abrigados, como a figura abaixo mostra:



Como a figura mostra o cabeçalho correspondente à camada mais baixa do protocolo vem primeiro. No modo ISO, o cabeçalho para o data link vem primeiro. Porém a camada 1 especifica o sinal elétrico ou óptico que será usado na transmissão de um frame, a camada 1 não aciona no cabeçalho como as outras camadas o fazem.

A figura acima ilustra um conceito generalizado, mas não mostra todas as possibilidades possíveis. Em particular, alguns softwares de protocolo fazem mais do que adicionar no começo um cabeçalho aos dados que estão saindo. Similarmente, alguns protocolos especificam que todas as informações adicionais deverão ser adicionadas ao frame, ao invés de serem adicionadas no começo.

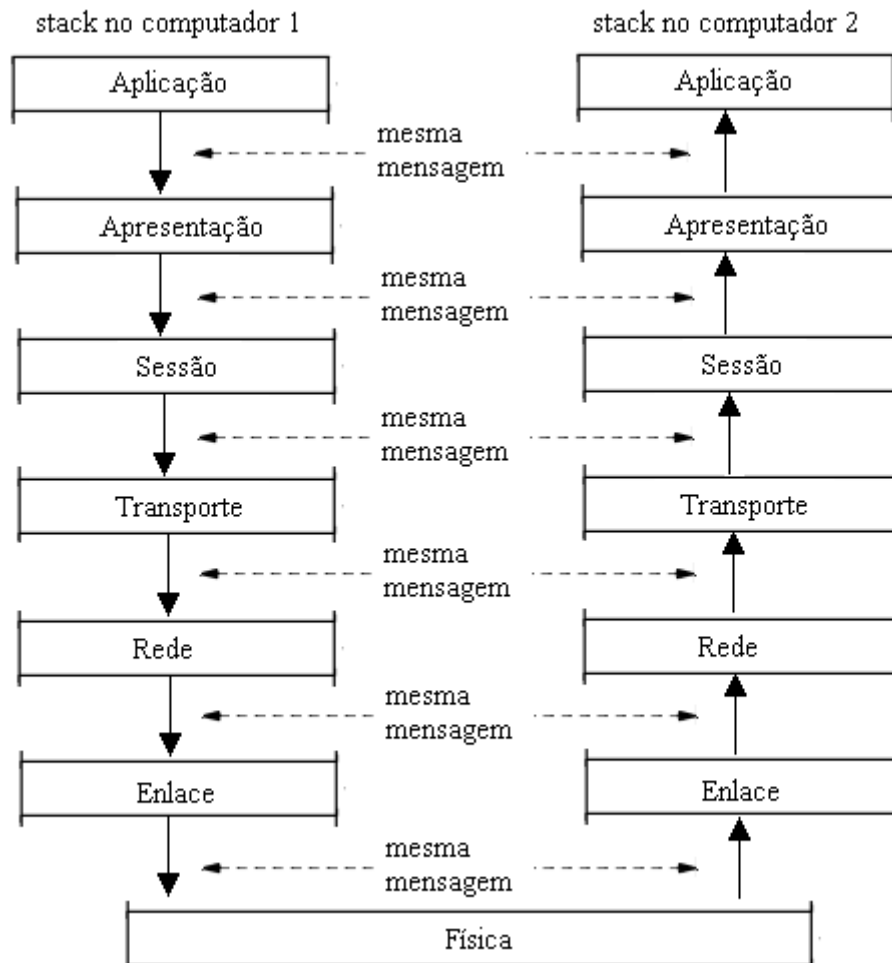
9.9 A Base Científica para a Divisão em Camadas

O significado da divisão em camadas chega diretamente ao princípio científico conhecido como *princípio das camadas*:

“O software de camadas N no computador destinatário deve receber a mensagem exata envia pela camada N no software do computador emissor”

Em outras palavras, qualquer transformação feita pelo protocolo antes de enviar um frame deve ser completamente revertida quando o frame é recebido. Se uma camada em particular no computador emissor adiciona um cabeçalho no início do frame, a camada correspondente no receptor deve remover o cabeçalho. Se uma camada codifica o frame antes de enviar, a camada correspondente deve decodificar o frame.

A idéia da divisão em camadas é poderosa, pois simplifica o projeto e o teste dos protocolos. A divisão em camadas previne que o software de protocolos, em uma camada introduza mudanças que podem ser visíveis em outras camadas. Como resultado, o software tanto emissor quanto o receptor devem ser projetados, implementados e testados independentemente das outras camadas. A figura abaixo ilustra o conceito:



9.10 Técnicas Usadas nos Protocolos

Alguns protocolos fazem mais do que detectar erros, eles fazem um esforço para reparar ou driblar os problemas. Os protocolos de transporte utilizam uma variedade de ferramentas para lidar com alguns dos problemas mais complicados na comunicação. Essa seção faz uma revisão de um conjunto de problemas que podem ocorrer e discute as técnicas que os protocolos usam para resolvê-los. Com as técnicas de data link discutidas anteriormente, os detalhes exatos dependem do projeto completo do protocolo switch.

9.10.1 Sequência de Entrega Fora da Ordem

Um sistema de rede sem conexão, que pode mudar as rotas pode entregar os pacotes fora da ordem. Para entender como, considere uma sequência de pacotes sendo enviada, e lembre que a rede tenta usar o menor caminho a todo o momento. Se um caminho mais curto se torna disponível imediatamente após o pacote de ordem i ser enviado, a rede deve enviar o pacote $i+1$ através do caminho mais curto, podendo este chegar mais cedo que o pacote i .

Para lidar com esse erro, o protocolo de transporte usa uma sequência. O emissor coloca um número da sequência em cada pacote. O receptor armazena o número da sequência do último pacote recebido e a lista adicional dos pacotes que chegaram fora de ordem. Quando um pacote chega, o receptor examina o número da sequência para determinar como deve lidar com esse pacote. Se o pacote é o próximo (chegou em ordem), o software de protocolos entrega o pacote a próxima camada e checa a lista para ver se há mais pacotes que podem ser entregues. Se o pacote chega fora de ordem, o software o adiciona a lista.

9.10.2 Seqüência para Eliminar Pacotes Duplicados

A seqüência resolve os problemas de duplicação. O software do receptor checa a duplicação quando examina o número do pacote que chega. Se a seqüência já tiver sido entregue ou o número bate com os pacotes que estão na lista, o software descarta a nova cópia.

9.10.3 Retransmissão de Pacotes Perdidos

A perda de pacotes é um problema fundamental nas redes de computadores, pois erros de transmissão podem corromper bits, tornando o frame inválido. Quando um receptor detecta esses problemas, o frame é descartado.

Para garantir uma transferência confiável, os protocolos utilizam *positive acknowledgement with retransmission*. Quando um frame chega intacto, o software de protocolos do receptor envia uma pequena mensagem que reporta uma recepção de sucesso. Essa mensagem é conhecida como *acknowledgement* (ACK). O emissor toma a responsabilidade de assegurar que cada pacote é transferido com sucesso. Quando ele envia um pacote, o software inicia um timer. Se a ACK chegar antes do timer expirar, o software cancela o timer. Se o timer expirar antes da ACK chegar, o software envia outra cópia do pacote e começa o timer novamente. A ação de enviar uma segunda cópia é conhecida como retransmissão.

A retransmissão não terá sucesso se houver uma falha de hardware como permanecer desconectado ou se o computador receptor tenha entrado em colapso. Então, protocolos que utilizam retransmissão normalmente têm um limite máximo de retransmissão. Quando este limite é atingido, o protocolo para de retransmitir e declara que a comunicação é impossível.

Note que a retransmissão pode introduzir pacotes duplicados. Porque um emissor não pode distinguir entre um pacote que foi perdido e um pacote que tenha experimentado um longo delay. Assim, protocolos, que usam retransmissão para prover confiança, deverão lidar também com o problema da duplicação.

9.10.4 Evitando Replay Causado pelo Delay Excessivo

Uma fonte de delay em um sistema de packet switching é causada pela aproximação de store-and-forward. Quando um pacote chega em um packet switch, o pacote é colocado em uma fila. Se o pacote chegar mais rapidamente que o switch possa enviá-los, a fila irá aumentar e o delay se torna excessivo. Delays extraordinários podem levar para erros de replay. Replay significa que um pacote antigo e atrasado pode afetar a comunicação futura.

Infelizmente, ao menos que o protocolo seja projetado cuidadosamente, um pacote de uma conversação antiga pode ser aceito em uma nova conversação e o pacote correto pode ser descartado.

O replay também pode ocorrer com os pacotes de controle. Por exemplo, protocolos sempre enviam pacotes especiais de controle para terminar a comunicação. Se uma cópia de um término de conversação chega da conversa anterior, pode fazer com que o software de protocolos termine a conversação imediatamente e prematuramente.

Para prevenir o replay, os protocolos fazem cada seção com um ID único, e requer um único ID para ser presente em cada pacote. O software de protocolos descarta qualquer pacote que chegue com o ID incorreto.

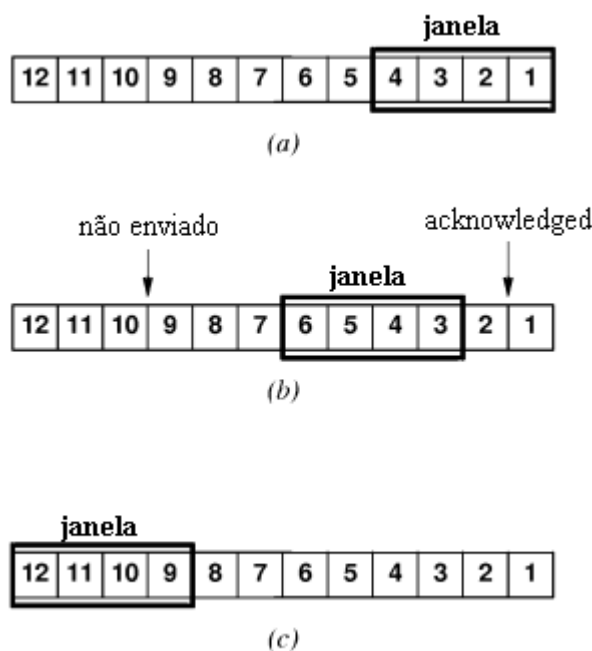
9.10.5 Controle de Fluxo para Evitar o Excesso de Dados

O excesso de dados acontece quando um computador envia dados mais rápido que o outro computador pode absorvê-los. Conseqüentemente, dados são perdidos.

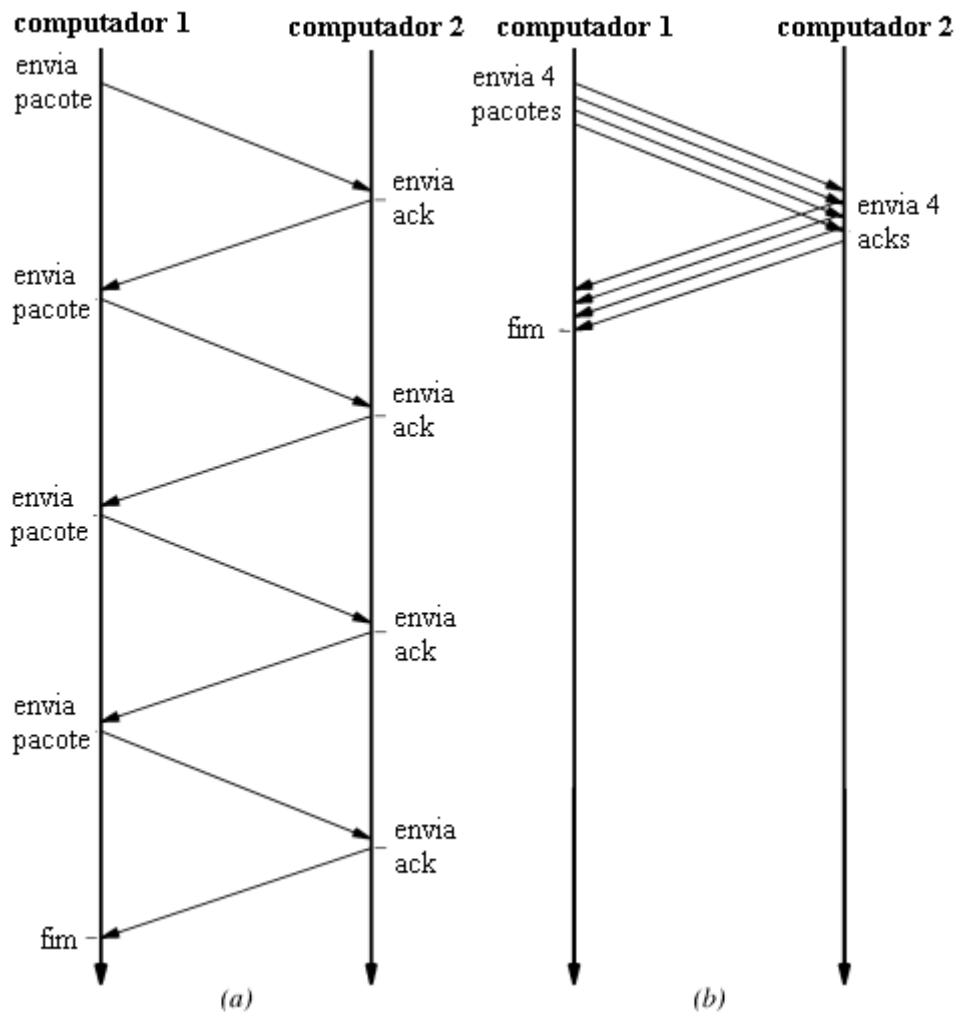
A técnica usada é conhecida como mecanismos de controle de fluxo. A forma simples do sistema *stop-and-forward* no qual o emissor espera até que cada pacote seja transmitido. Quando o receptor está pronto para o próximo pacote, ele envia uma mensagem de controle, usualmente conhecida como ACK.

Embora os protocolos de stop-and-go previnam um excesso de dados, eles podem causar uma ineficiência extrema no uso da largura de banda da rede.

Para obter altas taxas de transferências, protocolos utilizam uma técnica de controle de fluxo conhecida como *sliding window*. O emissor e receptor são programados para utilizarem uma “janela” de tamanho fixo, que é o máximo de dados que podem ser enviados antes que a ACK chegue. Por exemplo, o emissor e receptor devem concordar em um tamanho fixo de janela para quatro pacotes. O emissor começa com os dados a serem enviados, extrai os dados para encher a primeira janela, e transmite as cópias. Se for necessário segurança, o emissor guarda uma cópia no caso da necessidade da retransmissão. O receptor deve ter um espaço de buffer pronto para receber toda a janela. Quando um pacote chega na sequência, o receptor passa o pacote para o aplicativo receptor e transmite a ACK para o emissor. Quando um ACK chega, o emissor descarta a cópia do pacote e transmite o próximo pacote. A figura abaixo ilustra o conceito:



Uma janela deslizando pode incrementar a taxa de transferência drasticamente. Para entender como, considere a sequência de transmissões com o esquema stop-and-go e o esquema janela deslizando. A figura abaixo faz essa comparação:



Como a figura acima mostra a seqüência de transmissões do protocolo stop-and-go. Após enviar um pacote, o protocolo espera uma ACK antes de enviar outro pacote. Se o delay requerido para enviar um único pacote é N , o delay total para enviar quatro pacotes será $8N$.

Na seqüência de transmissão usando uma janela deslizante, o protocolo envia todos os pacotes numa janela, antes de esperar. Para ser realista, a figura mostra um pequeno delay entre o envio sucessivo dos pacotes. Porém o delay deve ser menor do que o mostrado, a transmissão nunca é instantânea - um tempo curto (usualmente poucos milissegundos) é requerido para o hardware completar a transmissão de um pacote, interromper a CPU, e começar a transmitir o próximo pacote. Assim, o tempo total para enviar quatro pacotes é

$2N + E$, onde E denota o pequeno delay.

Para entender o significado da janela deslizante, imagine uma comunicação extensa que envolva muitos pacotes. Nesses casos, o tempo total requerido para transmissão é tão grande que E pode ser ignorado. Para ver os benefícios da janela deslizante, considere uma rede com uma grande taxa de transferência e grande delay. Para essas redes, um protocolo de janela deslizante pode aumentar o desempenho por um fator muito maior que 1. De fato, o aumento do desempenho é:

$$T_w = T_g * W$$

Onde T_w é o ritmo de transferências que pode ser alcançado com um protocolo de janela deslizante, T_g é o ritmo de transferências que pode ser alcançado em um protocolo stop-and-go, e W é o tamanho da janela deslizante. A equação explica o porquê do protocolo de janela deslizante da figura acima ser aproximadamente 4 vezes menor do que o ritmo de transferências

do protocolo stop-and-go da mesma figura. É claro que o ritmo de transferências não pode ser feito arbitrariamente grande ao aumentar o tamanho da janela. A largura de banda de uma rede impõe um limite - bits não podem ser enviados mais rápido que o hardware possa transportá-los. Assim, a equação pode ser escrita como:

$$T_w = \min (B, T_g * W)$$

9.10.6 Mecanismos para Evitar Congestionamento da Rede

Congestionamento é um problema fundamental em um sistema de pacotes switching. Para entender o porquê, considere uma rede representada pela figura abaixo:



Se um computador conectado ao nó 2 começa a enviar pacotes para um destinatário ligado ao nó 6, os pacotes devem atravessar o link do meio. Se algum computador no nó 1 estiver enviando pacotes para um computador no nó 5, os dados chegarão ao nó 3 duas vezes mais rápido que possa ser enviado para o nó 4. O pacote switch que corresponde ao nó 3 coloca os pacotes que ele recebe do nó 1 e 2 em uma fila até que eles possam ser enviados, porque mais pacotes chegam e podem ser enviados, a fila cresce e o delay aumenta. A situação é conhecida como congestionamento.

Se o congestionamento persistir, o pacote switch irá ficar sem memória e começará a descartar pacotes. Mesmo que a retransmissão possa ser usada para recuperar os pacotes, ela leva muito tempo. Além do que, se a situação persistir, a rede toda se torna instável. Essa condição é conhecida como Colapso de Congestionamento. Os protocolos tentam evitar o colapso de congestionamento monitorando a rede e reagindo rapidamente uma vez que um congestionamento começa. Há duas aproximações:

- Arrume pacotes switches que informem o fabricante quando um congestionamento ocorre;
- Use a perda dos pacotes como uma estimativa de congestionamento.

Capítulo 10 – Internet: Conceitos, Arquitetura e Protocolos

10.1 Introdução

O capítulo anterior descreveu uma rede básica, incluindo os componentes de hardware usados em uma LAN e em uma WAN e tratou de conceitos como endereçamento e roteamento. Este capítulo começa analisando outro conceito fundamental em comunicação entre computadores: uma tecnologia entre redes que pode ser usada para conectar várias redes formando um único sistema de comunicação. O capítulo discute o motivo pelo qual o estudo da interação de redes é importante, fala do hardware usado e descreve a arquitetura nas quais os componentes são conectados.

10.2 A Motivação para Estudar Internet

Cada tecnologia de rede é desenvolvida para se ajustar a um grupo específico de constates. Por exemplo, uma LAN é desenvolvida para propiciar comunicação de alta velocidade a distâncias curtas, já uma WAN é desenvolvida para promover a comunicação entre uma área vasta, sem grandes preocupações com a velocidade. É importante ter em mente que não existe uma rede capaz de satisfazer todas as necessidades, para cada caso há um tipo de rede que melhor adequasse. Se uma empresa escolhe uma rede Ethernet para uma de suas sessões, ela poderá usar uma rede ATM em outra sessão e conectar as duas. Isso é uma Internet, ou seja, conexão entre redes.

10.3 O Conceito de Universal Service

O principal problema em termos múltiplas redes é bem óbvio: um computador de um determinado tipo de rede só pode se comunicar com outro computador se ele também estiver usando o mesmo tipo e rede. O problema começou a se evidenciar na década de 70, quando grandes empresas começaram a adquirir várias redes. Cada rede na organização formava uma “ilha”, pois não se comunicava com a outra. O problema teve fim com a evolução das redes e o surgimento do Universal Service que permite a comunicação entre quaisquer dois computadores da organização mesmo que eles estejam em redes de tipos diferentes. Por aí já se nota sua grande importância.

10.4 Serviço Universal em um Mundo Heterogêneo

Como já foi visto no capítulo 10 é impossível formar uma rede grande simplesmente conectando os fios das duas redes. Infelizmente as técnicas de expansão (com bridges) não podem ser usadas em redes heterogêneas porque tecnologias diferentes usam formatos de pacote diferentes, além de usarem um esquema de endereçamento diferente.

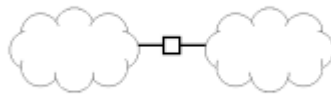
10.5 Internet (entre Redes)

A respeito das incompatibilidades entre redes diferentes, pesquisadores têm desenvolvido um esquema que usa o serviço universal para conectar redes diferentes. Esse esquema utiliza-se de um software e de um hardware para fazer isso possível. Com duas ou

mais redes ligadas têm-se o que se chama de Internet. Numa Internet não há um limite de tamanho ou no número de computadores.

10.6 Redes Conectadas com Roteadores

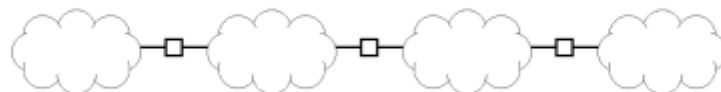
O componente básico usado para conectar duas redes é o roteador. Como já foi visto, o roteador funciona como um bridge conectando apenas um computador por segmento, quando se trata de conectar duas redes não é diferente: o roteador trata cada rede como sendo um único computador. A figura tenta ilustrar a idéia de conexão de duas redes com um roteador.



O mais importante é saber que um roteador pode interconectar redes que usam tecnologias diferentes, incluindo meios de transmissão diferentes, esquema de endereçamento, ou até mesmo formatos de frames diferentes.

10.7 Arquitetura de Internet

O uso do roteador torna possível que uma organização escolha um tipo de rede apropriado para cada caso e depois reúna tudo em uma única rede. A figura abaixo mostra quatro redes arbitrárias unidas por três roteadores formando uma Internet.



Apesar de a figura mostrar cada roteador com exatamente duas conexões, comercialmente um roteador aceita que mais de duas redes sejam conectadas a ele. Deste modo, na figura acima podemos substituir os três roteadores por apenas um obtendo-se o mesmo efeito. Mesmo isso sendo verdade, as empresas que tem uma Internet raramente fazem isso e os motivos principais são os seguintes:

- O hardware de um roteador é insuficiente para lidar com um tráfego de pacotes muito alto;
- No caso de uma falha em um dos roteadores os outros podem aprender suas rotas e manter a rede funcionando. Se o roteador for único isso não pode acontecer.
- Para sintetizar o assunto: uma Internet consiste em um grupo de redes interconectadas por roteadores.

10.8 Realizando o Universal Service

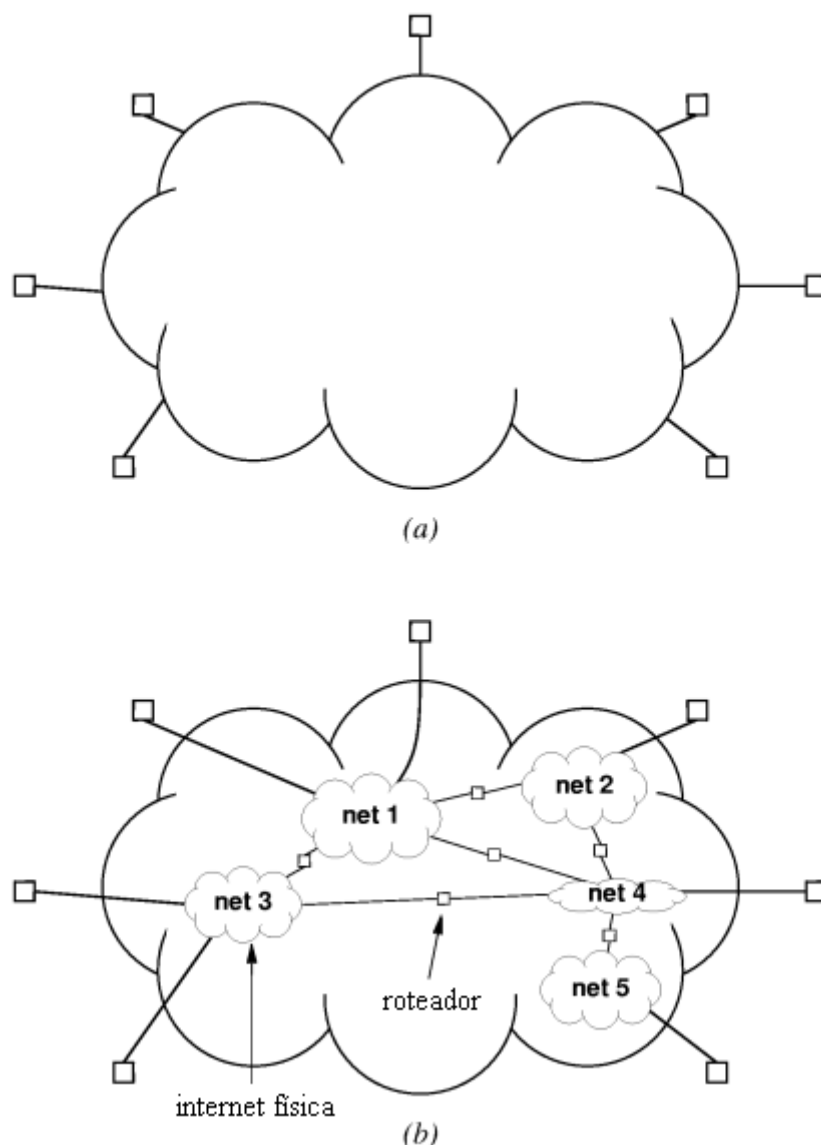
A meta da Internet é usar o universal service em redes heterogêneas. Para promover o universal service entre todos os computadores de uma Internet, roteadores devem ser capazes de enviar uma informação ou dado de uma fonte localizada em uma rede para um destino em outra rede, do outro lado do roteador. A questão não é simples porque o formato de frame e de endereçamento é completamente diferente nos diversos tipos de rede existentes. Como resultado é preciso de um protocolo, ou seja, um software deve converter os dados da rede em um formato que é padrão para todos os tipos de rede. Feito isso o universal service torna-se possível.

Capítulos passados já descreveram os softwares de protocolo de Internet em detalhes, portanto já se sabe (ou deveria saber) como os protocolos de Internet ultrapassam as diferenças entre os formatos de frame e tipos de endereçamento para fazer a comunicação possível entre redes de diferentes tecnologias. [Antes de entender como os protocolos de Internet funcionam, é importante entender o efeito que um sistema de Internet surte nos computadores ligados a ela.](#)

10.9 A Rede Virtual

Do ponto de vista do computador ligado à Internet tudo parece ser como uma rede normal. O sistema que faz uso do serviço universal tem as seguintes características: cada computador é identificado por um endereço, e qualquer computador pode enviar dados ou informações para qualquer outro da mesma Internet. Isso é possível porque os softwares de protocolo de Internet manipulam as informações e ocultam os detalhes da rede física, do esquema de endereçamento e as informações de roteamento, ou seja, o usuário e o computador de destino não são informados sobre o tipo de rede e o tipo de endereçamento do computador que deu origem à mensagem, no entanto, como todos usam o protocolo, a comunicação é possível.

Diz-se que a Internet é uma rede virtual porque o sistema de comunicação é abstrato. Os hardwares e softwares unem um emaranhado de redes e trazem a impressão de que tudo aquilo é uma única rede uniforme. A Figura abaixo mostra uma Internet como rede uniforme e depois mostra o que acontece de verdade.



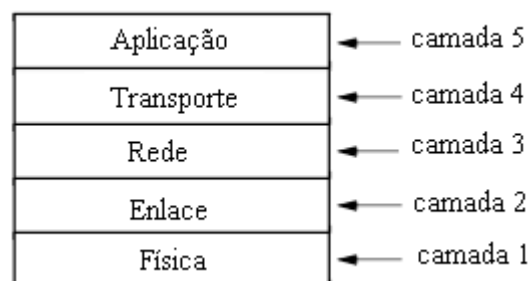
10.10 Protocolos para Internet

Apesar de muitos protocolos se adaptarem ao uso na Internet, um deles tem sido largamente usado. Trata-se do *TCP/IP*. O *TCP/IP* foi o primeiro protocolo a ser desenvolvido para uso em Internet, ele foi desenvolvido pelos mesmos pesquisadores que desenvolveram a arquitetura de Internet vista anteriormente neste capítulo, talvez por isso seja tão bom.

10.11 Camadas e Protocolos TCP/IP

As sete camadas modelos descritas no capítulo 14 foram criadas antes da Internet ser inventada, conseqüentemente o modelo não contém a camada para protocolo de Internet. Como resultado, pesquisadores tiveram de desenvolver um novo modelo de camadas para lidar com o *TCP/IP*. Esta sessão descreve brevemente as camadas criadas para o *TCP/IP*. Um estudo mais profundo do protocolo e das camadas será feito em capítulos posteriores.

O modelo de camadas *TCP/IP* contém apenas cinco camadas, como mostra a figura abaixo:



Quatro das camadas no TCP/IP correspondem a uma ou mais camadas do modelo ISO. No entanto, o modelo ISO não tem a camada de Internet. Resumidamente, as camadas do TCP/IP são:

- **Camada 1: Physical** - corresponde ao hardware básico da rede;
- **Camada 2: Network** - Interface: especifica como se organizam os dados dentro dos frames e como o computador transmite os frames através da rede;
- **Camada 3: Internet** - especifica o formato dos pacotes enviados através da Internet, bem como o mecanismo usado para transmitir os pacotes de um computador até seu destino final;
- **Camada 4: Transporte** - especifica como assegurar uma transferência confiável;
- **Camada 5: Aplicação** - especifica como uma aplicação ou aplicativo usa a Internet.

Capítulo 11 – IP: Internet Protocol Addresses

11.1 Introdução

Esse capítulo introduz o software de protocolo que faz com que uma Internet pareça ser uma rede simples. Serão discutidos o esquema de endereçamento usado pelo Internet protocol e a divisão dos endereços em classes.

Os próximos três capítulos expandem o conceito de IP, cada um deles tratará de uma particularidade em detalhes.

11.2 Endereços para a Internet Virtual

Para se alcançar à meta de uma Internet (que é promover a comunicação entre redes de computadores), o IP deve esconder os detalhes da rede física e oferecer facilidades de uma grande rede. Uma rede virtual funciona exatamente como qualquer outra rede, ou seja, permite que os computadores enviem e recebam pacotes de informações. A principal diferença entre uma Internet e uma rede física é que uma Internet é apenas uma abstração criada inteiramente por softwares, por isso, o formato do pacotes, o esquema de endereçamento e as técnicas de entrega dos pacotes podem ser escolhidos independentemente da rede física.

Para uma Internet, o ponto de maior cuidado deve ser o endereçamento. Para que uma Internet passe a impressão de ser uma rede única, uniforme, é preciso que todos os hosts usem o mesmo esquema de endereçamento. Isso não é muito simples por que como já foi visto cada tecnologia de rede usa um esquema de endereçamento próprio, fazendo com que os formatos dos endereços sejam diferentes e incompatíveis. Para garantir um esquema de endereçamento uniforme para todos os hosts, quem define o formato do endereço é um software, que endereça independentemente do tipo de rede. Com um formato de endereço igual para todos os hosts, a Internet pode funcionar como uma rede qualquer.

11.3 O Esquema de Endereçamento do IP

No protocolo TCP/IP o endereçamento é especificado pelo IP. O IP padrão especifica que cada host da Internet deve receber um único número de 32 bits (endereço de IP ou endereço de Internet). Todo pacote enviado através da rede carrega consigo o endereço de IP (número de 32 bits) do computador que o envio e do computador de destino. É importante saber que o endereço de Internet deve ser um número binário de 32 bits que não pode se repetir, ou seja, se for atribuído a um host nenhum outro na Internet poderá usá-lo.

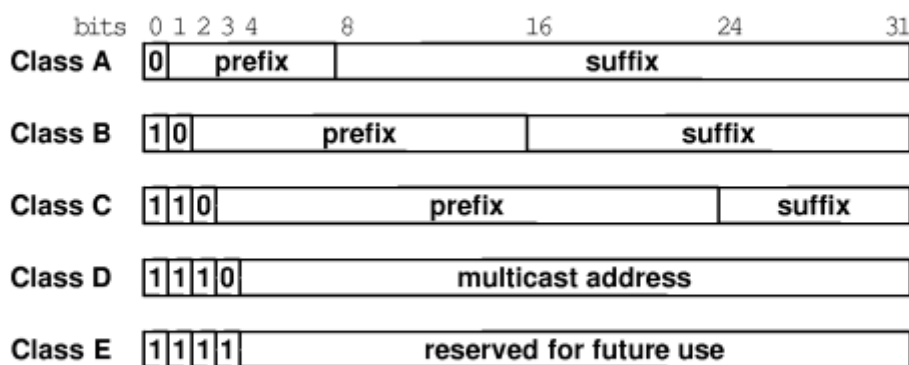
11.4 Hierarquia no IP

Para tornar o roteamento mais eficiente, o IP é dividido em duas partes: prefixo e sufixo. O endereço do prefixo identifica a rede na Internet, enquanto o sufixo identifica o computador na rede local (não na Internet). Para entender, basta pensar que cada rede da Internet recebe um valor único para identificação (network number, que em português significa número da rede). Falar em número da rede é o mesmo que falar em prefixo; falar em sufixo é o mesmo que falar em número do computador dentro da rede (cada computador recebe um único número que não pode ser usado por nenhum outro computador da mesma rede). Perceba que se dois computadores tiverem o mesmo sufixo, mas estiverem em redes diferentes, ou seja, tiverem prefixos diferentes, os endereços de IP serão válidos, porque serão únicos; um endereço de IP é formado por prefixo e sufixo.

11.5 Classes dos Endereços de IP

Depois de escolher tamanho do IP (32 bits) e dividi-lo em duas partes, os designers do IP precisavam determinar quantos bits colocar em cada uma das partes (prefixo e sufixo). O prefixo precisa de um número de bits suficiente para garantir que cada rede da Internet tenha um número único. Já o sufixo, precisa de um número de bits suficiente para permitir que todos os computadores da rede tenham um número único. A questão não é simples, se os bits reservados para o prefixo forem muitos, consegue-se acomodar muitas redes na Internet, porém todas com poucos computadores. Se a situação for contrária (muitos bits reservados para o sufixo), consegue-se acomodar poucas redes, mas elas poderão ter muitos computadores.

As redes que formam uma Internet são arbitrárias, ou seja, uma pode ser grande, outra pequena, etc. Conseqüentemente os designers tiveram que desenvolver um esquema de endereçamento que fosse capaz de acomodar redes grandes e pequenas. O esquema desenvolvido divide o endereço de IP em três classes, onde cada classe tem tamanho diferente de prefixo e sufixo. Na figura abaixo temos as classes de endereço de IP. Note que apenas com os quatro primeiros bits já é possível identificar a classe do IP.



As classes A, B e C são chamadas de classes primárias porque são usadas para endereços de hosts. A classe D é usada para multicasting, que é quando um pacote é transferido simultaneamente para um grupo de computadores da Internet. A classe E ainda não é usada, ela está reservada a usos futuros.

11.6 Computando a Classe de um Endereço

O software que computa o IP computa também a classe do IP de destino de um pacote, e faz isso sempre que ele (o software) recebe um pacote. Este processo deve ser eficiente porque ocorre muitas vezes. Diz-se que um endereço de IP é auto-identificável porque a classe pode ser verificada a partir do próprio endereço. A figura abaixo mostra a tabela usada para identificar a classe.

Primeiros 4 bits do endereço	Tabela em decimal	Classe dos endereços
0000	0	A
0001	1	A
0010	2	A
0011	3	A
0100	4	A
0101	5	A
0110	6	A
0111	7	A
1000	8	B
1001	9	B
1010	10	B
1011	11	B
1100	12	C
1101	13	C
1110	14	D
1111	15	E

11.7 Notação Decimal de Ponto

Apesar do endereço de IP ser um número binário de 32 bits, ele raramente é apresentado desta forma. Quando um usuário está interagindo com um computador é muito mais conveniente usar a notação decimal. Na notação decimal de ponto cada grupo de oito bits é transformado para décima, para separar os grupos de oito bits usa-se um ponto, daí a explicação do nome desta notação. A imagem abaixo ilustra alguns exemplos: do lado esquerdo está a notação binária, e do lado direito o mesmo endereço está representado em decimal de ponto.

32-bit Binary Number	Equivalent Dotted Decimal
10000001 00110100 00000110 00000000	129.52.6.0
11000000 00000101 00110000 00000011	192.5.48.3
00001010 00000010 00000000 00100101	10.2.0.37
10000000 00001010 00000010 00000011	128.10.2.3
10000000 10000000 11111111 00000000	128.128.255.0

Perceba que cada grupo de oito bits é tratado com um único decimal, por isso, os endereços de IP podem variar de 0.0.0.0 até 255.255.255.255.

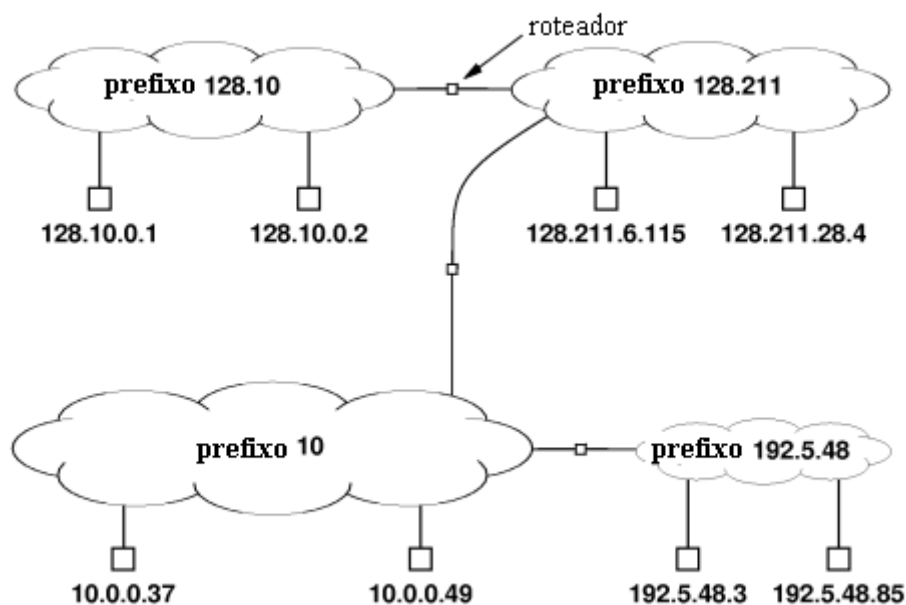
A tabela das classes de IP fica assim:

Classe	Faixa de valores
A	0 – 127
B	128 – 191
C	192 – 223
D	224 – 239
E	240 – 255

O número de redes possíveis numa Internet está diretamente relacionado com a classe dos IPs. O assunto pode ser facilmente abreviado pela tabela abaixo.

Endereço da Classe	Nº no prefixo	Max. Numero de redes	Nº no sufixo	Max. Número de Host por rede
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

Uma Internet pode ter várias redes distintas, e cada uma delas com uma classe de endereços. A próxima figura dedica-se a exemplificar uma Internet com mais de uma classe. No exemplo há duas redes de classe B, uma de classe A e uma classe C.



11.8 Endereços Especiais de IP

Além de cada computador ter seu endereço, é conveniente que exista um endereço que possa ser usado por todos os computadores de uma rede, ou por um grupo de computadores (podendo estar um em cada rede); um endereço assim é dito reservado. Endereços reservados nunca devem ser usados por hosts.

11.8.1 Endereços de Rede

É conveniente ter um endereço que possa ser usado por todos os computadores de uma rede. O software de protocolo de IP impede que um host use um endereço que contenha todos os bits do sufixo em zero porque este é o endereço relativo à rede. Por exemplo, se uma rede tem prefixo 128.211, então o endereço 128.211.0.0 é usado como endereço da rede. É importante ressaltar que apesar do endereço da rede existir um pacote nunca pode ter esse endereço como destino.

11.8.2 Endereço de Transferência Direta

Algumas vezes é conveniente enviar uma cópia de uma pacote para todos os hosts de determinada rede. Para fazer com que essa transferência seja fácil, o IP define um endereço de transferência direta para cada rede física. Quando um pacote é enviado por transferência direta, uma simples cópia do pacote viaja através da internet até que ele encontre a rede especificada. O pacote então é entregue a todos os hosts da rede. O endereço para transferência direta seta todos os bits do sufixo em um, ao contrario do endereço de rede. Se a rede tiver um hardware que suporta uma transferência direta, o pacote é transmitido diretamente para todos os hosts, se a rede não tiver um hardware de suporte é preciso que um software faça a distribuição para cada host.

11.8.3 Endereço de Transferência Limitada

Trata-se de uma transferência em uma rede local (LAN); informalmente diz-se que a transferência limita-se a um único fio. A transferência limitada é usada pelo computador quando ele ainda sabe qual é o número da rede (isso ocorre quando um computador está sendo iniciado).

11.8.4 This Computer Address

Um computador precisa conhecer seu endereço de IP para poder enviar e receber pacotes. O protocolo TCP/IP contém protocolos que podem ser usados para obter o endereço de IP automaticamente quando o computador é iniciado.

11.8.5 Endereço de Loopback

O IP define um endereço que é usado para testar os aplicativos de rede, este é o endereço de loopback. O IP reserva na classe A, o prefixo 127 para loopback; se um host usa o prefixo 127 ele é “invisível para a rede”. Por convenção, programadores costumam usar o endereço 127.0.0.1 para fazer o teste de loopback. Durante um teste nenhum pacote pode ser enviado pelo host, conseqüentemente, o endereço de loopback nunca viaja pela rede.

11.9 Resumo dos Endereços Especiais

Prefix	Suffix	Type Of Address	Purpose
all-0s	all-0s	this computer	used during bootstrap
network	all-0s	network	identifies a network
network	all-1s	directed broadcast	broadcast on specified net
all-1s	all-1s	limited broadcast	broadcast on local net
127	any	loopback	testing

Os endereços especiais são reservados e nunca devem ser usados por hosts; cada endereço especial tem um uso determinado, por exemplo, um endereço facilita o envio de um pacote em massa.

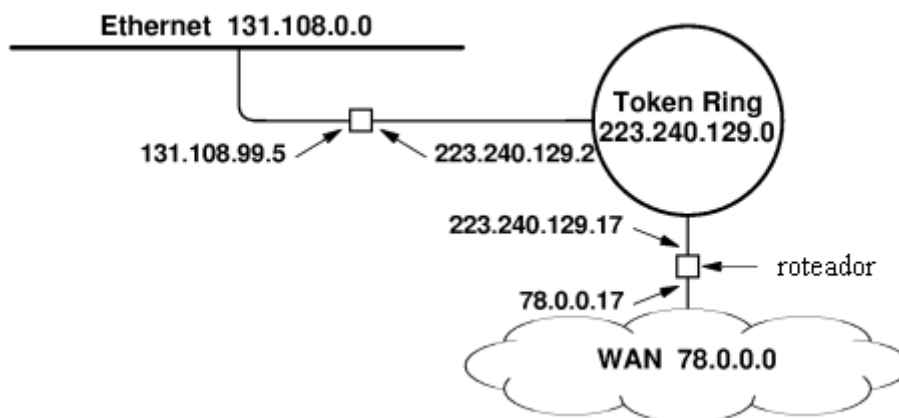
11.10 Roteadores e o Princípio do Endereçamento de IP

Além de marcar cada host na internet, o software de IP especifica que os roteadores também devem ser marcados por endereços de IP. De fato cada roteador deve receber no mínimo dois endereços de IP, para entender o motivo deve-se relembrar de dois fatos importantes:

- O roteador tem conexões com várias redes (mínimo duas);
- Cada endereço de IP contém um prefixo que identifica a rede;

Como o roteador está ligado a duas redes, um único endereço de IP não é suficiente para ele. O esquema de IP no roteador pode explicado pelo princípio fundamental:

Um endereço de IP não identifica um computador específico. No entanto cada endereço de IP identifica a conexão entre um computador e uma rede. Um computador com conexão a várias redes (um roteador) deve receber um IP para cada conexão.



O IP não requer que o mesmo sufixo seja dado para todas as interfaces do roteador.

11.11 Multi-Homed Hosts

A pergunta a ser respondida é: um host pode se conectar a mais de uma rede? A resposta é sim! E o procedimento recebe o nome de multi-homed. O multi-homed é, na verdade, usado para melhorar o desempenho, ou seja, conectar várias redes. Ele faz com que a passagem do pacote pelo roteador não seja necessária caso o roteador esteja congestionado, o multi-homed é uma rota alternativa para os pacotes. Do mesmo modo que o roteador, o multi-homed também precisa de mais de um endereço de IP.

Capítulo 12 – Binding Protocol Address

12.1 Introdução

Esse capítulo descreve três mecanismos que são utilizados para realizar o mapeamento dos endereços em uma rede, utilizando o Protocolo IP para ilustrar cada caso. O primeiro mecanismo é utilizado no hardware de rede e consiste em uma tabela que contém todas as relações de endereços dos Hosts na rede. O segundo método utiliza funções matemáticas para realizar a tradução. O terceiro mecanismo é o mais interessante porque usa a comunicação entre dois computadores através da rede para determinar os endereços.

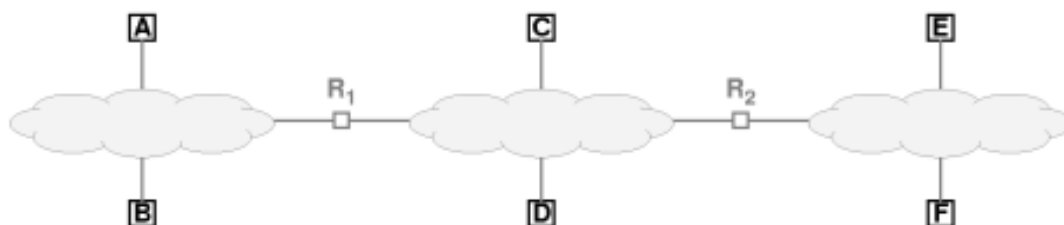
12.2 Protocolo de Endereçamento e Entrega de Pacotes

Quando um aplicativo gera dados para serem enviados através da Internet, o software encapsula os dados em um pacote, o qual contém o endereço do destinatário. O software em cada Host e Roteador usam este endereço para definir qual será o próximo passo para o pacote. Uma vez que esse passo é definido, o software transfere o pacote através da rede para o host ou roteador selecionado.

Assim, o software lida com o endereço IP quando esta enviando pacotes. Tanto o endereço do próximo passo ou pulo, e o endereço do destinatário são endereços IP. Infelizmente, este protocolo de endereços não pode ser usado quando se transmite pacotes através de uma rede física de hardware, pois o hardware não entende o endereçamento IP. Ao invés, um pacote enviado na rede deve usar o formato de pacote de hardware, e todos os endereços devem ser endereços de hardware.

12.3 Determinação dos Endereços

A tradução do endereço de protocolo de um computador para o equivalente endereço de hardware deste computador é conhecida como Address Resolution, ou determinação de endereço. Um computador pode determinar o endereço de outro computador apenas se estes computadores estão ligados na mesma rede física – um computador nunca determina o endereço de um computador que esta em uma rede remota.



Dizemos que a determinação de endereços é sempre restrita a uma única rede. Na figura acima, se um aplicativo no host A envia uma mensagem para um aplicativo no host F, o qual esta em uma rede remota, o software em A não determina o endereço do host F. Ao invés disso, o software em A primeiro determina que o pacote primeiro passe pelo roteador R1. O software em A então determina o endereço de R1, e envia o pacote para o roteador. O software em R1

determina que o pacote deva alcançar R2, determina o endereço de R2 e envia o pacote. Finalmente, R2 recebe o pacote, descobre que o destinatário F está ligado a rede a direta, então determina o endereço de F e entrega o pacote a F.

Resumindo:

Mapeamento entre um protocolo de endereços e o endereço de hardware é chamado de determinação de endereços (Address Resolution). Um host ou roteador utiliza a determinação de endereços quando precisa enviar um pacote para outro computador no mesmo enlace físico. Um computador nunca determina o endereço de um computador em uma rede remota.

12.4 Técnicas de Determinação de Endereços

Os algoritmos de determinação de endereços podem ser divididos em três categorias básicas:

- Busca em Tabelas. Os endereços ou mapeamentos são salvos em uma tabela memória, na qual o computador busca quando precisa determinar o endereço.
- Computação de Forma fixa. O endereço de protocolo designado para um computador é escolhido com cuidado assim o endereço de hardware do computador pode ser obtido usando funções booleanas e operações aritméticas.
- Troca de Mensagens. Computadores trocam mensagens pela rede para definir um endereço. Um computador envia uma mensagem que requisita o endereço, e outro computador responde com um mensagem que contém a informação requisitada.

12.5 Determinação de Endereços com Busca em Tabela

As tabelas de endereços requerem uma estrutura de dados que contém informações sobre os endereços de protocolo e hardware. A tabela consiste em um vetor. Cada entrada no vetor contém um par (P,H), onde P é o endereço de protocolo e H é o equivalente endereço de Hardware. A figura abaixo ilustra o método:

Endereço IP	Endereço de Hardware
197.15.3.2	0A:07:4B:12:82:36
197.15.3.3	0A:9C:28:71:32:8D
197.15.3.4	0A:11:C3:68:01:99
197.15.3.5	0A:74:59:32:CC:1F
197.15.3.6	0A:04:BC:00:03:28
197.15.3.7	0A:77:81:0E:52:FA

Para cada rede física, uma tabela de ligação de endereços é utilizada. Como consequência, todos os endereços de IP em uma tabela tem o mesmo prefixo.

A grande vantagem da tabela é a generalidade – uma tabela pode armazenar os endereços para um set de computadores arbitrários na rede. Em particular, um endereço de protocolo pode ser mapeado para um endereço de hardware arbitrário. Quando se tem o endereço N de IP do próximo salto, o software procura na tabela ate que encontre uma entrada com endereço N de IP. O software então extrai o endereço de hardware da entrada na tabela.

Para redes com grande número de host, uma busca seqüencial requer excessivo uso do tempo de processamento da CPU. Nesses casos o software pode utilizar duas implementações padrões para melhorar a eficiência computacional: hashing ou direct indexing.

Hashing é uma técnica de estrutura de dados, com vários propósitos e conhecida por muitos programadores. Já Direct Indexing é um pouco mais eficiente, mas menos geral. Em

particular, Direct Indexing é possível apenas em casos que o endereço de protocolo é determinado num conjunto compacto. Para utilizar Direct Indexing, o software mantém um vetor unidimensional de endereços de hardware, e utiliza o sufixo do IP do host para indexar o vetor. A figura abaixo ilustra a técnica.



12.6 Determinação de Endereços com Computação de forma fixa

Este método utiliza funções matemáticas para mapear o endereço de IP ao endereço de hardware. Se a relação entre o endereço de IP e o correspondente endereço de hardware for direta, o mapeamento requer apenas poucas operações aritméticas.

Como um exemplo, suponha que uma rede configurável foi determinada como Classe C, com o endereço de rede 220.123.5.0. Assim que os computadores são adicionados à rede, cada computador tem seu sufixo do endereço IP determinado e igual ao endereço de hardware do mesmo. O primeiro host a se conectar tem o endereço de IP 220.123.5.1 e o endereço de hardware 1. O segundo host tem endereço de IP 220.123.5.2 e endereço de hardware 2. Os sufixos não precisam ser sequenciais: se um roteador é ligado à rede, este tem endereço de IP 220.123.5.101, e logo endereço de hardware 101. Dando o endereço de IP de qualquer computador na rede, o endereço de hardware do computador pode ser determinado facilmente por uma operação booleana E :

$$\text{Hardware_address} = \text{ip_address} \& 0\text{xff}$$

Esta formula é trivial para o programa, assim não requer uma tabela de valores, sendo computacionalmente mais eficiente.

12.7 Determinação de Endereços com Troca de Mensagens

Os métodos de determinação de endereços descritos acima podem ser computados utilizando apenas um computador a cada passo: as instruções e dados necessários para encontrar o endereço são mantidos no sistema operacional do computador. Uma alternativa para a computação local é um acesso distribuído, no qual um computador que necessite determinar o endereço de outro computador envia uma mensagem através da rede e recebe uma resposta. Esta mensagem leva uma requisição que especifica o endereço de protocolo, e a resposta leva o correspondente endereço de hardware.

Quando essa mensagem de requisição de endereços deve ser enviada? A maioria dos sistemas de protocolos escolhe um entre dois possíveis métodos. No primeiro método, a rede tem um ou mais servidores que tem a função de responder essas requisições de endereços. Quando um host precisa do endereço, a mensagem é enviada para um dos servidores, o qual enviara uma resposta. O host pode enviar a mensagem para cada servidor em sequência até que receba uma resposta, ou envia em broadcast a requisição para todos servidores.

Em um segundo método, cada computador na rede participa na determinação de endereços concordando em responder a requisição para seu endereço. Quando um host precisa de um endereço, ele envia uma requisição em broadcast na rede. Todos os host recebem a requisição e examinam o endereço requisitado. Se a requisição for igual ao endereço do computador, este enviará uma resposta com seu endereço diretamente para o host que precisa do endereço.

A vantagem do primeiro método é a centralização. Como pouco servidores lidam com todas as requisições de endereços, o controle, configuração e gerenciamento das requisições ficam mais fáceis. A vantagem do segundo método é a distribuição computacional. Os servidores podem ser caros. Adicionando também custos adicionais de hardware (ex: memória extra), os servidores têm manutenção cara, pois as informações dos endereços de protocolo e de hardware têm que ser armazenados no servidor e são atualizados quando novos computadores entram na rede ou endereços de hardware mudam. Fazendo com que cada computador responda por seu endereço, eliminam-se completamente os servidores.

12.8 Protocolo de Determinação de Endereços

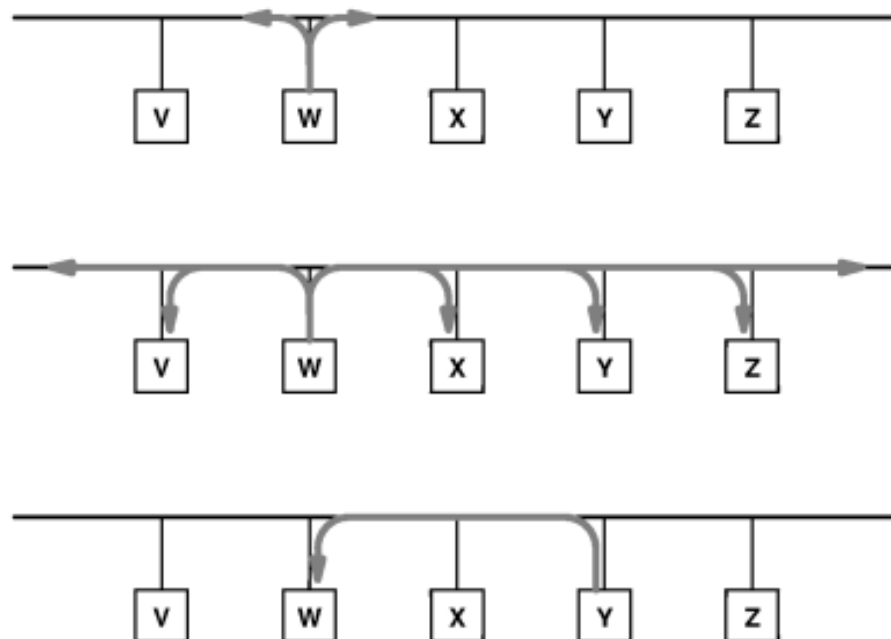
O protocolo TCP/IP pode utilizar qualquer um dos três métodos de determinação de endereços. O método da Busca em Tabelas é usado para determinar endereços em uma WAN. O método computacional é utilizado em redes configuráveis, e a troca de mensagens é utilizada em uma LAN que tem endereçamento estático.

Para garantir que todos os computadores concordam no mesmo e exato formato e no significado das mensagens utilizadas para a determinação dos endereços, o protocolo TCP/IP inclui o *Address Resolution Protocol* (ARP). O padrão ARP define dois tipos básicos de mensagens: uma requisição e uma resposta. A mensagem de requisição contém o endereço de IP e requer o endereço de hardware; a resposta contém ambos os endereços de IP enviados na requisição e o endereço de hardware.

12.9 Entrega da Mensagem ARP

O padrão ARP especifica exatamente como as mensagens ARP devem ser enviadas através de uma rede. O protocolo especifica que a mensagem deve ser enviada em broadcast para todos os computadores na rede. Cada computador recebe a mensagem e examina o endereço de IP. O computador mencionado na requisição envia a resposta; todos outros computadores processam e descartam a requisição, sem enviar resposta alguma.

Quando um computador envia uma resposta ARP, essa resposta não é enviada em broadcast. Ao invés disso, é enviada diretamente para o computador que realizou a requisição. A figura abaixo ilustra o processo descrito acima.



12.10 Formato da mensagem ARP

Mesmo que o protocolo ARP contenha exatamente a especificação da mensagem ARP, o padrão não define um formato fixo que deve ser usado em todas as comunicações. Os projetistas do ARP notaram que não poderiam escolher um tamanho fixo para o campo de endereço de hardware porque novas tecnologias de rede poderiam ser inventadas, tendo endereço de hardware maior que o projetado. Consequentemente os projetistas incluíram um campo “fixed-size” no início da mensagem ARP para especificar o tamanho do endereço de hardware que será usado.

Para aumentar a generalidade do ARP, os projetistas incluíram um campo tamanho de endereço também para o endereço de protocolo. Assim, o ARP não se restringe ao protocolo IP ou a um endereço de hardware específico.

A figura abaixo ilustra o formato da mensagem ARP, quando é utilizada com o protocolo de endereços IP e o protocolo de endereço de hardware Ethernet.

C	8	16	24	31
HARDWARE ADDRESS TYPE		PROTOCOL ADDRESS TYPE		
HADDR LEN	PADDR LEN	OPERATION		
SENDER HADDR (first 4 octets)				
SENDER HADDR (last 2 octets)		SENDER PADDR (first 2 octets)		
SENDER PADDR (last 2 octets)		TARGET HADDR (first 2 octets)		
TARGET HADDR (last 4 octets)				
TARGET PADDR (all 4 octets)				

Cada linha na figura corresponde a 32 bits na mensagem ARP. Os dois primeiros campos de 16 bits contêm o valor que especificam o tipo de endereçamento de hardware e protocolo estão sendo usados. Por exemplo, o campo **HARDWARE ADDRESS TYPE** contém 1 quando ARP é utilizado com a Ethernet, e o campo **PROTOCOL ADDRESS TYPE** contém 0x0800 quando o ARP é utilizado com IP. O segundo par de campos, **HADDR LEN** e **PADDR LEN** especificam o número de octetos no endereço de hardware e de protocolos. O campo **OPERATION** especifica se é uma mensagem de requisição (valor 1) ou resposta (valor 2).

12.11 Enviando a mensagem ARP

Quando um computador envia uma mensagem ARP, a mensagem ARP é transportada pelo hardware da rede. Esta mensagem é tratada como dados – o hardware de rede não entende o formato da mensagem ARP e não examina o conteúdo dos seus campos individuais.

12.12 Identificando a mensagem ARP

O campo *type field* no cabeçalho de um frame especifica que o frame contém uma mensagem ARP. Para a Ethernet, por exemplo, quando o campo *type field* tem o valor 0x806, este frame será composto de uma mensagem ARP. Assim, é desta maneira que um host identifica que a mensagem recebida é uma ARP.

Capítulo 13 – Datagramas IP e Encaminhamento de Datagramas

13.1 Introdução

Os capítulos anteriores descreveram a arquitetura da Internet, o endereçamento de Internet, e softwares de conversão de endereços para relacionar os endereços de Internet com os endereços físicos. Esse capítulo discutirá o serviço fundamental de comunicação em uma internet. Descreverá o formato dos pacotes que são enviados através da Internet, e discutirá como os roteadores processam e encaminham tais pacotes. Nos próximos capítulos estendem a discussão considerando como roteadores usam seus hardwares para transmitir pacotes.

13.2 Serviço sem conexão

O objetivo de redes de trabalho é prover um pacote de sistemas de comunicação que permita que um programa rodando em um computador possa enviar dados para um programa rodando em outro computador. Em uma internet bem projetada, programas de aplicação ficam sem conhecimento da base física da rede – eles podem enviar e receber dados sem conhecer os detalhes da rede local a qual está conectado, da rede remota a qual o destino se conecta, ou a interconexão entre elas.

Programadores precisam decidir quais serviços de comunicação o protocolo de internet irá oferecer e como entrega esses serviços de modo eficaz. Em particular, programadores precisam decidir entre oferecer aos programas um serviço com conexão, um serviço sem conexão, ou ambas.

Programadores do TCP/IP escolheram incluir protocolos de serviço com e sem conexão. Eles escolheram fazer o serviço fundamental de entrega sem conexão, e para adicionar a confiabilidade do serviço com conexão que utiliza como base o serviço sem conexão.

13.3 Pacotes virtuais

Serviço de internet sem conexão é uma extensão da troca de pacotes – o serviço permite que um host transmita pacotes de dados individuais através da internet. Cada pacote viaja independentemente e contém informação que identifica o conteúdo desejado.

Como um pacote passa através de uma internet? Em geral, a resposta é que um roteador encaminha cada pacote de uma rede para outra. Um host de origem cria um pacote, com o endereço de destino no cabeçalho do pacote, e então envia o pacote para o roteador mais próximo. Quando um roteador recebe um pacote, ele usa o endereço de destino para selecionar o próximo roteador no caminho para o destino, e então o transmite. Eventualmente, o pacote alcança um roteador que pode entregar o pacote a seu destino final.

Qual o formato é usado em um pacote de internet? Infelizmente, o formato convencional de hardware, os frames, não pode ser utilizado para a internet. Para entender o porquê, lembre-se que um roteador pode conectar redes heterogêneas. O roteador não pode transferir uma cópia de um frame de um tipo de rede para outro, pois o formato do frame difere. Mais importante, o roteador não pode simplesmente reformatar o cabeçalho do frame porque as duas redes podem utilizar formatos de endereço diferentes (por exemplo, o endereçamento de um frame recebido pode não fazer sentido algum para outra rede).

Para superar essa heterogeneidade, o protocolo de internet define um formato de pacote de internet que é independente do hardware utilizado. O resultado é um pacote universal e virtual que pode ser transmitido intacto através dos diversos hardwares. O termo virtual quer dizer que o protocolo cria e lida com os pacotes de internet – o hardware não entende ou

reconhece o formato do pacote de internet. O termo universal quer dizer que cada host ou roteador em uma internet contém o protocolo que entende os pacotes de internet.

13.4 O Datagrama IP

O protocolo TCP/IP usa o nome datagrama IP para se referir aos pacotes de internet. Surpreendentemente, um datagrama IP tem o mesmo do frame. O datagrama começa com um cabeçalho seguido de uma área de dados.

A quantidade de dados em um datagrama não é fixa. O host que envia escolhe o quanto de dado que é mais apropriado para seu propósito em particular. Por exemplo, uma aplicação que transmite caracteres através da internet pode colocar cada caractere em um datagrama diferente, enquanto que uma aplicação que transfere arquivos grandes pode enviar datagramas maiores.

Na atual versão do IP (versão 4), um datagrama pode conter um único Byte ou até 64KB, incluindo o cabeçalho. Na maioria dos datagramas, o cabeçalho é muito menor que a área de dados. Para entender o porquê, é necessário examinar o custo da transmissão de dados. Como cabeçalho de frame usado na camada física, um cabeçalho de datagrama representa perda – enquanto a rede é ocupada transferindo cabeçalhos, não poderá transferir os dados do usuário. Como o tamanho do datagrama é fixo, enviar grandes datagramas resulta em mais bytes de dados enviados por unidade de tempo (por exemplo, maior largura de banda).

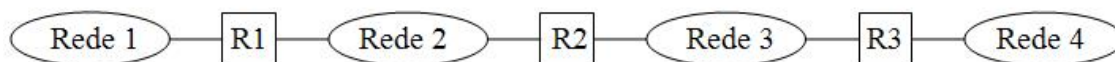
Similar ao cabeçalho de frame, o cabeçalho de um datagrama contém informação para o roteamento do mesmo através da internet. Por exemplo, o cabeçalho contém o endereço do computador que enviou o datagrama assim como o endereço do computador de destino. O endereço presente no cabeçalho de um datagrama difere do endereço usado no cabeçalho do frame – um datagrama contém um endereço IP, enquanto um frame contém um endereço de hardware.

13.5 Encaminhando um datagrama IP

Dizemos que um datagrama atravessa uma internet seguindo um caminho desde sua fonte inicial através dos roteadores até seu destino final. Cada roteador ao longo do caminho recebe o datagrama, extrai o endereço de destino do cabeçalho, e usa esse endereço para determinar o próximo salto para o qual o datagrama deveria ser enviado. O roteador então encaminha o datagrama para o próximo salto, que pode ser o destino final ou mais um roteador.

Para fazer uma escolha eficiente do próximo salto e tornar a computação compreensível para os homens, cada roteador IP mantém uma tabela de roteamento. A tabela de roteamento deve ser inicializada com o roteador, e deve ser atualizada se a topologia mudar ou algum hardware falhar.

Conceitualmente, a tabela de roteamento possui várias entradas, sendo que cada uma especifica um destino e um próximo salto utilizado para alcançar tal destino.



(a)

Destino	Próximo salto
Rede 1	Roteador 1
Rede 2	Entrega direta
Rede 3	Entrega direta

Rede 4	Roteador 3
--------	------------

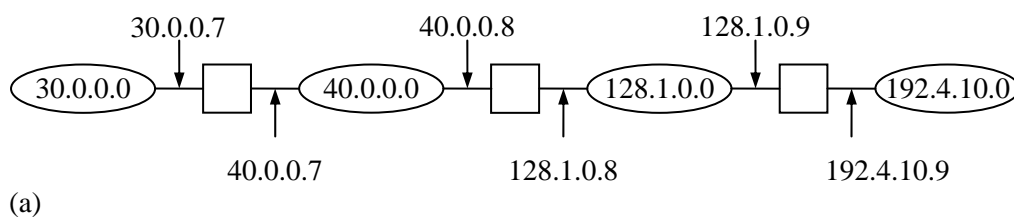
(b)

Como a figura mostra, o roteador R2 conecta-se diretamente às redes 2 e 3. Por isso, R2 pode entregar um datagrama diretamente para qualquer destino dessas duas redes. Quando um datagrama é destinado para a rede 4, por exemplo, R2 o enviaria para o roteador R3.

Todos os destinos listados em uma tabela de roteamento é uma rede, e não um host individual. Tal distinção é importante porque uma internet pode conter 1000 vezes mais hosts do que redes. Por isso, utilizar redes como destino mantém as tabelas de roteamento menores.

13.6 Endereços IP e entradas de tabelas de roteamento

Na prática, uma tabela de roteamento é um pouco mais complexa do que foi a ilustrada anteriormente. Primeiro, o campo Destino em cada entrada contém o prefixo de rede da rede de destino. Segundo, existe um campo adicional em cada entrada que contém a máscara de sub-rede que especifica quais bits do endereço de destino correspondem ao prefixo de rede. Terceiro, um endereço IP é utilizado quando o próximo salto é um roteador. A figura abaixo mostra como uma tabela de roteamento deveria se parecer:



Destino	Máscara	Próximo salto
30.0.0.0	255.0.0.0	40.0.0.7
40.0.0.0	255.0.0.0	Entrega direta
128.1.0.0	255.255.0.0	Entrega direta
192.4.10.0	255.255.255.0	128.1.0.9

(b)

Podemos observar que as duas primeiras redes têm um prefixo classe A, a terceira rede tem um prefixo classe B e a quarta rede possui um prefixo classe C. A cada roteador foram designados dois endereços IP, um para cada interface. Por exemplo, o roteador que conecta a rede 30.0.0.0 à rede 40.0.0.0 foi designado com os endereços 30.0.0.7 e 40.0.0.7. Embora ambas as interface do roteador tenham recebido o mesmo sufixo de host, IP não requer essa uniformidade – um administrador de rede é livre para designar diferentes valores para cada interface.

13.7 O campo Máscara e Encaminhamento de datagramas

O processo que utiliza a tabela de roteamento para selecionar o próximo salto para um dado datagrama é chamado roteamento ou encaminhamento. O campo Máscara em uma entrada de uma tabela de roteamento é usado para extrair a parte da rede de um endereço durante a

procura. Para entender como a Máscara é utilizada, imagine que a um software de roteamento é dado um datagrama para encaminhar. E assuma também que o datagrama possui um endereço IP D . O software de roteamento deve encontrar uma entrada na tabela de roteamento que especifique o próximo salto para D . Para isso, o software examina cada entrada na tabela usando a máscara para extrair o prefixo do endereço D e comparar o resultado com o campo Destino da mesma entrada. Se os dois forem iguais, o datagrama será encaminhado para o próximo salto daquela entrada.

A máscara torna a extração do prefixo eficiente – o software faz a operação Booleana e entre a máscara e o endereço de destino do datagrama, D . Sendo assim, para examinar a entrada i , pode ser expressa assim:

se $((\text{Máscara}[i] \& D) == \text{Destino}[i])$ encaminhar para o próximo salto $[i]$;

Como exemplo, considerar um datagrama com endereço de destino 192.4.10.3, e assuma que esse mesmo datagrama chegue ao roteador que possui a tabela de roteamento apresentada anteriormente. E assuma ainda que o software procure pelas entradas em ordem. A primeira entrada falharia porque $255.0.0.0 \& 192.4.10.3$ não é igual a 30.0.0.0. Após rejeitar a segunda e terceira entradas da tabela, o software de roteamento escolheria o próximo salto 128.1.0.9, por que:

$$255.255.255.0 \& 192.4.10.3 == 192.4.10.0$$

13.8 Destino e Endereços Próximo salto

Qual é a relação entre um endereço de destino de um cabeçalho de um datagrama e o endereço do próximo salto para o qual o datagrama é encaminhado? O campo *Endereço IP de Destino* em um datagrama possui o endereço de destino final. Quando um roteador recebe um datagrama, ele extrai o endereço de destino, D , e o utiliza para calcular o endereço do próximo roteador para o qual o datagrama deveria ser enviado, N . Embora o datagrama seja enviado diretamente para o endereço N , em seu cabeçalho estará ainda o endereço de destino D .

Todas as rotas são calculadas através dos endereços IP. Depois de calculado o endereço do próximo salto, N , o IP utiliza a ligação entre endereços para traduzir N para um endereço de hardware equivalente para a transmissão. No próximo capítulo, iremos aprender como o datagrama é enviado através da rede física.

13.9 Entrega a qualquer custo

Fora isso a definição do formato dos datagramas de internet, o IP define a semântica de comunicação, e usa o termo *A Qualquer Custo* para descrever o serviço que oferece. Na essência, o padrão especifica que embora o IP faça uma tentativa de entrega a qualquer custo de cada datagrama, o IP não garante que ele lidará dos problemas com:

- Duplicação de datagrama
- Entregas atrasadas ou fora de ordem
- Dados corrompidos
- Perda de datagramas

Camadas adicionais de protocolos são necessárias para lidar com cada um desses problemas.

Pode parecer estranho para o IP especificar que esses erros possam ocorrer. Entretanto, há uma razão importante: cada camada do protocolo é responsável por certos aspectos da comunicação, e o IP não lida com esses problemas. E mais, porque as redes físicas podem

causar esses problemas, qualquer software que utilize o IP deve tomar a responsabilidade de solucionar esses problemas.

13.10 Formato do cabeçalho IP

A figura abaixo mostra os campos de um cabeçalho de um datagrama IP, incluindo o *Endereço IP de Origem*, que contém o endereço de Internet da origem, o *Endereço IP de Destino*, que contém o endereço de Internet do destino planejado, e o campo *Tipo*, que especifica o tipo de dado transmitido.

0	4	8	16	19	24	31
Versão	Cabec. Tam.	Tipo do serv.	Tamanho Total			
Identificação			Flags	Offset do Fragmento		
Tempo de vida		Tipo	Checksum do cabeçalho			
Endereço IP de Origem						
Endereço IP de Destino						
Opções do IP (pode ser omitido)					Preenchimento	
Dados						
.						
.						
.						

Cada campo em um cabeçalho de um datagrama IP tem um tamanho fixo. O datagrama começa com um número de 4 bits da versão do protocolo (a versão atual é a 4) e 4 bits do tamanho do cabeçalho que especifica o número de grupos de 32 bits no cabeçalho. O campo *Tipo de Serviço* contém um valor que especifica se a origem prefere que o datagrama viaje através de uma rota com mínimo atraso ou com maior largura de banda: um roteador que saiba múltiplas rotas para o destino pode usar esse valor para escolher a rota. O campo *Tamanho Total* contém um número inteiro de 16 bits que especifica o número total de bytes em um datagrama, incluindo o cabeçalho e a área de dados. O capítulo 19 explicará os campos *Identificação*, *Flags*, e *Offset do Fragmento*.

O campo *Tempo de Vida* é usado para evitar que um datagrama viaje para sempre em um caminho que contenha um loop; esse tipo de caminho pode ocorrer quando o software não funciona corretamente ou quando um administrador configura erroneamente um roteador. A origem inicializa esse campo com um número inteiro e positivo entre 1 e 255. Cada roteador que lida com esse datagrama decrementa em uma unidade tal número. Se esse contador chegar à zero, o datagrama é descartado e uma mensagem de erro é enviada de volta para a origem.

O campo *Checksum do Cabeçalho* assegura que os bits do cabeçalho não foram modificados durante o trânsito. Um host de origem calcula a soma dos complementos de 1 de todas as quantidades de 16 bits no cabeçalho excluindo o próprio campo de checksum, e então guarda essa soma no campo *Checksum do Cabeçalho*. Um host receptor calcula a mesma soma de quantidades de 16 bits, incluindo o campo de checksum. Se o checksum estiver correto, o resultado será 0 (Matematicamente, o complemento de 1 é um aditivo inverso. Por isso, somando um valor ao seu complemento produzirá o valor 0).

Para manter os cabeçalhos dos datagramas menores possíveis, o IP define um conjunto de *Opções* que podem estar presentes, se necessário: Quando um datagrama IP não carrega tais opções, o campo *Tamanho do Cabeçalho* contém o valor 5, e o cabeçalho termina depois do campo *Endereço de IP de destino*. Devido ao tamanho do cabeçalho ser especificado por múltiplos de 32 bits, se o campo *Opções* não terminar em um limite de 32 bits, o campo *Preenchimento* conterá bits 0 para fazer com do cabeçalho um múltiplo de 32 bits.

Capítulo 14 – Encapsulamento IP, Fragmentação e Reunião

14.1 Introdução

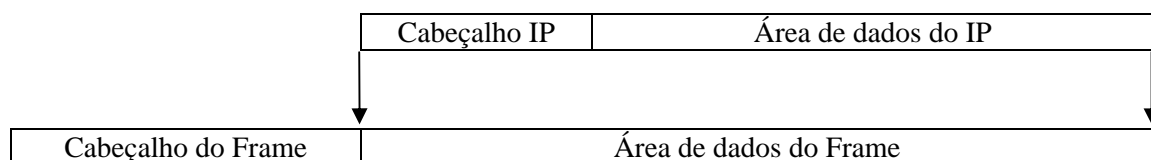
No capítulo anterior descrevemos um formato de um datagrama IP e discutimos a informação em uma tabela de roteamento é utilizada para selecionar o próximo salto a cada passo ao longo do caminho até o destino do datagrama. Esse capítulo concluirá a discussão sobre o IP descrevendo a transmissão do datagrama em detalhes. Mostrará como um host ou um roteador envia um datagrama através da rede física, e como os roteadores lidam com o problema de enviar grandes datagramas.

14.2 Transmissão de Datagramas e Frames

Quando um host ou um roteador lidam com um datagrama, o IP seleciona primeiro o próximo salto para o qual o datagrama será enviado, N , e então transmite o datagrama pela rede física para N . Infelizmente, o hardware de rede não entende o formato do datagrama ou do endereçamento de Internet. Ao invés disso, cada tecnologia de hardware define um formato de frame e um esquema de endereçamento físico; o hardware só aceita e entrega pacotes que tenham o formato específico de frame e o esquema de endereçamento de hardware específico. Mais importante ainda, como a Internet pode conter tecnologias de rede heterogêneas, o formato do frame necessário para atravessar uma rede pode diferir do formato de frame necessário para atravessar a rede anterior.

14.3 Encapsulamento

Como pode um datagrama ser transmitido através de uma rede física que não entende o formato do datagrama? A resposta está na tecnologia chamada *encapsulamento*. Quando um datagrama IP é encapsulado em um frame, o datagrama inteiro é colocado na área destinada aos dados, dentro de um frame. O hardware de rede trata o frame que contém o datagrama exatamente da mesma maneira que qualquer outro frame. Na verdade, o hardware não examina ou modifica o conteúdo da área de dados de um frame. A figura abaixo ilustra esse conceito.



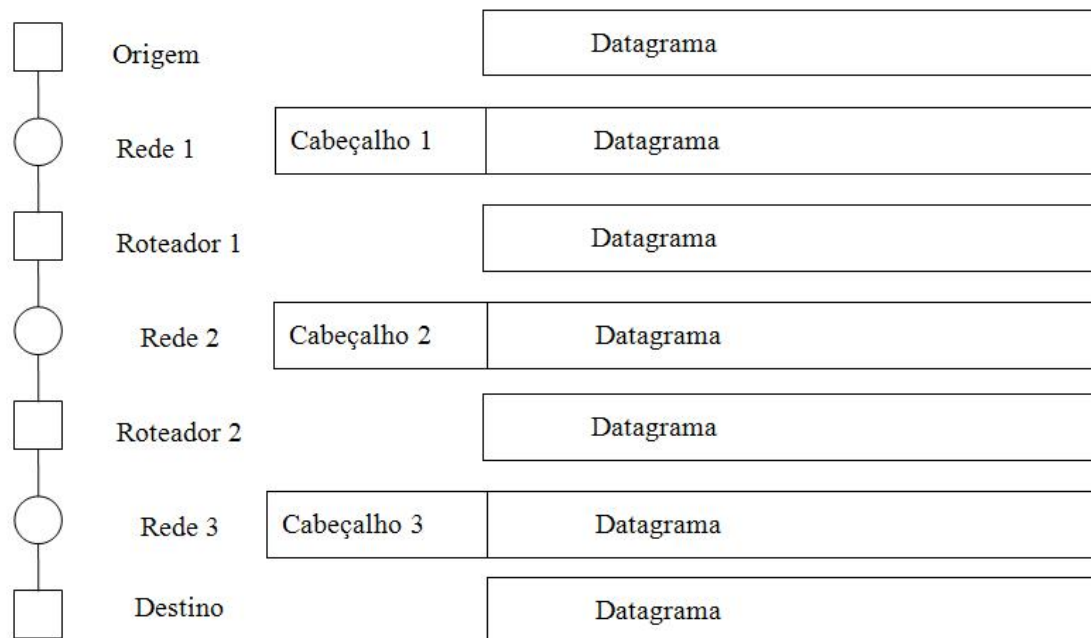
Como que um receptor sabe quando que a área de dados de um frame recebido contém um datagrama IP ou outro tipo de dado? A origem e o receptor devem estar em acordo quanto ao valor usado no campo *Tipo* do cabeçalho do frame. Quando um datagrama é colocado em um frame, a origem preenche o campo *Tipo* do frame com o número especial reservado para o *IP*. Quando um frame chega com o número especial no seu campo *Tipo*, o receptor sabe que a área de dados contém um datagrama IP.

Um frame que carregue um datagrama IP deve ter um endereço de destino como um frame normal. Para isso, em adição a colocar o datagrama na área de dados do frame, o encapsulamento requer que a origem forneça um endereço físico do próximo computador para o qual o datagrama deve ser enviado. Para calcular o endereço apropriado, um software no

computador de origem deve realizar a ligação entre endereços como descrito no capítulo 17. A ligação traduz o endereço IP do próximo salto em um endereço de hardware equivalente, o qual é então utilizado como o endereço de destino no cabeçalho do frame.

14.4 Transmissão através da Internet

O encapsulamento é aplicado a uma transmissão por vez. Depois que a origem seleciona o próximo salto, ela encapsula o datagrama em um frame e transmite o resultado pela rede física até o próximo salto. Quando um frame alcança o próximo salto, o software receptor remove o datagrama IP e descarta o frame. Se o datagrama precisar ser encaminhado através de outra rede, um novo frame será criado. A figura abaixo ilustra como um datagrama é quando está encapsulado e desencapsulado assim que ele caminha do host de origem até host de destino através de três redes diferentes e dois roteadores. Cada rede pode utilizar tecnologias de hardware diferentes, implicando que o formato dos frames irá diferir.



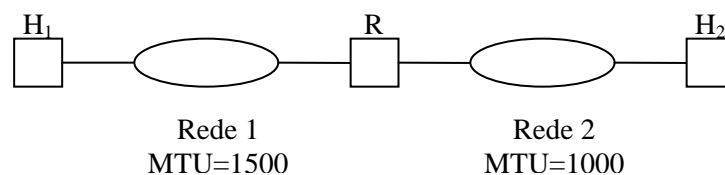
Como a figura mostra, hosts e roteadores guardam o datagrama na memória sem cabeçalhos adicionais. Quando um datagrama passa através de uma rede física, o datagrama é encapsulado de acordo com a rede. O tamanho do cabeçalho do frame que aparecerá antes do datagrama depende da tecnologia da rede. Por exemplo, se a rede 1 é uma Ethernet, o cabeçalho do frame 1 será um cabeçalho Ethernet. Similarmente, se a rede 2 for uma RDDI Ring, o cabeçalho do frame 2 será um cabeçalho FDDI.

É importante observar que cabeçalhos de frames não são acumulados ao longo da viagem pela internet. Antes de um datagrama ser transmitido através de uma dada rede, o datagrama é encapsulado, o que normalmente significa que um cabeçalho de frame será adicionado. Quando um frame chega ao próximo salto, o datagrama é removido do mesmo antes de ser encaminhado e encapsulado em um novo frame. Dessa forma, quando um datagrama alcança seu destino final, o frame que carrega o datagrama é descartado e o datagrama aparece exatamente com o mesmo tamanho que foi originalmente enviado.

14.5 MTU, Tamanho do Datagrama, e Encapsulamento

Cada tecnologia de hardware especifica uma quantidade máxima de dados que um frame pode carregar. Esse limite é conhecido como Unidade Máxima de Transmissão (MTU). Não existem exceções quanto ao limite de MTU – o hardware da rede não é projetado para aceitar ou transferir frames que carreguem mais dados que a MTU permite. Por isso, um datagrama deve ser menor ou igual ao MTU da rede ou não poderá ser encapsulado para a transmissão.

Em uma internet que contenha redes heterogêneas, restrições de MTU podem causar um problema. Em particular, devido a um roteador poder conectar redes com valores diferentes de MTU, um roteador pode receber um datagrama através de uma das redes que não pode ser enviado através da outra. A figura abaixo ilustra um roteador interconectando duas redes com valores de MTU diferentes.

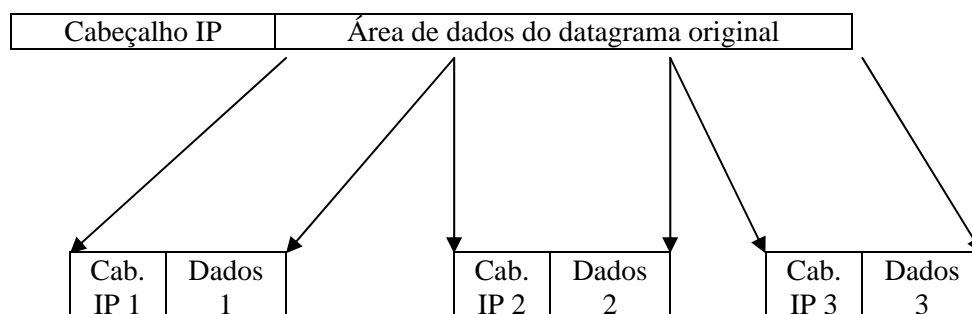


Na figura, o host 2 (H₂) está ligado a uma rede que tem um MTU de 1000. Então, cada datagrama que H₂ transmitir deve ter 1000 bytes ou menos. Entretanto, devido ao host 1 (H₁) estar ligado a uma rede que tem um MTU de 1500 Bytes, H₁ pode transmitir datagramas que contenham até 1500 Bytes. Se H₁ enviar um datagrama de 1500 bytes para H₂, o roteador (R) receberá o datagrama, mas não será capaz de enviar através da Rede 2.

Um roteador IP usa a técnica conhecida por *Fragmentação* para resolver o problema de MTUs heterogêneas. Quando um datagrama é maior que a MTU da rede pela qual ele deve ser enviado, o roteador divide o datagrama em pedaços menores chamados *fragmentos*, e envia cada fragmento independentemente.

Surpreendentemente, um fragmento tem o mesmo formato que os outros datagramas – um bit no campo *FLAGS* do cabeçalho indica quando um datagrama é um fragmento ou um datagrama completo. Outros campos do cabeçalho são preenchidos com informação que poderá ser utilizada para reunir os fragmentos para reproduzir o datagrama original. Em particular, o campo *Offset do Fragmento* no cabeçalho de um fragmento especifica em qual posição esse fragmento está localizado no datagrama original.

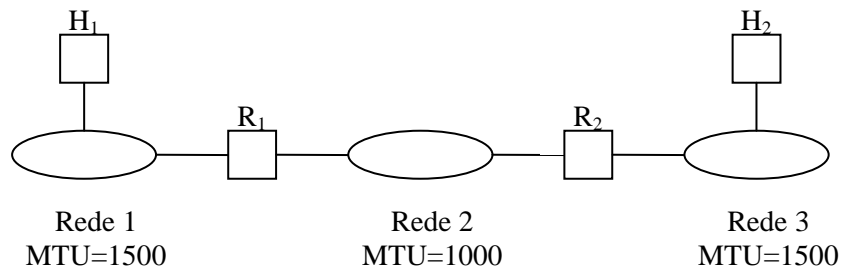
Para fragmentar um datagrama para transmissão através da rede, um roteador deve utilizar o MTU da rede e o tamanho do cabeçalho do datagrama para calcular o máximo de dados que poderão ser enviados em cada fragmento e o número de fragmentos que serão necessários. Então o roteador cria os fragmentos. Ele começa copiando o cabeçalho original no início de cada fragmento, e então modifica campos individuais. Por exemplo, o roteador atribui o valor 1 ao bit apropriado no campo *FLAGS* para indicar que aquele datagrama é um fragmento. Finalmente, o roteador copia os dados apropriados do datagrama original para cada fragmento, e os transmite. A figura abaixo ilustra o processo.



14.6 Reunião

O processo de criar uma cópia do datagrama original dos fragmentos é chamado de *Reunião*. Como cada fragmento começa com uma cópia do cabeçalho do datagrama original, todos os fragmentos têm o mesmo endereço de destino como o datagrama original do qual eles derivam. Além disso, o fragmento que carrega o pedaço final de dados atribui o valor 1 a um bit adicional no cabeçalho. Dessa forma, o receptor que irá fazer a reunião pode saber quando todos os fragmentos chegaram com sucesso.

Interessantemente, o Protocolo de Internet especifica que o host de destino final que deve reunir os fragmentos. Por exemplo, considere a internet da figura abaixo ilustrada:



Na figura, se o host H_1 enviar um datagrama de 1500 bytes para o host H_2 , o roteador R_1 dividirá o datagrama em dois fragmentos, que serão encaminhados para o roteador R_2 . O roteador R_2 não irá reunir os fragmentos. Ao invés disso R_2 utiliza o endereço de destino de cada fragmento para encaminhá-lo. O host de destino, H_2 , coleta os fragmentos e os reúne para reproduzir o datagrama original.

Sendo o destino final o responsável por reunir os fragmentos tem duas vantagens principais. Primeira, isso reduz a quantidade de processamento de informação nos roteadores. Quando um datagrama é encaminhado, o roteador não precisa saber quando um datagrama é um fragmento. Segundo, isso permite que as rotas mudem dinamicamente. Se um roteador intermediário reúne os fragmentos, todos os fragmentos precisariam alcançar tal roteador. Adiado a reunião até o destino final, IP fica livre para transmitir alguns fragmentos do datagrama através de uma rota diferente da dos outros fragmentos.

14.7 Identificando um Datagrama

Lembre-se que o IP não garante a entrega. Sendo assim, fragmentos individuais podem ser perdidos ou chegar fora de ordem. Mais importante que isso, se a origem enviar múltiplos datagramas para um dado destino, os fragmentos dos datagramas podem chegar em ordem arbitrária.

Como o IP reúne fragmentos que chegam fora de ordem? O host de origem coloca um número identificador único no campo *Identificação* de cada datagrama que é enviado. Quando um roteador fragmenta um datagrama, esse roteador copia o número de identificação em cada fragmento. Um host receptor utiliza esse número de identificação e o endereço IP de origem em um fragmento que chega para determinar a qual datagrama ele pertence. Além disso, o campo *Offset do Fragmento* diz ao host receptor como ordenar os fragmentos dentro de um dado datagrama.

14.8 Perda de Fragmento

Lembre-se que o IP não garante a entrega dos datagramas – se uma rede física perde pacotes, um datagrama encapsulado ou um fragmento podem ser perdidos. Quando todos os fragmentos de um datagrama chegam, o datagrama pode ser reorganizado. Entretanto, um problema aparece quando um ou mais fragmentos do datagrama chegam, e alguns fragmentos

estão atrasados ou perdidos. Embora o datagrama não possa ser reunido, o host receptor deve salvar os fragmentos no caso de aqueles fragmentos que estão faltando estejam apenas atrasados.

O host receptor não pode segurar fragmentos arbitrariamente por muito tempo porque fragmentos ocupam espaço em sua memória. Para evitar o esgotamento da memória, o IP especifica um tempo máximo para segurar os fragmentos. Quando o primeiro fragmento de um dado datagrama chega, o host receptor começa um contador. Se todos os fragmentos, de um mesmo datagrama, chegarem antes do contador expirar, o host receptor cancela o contador e reúne o datagrama. Entretanto, se o contador expirar antes de todos os fragmentos chegarem, o host receptor descarta aqueles fragmentos que já haviam chegado.

O resultado do contador de reunião IP é tudo-ou-nada: ou todos os fragmentos chegam e o IP reúne o datagrama, ou o IP descarta o datagrama inteiro. Em particular, não existe mecanismo para o host receptor dizer ao host de origem quais os fragmentos que chegaram. Isso faz sentido porque o host de origem não sabe sobre a fragmentação. Além disso, se o host de origem retransmitir o datagrama, as rotas podem ser diferentes, o que significa que a retransmissão não atravessa necessariamente os mesmos roteadores. Portanto, não há garantia que um datagrama retransmitido vá ser fragmentado da mesma maneira que o original.

14.9 Fragmentando um Fragmento

Depois de realizar uma fragmentação, um roteador encaminha cada fragmento para seu destino. O que acontece se um fragmento eventualmente alcance outra rede que tenha um MTU menor ainda? O esquema de fragmentação foi planejado cuidadosamente para tornar possível a posterior fragmentação de um fragmento. Outro roteador ao longo do caminho divide o fragmento em fragmentos menores. Em uma internet má projetada onde as redes são arranjadas em sequência decrescente de MTUs, cada roteador ao longo do caminho deve realizar uma posterior fragmentação de cada fragmento.

IP não distingue entre fragmentos originais e subfragmentos. Em particular, o host receptor não pode saber quando um fragmento foi resultado da fragmentação de um datagrama por um roteador ou de vários roteadores fragmentando fragmentos. A vantagem de fazer com que todos os fragmentos sejam iguais é que o host receptor realiza a reorganização do datagrama original sem primeiro reorganizar os subfragmentos. Fazendo assim economiza-se tempo de CPU, e reduz a quantidade de informação necessária nos cabeçalhos de cada fragmento.

Capítulo 15 – O IP do futuro (Ipv6)

15.1 Introdução

Em capítulos anteriores foi discutida a versão atual do Internet Protocol (IP). O capítulo 14 descreve como um datagrama IP é encapsulado em uma rede de pacotes e como é enviado através de uma rede física.

Este capítulo concentra no futuro do IP. O capítulo começa vendo as limitações e qualidades da atual versão do IP, e assim considera uma nova versão do IP que IETF propôs para substituir a atual versão.

15.2 O sucesso do IP

A versão atual do IP tem um sucesso considerável. Este sucesso foi possível por lidar com redes heterogêneas, mudanças drásticas na tecnologia de hardware, e um crescimento extremo em escala. Para lidar com redes heterogêneas, a norma IP define um pacote com formato uniforme (datagrama IP) e um mecanismo de transmissão de pacotes. O IP também define um conjunto de endereços, permitindo aplicações e camadas superiores de rede terem uma comunicação através de uma rede heterogênea, sem saber diferenças entre o endereçamento de hardware utilizado nas camadas inferior do sistema. O crescimento em escala pode ser notado com a expansão atual da Internet.

A versão atual também acomoda as mudanças nas tecnologias de hardware. O design original continua trabalhando bem pelas várias gerações de tecnologias de hardware criadas. O IP agora é usado em redes que operam em velocidade muito maior que das redes quando o mesmo foi projetado.

15.3 A motivação da mudança

O primeiro motivo para mudar tem como raiz o tamanho limitado no campo de endereços. Os projetistas decidiram usar 32 bits para o endereço IP porque fazendo isso garantiria que a Internet tenha mais que um milhão de endereços. Porém, com o crescimento exponencial da Internet global, onde se dobra o tamanho em menos um ano este tamanho não será suficiente. Assim, nessa taxa de crescimento, cada um dos endereços será preenchido, sem sobrar nenhum endereço livre, logo não haverá possibilidades de crescimento. Logo endereços maiores são necessários para acomodar o crescimento atual da Internet.

Mais motivações surge das novas aplicações criadas na Internet. Por exemplo, aplicações que entregam áudio e vídeo precisam entregar os dados em intervalos regulares. Para manter estas informações fluindo através da Internet sem interrupção, IP deve evitar mudar as rotas frequentemente. O cabeçalho do IP atual contém um campo que pode ser utilizado como requisição de um tipo de serviço, o protocolo não define um tipo de serviço que pode ser utilizado para entrega em tempo real de áudio e vídeo.

Novas aplicações estão sendo desenvolvidas e precisam de um endereçamento mais complexo e um roteamento mais capacitado. Assim, a nova versão do IP precisa incluir mecanismos que tornem este endereçamento e roteamento possíveis.

15.4 Nome e Número de Versão

Quando um protocolo específico é definido, o grupo que realizou o trabalho precisa distinguir este protocolo dos outros protocolos. Foi decidido utilizar um número de versão no cabeçalho no fim do protocolo normalizado. O versão atual do protocolo IP é a versão 4, contrariando a maioria dos pesquisadores, que esperavam que a próxima versão atual fosse a 5. Porém, a versão 5 já havia sido utilizada em um protocolo experimental chamado ST. Conseqüentemente, a nova versão do IP recebeu o número 6 como seu número de versão oficial, e o protocolo se tornou conhecido como IPv6.

15.5 Caracterização dos atributos no IPv6

O IPv6 contém vários dos atributos que tornaram IPv4 um sucesso. Como no IPv4, no IPv6 cada datagrama contém um endereço de destino e cada datagrama é encaminhado independentemente. Como no IPv4, no cabeçalho do datagrama temos o máximo número de pontos que o datagrama pode ser repassado até ser descartado.

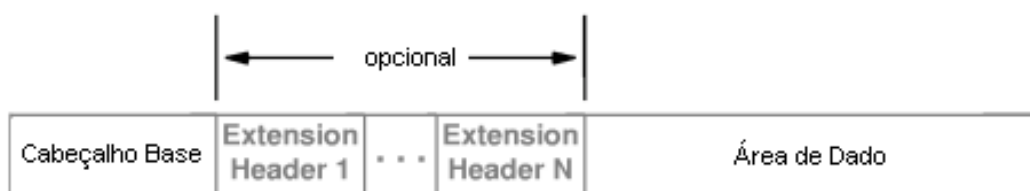
Mesmo mantendo todos os conceitos da versão anterior, o IPv6 modifica todos os detalhes. Por exemplo, o IPv6 utiliza um campo de endereçamento maior e um formato totalmente novo de cabeçalho. O IPv6 utiliza uma série de cabeçalhos de tamanho definidos ao invés de apenas um cabeçalho com a variável tamanho dentro de um campo.

Os novos atributos do IPv6 podem ser agrupados em cinco grupos principais:

- *Tamanho de Endereço.* Ao invés de 32 bits, cada campo de endereço do IPv6 contém 128 bits. Resultando em um campo de endereços maior que irá acomodar por décadas o crescimento da Internet.
- *Formato do Cabeçalho.* O datagrama IPv6 é completamente diferente que o cabeçalho do IPv4. Quase todos os campos no cabeçalho foram modificados ou substituídos.
- *Cabeçalhos de Extensão.* Diferente do IPv4, o qual usa um único formato de cabeçalho para todos datagramas, o IPv6 codifica a informação em cabeçalhos separados. O datagrama do IPv6 consiste em um cabeçalho base, seguido por zero ou mais cabeçalhos de extensão, seguidos por dados.
- *Suporte para áudio e vídeo.* O IPv6 inclui um mecanismo que permite o remetente e destinatário estabeleçam um caminho com alta qualidade através da rede e associar os datagramas a esse caminho. Embora o mecanismo tenha sido desenvolvido para aplicações de áudio e vídeo que necessitem de alta qualidade, o mecanismo também pode ser utilizado para associar os datagramas ao caminho com menor custo possível.
- *Protocolo Extensível.* Diferente do IPv4, o IPv6 não especifica todos os atributos possíveis no protocolo. Os projetistas desenvolveram um sistema que permita ao emissor adicionar informações ao datagrama. O esquema de extensão torna o IPv6 mais flexível que o IPv4, e significa que novas características podem ser adicionadas como o projetista necessitar.

15.6 Formato do Datagrama do IPv6

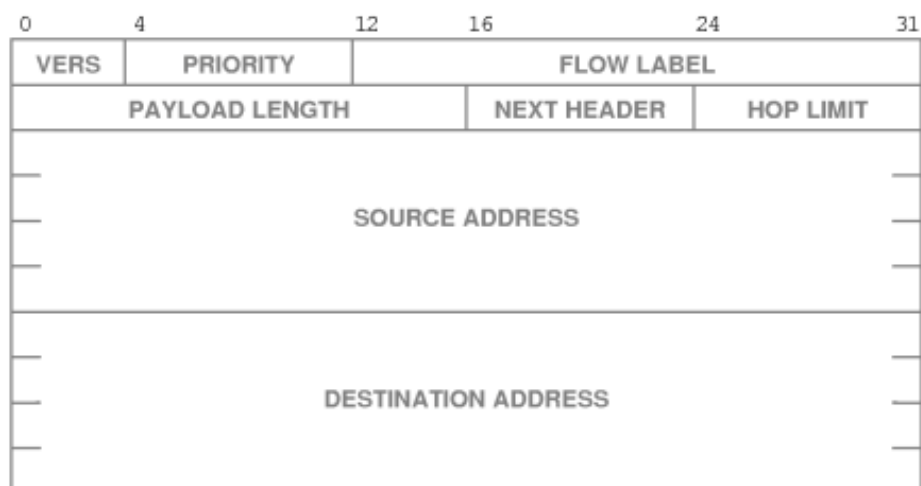
A figura abaixo ilustra um datagrama IPv6, que começa com um cabeçalho base, que é seguido por zero ou cabeçalhos estendidos, e dados.



Embora a figura ilustre a estrutura de um datagrama geral, os campos na figura não estão desenhados em escala. Em particular, algumas extensões são maiores que o cabeçalho base, enquanto outras são menores. Além do que, em muitos datagramas, o tamanho da área de dados é muito maior que os cabeçalhos.

15.7 Formato do Cabeçalho Base do IPv6

Embora este seja duas vezes maior que o cabeçalho do IPv4, o cabeçalho base do IPv6 contém uma quantidade menor de informações. A figura 20.2 ilustra o formato:

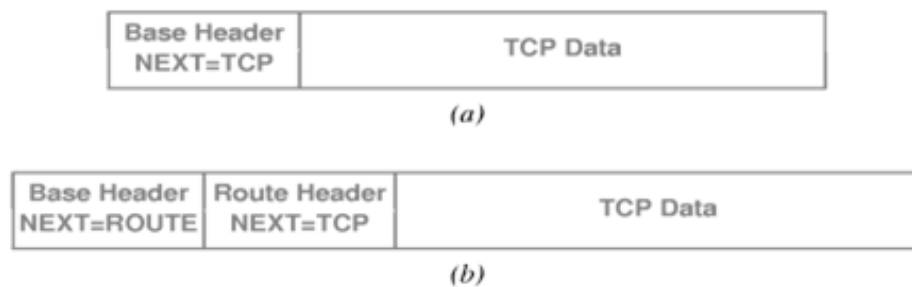


Como a figura ilustra, a maioria do espaço é preenchida com os dois campos de endereços que identificam o emissor e receptor. Como no IPv4, o campo SOURCE ADDRESS identifica o emissor, e o campo DESTINATION ADDRESS identifica o destinatário. Cada endereço ocupa dezesseis octetos, quatro vezes mais que no IPv4.

O campo VERS identifica o protocolo como versão 6. O campo PRIORITY especifica a classe de prioridade para roteamento. O campo PAYLOAD LENGTH especifica apenas o tamanho dos dados que está sendo transportado, o tamanho do cabeçalho não é incluído. O campo HOP LIMIT corresponde no IPv4 ao campo TIME-TO-LIVE. O IPv6 interpreta o HOP LIMIT estritamente, ou seja, o datagrama será eliminado se HOP LIMIT conte a zero antes que os dados cheguem ao destinatário.

O campo FLOW LABEL deveria ser utilizado com novas aplicações que necessitam de qualidade garantida. O campo então é utilizado para associar o datagrama a um caminho particular de roteamento. Para enviar em tempo real áudio e vídeo através da internet, o emissor requer da rede um caminho com delay menor que 100 milissegundos. Quando este caminho é determinado, o sistema de rede retorna um identificador que o emissor utiliza em cada datagrama que será enviado por esse caminho. Os roteadores utilizam o valor do campo FLOW LABEL para encaminhar o datagrama através do caminho pré-determinado.

O campo NEXT HEADER é utilizado para especificar qual tipo de informação vem após o cabeçalho atual. A figura abaixo ilustra o conceito:

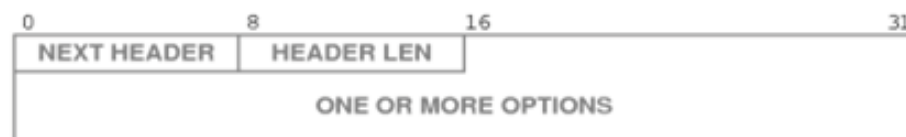


15.8 Como o IPv6 lida com múltiplos Cabeçalhos

Como a norma especifica um único valor para cada tipo de cabeçalho possível, nunca haverá ambigüidade na interpretação da informação no campo NEXT HEADER. O receptor usa o campo NEXT HEADER em cada cabeçalho para determinar o que se segue. Se o valor no campo corresponde ao tipo usado por dados, por exemplo, o receptor passa o datagrama para um modulo de software que lidara com dados.

Como o software do IPv6 sabe exatamente quando um cabeçalho começa ou termina? Alguns tipos de cabeçalhos têm o tamanho fixo. Por exemplo, o cabeçalho base tem o tamanho exato de 40 octetos. Para se mover para o item que segue o cabeçalho base, o software do IPv6 simplesmente soma 40 octetos no endereço do cabeçalho base.

Alguns cabeçalhos de extensão não têm um tamanho fixo. Nestes casos, o cabeçalho deverá conter informações suficientes para o IPv6 determinar onde termina o cabeçalho. Por exemplo, a figura 20.4 ilustra a forma geral de um cabeçalho IPv6 que tem a informação, similar ao IPv4.

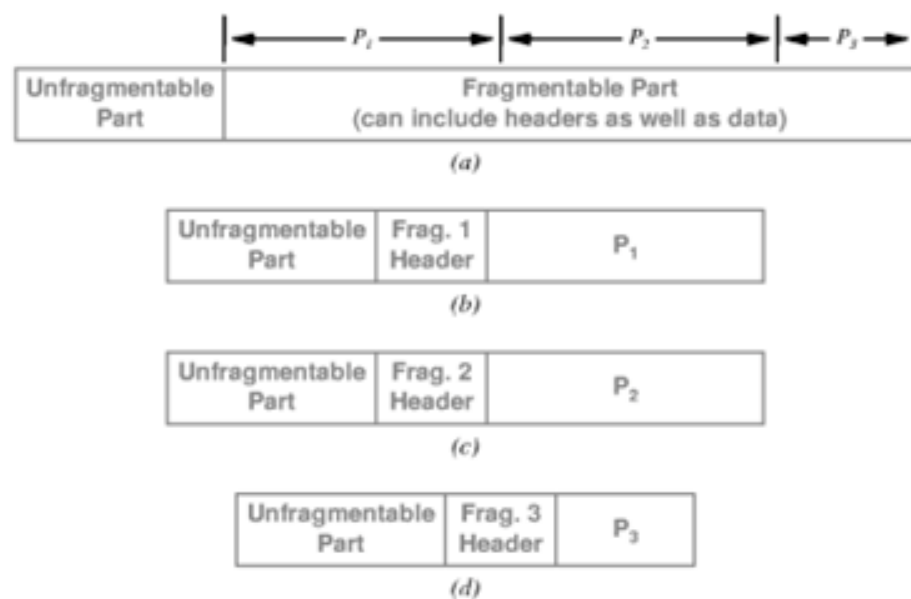


Quando um datagrama é composto, o emissor armazena o tamanho do cabeçalho de opções no campo HEADER LEN. Quando o receptor encontra um cabeçalho de opção, este usa

o campo HEADER LEN para determinar a localização exata do próximo item, e o campo NEXT HEADER determina o tipo.

15.9 Fragmentação e *path MTU*

Embora a fragmentação no IPv6 lembre a fragmentação no IPv4, os detalhes são diferentes. No IPv6 não são incluídos campos de informação de fragmentação no cabeçalho base. Ao invés disso, no IPv6 estes são colocados separadamente como cabeçalhos de extensão, com o nome de cabeçalhos de fragmentação. A presença do cabeçalho base identifica o datagrama como fragmentado. A figura 20.5 abaixo ilustra o conceito.



Como a figura mostra cada fragmento é menor que o datagrama original. Como no IPv4, o tamanho do fragmento é escolhido para ter o tamanho do transmissor máximo por unidade (MTU, *maximum transmission unit*) da rede onde o fragmento deverá ser enviado. Assim, o fragmento final pode ser menor que os outros porque representa o que sobrou após vários fragmentos com o tamanho MTU foram extraídos do datagrama original.

Além disso, no IPv6, o host emissor é responsável pela fragmentação. Isto é, os hosts deverão escolher o menor tamanho de datagrama que não irá necessitar de fragmentação; os roteadores pelo caminho que receberem um datagrama grande, não irão fragmentar o datagrama.

O host deverá aprender o MTU de cada rede no caminho até o destinatário, e escolher então o menor tamanho de datagrama. O MTU mínimo de um caminho entre emissor e receptor é conhecido como *path MTU*, e o processo de descoberta do *path MTU* é conhecido como *path MTU discovery*. Em geral, o processo *path MTU discovery* é iterativo. O host envia uma sequência com vários tamanhos de datagramas para o destinatário e verifica quais chegaram com erro. Uma vez que um datagrama é pequeno suficiente para passar sem ser fragmentado, o host escolhe o tamanho deste datagrama como o tamanho do *path MTU*.

15.10 O Propósito de Múltiplos Cabeçalhos

As duas principais razões para existir múltiplos cabeçalhos são: economia e capacidade de upgrades. Entender o porquê da economia é simples: dividindo as funcionalidades do datagrama em cabeçalhos separados economiza espaço. Ter cabeçalhos separados no IPv6 torna possível definir um conjunto de propriedades sem requerer em cada cabeçalho um espaço para cada propriedade. Como a maioria dos datagramas necessita de apenas alguns cabeçalhos, são evitados campos desnecessários, economizando um espaço considerável. Além do que, datagramas menores levam menos tempo para serem transmitidos. Assim, reduzindo o tamanho do datagrama também é reduzido o consumo da largura de banda (*bandwidth*).

Para entender a capacidade de upgrades, considere adicionar novas características ao protocolo. Um protocolo como IPv4 que utiliza um cabeçalho com formato fixo, assim para incluir mudanças o cabeçalho deveria ser projetado novamente para acomodar os campos necessários para a mudança. Já no IPv6, os cabeçalhos continuam inalterados. Apenas um novo tipo para o campo NEXT HEADER é definido bem como o formato deste novo cabeçalho.

A vantagem de propor uma nova funcionalidade para o cabeçalho também conta com a propriedade de se fazer o experimento antes de mudar todos os computadores na Internet. Por exemplo, suponha que os donos de dois computadores desejam testar uma nova técnica de criptografia em datagramas. Ambos devem concordar nos detalhes sobre o cabeçalho de criptografia experimental. O emissor adiciona este novo cabeçalho, e o receptor o interpreta. Se o cabeçalho aparecer após o cabeçalho utilizado para o roteamento, os roteadores na rede irão passar o datagrama sem entender o conteúdo deste cabeçalho. Assim que as propriedades do novo cabeçalho se mostrem interessantes, estas podem ser incorporadas à norma.

15.11 Endereçamento no IPv6

Como no IPv4, no IPv6 é destinado um único endereço para cada conexão entre o computador e a rede física. Assim, se um computador se conecta a três redes físicas, o computador recebe três endereços. No IPv6 cada endereço é separado em um prefixo que identifica a rede e um sufixo que identifica o computador em particular na rede.

Mesmo adotando a mesma aproximação para os endereços, o endereçamento no IPv6 se distingue e muito do endereçamento no IPv4. Primeiro todos os detalhes do endereço são diferentes. Em particular, os endereços não têm classes definidas. Ao invés disso, a fronteira entre o prefixo e sufixo pode estar em qualquer local com o endereço e não pode ser determinada a partir de um endereço sozinho. Assim, o tamanho do prefixo deverá ser associado a cada endereço permitindo ao software descobrir onde o prefixo acaba. Depois, o IPv6 define um conjunto de endereços especiais que difere completamente do conjunto de endereços especiais do IPv4. Em particular o IPv6 não inclui um endereço para *broadcasting* em uma determinada rede, ao invés disso cada endereço IPv6 se enquadra em um dos três tipos básicos abaixo:

- **Unicast:** O endereço corresponde a um único computador. O datagrama enviado para o mesmo é encaminhado pelo caminho mais curto até o computador.
- **Multicast:** O endereço corresponde a um conjunto de computadores, mesmo estando em várias localizações; a associação ao set pode ser alterada a qualquer momento. Quando um datagrama é enviado para o endereço, o IPv6 entrega uma cópia do datagrama para cada membro do set.
- **Anycast:** O endereço corresponde ao conjunto de computadores que compartilham o prefixo do endereço. O datagrama enviado para o endereço é

encaminhado pelo caminho mais curto e entregue exatamente a um único computador (computador mais próximo do emissor).

15.12 Notação Hexadecimal do IPv6

O endereço de 128 bits irá acomodar o crescimento da Internet, porém escrever este endereço pode se tornar difícil e complicado. Por exemplo, considere um número de 128 bits, escrito na notação decimal:

105.220.136.100.255.255.255.255.0.0.18.128.140.10.255.255

Para ajudar a reduzir o número de caracteres do endereço, os projetistas do IPv6 propuseram usar uma sintática compacta, conhecida como *colon hexadecimal notation (colon hex)*, na qual cada grupo de 16 bits é escrito em hexadecimal com dois pontos separando os grupos. Por exemplo, quando o número acima é escrito nessa notação temos:

69DC:8864:FFFF:FFFF:0:1280:8C0A:FFFF

Como o exemplo ilustra a notação *colon hex* requer uma menor quantidade de caracteres para expressar o endereço. Uma otimização adicional conhecida como *zero compression* também reduz o tamanho. A compressão de zero substitui a sequência de zeros por 2 dois pontos. Vejamos o exemplo abaixo:

FF0C:0:0:0:0:0:B1

Pode ser escrito como:

FF0C::B1

Como o endereço IPv6 tem maior quantidade de espaços, a compressão do zero é uma ferramenta essencial, pois muitos projetistas esperam o endereço IPv6 com um conjunto de zeros. Em particular isso ocorre porque os projetistas mapearam todos os endereços IPv4 existentes e os transformaram em IPv6, ou seja, qualquer endereço IPv6 que comece com 96 zeros, contém o endereço IPv4 nos últimos 32 bits.

Capítulo 16 – Capítulo 21 – Mecanismo de informação de erro (ICMP)

16.1 Introdução

Nos capítulos anteriores foi descrita o serviço de entrega de datagramas sem conectividade que o Protocolo de Internet provê. Eles definiram o formato do datagrama IP, explicaram como as tabelas de roteamento são utilizadas para selecionar o próximo salto para o qual o datagrama será encaminhado, e mostrou como os datagramas são encapsulados para transmissão. Este capítulo examina um protocolo de informação de erro integrado com o IP. Examinará os erros básicos que podem ser informados, e explica como e onde essas mensagens são enviadas.

Embora a pretensão original fosse prover um modo de a origem entender o porquê de datagramas não serem entregues, pesquisadores encontraram maneiras criativas de utilizar o sistema de mensagens de controle. Em particular, ferramentas foram criadas para colher informação sobre a internet através do envio de datagramas que gerassem mensagens de erro. O capítulo examina algumas ferramentas e técnicas que utilizam mensagens de erro para recolher informação.

16.2 Semânticas “A Qualquer Custo” e Detecção de Erro

Dizemos que o IP define uma comunicação a qualquer custo onde cada datagrama pode ser perdido, duplicado, atrasado, ou entregue fora de ordem. Isso pode parecer que um serviço a qualquer custo não precise de qualquer detecção de erro. Entretanto, é importante perceber que o serviço a qualquer custo não é negligente – IP se esforça para evitar erros e para reportar tais problemas quando ocorrem.

Já vimos um exemplo de detecção de erro no IP: o checksum do cabeçalho que é utilizado para detectar erros de transmissão. Quando um host cria um datagrama IP, ele inclui um checksum de todo o cabeçalho. Quando um datagrama é recebido, o checksum é verificado para assegurar que o cabeçalho chegou intacto. Para verificar o checksum, o host receptor recalcula o checksum incluindo o valor do campo *Checksum do Cabeçalho*. Se um bit do cabeçalho IP foi danificado durante a transmissão através da rede física, o receptor irá descobrir que o checksum não resultará em zero. Depois de modificar os campos do cabeçalho (por exemplo, decrementa o campo *Tempo de Vida*), um roteador deve recalculer o checksum antes de encaminhar o datagrama para o próximo salto.

A ação tomada em resposta a um erro de checksum é direta: o datagrama deve ser descartado imediatamente sem nenhum processamento posterior. O host receptor não pode confiar em nenhum dos campos do cabeçalho do datagrama porque o receptor não sabe quais bits foram alterados. Em particular, o host receptor não pode enviar uma mensagem de erro de volta para o computador que o enviou porque o receptor não pode confirmar no endereço de origem do cabeçalho. Da mesma forma, o host receptor não pode encaminhar um datagrama danificado porque o receptor não pode confirmar no endereço de destino do cabeçalho. Por isso, o host receptor não tem outra opção se não descartar o datagrama danificado.

16.3 Protocolo de Mensagem de Controle de Erro

Problemas menos sérios do que os erros de transmissão resultam em condições de erro que podem ser relatadas. Por exemplo, suponha que alguns caminhos físicos da internet falhem, fazendo com que a internet seja particionada em duas redes distintas sem nenhum caminho que

as interligue. Um datagrama enviado por um host de uma das redes para um host na outra rede não poderá ser entregue.

A pilha TCP/IP inclui um protocolo que o IP utiliza para enviar mensagens de erro quando condições, como a descrita acima, ocorrem: o *Protocolo de Mensagens de Controle da Internet* (ICMP). O protocolo é requerido em uma implementação padrão do IP. Veremos que os dois protocolos são co-dependentes. O IP utiliza o ICMP quando envia mensagens de erro, e o ICMP utiliza o IP para transportar essas mensagens.

A lista abaixo mostra as mensagens do ICMP, que incluem o erro e a mensagem de informação. Após analisar alguns exemplos, veremos como elas podem ser utilizadas.

Tipo	Nome
0	Echo Reply
1	Não atribuído
2	Não atribuído
3	Destination Unreachable
4	Source Quench
5	Redirect
6	Alternate Host Address
7	Não atribuído
8	Echo
9	Router Advertising
10	Router Selection
11	Time Exceeded
12	Parameter Problem
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
19	Reservado (para Segurança)
20-29	Reservado (para Experimentos de Robustez)
30	Traceroute
31	Datagram Conversion Error
32	Mobile Host Redirect
33	IPv6 Where-Are-You
34	IPv6 I-Am-Here
35	Mobile Registration Request
36	Mobile Registration Reply
37-255	Reservado

Exemplos de erros que incluem mensagens ICMP:

- *Source Quench*: Um roteador envia uma mensagem *Source Quench* sempre que ele recebe tantos datagramas que o seu espaço disponível de buffer se esgotar. Um roteador que tenha temporariamente acabado seu espaço de buffer deve descartar os datagramas que chegam. Quando ele descarta um datagrama, o roteador envia uma mensagem *Source Quench* ao host que criou tal datagrama. Quando um host recebe uma mensagem *Source Quench*, o host é obrigado a reduzir a taxa na qual ele está transmitindo.
- *Time Exceeded*: Uma mensagem *Time Exceeded* é enviada em dois casos. Sempre que um roteador reduz o campo *Tempo de Vida* de um datagrama a zero, o roteador descarta o datagrama e envia uma mensagem *Time Exceeded*. Fora isso, uma mensagem *Time Exceeded* é enviada a um host se o tempo de reunião dos fragmentos expirar antes que todos os fragmentos de um dado datagrama cheguem.

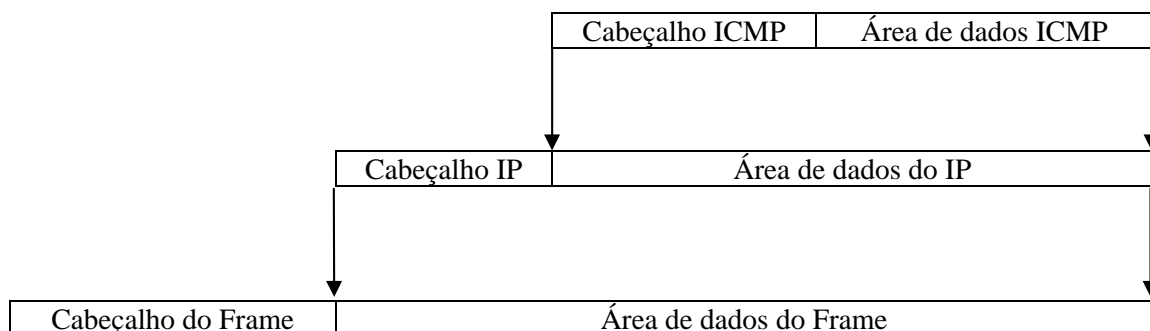
- *Destination Uncheachable*: Sempre que um roteador determina que um datagrama não possa ser entregue a seu destino final, ele envia uma mensagem *Destination Uncheachable* ao host que criou tal datagrama. A mensagem especifica tanto quando o host final de destino está inalcançável ou se a rede onde ele se encontra está inalcançável. Em outras palavras, a mensagem de erro distingue entre a situação em que a rede inteira está desconectada da internet (por exemplo, devido a um roteador que falhou) e no caso de apenas um host em particular estar desconectado (por exemplo, devido a um host estar desligado).
- *Redirect*: Quando um host cria um datagrama destinado a uma rede remota, o host envia o datagrama a um roteador, que encaminha o datagrama ao seu destino. Se o roteador determinar que o host tenha enviado o datagrama incorretamente, pois deveria ser enviado para um roteador diferente, o roteador envia uma mensagem *Redirect*, fazendo com que o host mude sua rota. Uma mensagem *Redirect* pode especificar ainda a mudança para um host específico ou para uma rede; a última opção é a mais comum.
- *Parameter Problem*: Um dos parâmetros especificado no datagrama está incorreto.

Fora as mensagens de erro, o ICMP define mensagens informativas, como:

- *Echo Request/Reply*: Uma mensagem de *Echo Request* pode ser enviada ao software ICMP de qualquer computador. Em resposta a essa mensagem *Echo Request*, o software ICMP é obrigado a enviar uma mensagem ICMP *Echo Reply*. A resposta carrega os mesmos dados que o requerimento.
- *Address Mask Request/Reply*: Um host transmite a todos os host de sua rede uma mensagem *Address Mask Request* quando é inicializado, e os roteadores que recebem o requerimento enviam uma mensagem *Address Mask Reply* que contém a máscara de sub-rede de 32 bits que está sendo utilizada por aquela rede.

16.4 Transporte da mensagem ICMP

O ICMP utiliza o IP para transportar cada mensagem de erro. Quando um roteador tem uma mensagem ICMP para enviar, ele cria um datagrama e encapsula a mensagem ICMP dentro do datagrama. Isso quer dizer, a mensagem ICMP é colocada na área de dados do datagrama IP. O datagrama é então encaminhado normalmente, com todo o datagrama encapsulado em um frame para a transmissão. A figura abaixo ilustra os dois níveis de encapsulamento.



Para onde uma mensagem ICMP deve ser enviada? Mensagens ICMP são sempre criadas em resposta a um datagrama. Ou quando o datagrama encontra algum problema (por exemplo, um roteador acha que o destino especificado no datagrama está inalcançável) ou quando o datagrama carrega uma solicitação ICMP para a qual um roteador cria uma resposta. Em ambos os casos, o roteador manda uma mensagem ICMP de volta à origem do datagrama.

Enviar a mensagem de volta a origem é direto porque cada datagrama carrega o endereço IP de sua origem no cabeçalho. O roteador extrai o endereço da origem do cabeçalho do datagrama que chegou e coloca esse endereço no campo *Destino* do cabeçalho do datagrama que carrega a mensagem ICMP.

Datagramas que carregam mensagens ICMP não tem prioridade especial alguma – eles são encaminhados como qualquer outro datagrama, com uma pequena exceção. Se o datagrama que carrega uma mensagem de erro ICMP causar um erro, nenhuma mensagem de erro é enviada. A razão é clara: os projetistas queriam evitar que uma internet se tornasse congestionada carregando mensagens de erro sobre mensagens de erro.

16.5 Utilizando as mensagens ICMP para testar o alcance

O capítulo 2 descreve o programa de *ping* que testa se um dado destino pode ser alcançado. Agora que já vimos o ICMP, podemos entender exatamente como o ping opera. Ping utiliza as mensagens ICMP *Echo Request/Reply*. Quando solicitado, o ping envia um datagrama IP que contém uma mensagem *Echo Request* para um destino específico. Depois de enviar a solicitação, ele aguarda por uma resposta durante um período pequeno de tempo. Se a resposta não chegar, o ping retransmite a solicitação. Se a resposta não chegar para as retransmissões (ou se uma mensagem ICMP *Unreachable Destination* chegar), ping declara que a máquina remota não está alcançável.

O software ICMP em uma máquina remota responde à mensagem *Echo Request*. De acordo com o protocolo, sempre que uma *Echo Request* chega, o software ICMP deve mandar uma *Echo Reply*.

16.6 Utilizando ICMP para traçar uma Rota

Dissemos que o campo *Tempo de Vida* no cabeçalho de uma datagrama é utilizado para prevenir erros de roteamento. Para evitar que um datagrama entre em um loop infinito, cada roteador deve decrementar o contador *Tempo de Vida* no cabeçalho. Se o contador chegar a zero, o roteador o descarta e envia uma mensagem ICMP *Time Exceeded* de volta a sua origem.

Mensagens ICMP são utilizadas pela ferramenta *Traceroute* descrita no capítulo 2 quando ela constrói uma lista de todos os roteadores ao longo do caminho até um dado destino. *Traceroute* determina o valor do *Tempo de Vida* do primeiro datagrama igual a 1 antes de enviá-lo. O primeiro roteador que recebe o datagrama decrementa o tempo de vida, descarta o datagrama, e envia de volta uma mensagem ICMP *Time Exceeded*. Devido à mensagem ICMP viajar em um datagrama IP, o *Traceroute* pode extrair o endereço IP de origem e dizer o endereço do primeiro roteador ao longo do caminho até o destino.

Depois de descobrir o endereço do primeiro roteador, o *Traceroute* envia um datagrama com o *Tempo de Vida* igual a 2. O primeiro roteador decrementa o contador e encaminha o datagrama: o segundo roteador descarta o datagrama e envia uma mensagem de erro. Similarmente, uma vez que ele tenha recebido a mensagem de erro do roteador que está a uma distância 2, o *Traceroute* envia outro datagrama com *Tempo de Vida* igual a 3, depois 4 e assim por diante.

O *Traceroute* deve ainda lidar com muitos detalhes. Devido ao IP utilizar a entrega a qualquer custo, datagramas podem ser perdidos, duplicados ou entregues fora de ordem. Devido a isso, o *Traceroute* deve estar preparado para receber respostas duplicadas e para retransmitir datagramas que foram perdidos. Escolher um tempo de espera pode ser difícil porque o *Traceroute* não pode saber quanto tempo deve esperar por uma resposta – o *Traceroute* permite que o usuário decida esse tempo.

O *Traceroute* enfrenta outro problema: rotas podem mudar dinamicamente. Se as rotas mudam entre duas probabilidades, a segunda probabilidade pode ter um caminho maior ou menor que o caminho da primeira. Mais importante que isso, a sequência de roteadores que o *Traceroute* encontrou pode não corresponder a um caminho válido através da internet. Por isso, o *Traceroute* é mais útil em uma internet onde as rotas sejam relativamente estáveis.

O *Traceroute* também precisa lidar com a situação onde o *Tempo de Vida* é grande o bastante para alcançar o seu destino. Para determinar quando um datagrama teve sucesso em atingir seu destino, o *Traceroute* envia um datagrama para o qual o host de destino irá responder. Embora ele pudesse enviar uma mensagem ICMP *Echo Request*, ele não o faz. Ao

invés disso, o Traceroute utiliza a mensagem ICMP número 30, ou usa o UDP (*User Datagram Protocol*), um protocolo que permite programas de aplicação enviar e receber mensagens individuais. Quando usa o UDP, o Traceroute envia um datagrama UDP para um programa não existente na máquina de destino. Quando essa mensagem chega para um programa não existente, o ICMP envia uma mensagem *Destination Unreachable*. Por isso, cada vez que o Traceroute envia um datagrama, ou ele recebe uma mensagem *Time Exceeded* de um roteador ao longo do caminho ou uma mensagem ICMP *Destination Unreachable* do computador de destino.

16.7 Utilizando ICMP para descobrir o MTU do Caminho

Em um roteador, o IP fragmenta qualquer datagrama que é maior que a MTU da rede pela qual o mesmo está sendo transmitido. Embora a fragmentação solucione o problema de redes heterogêneas, a fragmentação freqüentemente reduz o desempenho. Um roteador utiliza memória e tempo de processamento para construir fragmentos. Similarmente, o host de destino utiliza memória e tempo de processamento para coletar os fragmentos e reuni-los em um datagrama completo. Em algumas aplicações, a fragmentação pode ser evitada se o host de origem escolher um tamanho de datagrama menor. Por exemplo, uma aplicação de transferência de arquivos pode enviar qualquer quantidade de dados por datagrama. Se a aplicação escolhesse um tamanho de datagrama menor que ou igual ao menor MTU das redes ao longo do caminho até o destino, nenhum roteador precisaria fragmentar o datagrama.

Tecnicamente, o menor MTU ao longo do caminho entre a fonte e o destino é conhecido como *MTU do caminho*. É claro que, se as rotas mudam (por exemplo, o caminho muda), o MTU do caminho pode mudar também. Entretanto, em muitas partes da internet, as rotas tendem a permanecer estáveis por dias ou semanas. Nesses casos, faz sentido um computador descobrir o MTU do caminho, e criar datagramas suficientemente pequenos.

Qual o mecanismo um host pode utilizar para determinar o MTU do caminho? A resposta está em uma mensagem de erro ICMP e a causa que causará o envio da mensagem de erro. A mensagem de erro consiste em uma mensagem ICMP que relata que a fragmentação foi necessária, mas não permitida, e a técnica para fazer isso é um bit do campo *Flags* no cabeçalho IP que especifica que o datagrama não deve ser fragmentado. Quando um roteador determina que um datagrama deva ser fragmentado, o roteador examina tal bit no cabeçalho para verificar se a fragmentação é permitida. Se esse bit for igual a 1, o roteador não realiza a fragmentação. Ao invés disso, o roteador envia uma mensagem de erro ICMP de volta à fonte, e descarta o datagrama.

Para determinar o MTU do caminho, o IP no host envia uma seqüência de tentativas, onde cada tentativa consiste em um datagrama que tem o bit que previne a fragmentação igual a 1. Se o datagrama é maior que o MTU de uma rede ao longo do caminho, o roteador conectado a essa rede descartará o datagrama e envia a mensagem ICMP apropriada ao host. O host pode então enviar uma tentativa menor até que uma tenha sucesso. Como no Traceroute, o host deve estar preparado para retransmitir tentativas que não tenham tido resposta.

Capítulo 17 – TCP: Serviço de Transporte Confiável

17.1 Introdução

Nos capítulos anteriores foram descritos o serviço de entrega de pacotes provido pelo IP e os protocolos utilizados para reportar erros. Este capítulo examina o TCP, o principal protocolo utilizado acompanhando o IP, e explica como o protocolo fornece transporte confiável.

O TCP alcança aparentemente uma tarefa impossível: utiliza um serviço de datagramas não confiável oferecido pelo IP quando envia dados para outro computador, mas fornece uma entrega de dados confiável para aplicativos. O TCP devesse compensar a perda ou o delay na Internet para conceder uma transferência de dados eficiente, e isto deve ser feito sem sobrecarregar o meio de transmissão e roteadores.

17.2 A necessidade de um transporte confiável

Para permitir que os programadores sigam as técnicas convencionais quando criam aplicativos para utilizar a Internet, o software na Internet deve fornecer a mesma semântica que os sistemas computacionais convencionais. Isto é, o software deve garantir rapidez e transporte confiável. Os dados devem ser entregues exatamente na mesma ordem de envio e não deverá existir perda ou duplicação.

17.3 O Protocolo de Controle de Transmissão (TCP)

Confiança é a responsabilidade do protocolo de transmissão: aplicativos interagem com o serviço de transporte para enviar e receber dados. No conjunto TCP/IP, o Protocolo de Controle de Transmissão (TCP) concede o serviço de transporte confiável. O TCP é extraordinário porque resolve um problema difícil, e nenhum outro protocolo de propósito geral provou que funciona tão bem. Consequentemente, maioria dos aplicativos para Internet foram construídos para utilizar o TCP.

17.4 O Serviço que o TCP fornece aos Aplicativos

Do ponto de vista dos aplicativos, o TCP fornece sete funções principais:

- *Conexão Orientada.* TCP fornece serviço de conexão orientada, ou seja, um aplicativo primeiro solicita a conexão com o destinatário, e então utiliza a conexão para transferir dados.
- *Comunicação Ponto a Ponto.* Cada conexão TCP tem exatamente dois pontos finais.
- *Confiança Completa.* TCP garante que os dados enviados através da conexão serão entregues exatamente como foram enviados, sem perder nenhum dado ou mudança de ordem.

- *Comunicação Full Duplex.* A conexão TCP permite enviar dados em ambas as direções e permite cada programa aplicativo enviar dado a qualquer momento. O TCP pode armazenar dado que sai e entra em ambas as direções, tornando possível para um aplicativo enviar dados e continuar a computação enquanto os dados são transferidos.
- *Interface Corrente.* O TCP fornece uma interface corrente na qual o aplicativo envia uma seqüência continua de octetos através da conexão. Isto é, TCP não provém à noção de registros, e não garante que os dados serão entregues ao aplicativo destinatário nos mesmos pedaços que foram transferidos do aplicativo emissor.
- *Inicialização de Conexão Segura.* TCP requer que quando dois aplicativos criam uma conexão, ambos devem concordar com a nova conexão.
- *Ótimo fim de Conexão.* Um programa pode abrir uma conexão, enviar dados quando necessitar, e então requisitar que a conexão seja finalizada. O TCP garante entregar todos os dados antes de fechar a conexão.

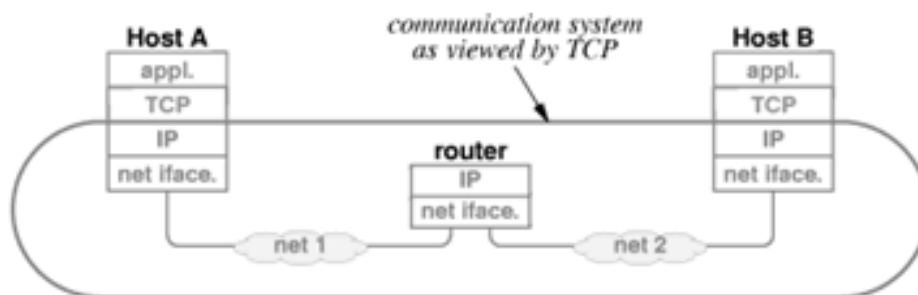
17.5 Serviço Ponto a Ponto e Datagramas

O TCP é chamado de Protocolo de Ponto a Ponto (End-To-End Protocol) porque fornece conexão direta de um aplicativo em um computador para um aplicativo em um computador remoto. Os aplicativos podem requisitar que o TCP crie uma conexão, envie e recebam dados, e depois feche a conexão.

A conexão feita pelo TCP é chamada de *conexão virtual* porque são executadas em software. Realmente, o sistema base de internet não fornece suporte de hardware ou software a conexões. Em vez disso, o TCP modula nas duas máquinas uma troca de mensagens para alcançar a ilusão de uma conexão.

O TCP utiliza o IP para transmitir mensagens. Cada mensagem TCP é encapsulada em datagrama IP e enviada na rede. Quando o datagrama chega ao host destinatário o IP passa o conteúdo ao TCP. Note que embora o TCP utilize o IP para transmitir as mensagens, o IP não lê ou interpreta o conteúdo da mensagem. Assim o TCP trata o IP como um sistema de comunicação que conecta dois pontos finais de uma conexão, e o IP trata cada mensagem TCP como dados a serem transferidos.

A figura 22.1 abaixo contém um exemplo de conexão entre dois hosts e um roteador ilustrando a relação entre TCP e IP.



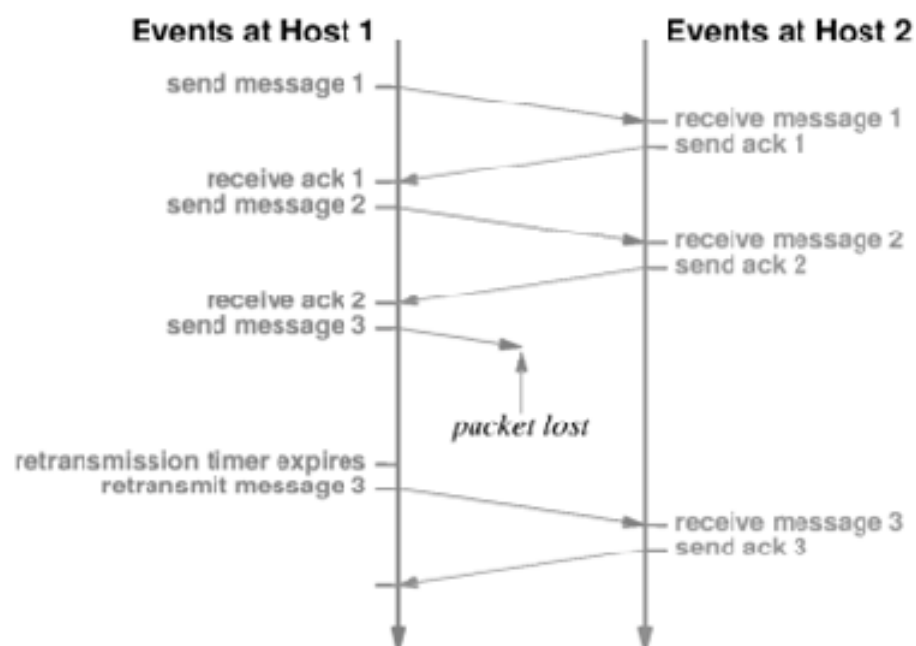
17.6 Alcançando Confiança

Um protocolo de transporte como TCP deve ser projetado cuidadosamente para alcançar confiança. Os principais problemas são: Entrega não confiável devido ao sistema básico de comunicação e boot de computador. Considere uma situação na quais dois computadores formam uma conexão, transferem dados e fecham a conexão. Após isso, abrem uma nova conexão. Como cada mensagem pode ser perdida, duplicada, atrasada ou entregue fora de ordem, mensagens da primeira conexão podem ser duplicadas e a copia entregue tarde suficiente para a segunda conexão já estar formada. Mensagens não podem ser ambíguas, ou o protocolo irá aceitar as mensagens duplicadas da conexão antiga e permitir que interfiram na nova conexão.

Quando o sistema de um computador é reiniciado, este é um serio desafio para o protocolo TCP. Imagine a situação onde dois aplicativos estabelecem a conexão e então um computador reinicia. Embora o software de protocolo no computador que reinicia tenha conhecimento da conexão, o software de protocolo no computador que não reiniciou considera a conexão valida. Mais importante, pacotes duplicados que estão atrasados se tornam um desafio porque o protocolo tem que estar hábil para rejeitar os pacotes de antes de o computador reiniciar.

17.7 Perca de Pacotes e Retransmissão

Uma das técnicas mais importantes do TCP é a retransmissão. Quando o TCP envia algum dado, o emissor compensa a perda de pacotes utilizando o esquema de retransmissão. Ambos os lados de uma comunicação participam. Quando o TCP recebe dado, este envia uma mensagem de acknowledgment (reconhecimento) ao emissor. No momento em que o emissor envia um dado, o TCP inicia um timer. Se o timer expirar antes da mensagem de acknowledgment chegar, o emissor retransmite os dados. A figura 22.2 abaixo ilustra a retransmissão.



A retransmissão no TCP é a chave para o sucesso do mesmo, pois lida com comunicações através de redes arbitrárias e permite múltiplos aplicativos comunicarem ao mesmo tempo. A questão é: quanto tempo o TCP deveria esperar antes de retransmitir? Mensagens de acknowledgment de um computador em uma rede local são esperadas em poucos milissegundos. Esperar muito tempo por uma mensagem de acknowledgment deixa a rede ociosa e não maximiza a máxima taxa de transferência. Assim, em uma rede local o TCP não deverá esperar muito tempo antes de retransmitir. Porém, retransmitir após poucos milissegundos não funciona bem em uma conexão de longa distância a satélite, porque o tráfego desnecessário consumiria toda largura de banda e diminuiria o ritmo de transferências.

O TCP encontra mais um desafio ao ter que distinguir entre destinatários remotos ou locais: uma explosão de datagramas pode causar congestionamento, que causa delays em transmissões através do caminho. De fato, o tempo total necessário para enviar uma mensagem e receber o acknowledgment pode aumentar ou diminuir em magnitude de ordem de milissegundos.

17.8 Retransmissão Adaptativa

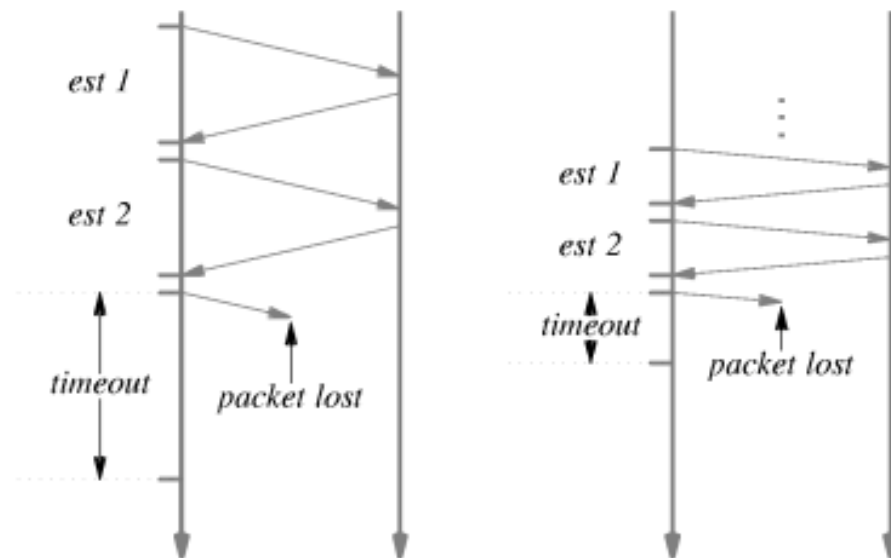
Antes de o Protocolo TCP ser inventado, os protocolos de transporte usavam um valor fixo de delay para retransmissão – o projetista do protocolo ou administrador da rede escolhia um valor que fosse grande o suficiente para o delay esperado. Projetistas que trabalhavam no TCP perceberam que um valor fixo de tempo não operaria bem para Internet. Logo, eles escolheram tornar a retransmissão no TCP adaptativa. Isto é, o TCP monitora o delay em cada conexão e adapta o timer de retransmissão para acomodar mudanças nas condições da rede.

O TCP estima o tempo de delay (round-trip delay) para cada conexão ativa medindo o tempo necessário para obter uma resposta. A toda hora este envia uma mensagem e espera a resposta da mesma. O TCP grava o tempo no qual a mensagem foi enviada e quando recebe a resposta, subtrai o tempo do envio, assim temos uma estimativa do delay da conexão. Ao mesmo tempo em que o TCP envia pacotes e recebe mensagens de acknowledgment, o TCP gera uma sequência de estimativas de tempo de delay e utiliza uma função estatística para gerar a média de tempos de sobre carregamento. Além do tempo de sobre carregamento, o TCP armazena uma estimativa da variação, e utiliza uma combinação linear para estimar a média e variação como o valor para retransmissão.

Usando a ajuda da variação o TCP reage rapidamente quando o delay aumenta devido a uma explosão de pacotes. Usando o tempo de sobre carregamento o TCP reinicia o tempo de retransmissão se o delay retorna a um valor menor após a explosão de pacotes. Quando o delay permanece constante, o TCP ajusta o tempo de saída de retransmissão para um valor um pouco maior que a média do tempo de resposta. Quando o delay começa a variar, o TCP ajusta o tempo de retransmissão para um valor maior que a média do recebimento de mensagens de acknowledgment.

17.9 Comparação entre Tempos de Retransmissão

Para entender como a retransmissão adaptativa ajuda o TCP maximizar o rendimento em cada conexão, considere o caso de perda de pacotes em duas conexões que tem tempos de resposta diferentes. A figura 22.3 abaixo ilustra o tráfego em cada conexão.

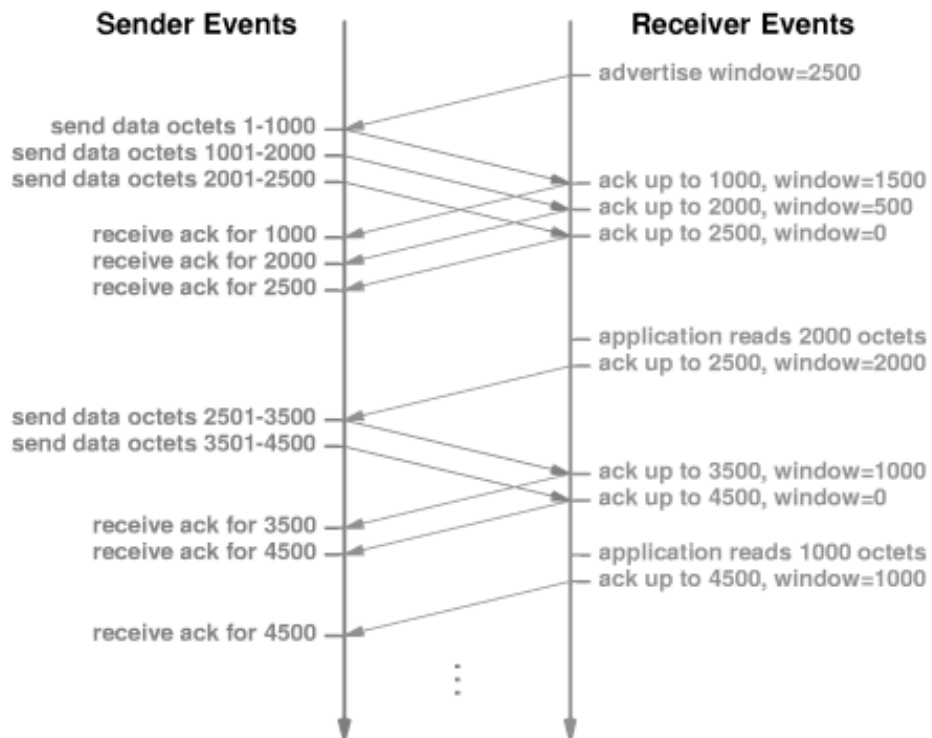


Como a figura mostra, o TCP configura o tempo de retransmissão para ser um pouco maior que a media dos tempos de resposta. Se o delay é grande, o TCP utiliza um tempo grande de retransmissão; se o delay é pequeno, o TCP utiliza um tempo pequeno de retransmissão. A meta é esperar o tempo suficiente para determinar se um pacote se perdeu ou não, sem esperar mais tempo do que o necessário.

17.10 Buffers e Controle de Fluxo

O TCP utiliza um mecanismo conhecido como *janela* (*window*) para controlar o fluxo de dados. Quando uma conexão é estabelecida, cada ponto da conexão aloca um buffer para armazenar os dados que chegam, e envia uma mensagem para a outra ponta dizendo qual o tamanho do buffer que a mesma alocou. Com a chegada dos dados, o receptor envia mensagem de acknowledgment, que também especifica o tamanho restante no buffer. O espaço total disponível no buffer é chamado de *janela* (*window*) e a notificação que especifica o tamanho é chamado de *aviso de janela* (*window advertisement*). O receptor envia o *aviso de janela* com cada acknowledgment.

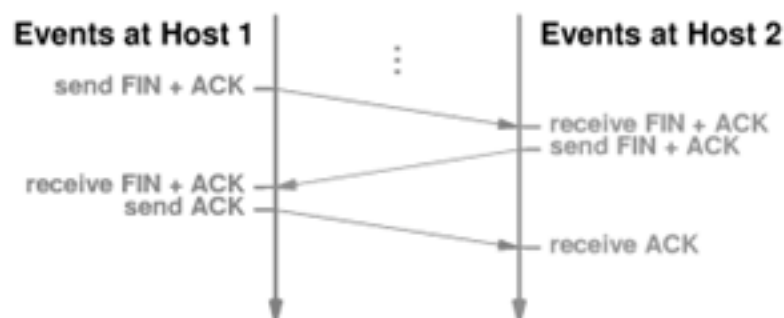
Se o aplicativo que recebe os dados pode ler os dados na mesma rapidez que chegam, o receptor irá enviar um aviso de janela positivo com cada acknowledgment. Porém, se o emissor operar mais rápido que o receptor, os dados que chegam acabarão completando todo espaço no buffer, e o receptor irá enviar uma mensagem de aviso conhecida como *zero window*. Quando o emissor recebe a mensagem de aviso *zero window*, o mesmo deve parar de enviar dados até que receba uma mensagem de janela positiva. A figura 22.4 abaixo ilustra os avisos de janela.



17.11 Three-Way Handshake

Para garantir que as conexões estejam estabelecidas ou realmente finalizadas, o TCP utiliza o *3-way handshake* no qual temos três mensagens trocadas. Cientistas provaram que a troca das três mensagens é necessária e suficiente para evitar comunicações ambíguas apesar de perda de pacotes, duplicação e delay.

O TCP utiliza o termo synchronization segment (SYN segment) para descrever mensagens do *three-way handshake* que criam a conexão, e o termo FIN segment (abreviação para finish) para descrever a mensagem do *three-way handshake* que finaliza uma conexão. A figura 22.5 abaixo ilustra o *three-way handshake* usado para finalizar uma conexão.



Uma parte do *three-way handshake* usado para criar uma conexão requer que cada ponto da conexão gere uma sequência randômica de 32 bits. Se um aplicativo tenta estabelecer uma nova conexão após o computador reiniciar, o TCP irá escolher uma nova sequência randômica de 32 bits. Como cada nova conexão recebe esta sequência randômica, um par de aplicativos que use o TCP pode fechar a conexão, estabelecer uma nova conexão e não terá interferência de pacotes duplicados ou com delay.

17.12 Controle de Congestionamento

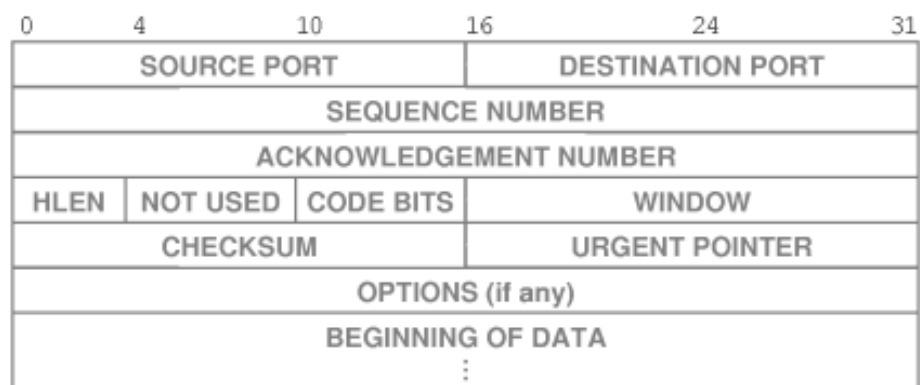
Na maioria das redes modernas, a perda de pacotes (ou delay extremo) tem com causa mais freqüente o congestionamento, ao invés das falhas de hardware. Os protocolos de transporte que retransmitem em excesso podem aumentar o problema de congestionamento ao adicionar cópias de mensagens na rede. Se o congestionamento aciona retransmissões, o sistema pode alcançar um estado de colapso de congestionamento, análogo a um engarrafamento em uma estrada. Para evitar o problema, o TCP utiliza a perda de pacotes como medida de congestionamento, e responde pela diminuição da taxa ao momento que retransmite dados.

Quando uma mensagem se perde, o TCP começa o controle de congestionamento. Ao invés de retransmitir dados suficientes para encher o buffer do receptor, o TCP começa enviando uma única mensagem contendo dados. Se o acknowledgement chega sem perdas, o TCP dobra a quantia de dados a serem enviados e envia duas mensagens adicionais. Se o acknowledgement de ambas as mensagens chegar, o TCP envia quatro mensagens, e assim em diante. O crescimento exponencial continua até TCP estiver enviando metade do tamanho da janela do receptor.

O controle de congestionamento do TCP responde bem ao aumento de tráfego na rede. Por recuar rapidamente, o TCP é hábil para aliviar o congestionamento. Mais importante, por causa disso evita adicionar retransmissões para congestionar a rede e o esquema de controle ajuda então na prevenção do colapso de congestionamento.

17.13 Formato do Segmento TCP

O TCP utiliza um único formato para todas as mensagens, incluindo mensagens de dados, acknowledgements, e mensagens do *three-way handshake* usadas para criar e terminar uma conexão. No TCP usamos o termo segmento para referir a uma mensagem. A figura 22.6 abaixo ilustra o formato do segmento.



Para entender o formato do segmento, é necessário lembrar que a conexão TCP tem dois pontos finais e dois fluxos de dados, um em cada direção. Se os aplicativos em cada ponto enviam dados simultaneamente, o TCP pode enviar um único segmento contendo o acknowledgement pros dados que chegam. O aviso de janela que especifica o tamanho adicional de buffer disponível para os dados que chegarão, e os dados que estão de saída. Assim, alguns campos no segmento referem-se aos dados que vão à direção direta, enquanto outros campos se referem aos dados que viajam na direção reversa.

Quando o computador envia um segmento, os campos *ACKNOWLEDGMENT NUMBER* e *WINDOW* se referem aos dados que chegam: o *ACKNOWLEDGMENT NUMBER* especifica a seqüência de números dos dados que são recebidos, e *WINDOW* especifica o quanto de espaço esta disponível no buffer para receber dados. O campo *SEQUENCE NUMBER* se refere aos dados que são enviados. Ele dá uma seqüência numérica para os dados que estão sendo transportados no segmento. O receptor utiliza a seqüência de números para reordenar os segmentos que chegam fora da ordem e para computar o número do acknowledgement. O campo *DESTINATION PORT* identifica qual aplicativo no computador receptor deverá receber os dados, enquanto o campo *SOURCE PORT* identifica o aplicativo que enviou os dados. Finalmente, o campo *CHECKSUM* contém a soma dos bits, abrangendo cabeçalho e dados.

Capítulo 18 – Interação Cliente-Servidor

18.1 Introdução

Esse capítulo começa uma nova seção do texto que foca as aplicações de rede. Seções anteriores forneceram um embasamento geral necessário para entender como aplicações utilizam as redes. Explicaram o hardware usado em LANs e WANs, mostraram como redes heterogêneas poderiam se interconectar, e examinaram os protocolos usados para tornar possível o transporte confiável através da Internet. A combinação de hardwares de rede e protocolos resulta em uma infra-estrutura de comunicação de propósito geral que permite que programas de aplicação em um par arbitrário de computadores possam se comunicar. Para todo o texto vamos assumir que tal infra-estrutura está no lugar. Em particular, assumiremos que o protocolo e a informação de roteamento necessária estão disponíveis em todos os computadores (por exemplo, todos os computadores estão conectados a redes que estejam funcionando).

Os capítulos dessa seção focam serviços de alto nível disponíveis em uma internet e softwares de aplicação que fornecem tais serviços. O texto explica princípios, técnicas, e a interface de programação utilizada para produzir as aplicações de rede assim como a estrutura do software. Muitos capítulos contêm exemplos de aplicações populares. Além da caracterização do serviço que a aplicação fornece, cada capítulo descreve a estrutura do software e mostra como a aplicação usa a comunicação de rede.

Esse capítulo apresenta o conceito fundamental que forma a base para todas as aplicações de rede: Interação Cliente-Servidor. O capítulo examina o modelo Cliente-Servidor básico e descreve como que a necessidade da interação Cliente-Servidor aparece como a maneira dos protocolos de rede operar. Capítulos futuros expandiram a discussão, mostrando como aplicações utilizam o paradigma Cliente-Servidor.

18.2 As Funcionalidades que o Software de Aplicação Fornece

Embora conexões de redes físicas e protocolos de comunicação sejam obrigatórios para a comunicação através de uma internet, a funcionalidade mais interessante e útil é provida pelo software de aplicação. Aplicações fornecem os serviços de alto nível os quais são utilizados pelo usuário, e determinam como usuários enxergam as capacidades da internet. Por exemplo, softwares de aplicação tornam possível a troca de mensagens eletrônicas, a visualização de arquivos de informação, ou a transferência de uma cópia de um arquivo entre computadores.

Aplicações determinam o formato no qual a informação será mostrada e os mecanismos que os usuários têm para selecionar ou acessar as informações. Mais importante ainda, é que as aplicações definem nomes simbólicos usados para identificar tanto fontes físicas como abstratas disponíveis na rede. Por exemplo, softwares de aplicação definem nomes para computadores e dispositivos I/O como as impressoras assim como definem nomes para itens abstratos como arquivos, caixas de entrada eletrônicas e bancos de dados. Tais nomes simbólicos possibilitam aos usuários especificar ou encontrar informações ou serviços sem ter de entender ou lembrar os endereços de baixo nível usados pelo protocolo base. De fato, a maioria dos usuários da Internet acessa remotamente computadores pelo nome – um usuário nunca precisa saber o endereço IP de um computador. Similarmente, um usuário pode acessar um serviço pelo nome sem ter de conhecer o número interno utilizado pelo protocolo para identificá-lo. Softwares de aplicação lida com os detalhes da tradução dos nomes simbólicos para os valores numéricos equivalentes automaticamente.

18.3 As Funcionalidades Providas pela Internet

Internets provêem uma infra-estrutura geral de comunicação sem especificar quais serviços oferecerá, quais computadores irão rodar tais serviços, como a disponibilidade dos serviços se tornará conhecida, ou como os serviços serão utilizados – essas questões são deixadas à aplicação e aos usuários. De fato, uma internet é muito parecida com um sistema telefônico. Mesmo que ela provenha a habilidade de se comunicar, a internet não especifica como os computadores interagem ou o que tais computadores fazem com o serviço de comunicação.

Uma internet é como um sistema telefônico em outro aspecto signficante – protocolos não sabem quando iniciar um contato, ou quando aceitar uma tentativa de comunicação de um computador remoto. Ao invés disso, como no serviço telefônico, a comunicação através de uma internet requer um par de programas para auxiliar. Uma aplicação em um dos computadores tenta se comunicar com uma aplicação em outro (análogo de fazer uma ligação telefônica), e uma aplicação no outro computador responde ao pedido (análogo a atender a uma ligação telefônica).

18.4 Fazendo Contato

Apesar das similaridades entre a comunicação via internet e ligações telefônicas, tem uma diferença importante entre o modo como duas aplicações usam a internet e o modo como duas pessoas usam o sistema telefônico. A diferença é que o protocolo não possui um mecanismo análogo à campainha do telefone – não existe um modo do protocolo informar a aplicação de que uma tentativa de comunicação tenha chegado e também não há modo da aplicação concordar em aceitar mensagens arbitrárias.

Se não é dado sinal algum, como uma aplicação pode saber quando uma tentativa de comunicação chegou? A resposta está em um método de interação que difere do modelo telefônico. Ao invés de esperar por uma mensagem arbitrária chegar, uma aplicação que espera comunicação deve interagir com o protocolo antes que uma fonte externa tente se comunicar. A aplicação informa ao protocolo local que um tipo específico de mensagem é esperado, e então a aplicação espera. Quando uma mensagem que chega é exatamente compatível com a especificação da aplicação, o protocolo repassa a mensagem para aquela aplicação. É claro que ambas as aplicações envolvidas em uma comunicação não podem esperar pela chegada de uma mensagem – uma aplicação deve iniciar a interação, enquanto a outra espera passivamente.

18.5 O Paradigma Cliente-Servidor

O paradigma da disposição de um programa de aplicação para esperar passivamente por outra aplicação iniciar uma comunicação utiliza tanto processamento de computador distribuído que recebeu o nome: o *Paradigma da Interação Cliente-Servidor*.

Os termos *cliente* e *servidor* referem-se a duas aplicações envolvidas em uma comunicação. A aplicação que ativamente inicia o contato é chamada de *cliente*, enquanto que a aplicação que espera passivamente pelo contato é chamada de *servidor*.

18.6 Características de Clientes e de Servidores

Embora existam pequenas variações, a maior parte das interações Cliente-Servidor tem as mesmas características gerais. Em geral, um software cliente:

É um programa de aplicação arbitrário que se torna cliente temporariamente quando acesso remoto é necessário, mas também executa outros processos localmente.

- É iniciada diretamente por um usuário, e executa apenas uma seção.
- Roda localmente em um computador pessoal do usuário.
- Inicia efetivamente o contato com um servidor.

- Pode acessar quantos serviços forem necessários, mas contata efetivamente um servidor por vez.
- Não requer hardware especial ou sistema operacional sofisticado.

Já um software servidor:

- É um programa especialmente dedicado a prover um serviço, mas lidar com múltiplos clientes remotos ao mesmo tempo.
- É iniciado automaticamente quando um sistema inicia, e continua a ser executado por várias seções.
- Roda em um computador compartilhado (por exemplo, não roda em um computador pessoal de um usuário).
- Espera passivamente pelo contato de clientes remotos arbitrários.
- Aceita contato de clientes arbitrários, mas oferece um serviço.
- Requer hardware potente e de um sistema operacional sofisticado.

18.7 Programas de Servidores e Computadores da classe de Servidores

O termo *servidor* pode causar confusão algumas vezes. Formalmente, o termo se refere a um programa que espera passivamente por comunicação, e não do computador no qual está rodando. Entretanto, quando um computador é dedicado a rodar um ou mais programas servidores, o próprio computador é às vezes (erroneamente) chamado de *servidor*. Vendedores de hardware contribuem para essa confusão porque eles classificam computadores que tem CPUs rápidas, grandes memórias e sistemas operacionais poderosos como máquinas servidores.

Aderimos à terminologia científica correta e usamos o termo *servidor* para nos referir aos programas e não a computadores. O termo *computadores da classe de servidores* refere-se a computadores poderosos usados para rodar software servidor.

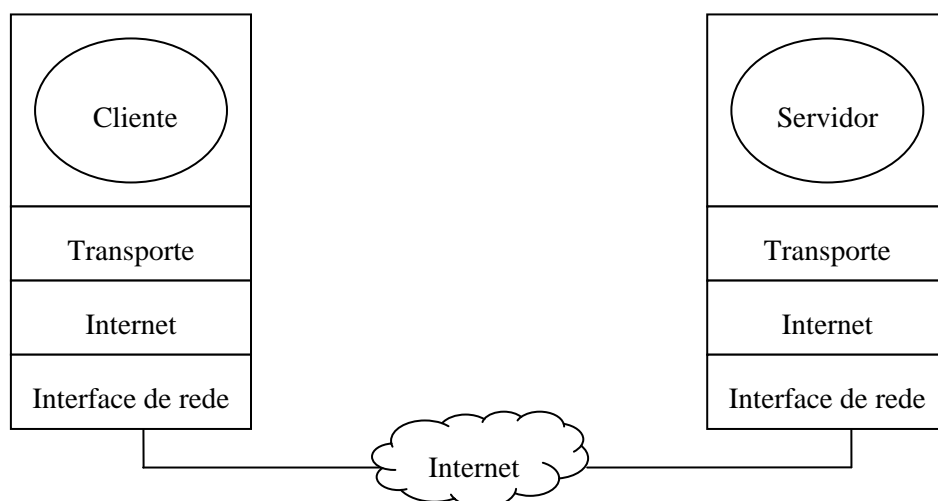
18.8 Pedidos, Respostas e Direção do Fluxo de Dados

A informação pode passar em uma ou ambas as direções entre um cliente e um servidor. Tipicamente, um cliente envia um pedido ao servidor e o servidor retorna uma resposta ao cliente. Em alguns casos, o cliente envia uma série de pedidos e o servidor emite uma série de respostas (por exemplo, um cliente de banco de dados pode permitir que um usuário procure por mais de um item por vez). Em outros casos, o servidor envia dados continuamente sem qualquer pedido – assim que um cliente contate tal servidor, ele começa a enviar os dados (por exemplo, um servidor de tempo pode enviar continuamente informações do tempo com a temperatura e pressão atmosférica atualizadas).

É importante entender que servidores podem aceitar informações que chegam assim como entregar informação. Por exemplo, a maioria dos servidores de arquivos é configurada para exportar um conjunto de arquivos aos clientes. Isto é, um cliente envia um pedido que contém o nome do arquivo e o servidor responde enviando uma cópia do arquivo. Mas um servidor de arquivos pode também ser configurado para importar arquivos (por exemplo, para permitir que um cliente envie a cópia de um arquivo, o qual o servidor aceita e guarda no disco).

18.9 Protocolos de Transporte e Interação Cliente-Servidor

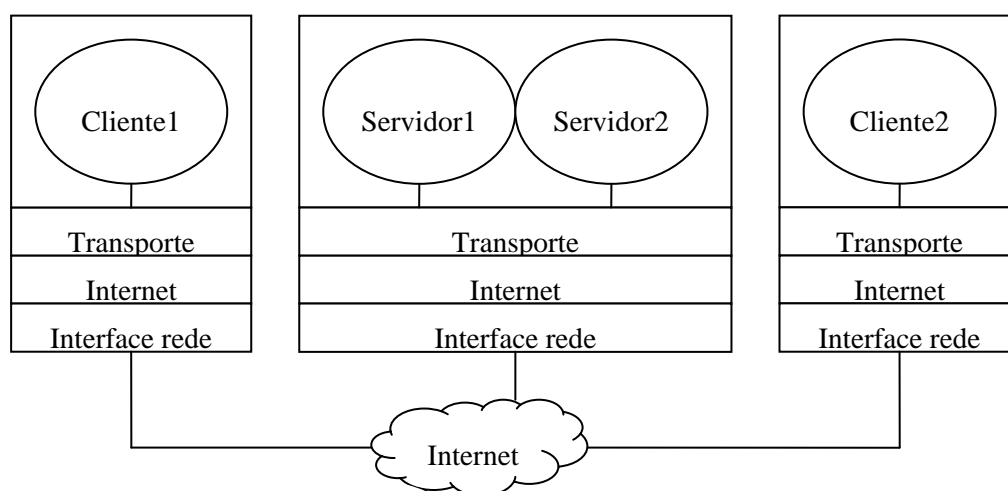
Como a maioria dos programas de aplicação, o cliente e o servidor usam o protocolo de transporte para se comunicar. Por exemplo, a figura abaixo ilustra um cliente e um servidor utilizando a pilha TCP/IP:



Como a figura mostra, aplicações do cliente ou do servidor interagem diretamente com um protocolo da camada de transporte para estabelecer comunicação e para enviar e receber mensagens individuais. Por isso, um computador precisa da pilha completa dos protocolos para rodar tanto um cliente como um servidor.

18.10 Múltiplos Serviços em um Computador

Um sistema computacional suficiente robusto pode rodar vários clientes e servidores ao mesmo tempo. Duas potencialidades são requeridas. Primeira, o computador deve ter recursos de hardware suficientes (por exemplo, um processador rápido e muita memória). Segundo, o computador deve ter um sistema operacional que permita que múltiplos programas de aplicação possam ser executados simultaneamente (Por exemplo, UNIX ou Win98). Nesses sistemas, um programa de servidor roda para cada serviço oferecido. Por exemplo, um único computador pode rodar um arquivo de servidor ou de servidor de Web. A figura ilustra um arranjo possível.



A figura ilustra clientes em dois computadores acessando dois servidores em um terceiro computador. Embora o computador possa operar como múltiplos servidores, o computador precisa de uma só conexão física com a internet.

Permitindo que um computador opere múltiplos servidores é útil porque o hardware pode ser compartilhado entre os múltiplos serviços. Condensando vários servidores em um maior também ajuda a reduzir as despesas com o sistema de administração porque isso resulta em menos sistemas computacionais para se administrar. Mais importante, a experiência mostra que a demanda por um servidor é freqüentemente esporádica – um dado servidor pode permanecer sem atividade por longos períodos de tempo. Um servidor sem atividade não utiliza o CPU do computador enquanto espera pela chegada de um pedido. Por isso, se a demanda por serviços é baixa, condensar servidores em um único computador pode reduzir dramaticamente os custos sem reduzir o desempenho significativamente.

18.11 Identificando um Serviço Particular

Protocolos de transporte fornecem o mecanismo que permite a um cliente especificar qual serviço é desejado. O mecanismo designa a cada serviço um identificador único e requer a ambos o servidor e cliente a usarem tal identificador. Quando o servidor começa uma execução, ele registra com o protocolo local especificando o identificador para aquele serviço que ele oferece. Quando um cliente contata um servidor remoto, o cliente especifica o identificador do serviço desejado. O protocolo de transporte na máquina cliente envia o identificador à máquina servidor quando faz um pedido. O protocolo de transporte na máquina servidor usa o identificador para determinar qual programa deverá lidar com aquele pedido.

Como exemplo de identificação de serviço, considere o Protocolo de Controle de Transporte (TCP) descrito no capítulo 22. TCP usa valores inteiros de 16 bits conhecidos como *Números de Porta de Protocolo* para identificar serviços e designa um único número de porta para cada serviço. Um servidor especifica o número da porta para o serviço oferecido e então espera passivamente por comunicação. O cliente especifica o número da porta do serviço desejado quando envia o requerimento. Então, o software TCP presente no computador servidor usa o número da porta na mensagem para determinar qual servidor deve receber aquele pedido.

18.12 Múltiplas Cópias de um Servidor para um só Serviço

Tecnicamente, um sistema computacional que permita que múltiplos programas de aplicação sejam executados ao mesmo tempo é dito a suportar **Concurrency** e um programa que tenha mais de uma **thread of control** é chamado *Programa Concurrency*. **Concurrency** é fundamental ao modelo cliente-servidor de interação porque um servidor concorrente oferece serviço a múltiplos clientes ao mesmo tempo, ou seja, nenhum cliente tem que esperar outro cliente terminar de usar um mesmo serviço.

Para entender como serviço simultâneo é importante, considere o que acontece se um serviço requer um tempo significativo para satisfazer cada pedido. Por exemplo, um serviço de transferência de arquivos permite que um cliente obtenha uma cópia de um arquivo remoto: o cliente envia o nome do arquivo no pedido e o servidor retorna uma cópia do mesmo. Se o cliente pedir um arquivo pequeno, o servidor pode enviá-lo por inteiro em alguns milissegundos. Entretanto, um servidor pode precisar de muitos minutos para transferir um arquivo que contenha um conjunto de imagens digitais de alta resolução.

Se um servidor de arquivos lidar com um pedido por vez, todos os clientes devem esperar enquanto o servidor transfere arquivos para um deles. Em contra partida, um servidor concorrente de arquivos pode lidar com múltiplos clientes simultaneamente. Quando um pedido chega, o servidor designa-o a uma **thread of control** que possa executá-lo concorrentemente a outros **threads** existentes. Na essência, uma cópia separada do servidor lida com cada pedido. Por isso, pedidos pequenos podem ser satisfeitos rapidamente, sem ter de esperar pelo término de pedidos longos.

18.13 Criação Dinâmica de Servidor

A maioria dos servidores **concurrent** opera dinamicamente. O servidor cria uma nova **thread** para cada pedido que chega. De fato, o programa do servidor é construído em duas partes: uma aceita os pedidos e cria uma nova **thread** para o mesmo, e outra que consiste no código que lida com cada pedido individualmente. Quando um servidor **concurrent** começa a executar, só a primeira parte roda. Isto é, o **thread** principal do servidor espera pela chegada de um pedido. Quando um pedido chega, o **thread** principal cria um novo serviço **thread** para lidar com o mesmo. O serviço **thread** lida com um pedido e então é fechado. Enquanto isso, o **thread** principal mantém o servidor vivo – depois de criar um **thread** para lidar com um pedido, o **thread** principal espera pela chegada de outro pedido.

18.14 Protocolos de Transporte e Comunicação Não Ambígua

Se múltiplas cópias de um servidor existem, como um cliente pode interagir com a cópia correta? E mais ainda, como um pedido pode ser transmitido para a cópia correta do servidor? A resposta para estas questões está no método que o protocolo de transporte utiliza para identificar um servidor. Dissemos que a cada serviço é designado um identificador único e que cada pedido de um cliente inclui um identificador de serviço, tornando possível para o software do protocolo de transporte presente no computador do servidor associar um pedido com o servidor correto. Na prática, a maioria dos protocolos de transporte designa cada cliente com um identificador único e exige que o cliente inclua esse identificador quando fizer um pedido. O software do protocolo de transporte presente no computador servidor usa tanto o identificador do cliente como o do servidor para escolher a cópia do servidor que foi criada para lidar com aquele cliente.

Como exemplo, considere os identificadores utilizados em uma comunicação TCP. O TCP requer que cada cliente escolha um número de porta local que não esteja designada para outro serviço. Quando ele envia um segmento TCP, o cliente deve colocar o seu número de porta local no campo *Porta de Origem* e o número da porta do servidor no campo *Porta de Destino*. No computador servidor, o TCP usa a combinação dos números de porta de origem e de destino (como endereços IP de cliente e servidor) para identificar uma comunicação em particular. Por isso, mensagens podem chegar de dois ou mais cliente para um mesmo serviço sem causar problemas. O TCP transmite cada segmento para a cópia do servidor que tenha concordado em lidar com o cliente.

18.15 Conexão-Orientada e Transporte Sem Conexão

Protocolos de transporte suportam duas formas básicas de comunicação: com conexão orientada ou sem conexão. Para utilizar um protocolo de transporte com conexão, as duas aplicações devem estabelecer uma conexão e então enviar os dados através da conexão. Por exemplo, o TCP provê uma interface de conexão orientada para as aplicações. Quando elas usam o TCP, a aplicação deve primeiramente requisitar ao TCP a abertura de uma conexão com outra aplicação. Uma vez estabelecida a conexão, as duas aplicações podem trocar dados. Quando as aplicações terminam a comunicação, a conexão deve ser fechada.

A alternativa para a comunicação com conexão orientada é a interface sem conexão que permite que uma aplicação envie uma mensagem para qualquer destino a qualquer hora. Quando se utiliza um protocolo de transporte sem conexão, a aplicação de origem deve especificar o destino de cada mensagem que envia. Por exemplo, na pilha de protocolos TCP/IP, o *User Datagram Protocol (UDP)* provê transporte sem conexão. Uma aplicação que utiliza o UDP pode enviar uma sequência de mensagens, onde cada mensagem é enviada a um destino diferente.

Clientes e servidores usam tanto protocolos de conexão orientada quanto protocolos sem conexão para se comunicar. Quando utilizam transporte com conexão orientada, o cliente primeiramente estabelece uma conexão com um servidor específico. A conexão então se mantém estável enquanto o cliente envia pedidos e recebe respostas. Quando ele termina de usar o serviço, o cliente fecha a conexão.

Clientes e servidores que usam protocolos sem conexão trocam mensagens individuais. Por exemplo, muitos serviços que usam transporte sem conexão requerem um cliente para mandar cada pedido em uma única mensagem e um servidor para retornar cada resposta em mensagens individuais.

18.16 Um Serviço Através de Vários Protocolos

Servidores não precisam escolher entre transporte com ou sem conexão; é possível oferecer ambos. Isto é, o mesmo serviço pode estar disponível através de dois ou mais protocolos de transporte, deixando a escolha do tipo de transporte para o cliente fazer. Permitindo o uso de vários protocolos de transporte aumenta a flexibilidade por fazer o serviço disponível a clientes que não tem acesso a um protocolo de transporte em particular.

Existem duas implementações possíveis de servidores multi-protocolos. A primeira implementação é direta: dois servidores existem para o mesmo serviço. Um servidor utiliza transporte sem conexão, enquanto o outro usa transporte com conexão orientada. A segunda implementação é mais complexa: um só programa do servidor interage com dois ou mais protocolos de transporte ao mesmo tempo. O servidor aceita pedidos de qualquer tipo de protocolo suportado e utiliza o protocolo do pedido para enviar a resposta.

18.17 Interações Complexas Cliente-Servidor

Algumas das funcionalidades mais interessantes e úteis das computações entre cliente e servidor surgem de interações arbitrárias entre clientes e servidores. Em particular, é importante notar:

- Uma aplicação do cliente não está restrita a acessar apenas um único serviço. Uma só aplicação pode primeiramente se tornar cliente de um serviço e depois se tornar cliente de outro. O cliente contata um servidor diferente (talvez até em um computador diferente) para cada serviço.
- Uma aplicação não está restrita a acessar um único servidor para um dado serviço. Em alguns serviços, cada servidor fornece informações diferentes às informações de outros servidores rodando em outros computadores. Por exemplo, um servidor de datas pode prover hora certa e data do computador onde ele está rodando. Um servidor em um computador em um fuso horário diferente fornecerá uma resposta diferente. Em outros serviços, todos os servidores podem fornecer a mesma informação. Nesses casos, um cliente pode enviar um pedido para vários servidores para aumentar a performance – o cliente utiliza a informação enviada pelo servidor que responder primeiro.
- Um servidor não está restrito a executar interações além das de cliente-servidor – um servidor de um serviço pode se tornar cliente de outro. Por exemplo, um servidor de arquivos que precise da hora que um arquivo foi acessado pode se tornar cliente de um servidor de horários. Isto é, enquanto ele está lidando com um pedido de um arquivo, o servidor de arquivos envia um pedido ao servidor de horários, espera a resposta e então continua a lidar com o pedido do arquivo.

18.18 Interações e Dependências Circulares

É claro, servidores devem ser planejados cuidadosamente para evitar dependências circulares. Para entender os problemas que podem aparecer, considere um servidor de arquivos que utilize um servidor de horários para obter o horário atual sempre que um arquivo fosse acessado. Uma dependência circular pode ocorrer se o servidor de horário também utilizar o

servidor de arquivos. Por exemplo, suponha que um programador seja chamado para modificar o servidor de horários para que ele mantenha um registro de cada pedido. Se o programador escolher ter o servidor de horários como um cliente do servidor de arquivos, um ciclo pode aparecer: o servidor de arquivos se torna cliente do servidor de horários, o qual se torna cliente do servidor de arquivos e assim por diante. O resultado é um desastre análogo a um loop infinito dentro de um programa.

Embora dependências entre pares de servidores possam ser facilmente vistas e evitadas, um conjunto grande de dependências pode não ser tão óbvio. Imagine um ciclo de dependências que inclua uma dúzia de servidores, cada um operando em um computador diferente. Se cada servidor for mantido por um programador diferente, as dependências entre eles podem ser difíceis de identificar.

Capítulo 19 – A Interface Socket

19.1 Introdução

Este capítulo fornece detalhes adicionais sobre a interação cliente-servidor, explicando a interface entre o aplicativo e o software de protocolo. O capítulo explica como um aplicativo utiliza o software de protocolo para se comunicar, e mostra como exemplo um conjunto de procedimentos que o aplicativo utiliza para se tornar cliente ou servidor, para contatar um destinatário, ou transferir dados.

19.2 Application Program Interface

Os aplicativos no cliente e servidor utilizam os protocolos de transporte para se comunicarem. Quando há interação com o software de protocolo, o aplicativo deverá especificar detalhes como se é um servidor ou cliente. Em adição, os aplicativos que se comunicam devem especificar também detalhes como quais dados o emissor vai enviar, e o receptor deve especificar onde os dados recebidos devem ser armazenados.

A interface que o aplicativo utiliza quando interage com o software de protocolo de transporte é conhecida como *Application Program Interface* (API). O API define um conjunto de operações que o aplicativo pode realizar quando interage com o software de protocolo. Assim, o API determina qual funcionalidade é disponível para o aplicativo bem como a dificuldade de criar um programa para usar a funcionalidade.

A maioria dos sistemas de programação define o API como um conjunto de procedimentos que o aplicativo pode chamar e os argumentos que cada procedimento exige. Usualmente, o API contém procedimentos separados para cada operação básica. Por exemplo, o API pode ter um procedimento que é utilizado para estabelecer comunicação e outro procedimento para enviar dados.

19.3 O Socket API

A norma do protocolo de comunicação não especifica um API para os aplicativos interagirem com os protocolos. Ao invés disso, o protocolo especifica as operações gerais que devem ser fornecidas, e permite que cada sistema operacional defina um API específico usado para realizar as operações. Assim, a norma dos protocolos pode sugerir uma operação necessária para permitir o aplicativo enviar dados e o API especifica o nome exato da função e tipo de cada argumento.

Embora o padrão dos protocolos permita que os projetistas do sistema operacional escolham um API, muitos adotaram o *Socket API*. O Socket API é disponível para vários sistemas operacionais, incluindo sistemas utilizados em computadores pessoais (Microsoft NT e Windows 98) bem como em vários sistemas UNIX (Sun Microsystems' Solaris).

O socket API originou-se como parte do sistema operacional BSD UNIX. O trabalho era apoiado por uma concessão governamental, sobre a qual a Universidade da Califórnia, em Berkeley desenvolveu e distribuiu uma versão do UNIX que continha os protocolos de ligação TCP/IP. Muitos vendedores de computadores portaram o sistema BSD nos hardwares, e usou isto como produto básico de sistema operacional. Assim, o socket API se tornou de fato, o padrão na indústria.

19.4 Sockets and Socket Libraries

Com o uso maior do sockets (socket API), os fornecedores de outros sistemas operacionais decidiram adicionar o socket API aos seus sistemas. Em muitos casos, ao invés de modificar os seus sistemas operacionais, os fornecedores criaram a *socket library* que provém o socket API.

Do ponto de vista do programador, a socket library fornece a mesma semântica da implementação de sockets no sistema operacional. O programa chama os procedimentos do socket, que são ou supridos pelos procedimentos do sistema operacional ou da biblioteca de rotinas. Assim, um aplicativo que usa socket pode ser copiado para um novo computador, compilado, carregar a biblioteca de socket do computador, e então ser executado – o código base não necessita ser mudado quando mudamos o programa de um computador para outro.

A socket library tem uma implementação completamente diferente do que o socket API nativo suprido pelo sistema operacional. Diferentemente das rotinas do socket nativo, o código dos procedimentos da socket library são ligados dentro do aplicativo e reside no espaço de endereços do aplicativo. Quando o aplicativo chama um procedimento da socket library, o controle passa para a biblioteca a rotina que, em turno, faz uma ou mais chamadas nas funções básicas do sistema operacional para alcançar o efeito desejado. Funções fornecidas pelo sistema operacional básico não precisam parecer totalmente com o socket API – rotinas na socket library escondem o sistema operacional nativo do aplicativo e apresentam apenas a interface socket.

19.5 Comunicação Socket e UNIX I/O

Como os sockets foram originalmente desenvolvidos como parte do sistema operacional UNIX, os sockets utilizam muitos conceitos encontrados em outras partes do UNIX. Em particular, os sockets foram integrados com o I/O. Assim, para entender o sockets precisamos entender as facilidades do UNIX I/O.

O UNIX utiliza o paradigma *abrir-ler-escrever-fechar* (*open-read-write-close*) para todos os I/O. Por exemplo, um aplicativo deve primeiro chamar a rotina *open* para acessar o arquivo. O aplicativo então utiliza *read* ou *write* para ler os dados ou armazenar dados no arquivo. Finalmente o aplicativo usa a rotina *close* para especificar que já acabou de usar o arquivo.

Quando o aplicativo abre um arquivo ou componente, a rotina *open* retorna um *descriptor*, um pequeno número inteiro que identifica o arquivo; o aplicativo deve especificar o descriptor quando requisitar transferência de dados. Por exemplo, se um aplicativo utiliza o *open* para acessar um arquivo chamado *foobar*, o procedimento de abrir o arquivo retorna o descriptor 4. Na subsequente ação *write* que especifica o descriptor 4 irá escrever dados em *foobar*; ou seja, o nome do arquivo não aparece na rotina *write*.

19.6 Sockets, Descriptors, e Network I/O

Antes de um aplicativo utilizar os protocolos para se comunicar, o aplicativo deve requisitar ao sistema operacional a criação de um *socket* que será utilizado na comunicação. O sistema irá retornar um pequeno número inteiro, o *descriptor* que irá identificar o socket. O aplicativo então passa o descriptor como argumento quando chama os procedimentos para transferir os dados através da rede; o aplicativo não necessita especificar detalhes sobre o destinatário remoto cada vez que for transferir dados.

Em uma implementação UNIX, os sockets são completamente integrados com outros I/O. O sistema operacional fornece um conjunto único de descriptors para arquivos, componentes, processos internos e comunicação em rede. Como resultado, procedimentos como *read* e *write* são completamente gerais – um aplicativo pode utilizar o mesmo procedimento

para enviar dados para outro programa, arquivo, ou através da rede. Na terminologia atual, o descriptor representa um objeto, e o procedimento *write* representa o método aplicado ao objeto.

A vantagem principal deste sistema é a flexibilidade: um único aplicativo para gravar os dados pode transferir dados para um local arbitrário. Se o aplicativo recebe um descriptor que corresponde um componente, o aplicativo irá enviar dados ao componente. Se o aplicativo recebe um descriptor que corresponde a um arquivo, o aplicativo irá armazenar dados no arquivo. Se o aplicativo recebe um descriptor que corresponde a um socket, o aplicativo enviará dados através da rede para um computador remoto.

19.7 Parâmetros e o Socket API

A programação dos sockets difere do convencional I/O porquê um aplicativo deve especificar vários detalhes para usar o socket. Por exemplo, um aplicativo deve escolher um protocolo de transporte em particular, fornecer o endereço de protocolo de um computador remoto, e especificar quando o aplicativo é cliente ou servidor. Para acomodar todos os detalhes, cada socket tem muitos parâmetros e opções – o aplicativo pode fornecer valores para todos.

Como as opções e parâmetros são representados no API? Para evitar que uma única função do socket tenha parâmetros separados para cada opção, os projetistas do socket decidiram definir o API com muitas funções. Na essência, um aplicativo cria o socket e então chama as funções para especificar em detalhes como o socket será usado. A vantagem é que a maioria das funções tem três ou menos parâmetros; a desvantagem é que o programador deve lembrar de chamar várias funções quando usa o socket. As próximas seções ilustram o conceito.

19.8 Procedimentos do Socket API

19.8.1 O Procedimento Socket

O procedimento *socket* cria um socket e retorna um descriptor inteiro:

```
descriptor = socket(protofamily,type,protocol)
```

O argumento *protofamily* especifica a família do protocolo que será usada com o socket. Por exemplo, o valor *PF_INET* é usado para especificar o protocolo TCP/IP, e *PF_DECnet* é usado para especificar o protocolo Digital Equipment Corporation.

O argumento *type* especifica o tipo de comunicação que o socket utiliza. Os dois tipos mais comuns são as transferências por conexão-orientada (especificada com o valor *SOCK_STREAM*) e transferência de mensagem sem conexão orientada (especificada com o valor *SOCK_DGRAM*).

O argumento *protocol* especifica um protocolo de transporte particular usado com o socket. Tendo o argumento *protocol* em adição do argumento *type*, permite um único protocolo adicionar dois ou mais protocolos que fornecem o mesmo serviço. É claro, os valores utilizados no argumento *protocol* dependem da família do mesmo. Por exemplo, embora o protocolo TCP/IP incluam o protocolo TCP, o protocolo AppleTalk não o contém.

19.8.2 O Procedimento Close

O procedimento *close* ordena ao sistema terminar o uso do socket. A forma deste é:

```
close(socket)
```

onde *socket* é o descriptor referente ao socket que será fechado. Se o socket usado é um protocolo de conexão orientada, o *close* fecha a conexão antes de fechar o socket. Fechando um socket imediatamente o uso do mesmo é finalizado e o descriptor é liberado, prevenindo o aplicativo enviar mais dados, e o protocolo de transporte não irá aceitar mais mensagens enviadas diretamente ao socket, prevenindo o socket de receber mais dados.

19.8.3 O Procedimento Bind

Quando o socket é criado, este não tem nem um endereço local nem endereço remoto. O servidor utiliza o procedimento *bind* fornece um número de porta do protocolo no qual o servidor irá esperar pelo contato. O *bind* leva três argumentos:

```
bind(socket,localaddr,addrlen)
```

O argumento *socket* é o descriptor do socket q foi criado anteriormente, e que não esteja limitado; a chamada é uma requisição para o socket seja designado em um número de porta do protocolo. O argumento *localaddr* é a estrutura que especifica o endereço local que será designado ao socket, e o argumento *addrlen* é um inteiro que especifica o comprimento do endereço. Como os sockets podem ser usados com protocolos arbitrários, o formato do endereço depende do protocolo a ser utilizado. O socket API define uma forma genérica para representar endereços, e então solicita que cada família de protocolo especifique como é a forma geral do endereço do protocolo.

O formato genérico de se representar um endereço é definido com a estrutura *sockaddr*. Embora várias versões fossem liberadas, o ultimo código Berkeley define a estrutura *sockaddr* tendo três campos:

```
struct sockaddr{
    u_char  sa_len;           /*tamanho total do endereço*/
    u_char  sa_family;       /* família do endereço */
    char    sa_data[14];     /* o próprio endereço*/
};
```

O campo *sa_len* consiste em um único octeto que especifica o tamanho do endereço. Campo *sa_family* especifica a família que o endereço pertence (o símbolo constante *AF_INET* usado para representar o TCP/IP por exemplo). Finalmente, o campo *sa_data* contém o endereço.

Cada família de protocolo define o formato exato do endereço no campo *sa_data* da estrutura *sockaddr*. Por exemplo, o protocolo TCP/IP utiliza a estrutura *sockaddr_in* para definir o endereço:

```
struct sockaddr_in{
    u_char  sin_len;         /*tamanho total do endereço*/
    u_char  sin_family;      /* família do endereço */
    u_short sin_port;        /* número da porta do protocolo*/
    struct  in_addr sin_addr; /* endereço IP do computador*/
    char    sin_zero[8]     /* não usado - configurado p/ zero*/
};
```

Os dois primeiros campos da estrutura, *sockaddr_in* correspondem exatamente aos dois primeiros campos da estrutura genérica *sockaddr*. Os últimos três campos definem exatamente a forma de endereços que o protocolo TCP/IP espera. Cada endereço identifica ambos, o computador e o aplicativo em particular no computador. O campo *sin_addr* contém o endereço IP do computador, e o campo *sin_port* contém o número da porta do protocolo referente ao aplicativo. Embora o TCP/IP precise de seis octetos para armazenar um endereço completo, a

estrutura genérica *sockaddr* reserva catorze octetos. Assim o campo final na estrutura *sockaddr_in* define um campo de oito octetos de zeros.

O servidor chama a rotina *bind* para especificar número da porta que o servidor irá aceitar o contato. Porém, em adição ao número da porta, a estrutura *sockaddr_in* contém também um campo para o endereço IP. Como o servidor pode escolher o endereço IP quando especifica o endereço, quando um host é multi-homed isso causa problemas, ou seja, tenha vários endereços, pois significará que o servidor irá aceitar apenas requisições enviadas por um endereço específico. Para permitir que o host opere no modo multi-homed, o socket API inclui uma constante simbólica especial, *INADDR_ANY*, que permite o servidor usar uma porta específica no endereço específico de IP do computador.

19.8.4 O Procedimento Listen

Após especificar a porta, o servidor deve instruir o sistema operacional para colocar o socket no modo passivo, então ele irá esperar a conexão dos clientes. Para fazer isso, o servidor chama o procedimento *listen*, o qual toma dois argumentos:

```
listen(socket,queusize)
```

O argumento *socket* é o descriptor do socket q foi criado e amarrado a um endereço local, e o argumento *queusize* especifica o comprimento da requisição de fila do socket.

O sistema operacional constrói requisições de filas separadas para cada socket. Inicialmente, a fila está vazia. Com a chegada das requisições vindas dos clientes, cada uma é colocada na fila; quando o servidor pede uma resposta da requisição vinda do socket, o sistema retorna a próxima da fila e assim por diante. Se a fila estiver cheia quando a requisição chega, a mesma é rejeitada. Ter uma fila de requisições permite o sistema segurar novas requisições que chegam enquanto o servidor está ocupado lidando com a requisição anterior. Os parâmetros permitem cada servidor escolher o tamanho máximo de fila que é apropriado para o serviço esperado.

19.8.5 O Procedimento Accept

Todos servidores começam chamando o processo *socket* para criar um socket e *bind* para especificar o número de porta do protocolo. Após executar as duas chamadas, o servidor que usa protocolo de transporte com conexão não orientada está pronto para aceitar mensagens. Já, se o servidor que usa protocolo de transporte com conexão orientada, é necessário o servidor chamar o processo *listen* para configurar o socket no modo passivo, e então deve aceitar a conexão requisitada. Uma vez que a conexão for aceita, o servidor pode utilizar a mesma para se comunicar com um cliente. Após terminar a comunicação, o servidor fecha a conexão.

O servidor que utiliza protocolo de transporte com conexão orientada deve chamar o procedimento *accept* para aceitar a próxima conexão requisitada. Se a requisição esta presente na fila, o *accept* tem retorno imediato, se a requisição não tiver chegado, o sistema bloqueia o servidor até que o cliente forme uma conexão. A chamada *accept* tem a seguinte forma:

```
newsock=accept(socket,caddress,caddresslen)
```

O argumento *socket* é o descriptor do socket que foi criado e amarrado à uma porta específica do protocolo. O argumento *caddress* é o endereço da estrutura do tipo *sockaddr*, *caddresslen* é um ponteiro para um inteiro. O *accept* preenche os campos do argumento *caddress* com o endereço do cliente que formou a conexão, e configura *caddresslen* para o tamanho do endereço. Finalmente, o *accept* cria um novo socket para a conexão, e retorna o descriptor para o sistema. O servidor usa o novo socket para comunicar com o cliente, e então fecha o socket quando tiver acabado. No intervalo, o socket original do servidor mantém inalterado – após ter terminado a comunicação com o cliente, o servidor usa o socket original para aceitar novas conexões de um cliente.

19.8.6 O Procedimento Connect

Os clientes utilizam o processo *connect* para estabelecer uma conexão com um servidor específico. A forma é:

```
connect(socket,saddress,saddresslen)
```

O argumento *socket* é o descriptor do socket no cliente utilizado para a conexão. O argumento *saddress* é a estrutura *sockaddr* que especifica o endereço do servidor e número da porta do protocolo. O argumento *saddresslen* especifica o tamanho do endereço do servidor, medido em octetos.

Quando o protocolo de transporte é utilizado com conexão orientada como TCP, o *connect* inicializa a conexão de transporte de nível com o servidor específico. Na essência, *connect* é o procedimento que o cliente usa para conectar ao servidor que estiver utilizando o procedimento *accept*.

Quando o cliente utiliza protocolo de transporte com conexão não orientada, este pode também utilizar o *connect*. De qualquer forma, utilizar o *connect* não inicia uma conexão ou faz com que pacotes passem pela rede. O *connect* meramente marca o socket como *connected*, e grava o endereço do servidor.

Para entender o porquê de utilização do *connect* em um protocolo de conexão não orientada, é só lembrar que protocolos de conexão não orientada requerem que o emissor especifique o endereço de destinatário em cada mensagem. Em muitos aplicativos, todas as mensagens vão para o mesmo destinatário. Nestes casos, o socket conectado fornece o endereço apenas uma vez, ao invés de especificar o endereço em cada mensagem.

19.8.7 Os Procedimentos Send, Sendto e Sendmsg

Os clientes e servidores necessitam ambos enviar informações. Usualmente, o cliente envia uma requisição, e o servidor envia uma resposta. Se o socket está conectado, o procedimento *send* pode ser utilizado para transferir dados. *Send* tem quatro argumentos:

```
send(socket,data,length,flags)
```

O argumento *socket* é o descriptor do socket usado, o argumento *data* é o endereço de memória dos dados que serão utilizados, o argumento *length* é um inteiro que especifica o número de octetos dos dados, e o argumento *flags* contém os bits que solicitam opções especiais.

Os procedimentos *sendto* e *sendmsg* permitem ao cliente ou servidor enviar uma mensagem com um socket que esteja desconectado; ambos solicitam ao emissor que o mesmo especifique o destinatário. *Sendto* leva o endereço do destinatário como um argumento. Este tem a seguinte forma:

```
sendto(socket,data,length,flags,destaddress,addresslen)
```

Os quatro primeiros argumentos correspondem aos quatro argumentos do procedimento *send*. Os dois argumentos finais especificam o endereço do destinatário e o tamanho do endereço. A forma do endereço no argumento *destaddress* é a estrutura *sockaddr* (especificamente, a estrutura *sockaddr_in* quando usado com TCP/IP).

O procedimento *sendmsg* realiza a mesma operação do *sendto*, mas abrevia o argumento ao definir uma estrutura. A lista menor de argumentos pode fazer com que programas que utilizem o *sendmsg* ter maior facilidade de leitura:

sendmsg(socket,msgstruct,flags)

O argumento *msgstruct* é uma estrutura que contém informações sobre o endereço do destinatário, o tamanho do endereço, a mensagem a ser enviada e o tamanho da mensagem.

```
struct  msgstruct{
    struct  sockaddr      *m_saddr;
    struct  datavec *m_dvec;
    int     m_dvlength;
    struct  access        *m_rights;
    int     m_alength;
};
```

Os detalhes da estrutura não são importantes – devem ser vistos como uma maneira de combinar vários argumentos em uma única estrutura. A maioria dos aplicativos usa apenas os três campos, os quais especificam o endereço de protocolo e uma lista de itens dos dados que abrangem a mensagem.

19.8.8 Os Procedimentos Recv, Recvfrom e Recvmsg

O cliente e servidor necessitam ambos de receber dados enviados pelo outro. O socket API fornece vários processos que podem ser utilizados. Por exemplo, um aplicativo pode utilizar o processo *recv* para receber dados de um socket conectado. O processo tem a seguinte forma:

recv(socket,buffer,length,flags)

O argumento *socket* é o descriptor do socket nos quais os dados são recebidos. O argumento *buffer* especifica o endereço na memória no qual a mensagem deve ser armazenada, e o argumento *length* especifica o tamanho do buffer. Finalmente, o argumento *flags* permite ao sistema controlar detalhes, como fazer uma cópia exata da mensagem sem retirar a mesma do socket.

Se o socket não se encontra conectado, este pode ser utilizado para receber mensagens de um conjunto de clientes arbitrários. Nestes casos, o sistema retorna o endereço do emissor com a mensagem. Aplicativos usam o processo *recvfrom* para receber a mensagem e o endereço do emissor.

recvfrom(socket,buffer,length,flags,sndraddr,saddrlen)

Os primeiros quatro argumentos correspondem aos argumentos do processo *recv*; os dois argumentos adicionais, *sndraddr* e *saddrlen*, são utilizados para armazenar o endereço IP do emissor. O argumento *sndraddr* é um ponteiro para a estrutura *sockaddr* na qual o sistema grava o endereço do emissor, e o argumento *saddrlen* é um ponteiro para um inteiro que o sistema usa para gravar o tamanho do endereço. O processo *recvfrom* grava o endereço do emissor exatamente no mesmo formato que o processo *sendto* espera. Assim, se um aplicativo usa *recvfrom* para receber uma mensagem, enviar uma resposta é fácil – o aplicativo simplesmente usa a gravação do endereço como destinatário para a resposta.

O socket API inclui um processo de entrada análogo ao processo de envio *sendmsg*. O procedimento *recvmsg* opera da mesma maneira que o *recvfrom*, mas necessita menos argumentos. Este tem a seguinte forma:

recvmsg(socket,msgstruct,flags)

Onde o argumento *msgstruct* fornece o endereço da estrutura que suporta o endereço da mensagem recebida, bem como a localização do endereço de IP do emissor. O argumento *msgstruct* gravado pelo processo *recvmsg* tem exatamente o mesmo formato da estrutura requisitada pelo processo *sendmsg*. Assim, os dois processos trabalham muito bem para receber uma mensagem e enviar uma resposta.

19.9 Lendo e Escrevendo com Sockets

Foi dito que o socket API foi originalmente projetado para ser uma parte do UNIX, o qual usa *read* e *write* para I/O. Consequentemente, o socket também permite aos aplicativos utilizarem *read* e *write* para transferir dados. Como no *send* e *recv* não há argumentos que permitam o emissor especificar o destino, os processos *read* e *write* tem três argumentos: o socket descriptor, a localização do buffer na memória utilizado para armazenar os dados, e o tamanho do buffer de memória. Assim, o *read* e *write* devem ser usados com sockets conectados.

A maior vantagem de usar o *read* e *write* é a generalização – um aplicativo pode ser criado para transferir dados para um descriptor sem saber se o descriptor corresponde a um arquivo ou socket. Assim, os programadores podem usar um arquivo no disco local para testar a função de cliente ou servidor antes de tentar a comunicação através da rede.

19.10 Sockets, Processos e Heranças.

Como vários servidores são coexistentes, o socket API é projetado para trabalhar com programas coexistentes. Embora os detalhes envolvam o sistema operacional, as implementações do socket API seguem o princípio que cada novo processo em execução é criado e herda uma cópia de todos sockets abertos pelo processo que o criou.

Para entender como os servidores usam as heranças de socket, é importante saber que os sockets utilizam um mecanismo chamado contagem de referência. Quando o socket é criado pela primeira vez, o sistema configura a contagem de referência para 1; o socket existe enquanto a contagem de referência permanece positiva. Quando o programa cria um processo adicional, o sistema fornece o processo com a lista de todos os sockets que o aplicativo pertence, e incrementa a contagem de referência com o valor 1. Quando o processo chama a rotina *close* para o socket, o sistema faz um decremento de 1 da contagem de referência, e remove o socket da lista de processos.

O processo principal de um aplicativo cria um socket que o servidor usa para aceitar as conexões. Quando um pedido de conexão chega, o sistema cria um novo socket para a conexão. Imediatamente após o processo principal criar o processo auxiliar para lidar com a nova conexão, ambos os processos tem acesso ao antigo e novo socket, e a contagem de referência tem o valor de 2. De qualquer forma, o processo principal não irá usar o novo socket, e o processo auxiliar não irá utilizar o socket original. Então, o processo principal usa a rotina *close* para o novo socket, e o processo auxiliar utiliza a rotina *close* para o socket original, reduzindo a contagem de referência para 1.

Após o processo de serviço ter acabado este irá chamar rotina *close* no novo socket, reduzindo a contagem de referência para zero e apagando assim o novo socket.

Capítulo 20 – Exemplo de um Cliente e Servidor

20.1 Introdução

O capítulo anterior descreve o socket API, incluindo os processos individuais que podem ser usados, os argumentos usados em cada processo e as operações que os processos realizam. Este capítulo continua a explicação do socket API examinando um cliente e servidor simples que utilizam o socket para comunicar. Embora isto não ilustre todas as possibilidades de projeto de clientes e servidores, o código exemplo ilustra a seqüência das chamadas dos processos, e mostra a diferença entre a utilização em um cliente e em um servidor.

20.2 Comunicação com Conexão Orientada

Para realizar uma comunicação, o cliente e servidor deverão selecionar o protocolo de transporte que irá suportar o serviço de conexão orientada ou de conexão não orientada. Serviços com conexão não orientada permitem ao aplicativo enviar mensagens para um destinatário arbitrário a qualquer momento; o destinatário não precisa concordar em aceitar a mensagem antes de a transmissão ocorrer. Já no serviço de conexão orientada, é necessário que os dois aplicativos estabeleçam uma conexão antes de enviar dados. Para estabelecer a conexão, os aplicativos interagem com o software do protocolo de transporte no computador local, e os dois módulos do protocolo de transporte trocam mensagens via rede. Após os dois lados aceitarem que a conexão foi estabelecida, os aplicativos podem enviar dados.

20.3 Um exemplo de Serviço

Um exemplo de cliente e servidor irá ajudar a esclarecer muitos detalhes da interação com conexão orientada, e mostrará como o software para o serviço de conexão orientada usa o socket. Para manter o tamanho do programa pequeno e focalizar nas chamadas do socket, foi escolhido um serviço trivial: o servidor mantém uma contagem do número de clientes que acessaram o serviço e relata a contagem quando o cliente contata o servidor.

Para simplificar a implementação, o serviço é projetado para usar ASCII. O cliente forma a conexão com o servidor e espera a resposta. Quando a requisição de conexão chega, o servidor cria uma mensagem no formato ASCII, e envia a mensagem através da conexão, e então fecha a conexão. O cliente mostra a mensagem recebida e então fecha a tela.

Por exemplo, na décima vez que o cliente conecta ao servidor, o cliente irá receber e imprimir a seguinte mensagem:

```
This server has been contacted 10 times.
```

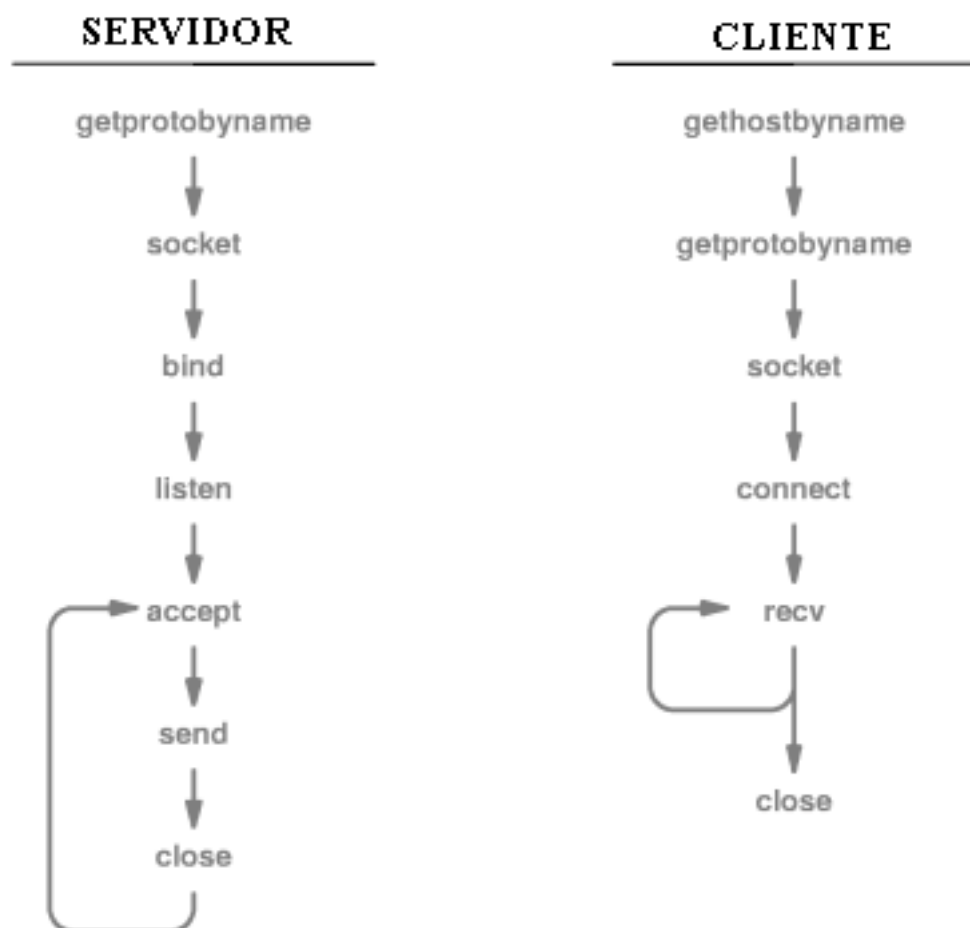
20.4 Argumentos das Linhas de Comandos deste Exemplo de Programa

O exemplo do servidor leva apenas um argumento para a linha de comando, o número da porta no qual o protocolo irá aceitar a requisição. O argumento é opcional; se nenhum número de porta é especificado, o código usará a porta 5193.

No exemplo do cliente temos dois argumentos para linhas de comando; o nome do host no qual irá conectar com o servidor e o número da porta de protocolo usada. Ambos os argumentos são opcionais. Se nenhum número de porta de protocolo é designado, o cliente usará a porta 5193. Se nenhum argumento é especificado, o cliente utiliza a porta padrão e o nome de host *localhost*.

20.5 Seqüência de Processos no Socket

A figura 25.1 abaixo ilustra a seqüência de chamada de processos no socket que o servidor e cliente utilizam para a comunicação.



Como a figura mostra, o servidor utiliza sete processos do socket e o cliente utiliza seis. O cliente começa chamando a biblioteca de procedimentos *gethostbyname* para converter o nome do computador para um endereço IP e *getprotoname* para converter o nome do protocolo para binários internos usados na forma do processo do socket. O cliente então utiliza o processo *socket* para criar o socket e *connect* para conectar o socket ao servidor. Uma vez que a conexão é estabelecida, o cliente utiliza repetidamente o processo *recv* para receber os dados que o servidor envia. Finalmente, após todos dados serem transferidos, o cliente usa a rotina *close* para fechar o socket.

O servidor também utiliza *getprotobyname* para gerar os binários internos que identificam o protocolo antes de usar o *socket* para criar o socket. Uma vez que o socket foi criado, o servidor utiliza *bind* para especificar a porta do protocolo para o socket, e *listen* para ativar o modo passivo. O servidor então entra em um loop infinito no qual chama *accept* para aceitar as próximas requisições de conexão, usa então *send* para enviar uma mensagem para o cliente, e *close* para fechar a nova conexão. Após fechar a conexão, o servidor chama a rotina *accept* para aceitar as próximas novas conexões.

20.6 Exemplo de Código para Cliente

/* client.c – exemplo de programa para cliente que utiliza o TCP*/

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT 5193

extern int errno;
char localhost[] = "localhost";

main(argc, argv)
int argc;
char *argv[];
{
    struct hostent *ptrh;
    struct protoent *ptrp;
    struct sockaddr_in sad;
    int sd;
    int port;
    char *host;
    int n;
    char buf[1000];
#ifdef WIN32
    WSDATA wsaData;
    WSASStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad, 0, sizeof(sad));
    sad.sin_family = AF_INET;

    if (argc > 2) {
        port = atoi(argv[2]);
```

```

    }else {
        port = PROTOPORT
    }
    if (port>0)
        sad.sin_port = htons((u_short)port);
    else{
        fprintf(stderr, *bad port number %s/n",argv[2]);
    }

    if (argc >1) {
        host = argv[1];
    } else {
        host = localhost;
    }

    ptrh = gethostbyname (host);
    if ( ((char *) ptrh) == NULL) {
        fprintf (stderr, "invalid host : %s\n", host);
        exit(1);
    }

    memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

    if ( ((int) (ptrp = getprotobyname("tcp"))) == 0 ) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    sd = socket (PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd<0) {
        fprintf(stderr. "socket creation failed\n");
        exit(1);
    }

    if (connect(sd, (struct sockaddr *)&sad, sizeof(sad))<0) {
        fprintf(stderr, "connect failed\n");
        exit(1);
    }

    n = recv(sd, buf , sizeof(buf), 0);
    while (n>0) {
        write(1,buf,n);
        n=recv(sd, buf, sizeof(buf), 0) ;
    }

    closesocket(sd);

    exit(0)
}

```

20.7 Exemplo de Código para o Servidor

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT 5193
#define QLEN      6

int    visits    = 0 ;

main(argc, argv)
int    argc;
char   *argv[];
{
    struct hostent    *ptrh;
    struct protoent    *ptrp;
    struct sockaddr_in sad;
    struct sockaddr_in cad;
    int    sd, sd2;
    int    port;
    int    alen;
    char    buf[1000];

#ifdef WIN32
    WSDATA wsaData;
    WSStartup(0x0101, &wsaData);
#endif

    memset((char *)&sad,0,sizeof(sad));
    sad.sin_family = AF_INET;
    sad.sin_addr.s_addr = INADDR_ANY;

    if(argc>1) {
        port = atoi (argv[1]);
    } else {
        port = PROTOPORT
    }

    if (port >0)
        sad.sin_port = htons((u_short)port);
    else {
        fprintf(stderr, "bad port numer %s\n",argv[1]);
        exit(1);
    }
}
```

```

sd= socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd<0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

if ( bind (sd, (struct sockaddr *)&sad, sizeof (sad)) < 0 ) {
    fprintf(stderr, "bind failed \n");
    exit (1);
}

if (listen(sd, QLEN) <0 ) {
    fprintf(stderr, "listen failed\n");
    exit (1);
}

while (1) {
    alen = sizeof(cad);
    if (( sd2 = accept (sd, (struct sockaddr *)&cad, &alen)) <0) {
        fprintf(stderr, "accept failed\n");
        exit (1);
    }

    visits++;
    sprintf( buf, "This server has been contacted %d
times%s\n",visits, visits == 1?"":"s.");
    send(sd2,buf,strlen(buf), 0);
    closesocket (sd2);
}
}

```

20.8 Serviço Stream e Múltiplas Chamadas Recv

Embora o servidor faça apenas uma chamada *send* para enviar os dados, o código do cliente interage para receber os dados. Durante cada iteração, o código cliente utiliza o *recv* para obter os dados, as iterações param quando o cliente recebe uma mensagem que sinaliza o fim do arquivo.

Na maioria dos casos, o TCP no servidor irá colocar toda mensagem em um único segmento TCP, e então transmitir o segmento através do TCP/IP em um datagrama IP. De qualquer maneira, o TCP não garante que os dados serão enviados em um único segmento, e não garante também que cada chamada *recv* irá retornar exatamente a mesma quantidade de dados que o servidor enviou com a chamada *send*. Ao invés disso, o TCP apenas acerta que os dados chegarão ordenados e cada chamada *recv* retorna um ou mais octetos de dados. Consequentemente, o programa que chama *recv* deve estar preparado para fazer repetidas chamadas até que todos dados sejam extraídos.

20.9 Processos do Socket e Bloqueios

A maioria dos processos no socket API são *sincronizados* ou *bloqueados*, da mesma forma que a maioria de chamadas I/O. Isto é, quando um programa usa um processo socket, o programa suspende a computação até que o processo esteja completo. Não há tempo limite para a suspensão – a operação pode demorar um tempo arbitrário.

Para entender como clientes e servidores usam processos de bloqueio, primeiro considere o servidor. Após criar o socket, configurar uma porta e configurar o socket no modo passivo, o servidor usa o processo *accept*. Se um cliente já estiver requisitado uma conexão antes do servidor chamar o *accept*, a chamada será retornada imediatamente. Se o servidor alcança o *accept* antes que o cliente faça o pedido de conexão, o servidor estará suspenso até que a requisição chegue. De fato, o servidor gasta a maioria do tempo suspenso na rotina *accept*.

O cliente também pode ser bloqueado quando faz algumas chamadas no socket. Por exemplo, algumas implementações da biblioteca como *gethostbyname* enviam uma mensagem através da rede, logo espera uma resposta. Nestes casos, o cliente se mantém suspenso até que a resposta chegue. Similarmente, a chamada *connect* bloqueia até que o TCP possa realizar o three – way handshake para estabelecer a conexão.

A suspensão mais importante é a que acontece durante a transmissão de dados. Após a conexão ser estabelecida, o cliente usa o processo *recv*. Se nenhum dado for recebido na conexão, o cliente fica suspenso. Assim, se servidor tem uma fila de requisições de conexões o cliente permanece suspenso até que o servidor envie dados.

20.10 Tamanho do Código e Relatório de Erros

Como cada exemplo de programa tem um tamanho considerável, a maioria do tamanho é atribuída aos comentários. Por exemplo, remover linhas em branco e comentários pode reduzir o programa em até 40 por cento.

Muitas linhas do código checam erros. Em adição a verificação de valores especificados pelos argumentos das linhas de comando, o código checa o valor do retorno de cada procedimento, para ter certeza que a operação obteve sucesso. Erros não são esperados, então quando estes acontecem, o programador já configurou o programa para mostrar uma pequena mensagem de erro e ser finalizado. Temos 15 por cento do código para verificar erros.

20.11 Usando o Exemplo de Cliente com outro Serviço

Embora o exemplo do serviço apareça trivial, tanto o cliente e servidor podem ser utilizados com outros serviços. Por exemplo, o TCP/IP define o serviço DAYTIME para exibir data e horário. O DAYTIME utiliza o mesmo paradigma de iteração que nosso exemplo – o cliente realiza uma conexão ao servidor DAYTIME então mostra a informação enviada pelo servidor.

Para usar o exemplo do cliente com o serviço DAYTIME, o programa do cliente deve chamar dois argumentos, um especificar o host no qual o DAYTIME esta sendo executado, e outro argumento referente ao número da porta para o serviço DAYTIME, 13. Por exemplo, se o código do cliente é compilado e o resultado é colocado no arquivo executável com nome *client*, este pode ser usado para contatar o serviço DAYTIME em computadores em todo mundo.

```
$ client localhost 13
Mon Aug 17 20:58:08 2006
$ client sbforums.co.jp 13
Tue Aug 18 10:57:46 2006
$ client xx.lcs.mit.edu 13
Mon Aug 17 21:58:08 2006
```


20.12 Utilizando Outro Cliente para Testar o Servidor

O exemplo de servidor pode ser testado separadamente do cliente. Para fazer isso, podemos usar o programa cliente *Telnet* para contatar o servidor. O programa *Telnet* requer dois argumentos: o nome do computador no qual o servidor é executado e o número da porta do protocolo no servidor. Por exemplo, a seguinte saída é resultado do uso do *Telnet* para conectar ao servidor exemplo.

```
$ telnet xx.yy.nonexist.com 5193
Trying. . .
Connected to xx.yy.nonexist.com 5193
Escape character is '^]'.
This server has been contacted 4 times.
Connection closed by foreign host.
```

Embora a saída contenha cinco linhas, apenas quatro foram emitidas pelo servidor; as outras vem do programa cliente *Telnet*.

Capítulo 21 – Nomeando com o Domain Name System

21.1 Introdução

Foi descrito nos capítulos passados que a cada computador é designado um Endereço do Protocolo de Internet que aparece em cada datagrama IP enviado pelo computador. Embora Endereços IP sejam fundamentais para a pilha TCP/IP, qualquer pessoa que tenha utilizado a Internet sabe que usuários não precisam lembrar ou entrar endereços IP. Ao invés disso, aos computadores são também designados nomes simbólicos; softwares de aplicação permitem ao usuário entrar um dos nomes simbólicos quando estiver especificando certo computador. Por exemplo, quando for especificar um destino para uma mensagem de correio eletrônico, o usuário entra um texto que identifica o receptor para o qual a mensagem deveria ser entregue e o nome do computador do mesmo. Parecido a isso, o nome de um computador é embutido em um texto que o usuário entra para especificar um site na World Wild Web.

Embora nomes simbólicos sejam convenientes aos homens, eles são inconvenientes para os computadores. Devido à forma binária de um endereço IP ser mais compacta que os nomes simbólicos, ela requer menos computação para ser manipulada (por exemplo, ser comparada). Além disso, um endereço IP ocupa menos espaço na memória e requer menos tempo para ser transmitido através da rede do que um nome. Assim, embora softwares de aplicação permitam que os usuários entrem nomes simbólicos, os protocolos da camada de rede precisam de endereços – uma aplicação deve traduzir cada nome em um endereço IP equivalente antes de usá-lo para a comunicação. Na maioria dos casos, a tradução é feita automaticamente e os resultados não são revelados ao usuário – o endereço IP é mantido na memória e só é utilizado para enviar ou receber datagramas.

O software que traduz os nomes dos computadores em endereços de Internet equivalentes provê um exemplo interessante de interação Cliente-Servidor. O banco de dados de nomes não é mantido em um só computador. Ao invés disso, as informações sobre os nomes são distribuídas ao longo de um grande conjunto de potenciais servidores localizados por toda a Internet. Sempre que um programa de aplicação precisa traduzir um nome, a aplicação se torna cliente do sistema de nomenclatura. O cliente envia uma mensagem de pedido para o servidor de nomes, o qual encontra o endereço correspondente e envia uma mensagem de resposta. Se ele não puder responder a um pedido, o servidor de nomes se torna temporariamente cliente de outro servidor de nomes, até que um servidor seja encontrado e responda ao pedido.

Esse capítulo descreve a hierarquia de nomeação, a organização dos servidores de nomes e os detalhes da interação Cliente-Servidor entre eles. O capítulo também explica como o **caching** melhora a eficiência do sistema de nomenclatura e torna possível o funcionamento de um sistema distribuído de larga escala.

21.2 Estrutura de Nomes de Computadores

O esquema de nomenclatura usado na Internet é chamado de *Domain Name System* (DNS). Resumidamente, cada nome de computador consiste em uma seqüência de segmentos alfas-numéricos separados por pontos. Por exemplo, um computador em um departamento de Ciências da Computação na Universidade de Purdue tem o nome de domínio:

mordred.cs.purdue.edu

e um computador no Colégio de Engenharia na Universidade de Bucknell tem o nome de domínio:

www.eg.bucknell.edu

Nomes de domínio são hierárquicos, com a parte mais significativa do nome à direita. O segmento mais à esquerda do nome (*mordred* e *www* nos exemplos dados) é o nome de um computador individual. Os outros segmentos em um nome de domínio identificam o grupo ao qual pertence aquele nome. Por exemplo, o segmento *purdue* e *bucknell* cada um fornece o nome de uma universidade.

Quantos segmentos um nome tem e como eles são designados? A resposta é que além de especificar como os segmentos mais importantes são escolhidos, o DNS não especifica um número exato de segmentos em cada nome ou o que cada um desses segmentos representa. Ao invés disso, cada organização pode escolher quantos segmentos usar para computadores dentro dela e o que aqueles segmentos representam.

O DNS não especifica valores para o segmento mais significativo, o qual é chamado de *Nível Máximo* do DNS. A tabela abaixo lista os domínios Nível Máximo:

Nome de Domínio	Designado para
Com	Organizações Comerciais
Edu	Instituições Educacionais
Gov	Organizações Governamentais
Mil	Grupos Militares
Net	Major Network Support Center
Org	Organizações outras que não as acima
Arpa	Domínio ARPA temporário (ainda usado)
Int	Organizações Internacionais
Código do país	Um país

Mais sete domínios Nível Máximo foram propostos em 1997 para posteriormente dividir o espaço do nome e eliminar o excesso de nomes que tem o domínio comercial. Os nomes propostos foram *firm*, *store*, *web*, *arts*, *rec*, *info* e *nom*. *Arts* e *Rec* seriam propostos para acomodar organizações como museus de arte e web sites de recreação; *Nom* foi proposto para permitir que pessoas registrassem seus próprios nomes. Por muitas outras razões, entretanto, os novos nomes não receberam aprovação oficial.

Quando uma organização quer participar do DNS, a organização deve registrar um nome dentro de um domínio de Nível Máximo. A maioria das organizações escolhe registrar-se sob o domínio *com*. Por exemplo, a corporação nomeada *Foobar* poderia pedir para designar o domínio *foobar* sob o domínio de Nível Máximo *com*. Se o pedido fosse aprovado, a autoridade da Internet responsável pelos nomes de domínio designaria à Corporação Foobar o domínio:

`foobar.com`

Uma vez que um domínio tenha sido designado a uma organização, o sufixo é reservado à organização – nenhuma outra organização designará esse mesmo sufixo para seu nome. Por exemplo, uma vez que *foobar.com* tenha sido designado, outra organização chamada Foobar poderia solicitar por *foobar.edu* ou *foobar.org*, mas não por *foobar.com*.

21.3 Estrutura Geográfica

Além da estrutura organizacional habitual, o DNS permite que organizações usem um registro geográfico. Por exemplo, a Corporation For National Research Initiatives registrou o domínio:

`cnri.reston.va.us`

porque a corporação está localizada na cidade de Reston, Virginia nos Estados Unidos. Por isso, nomes de computadores na corporação terminam em *.us* ao invés de *.com*.

Alguns países adotaram uma combinação de nomes de domínios geográficos e organizacionais. Por exemplo, universidades no Reino Unido registram-se sob o domínio:

ac.uk

onde *ac* é uma abreviação para *Acadêmico*, e *uk* é o código oficial de países para o Reino Unido.

21.4 Nomes de Domínios dentro de uma Organização

Uma vez que uma organização tenha um domínio particular, a organização pode decidir como introduzir estruturas hierárquicas adicionais. Uma organização pequena não pode escolher hierarquias adicionais, enquanto que grandes organizações podem escolher vários níveis. Por exemplo, se a corporação Foobar for pequena, ela pode decidir que todos os nomes tenham a forma:

computador.foobar.com

onde *computador* denota o nome designado a cada computador. Entretanto, se Foobar é grande o suficiente para ter várias localizações, um nível da hierarquia de domínio pode ser usada para denotar a localização, resultando em nomes de domínio da forma:

computador.localização.foobar.com

Finalmente, se a Foobar tiver várias divisões em cada localização, um nível da hierarquia de nomes de domínio pode ser usada para distinguir essas várias divisões, resultando em nomes da forma:

computador.divisão.localização.foobar.com

Devido aos nomes de domínio serem conceitos lógicos, eles não precisam se adequar às localizações físicas. Por exemplo, Foobar pode possuir plantas em 5 localizações, com cada uma delas contendo computadores de duas divisões. A empresa poderia designar seus nomes de domínio por divisão ao invés de designar por localização. Como resultado, o nome não conterá a localização do computador:

computador.divisão.foobar.com

A liberdade da escolha da hierarquia da nomenclatura se estende aos grupos dentro da organização. Conseqüentemente, dois computadores em uma dada organização podem ter um número diferente de segmentos em seu nome de domínio. Por exemplo, suponha que uma grande divisão da Corporação Foobar faça doces, enquanto que uma pequena divisão faça sopas. Devido a última ser pequena, seu nomes de domínio podem ser da forma:

computador.soap.foobar.com

A divisão de doces pode adicionar mais um nível à hierarquia do domínio para distinguir as suas várias subdivisões que fazem cada tipo de doce:

computador.subdivisão.doces.foobar.com

21.5 O Modelo Cliente-Servidor do DNS

Uma das principais características do DNS é sua autonomia – o sistema foi projetado de modo a permitir que cada organização designe nomes aos seus computadores ou mude tais nomes sem se reportar a uma autoridade central. A hierarquia da nomenclatura ajuda a alcançar essa autonomia permitindo às organizações controlar todos os nomes com um sufixo particular. Devido a isso, a Universidade de Purdue é livre para criar ou mudar qualquer nome que termine com *purdue.edu*, enquanto que a IBM é livre para criar ou mudar nomes que terminem com *ibm.com*.

Além da hierarquia de nomenclatura, o DNS usa as interações Cliente-Servidor para dar assistência à autonomia. Na essência, o sistema inteiro de nomenclatura opera como um banco de dados grande e distribuído. A maioria das organizações que tem uma conexão com a Internet roda um servidor de nomes de domínio. Cada servidor contém informações que conecta ele a outros servidores de nomes de domínio; o grande conjunto de servidores resultante funciona como um banco de dados de nomes grande e coordenado.

Sempre que uma aplicação precisa traduzir um nome em um endereço IP, a aplicação se torna cliente do sistema de nomenclatura. O cliente coloca o nome a ser traduzido em uma mensagem de pedido DNS e a envia ao servidor DNS. O servidor extrai o nome do pedido, o traduz para um endereço IP equivalente e retorna o endereço resultante à aplicação em uma mensagem de resposta.

21.6 A Hierarquia de Servidores DNS

Os servidores DNS são organizados em uma hierarquia que é compatível à hierarquia de nomenclatura, com cada um sendo a autoridade por parte da hierarquia de nomenclatura. Um *Servidor Raiz* ocupa o topo da hierarquia e é uma autoridade para os domínios de Nível Máximo (por exemplo, *.com*). Embora ele não contenha todos os nomes de domínio possíveis, um servidor raiz contém informação sobre como alcançar outros servidores. Por exemplo, embora ele não conheça os nomes dos computadores da Corporação IBM, um servidor raiz sabe como alcançar um servidor que lide com pedidos para *ibm.com*. Similarmente, um servidor raiz sabe como alcançar o servidor que lide com os pedidos para *purdue.edu*.

Embora a hierarquia dos servidores DNS siga a hierarquia de nomenclatura, a estrutura não é idêntica. Uma corporação pode escolher entre colocar todos os seus nomes de domínio em um só servidor ou pode rodar vários servidores. Por exemplo, a corporação Foobar poderia alocar sua hierarquia de nomenclatura em dois servidores, sendo um para a linha de doces e outro para a linha de sopas.

Como pôde ser percebido no exemplo, os servidores não estão restritos a lidar com apenas um nível na hierarquia dos nomes de domínio, nem mesmo solicitados a conter vários níveis.

21.7 Arquiteturas de Servidores

Como uma arquitetura de um Servidor de Nomes de Domínio deveria ser escolhida? Em particular, quando que uma organização precisa de mais de um servidor? Em geral, a arquitetura na qual uma organização utiliza apenas um servidor é simples – uma organização pequena pode minimizar os custos colocando toda a sua informação sobre nomes de domínio em um só servidor.

Organizações maiores usualmente acham que apenas um servidor central não é suficiente por duas razões. Primeira, um único servidor e o computador no qual ele roda não podem lidar com pedidos arbitrários em alta velocidade. Segunda, grandes organizações freqüentemente encontram dificuldade em administrar um banco de dados central. O problema é especialmente grave porque a maioria dos softwares de DNS não provê atualizações automáticas – um humano deve entrar as mudanças e adições ao banco de dados manualmente. Devido a isso, o grupo de pessoas que é responsável por administrar o servidor central da organização deve ser coordenado para assegurar que apenas um administrador faça mudanças em um dado tempo. Se a organização rodar vários servidores, cada grupo pode administrar o

servidor que é a autoridade dos seus computadores. Mais importante ainda, cada grupo pode fazer mudanças no seu servidor de banco de dados sem uma coordenação centralizada.

21.8 Localidade de Referência e Múltiplos Servidores

O princípio da *Localidade de Referência* discutido no capítulo 7 aplica-se ao DNS e ajuda a explicar como vários servidores funcionam bem. O DNS segue o princípio da localidade de referência de duas maneiras. Primeiro, os usuários tendem a procurar os nomes de computadores locais mais frequentemente que nomes de computadores remotos. Segundo, um usuário tende a procurar pelo mesmo grupo de nomes de domínio repetidamente.

Ter vários servidores dentro de uma organização funciona bem porque um servidor pode ser colocado dentro de cada grupo. O servidor local é a autoridade para os nomes dos computadores de dentro daquele grupo. Devido ao DNS obedecer ao princípio da localidade, o servidor local pode lidar com a maioria dos pedidos. Por isso, em adição à facilidade de administrar, múltiplos servidores ajudam a balancear o tráfego, reduzindo assim os problemas de contenção que um servidor central pode causar.

21.9 Links entre Servidores

Embora o DNS permita o uso de vários servidores, uma hierarquia de domínios não pode ser dividida em servidores de forma arbitrária. A regra é: um servidor deve ser responsável que tem um dado sufixo.

Servidores no DNS estão ligados, tornando possível a um cliente encontrar o servidor correto apenas seguindo os links. Em particular, cada servidor é configurado para saber a localização dos servidores das subpartes da hierarquia. Por exemplo, o servidor para *.com* deve saber a localização do servidor para *foobar.com* e assim por diante.

21.10 Decidindo um Nome

A tradução de um nome de domínio em um endereço IP equivalente é chamada *Resolução de Nomes*, e se diz que um nome foi *resolvido* para um endereço. Softwares que fazem tal tradução são chamados de Softwares *Solucionadores de Nomes* (ou apenas *Solucionadores*).

Muitos sistemas operacionais provêm softwares solucionadores de nomes como uma biblioteca de rotinas que uma aplicação pode chamar.

Como os softwares solucionadores funcionam? Cada solucionador é configurado com o endereço de um servidor de nomes de domínios. Para se tornar um cliente do servidor DNS, o solucionador coloca o nome em uma mensagem de *Pedido DNS* e a envia ao servidor local. O solucionador então aguarda a mensagem de *Resposta DNS* do servidor que conterà a resposta. Embora um cliente possa escolher entre UDP ou TCP, quando estiver se comunicando com um servidor DNS, a maioria dos solucionadores são configurados para utilizar UDP porque ele requer menos cabeçalhos para um único pedido.

Quando um pedido especifica um nome do qual o servidor é responsável, ele responde ao pedido diretamente. Isto é, o servidor procura o nome em sua base de dados local e envia a resposta ao solucionador. Entretanto, quando um pedido chega com um nome fora do conjunto do qual aquele servidor é responsável, existiram mais interações cliente-servidor. O servidor temporariamente se torna cliente de outro servidor de nomes. Quando um segundo servidor retorna a resposta, o servidor original envia uma cópia da mesma de volta ao solucionador que enviou o pedido.

Como que um servidor DNS sabe qual outro servidor DNS é responsável por um dado nome? Ele não sabe. Entretanto, cada servidor sabe o endereço de um servidor raiz. Saber a localização de um servidor raiz é suficiente porque o nome pode ser traduzido a partir daí. Por

exemplo, suponha que um site remoto (por exemplo, de uma universidade) envie um pedido para o seu servidor local, *L*, com o nome:

`venus.walnut.candy.foobar.com`

O servidor *L* não é responsável por esse nome, então ele atua como cliente de outros servidores. Em um primeiro passo, o servidor *L* envia um pedido ao servidor raiz. O servidor raiz não é responsável por esse nome, mas a resposta dele fornece a localização do servidor responsável pela *foobar.com*.

Quando o servidor *L* recebe a resposta do servidor raiz, ele contata o servidor da *foobar.com*. Embora ele não seja responsável pelos nomes da subdivisão *walnut*, esse servidor principal da Foobar sabe a localização do servidor para tal subdivisão. Por isso, ele retorna uma resposta para informar o servidor *L*. Finalmente, o servidor *L* contata o servidor que é responsável pelos nomes com a forma:

`computador.walnut.candy.foobar.com`.

O servidor responsável envia uma resposta (chamada *Authoritative Answer*) ao servidor *L*, com o endereço IP para aquele nome ou com a indicação que aquele nome não existe.

Esse processo de procura através da hierarquia dos servidores a fim de encontrar o servidor que é responsável por um nome é chamado de *Pesquisa de Resolução Repetitiva* e é utilizada somente quando um servidor precisa traduzir um nome. Os solucionadores, os quais as aplicações chamam, sempre requerem uma *Pesquisa de Resolução Recursiva*. Isto é, ele requer a tradução completa – a resposta para um pedido recursivo é o endereço IP procurado ou uma instrução do servidor responsável que tal nome não existe.

21.11 Otimização do Desempenho do DNS

Medições mostram que o DNS como descrito acima é infelizmente ineficiente. Sem otimizações, o tráfego em um servidor raiz seria intolerável porque os servidores raiz receberiam um pedido toda vez que alguém mencionasse o nome de um computador remoto. Além disso, o princípio de localização sugere que um dado computador emitirá o mesmo pedido repetidamente – se um usuário entrar o nome de um computador remoto, é provável que o usuário procure por ele mais vezes.

Existem duas otimizações primárias usadas em no DNS: Réplicas e **Caching**. Cada servidor raiz deve ser replicado; muitas cópias do servidor existem ao redor do mundo. Quando um novo site se junta a Internet, ele configura seu servidor DNS local com uma lista de servidores raiz. O servidor do site utiliza o servidor que está respondendo mais rapidamente. Na prática, o servidor geograficamente mais próximo responde melhor.

O **Caching** do DNS é mais importante do que as réplicas de um servidor porque afeta mais o sistema. Cada servidor mantém um cache de nomes. Sempre que ele procura por um nome novo, o servidor copia o link do mesmo em seu cache. Antes de contatar outro servidor para pedir a localização de um nome, o servidor checa o seu cache. Se o cache contiver a resposta, o servidor a utiliza para gerar uma resposta.

O **Caching** funciona bem porque a tradução de nomes mostra uma forte tendência em direção da localidade de referência temporal. Isto é, em um dado dia, um usuário provavelmente procurará o mesmo nome repetidamente. Por exemplo, se um usuário manda um e-mail para um endereço, o usuário provavelmente receberá uma resposta do mesmo endereço e assim por diante. Quando uma aplicação procura por um nome pela primeira vez, o servidor DNS local guarda o link do mesmo em seu cache. O servidor pode então responder a pedidos subsequentes, retornando o link do seu cache, ao invés de contatar o servidor responsável novamente.

21.12 Tipos de Entradas do DNS

Cada entrada em um banco de dados DNS consiste em três itens: o nome de domínio, o tipo de registro e um valor. O tipo de registro especifica como o valor deve ser interpretado. Mais importante ainda, um pedido enviado para um servidor DNS especifica o nome de domínio e o tipo; o servidor só retorna um link que seja compatível com tipo enviado no pedido.

Discutimos o tipo usado para a ligação entre um nome de domínio e um endereço IP equivalente. O DNS classifica essa ligação de tipo A (o A vem de *address*). Ligações tipo A são comuns porque elas são utilizadas na maioria das aplicações. Por exemplo, quando um usuário fornece o nome de um computador a um programa de aplicação como o FTP, *ping* ou um browser de Internet, ele requer uma ligação do tipo A.

Além do tipo A, o DNS suporta vários outros tipos. Um tipo popular é o *MX* (abreviação de Mai eXchanger), o qual é utilizado para traduzir um nome de computador encontrado em um endereço de e-mail para um endereço IP. O software de e-mail especifica o tipo *MX* quando ele envia um pedido a um servidor DNS. A resposta que o servidor retorna é compatível com o tipo enviado no pedido. Por isso, um sistema de e-mail receberá uma resposta compatível com o tipo *MX*.

21.13 Apelidos Utilizando o Tipo CNAME

Outro tipo, *CNAME*, é especialmente útil. Entradas *CNAME* são análogas a um atalho de programa em um sistema de arquivos – a entrada fornece um apelido para outra entrada DNS. Para entender o quanto os apelidos são úteis, suponha que a corporação Foobar tenha dois computadores chamados *hobbes.foobar.com* e *calvin.foobar.com*. E ainda suponha que a Foobar rode um servidor Web e queira seguir a convenção de utilizar o nome *www* para o computador que roda o servidor Web da organização. Embora a organização possa renomear um deles (por exemplo, o computador *hobbes*), existe uma solução muito mais simples: a organização cria uma entrada *CNAME* para *www.foobar.com* que aponta para o computador *hobbes*, que se tornará o servidor Web. Sempre que for enviado um pedido para *www.foobar.com*, o servidor responderá com o endereço do computador *hobbes*.

O uso de apelidos é especialmente conveniente porque ele permite a uma organização a mudar um computador utilizado por um serviço particular sem mudar os nomes ou endereços dos mesmos. Por exemplo, a corporação Foobar pode mover seu servidor Web do computador *hobbes* para o computador *calvin* movendo o servidor e modificando o registro *CNAME* no servidor DNS – ambos os computadores retêm seus nomes e endereços IPs originais.

21.14 Uma Consequência Importante de Tipos Múltiplos

O sistema de tipos no DNS é conveniente porque ele permite a um administrador utilizar um único nome para diversos propósitos (por exemplo, direcionar o tráfego da Web para um computador, enquanto envia e-mail para um computador diferente). Entretanto, usuários são às vezes surpreendidos com a consequência de ter tipos específicos em pedidos DNS – um nome que funciona com uma aplicação pode não funcionar com outra. Por exemplo, pode ser possível enviar e-mail a um computador, enquanto que a tentativa de comunicação com mesmo usando um programa como *ping* ou *traceroute* resulta em uma mensagem de que aquele computador não existe. A aparente inconsistência ocorre porque o tipo DNS requerido pelo e-mail difere do tipo requerido por outras aplicações. Se o banco de dados do domínio contiver um registro do tipo *MX* para o dado nome, um pedido do sistema de e-mail obterá resposta. Entretanto, se o banco de dados não contiver um registro do tipo A para o mesmo nome, um pedido de um programa como o *ping* resultará em uma resposta negativa.

O sistema de tipos usados pelo DNS pode produzir resultados inesperados porque algumas aplicações estão configuradas para utilizar vários tipos. Por exemplo, os solucionadores utilizados em alguns programas de e-mail tentam dois tipos quando traduzem um nome. Primeiro ele tenta enviando ao servidor um pedido tipo *MX*. Se o servidor responder negativamente, ele tenta então um pedido tipo A. O esquema pode ajudar em situações em que

uma organização especifica um registro tipo *A* em seu banco de dados de domínios para um dado nome, mas falha em especificar também um registro do tipo *MX* para o mesmo nome.

21.15 Abreviações e o DNS

Devido ao fato de que usuários tendem a entrar nomes para computadores locais mais freqüentemente do que eles entram nomes para computadores remotos, abreviações para nomes locais são convenientes. Por exemplo, a corporação Foobar pode escolher permitir usuários a emitir o sufixo *foobar.com* quando entrar um nome de domínio.

Servidores de nomes de domínio não entendem abreviações – um servidor só responde a nomes completos. Para lidar com abreviações, solucionadores são configurados para tentar um conjunto de sufixos. Por exemplo, cada solucionador dentro da corporação Foobar pode ser programado para procurar um nome duas vezes: uma sem mudanças e uma com o sufixo *foobar.com* agregado. O esquema de sufixo permite ambos os nomes locais e remotos a serem lidados da mesma maneira. Dando o nome válido de um computador remoto, o servidor DNS retornará uma resposta válida, a qual o solucionador utilizaria. Entretanto, dado um nome local abreviado, o servidor DNS retornará uma mensagem de erro (porque tal nome não existe). O solucionador pode então tentar agregar cada um dos sufixos.

Usualmente, cada computador contém um software solucionador que todas as aplicações naquele computador utilizam. Devido ao fato de que as abreviações são lidadas pelo solucionador, o conjunto de abreviações permitidas em cada computador pode diferir.

Capítulo 22 – Descrição de Correio Eletrônico e Transferências

22.1 Introdução

O capítulo nesta seção define o modelo de interação entre cliente e servidor, formando a base para toda computação distribuída. Serão discutidos os conceitos fundamentais bem como o socket API que os programas do cliente e servidor utilizam para interagir.

Este capítulo continua a discussão da programação cliente e servidor examinando uma das aplicações mais utilizadas: o correio eletrônico, ou simplesmente *e-mail*.

22.2 O Paradigma do E-mail

Originalmente, o e-mail foi projetado como extensão direta das tradicionais notas de escritório. Isto é, o sistema original foi construído para permitir que uma pessoa se comunique com outra pessoa; um indivíduo cria uma mensagem e especifica outros indivíduos como receptores. O software de e-mail transmite uma cópia da mensagem para cada destinatário.

Os sistemas de e-mail evoluíram desde o projeto original, e são automatizados para permitir interações complexas. Em particular, como um programa de computador pode responder uma mensagem de e-mail e enviar uma resposta, o e-mail pode ser utilizado de várias maneiras. Por exemplo, uma companhia pode estabelecer um programa para responder automaticamente todas as requisições de informações que chegam num e-mail. O usuário envia uma requisição ao programa e recebe a informação desejada na resposta.

22.3 Caixas de E-mail e Endereços

O sistema de e-mail utiliza muitos termos e conceitos do ambiente tradicional dos escritórios. Antes de o e-mail ser enviado para um indivíduo, o emissor deve determinar uma caixa eletrônica de mensagens (*electronic mailbox*), ou caixa de e-mail. A caixa consiste em uma área passiva de armazenamento. Como a caixa postal convencional, a caixa de e-mail é privada – as permissões são configuradas para permitir ao software adicionar as mensagens que chegam à caixa, mas não permite ninguém exceto ao dono examinar e remover mensagens. Na maioria dos casos, a caixa de e-mail é associada com uma conta. Assim, o indivíduo que possuir múltiplas contas pode ter várias caixas de e-mail.

Cada caixa de e-mail é designada com um único *endereço de correio eletrônico* (endereço de e-mail); quando alguém envia uma nota, é usado o endereço de e-mail para especificar o destinatário. Um endereço de e-mail completo contém duas partes, a segunda especifica um computador e a primeira a caixa de mensagem neste computador. Seguem então o formato abaixo:

mailbox@computer

onde *mailbox* é string que denota a caixa de e-mail e *computer* é a string que define o computador onde a caixa de e-mail está localizada.

A divisão do endereço de e-mail em duas partes é importante porque alcança duas metas. Primeiro, a divisão permite cada sistema de computador designar identificadores de caixa

de e-mail independentes. Segundo, a divisão permite usuários em sistemas de computadores arbitrários trocarem mensagens. O software de e-mail no emissor usa a segunda parte para determinar qual computador contatar, e o computador receptor utiliza a primeira parte para determinar qual caixa de e-mail a mensagem será depositada. Assim a primeira parte é interpretada localmente.

Qual o formato é usado na primeira parte do endereço? A resposta depende do software de e-mail disponível no computador bem como o sistema operacional que está sendo usado. Alguns sistemas de software permitem ao administrador do sistema escolher nome das caixas de mensagem; enquanto outros sistemas permitem ao usuário escolher o identificador da caixa de mensagem, sendo o mesmo do login do usuário. Por exemplo, o endereço de e-mail do empregado João Queiroz Álvares na companhia Foobar pode ser:

joao_q_alvares@foobar.com

22.4 Formato da Mensagem de E-mail

A mensagem de e-mail tem um formato simples. A mensagem consiste em um texto ASCII que é separado em duas partes por uma linha em branco. A primeira parte, chamada *cabeçalho*, contém informações sobre a mensagem: o emissor, os destinatários, a data em que a mensagem foi enviada, e o formato da mesma. A segunda parte é chamada de *corpo* e contém o texto da mensagem.

Embora o corpo da mensagem possa ter um texto arbitrário, o cabeçalho segue uma forma normalizada que o software de e-mail usa quando envia ou recebe uma mensagem. Cada linha do cabeçalho começa com uma *palavra chave* seguida por dois pontos e informação adicional. A palavra chave conta ao software de e-mail como interpretar o restante da linha.

Algumas palavras chave são requisitadas em cada cabeçalho de e-mail; outras são opcionais. Por exemplo, cada cabeçalho deve ter a linha que começa com a palavra chave *To* e especificar a lista de destinatários. O restante do cabeçalho da linha *To* contém a lista de um ou mais endereços de e-mail, onde cada endereço corresponde a um destinatário. O software de e-mail coloca a linha que começa o cabeçalho com a palavra chave *From* seguida pelo endereço de e-mail do emissor da mensagem, no cabeçalho de cada mensagem.

Se o software de e-mail não entende a linha do cabeçalho, o software passa por esta linha sem fazer alterações. Assim, aplicativos que usam mensagens de e-mail para comunicar podem adicionar linhas ao cabeçalho da mensagem para controlar o processamento. Mais importante, o produtor do software pode construir o software que usa linhas do cabeçalho para adicionar funcionalidades – se a mensagem que chega contém uma linha especial de cabeçalho, o software sabe que a mensagem foi criada por uma determinada companhia.

22.5 Extensões do E-mail com Múltiplos Propósitos

O sistema original de e-mail foi projetado para lidar apenas com texto. O corpo da mensagem de e-mail era restrito a apenas caracteres ASCII e não poderiam conter bytes arbitrários.

Pesquisadores acharam o sistema de e-mail útil, assim inventaram esquemas para permitir que o e-mail seja usado para enviar dados arbitrários. Em geral, todos os esquemas codificam os dados na forma de texto, a qual pode ser enviada na mensagem de e-mail. Uma vez que chegue, o corpo da mensagem deve ser extraído e convertido de volta, formando um código binário.

Para ajudar a coordenação e unificar os vários esquemas que foram inventados para codificar dados binários, o IETF criou o *MIME*, o *Multipurpose Internet Mail Extensions*. O MIME não impõe uma única norma para codificar dados. Ao invés disso, o MIME permite ao

emissor e receptor que escolham um formato de codificação conveniente. Quando o MIME é usado, o emissor adiciona linhas ao cabeçalho para especificar que a mensagem segue o formato MIME bem como adiciona linhas no corpo do texto para especificar o tipo de dados e a codificação. Além de permitir ao emissor e receptor escolherem a codificação, o MIME permite ao emissor dividir a mensagem em várias partes e especificar a codificação de cada parte, independentemente. Assim, com o MIME, o usuário pode enviar um texto e anexar imagens. Quando o receptor visualiza a mensagem, o sistema de e-mail mostra a mensagem de texto e pergunta ao usuário como lidar com a imagem. Quando o usuário decide como lidar com o anexo, o software MIME decodifica automaticamente o anexo.

Para alcançar transparência na codificação e decodificação, o MIME adiciona duas linhas no cabeçalho do e-mail: uma declara que o MIME foi usado para criar a mensagem, e outra especifica como a informação MIME está disposta no corpo de texto. Por exemplo, as linhas de cabeçalho abaixo:

```
MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary=Mime_separator
```

Estas linhas especificam que a mensagem foi composta usando a versão 1.0 do MIME, e que a linha que contém *Mime_separator* irá aparecer no corpo antes de cada parte da mensagem. Quando o MIME é usado para enviar uma mensagem padrão de texto, a segunda linha se torna:

```
Content-Type: text/plain
```

A vantagem principal do MIME é a flexibilidade – a norma não especifica um esquema único de codificação que todos os emissores e receptores deverão usar. Ao invés disso, o MIME permite novos sistemas serem inventados a qualquer momento. O emissor e receptor podem usar um sistema convencional de e-mail para comunicar, fornecido pelo acordo no esquema de codificação e um nome único para o mesmo. Além do que, o MIME não especifica um valor para ser usado ao separar as partes da mensagem ou uma maneira de nomear o sistema de codificação usado. O emissor pode escolher qualquer separador, desde que este não já apareça no corpo de texto; o receptor usa então a informação no cabeçalho para determinar como decodificar a mensagem.

O MIME é compatível com sistemas de e-mail antigos. Em particular, um sistema de e-mail que transfere a mensagem não necessita entender que a codificação foi utilizada no corpo de texto ou a linha de cabeçalho MIME – a mensagem é tratada exatamente como qualquer outra mensagem de e-mail. O sistema de e-mail transfere a mensagem sem interpretar as linhas do cabeçalho e trata o corpo de texto como um único bloco de texto.

22.6 E-mail e Aplicativos

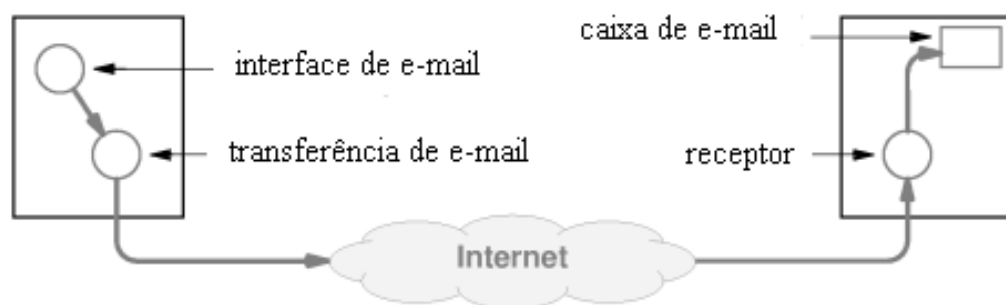
Como programas de computador podem processar e-mail, a distribuição e processamento de e-mails diferem das tradicionais notas de escritório. Em particular, é possível configurar um endereço de e-mail para corresponder a um programa, ao invés de uma caixa de mensagens no disco rígido. Quando um e-mail chega, este é destinado para o endereço, o sistema de e-mail envia uma cópia ao programa especificado ao invés de fazer uma cópia no disco.

Permitir que um programa enviasse e recebesse e-mails torna possível inventar maneiras interessantes de interação. Por exemplo, considere um aplicativo que permita ao usuário obter informações de um banco de dados. O usuário cria uma requisição em uma mensagem de e-mail, e envia a mensagem ao programa. O programa extrai a requisição da mensagem, acha a resposta no banco de dados e envia um e-mail com a resposta.

22.7 Transferência de E-mail

Após o usuário compor uma mensagem de e-mail e especificar todos os destinatários, o software de e-mail transfere uma cópia da mensagem para cada destino. Na maioria dos sistemas, duas peças separadas do software são necessárias. O usuário interage com a *interface de e-mail* do programa quando compõe ou lê as mensagens. O sistema base do e-mail é chamado de *transferência de e-mail* e lida com os detalhes de enviar uma cópia da mensagem para um computador remoto. Quando o usuário acaba de compor uma mensagem e a envia, a interface de e-mail coloca a mensagem em uma fila para que a transferência de e-mail lide com a mesma.

O programa de transferência de e-mail espera as mensagens serem colocadas em sua fila, e então transfere uma cópia da mensagem para cada destinatário. Enviar uma cópia da mensagem para o computador local é trivial, pois o programa de transferência pode adicionar a mensagem à caixa de mensagens do usuário. Já enviar uma cópia para um computador remoto é mais complexo. O programa de transferência de e-mail se torna um cliente que conecta ao servidor no computador remoto. O cliente envia a mensagem para o servidor, o qual passa uma cópia da mensagem para a caixa de mensagens do destinatário. A figura 27.3 abaixo ilustra a interação.



22.8 Simple Mail Transfer Protocol

Quando um programa de transferência de e-mail se conecta ao servidor no computador remoto, este forma uma conexão TCP através da qual se comunicam. Uma vez que a conexão é formada, os dois protocolos seguem o *Simple Mail Transfer Protocol* (SMTP) que permite ao emissor se identificar, especificar o receptor e transferir a mensagem de e-mail.

Embora a transmissão de e-mails pareça simples, o SMTP lida com muitos detalhes. Por exemplo, o SMTP necessita de entrega confiável – o emissor deve manter uma cópia da mensagem até que o computador remoto tenha armazenado a mensagem em um local não volátil. Além do mais, o SMTP permite ao emissor perguntar se determinada caixa de mensagem existe no computador remoto.

22.9 Otimização de Múltiplos Destinatário em um Computador

A figura 27.3 mostra uma mensagem de e-mail sendo enviada para uma única caixa de mensagens no computador remoto. Quando existem vários destinatários no mesmo computador

remoto, os programas de transferência são otimizados para lidar com todos estes destinatários ao mesmo tempo. Isto é feito criando apenas uma conexão ao servidor, e são especificados então todos os destinatários e transferido apenas uma cópia da mensagem. O servidor recebe apenas uma cópia da mensagem e a transfere a todos os destinatários.

Esta otimização é importante por duas razões. Primeiro, ela diminui drasticamente o consumo de banda para enviar uma mensagem de e-mail. Segundo, a otimização reduz o delay necessário para todos os usuários receberem uma cópia da mensagem – usuários que tenham caixa de mensagem no mesmo servidor irão receber a mensagem aproximadamente ao mesmo tempo. Mais importante, caso ocorra uma falha entre o cliente e servidor, ou todos os destinatários recebem ou nenhum destinatário recebe.

22.10 Listas e Remetentes de E-mails

Como os programas de computador podem processar mensagens de e-mail, estes podem ser manipulados para remeter mensagens. Por exemplo, muitos sistemas de e-mail incluem um *mail exploder* ou *mail forwarder*, isto é, um programa que pode remeter cópias de mensagens. O exploder usa um banco de dados para determinar como lidar com a mensagem. Normalmente chamada de lista de e-mails, cada entrada no banco de dados é um conjunto de endereços de e-mails.

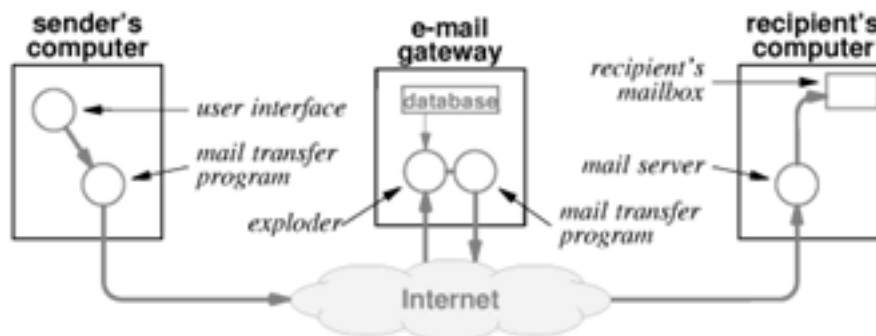
Quando um e-mail chega, o exploder examina o destinatário. Se o endereço do destinatário corresponder a uma lista no banco de dados, o exploder repassa uma cópia da mensagem para cada endereço na lista.

Os exploders tornam possível um grande grupo de pessoas se comunicarem via e-mail sem que o emissor especifique todos os destinatários. Para enviar a mensagem para todo o grupo, a mensagem é enviada para a lista de endereços. O exploder recebe a mensagem e a passa uma cópia da mensagem para cada membro da lista. Para receber um e-mail enviado pelo grupo, o indivíduo deve requisitar que seu endereço seja incluído na lista de endereços.

22.11 Mail Gateways

Um exploder de e-mail pode operar em qualquer computador, repassando mensagens de e-mail para uma lista grande de endereços, porém isso requer um tempo de processamento significativo. Assim, muitas organizações não permitem exploders ou listas de endereços muito grandes em computadores convencionais. Ao invés disso, a organização seleciona um pequeno grupo de computadores para executar o exploder e repassar mensagens. O computador dedicado a processar o e-mail é normalmente chamado de *mail gateway*, *email gateway* ou *e-mail relay*.

A figura 27.4 abaixo ilustra o processo de interação quando um usuário envia uma mensagem a uma lista de e-mail ou a um e-mail gateway.



Como a figura mostra, a mensagem passa através da Internet no mínimo duas vezes. Inicialmente uma única copia é enviada do computador emissor ao mail gateway. Após este consultar o banco de dados das listas de endereços, o exploder gera uma requisição para enviar cópias da mensagem. Um programa de transferência de e-mail no computador do gateway envia cada cópia da mensagem através da Internet para os computadores destinatários, onde um servidor armazena a mensagem na caixa de mensagem do destinatário.

22.12 Transmissão de E-mail e Endereços de E-mail

Como o endereço de e-mail do usuário engloba o nome do computador do usuário, uma organização que tem vários computadores pode ter uma variedade de endereços de e-mail. Por exemplo, considere o nome dos computadores para a hipotética Corporação Foobar. Se cada endereço de e-mail de um empregado engloba o nome do computador deles, os endereços de e-mail de dois empregados serão diferenciados:

e

smith@venus.walnut.candy.foobar.com
s_johnson@susie.soap.foobar.com

Saber o endereço de e-mail de um empregado não irá ajudar alguém adivinhar outro endereço de e-mail de outro empregado.

Para evitar confusão e tornar o endereço de e-mail para todos empregados uniforme, a organização pode escolher executar um mail gateway, e assinar todos os endereços relativos ao gateway. Por exemplo, se a Corporação Foobar nomear seu computador de gateway como:

foobar.com

a corporação pode designar cada endereço de e-mail dos empregados como:

empregado@foobar.com

Onde *empregado* é a string escolhida para designar um único empregado.

Como cada endereço tem o nome do gateway, a mensagem enviada para o empregado na Corporação Foobar irá chegar ao computador gateway. O banco de dados no gateway deve conter a entrada para cada empregado especificando qual a caixa de mensagem do mesmo em

um computador específico da corporação. Mais importante, o banco de dados no gateway permite endereços internos e externos serem diferenciados – endereços de e-mail externos podem ser independentes dos usados por sistemas particulares. Por exemplo, se o empregado John T. Doe usa o computador:

bubbles.soap.foobar.com

e tem o número de 7 dígitos 8456311 designado como identificador de caixa de mensagem, a entrada no gateway será:

<u>Lista</u>	<u>Conteúdo</u>
john_t_doe	8456311@bubbles.soap.foobar.com

Além de tornar os endereços de e-mail uniformes para toda corporação, o gateway permite uma flexibilidade maior. Como ninguém fora da corporação conhece o computador específico de um empregado ou o identificador da caixa de mensagem de um empregado, a corporação pode mover um empregado ou mudar o nome do computador do mesmo sem trocar o endereço de e-mail do empregado.

22.13 Acesso a Caixa de Mensagens

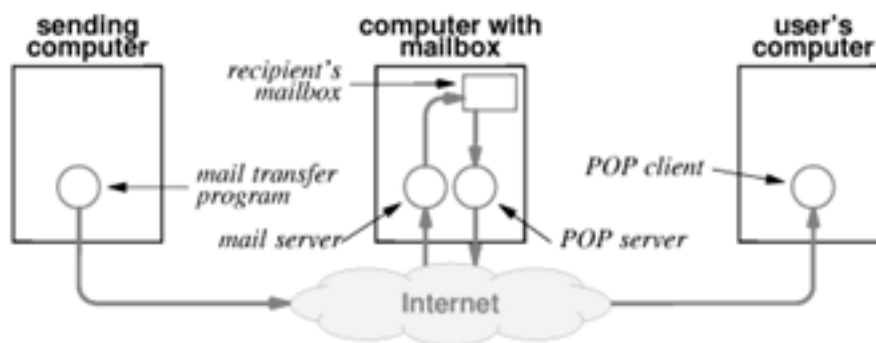
Em qual computador a caixa de mensagens deve ser instalada? Dada opção de escolha, a maioria dos usuários prefere ter a caixa de mensagens localizada no computador mais utilizado pelos mesmos.

Infelizmente, as caixas de mensagem não podem ser instaladas em todos os sistemas computacionais. Para entender o porquê, considere como a transferência de e-mail funciona. A caixa de mensagens é apenas uma área de memória em um disco; programas remotos não acessam a caixa de mensagens diretamente. Ao invés disso, cada sistema computacional que tenha uma caixa de mensagem deve executar um servidor de e-mail para aceitar as mensagens que chegam e armazená-las na caixa de mensagens correta. Em sistemas computacionais poderosos, o servidor de e-mail opera no plano de fundo, permitindo o usuário executar outros aplicativos ao mesmo tempo. Para permitir que múltiplos clientes enviem e-mails ao mesmo tempo, a maioria dos servidores faz um arranjo para executar várias cópias do programa servidor ao mesmo tempo.

Uma caixa de mensagens não pode ser instalada em um computador a não ser que o mesmo execute um servidor de e-mail. Por exemplo, um computador que não tem memória suficiente, não tem um sistema operacional que permita ao servidor ser executado no plano de fundo, ou não tem capacidade suficiente de CPU, não pode ser um servidor. Mais importante, é esperado que os servidores sejam executados continuamente – um computador que permaneça desligado ou desconectado da Internet por períodos extensos de tempo não será suficiente como servidor de e-mail. Assim um computador pessoal não é usualmente escolhido para executar um servidor de e-mail.

Usar um computador separado para o e-mail pode ser inconveniente. Por exemplo, alguém que use um computador pessoal em seu escritório irá achar um incômodo mover-se até outro computador para apenas ler um e-mail. Para evitar este problema, os protocolos TCP/IP incluem um protocolo que fornece acesso à caixa de mensagens. O protocolo permite a caixa de mensagem de um usuário residir em um computador que executa o servidor de e-mail e permite ao usuário acessar itens na caixa de mensagem de outro computador.

Conhecido como *Post Office Protocol* (POP), o protocolo requer que um servidor adicional seja executado no computador com a caixa de mensagens. O servidor adicional usa o protocolo POP. O usuário executa um software de e-mail que se torna o cliente do servidor POP para acessar o conteúdo da caixa de mensagens. A figura 27.5 abaixo ilustra o uso do POP.



Como a figura mostra, o computador que tem a caixa de mensagens deve executar dois servidores. O servidor convencional de e-mail aceita as mensagens de e-mail e as armazena na caixa de mensagens correta. O e-mail pode ir tanto à direção do emissor original ou do gateway de e-mail. O servidor POP permite ao usuário em um computador remoto acessar a caixa de mensagens.

Ambos servidores, o de e-mail e o POP, se comunicam através da internet, porém temos grandes diferenças. Primeiro, o servidor de e-mail utiliza o protocolo SMTP, enquanto que o servidor POP utiliza o protocolo POP. Segundo, o servidor de e-mail aceita uma mensagem de um emissor arbitrário, enquanto que o servidor POP permite apenas o usuário acessar sua caixa de mensagens após o mesmo fazer uma autenticação com senha. Terceiro, o servidor de e-mail pode transferir apenas mensagens de e-mail enquanto o servidor POP pode fornecer informações sobre o conteúdo da caixa de mensagens.

22.14 Conexão Dial-up e POP

Embora a figura 27.5 mostre um cliente acessando o servidor POP através da Internet, o POP é especialmente popular com os usuários que utilizam conexões dial-up de telefone. Nestes casos, o computador com a caixa de mensagens permanece conectado a Internet como a figura mostra, porém o computador do usuário não necessita estar permanentemente conectado a Internet. Ou seja, para receber um e-mail o usuário simplesmente forma uma conexão dial-up com o modem e seu telefone para o computador com a caixa de mensagens ou para outro computador conectado a Internet. Uma vez conectado a um computador na Internet, o usuário executa o cliente POP para conectar ao servidor e acessar o e-mail.

Capítulo 23 – Transferência de arquivos e Acesso Remoto de arquivos

23.1 Introdução

Os capítulos anteriores definiram o paradigma Cliente-Servidor e deram um exemplo de uma aplicação de rede. Este capítulo apresentará outro exemplo que pode transferir cópias de um arquivo de um computador para outro. Além da discussão sobre a interface de transferência de arquivos, o capítulo considera o acesso aos arquivos e explica como o software utilizado usa o paradigma Cliente-Servidor.

23.2 Transferência de Dados e Processamento Distribuído

Antes das redes, a transferência de dados de um computador para outro requeria o uso de mídias magnéticas como fitas e discos. Os dados eram escritos no meio magnético por uma aplicação em um dos computadores e esse meio era fisicamente movido até o outro computador. Para atravessar longas distâncias, a mídia era transportada (por exemplo, pelo correio). As redes de computadores reduziram substancialmente esse delay e tornaram possível uma nova forma de computação, na qual dois programas em dois ou mais computadores cooperam para executar uma solução. A saída de um programa se torna a entrada de outro.

A maior desvantagem da comunicação direta através de programas é que ela requer coordenação entre as aplicações dos vários computadores e tal coordenação pode ser difícil. Os administradores devem garantir que os computadores estejam rodando e que as aplicações estejam prontas ao mesmo tempo. Além disso, para executar com maior eficiência, os administradores devem prevenir que outros programas nos computadores utilizem grandes quantidades de processamento da CPU, de espaço da memória ou da largura de banda da rede.

Uma segunda desvantagem da comunicação direta está na inabilidade de recuperação de falhas. Se algum computador ou programa falhar a computação inteira deve ser recomeçada do início. Porque nenhum resultado intermediário é salvo, tais falhas podem ser especialmente onerosas se elas ocorrem tardiamente em uma longa computação (por exemplo, depois de várias horas de processamento).

23.3 Salvando Resultados Intermediários

Programadores e administradores observaram que uma simples técnica ajuda a superar ambas as desvantagens da comunicação direta. Ao invés de enviar dados através da rede assim que ela é gerada, cada aplicação guarda resultados intermediários em arquivos no disco. Isto é, uma aplicação lê entradas de um arquivo no disco, processa-a e escreve a saída em um arquivo. Os dados são transferidos de um arquivo de saída de um computador para um arquivo de entrada do outro.

As vantagens da utilização de arquivos intermediários são evidentes. Primeiro, devido aos dados serem salvos a cada passo do processamento, um administrador pode recuperar o mesmo depois de uma falha sem ter que refazer os passos já processados. Segundo, devido a isso permitir aos administradores agendar cada passo do processo independentemente, a solução do arquivo intermediário elimina a maioria dos problemas de logística presente quando são utilizados vários computadores simultaneamente. Se todos os computadores se conectam a uma rede compartilhada, usando arquivos intermediários é possível evitar que todos os passos do processamento compitam por faixas de banda da rede.

23.4 Generalização da Transferência de Arquivos

Assim que as aplicações de rede começaram a utilizar os arquivos intermediários, programadores escreviam códigos que transferiam arquivos completos de um computador para outro. Devido aos passos requeridos para transferir um arquivo para uma aplicação serem similares aos passos requeridos para transferir um arquivo para outro, programadores duplicaram o código, fazendo apenas pequenas modificações em nomes de arquivos ou o modo como os dados eram representados. Começou então a parecer que uma só aplicação geral poderia ser criada para trabalhar com muitas aplicações. O problema se tornou conhecido como problema de transferência de arquivos e o software que movia dados arbitrários de um computador para outro ficou conhecido como software de transferência de arquivos.

Para ser útil, softwares de transferência de arquivos deviam ser genéricos e flexíveis. Deviam permitir a transferência de arquivos arbitrários e acomodar diversos tipos de arquivos. Devido ao fato de que uma internet pode conectar diferentes tipos de sistemas, os softwares de transferência de arquivos devem adaptar as diferenças entre as maneiras como os sistemas guardam seus arquivos. Por exemplo, cada sistema operacional tem regras sobre nomes de arquivos – um nome que é válido para um sistema pode ser inválido para outro. Além disso, devido à maioria dos sistemas operacionais utilizarem as contas de login para definir a posse do arquivo, o dono em um sistema operacional pode não ter uma conta correspondente no outro computador. Finalmente, os softwares de transferência de arquivos devem adaptar outras pequenas diferenças na representação dos arquivos, informações sobre o tipo e mecanismos de proteção dos arquivos.

23.5 Interatividade e Conjunto de Paradigmas da Transferência

Alguns sistemas de transferência de arquivos usavam acesso **Batch** similar ao usado para transferência de e-mail. Um usuário abre um programa que o permite a formar um pedido que especifica detalhes como o computador remoto a contatar e os arquivos a serem transmitidos. O programa então coloca o pedido em uma fila e começa um programa de transferência. O programa de transferência contata um servidor em uma máquina remota e transfere os arquivos especificados. Se a máquina remota ou a rede estiverem indisponíveis, o programa de transferência automaticamente tenta refazer a transmissão mais tarde. Quando a transferência estiver completa, o programa de transferência avisa o usuário.

Transferência **batch** é mais útil quando a probabilidade de conseguir acesso ao computador remoto é pequena ou o tempo necessário para a transferência é grande. Por exemplo, a transferência **batch** funcionou muito bem no início porque as primeiras redes não eram confiáveis; não era estranho que uma rede não estivesse funcionando. A transferência **batch** também funciona bem quando um grande arquivo deve ser enviado por uma conexão com pequena largura de banda. Em tais circunstâncias, um programa de transferência **batch** lida com a transferência automaticamente, sem que o usuário precise esperar. O programa de transferência tenta contato com a máquina remota periodicamente. Quando a comunicação é bem sucedida, o programa espera pela transferência do arquivo por completo e termina a comunicação.

A transferência **batch** tem desvantagens também. Em situações onde redes e computadores permanecem disponíveis a maior parte do tempo, transferência interativa é mais conveniente. O usuário recebe informação continuamente assim que o programa de transferência contata o servidor no computador remoto e transfere o arquivo requerido. Por exemplo, o usuário descobre imediatamente se o nome da máquina remota contém um erro. Similarmente, um usuário toma conhecimento rapidamente sobre um nome de arquivo incorreto ou de um pedido que foi negado porque a proteção do arquivo remoto não permite a transferência. Finalmente, um usuário é notificado imediatamente quando a transferência estiver completa.

Um serviço de transferência de arquivo pode oferecer as vantagens de ambas as abordagens de transferências **batch** e interativa. Para tal, o serviço deve prover uma interface

que permita tanto um usuário humano ou um programa utilizem-no. Para operar interativamente, o usuário humano abre o programa, entra com um pedido e espera pela resposta. Para operar no modo **batch**, um programa de transferência administra a fila de pedidos. Quando ele lida com um pedido, o programa de transferência passa o pedido para o serviço e espera pelo término da transferência.

23.6 O Protocolo de Transferência de Arquivos (FTP)

A maioria dos serviços de transferência de arquivos pela Internet usa o *File Transfer Protocol (FTP)*. Um protocolo de propósito geral, FTP lida com muitos conceitos discutidos antes. O FTP permite transferir arquivos arbitrários e inclui um mecanismo que permite os arquivos a serem posse de algum usuário e a terem restrições de acesso. Mais importante ainda, ele esconde os detalhes dos sistemas do computador individual. FTP acomoda heterogeneidade – ele pode ser usado para transferir uma cópia de um arquivo entre dois computadores quaisquer.

O FTP está entre os protocolos de aplicação mais antigos ainda usados na Internet. Originalmente definido como parte dos protocolos ARPANET, o FTP veio antes do TCP e do IP. Assim que a pilha TCP/IP foi criada, uma nova versão do FTP foi desenvolvida para trabalhar com os novos protocolos da Internet.

O FTP está entre as aplicações mais utilizadas. Anteriormente na história da Internet, datagramas carregando transferências de arquivos tomava aproximadamente um terço de todo o tráfego da Internet; o tráfego gerado por serviços como e-mail e o DNS não chegavam nem perto de exceder o que era gerado pelo FTP.

23.7 Modelo Geral do FTP e Interface com o Usuário

O FTP é projetado para permitir o uso interativo ou **batch**. A maioria dos usuários executa o FTP interativamente – eles rodam um cliente FTP que estabelece a comunicação com o servidor especificado para transferir os arquivos. Entretanto, alguns softwares executam o FTP automaticamente, sem que o usuário tenha que interagir com um cliente FTP. Por exemplo, uma interface MIME para e-mails pode extrair e seguir uma referência FTP. Quando ela executa o FTP, um programa lida com todos os detalhes. O programa interage com o FTP e então informa ao usuário se a operação foi bem sucedida ou não; o programa esconde completamente a interface FTP do usuário.

Quando um usuário interage interativamente com o FTP, o usuário comunica-se com uma interface de comandos. O FTP executa um *prompt* no qual o usuário responde através da inserção do comando. O FTP executa tal comando e lança outro prompt.

O FTP tem comando que permitem ao usuário especificar um computador remoto, prover autorização, descobrir se arquivos remotos estão disponíveis e requerer a transferência de um ou mais arquivos. Alguns comandos FTP requerem pouco ou nenhum tempo para executar, enquanto que outros podem levar um tempo significativo. Por exemplo, pode levar muitos segundos para transferir a cópia de um arquivo grande.

23.8 Comando FTP

Embora o padrão do protocolo FTP especifique como o software de FTP em um computador interage com o software de FTP de outro, ele não especifica a interface com o usuário. Consequentemente, a interface disponível ao usuário pode variar de uma implementação de FTP para outra. Para ajudar a manter a similaridade entre os produtos, muitos fabricantes têm escolhido adotar a interface que primeiro apareceu em uma versão anterior do FTP escrita para o sistema BSD UNIX.

A interface BSD para o FTP suporta mais de 50 comandos individuais. Segue abaixo a lista dos nomes dos comandos:

!	cr	macdef	proxy	sendport
\$	delete	mdelete	put	status
account	debug	mdir	pwd	struct
append	dir	mget	quit	sunique
ascii	disconnect	mkdir	quote	tenex
bell	form	mls	Recv	trace
binary	get	mode	remotehelp	type
bye	glob	mput	rename	user
case	hash	nmap	reset	verbose
cd	help	ntrans	rmdir	?
cdup	lcd	open	runique	
close	ls	prompt	send	

A lista de comandos pode parecer grande demais para um iniciante por duas razões. Primeira, a interface BSD contém opções que são raramente implementados (por exemplo, o comando *proxy* que permite comunicação simultânea com dois sites remotos). Segunda, a interface provê muitas opções que lidam com detalhes antigos, alguns dos quais se tornaram irrelevantes. Por exemplo, quando a interface foi definida, já existia uma versão do FTP para um sistema operacional chamado *TENEX*. Entretanto, devido à representação usada pelo TENEX diferir da representação de arquivos usada pelo UNIX, foi introduzido um esquema para traduzir uma representação na outra e para isso foi criado o comando *tenex*, o qual executava tal tradução. Hoje em dia, outros sistemas operacionais têm tomado o lugar do TENEX, fazendo com que esse comando ficasse inutilizado.

Similarmente, antigos sistemas computacionais implementavam o **carriage control** colocando um caractere extra no início de cada linha do arquivo de texto para especificar o espaçamento da impressora. Existem muitos padrões de **carriage control**. Quando a interface BSD foi projetada, opções foram incluídas a fim de permitir ao usuário especificar que um arquivo particular incluía **carriage control**. A intenção era ter uma tradução do FTP entre os diferentes padrões de **carriage control**. Devido aos arquivos serem agora armazenados raramente com **carriage control**, somente alguns fabricantes projetaram essa opção.

O FTP também acomoda diversas representações de arquivos de texto. Por exemplo, alguns sistemas operacionais usam um único caractere de preenchimento de linha para separar as linhas de um texto. Outros sistemas usam uma seqüência de dois caracteres que consiste em um **carriage return** seguido por um preenchimento de linha. O comando *cr* permite ao usuário especificar qual representação uma máquina remota usa, tornando possível ao FTP traduzir as representações local e remota.

Uma forma final de complexidade aparece porque a interface BSD também inclui apelidos (por exemplo, muitos nomes para a mesma função). Por exemplo, ambos os comandos *close* ou *disconnect* podem ser utilizados para terminar uma conexão com um computador remoto. Similarmente, *bye* e *quit* podem ser utilizados para sair do programa FTP e *help* e *?* podem ser utilizados para obter uma lista dos comandos disponíveis.

23.9 Conexões, Autorizações e Permissões de Arquivos

Felizmente, a maioria dos usuários precisa apenas de alguns comandos FTP para transferir um arquivo. Após iniciar o programa FTP, o usuário deve inserir o comando *open*

antes de qualquer arquivo a ser transferido. Esse comando precisa que o usuário forneça um nome de domínio de um computador remoto para então formar uma conexão TCP com o computador.

Conhecido como uma *Conexão de Controle*, a conexão TCP a uma máquina remota é usada para enviar os comandos. Por exemplo, uma vez que uma conexão tenha sido aberta, o FTP requer do usuário o fornecimento da autorização para o computador remoto. Para tal, o usuário deve entrar o nome de login e a senha: a maioria das versões do FTP lança um prompt para cada um. O nome de login, o qual deve ser correspondente a uma conta válida no computador remoto, determina quais arquivos podem ser acessados. Se o usuário FTP fornecer o nome de login *smith*, ele terá a mesma permissão de acesso aos arquivos que alguém que entre diretamente no computador remoto com o mesmo login.

Depois de abrir uma conexão de controle e obter autorização, o usuário pode transferir arquivos. A conexão de controle permanece aberta pelo tempo que for necessário. Quando o usuário termina de acessar um computador, ele deve entrar com o comando *close* para terminar a conexão de controle. Fechar a conexão de controle não termina o uso do programa FTP – o usuário pode abrir uma nova conexão de controle com outro computador.

23.10 Acesso Anônimo de Arquivos

Embora o uso de nomes de login e senhas possam ajudar a manter a segurança dos arquivos de acessos não autorizados, tal autorização pode também ser inconveniente. Em particular, pedir que cada usuário tenha um nome de login e senha válidos torna difícil permitir acessos arbitrários. Por exemplo, suponha que uma corporação encontre um problema em um dos programas que vende. Ela pode criar um arquivo de mudanças e torná-lo disponível para qualquer pessoa.

Para permitir o acesso arbitrário, muitos sites seguem a convenção de estabelecer uma conta especial usada pelo FTP. A conta, a qual tem o nome de login *anônimo*, permite a um usuário arbitrário acesso mínimo aos arquivos. Sistemas mais antigos usavam a senha *guest* para o acesso anônimo. Versões mais recentes do FTP freqüentemente pedem que o usuário envie o seu endereço de e-mail como sua senha, possibilitando ao programa FTP remoto a enviar ao e-mail do usuário quaisquer problemas que ocorram. Em ambos os casos, o termo *FTP anônimo* é usado para descrever o processo de obtenção de acesso com o nome de login *anônimo*.

23.11 Transferência de Arquivos em Ambas as Direções

Interessantemente, o FTP permite a transferência de arquivos em ambas as direções. Depois que um usuário estabelece a conexão com o computador remoto, ele pode obter uma cópia de um arquivo remoto ou transferir a cópia de um arquivo local para a máquina remota. É claro que, tais transferências estão sujeitas às permissões de acesso – o computador remoto pode estar configurado para proibir a criação de novos arquivos ou mudanças nos já existentes e o computador local com as restrições convencionais de restrições de acesso de cada usuário.

O usuário entra com o comando *get* ou *mget* para resgatar uma cópia de um arquivo remoto. O comando *get*, mais freqüentemente utilizado, lida com a transferência de um só arquivo por vez. O *get* requer do usuário a especificação do nome do arquivo remoto a ser copiado; o usuário pode entrar um segundo nome se o arquivo local no qual a cópia deveria ser colocada tiver um nome diferente do arquivo remoto. Se o usuário não fornecer o nome do arquivo remoto na linha em que colocar o comando, o FTP exige do usuário a solicitação do nome. Uma vez que ele sabe o nome do arquivo, o FTP executa a transferência e informa ao usuário quando ela estiver completa. O comando *mget* permite ao usuário o pedido de vários arquivos no mesmo pedido. O usuário especifica uma lista de arquivos remotos e o FTP transfere cada arquivo para o computador do usuário.

Para transferir uma cópia de um arquivo local para o computador remoto, o usuário entra o comando *put*, *send* ou *mput*. *Put* e *send* são dois nomes para o mesmo comando que transfere um só arquivo. Assim como com o *get*, o usuário deve entrar o nome de um arquivo no

computador local e pode também entrar um nome de arquivo diferente para usar no computador remoto. Se nenhum nome de arquivo estiver presente na mesma linha de comando, o FTP pede-o ao usuário. O comando *mput* é análogo ao *mget* – ele permite ao usuário pedir a transferência de vários arquivos com um só comando. O usuário especifica uma lista de arquivos e o FTP transfere cada um.

23.12 Expansão por Caractere Curinga em Nomes de Arquivos

Para facilitar aos usuários a especificação de um conjunto de arquivos, o FTP permite que o sistema do computador remoto execute uma expansão de nomes de arquivos tradicional. O usuário entra com uma abreviação, a qual o FTP expande para produzir um nome de arquivo válido. Nas abreviações, o caractere curinga toma o lugar de zero ou mais caracteres. Muitos sistemas operacionais utilizam o asterisco (*) como caractere curinga. Em tais sistemas, a abreviação:

*li**

encontraria todos os arquivos que começam com o prefixo *li*. Além disso, se o computador remoto tiver os seis arquivos:

dark light lone crab link tuft

o FTP expandiria a abreviação *li** para dois nomes: *light* e *link*. A expansão de nomes de arquivos pode ser especialmente útil com os comandos *mget* e *mput* porque a expansão torna possível a especificação de um grande conjunto de arquivos sem ter que entrar cada nome de arquivo explicitamente.

23.13 Tradução de Nomes de Arquivos

Devido ao FTP poder ser usado entre sistemas operacionais diferentes, o software deve acomodar as diferenças entre a sintaxe dos nomes de arquivo. Por exemplo, alguns sistemas operacionais limitam os nomes de arquivos a letras maiúsculas, enquanto que outros permitem uma mistura de letras minúsculas e maiúsculas. Similarmente, alguns sistemas operacionais permitem que o nome do arquivo contenha até 128 caracteres, enquanto que outros restringem os nomes a 8 ou menos caracteres.

As diferenças entre os nomes de arquivos podem ser especialmente importantes quando usados com abreviações (por exemplo, em um comando *mget* ou *mput*). Em tais comandos, o usuário pode especificar uma abreviação que o FTP expande em uma lista de arquivos. Infelizmente, o nome de arquivo que é válido em um computador pode ser ilegal no outro.

Para lidar com as incompatibilidades ao longo dos sistemas operacionais, a interface BSD para o FTP permite que o usuário defina regras que especifiquem como traduzir um nome de arquivo quando o mesmo for transferido para um novo sistema operacional. Dessa forma, o usuário pode especificar que o FTP deve traduzir cada letra minúscula para a sua maiúscula equivalente.

23.14 Mudando de Diretório e Listando o Conteúdo

Muitos computadores têm um sistema hierárquico de arquivos que posiciona cada arquivo dentro de um *diretório* ou *pasta*. A hierarquia existe porque um diretório pode conter outros diretórios assim como arquivos. O FTP suporta um sistema hierárquico de arquivos incluindo o conceito do diretório atual – a qualquer momento, os pontos local e remoto de uma conexão de controle estão cada um em um diretório específico. Todos os nomes de arquivos são

interpretados no diretório atual e todas as transferências de arquivos afetam o diretório atual. O comando *pwd* pode ser usado para encontrar o nome do diretório remoto.

Os comandos *cd* e *cdup* permitem ao usuário controlar o diretório que o FTP está usando no computador remoto. O comando *cd* muda para um diretório específico; um nome de diretório válido deve ser fornecido na linha de comando. O comando *cdup* muda para um diretório superior (por exemplo, move um nível para cima na hierarquia). O *cdup* é conveniente porque o nome necessário para se referir a um diretório superior pode não ser óbvio. Por exemplo, o sistema UNIX usa o nome “..” para se referir ao diretório superior do diretório atual.

Para determinar um conjunto de arquivos disponíveis em um dado diretório no computador remoto, o usuário pode entrar com o comando *ls*. Ele produz uma lista dos nomes de arquivos, mas não diz nada sobre o tipo ou conteúdo de cada arquivo. Por isso, um usuário não pode determinar quando que um dado nome se refere a um arquivo de texto, a uma imagem gráfica ou outro diretório.

23.15 Tipos de Arquivos e Modos de Transferência

Embora as representações de arquivos utilizadas por dois sistemas operacionais possam diferir, o FTP não tenta lidar com todas as representações possíveis. Ao invés disso, o FTP define dois tipos básicos de transferência que englobam a maioria dos arquivos: textual e binária. O usuário deve selecionar o tipo de transferência, que permanece o mesmo durante toda a transferência do arquivo.

A transferência textual é utilizada para arquivos de texto básicos. Um arquivo de texto contém uma seqüência de caracteres separados em linhas. A maioria dos sistemas operacionais utiliza um conjunto de caracteres *ASCII* ou *EBCDIC* para representar os caracteres em um arquivo de texto. O usuário, que sabe o conjunto de caracteres usado pelo computador remoto, pode usar o comando *ascii* ou *ebcdic* para especificar a transferência textual e usa o FTP para traduzir entre os conjuntos de caracteres dos computadores local e remoto quando copiando um arquivo.

A única alternativa para a transferência textual no FTP é a transferência binária, a qual deve ser usada para todos os arquivos que não sejam texto. Por exemplo, um arquivo de áudio, uma imagem gráfica ou uma matriz de números de ponto flutuante devem ser transferidos no modo binário. O usuário deve colocar o comando *binary* para colocar o FTP em modo binário.

O FTP não interpreta o conteúdo de um arquivo transferido no modo binário e não traduz entre uma representação e outra. Ao invés disso, a transferência binária simplesmente produz uma cópia – os bits de um arquivo são reproduzidos sem mudanças. Infelizmente, uma transferência binária pode não produzir o resultado esperado. Por exemplo, considere um arquivo de números de ponto flutuante de 32 bits. No modo binário, o FTP copiará os bits do arquivo de um computador para outro sem mudanças. Mas se as representações de ponto flutuante usadas pelos computadores diferir, os computadores interpretaram os valores no arquivo diferentemente.

23.16 Exemplos Utilizando FTP

Ver o exemplo no livro do Comer

23.17 Verbose Output

Toda saída do FTP começa com um número de três dígitos que identifica a mensagem. Por exemplo, o FTP coloca 226 no começo de uma mensagem que informa o usuário que a transferência foi completada. Similarmente, o FTP remoto coloca 221 no começo de mensagens que confirmam um pedido de fechamento de conexão de controle.

O usuário pode escolher entre o FTP fornecer mensagens com informações (modo *verbose*) ou omitir tais mensagens e exibir apenas os resultados (modo *silencioso*). Por exemplo,

no modo **verbose**, o FTP calcula e mostra o número total de bytes de uma transferência, o tempo necessário para a sua conclusão e o número de bytes transferidos por segundo. Para controlar o modo, o usuário entra o comando *verbose*. Tal comando é como uma chave que reverte o modo toda vez que é executado. Por isso, ao entrar o comando *verbose* uma vez, o modo **verbose** é desligado, e ao entrá-lo mais uma vez, liga o modo **verbose** mais uma vez.

23.18 Interações Cliente-Servidor no FTP

Como outras aplicações de rede, o FTP utiliza o paradigma cliente-servidor. O usuário roda uma aplicação FTP local, a qual interpreta seus comandos. Quando o usuário entra com o comando *open* e especifica um computador remoto, a aplicação local se torna um cliente FTP que utiliza o TCP para estabelecer uma conexão de controle com um servidor FTP em um computador específico. O cliente e o servidor usam o protocolo FTP quando estão se comunicando através da conexão de controle. Isto é, o cliente não transmite os caracteres digitados diretamente para o servidor. Ao invés disso, quando o usuário entra um comando, o cliente interpreta o comando. Se o comando precisar de interação com o servidor, o cliente forma um pedido usando o protocolo FTP e o envia ao servidor. O servidor usa o protocolo FTP quando envia a resposta.

23.19 Conexões de Controle e de Dados

O FTP usa uma conexão de controle apenas para enviar comandos e receber respostas. Quando ele transfere um arquivo, não envia os dados pela conexão de controle. Ao invés disso, o cliente e o servidor estabelecem uma *conexão de dados* separada para cada transferência de arquivo, a utiliza para enviar um arquivo e então a fecha. Se o usuário começa uma nova transferência, o cliente e o servidor estabelecem uma nova conexão. Para evitar conflitos entre as duas conexões, o FTP utiliza números de portas diferentes para cada um.

Embora as conexões de dados apareçam e desapareçam frequentemente, a conexão de controle permanece por toda a sessão. Por isso, enquanto uma transferência está ocorrendo, o cliente e o servidor têm duas conexões abertas: a de controle e a de dados para a transferência. Uma vez que a transferência estiver completa, o cliente e o servidor fecham a conexão de dados e continuam a utilizar a conexão de controle.

23.20 Conexões de Dados e Fim do Arquivo

A utilização de conexões separadas para a transferência e para o controle tem diversas vantagens. Primeira, esse esquema mantém os protocolos mais simples e torna a implementação mais fácil – dados de um arquivo nunca serão confundidos com comando FTP. Segunda, devido à conexão de controle se manter por toda a sessão, ela pode ser utilizada durante a transferência (por exemplo, um usuário pode enviar um pedido para abortar a transferência). Terceira, o cliente e o servidor podem usar uma condição de fim de arquivo na conexão de dados para informar a outra parte quando todos os dados foram recebidos.

Utilizar a condição de fim de arquivo para terminar a transferência é importante porque permite que o arquivo mude de tamanho durante a transferência. Por exemplo, considere o caso onde uma aplicação está escrevendo em um arquivo no computador servidor enquanto o FTP está enviando uma cópia de um arquivo para o cliente. Devido à transferência do arquivo ser feita em uma conexão diferente, o servidor não precisa dizer ao cliente o tamanho do arquivo. Ao invés disso, o servidor abre uma conexão, lê os dados do arquivo e os envia através da nova conexão. Quando o servidor chega ao fim do arquivo, ele fecha a conexão de dado, fazendo com que o cliente receba uma condição de fim de arquivo. Devido ao servidor não dizer ao cliente quantos dados esperar, o arquivo pode aumentar durante a transferência sem causar problemas.

23.21 Protocolo de Transferência de Arquivos Trivial

Os protocolos de Internet incluem um segundo serviço de transferência de arquivos conhecido por *TFTP*, o *Protocolo de Transferência de Arquivos Trivial*. O TFTP difere do FTP em muitas maneiras. Primeira, a comunicação entre um cliente e um servidor TFTP usa UDP ao invés do TCP. Segunda, TFTP só suporta transferência de arquivos. Isto é, o TFTP não suporta interação e não possui um grande conjunto de comandos. Mais importante ainda, o TFTP não permite ao usuário listar os arquivos de um diretório ou perguntar a um servidor quais os nomes dos arquivos disponíveis. Terceira, o TFTP não tem autorização. O cliente não precisa enviar um nome de login e senha; o arquivo pode ser transferido apenas se suas permissões permitirem acesso global.

Embora o TFTP seja menos poderoso que o FTP, ele tem duas vantagens. Primeira, o TFTP pode ser usado em ambientes onde o UDP está disponível, mas o TCP não. Segundo, o código para o TFTP requer menos memória que o código para o FTP. Embora essas vantagens não sejam importantes em computadores de propósito geral, ele pode ser importante em um computador pequeno ou um dispositivo de hardware de propósito específico.

O TFTP é especialmente útil para **bootstrapping** um dispositivo de hardware que não tenha um disco onde armazenar o software do sistema. Tudo o que o dispositivo precisa é de uma conexão com a rede e uma pequena quantidade de memória *Read-Only (ROM)* na qual o TFTP, UDP e o IP estão configurados. Quando ele é ligado, o dispositivo executa o código em sua ROM, o qual envia um broadcast de um pedido TFTP para a rede local. O servidor TFTP na rede é configurado para responder o pedido enviando um arquivo que contém o programa binário a ser executado. O dispositivo recebe o arquivo, o carrega em sua memória e começa a executar o programa.

O **bootstrapping** através da rede adiciona flexibilidade e reduz custos. Devido a um servidor separado existir para cada rede, um servidor pode ser configurado para suprir a versão do software que está configurado para a rede. O custo é reduzido porque o software pode ser mudado sem ter que mudar o hardware. Por exemplo, o fabricante pode soltar uma nova versão do software para o dispositivo sem alterar o hardware ou ter que instalar uma nova ROM.

23.22 Sistema de Arquivos de Rede

Embora ele seja útil, a transferência de arquivos não é ótima para todas as transferências de dados. Para entender o porquê, considere uma aplicação rodando no computador *A* que precisa adicionar uma mensagem de uma linha a um arquivo no computador *B*. Antes que a mensagem possa ser anexada, um serviço de transferência de arquivo requer que o arquivo inteiro seja transferido do computador *B* para o *A*. Então, o arquivo atualizado deve ser transferido do *A* de volta para o *B*. Transferir um arquivo grande de um computador para o outro duas vezes introduz grandes delays e consome banda da rede. Mais importante ainda, a transferência é desnecessária porque os conteúdos do arquivo nunca são utilizados no computador *A*.

Para acomodar aplicações que só precisam ler ou escrever parte de um arquivo, o TCP/IP inclui o *serviço de acesso a arquivos*. Diferente de um serviço de transferência de arquivos, o serviço de acesso a arquivos permite que um cliente remoto copie ou mude pequenos pedaços sem ter que copiar o arquivo inteiro.

O mecanismo de acesso a arquivos usado com o TCP/IP é conhecido como *Network File System (NFS)*. O NFS permite que uma aplicação abra um arquivo remoto, mova-se para uma posição específica dentro dele e leia ou escreva dados começando dessa posição. Por exemplo, para adicionar dados a um arquivo usando NFS, uma aplicação vai para o fim do arquivo e escreve os dados. O software NFS cliente envia os dados para o servidor onde o arquivo é guardado com um pedido de escrita de dados para ele. O servidor atualiza o arquivo e envia uma mensagem. Apenas os dados que estão sendo lidos ou escritos viajam através da rede: uma pequena quantidade de dados pode ser adicionada a um grande arquivo sem que o arquivo inteiro seja copiado.

Em adição a redução de banda de rede requerida, o esquema de acesso a arquivos usado pelo NFS permite acesso compartilhado a arquivos. Um arquivo que está em um servidor NFS pode ser acessado por vários clientes. Para prevenir que outros clientes interfiram com atualizações, o NFS permite que um cliente trave o arquivo. Quando ele termina de fazer as modificações, destrava o arquivo, permitindo o acesso dos outros.

A interface para o NFS é diferente da interface para o FTP. Ao invés de criar uma aplicação cliente separada, o NFS é integrado em um sistema de arquivos de computador. Tal integração é possível porque o NFS provê as operações convencionais de arquivos como *abrir*, *ler* e *escrever*. Para configurar o NFS, um diretório especial é criado no sistema de arquivos do computador e associado a um computador remoto. Quando um programa de aplicação realiza uma operação em um arquivo nesse diretório, o software NFS cliente utiliza a rede para realizá-la em um arquivo no sistema de arquivos remoto. Por isso, uma vez que o NFS tenha sido instalado e configurado, o sistema de arquivos do computador conterá diretórios que correspondem a sistemas de arquivos remotos – qualquer operação realizada em um arquivo dentro de um diretório especial ocorre no arquivo remoto correspondente. A principal vantagem em tal esquema é a flexibilidade: qualquer programa de aplicação pode ser usado em um arquivo remoto porque a aplicação precisa apenas realizar operações padrão de arquivos.

Capítulo 24 – Páginas da World Wild Web e Navegação

24.1 Introdução

Capítulos anteriores mostraram exemplos de serviços de rede providos por programas de aplicação. Os capítulos descrevem as funções dos clientes e servidores, e mostra como cada serviço usa o paradigma cliente-servidor. Esse capítulo continua a discussão considerando a World Wide Web e **Web Browsing** interativo.

Depois de descrever o modelo do hipertexto e o conceito geral da Web, o capítulo examinará a estrutura do software do browser. Mostra como o browser reage a uma seleção do usuário tornando-se um cliente, contatando o servidor para obter a informação e então disponibilizá-la. O capítulo também explica como uma URL é colocada em um documento e descreve como um browser usa uma URL para determinar qual protocolo utilizar e qual servidor contatar.

24.2 Interface de Browser

A *World Wild Web (WWW)* é um local onde se encontra informações em larga escala e on-line as quais podem ser procuradas por usuários através de um programa de aplicação interativo chamado *browser*. A maioria dos browsers tem uma interface *point and click* – o browser mostra a informação em uma tela de computador e permite que o usuário navegue com o mouse. A informação disponibilizada inclui texto e gráficos. Ainda mais, algumas das informações na tela são destacadas para indicar que um item é *selecionável*. Quando o usuário posiciona o cursor sobre um item selecionável e clica, browser mostra novas informações correspondentes ao item selecionado.

24.3 Hipertexto e Hipermídia

Tecnicamente, a Web é um sistema de *hipermídia* distribuída que suporta acesso interativo. O sistema de hipermídia provê uma extensão direta do sistema tradicional de *hipertexto*. Em ambos os sistemas, a informação é armazenada como um conjunto de documentos. Além da informação básica, o documento pode conter ponteiros para outros documentos nesse conjunto. Cada ponteiro é associado a um item selecionável que permite que o usuário selecione o item e siga o ponteiro até o documento relacionado. A diferença entre o hipertexto e a hipermídia está no conteúdo do documento: documento de hipertexto contém apenas informação textual, enquanto que documentos de hipermídia podem conter representações adicionais de informação, incluindo gráficos ou imagens fotográficas digitalizadas.

A diferença entre um sistema de hipermídia distribuído e um não distribuído é significativa. Em um sistema não distribuído, a informação fica dentro de um só computador, usualmente em um mesmo disco. Devido a um grande conjunto de documentos estarem disponíveis localmente, links entre eles podem ser checados para consistência. Isto é, um sistema de hipermídia não distribuído pode garantir que todos os links são válidos e consistentes.

Ao contrário dele, a Web distribui documentos ao longo de um grande conjunto de computadores. Ainda mais, um administrador de um sistema pode escolher entre adicionar, remover, mudar ou renomear um documento em um computador sem notificar outros sites. Conseqüentemente, links entre documentos Web não são sempre consistentes. Por exemplo, suponha que um documento D1 em um computador C1 contenha um link para um documento

D2 em um computador C2. Se o administrador responsável pelo computador C2 resolver excluir o documento D2, o link em C1 se tornará inválido.

24.4 Representação de Documentos

Um documento de hipermídia disponível na Web é chamado de *página*: a página principal de uma organização ou de uma pessoa é conhecida como *homepage*. Devido à página poder conter muitos itens, o formato deve ser definido cuidadosamente para que o browser possa interpretar os diversos tipos de conteúdo. Em particular, um browser deve ser capaz de distinguir entre textos arbitrários, gráficos e links para outras páginas. Mais importante, o autor da página deve ser capaz de descrever o layout geral do documento (por exemplo, a ordem em que os itens são apresentados).

Cada página da Web que contenha um documento hipermídia utiliza um padrão de representação. Conhecido como *HyperText Markup Language (HTML)*, o padrão permite a um autor designar as linhas principais para o layout da página e especificar o conteúdo da página.

A HTML é uma *linguagem de marcação de hipertexto* porque ela não inclui instruções de formatação detalhadas. Por exemplo, embora a HTML contenha extensões que permitem ao autor especificar o tamanho e fonte do texto ou a largura da linha, a maioria dos autores prefere especificar apenas um nível de importância como um número de 1 a 6. O browser escolhe a fonte e mostra um tamanho apropriado para cada nível. Similarmente, a HTML não especifica exatamente como um browser marca um item como selecionável – alguns browsers sublinham itens selecionáveis, outros os mostram com uma cor diferente e alguns fazem ambos.

24.5 Formato e Representação HTML

Cada documento HTML é dividido em duas partes principais: um *cabeçalho* seguido de um *corpo*. O cabeçalho contém detalhes sobre o documento, enquanto que o corpo contém a maioria da informação. Por exemplo, o cabeçalho contém o título de um documento – a maioria dos browsers utiliza o título como um texto que permite ao usuário saber qual página está sendo vista.

Sintaticamente, cada documento HTML é representado como um arquivo de texto que contém *sinalizadores* ao longo do texto com outras informações. Como na maioria das linguagens de programação, espaços em branco (por exemplo, linhas extras e caracteres em branco) podem ser inseridos em um documento para tornar sua fonte legível; eles não tem efeito algum sobre a versão formatada que o browser mostra.

Os sinalizadores do HTML fornecem estrutura ao documento assim como dicas de formatação. Alguns sinalizadores especificam uma ação que ocorre imediatamente (por exemplo, mover para uma nova linha da página); o sinalizador é colocado exatamente onde a ação deveria ocorrer. Outros sinalizadores são usados para especificar uma operação de formatação que é aplicada a todo o texto que segue esse sinalizador. Tais sinalizadores ocorrem aos pares, com um sinalizador inicial e um final que começa e termina a ação, respectivamente.

Um sinalizador usado para especificar uma ação imediata ou para iniciar uma operação de formatação deve ser escrito com o nome do sinalizador envolto pelos sinais de menor e maior:

<TAGNAME>

O sinalizador correspondente usado para terminar uma operação começa com uma sequência de dois sinais: o sinal de menor e a barra, e termina com o sinal de maior:

</TAGNAME>

Por exemplo, um documento HTML começa com o sinalizador <HTML>. O par de sinalizadores <HEAD> e </HEAD> limitam o espaço para o cabeçalho, enquanto que o par de sinalizadores <BODY> e </BODY> limitam o espaço para o corpo. No cabeçalho, os sinalizadores <TITLE> e </TITLE> limitam o texto que forma o título do documento. Abaixo veremos a formatação básica de um documento HTML.

```
<HTML>
<HEAD>
<TITLE>
    Texto que forma o título do documento
</TITLE>
</HEAD>

<BODY>
    Corpo do texto é colocado aqui
</BODY>
</HTML>
```

No documento, cada sinalizador aparece em uma nova linha e a tabulação é usada para mostrar a estrutura. Entretanto, tais convenções são apenas importantes para outras pessoas que irão ler esse documento – um browser ignora todo tipo de espaçamento. Por isso, o documento HTML apresentado é equivalente a:

```
<HTML><HEAD><TITLE> Texto que forma o título
do documento </TITLE></HEAD><BODY> Corpo do
texto é colocado aqui </BODY></HTML>
```

24.6 Exemplo de Sinalizadores de Formatação HTML

Devido ao fato de que um documento HTML é armazenado em um arquivo de texto, o documento deve conter sinalizadores explícitos que especificam como o texto deveria ser mostrado. Por exemplo, o sinalizador
 instrui um browser a introduzir uma quebra de linha. Isto é, quando ele encontrar um
 no texto fonte, o browser move para o início da próxima linha na página. Por isso, a seguinte linha de programação HTML:

Olá.
Isto é um exemplo
de HTML.

Fará com que um browser gere:

Olá.
Isto é um exemplo
de HTML.

Uma sequência de duas quebras deixa uma linha em branco. Por isso:

Olá.

Isto mostra
o espaçamento HTML.

Resulta no seguinte texto:

Olá.

Isto mostra
o espaçamento HTML.

24.7 Títulos

O HTML inclui seis pares de sinalizadores que podem ser usados para mostrar títulos na página. Um sinalizador que tem a forma `<Hi>` marca o início de um título nível *i* e um sinalizador que tem a forma `</Hi>` marca o fim. Por exemplo, o texto do título de nível mais importante deve estar entre os sinalizadores `<H1></H1>`. Os browsers normalmente mostram o texto de um título tipo 1 no maior tamanho de letra, o título nível 2 um pouco menor, e assim por diante. Por isso, quando um browser tem o seguinte código:

```
Olá.<BR><H1> Isto É Um Título</H1><BR>De volta ao normal.
```

O browser irá escolher um tamanho grande para o título:

```
Olá
Isto É Um Título
De volta ao normal.
```

24.8 Listas

Além do título, o HTML permite que um documento contenha listas. O modo mais simples é a *lista sem ordenação*, a qual o browser irá apenas mostrar sem fazer qualquer tipo de ordenação. Na especificação HTML, os sinalizadores `` envolvem toda a lista e cada item da lista deve começar com o sinalizador ``. Usualmente, um browser coloca um ponto na frente de cada item.

Na maioria dos browsers, o código fonte:

Aqui está uma lista com 5 nomes:

```
<UL>
<LI> Scott
<LI> Sharon
<LI> Jan
<LI> Stacey
<LI> Rebecca
</UL>
```

Esse texto vem depois da lista.

Produziria o texto seguinte:

Aqui está uma lista com 5 nomes:

- Scott
- Sharon
- Jan
- Stacey
- Rebecca

Esse texto vem depois da lista.

24.9 Imagens Gráficas Colocadas em uma Página da Web

De um documento HTML é um arquivo de texto, como uma página da Web pode conter informações não textuais? Em geral, informações não textuais como imagens gráficas ou fotos

digitais não são inseridas diretamente em um documento HTML. Ao invés disso, os dados ficam em um local separado e o documento contém uma referência dos mesmos. Quando o browser encontra tal referência, ele vai até o local especificado, obtém uma cópia da imagem e a insere na página apresentada.

O HTML usa o sinalizador *IMG* para codificar a referência a uma imagem externa. Por exemplo, o sinalizador:


```
<IMG SRC="fred_photo.gif">
```

Especifica que o arquivo *fred_photo.gif* contém uma imagem que o browser deveria inserir no documento.

Arquivos listados em um sinalizador *IMG* diferem dos arquivos usados para guardar as páginas Web. Arquivos de imagens não são guardados como arquivos de texto e não seguem o formato HTML. Ao invés disso, cada arquivo de imagem contém dados binários que correspondem a uma imagem e o arquivo é armazenado em *graphics interchange format (gif)*. Devido aos arquivos de imagem não incluírem qualquer informação de formatação, o sinalizador *IMG* inclui parâmetros adicionais que podem ser usados para sugerir o posicionamento. Em particular, quando uma imagem aparece com outros itens (por exemplo, texto ou outras imagens), a palavra *ALIGN* pode ser usada para especificar entre o topo, o meio e o fundo da imagem que deveria ficar alinhado com outros itens na mesma linha. Por exemplo, o código abaixo:

```
Aqui está uma figura. <IMG SRC="fred_photo.gif" ALIGN="middle">
```

O browser alinha o meio da imagem com o texto:

Aqui está uma figura.  ← Posição da figura

24.10 Identificando uma Página

Quando um usuário abre um browser, ele deve especificar uma página inicial para ver. Identificar a página é complicado por várias razões. Primeira, a Web inclui muitos computadores e a página pode residir em qualquer um deles. Segunda, um dado computador pode conter várias páginas; a cada uma deve ser dado um nome único. Terceira, a Web suporta muitas representações de documentos, por isso o browser deve saber qual delas a página utiliza (por exemplo, entre o nome que se refere a um documento HTML e o que se refere a uma imagem armazenada no formato binário). Quarta, devido à Web ser integrada com outras aplicações, um browser deve saber qual protocolo deve ser usado para acessar cada página.

Um formato sintático foi inventado que incorpora toda a informação necessária para especificar um item remoto. A forma sintática codifica a informação em uma seqüência de caracteres conhecida como *Uniform Resource Locator (URL)*. O formato geral da URL é:

```
Protocolo://nome_do_computador:porta/nome_do_documento
```

Onde o *protocolo* é o nome do protocolo utilizado para acessar o documento, o *nome_do_computador* é o nome de domínio do computador onde o documento reside, a *:porta* é um número de porta de protocolo opcional e o *nome_do_documento* é o nome do documento em um computador específico. Por exemplo, a URL:

```
http://www.netbook.cs.purdue.edu/cs363/index.html
```


especifica o protocolo *http*, o computador chamado *www.netbook.cs.purdue.edu* e o arquivo chamado *cs363/index.html*.

Como o exemplo mostrou, a URL contém a informação que o browser precisa para exibir a página. O browser utiliza os caracteres de separação dois pontos (:) e barra (/) para dividir a URL em três partes: o protocolo, o nome do computador e o nome do documento. O browser utiliza então a informação para acessar o documento especificado.

24.11 Links em Hipertextos de um Documento para Outro

Embora o HTML inclua muitas opções usadas para descrever o conteúdo e o formato do documento, a opção que distingue o HTML das linguagens convencionais de formatação de documento é a sua habilidade de incluir referências de hipertexto. Cada referência de hipertexto é um ponto passivo para outro documento. Diferente da referência *IMG* a qual faz com que o browser adquira informação externa imediatamente, uma referência de hipertexto não causa nenhuma ação imediata. Ao invés disso, o browser torna-a um item selecionável quando o documento for mostrado. Se o usuário selecionar o item, o browser segue a referência, encontra o documento ao qual ela se refere e substitui o conteúdo atual pelo novo documento.

O HTML permite que qualquer item seja referenciado por uma referência de hipertexto. Por isso, uma só palavra, uma frase, um parágrafo inteiro ou uma imagem pode se referir a outro documento. Se uma imagem inteira é designada como referência de hipertexto, o usuário pode selecionar a referência clicando em qualquer ponto da mesma. Similarmente, se um parágrafo inteiro for designado como referência de hipertexto, clicar em qualquer caractere do mesmo faz com que o browser siga a referência.

O mecanismo usado pelo HTML para especificar uma referência de hipertexto é chamada de *âncora*. Para permitir que textos arbitrários e gráficos pudessem ser incluídos em uma mesma referência, o HTML utiliza os sinalizadores `<A>` e `` como limitadores da referência; todos os itens entre eles são parte da âncora. O sinalizador `<A>` inclui informação que especifica uma URL; se o usuário selecionar a referência, o browser usa a URL para obter o documento. Por exemplo, o código abaixo:

```
Esse é um livro publicado pela  
<A HREF= "http://www.prenhall.com" >  
Prentice Hall, </A> uma das  
maiores editoras de livros de computação.
```

Contém uma âncora que se refere à URL *http://www.prenhall.com*. Quando exibida na tela, o código produziria:

```
Esse é um livro publicado pela Prentice Hall, uma das maiores  
editoras de livros de computação.
```

O exemplo mostra uma convenção que muitos browsers utilizam para indicar um texto que é selecionável: o texto ancorado é exibido sublinhado. Apenas as palavras *Prentice Hall* são sublinhadas, porque as outras não estão ancoradas.

Tudo entre os dois sinalizadores que começa e termina uma âncora forma parte do item selecionável. A entrada em uma âncora pode incluir texto, sinalizadores que especifiquem alguma formatação ou imagens gráficas. Por isso, um documento HTML pode conter uma imagem ou um ícone que corresponda a um link de hipertexto assim como uma sequência de palavras.

24.12 Interação Cliente-Servidor

Como outras aplicações de rede, a visualização do browser utiliza o paradigma cliente-servidor. Quando uma URL de um documento é fornecida, o browser se torna cliente que contata no servidor do computador especificado na URL para pedir o documento. Finalmente o browser disponibiliza o documento para o usuário.

Diferente das aplicações de rede que mantém uma conexão estabelecida entre o cliente e o servidor, a conexão entre o browser de Web e um servidor tem uma duração curta. O browser estabelece a conexão, envia o pedido e recebe o item desejado ou uma mensagem de que ele não existe. Assim que o documento ou imagem tenha sido transferido, a conexão é fechada; o cliente não permanece conectado ao servidor.

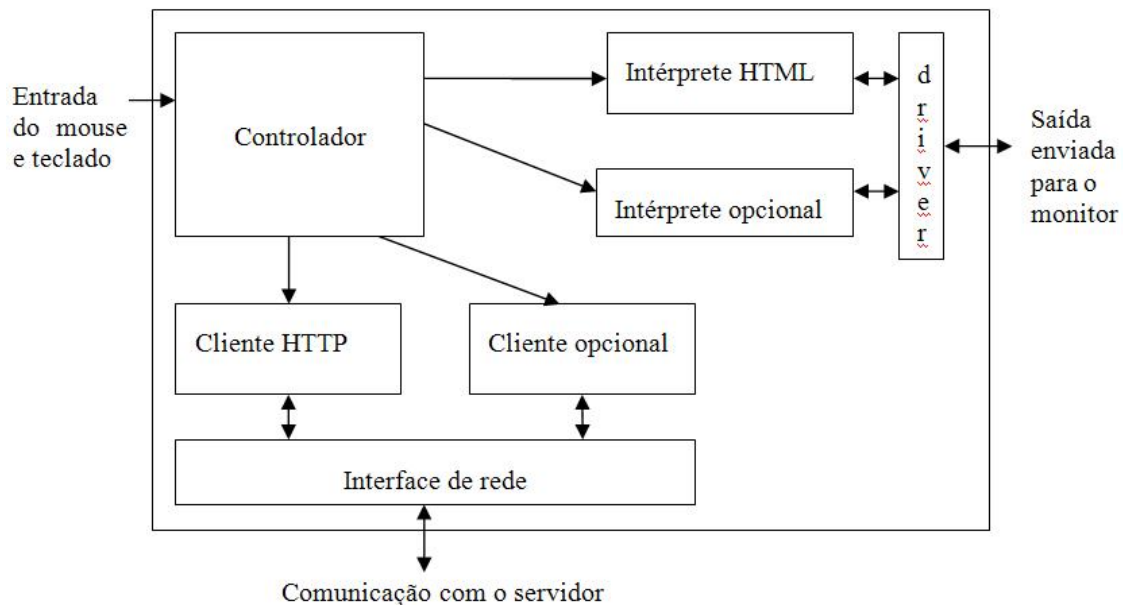
Terminar as conexões rapidamente funciona muito bem na maioria dos casos porque a navegação na Web não exige muita localidade. O usuário pode acessar uma página da Web em um computador e então imediatamente seguir um link para outra página em outro computador. Entretanto, terminar as conexões de maneira tão abrupta pode introduzir tráfego desnecessário em casos onde o browser deve retornar ao mesmo servidor para pegar vários documentos. Por exemplo, considere uma página que contenha referências para várias imagens, todas as quais residem no mesmo computador que o da página. Quando um usuário acessa a página, seu browser abre uma conexão, obtém a página e fecha a conexão. Quando ele precisa mostrar uma figura (por exemplo, quando encontra um sinalizador `` na página), o browser deve abrir uma nova conexão com o servidor para obter uma cópia dela.

24.13 Transporte de Documentos Web e HTTP

Quando um browser interage com um servidor de Web, os dois programas seguem o *HyperText Transport Protocol (HTTP)*. A princípio, o HTTP é direto: ele permite que o browser peça um item específico, o qual o servidor retorna. Para garantir que clientes e servidores possam interoperar sem ambigüidade, o HTTP define o formato exato dos pedidos enviados do browser para o servidor assim como o formato das respostas que o servidor retorna.

24.14 Arquitetura do Browser

Os browsers Web têm uma estrutura mais complexa que os servidores de Web. Um servidor realiza ações diretas repetidamente: o servidor espera o browser abrir uma conexão e pedir uma determinada página. O servidor então envia uma cópia do item requerido, fecha a conexão e espera pela próxima conexão. O browser lida com a maioria dos detalhes do acesso ao documento e a sua exibição. Conseqüentemente, o browser contém vários componentes de software grandes que trabalham juntos para fornecer a ilusão de um serviço sem união. A figura abaixo ilustra a organização conceitual de um browser.



Conceitualmente, o browser consiste em um conjunto de clientes, um conjunto de intérpretes e um controlador que os administra. O controlador forma a peça central do browser. Ele interpreta tanto os cliques do mouse quanto as entradas do teclado e atende a outros componentes para realizar operações especificadas pelo usuário. Por exemplo, quando um usuário entra uma URL ou clica em uma referência de hipertexto, o controlador chama o cliente para obter o documento desejado no servidor remoto em que ele reside e um intérprete para mostrá-lo ao usuário.

Cada browser deve conter um intérprete do HTML para exibir os documentos. Outros intérpretes são opcionais. A entrada para um intérprete do HTML consiste em um documento que está em conformidade com a sintaxe HTML; a saída consiste em uma versão formatada do documento no monitor do usuário. O intérprete lida com os detalhes do layout, traduzindo as especificações do HTML em comandos que são apropriados para o monitor do usuário. Por exemplo, se ele encontra um sinalizador de título no documento, o intérprete muda o tamanho do texto usado para exibi-lo. Similarmente, se ele encontra um sinalizador de quebra de linha, o intérprete começa a exibição do documento em uma nova linha.

Uma das funções mais importantes em um intérprete HTML envolve os itens selecionáveis. O intérprete deve guardar informação sobre a relação entre as posições no monitor e itens ancorados no documento HTML. Quando o usuário seleciona um item com o mouse, o browser usa a posição do cursor e a informação armazenada para aquela posição para determinar o item que o usuário selecionou.

24.15 Clientes Opcionais

Além do cliente e do intérprete HTML, o browser pode conter componentes que tornam o browser capaz de executar tarefas adicionais. Por exemplo, muitos browsers incluem um cliente FTP que é usado para acessar serviços de transferência de arquivos. Alguns browsers também contêm um software cliente de e-mail que permite ao browser enviar e receber mensagens de e-mail.

Como um usuário pode acessar um serviço como o FTP de dentro do browser? A resposta é que um usuário não pode abrir tais serviços explicitamente e nem interagir com softwares clientes convencionais. Ao invés disso, o browser abre o serviço automaticamente e o usuário executa a tarefa necessária. Se o browser for bem desenvolvido, ele esconde todos os detalhes do usuário, quem pode não estar ciente de que um serviço opcional foi executado. Por exemplo, uma transferência de arquivo pode ser associada a um item selecionável na tela.

Quando o usuário seleciona o item, o browser usa o cliente FTP para obter uma cópia do arquivo.

Como uma transferência de arquivo pode ser especificada em uma página da Web? Lembre-se que o primeiro campo em uma URL especifica o protocolo. O controlador usa o campo para determinar qual o cliente a ser chamado. Por exemplo, se a URL especificar o *HTTP*, o controlador chama o cliente HTTP. Similarmente, a URL:

`ftp://ftp.cs.purdue.edu/pub/comer/netbook/client.c`

especifica que o browser deveria usar anonimamente o FTP para obter o arquivo *pub/comer/netbook/client.c* do computador *ftp.cs.purdue.edu*.

Uma URL que especifique o FTP pode ser colocada em uma âncora HTML tão facilmente como uma URL que especifique o HTTP. Por exemplo, o código HTML abaixo:

A maioria dos exemplos nesse texto está disponível on-line. O código fonte de

um exemplo de programa cliente

ou o código de

um exemplo de programa servidor

estão disponíveis.

faz com que o browser mostre duas sentenças, parte das quais será selecionável:

A maioria dos exemplos nesse texto está disponível on-line. O código fonte de um exemplo de programa cliente ou o código de um exemplo de programa servidor estão disponíveis.

Se o usuário selecionar o primeiro segmento sublinhado, o browser usa o cliente FTP para obter uma cópia do arquivo *client.c*; clicando no segundo segmento faz com que o browser retorne o arquivo *server.c*. Depois de retornar uma cópia do arquivo, o browser verifica o conteúdo e exibe o resultado.

24.16 Armazenamento em Cache nos Browsers Web

A dinâmica do contato cliente-servidor em browsers Web é diferente da dinâmica em muitas aplicações já descritas. Tem duas razões. Primeira, devido aos usuários tenderem a ver páginas da Web de fora da sua rede local, os browsers Web tendem a referenciar páginas remotas mais freqüentemente que páginas locais. Segunda, devido aos usuários não procurarem a mesma informação repetidamente, eles não tendem a repetir os acessos dia após dia.

Devido à navegação pelo browser produzir uma referência de localidade diferente das outras aplicações, as técnicas usadas pelo browser para otimizar o processamento difere das outras aplicações. Em particular, nem os browsers ou servidores Web são otimizados para a sua localidade física. Ainda mais, os browsers empregam técnicas não usuais para lidar com localidades de referência temporais.

Como outras aplicações, os browsers usam um *cache* para melhorar o acesso a documentos. O browser coloca uma cópia de cada item que ele obtém em um cache no disco local. Quando um usuário seleciona um item, o browser verifica o cache do disco antes de tentar

obter uma nova cópia. Se o cache contiver o item, o browser obtém a cópia do cache sem usar a rede.

Manter itens em um cache pode melhorar o desempenho drasticamente – um browser pode ler um item do cache sem ter que esperar pelos delays da rede. Esse armazenamento é especialmente importante para usuários que possuem conexões de rede lentas. Por exemplo, considere um usuário que se conecte a Internet por uma conexão discada. Embora um modem rápido possa transferir dados a uma taxa teórica de 56Kbps, a taxa efetiva pode ser substancialmente menor se a conexão possuir ruídos indesejados. A tais velocidades, obter um item grande através da Internet pode levar muito mais tempo do que obtê-lo do cache do disco local. De fato, o acesso local pode parecer instantâneo quando comparado ao acesso à Internet.

Sem contar com a grande melhoria na velocidade de acesso, reter itens em um cache por longos períodos nem sempre é desejável. Primeiro, um cache pode ocupar muito espaço no disco. Por exemplo, suponha que um usuário visite 10 páginas, com 5 figuras grandes em cada uma delas. O browser armazenará os documentos da página mais todas as 50 figuras no cache do disco local. Segundo, a melhoria no desempenho só ocorre quando um usuário decidir ver o mesmo item de novo. Infelizmente, usuários freqüentemente navegam pela rede porque estão procurando informação; quando a informação é encontrada, o usuário pára de navegar. Por exemplo, um usuário que vê 10 páginas pode decidir que 9 delas não contém nada de interessante. Por isso, manter cópias de tais páginas no cache não melhorará o desempenho porque o usuário nunca retornará àquelas páginas. De fato, em tais situações, o armazenamento no cache diminui o desempenho porque o browser leva tempo para escrever os itens no disco desnecessariamente.

Para ajudar os usuários a controlar o modo como o browser lida com o cache, a maioria deles permite que o usuário ajuste a política de armazenamento. O usuário pode ajustar um tempo limite de armazenamento e o browser remove itens do cache depois que o tempo limite expira. Os browsers normalmente mantêm um cache durante uma sessão particular. O usuário que não quer que os itens permaneçam no cache depois de terminada uma sessão e iniciada outra pode solicitar um tempo de armazenamento igual a zero. Em tais casos, um browser remove o cache sempre que o usuário termina uma sessão.

Capítulo 25 – RPC e Middleware

25.1 Introdução

Os capítulos anteriores descreveram aplicações específicas que usam o modelo cliente-servidor para se comunicar através da rede e mecanismos como o CGI que programadores usam para construir sistemas cliente-servidor. Esse capítulo continua a discussão sobre as ferramentas que ajudam os programadores a criar sistemas cliente-servidor.

25.2 Programando Clientes e Servidores

Qualquer programador que tenha construído um software cliente-servidor sabe que tal programação é difícil. Em adição às tarefas usuais, os programadores que criam clientes e servidores devem lidar com as complexas questões de comunicação. Embora muitas das funções necessárias sejam fornecidas por um API padrão como a interface de soquete, os programadores ainda enfrentam um desafio. A chamada do soquete requer que o programador especifique muitos detalhes de baixo nível como nomes, endereços, protocolos e portas. Mais importantes ainda, pequenos erros no código que podem facilmente passar despercebidos quando o programa é testado podem causar grandes problemas quando o programa roda em uma produção.

Programadores que escrevem clientes e servidores perceberam que vários programas freqüentemente seguiam a mesma estrutura geral e repetiam muitos dos mesmos detalhes. Isto é, a maioria dos sistemas cliente-servidor seguia alguns passos básicos. Ainda mais, devido às implementações tenderem a usarem os mesmos API padrões (por exemplo, a interface de soquetes), muito do código detalhado encontrado em um programa era replicado em outros. Por exemplo, todos os programas cliente que usam transporte por conexão orientada devem criar um soquete, especificar o endereço do servidor final, abrir uma conexão com o mesmo, enviar pedidos, receber respostas e fechar a conexão quando a interação for completada.

Para evitar que o mesmo código seja escrito repetidamente e para produzir código que é correto e eficiente, programadores usam ferramentas automáticas para construir os clientes e os servidores. Uma *ferramenta* é um software que gera tudo ou parte de um programa computacional. A ferramenta aceita descrições de alto nível de serviços oferecidos e produz automaticamente grande parte do código necessário. A ferramenta não pode eliminar toda a programação – um programador deve fornecer código que desempenhe as funções do serviço em particular. Entretanto, a ferramenta pode lidar com os detalhes da comunicação. Como resultado, o código conterá poucos erros.

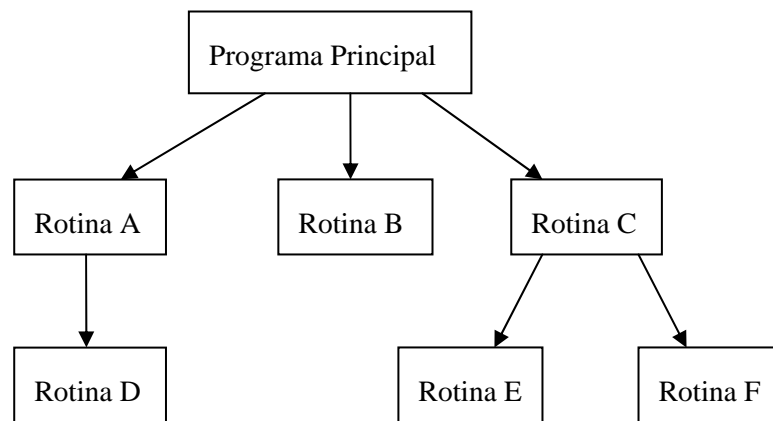
25.3 Paradigma de Chamada de Rotinas Remotas

Uma das primeiras facilidades que foram criadas para ajudar os programadores a escrever softwares cliente-servidor é conhecida genericamente como mecanismo de *Remote Procedure Call (RPC)*. A idéia do RPC veio da observação que a maioria dos programadores está familiarizada com as chamadas de rotinas como uma construção estrutural. Quando os programadores constroem um grande programa, eles começam dividindo-o em grandes pedaços. A divisão mais usada das linguagens de programação disponíveis hoje para tais grandes divisões é conhecida como *rotina*; um programador começa associando uma rotina com cada um dos grandes pedaços do design.

Quando desenvolve um programa, o programador usa as rotinas para manter o código administrável. Ao invés de definir uma rotina grande e única que faz muitas tarefas, o programador divide as tarefas em vários conjuntos e usa rotinas pequenas para lidar com cada

um deles. Se as tarefas resultantes precisarem de quantidades substanciais de código, o programador subdivide a tarefa e usa sub-rotinas para realizar cada uma das subtarefas. Por isso, a forma geral de um programa consiste em uma hierarquia de chamadas de rotinas.

A hierarquia de rotinas de um programa pode ser representada com um gráfico direcional em que cada nó representa uma rotina e a seta entre um nó *X* para um nó *Y* significa que a rotina *X* contém uma chamada para a rotina *Y*. A representação gráfica é conhecida como *gráfico de chamadas de rotinas*. A figura abaixo mostra um exemplo:



Na figura, o programa principal chama três rotinas, *A*, *B* e *C*. A rotina *A* chama a rotina *D* e a rotina *C* chama duas outras rotinas, *E* e *F*.

Outro aspecto de chamadas de rotinas convencionais será importante quando considerarmos chamadas remotas: parâmetros. Para tornar uma rotina geral e habilitar uma dada rotina a resolver um conjunto de tarefas relacionadas, cada rotina é *parametrizada* – a definição é dada com um conjunto de *parâmetros formais*. Quando a rotina é chamada, quem a chama deve fornecer *argumentos* que correspondem aos parâmetros formais.

Devido às rotinas parametrizadas terem funcionado tão bem como uma abstração da programação, pesquisadores examinaram maneiras de usar a abstração de rotinas para construir clientes e servidores.

A premissa da pesquisa era:

Se um programador seguir o mesmo paradigma de chamada de rotinas usado para construir programas convencionais quando estivesse construindo softwares clientes e servidores, o programador achará a tarefa mais fácil e cometerá menos erros.

Para seguir tal premissa, pesquisadores têm investigado maneiras de fazer com que a programação de clientes e servidores seja o mais parecido possível da programação convencional. Infelizmente, a maioria das linguagens de programação é desenvolvida para produzir código para um só computador. Nem a linguagem ou mesmo o compilador são feitos para um ambiente distribuído – o código que é gerado restringe o fluxo de controle e a transmissão de parâmetros ao espaço de um só endereço.

Poucos pesquisadores têm investigado modificações na linguagem que incluam facilidades para o desenvolvimento cliente-servidor. Por exemplo, o *Modula-3* e o *Java Remote Method Invocation (Java RMI)* ambos apresentam facilidades que permitem a um programa chamar rotinas em outros computadores. E ainda, ambos têm facilidades que sincronizam os processamentos de máquinas separadas para assegurar resultados consistentes.

A maior parte da pesquisa sobre essas facilidades que suportam a programação cliente-servidor tem evitado modificações diretas às linguagens de programação definindo um conjunto de ferramentas que programadores podem utilizar para construir programas distribuídos. Em particular, as ferramentas disponíveis aceitam especificações de programas de alto nível, a inserção de código necessário para a comunicação em rede e automaticamente traduz o código

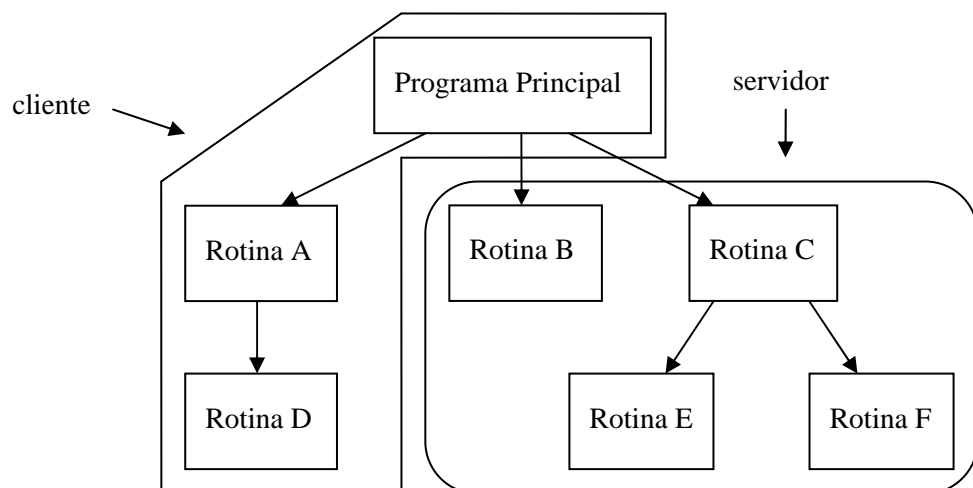
resultante para produzir os programas clientes e servidores. A abordagem baseada em conjunto de ferramentas é hoje o mais aceito.

25.4 Paradigma RPC

Quando usam o RPC, a atenção dos programadores não está focada em uma rede de computadores ou em protocolos de comunicação. Ao invés disso, o programador é encorajado a pensar sobre o problema a ser resolvido. Isto é, o programador não começa pensando em dois programas separados. O programador começa imaginando como o problema será resolvido por um programa convencional rodando em um só computador. O programador desenvolve e constrói o programa, usando chamadas de rotinas como a estrutura básica.

Uma vez que um programador tenha construído um programa convencional para solucionar um problema local, o programador então considera como dividir o programa em duas partes. Na essência, o programador deve particionar o gráfico de chamadas em duas partes. A parte que contém o programa principal e as rotinas que ele chama se torna o cliente, enquanto que as rotinas remanescentes se tornam o servidor. Quando seleciona o conjunto de rotinas que formaram o cliente e o conjunto que compreenderá o servidor, o programador deve considerar os dados – os dados globais que cada rotina se refere devem estar localizados no mesmo computador que tal rotina.

O desenho abaixo exemplifica uma maneira de dividir o programa do exemplo anterior em duas partes.



Na figura, o cliente consiste no programa principal mais as rotinas A e D. O servidor consiste nas rotinas B, C, E e F. Outras divisões do gráfico de chamadas são possíveis; não podemos dizer através da figura porque que o programador fez essa escolha. Por exemplo, as rotinas A e D podem estar no programa cliente porque elas lidam com interações com o usuário ou podem estar agrupadas com o programa principal porque elas se referem às variáveis globais no programa principal.

O RPC estende o mecanismo de chamada de rotinas para permitir que uma rotina no cliente chame uma rotina através da rede no servidor. Isto é, quando tal chamada ocorre, o **thread of control** aparece para transmitir através da rede do cliente para servidor; quando a rotina chamada responde, o **thread of control** aparece novamente para transmitir do servidor para o cliente.

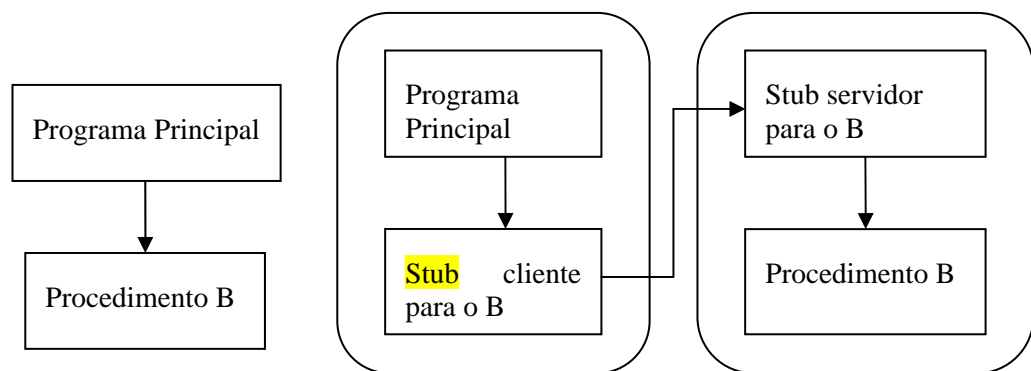
Depois de decidir quais rotinas ficarão no servidor e quais ficarão no cliente, o programador está pronto para utilizar a ferramenta RPC. Para fazê-lo, o programador cria uma especificação que descreve o conjunto de rotinas que serão *remotas* (por exemplo, o conjunto de rotinas que forma o servidor). Para cada rotina remota, o programador deve especificar os tipos de seus parâmetros. A ferramenta cria códigos que lidam com a comunicação necessária.

25.5 Stubs da Comunicação

Um **thread of control** não pode pular de um programa em um computador para uma rotina em outro. Ao invés disso, ele usa a interação cliente-servidor. Quando um cliente chama uma rotina remota, ele usa os protocolos convencionais para enviar a mensagem de pedido através da rede para o servidor. O pedido identifica uma rotina a ser executada. Quando o programa remoto (por exemplo, o servidor) recebe um pedido de um cliente, ele executa a rotina especificada e então envia o resultado de volta para o cliente.

Obviamente, softwares extras devem ser adicionados a cada uma das partes do programa para implementar a interação. O software extra no lado do cliente lida com os detalhes de enviar uma mensagem através da rede e de esperar uma resposta. O software extra no servidor lida com os detalhes de receber mensagens de pedido, chamar a rotina especificada e enviar uma resposta.

Tecnicamente, cada pedaço de software adicionado é conhecido como um **stub** da comunicação ou *proxy* (a tradução literal de *proxy* é *substituto*). Os dois **stubs**, um no cliente e um no servidor, lidam com todos os detalhes da comunicação. Em adição, os **stubs** são construídos para se encaixar no programa existente; o resto do programa simplesmente utiliza as chamadas de rotinas como se todas as rotinas fossem locais. Se a chamada é feita para uma rotina não local, o **stub** da comunicação intercepta-a, encontra os valores dos argumentos (conhecido como *organização dos argumentos*) e envia a mensagem através da rede para o **stub** de comunicação no servidor. O **stub** de comunicação no servidor utiliza o mecanismo convencional de chamada de rotinas para executar a rotina especificada e envia o resultado de volta para o **stub** do cliente. Quando o **stub** do cliente recebe a resposta, ele retorna o resultado para o seu requisitante exatamente como se fosse uma rotina local retornando o seu resultado. A figura abaixo mostra como os **stubs** de comunicação são adicionados aos lados do cliente e do servidor de um programa.



O exemplo (a) mostrou a chamada de rotina no programa original antes dos **stubs** do RPC terem sido adicionados. Quando a rotina *principal* chama a rotina *B*, os argumentos que ela transmite devem concordar exatamente com o formato dos parâmetros de *B*. Isto é, a chamada deve conter o número correto de argumentos e o tipo de cada argumento deve combinar com o tipo declarado para o parâmetro.

Já o exemplo (b) mostrou os **stubs** de comunicação que devem ser adicionados ao programa quando ele é dividido em cliente e servidor. É importante entender que as interfaces de rotina no exemplo (b) usam o mesmo número e tipo de argumentos que a interface original em (a). Por isso, a chamada da rotina *principal* para o **stub** cliente e a chamada do **stub** servidor para a rotina *B* utilizam a mesma interface que uma chamada convencional da rotina *principal* para a *B*. Mais importante, os **stubs** cliente podem receber o mesmo nome da rotina que eles

estão substituindo. Como resultado disso, o código do programa original não precisa ser modificado.

25.6 Representação de Dados Externos

Sistemas computacionais não usam a mesma representação interna de dados. Por exemplo, alguns computadores guardam o byte mais significativo de um número inteiro no endereço mais baixo e outros guardam o byte menos significativo no menor endereço. Por isso, quando clientes e servidores enviam valores inteiros entre computadores heterogêneos, eles devem concordar nos detalhes como quando que o cliente ou o servidor deverá converter ou qual representação está sendo enviada através da rede.

O termo *representação de dados externos* refere-se à forma dos dados que são enviados através da rede. Em adição a definir uma interface de linguagem, a tecnologia de chamada de rotina remota define uma representação de dados externos. Alguns sistemas negociam a representação – os **stubs** cliente e servidor trocam mensagens para determinar qual a representação cada um usará e o que precisa ser traduzido. Muitos sistemas escolhem outra abordagem na qual a representação externa é fixa. Nesses casos, quem envia sempre converte da representação local para a externa e quem recebe sempre converte da representação externa para a local.

25.7 Middleware e Middleware com Orientação a Objeto

Revisando, a ferramenta pode ajudar o programador a construir um programa que usa chamada de rotina remota. O programador especifica um conjunto de rotinas que serão remotas através de detalhes da interface (por exemplo, número e tipo de argumentos). Para isso, o programador utiliza a ferramenta de *Interface Definition Language* ou *Interface Description Language (IDL)*. A ferramenta lê as especificações IDL e gera os **stubs** cliente e servidor necessários. O programador então compila e liga os dois programas separados. Os **stubs** servidores são combinados com as rotinas remotas para formar o programa servidor. Os **stubs** clientes são combinados com o programa principal e com as rotinas locais para formar o programa cliente.

Uma variedade de ferramentas comerciais tem sido desenvolvida para ajudar os programadores a construir softwares cliente-servidor através do paradigma descrito acima. Tais ferramentas são chamadas genericamente de *middleware* porque fornecem software que se encaixa entre um programa de aplicação convencional e os softwares de rede.

Nos anos 90, as pesquisas sobre linguagem de programação mudaram o foco do paradigma de rotinas para o paradigma da orientação a objetos. Como resultado, a maioria das novas linguagens de programação são *linguagens com orientação a objeto*. A facilidade estrutural básica em uma linguagem com orientação a objeto é chamada *objeto*, a qual consiste em itens de dados mais um conjunto de operações para tais itens, as quais são chamadas de *métodos*. O mecanismo básico de controle em uma linguagem com orientação a objeto é a *execução de métodos*. Em resposta às mudanças nas linguagens, desenvolvedores estão criando novos sistemas de middleware que estendem a execução de métodos através dos computadores do mesmo modo que a chamada de rotinas remotas estendeu a chamada de rotinas. Tais sistemas são conhecidos como sistemas de objetos distribuídos. As próximas seções descreverão brevemente muitos middlewares e tecnologias de middlewares com orientação a objeto largamente utilizados.

25.7.1 ONC RPC

Um dos primeiros mecanismos RPC que atingiram aceitação geral foi desenvolvido pela Sun Microsystems, Incorporated. Formalmente conhecido por *Open Network Computing Remote Procedure Call (ONC RPC)*, essa tecnologia é freqüentemente conhecida como *Sun*

RPC. Em adição a especificação de uma IDL e um formato de mensagem que os **stubs** cliente e servidor usam para se comunicar. O *ONC RPC* inclui um padrão de representação de dados conhecida por *eXternal Data Representation (XDR)*. Finalmente, como a maioria das outras tecnologias *RPC*, o *ONC RPC* usa os protocolos *TCP/IP* e permite que o programador escolha entre o transporte *UDP* ou *TCP*.

25.7.2 DCE RPC

A *The Open Group* definiu o *Distributed Computing Environment (DCE)*, o qual é composto de muitos componentes e ferramentas que são designadas para trabalharem juntas. O *DCE* inclui sua própria tecnologia de chamada de rotinas remotas, a qual é frequentemente chamada de *DCE/RPC* e sua própria IDL, a qual difere das outras em pequenos detalhes. O *DCE/RPC* também permite que um cliente acesse mais de um servidor (por exemplo, algumas rotinas remotas podem estar localizadas em um servidor, enquanto que outras rotinas podem estar em outro). Em todos os casos, o *DCE/RPC* pode usar os protocolos *TCP/IP*, com a escolha do programador entre transporte por *UDP* ou *TCP*.

25.7.3 MSRPC

A *Microsoft Corporation* definiu sua própria tecnologia de chamada de rotinas remotas, a *Microsoft Remote Procedure Call (MSRPC)*. Devido à *MSRPC* ter sido derivada da *DCE/RPC*, ela compartilha os mesmos conceitos centrais e estrutura geral do programa. Entretanto, a *MSRPC* difere em alguns pequenos detalhes. Ela define sua própria IDL e tem seu próprio protocolo que os **stubs** cliente e servidor usam para se comunicar.

25.7.4 CORBA

Talvez o middleware com orientação a objeto mais conhecido é chamado *Common Object Request Broker Architecture (CORBA)*. O *CORBA* permite que um objeto inteiro seja colocado no servidor e estende a execução de métodos usando a mesma abordagem geral descrita anteriormente. Existe uma diferença porque os proxies são instanciados em tempo real como outros objetos. Quando um programa recebe uma referência de um objeto remoto, um proxy local é criado que corresponde ao objeto criado. Quando o programa executa um método daquele objeto, o controle passa para o proxy local. O proxy então envia uma mensagem através da rede para o servidor, o qual executa o método especificado e retorna os resultados. Por isso, o *CORBA* faz com que a execução de métodos para objetos locais e remotos pareçam idênticos.

Em adição a focar objetos ao invés de rotinas, o *CORBA* difere das tecnologias *RPC* convencionais porque ele é mais dinâmico. Em tecnologias *RPC* convencionais, o programador utiliza uma ferramenta para criar rotinas de **stub** quando constrói o programa. No *CORBA*, o software cria um proxy em tempo real quando é necessário (por exemplo, quando o método é executado em um objeto remoto para o qual não existe proxy).

25.7.5 MSRPC2

A *Microsoft* desenvolveu uma segunda geração do *MSRPC* chamado *MSRPC2*. a *Microsoft* frequentemente utiliza o nome *Object RPC (ORPC)* ao invés de *MSRPC2* porque a segunda versão representa uma mudança mais significativa que uma simples revisão. Ela estende os conceitos do *MSRPC* para fornecer mais suporte aos objetos. As idéias no *MSRPC2* são derivadas de muitas fontes, incluindo a pesquisa feita pela *Digital Equipment Corporation* usando o *Modula-3*.

25.7.6 COM/DCOM

Em outro esforço, a Microsoft Corporation também produziu o COM e o DCOM, os quais são tecnologias com orientação a objeto. O *Component Object Model (COM)* é uma estrutura de software com orientação a objeto que foi definido em 1994. O padrão define uma interface binária entre os componentes em um dado computador, algumas vezes sendo referido como um *esquema de empacotamento*. Com o COM, todos os objetos tinham nomes globais únicos e todas as referências de objetos usavam esquema global.

Definido em 1998, o *Distributed COM (DCOM)* estendeu o COM criando um protocolo de nível de aplicação para as chamadas de rotinas remotas com orientação a objeto. O DCOM, o qual foi proposto para a IETF como padrão para a Internet, utiliza o ORPC para transporte. Como o ORPC foi derivado do DCE/RPC, o DCOM usa exatamente o mesmo formato de pacote e semântica para execução remota que o DCE/RPC utiliza para chamadas de rotinas remotas.

Capítulo 26 – Tecnologia CGI para Documentos Dinâmicos Web

26.1 Introdução

Este capítulo continua a discussão do software do navegador, explicando duas formas alternativas dos documentos Web e examina uma forma em detalhes. Diferente dos documentos HTML discutidos anteriormente, esta alternativa fornece aos programas o controle do conteúdo de um documento dinamicamente. Em um grupo, o servidor Web executa um programa auxiliar para criar documentos quando a requisição chega do navegador. Em outro grupo, o servidor fornece ao navegador um programa. O navegador executa o programa localmente, e permite ao programa exibir informações, interagir com o usuário, e se tornar cliente de outros serviços de rede.

Este capítulo descreve duas alternativas do modelo de documentos, e resume as vantagens e desvantagens de cada um. O capítulo se concentra em um dos dois modelos, explicando como o servidor pode usar aplicativos para gerar documentos dinamicamente. Mais importante, o capítulo discute o conceito de informação do estado, e mostra como a informação do estado pode ser codificada em um documento criado dinamicamente.

26.2 Três Tipos Básicos de Documentos Web

O capítulo anterior descreveu a página Web como um simples documento de texto armazenado em um arquivo. Em geral, todo documento Web pode ser agrupado em três categorias gerais:

- *Estático.* Um documento de Web estático mora em um arquivo associado com servidor Web. O autor do documento estático determina o conteúdo do arquivo no tempo que este é gravado. Como o conteúdo não muda a cada requisição do documento estático, o resultado é sempre a mesma resposta.
- *Dinâmico.* Um documento dinâmico não existe em uma forma pré-determinada. Ao invés disso, o documento dinâmico é criado pelo servidor Web quando o navegador faz a requisição do documento. Quando a requisição chega, o servidor Web executa um aplicativo para criar o documento dinâmico. O servidor retorna a saída do programa como resposta ao navegador que requisitou o documento. Como um documento novo é criado a cada requisição, o conteúdo do documento dinâmico pode variar de uma requisição a outra.
- *Ativo.* Um documento ativo não é especificado completamente no servidor. O documento ativo consiste em um programa que entende como computar e mostrar valores. Quando o navegador requisita o documento ativo, o servidor retorna a cópia do programa que o navegador deve executar localmente. Quando este é executado, o programa do documento ativo interage com o usuário e muda as exibições continuamente. Assim, o conteúdo de um documento ativo nunca é fixo – eles podem mudar continuamente desde que o usuário permita que o programa possa ser executado.

26.3 As Vantagens e Desvantagens de Cada Tipo de Documento

A vantagem principal de um documento estático é simplicidade, confiabilidade, e desempenho. Como estes contêm especificações de formatação fixas, o documento estático pode ser criado por alguém que não sabe criar um programa de computador. Mais importante, após ser criado e testado completamente, o documento permanece válido indefinidamente. Finalmente, o navegador pode exibir um documento estático rapidamente, e pode armazenar uma cópia do documento no disco rígido para aumentar a velocidade de requisição do documento em uma próxima requisição.

A desvantagem principal do documento estático é a falta de flexibilidade – o documento deve ser revisado quando as informações mudam. Além do que, as mudanças se tornam consumo de tempo, pois necessitam de um humano para editar o arquivo. Assim, o documento estático não é usado para reportar informações que mudam com frequência.

A vantagem do documento dinâmico é a habilidade de reportar informação corrente. Por exemplo, o documento dinâmico pode ser usado para exibir informações como condições climáticas momentâneas, disponibilidade de ingressos para um concerto. Quando o navegador requisita a informação, o servidor executa um aplicativo para acessar as informações necessárias e criar o documento. O servidor então envia o documento ao navegador.

Os documentos dinâmicos geram grande responsabilidade no servidor, o navegador tem a mesma interação obtendo uma página dinâmica bem como aquela necessária para obter uma imagem estática. De fato, do ponto de vista do navegador, a página estática é indistinguível de uma página dinâmica. Como as páginas estáticas e dinâmicas usam o HTML, o navegador não sabe quando o servidor extrai a página do disco rígido, ou obtém a página dinamicamente de um programa.

As desvantagens principais do documento dinâmico são os custos altos e falta de habilidade ao exibir as informações. Como o documento estático, o documento dinâmico não muda após o navegar salvar uma cópia. Assim, a informação do documento dinâmico começa a envelhecer após ser enviada ao navegador. Por exemplo, considere um documento dinâmico que exiba preços de estoque. Como os preços variam continuamente, o documento se torna obsoleto enquanto o usuário está navegando em seus conteúdos.

O documento dinâmico tem maior custo para ser criado e acessado do que um documento estático. Os custos pessoais de criação de um documento dinâmico são maiores porque o criador da página Web dinâmica deve conhecer como escrever um programa de computador. Além disso, o programa deve ser criado com cuidado e testado exaustivamente para ter certeza que a resposta é válida.

Em adição aos custos de criação do documento dinâmico, os custos de hardware crescem para os documentos dinâmicos porque um sistema computacional poderoso é necessário para operar o servidor. Finalmente, o documento dinâmico tem uma velocidade de resposta um pouco maior do que a de um documento estático, pois o servidor necessita de tempo adicional para executar o aplicativo que cria o documento.

Embora o documento dinâmico seja criado quando a requisição é feita, a informação pode envelhecer rapidamente. A vantagem principal do documento ativo é a capacidade de modificar a informação continuamente, diferente do documento dinâmico. Por exemplo, somente o documento ativo pode modificar a exibição tão rapidamente para mostrar uma imagem animada. Mais importante, o documento ativo pode acessar fontes de informações diretamente, e assim atualizar a exibição continuamente. Por exemplo, o documento ativo que exibe preços de estoque, pode receber informações contínuas do estoque e modificar a exibição sem ação nenhuma do usuário.

As desvantagens principais do documento ativo são os custos adicionais para a criação e execução deste documento e a carência de segurança. Primeiro, o documento ativo requer para sua exibição um navegador sofisticado e um sistema computacional poderoso para executar o navegador. Segundo, criar um documento ativo que esteja correto requer mais habilidade na

programação do que outras formas, resultando em documentos mais difíceis de serem testados. Em particular, como o documento ativo é executado em um computador arbitrário ao invés do servidor, o programa deve ser criado para evitar dependência de componentes disponíveis em um sistema que não estão disponíveis em outros sistemas. Finalmente, o documento ativo é um risco potencial de segurança porque o documento pode exportar bem como importar informações.

26.4 Implementação de Documentos Dinâmicos

Como a responsabilidade de criação do documento dinâmico é do servidor Web que gerencia o documento, as mudanças requeridas para suportar documentos dinâmicos afetam apenas estes servidores. Na prática, as mudanças envolvem extensões – o servidor que gerencia documentos dinâmicos também contém o código para lidar com documentos estáticos.

Para o servidor convencional lidar com documentos dinâmicos, três adições são necessárias. Primeiro, o programa do servidor deve ser estendido até que seja capaz de executar um aplicativo separado que irá criar o documento a cada vez que uma requisição chegue. O servidor deve ser programado para capturar a resposta do aplicativo e retornar o documento ao navegador. Segundo, o aplicativo separado deve ser escrito para cada documento dinâmico. Terceiro, o servidor deve ser configurado para que saiba quais URLs correspondem a documentos dinâmicos e quais pertencem a documentos estáticos. Para cada documento dinâmico, a configuração deve especificar qual aplicativo irá gerar o documento.

Cada requisição contém uma URL correspondente a um documento estático ou dinâmico. O servidor usa a informação configurada e a URL em cada requisição para determinar como proceder. Se a informação configurada especifica que a URL corresponde a um documento estático, o servidor fornece o documento com a maneira usual. Se a URL corresponde a um documento dinâmico, o servidor determina o aplicativo que irá gerar o documento, executa o aplicativo, e usa a resposta do programa como documento a ser retornado ao navegador.

26.5 O Padrão CGI

A tecnologia usada largamente para construir documentos Web dinâmicos é conhecida como *Common Gateway Interface* (CGI). Originalmente desenvolvida pelo *National Center for Supercomputer Applications* (NCSA) para uso do servidor Web NCSA, o padrão CGI especifica como o servidor interage com o aplicativo que implementa o documento dinâmico. Este aplicativo é chamado de *programa CGI*.

O CGI fornece linhas gerais de guia, e permite o programador escolher a maioria dos detalhes. Por exemplo, o CGI não especifica uma linguagem de programação particular. Ao invés disso, o padrão permite ao programador escolher a linguagem e usar diferentes linguagens para diferentes documentos dinâmicos. Assim, o programador pode escolher a linguagem apropriada para cada documento.

26.6 Resposta de um Programa CGI

Na prática, a resposta de um programa CGI não é restrita ao HTML – o padrão permite que os aplicativos CGI criem tipos de documentos arbitrários. Por exemplo, em adição ao HTML, o programa CGI pode gerar um texto comum ou uma imagem digital. Para distinguir

dentro da variedade dos tipos de documentos, o padrão permite ao programa CGI colocar um cabeçalho na resposta. O cabeçalho consiste em um texto que descreve o tipo do documento.

Os cabeçalhos gerados pelo programa CGI seguem o mesmo formato geral dos cabeçalhos do servidor Web usados ao enviar um documento ao navegador: o cabeçalho consiste em uma ou mais linhas de texto e uma linha em branco. Cada linha do cabeçalho especifica informações sobre o documento e a representação de dados.

Após o executar o programa CGI, o servidor examina o cabeçalho antes de retornar o documento ao navegador que criou a requisição. Assim, quando necessário o programa CGI pode usar o cabeçalho para se comunicar com o servidor. Por exemplo, o cabeçalho que consiste da linha:

Content-type: text/html

Seguido de uma linha em branco especifica que a resposta é um documento HTML.

O cabeçalho da resposta de um programa CGI também pode ser usado para especificar que o documento está em uma nova localização. A técnica é conhecida como *redirecionamento*. Por exemplo, suponha que o programa CGI é associado com a seguinte URL:

http://someserver/cgi+bin/foo

e necessita referir as requisições que chegam ao documento associado com a URL:

http://someserver/new/bar.txt

O programa CGI pode gerar duas linhas de resposta:

Location: /new/bar.txt
<linha em branco>

Quando o servidor detecta a diretiva *Location* no cabeçalho, o mesmo irá responder o documento como se o navegador tivesse requisitado */new/bar.txt* (ao invés de *cgi+bin/foo*) com a URL:

http://someserver/new/bar.txt

26.7 Exemplo de um Programa CGI

Abaixo temos o código fonte de um programa CGI trivial escrito na linguagem shell UNIX. Quando executado, o programa cria um documento de texto comum que contém a data e hora atual.

```
#!/bin/sh

echo Content-type: text/plain
echo

echo This document was created on `date`.
```

Os scripts shell podem ser difíceis de entender. Na essência, o shell é um interpretador de comandos – o script shell contém comandos no mesmo formato que o usuário segue ao entrar com comandos no teclado. Exceto a primeira linha, o shell ignora linhas em branco ou linhas que começam com o símbolo #. No exemplo, linhas que contém o símbolo # são usadas para

comentários. Outras linhas contêm um comando válido. A primeira palavra de uma linha é o nome do comando; palavras subsequentes formam o argumento do comando.

O único comando usado no exemplo é *echo*. Cada chamada do *echo* gera uma linha de resposta. Quando executado, o script invoca três vezes o comando *echo*. Assim, o script irá gerar exatamente três linhas de resposta.

Quando invocado sem argumentos, o *echo* cria uma linha em branco de resposta. De outras maneiras, o *echo* escreve exatamente uma cópia dos seus argumentos. Por exemplo, o comando:

```
echo smord hplar
```

gera uma linha de saída que contém as duas palavras:

```
smord hplar
```

O único item não usual no script de exemplo é a construção 'date'. O shell interpreta as aspas graves como requisição de executar o programa *date* e substitui o mesmo na saída. Assim, antes de invocar o *echo* pela terceira vez, o shell substitui a string 'date' pela data e hora atual. Como resultado, a terceira invocação de *echo* gera uma linha de resposta que contém a data atual. Por exemplo, se o script é executado em 3 de Junho de 1999, as 14 horas 39 minutos e 37 segundos, o script irá gerar a seguinte resposta:

```
Content-type: text/plain
```

```
This document was created on Thu Jun 3 14:19:37 EST 1999
```

O servidor que é capaz de executar um programa CGI deve ser configurado antes de executar o script. A configuração especifica a URL que o servidor usa para localizar o script. Quando o navegador contata o servidor e requisita a URL específica, o servidor executa o programa. Quando o navegador recebe o documento, este irá exibir a seguinte linha ao usuário:

```
This document was created on Thu Jun 3 14:19:37 EST 1999
```

Diferente do documento estático, o conteúdo do documento dinâmico muda a cada vez que o usuário instrui o navegador a recarregar o documento. Isto porque o navegador não armazena um documento dinâmico, a requisição para recarregar o documento faz com que o navegador contate o servidor. O servidor executa o programa CGI, o qual cria um novo documento com a data e hora atual. Assim, o usuário vê a mudança do conteúdo do documento a cada vez que requisita recarregar a página.

26.8 Parâmetros e Variáveis Globais

A norma permite o programa CGI ser parametrizada. Isto é, o servidor pode passar argumentos ao programa CGI quando o programa é executado. A parametrização é importante porque permite a um único programa CGI lidar com um conjunto de documentos dinâmicos que diferem apenas em pequenos detalhes. Mais importante, os valores dos parâmetros podem ser fornecidos pelo navegador. Para fazer isso, o navegador adiciona informações a URL. Quando a requisição chega, o servidor divide a URL na requisição em duas partes: um prefixo que especifica o documento em particular, e o sufixo que contém informações adicionais. Se o prefixo da URL corresponde ao programa CGI, o servidor executa o mesmo e passa o sufixo da URL como argumento.

Sintaticamente, a marca de pontuação (?) na URL separa o prefixo do sufixo. Tudo antes da marca (?) é o prefixo, o qual especifica o documento. O servidor utiliza sua informação configurada para determinar como mapear o prefixo a um programa CGI. Quando este executa

o programa CGI, o servidor passa o argumento ao programa, que corresponde aos caracteres na URL após a marca (?).

Como o sistema foi projetado para ser executado nos sistemas operacionais UNIX e Windows, o padrão CGI segue uma convenção não muito usual para passar os argumentos ao programa CGI. Ao invés de usar a linha de comando, o servidor passa a informação nas *variáveis globais* do UNIX, e então executa o programa CGI. O programa CGI herda uma cópia das variáveis globais, das quais extrai os valores.

O servidor passa valores para outras variáveis globais que o programa CGI possa usar. A tabela na figura 30.2 lista exemplos de variáveis globais CGI e seus significados.

Name of Variable	Meaning
SERVER_NAME	The domain name of the computer running the server.
GATEWAY_INTERFACE	The version of the CGI software the server is using.
SCRIPT_NAME	The path in the URL after the server name.
QUERY_STRING	Information following "?" in the URL.
REMOTE_ADDR	The IP address of the computer running the browser that sent the request.

26.9 Informação de Estado

O servidor executa o programa CGI cada vez que uma requisição chegue associada à URL do mesmo. Além disso, como o servidor não mantém nenhum histórico de requisições, o servidor não pode contar ao programa CGI as requisições anteriores do mesmo usuário. No entanto, o histórico é útil porque permite ao programa CGI participar de uma conversação. Por exemplo, o histórico de interações anteriores evita que o usuário responda questões repetidamente quando especifica requisições adicionais. Para fornecer um dialogo não repetitivo, o programa CGI deve salvar informações entre as requisições.

A informação que um programa salva entre as requisições é chamada de *informações de estado* (*state information*). Na prática, os programas de documentos dinâmicos usam dois modos para salvar informações; a escolha é determinada pelo espaço de tempo que a informação deve ser armazenada. O primeiro método é usado para as informações que devem ser gravadas através das requisições do navegador. O programa de documento dinâmico usa armazenamento de longo período no computador servidor, como disco rígido. O segundo método é usado para informações que devem ser mantidas enquanto o navegador esta sendo executado. O programa de documento dinâmico codifica essas informações na URL que aparece no documento. A próxima seção contém exemplos que ajudam no conceito.

26.10 O Script CGI com Informação de Estado de Longo Termo

O script CGI na figura 30.3 abaixo ilustra dois conceitos que discutimos. Primeiro, o script ilustra como o servidor utiliza as variáveis globais para passar argumentos ao programa CGI. Segundo, o script mostra como o programa de documento dinâmico pode armazenar informações de estado de longo termo em um arquivo.

O exemplo de script armazena uma cópia do endereço IP dos computadores que estivera em contato. Quando a requisição chega, o script analisa o endereço IP do navegador e gera um documento. O documento conta se é a primeira requisição que chegou daquele computador.

```
#!/bin/sh
FILE=ipaddrs

echo Content-type: text/plain
echo

if grep -s $REMOTE_ADDR $FILE > /dev/null 2>&1
then
echo Computer $REMOTE_ADDR has requested this URL previously.
else
echo $REMOTE_ADDR >> $FILE
echo This is the first contact from computer $REMOTE_ADDR.
fi
```

Embora muitos detalhes sintáticos façam com que o script pareça complexo, sua operação é direta. Na essência, o script mantém a lista de endereços de IP em um arquivo local chamado *ipaddrs*. Quando executado, o script procura o arquivo para o endereço de IP do navegador. Se for encontrado, o script reporta que ele já foi contatado pelo navegador do computador anteriormente. Se não, o script adiciona o endereço IP do navegador ao arquivo, e reporta que o contato foi o primeiro feito pelo navegador do computador.

O script usa a variável global *REMOTE_ADDR* para obter o endereço de IP do computador que está executando o navegador que formou a requisição. Na linguagem shell, as variáveis são referenciadas com o signo dólar (\$) na frente do nome da variável. Assim, a string *\$REMOTE_ADDR* é substituída pelo valor da variável *REMOTE_ADDR*.

O corpo principal do script consiste em um único enunciado *if-then-else*. A instrução usa o comando *grep* para determinar se o endereço IP do navegado aparece no arquivo *ipaddrs*. O resultado da busca determina que o script siga o comando *then* ou siga o comando *else*, partes do enunciado *if*. Se o endereço é encontrado, o script emite uma mensagem que a requisição não é a primeira. Se o endereço não esta presente, o script adiciona o endereço ao arquivo, e então emite uma mensagem que aquele é o primeiro contato.

Quando executado, o script na figura 30.3 gera três linhas de resposta. Duas linhas de cabeçalho seguidas por uma única linha de texto. A linha de texto contém o endereço IP do computador, e conta se alguma requisição foi feita anteriormente. Por exemplo, a primeira vez que o computador 128.10.2.26 requisita o documento, o usuário receberá a seguinte linha de resposta:

This is the first contact from computer 128.10.2.26

As requisições subseqüentes para o documento feitas pelo mesmo computador produzira:

Computer 128.10.2.26 has requested this URL previously.

26.11 O script CGI com Informação de Estado de Curto Termo

Para entender como a informação de estado pode ser codificada na URL, lembre que o texto após a marca (?) na URL é passada ao programa de documento dinâmico como argumento. Como o programa de documento dinâmico cria uma cópia do documento necessitado, cada chamada do programa pode produzir um documento que contém um conjunto de URL. Em particular, qualquer texto que o programa colocar após a marca de questão (?) na URL se tornará o argumento se o usuário selecionar a URL.

Como exemplo, considere o programa CGI na figura 30.4 abaixo:

```
#!/bin/sh

echo Content-type: text/html
echo

N=$QUERY_STRING
echo "<HTML>"

case "x$N" in

x)      N=1
        echo "This is the initial page.<BR><BR>"
        ;;

x[0-9]*) N=`expr $N + 1`
        echo "You have refreshed this page $N times.<BR><BR>"
        ;;

*)      echo "The URL you used is invalid.</HTML>"
        exit 0
        ;;

esac

echo "<A HREF=\"http://$SERVER_NAME$SCRIPT_NAME?$N\">"
echo "Click here to refresh the page.</A></HTML>"
```

O script na figura 30.4 é mais complexo que o exemplo anterior. Primeiro, o script emite um documento HTML ao invés de um texto comum. Segundo, o script usa a variável global *QUERY_STRING* para determinar quando o servidor passou um argumento ao script. Terceiro, o script concatena as variáveis *SERVER_NAME* e *SCRIPT_NAME* para obter a URL do script, e adiciona o marca de questão (?) e conta a variável *N*. Quarto, o script usa um enunciado *case* para escolher dentre três possibilidades: a string do argumento está vazia, a string do argumento contém um inteiro, ou a string do argumento contém algo mais. Em dois casos, o script emite resposta apropriada; no ultimo caso, o script reporta que a URL é invalida.

Para entender como o script passa a informação de estado, assuma que o servidor www.nonexist.com tenha sido configurado para que o script corresponda ao caminho */cgi/ex4*. Quando o usuário executa a URL:

http://www.nonexist.com/cgi/ex4

O script irá gerar seis linhas de resposta:

```
Content-type: text/html

<HTML>
This is initial page. <BR><BR>
<A HREF = "http://www.nonexist.com/cgi/ex4?1">
Click here to refresh the page. </A> </HTML>
```

Como o cabeçalho especifica que o documento é do tipo HTML, o navegador ira interpretar os comandos HTML e exibir três linhas de resposta.

This is the initial page.

[Click here to refresh the page.](#)

Embora isto não apareça na tela do usuário, a informação de estado está embutida na URL do documento. Para ver a informação de estado, olhe para a URL na versão HTML do documento. Os dois últimos caracteres, *?1*, codificam a informação sobre o número de vezes que a página foi atualizada. Se o usuário clicar na segunda sentença, o navegador irá requisitar a seguinte URL:

<http://www.nonexist.com/cgi/ex4?1>

Quando executado, o script irá descobrir que *QUERY_STRING* contém *1*. Assim, o script irá gerar seis linhas de resposta:

Content-type: text/html

```
<HTML>
You have displayed this page 2 times. <BR><BR>
<A HREF =” http://www.nonexist.com/cgi/ex4?2”>
Click here to refresh the page. </A> </HTML>
```

Quando o navegador exibir o documento, o tela irá mudar e mostrará a seguinte mensagem:

You have displayed this page 2 times.

[Click here to refresh the page.](#)

Note que o novo documento tem o argumento *?2* embutido na URL. Assim, se o usuário clique na segunda sentença novamente, o navegador mostrará que a página foi exibida 3 vezes.

É importante saber que o programa de documento dinâmico não sabe quantas vezes que a página foi atualizada – o programa meramente reporta o que o argumento especifica. Por exemplo, se o usuário manualmente entrar com a URL:

<http://www.nonexist.com/cgi/ex4?25693>

O script irá reportar incorretamente que a página foi mostrada 5694 vezes.

26.12 Formulário e Interações

A noção de embutir a informação de estado na URL se tornou generalizada e estendida. O HTML permite ao servidor declarar que o documento é um formulário que contém um ou mais itens que o usuário deve fornecer. Para cada item, o formulário especifica um nome para o item. Quando o usuário seleciona um hyperlink no formulário, o navegador adiciona um argumento string a URL que contém cada um dos itens nomeados. Assim, o programa dinâmico pode obter valores que o usuário fornece.

Por exemplo, suponha o formulário que contenha três itens chamados *AA*, *BB*, e *CC*, e suponha que o usuário forneceu os valores *yes*, *no*, e *maybe*. Quando é coletado na URL, o navegador cria a string:

?AA=yes, BB=no, CC= maybe

A qual é adicionada a URL especificada.

Capítulo 27 – Tecnologia Java para Documentos Ativos Web

27.1 Introdução

Este capítulo considera o terceiro tipo de documento Web: o ativo. O capítulo descreve a motivação para documentos ativos, explica os conceitos básicos, e compara a aproximação dos documentos ativos à tecnologia de um servidor usada para atualizações contínuas. Em adição, o capítulo descreve os passos de tradução usados para preparar e carregar um documento ativo, e mostra como o navegador é estendido antes de poder executar um documento ativo. Finalmente, o capítulo considera a tecnologia Java que é usada para criar e executar um documento ativo. O mesmo caracteriza a linguagem Java e apresenta um simples exemplo de aplicativo Java.

27.2 Uma maneira precoce de Atualizações Contínuas

Quando o http e os navegadores Web foram projetados, se tornou claro que os documentos dinâmicos não irão satisfazer a todos os propósitos. Em particular, como a informação contida em um documento dinâmico é fixa quando o documento é criado, o documento dinâmico não pode atualizar informações na tela de exibição quando as informações mudam e não pode modificar imagens gráficas para prover uma animação.

Duas técnicas foram inventadas para permitir atualizações contínuas na tela do usuário. A primeira técnica passa a responsabilidade ao servidor. Chamada de *server push*, o mecanismo requer que o servidor gere e envie novas cópias do documento periodicamente. Na essência, o servidor deve executar um aplicativo associado com o documento dinâmico continuamente. Quando este é executado, o programa gera uma resposta “fresca”, a qual o servidor envia para o navegador exibir. Do ponto de vista do usuário, o conteúdo da página continua mudando enquanto esta sendo visualizada.

Embora este método forneça a habilidade de mudar as informações continuamente, o *server push* tem duas desvantagens: causa sobre carga no servidor e introduz delay. Para entender como a sobrecarga é significativa, observe que o servidor deve satisfazer muitos clientes. Quando o usuário visualiza uma página que requer server push, o servidor deve executar um programa de documento ativo associado com a página. Se muitos clientes requisitam documentos que utilizam o server push, o servidor deve executar vários aplicativos ao mesmo tempo. Mais importante, como os conteúdos do documento dinâmico podem depender da fonte da requisição, o servidor deve executar cópias separadas do documento dinâmico, para cada requisição.

O delay é causado pelos limites disponíveis na CPU e a largura de banda da rede. Quando N aplicativos são executados ao mesmo tempo em um computador, cada programa recebe no máximo $1/N$ do poder de processamento da CPU. Assim, o montante da CPU disponível para os aplicativos cai rapidamente com o crescimento do número de requisições. Se muitos navegadores visualizam estes documentos simultaneamente, o CPU no servidor terá uma sobrecarga, e as atualizações serão atrasados.

Largura de banda limitada pode introduzir também atrasos. O server push requisita que cada navegador mantenha uma conexão TCP ativa, através da qual o servidor envia as atualizações continuamente. Infelizmente, a maioria dos computadores que executam servidores tem apenas uma conexão de Internet que os dados deverão passar. Se muitos aplicativos tentam enviar dados, a rede pode ter um engarrafamento. Para introduzir igualdade, o sistema

operacional requisita que os aplicativos tomem turnos para enviar pacotes. Como resultado, um aplicativo individual tem atraso ao esperar o acesso à rede.

27.3 Documentos Ativos e Sobrecarga no Servidor

A utilização de documentos ativos é a segunda técnica usada para fornecer atualizações contínuas de informação ao cliente, sendo uma maneira que evita sobrecargas no servidor. Ao invés de requisitar computações de atualizações contínuas no servidor para vários clientes, o documento ativo aproxima a computação móvel ao navegador. Quando o navegador requisita um documento ativo, o servidor retorna uma cópia do programa e o navegador o executa localmente. Uma vez que o servidor fornece a cópia do documento, este não tem responsabilidades futuras de execução e atualizações – o navegador lida com toda computação localmente. Assim, diferentemente do server push, o documento ativo não requisita que o servidor use grandes quantidades de tempo destinadas ao processamento. Além do que, como o método não requisita que o servidor transmita atualizações continuamente, o documento ativo não usa muito da largura de banda.

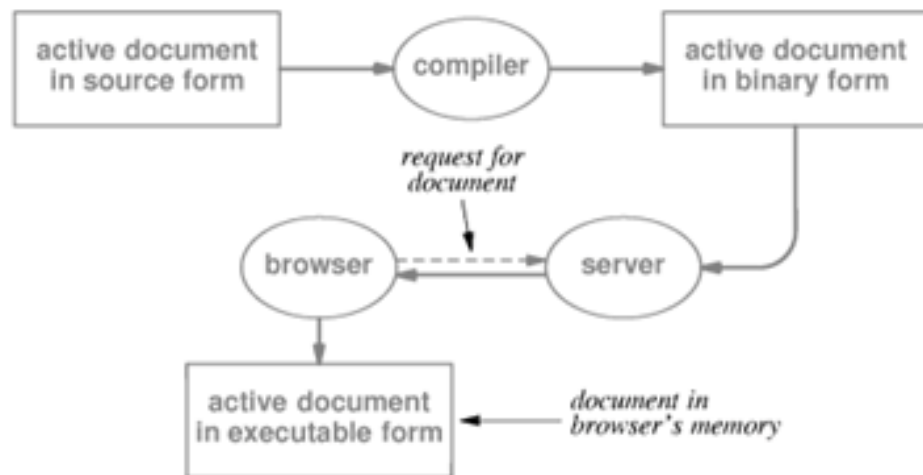
Um documento ativo pode introduzir menos sobrecarga ao servidor do que o documento dinâmico. Para entender como, é importante notar três fatos. Primeiro, para propósitos de transporte, o navegador e servidor tratam o documento ativo como um documento estático. Cada documento ativo reside em um arquivo no servidor. O conteúdo de um documento ativo não muda ao menos que o programador revise o documento. Assim, diferente do documento dinâmico, o navegador pode armazenar em cachê uma cópia do documento ativo. Segundo, o documento ativo não contém todo software – o navegador que executa um documento ativo contém o software de suporte. Terceiro, o documento ativo pode ser traduzido para uma representação compactada, que tem tamanho menor que o documento dinâmico. Assim, de vários pontos de vista tais como armazenamento, acesso e transporte os documentos ativos podem ser executados com mais eficiência.

27.4 Representação e Tradução de um Documento Ativo

Qual representação deve ser usada para um documento ativo? A resposta é complexa. Como o documento ativo é um programa que especifica como computar e exibir informações, muitas representações são possíveis. Além do que, nenhuma representação única é ótima para todos os propósitos – varias representações são necessárias. Por exemplo, quando o programador cria um programa de computador, o programador precisa de uma forma que possa ser entendida e facilmente manipulada por humanos. Para alcançar a maior velocidade de execução, o hardware requer que o programa seja armazenado na forma binária. Finalmente, uma representação compactada minimiza o delay e minimiza o consumo da largura de banda necessário para transferir o programa via Internet.

Para satisfazer todos os requerimentos, a tecnologia de documento ativo usa a mesma aproximação geral para a representação do programa como um programa convencional de uma linguagem: a tradução. Em particular, a maioria dos sistemas de documentos ativos define múltiplas representações, onde cada representação é otimizada para um propósito. O software é então criado e pode traduzir automaticamente dentre várias representações. Assim, o programador pode definir e editar o documento usando uma forma, enquanto o navegador que executa o documento pode utilizar outra forma.

A figura 31.1 abaixo ilustra três representações comuns dos documentos ativos, e mostra as traduções de uma forma à outra.



Como a figura mostra, um documento ativo começa com uma *source representation*. A source representation contém declarações para os dados locais e algoritmos de implementação usados para computar e exibir informações. Em adição, a source representation permite ao programador inserir comentários que ajudem humanos entenderem o programa. De fato, como a source representation de um documento ativo é similar a source representation de programas de computador, o programador pode usar ferramentas convencionais de programação como editores de texto para criar e mudar a fonte de um documento ativo.

Embora algumas tecnologias de documento ativo obriguem que o navegador aceite e interpretem um documento source, a maioria dos sistemas fornece uma otimização para grandes documentos: o compilador é usado para traduzir um documento com source representation para a *binary representation*, similar a representação de objeto usado em programas convencionais. Na binary representation, cada declaração executável é representado na forma binária, e identificadores no programa são substituídos por marcas binárias. Também como um arquivo objeto convencional, a representação binária inclui uma tabela de símbolos que grava as relações entre nomes na representação fonte e localizações na representação binária.

Após o navegador obter uma cópia do documento ativo na forma binária, o software no navegador traduz o documento para a *executable representation*, e carrega o programa resultante dentro da memória do navegador. A tradução é similar aos passos finais que um sistema operacional realiza quando carrega um programa na forma objeto. O navegador deve resolver as diferenças que permaneceram e deve conectar o programa à biblioteca de rotinas e as funções do sistema. Uma vez que a conexão esta completa, a tabela de símbolos não é mais necessária – todo o documento foi traduzido para valores binários.

27.5 Tecnologia Java

Desenvolvida pela Sun Microsystems, *Java* é o nome de uma tecnologia específica usada para criar e executar documentos ativos. Java utiliza o termo *applet* para descrever um documento ativo e distinguir documentos ativos de programas computacionais convencionais. A tecnologia Java inclui três componentes chaves relacionadas ao applets:

- *Linguagem de Programação.* Java inclui uma nova linguagem de programação que pode ser usada como linguagem fonte para programas de computador e para Java applets.
- *Ambiente de Tempo de Execução.* O sistema Java define um ambiente de tempo de execução que fornece as facilidades necessárias para executar um programa Java.
- *Biblioteca de Classes.* Para tornar os applets fáceis de escrever, Java fornece uma grande biblioteca que contém softwares para realizarem muitas das tarefas que o applet realiza.

Na prática, isso pode tornar a distinção entre os componentes difícil, pois estes são projetados para trabalharem juntos. Por exemplo, a linguagem contém atributos que dependem do suporte de tempo de execução, e a biblioteca contém o software que fornece a interface que facilita o ambiente de tempo de execução. As próximas seções caracterizam cada componente separadamente; seções à frente explicam as relações entre eles.

27.6 A Linguagem de Programação Java

Quando o programador criar um documento ativo, o programador escreve o programa na linguagem fonte. Ao invés de contar com uma linguagem existente, a tecnologia Java define uma nova linguagem de programação que tenta tornar a escrita e leitura de programas fontes convenientes aos applets. Formalmente, a linguagem é chamada de *Java Programming Language*; informalmente, os programadores utilizam a abreviatura *Java*.

27.6.1 Características da Linguagem

A linguagem Java pode ser caracterizada como:

- *Alto Nível.* Como a maioria das linguagens de programação, Java é classificada como linguagem de programação de alto nível, pois é projetada para esconder detalhes de hardware e tornar fácil a computação expressa para o programador.
- *Propósitos Gerais.* Embora a linguagem fosse originalmente projetada como uma ferramenta para escrever applets, a linguagem teve poder suficiente para se tornar útil em outras aplicações. Mais importante, não existem limitações que previnam a linguagem Java de ser usada por programas.
- *Similaridade ao C++.* A linguagem Java descende da linguagem de programação C++. Assim, semelhanças sintáticas e semânticas existem entre as duas linguagens.
- *Orientada ao Objeto.* Como em outras linguagens orientadas ao objeto, Java requer que o programador defina um conjunto de objetos e cada objeto contém dados bem como *métodos* para operar os dados.
- *Dinâmica.* A instância de um objeto é criada dinamicamente no tempo de execução. Além do que, Java permite algumas ligações se atrasarem até no tempo de execução.

- *Fortemente Classificada em Tipos.* No Java, cada item de dados é declarado com um tipo, e a linguagem proíbe que operações sejam aplicadas aos dados que tem tipos diferentes dos especificados para a operação.
- *Checação de Digitação Estática.* No Java, a checagem dos dados digitados não ocorre no tempo de execução. A checagem de compatibilidade de todos os itens no programa é determinada quando o programa é compilado.
- *Concorrência.* O Java permite ao programa ter vários processos de controle; os processos são executados simultaneamente.

27.6.2 Similaridades ao C++

O Java sofre uma forte semelhança com o C++; a linguagem mantém as mesmas construções semânticas do C++ bem como muito da sintaxe. Por exemplo, a maioria das declarações é encontrada no C++. Em adição, o mecanismo de tipos no Java usa hierarquia de classes.

As diferenças básicas entre Java e C++ ocorrem porque os projetistas desejaram tornar a programação mais fácil e os programas mais confiáveis. Consequentemente, Java teve que reduzir alguma das complexidades do C++. Em particular, Java omite características de linguagem que são pouco usadas, facilmente más compreendidas ou que tendam a erros. Por exemplo, Java não tem operador de sobre-carregamento, múltiplas heranças, ou coerção automática extensiva de dados. Em alguns casos, Java mantém características do C++, mas restringe o uso das mesmas.

27.7 O Ambiente do Tempo de Execução no Java

A tecnologia Java define um ambiente do tempo de execução no qual os programas Java são executados. O ambiente do tempo de execução é caracterizado por:

- *Execução Interpretativa.* Embora o programa Java possa ser compilado em um código para um computador específico, os projetistas planejaram que a linguagem seja usada com um intérprete. Isto é, diferente dos compiladores tradicionais que traduzem o programa fonte para o programa binário objeto para uma arquitetura específica, o compilador Java traduz o programa Java para uma representação binária independente chamada *Java bytecode representation*. Um programa chamado *interpreter* lê a representação bytecode e interpreta as instruções.
- *Automatic Garbage Collection.* O gerenciamento de memória no Java difere dramaticamente do gerenciamento de memória usado em linguagens como C e C++. Ao invés de requisitar que o programa chame os procedimentos como *malloc* e *free* para alocar e liberar memória, o sistema de tempo de execução Java fornece o *automatic garbage collection*. Isto é, o programa pode depender do sistema tempo de execução para achar e exigir uma memória que não esta sendo usada. Na essência, quando um programa descarta todas referencias ao objeto, o garbage collector exige que o objeto seja alocado na memória.
- *Execução de Múltiplos Processos.* O sistema de tempo de execução Java fornece suporte para processos concorrentes serem executados. Na essência, o sistema de tempo de execução contém partes do sistema operacional que

trabalham agendando e fazendo trocas. Quando há trocas no processamento entre processos ativos, o sistema permite que todos os processos prossigam.

- *Acesso a Internet.* O sistema de tempo de execução Java contém uma biblioteca socket que os programas usam para acessar a Internet. Em particular, o programa Java pode se tornar um cliente – o programa pode usar o TCP ou UDP para contatar um servidor remoto.
- *Suporte Gráfico.* Como o applet necessita controlar a exibição gráfica, o sistema de tempo de execução contém mecanismos básicos que permitem ao programa Java criarem janelas na tela que contenham texto ou gráficos.

A linguagem Java e o sistema de tempo de execução são projetados para fazer o Java independente do hardware do computador. Por exemplo, a linguagem não contém nenhum atributo de implementação definida. Ao invés disso, o manual de referência da linguagem define o significado de cada operador e instrução sem ambigüidade. Em adição, a representação bytecode é independente do sistema computacional base. Consequentemente, uma vez que um programa Java for compilado na representação bytecode, o programa produz exatamente a mesma resposta em qualquer computador.

Usar uma máquina de representação binária independente e um intérprete torna os programas Java portáveis através de várias arquiteturas de computadores. Considere um programa Java compilado em um computador com processador Pentium, armazenado em um servidor Web que usa processador Sparc, e executado em computador que usa o processador Alfa. A mesma representação bytecode irá resultar se o programa é compilado pelo compilador Java que é executado em outra arquitetura. Similarmente, a mesma resposta será o resultado se o compilador é executado em outro computador.

Manter o applet independente do sistema básico de hardware é essencial por três razões. Primeiro, na Internet os usuários tem vários tipos de computadores. A independência entre máquinas permite o navegador ser executado em uma marca arbitrária de computador, fazer download e executar uma cópia do documento ativo. Segundo, a independência de máquinas garante que o documento irá produzir a resposta correta em todos navegadores. Terceiro, a independência de máquinas reduz drasticamente os custos de criação do documento, porque o programador pode construir e testar uma versão do documento ativo ao invés de construir uma versão para cada tipo de computador.

27.8 A Biblioteca Java

O terceiro componente da tecnologia Java é a biblioteca de definições de classe. A biblioteca é extensa – contém dúzias de definições de classes com aproximadamente 2000 métodos individuais. Mais importante, a biblioteca de classes contém classes que transpõe uma variedade de necessidades. Por exemplo, a biblioteca contém classes para:

- *Manipulação Gráfica.* As rotinas de manipulação gráfica na biblioteca permitem ao programa Java ter controle preciso dos conteúdos e aparência da tela do usuário.
- *Rede I/O de Baixo Nível.* A biblioteca contém classes que fornecem acesso ao nível do socket I/O com facilidade no

sistema de tempo de execução. O programa Java pode usar as facilidades da rede de baixo nível I/O para enviar e receber datagramas UDP, ou abrir uma conexão TCP para enviar e receber streams de dados.

- *Interação com Servidor Web.* O applet talvez precise de acesso a documentos estáticos ou dinâmicos ou até de outros applet. Por exemplo, um applet pode seguir uma URL para retomar e exibir um documento estático. A biblioteca contém classe para lidar com estes casos.
- *Acesso ao Sistema de Tempo de Execução.* A biblioteca Java contém classes que o applet pode usar para acessar facilidades no ambiente de tempo de execução. Por exemplo, um programa sendo executado pode requisitar a criação de um processo.
- *Arquivos I/O.* O applet usa as facilidades do arquivo I/O para manipular arquivos no computador local. O arquivo I/O é usado por programas que salvam informações de estado de longo termo.
- *Estruturas de Dados Tradicionais.* A biblioteca contém classes que definem estruturas de dados tradicionais. Por exemplo, o programa pode usar a classe *dictionary* para criar uma estrutura de dados eficiente para armazenar itens para devolução futura.
- *Captura de Eventos.* Eventos ocorrem quando o usuário aperte ou solte o botão do mouse ou tecla no teclado. A biblioteca contém classes que permite ao programa Java capturar e lidar com estes eventos.
- *Lidar com Exceções.* Quando uma condição inesperada ou um erro acontece em um programa Java, o ambiente de tempo de execução levanta um *exception*. A biblioteca Java contém um conjunto de classes que tornam a exceção fácil de lidar.

27.9 O Conjunto de Ferramentas Gráficas

O ambiente de tempo de execução fornece facilidades que permite ao applet manipular a tela do usuário, e a biblioteca Java contém o software que permite uma interface gráfica de alto nível. Juntos, o suporte gráfico de tempo de execução e biblioteca gráfica são conhecidos como *graphics toolkit*, este conjunto de ferramentas gráficas na Java é chamado de *Abstract Window Toolkit* (AWT). São duas as razões porque o Java necessita de um conjunto de ferramentas gráficas. Primeiro, o propósito central de um applet é a exibição complexa – o applet é usado quando uma exibição estática é inadequada.

Segundo, o programa que controla uma exibição gráfica deve especificar muitos detalhes. Por exemplo, quando o programa Java necessita exibir um novo item ao usuário, o programa deve escolher entre exibir em uma subárea na janela existente ou criar uma nova janela independente. Se a nova janela é criada, o programa deve especificar um cabeçalho para a janela, o tamanho da janela, as cores a serem usadas, e detalhes de onde colocar a janela e quando a janela tem uma barra de rolagem.

O AWT não especifica um estilo particular de gráficos ou nível de interação. Ao invés disso, o conjunto de ferramentas classes tanto para nível baixo e nível alto de manipulação; o programador pode escolher se gerencia os detalhes ou chama os métodos do conjunto de ferramentas para fazer isso. Por exemplo, o conjunto de ferramentas não precisa ditar a forma da janela. Em uma mão, o conjunto inclui classes de baixo nível para criar um painel retangular em branco na tela, desenhar linhas e polígonos, ou capturar teclas e eventos do mouse que ocorrem no painel. Em outra mão, o conjunto inclui classes de alto nível que fornecem uma janela completa, com cabeçalho, bordas e barras verticais e horizontais. O programador pode escolher quando usar a janela estilizada fornecida ou programar os detalhes.

O programador deve escolher entre as facilidades de alto nível ou de baixo nível para interagir com o usuário. Por exemplo, um applet pode usar classes de alto nível do conjunto de ferramentas para criar botões, menus pull-down, ou caixas de dialogo na tela. Alternativamente, o applet pode criar estes itens usando conjunto de ferramentas de baixo nível de classes para desenhar linhas, especificar sombras, e controlar a fonte usada no texto exibido.

Como o applet pode necessitar de interações com documentos estáticos, o conjunto de ferramentas fornece classe que realizam a navegação Web convencional. Por exemplo, dada uma URL, o applet pode usar o conjunto de ferramentas de classes para buscar e exibir um documento HTML estático, buscar e exibir uma imagem, ou um clipe de áudio.

27.10 Usando os Gráficos Java em um Computador Particular

Como o AWT é projetado para ser independente de uma marca específica de hardware e de um sistema gráfico, a tecnologia Java não necessita usar o display de hardware diretamente. Ao invés disso, o Java pode executar em cima dos sistemas convencionais de janelas. Por exemplo, se um usuário executa um navegador no computador que usa o Sistema de Janela X, o applet Java usa o X para criar e manipular a janela na exibição. Se outro usuário executa outro sistema diferente de janela, o applet ira usar aquele sistema de janela do usuário para criar e manipular exibições.

Como o applet Java pode trabalhar com múltiplos sistemas de janela? A resposta reside em um mapeamento importa construído dentro do sistema de tempo de execução e um conjunto de funções intermediárias. O sistema de tempo de execução Java contém um software intermediário de camada. Cada função é intermediada pela camada, que usa o sistema de janela do computador para programar uma operação gráfica Java específica. Antes de o applet executar, o ambiente de tempo de execução estabelece um mapeamento entre cada método gráfico Java e a função intermediaria apropriada. Quando o applet

requisita uma operação do AWT, o controle passa para um método na biblioteca Java. A biblioteca de métodos então repassa o controle para a função intermediária apropriada, a qual usa o sistema de janelas do sistema para fornecer a operação. O conceito é conhecido como *factory*.

A vantagem de usar uma camada intermediária de software são a portabilidade e generalização. Como as funções intermediárias usam o sistema de janelas do computador, apenas uma versão da biblioteca AWT é necessária – a biblioteca não contém códigos relacionados a hardwares gráficos específicos. Como a ligação entre os métodos gráficos Java e o sistema de software das janelas do computador é estabelecido pelo sistema de tempo de execução, o applet Java é portátil. A desvantagem principal é a imprecisão que chega devido a interpretações das operações pela camada intermediária de software. Como resultado, o programa Java executado em dois computadores não produz sempre dois resultados idênticos.

27.11 Interpretes Java e Navegadores

Lembre do capítulo 29 que o navegador pode conter um ou mais interpretes. O navegador convencional contém um interprete HTML usado para exibir documentos estáticos ou dinâmicos. O navegador que executa o Java contém dois: um interprete HTML e um interprete adicional para os applets.

O interprete Java é um programa complexo. O coração do interprete é um simples loop que simula o computador. O interprete mantém um *instruction pointer* que é iniciado no começo do applet. A cada iteração, o interprete busca o código bytecode encontrado no endereço no instruction pointer. O interprete então decodifica a instrução e realiza a operação específica. Por exemplo, se o interprete acha o bytecode *add* e dois operando inteiros, o interprete adiciona os dois inteiros, atualiza o instruction pointer, e continua com a próxima iteração.

Em adição ao decodificador de instruções básicas, o interprete Java deve incluir suporte ao ambiente de tempo de execução. Isto é, o interprete Java deve ser hábil para exibir gráficos na tela do usuário, acessar a Internet, e realizar I/O. O interprete também deve ser projetado para permitir que um applet use as facilidades no navegador que recuperam e mostram documentos estáticos e dinâmicos. Assim, o interprete Java em um navegador deve ser hábil para comunicar com o cliente http do navegador e com o interprete HTML.

27.12 Compilando um Programa Java

Como a figura 31.1 ilustra, o applet Java deve ser compilado e armazenado em um servidor Web, assim o navegador pode fazer o download e executar o applet. A tecnologia Java inclui um compilador chamado *javac*. O programador executa o *javac* para compilar o programa fonte Java para a representação bytecode Java.

A entrada para o *javac* é programa fonte Java. O nome do arquivo de entrada deve terminar com o sufixo *.java*. O *Javac* verifica se o programa é sintaticamente correto, traduz o programa para a representação bytecode, e coloca a saída em um arquivo com o sufixo *.class*.

O programa Java consiste em uma sequência de declarações. Um item que é declarado *public* é exportado para ser disponível em outros applet; um

item que é declarado *private* não é exportado e não pode ser referenciado externamente. O arquivo fonte deve conter exatamente uma classe pública, a qual deve usar o mesmo nome do prefixo do arquivo fonte. Isto é, a classe pública no arquivo *zzz.java* deve ser nomeada *zzz*. Assim, o arquivo que contém o bytecode compilado da classe *zzz* tem o nome *zzz.class*.

27.13 Um Exemplo de Applet

A figura 31.2 abaixo contém um simples exemplo de applet para ilustrar conceitos básicos. O applet começa com a seqüência de declarações *import*. Cada declaração *import* direciona o compilador a usar um conjunto de classes da biblioteca Java.

```
import java.applet.*; import java.awt.*;

public class clickcount extends Applet {
    int count;
    TextField f;

    public void init() {
        count = 0;
        add(new Button("Click Here"));
        f = new TextField("The button has not been clicked at all.");
        f.setEditable(false);
        add(f);
    }

    public boolean action(Event e, Object arg) {
        if (((Button) e.target).getLabel() == "Click Here") {
            count += 1;
            f.setText("The button has been clicked " + count + " times.");
        }
        return true;
    }
}
```

Cada enunciado *import* contém o molde que especifica o conjunto de classes a serem importadas. No molde, a maioria dos caracteres representa um encontro literal; um asterisco, *, é usado para denotar uma combinação de caracteres que seja o mesmo que uma string arbitrária. Assim, o enunciado:

```
import java.applet.*;
```

Especifica que o compilador deve importar todas as classes que o nome começam com a string *java.applet*

No exemplo de applet, os dois enunciados *import* são usados para importar o padrão Applet e Abstract Window Toolkit classes. Como resultado, o applet terá acesso aos métodos que criam o ambiente applet e os métodos para manipular exibições.

O exemplo applet define uma subclasse do tipo *Applet* chamada *clickcount*. Em adição aos métodos que a classe herda da *Applet*, a classe *clickcount* redefine dois métodos: *init* e *action*.

Quando o navegador chama a referencia ao applet, o Java cria uma nova instancia do objeto *clickcount*. O *Clickcount* define dois itens de dados: um inteiro chamado *count* e um *Textfiel* chamado *f*. Como parte da criação, o sistema de tempo de execução invoca o método *init*. Este método realiza vários passos da iniciação. Após configurar *count* para zero, *init* usa o operador *new* para criar a instancia de objeto *Button*, e chama *add* para adicionar o botão a janela do applet. Finalmente, *init* usa *new* para criar a instancia de um *Textfield* e designar ele a variável *f*, usa o método *setEdittable* para tornar o texto em *f* apenas leitura, e *add* para associar *f* a janela do applet.

A figura 31.3 ilustra como o display irá aparecer após *init* ter adicionado dois itens gráficos a janela do applet.



O coração do exemplo é o método *action*. Se o botão nomeado *Click Here* é selecionado, o applet incrementa o valor da variável *count* e muda a exibição. Para fazer isso, o applet invoca o método *setText*, o qual substitui a mensagem em *f*. Assim, cada vez que o usuário clicar no botão, o display muda. A figura 31.4 abaixo mostra como o display será após o usuário clicar uma vez no botão.



Do ponto de vista do usuário, o exemplo de applet parece operar similiarmente ao script CGI na figura 30.4. Do ponto de vista da implementação, os dois operam um pouco diferentes. Diferente do script CGI, o applet mantém a informação de estado localmente. Assim, o exemplo do applet não contata o servidor antes de atualiza a tela.

27.14 Chamando um Applet

Dois métodos são usados para chamar o applet. Primeiro, o usuário pode fornecer a URL do applet em um navegador que entende Java. Quando o navegador contata o servidor especificado na URL, o servidor irá informar ao navegador que o documento é um applet Java. Segundo, um documento HTML que contém o *applet tag* que se refere ao applet. Quando o navegador encontrar a etiqueta, o navegador contata o servidor para obter uma cópia do applet.

Na forma mais simples, o *applet tag* especifica a localização do arquivo *.class* que será copiado e executado. Por exemplo, suponha que o servidor Web na máquina www.nonexist.edu armazene o arquivo *bbb.class* no diretório *example*. A URL para o applet é:

<http://www.nonexist.edu/example/bbb.class>

A *applet tag* que especifica a localização contém dois itens: o *codebase* que especifica a localização do servidor e o caminho, e o *code* que fornece o nome do arquivo class. Por exemplo, o applet tag abaixo:

```
<applet codebase="http://www.nonexist.edu/example/" code="bbb.class">
```

Contém a mesma informação que a URL acima. O navegador que encontrar a etiqueta acima irá contatar o servidor, obter uma cópia do arquivo class, criar uma instancia da classe *bbb*, e chamar o método *init*.

27.15 Exemplo da Interação com o Navegador

Um applet pode interagir com o cliente HTTP e com o interprete HTML em um navegador. O applet usa o cliente HTTP do navegador para receber documentos e o interprete HTML do navegador para mostrar as informações da pagina. O applet na figura 31.5 abaixo ilustra o conceito.

```

import java.applet.*; import java.net.*; import java.awt.*;

public class buttons extends Applet {

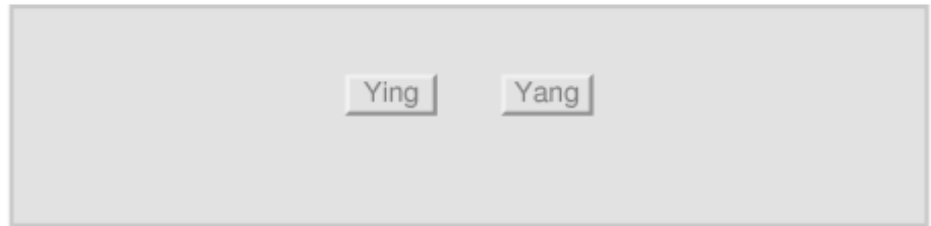
    public void init() {
        add(new Button("Ying"));
        add(new Button("Yang"));
    }

    public boolean action(Event e, Object arg) {
        if (((Button) e.target).getLabel() == "Ying"){
            try {
                getAppletContext().showDocument(new
                    URL("http://www.nonexist.com/ying"));
            }
            catch( Exception ex ) {
                // note: code to handle the exception goes here //
            }
        }
        else if (((Button) e.target).getLabel() == "Yang"){
            try {
                getAppletContext().showDocument(new
                    URL("http://www.other.com/yang"));
            }
            catch( Exception ex ) {
                // note: code to handle the exception goes here //
            }
        }
        return true;
    }
}

```

Como o exemplo mostra, o Java fornece bibliotecas de alto nível que simplificam a interação entre um applet e o navegador. Para usar esses instrumentos, o applet começa com três enunciados *import*. Em adição ao Applet e Abstract Window Toolkit, o applet importa o *java.net.**, o qual contém definições de classes relacionadas a rede.

As operações do applet são diretas. Quando o navegador inicia o applet, o navegador cria uma instancia da classe *button*, e então chama o método *init*. O *init* cria dois objetos botões, e utiliza o método *add* para adicionar ambos ao display do applet. Um botão é nomeado *Yin*, enquanto o outro é nomeado *Yang*. Uma vez que o *init* esta completo, a tela aparece igual a figura 31.6 abaixo ilustra:



O código que interage com o navegador pode ser encontrado no método *action*. Quando o usuário clica no botão *Yin*, o método *action* realiza três tarefas. Primeiro, o enunciado *if* checa qual dos botões foram clicados. Segundo, o applet retorna o applet correto. Por exemplo, se o usuário clicar no botão nomeado *Yin*, o applet irá invocar:

```
new URL(http://www.nonexist.com/yin)
```

Requisitando ao cliente HTTP do navegador que ele consiga uma cópia do documento associado com <http://www.nonexist.com/yin>. A operação *new* cria um objeto local para armazenar o documento. Terceiro, o código Java:

```
getAppletContext().showDocument
```

Requisita ao interprete HTML que exiba o documento na tela do usuário. O navegador irá substituir a exibição atual com o novo documento; o usuário deve clicar no botão *voltar* no navegador para retornar ao applet.

27.16 Erros e Lidar com Exceções

No Java, todos os erros produzem exceções; um applet é requisitado para lidar com essas exceções explicitamente. Por exemplo, o applet na figura 31.5 usa o objeto URL para requisitar que o cliente HTTP acesse um documento. Se a URL está incorreta, a rede está com problemas, ou o servidor está indisponível, a invocação irá resultar em uma exceção.

O exemplo de applet utiliza a construção *try-catch* para lidar com exceções. A forma sintática é:

```
try { S } catch { E } { H }
```

Onde se a exceção *E* ocorre enquanto esta sendo executado *S*, invoque *H*. No código de exemplo, *E* é a variável *ex* é declarada para ser do tipo *exception*, e quem irá lidar com as exceções não é especificado, logo todas as exceções serão ignoradas. Assim, o exemplo simples ignora erros, e força o usuário clicar em um botão para repetir a operação.

27.17 Alternativas e Variações

Várias alternativas existem para a tecnologia Java descrita neste capítulo. Por exemplo, uma variação conhecida como *JavaScript* é usada por applets pequenos que não contêm códigos grandes ou complexos. Ao invés de compilar o applet para representação bytecode, o JavaScript fornece uma linguagem de script, e faz com que o navegador leia e interprete o script na forma fonte. Mais importante, o JavaScript pode ser integrado junto como HTML – uma página HTML pode ter uma função JavaScript que fornece uma simples interação com o usuário. Por exemplo, uma função JavaScript pode requisitar ao usuário que entre com uma informação, e então a informação é verificada antes de começar a comunicação com o servidor.

A variação JavaScript tem tanto vantagens quanto desvantagens. As principais vantagens são a simplicidade e facilidade de uso. Um pequeno script pode estar embutido em uma página. Como o script não necessita ser compilado, as mesmas ferramentas podem ser usadas para criar um script como página ordinária Web. A desvantagem principal é velocidade e ajuste da escala de aplicação – como a representação fonte é menos compacta que a representação bytecode, transportar um programa fonte demora mais tempo. Além do mais, o script demora mais tempo para ser interpretado do que um programa na representação bytecode.

Outras variações permitem ao programador escrever applet em uma linguagem diferente da Java Programming Language. Por exemplo, o sistema *AdaMagic* consiste em um compilador *Ada* que foi adaptado para produzir representação bytecode Java. O compilador aceita programas escritos na Ada Programming Language, e produz saídas compatíveis com o ambiente de tempo de execução Java.

Outras companhias também exploraram alternativas tecnológicas. Por exemplo, o Lucent Bell Laboratories desenvolveram uma tecnologia de documento ativo chamado *Inferno*. Embora o Inferno seja conceitualmente similar ao Java, os detalhes diferem dramaticamente.

Capítulo 28 – Administração de Rede (SNMP)

28.1 Introdução

Os capítulos anteriores descreveram uma variedade de aplicações que usam a Internet. Cada capítulo examina a estrutura de software para um só serviço e mostra como um cliente e um servidor interagem.

Esse capítulo expande nosso estudo sobre aplicações de rede considerando softwares de aplicação que administram o uso para medir ou controlar redes. Depois de explicar porque a administração de redes é importante e difícil, o capítulo descreve como o software de administração de rede opera e as funcionalidades que ele provê. Finalmente, o capítulo considera um exemplo específico de um protocolo de administração de rede e explica como o software para tal protocolo opera.

28.2 Administrando uma Internet

Um *administrador de rede* é uma pessoa responsável por monitorar e controlar os sistemas de hardware e software que compreendem uma internet. Um administrador trabalha para detectar e corrigir problemas que tornam a comunicação ineficiente ou impossível e para eliminar condições que produzirão o problema novamente. Devido a ambas as falhas de hardware e de software poderem causar problemas, um administrador de rede deve monitorar ambos.

A administração de uma rede pode ser difícil por duas razões. Primeira, a maioria das internets é heterogênea. Isto é, a internet contém componentes de hardware e de software produzidos por várias empresas. Pequenos erros de um vendedor podem ocasionar incompatibilidade de componentes. Segunda, a maioria das internets é grande. Em particular, a Internet global abrange muitos sites em diversos países ao redor do mundo. Detectar a causa de um problema de comunicação pode ser especialmente difícil se o problema ocorrer entre dois computadores em dois lugares.

Interessantemente, falhas que causam a maioria dos piores problemas são freqüentemente as mais simples de diagnosticar. Por exemplo, se o cabo coaxial em uma Ethernet é cortado ou o switch de uma LAN é desligado, os computadores conectados a essa LAN não serão capazes de enviar ou receber pacotes. Devido ao dano afetar a todos os computadores em uma rede, o administrador pode localizar a fonte da falha rápida e eficientemente. Similarmente, o administrador pode também detectar falhas catastróficas em um software como uma rota invalidada o que faria com que um roteador descartasse todos os pacotes para um dado destino.

Em contraste com isso, falhas intermitentes ou parciais freqüentemente geram um desafio difícil. Por exemplo, imagine um dispositivo de interface de rede que corrompe bits sem uma freqüência fixa ou um roteador que erra a rota de poucos pacotes e redireciona muitos outros corretamente. No caso de uma falha de interface intermitente, o checksum ou o CRC em um frame podem ser suficientes para esconder esse problema – o receptor descarta o frame corrompido e quem o enviou deve eventualmente retransmiti-lo. Por isso, da perspectiva de um usuário, a rede parece operar corretamente porque os dados eventualmente passam através do sistema.

Por isso que devido aos protocolos acomodarem a perda de pacotes, as falhas intermitentes do hardware ou do software podem ser dificilmente detectadas e isoladas.

28.3 O Perigo de Falhas Ocultas

Infelizmente, embora elas permaneçam ocultas, falhas intermitentes podem afetar o desempenho da rede. Cada retransmissão usa largura de banda da rede que poderia ser usada para enviar novos dados. Mais importante, as falhas em hardware frequentemente se tornam piores com o passar do tempo porque o hardware se deteriora. Com o aumento da frequência dos erros, mais pacotes devem ser retransmitidos. Devido às retransmissões diminuírem a taxa de transferência e aumentarem o delay, o desempenho geral da rede diminui.

Em redes compartilhadas, a falha associada a um computador pode afetar os outros. Por exemplo, suponha que a interface comece a falhar em um computador anexado a uma Ethernet. Devido à falha do hardware corromper alguns pacotes que o computador envia, o computador eventualmente os retransmitirá. Enquanto a rede é utilizada para retransmissão, outros computadores devem esperar para enviar seus pacotes. Por isso, cada retransmissão reduz a largura de banda disponível para todos os computadores.

28.4 Software de Administração de Rede

Como um administrador de redes pode encontrar problemas e isolar suas causas? A resposta está no *software de administração de rede* que permite que o administrador monitore e controle os componentes da rede. Por exemplo, o software de administração de rede permite que o administrador interrogue dispositivos como computadores, roteadores, switches e bridges para determinar seu status e obter as estatísticas sobre as redes as quais eles estão anexados. O software ainda permite que o administrador controle tais dispositivos mudando rotas e configurando as interfaces de rede.

28.5 Clientes, Servidores, Administradores e Agentes

Surpreendentemente, a administração de rede não é definida como parte dos protocolos de transporte ou de internet. Ao invés disso, os protocolos que um administrador de rede utiliza para monitorar e controlar os dispositivos de rede opera no nível de aplicação. Isto é, quando um administrador precisa interagir com um dispositivo de hardware específico, o software de administração segue o modelo convencional cliente-servidor: um programa de aplicação no computador do administrador age como um cliente e a aplicação no dispositivo de rede age como um servidor. O cliente no computador do administrador usa os protocolos de transporte convencionais (por exemplo, TCP ou UDP) para estabelecer comunicação com o servidor. Os dois então trocam pedidos e respostas de acordo com o protocolo de administração.

Para evitar confusão entre os programas de aplicação que o usuário executa e aplicações que são reservadas para os administradores de rede, os sistemas de administração de rede evitam os termos *cliente* e *servidor*. Ao invés disso, a aplicação cliente que roda no computador do administrador é chamada de um *administrador* e a aplicação que roda em um dispositivo de rede é chamada de um *agente*.

Pode parecer estranho que redes e protocolos de transporte convencionais sejam utilizados para a administração da rede. Afinal de contas, falhas nos protocolos ou no hardware utilizado podem impedir pacotes de viajar de ou para um dispositivo, tornando impossível o controle de um dispositivo enquanto falhas estiverem ocorrendo. Na prática, entretanto, usar um protocolo de aplicação para a administração da rede funciona bem por dois motivos. Primeiro, em casos onde uma falha do hardware evita a comunicação, o nível do protocolo não importa – o administrador pode se comunicar com aqueles componentes que permanecem operando e utiliza sucessos e falhas para ajudar a localizar o problema. Segundo, usar protocolos de transporte convencional significa que os pacotes de um administrador estarão sujeitos às mesmas condições de tráfego normal. Por isso, se o delay for grande, o administrador descobrirá imediatamente.

28.6 Protocolo de Administração de Rede Simples

O protocolo padrão utilizado para administrar uma internet é conhecido como *Simple Network Management Protocol (SNMP)*. O protocolo SNMP define exatamente como que um administrador se comunica com um agente. Por exemplo, o SNMP define o formato dos pedidos que um administrador envia para um agente e o formato das respostas que um agente retorna. Em adição, o SNMP define o significado exato para cada pedido e resposta possíveis. Em particular, o SNMP especifica que uma mensagem SNMP é codificada utilizando um padrão conhecido como *Abstract Syntax Notation.1 (ASN.1)*.

Embora todos os detalhes da codificação ASN.1 estejam além do escopo desse texto, um exemplo simples ajudará a explicar a codificação: considere o envio de um valor inteiro entre um agente e um administrador. Para acomodar grandes valores sem gastar espaço em cada transferência, o ASN.1 utiliza a combinação do tamanho e do valor para cada objeto a ser transferido. Por exemplo, um valor inteiro entre 0 e 255 podem ser transferidos em um só octeto (1 Byte). Valores inteiros dentro da faixa entre 256 e 65535 requerem dois octetos, enquanto que números maiores requerem três ou mais octetos. Para codificar um inteiro, o ASN.1 envia um par de valores, o tamanho, *L*, seguido de *L* octetos que contém o valor inteiro. Para permitir a codificação de grandes valores inteiros grandes e arbitrários, o ASN.1 permite também que o tamanho ocupe mais de um octeto; tamanhos estendidos normalmente não são necessário para valores inteiros usados com o SNMP. A tabela abaixo mostra alguns exemplos numéricos.

Inteiro em decimal	Equivalente em hexadecimal
27	1B
792	318
24567	5FF7
190345	2E789

Octeto de tamanho	Octetos de valor (em hexa)
01	1B
02	03 18
02	5F F7
03	02 E7 89

28.7 Paradigma Fetch-Store

O protocolo SNMP não define um grande conjunto de comandos. Ao invés disso, o protocolo utiliza o *paradigma Fetch-Store* no qual existem duas operações básicas: *fetch*, utilizada para obter um valor em um dispositivo, e *store*, utilizado para determinar um valor em um dispositivo. Cada objeto que pode ser lido ou escrito recebe um nome único; um comando que determina uma operação fetch ou store deve especificar o nome do objeto.

Deveria ser óbvio como operações fetch podem ser utilizadas para monitorar um dispositivo ou obter o seu estado: um conjunto de estados de dispositivo deve ser definido e ser dado um nome. Para obter informações sobre o estado, um administrador lê o valor associado àquele dado objeto. Por exemplo, um objeto pode ser definido para contar o número de frames que um dispositivo descarta devido a checksums incorretos. O software deve ser programado para incrementar o contador sempre que um erro de checksum for detectado. Um administrador pode utilizar o SNMP para obter o valor associado com o contador para determinar sempre que erros de checksum ocorrem.

Utilizando o paradigma fetch-store para controlar um dispositivo pode não parecer tão óbvio; operações de controle são definidas para transformar o efeito da escrita em um objeto. Por exemplo, o SNMP não inclui comandos separados de *reset* do contador de erros de checksum ou para *reiniciar* um dispositivo. Nesse caso do contador de erros de checksum, escrever um zero no objeto é intuitivo porque isso reinicia o contador para zero. Para operações como a reiniciação de um dispositivo, entretanto, um agente SNMP deve ser programado para interpretar o pedido de escrita e para executar a sequência correta de operações para alcançar o efeito desejado. Por isso, o SNMP pode definir um objeto de reiniciação e especificar que a escrita de um zero nele deveria causar a reiniciação do sistema. Na prática, entretanto, a maioria

dos sistemas não tem um contador de reiniciação – o software no agente deve checar explicitamente para uma operação de escrita que especifique o objeto de reiniciação e deve então executar uma sequência de passos para a reiniciação do sistema.

28.8 O MIB e Nomes de Objetos

Cada objeto para o qual o SNMP tem acesso deve ser definido e receber um nome único. Mais ainda, ambos os programas administrador e agente devem concordar nos nomes e significados das operações de fetch e store. Coletivamente, um conjunto de todos os objetos que o SNMP pode acessar é conhecido como *Management Information Base (MIB)*.

De fato, o SNMP não define uma MIB. Ao invés disso, o padrão SNMP só especifica o formato da mensagem e descreve como as mensagens são codificadas; um padrão separado especifica as variáveis MIB com o significado de operações fetch e store para cada variável.

Objetos em uma MIB são definidos com o esquema de nomenclatura ASN.1, o qual nomeia cada objeto com um prefixo longo que garante ao nome sua unicidade. Por exemplo, um número inteiro que conta o número de datagramas IP que um dispositivo recebeu pode ser chamado:

iso.org.dod.internet.mgmt.mib.ip.ipInReceives

Mais ainda, quando o nome do objeto é representado em uma mensagem SNMP, para cada parte do nome é designado um número inteiro. Por isso, em uma mensagem SNMP, o nome do *ipInReceives* é:

1.3.6.1.2.1.4.3

28.9 A Variedade de Variáveis MIB

Devido ao SNMP não especificar um conjunto de variáveis MIB, o desenvolvimento é flexível. Novas variáveis MIB pode ser definidas e padronizadas se necessário, sem mudar o protocolo base. Mais importante, a separação do protocolo de comunicação da definição de objetos permite que grupos de pessoas definam as variáveis MIB necessárias. Por exemplo, quando um novo protocolo é designado, o grupo que o criou pode também definir variáveis MIB que serão usadas para monitorar e controlar o software dele. Similarmente, quando um grupo cria um novo dispositivo de hardware, esse grupo pode especificar variáveis MIB usadas para monitorar e controlá-lo.

Como os principais desenvolvedores desejavam, muitos conjuntos de variáveis MIB tem sido criados. Por exemplo, existem variáveis MIB que correspondem aos protocolos UDP, TCP, IP e ARP, assim como as variáveis MIB para hardware de rede como Ethernet, Token Ring e FDDI. Em adição, grupos têm definido MIBs para dispositivos de hardware como bridges, switches e impressoras.

28.10 Variáveis MIB Que Correspondem a Vetores

Em adição as variáveis mais simples como os números inteiros que correspondem aos contadores, uma MIB pode incluir variáveis que correspondem a tabelas ou vetores. Tais definições são úteis porque elas correspondem a implementação de informação em um sistema operacional. Por exemplo, considere uma tabela de roteamento IP. Na maioria das implementações, a tabela de roteamento pode ser vista como um vetor de entradas, onde cada entrada contém um endereço de destino e o próximo salto utilizado para alcançá-lo.

Diferente das linguagens de programação convencionais, o ASN.1 não inclui uma operação de indexação. Ao invés disso, referências indexadas estão implícitas – quem envia

deve saber que o objeto referido é uma tabela e deve anexar a informação de indexação no nome dele. Por exemplo, a variável MIB:

Prefixo MIB padrão.ip.TabelaRoteamentoIP

corresponde a uma tabela de roteamento IP, da qual cada entrada possui vários campos. Conceitualmente, uma tabela é indexada pelo endereço IP de destino. Para obter o valor de um campo em particular dentro de uma entrada, o administrador especifica o nome na forma:

Prefixo MIB padrão.ip.TabelaRoteamentoIP.EntradaRotaIP.campo.endIPdest

onde o *campo* corresponde a um dos campos válidos em uma entrada e o *endIPdest* é um endereço IP de 8 Bytes que é usado para indexar. Por exemplo, o campo *ProxSaltoRotaIP* corresponde ao próximo salto de uma entrada. Quando converte a representação de números inteiros, o pedido de próximo salto fica assim:

1.3.6.1.2.1.4.21.1.7.destino

onde *1.3.6.1.2.1* é o prefixo MIB padrão, *4* é o código para *ip*, *21* é o código para *TabelaRoteamentoIP*, *1* é o código para *EntradaRotaIP*, *7* é o código para o campo *ProxSaltoRotaIP* e o *destino* é o valor numérico do endereço IP do destino.

Capítulo 29 – Segurança de Rede

29.1 Introdução

Nos capítulos anteriores foi descrito como os sistemas de hardwares e softwares de rede operam e explicado como aplicações cliente e servidor usavam os equipamentos de rede para se comunicar. Este capítulo considera o importante aspecto da segurança de rede. O capítulo caracteriza os problemas de segurança, descreve os níveis de segurança os usuários esperam de um sistema de rede e explica técnicas básicas utilizadas para melhorar a segurança da rede.

29.2 Redes Seguras e Políticas de Segurança

O que é a rede segura? Uma internet pode ser segura? Embora seja atrativo o conceito de uma rede segura para a maioria dos usuários, redes não podem ser simplesmente classificadas como segura ou não segura porque o termo não é absoluto – cada organização define o nível de acesso que é permitido ou negado. Por exemplo, algumas organizações guardam dados que são valiosos. Tais organizações definem uma rede segura para ser um sistema que previne pessoas de fora de acessarem os computadores delas. Outras organizações precisam tornar a informação disponível para pessoas de fora, mas proibi-las de alterá-la. Tais organizações podem definir uma rede segura como uma que permita acesso arbitrário aos dados, mas inclui mecanismos que previnem mudanças não autorizadas. Outros grupos focam em manter a comunicação privada: eles definem uma rede segura como sendo uma na qual nenhuma outra pessoa, a não ser o destino desejado, possa interceptar e ler uma mensagem. Finalmente, muitas organizações grandes precisam de uma definição complexa de segurança que permita acesso a dados ou serviços específicos que a empresa escolha, enquanto previne acesso ou modificação de dados e serviços importantes que são mantidos em privacidade.

Devido a não existência de uma definição absoluta para *rede segura*, o primeiro passo que uma empresa deve tomar para alcançar um sistema de segurança é definir as *políticas de segurança* dela. A política não especifica como alcançar a proteção. Ao invés disso, ela diz claramente e sem ambigüidades os itens que serão protegidos.

A definição de uma política de segurança de rede é complexa. A primeira complexidade vem do fato de que uma política de segurança de rede não pode ser diferente da política de segurança dos sistemas computacionais anexados à rede. Em particular, definir uma política para dados que viajam através de uma rede não garante que eles estarão seguros. Por exemplo, considere que os dados guardados em um arquivo podem ser lidos. A segurança de rede não pode prevenir que usuários não autorizados que tenham contas nesse computador obtenham uma cópia desses dados. Por isso, para ser eficiente, uma política de segurança deve ser aplicada a todo o momento. A política deve abranger os dados guardados em disco, os dados comunicados através de uma linha telefônica com um modem de dial-up, as informações impressas em papel, os dados transportados em mídia portátil como disquetes e os dados comunicados pela rede local de computadores.

Avaliar os custos e benefícios de várias políticas de segurança também adiciona complexidade. Em particular, uma política de segurança não pode ser definida a não ser que a empresa entenda o valor de sua informação. Em muitos casos, o valor da informação é difícil de ser estimado. Considere, por exemplo, um banco de dados de folha de pagamento simples que contenha um registro para cada empregado, suas horas trabalhadas e seu salário por hora. O aspecto de valor mais fácil de avaliar é o custo de recuperação. Isto é, calcular o tempo de mão de obra necessário para recriar ou verificar o conteúdo do banco de dados (por exemplo, de recuperar os dados de um arquivo ou realizar o trabalho necessário para coletar essa informação). Um segundo aspecto de valor está nas perdas financeiras que uma organização pode sofrer se a informação estiver incorreta. Por exemplo, se uma pessoa não autorizada

aumentar o salário por hora em um banco de dados de folha de pagamento, a companhia pode ter custos não esperados, pois os empregados receberiam dinheiro a mais. Um terceiro aspecto de valor está nos custos indiretos que podem ocorrer de violações na segurança. Por exemplo, se as informações da folha de pagamento se tornarem públicas, competidores podem escolher contratar trabalhadores, o que resulta em custos de contratação e treinamento de substitutos assim como o aumento de salário necessário para manter outros empregados.

29.3 Aspectos da Segurança

Definir uma política de segurança é também complicado porque cada organização deve decidir quais aspectos de proteção são os mais importantes e deve manter uma boa relação segurança x facilidade de uso. Por exemplo, uma empresa pode considerar:

Integridade de dados. A integridade se refere à proteção da troca de dados: Os dados que chegam para alguém são exatamente os mesmos dados que foram enviados?

Disponibilidade dos Dados. Disponibilidade se refere à proteção contra a interrupção de um serviço: Os dados continuam acessíveis para usuários legítimos?

Confiabilidade e Privacidade dos Dados. Confiabilidade e privacidade se referem à proteção contra observadores não autorizados ou coleta ilegal de dados: Os dados estão protegidos contra acesso não autorizado?

29.4 Responsabilidade e Controle

Muitas empresas descobrem que elas não podem desenvolver uma política de segurança porque ela não tem especificado como a responsabilidade pela informação está distribuída ou controlada. O assunto sobre responsabilidade tem muitos aspectos a serem considerados:

Contabilidade. A contabilidade se refere a como um registro de auditoria é mantido: qual grupo é responsável por cada item de dados? Como o grupo mantém registros de acesso e modificações?

Autorização. A autorização se refere à responsabilidade por cada item de informação e como ela é delegada para outros: quem é responsável por onde a informação reside e como uma pessoa responsável aprova acessos e mudanças?

O ponto crítico por debaixo da contabilidade e da autorização é o *controle* – uma organização deve controlar o acesso à informação analogamente ao modo como ela controla o acesso aos seus recursos físicos como escritórios, equipamentos e suprimentos.

29.5 Mecanismos de Integridade

O capítulo 6 discutiu técnicas usadas para assegurar a integridade dos dados contra danos acidentais: *paridade de bits*, *checksums* e *cyclic redundancy checks (CRCs)*. Para utilizar tais técnicas, quem envia calcula um valor inteiro e pequeno como uma função dos dados em um pacote. O receptor recalcula a função através dos dados que chegaram e compara o resultado com o valor que quem enviou calculou.

Um checksum ou CRC não podem garantir absolutamente a integridade dos dados por duas razões. Primeira, se o mau funcionamento do hardware mudar o valor de um checksum assim como o valor dos dados, é possível que o checksum alterado seja válido para os dados alterados. Segundo, se mudanças nos dados resultantes de um ataque planejado, o atacante pode criar um checksum válido para os dados alterados.

Muitos mecanismos têm sido utilizados para garantir a integridade de mensagens contra mudanças intencionais. Em geral, os métodos codificam os dados transmitidos com um *message authentication code (MAC)* que o atacante não possa quebrar ou forjar. Esquemas de codificação típicos usam mecanismos de **hashing** de criptografia. Por exemplo, um esquema de

hashing de criptografia usa uma *chave secreta* conhecida apenas por quem envia e por quem recebe. Quando quem envia codifica a mensagem, a função **hash** da criptografia utiliza a chave secreta para embaralhar a posição dos bytes dentro da mensagem assim como codificar os dados. Apenas o receptor pode desembaralhar os dados: um atacante, que não tem a chave secreta, não pode decodificar a mensagem sem introduzir erros. Por isso, o receptor sabe que qualquer mensagem que pode ser decodificada corretamente é autêntica.

29.6 Controle de Acesso e Senhas

Muitos sistemas operacionais utilizam o sistema de *senhas* para controlar o acesso aos recursos. Cada usuário tem uma senha, a qual é mantida em segredo. Quando o usuário precisa acessar recursos protegidos, ele tem que fornecer sua senha.

Um simples esquema de senha funciona bem para sistemas operacionais convencionais porque o sistema não revela a senha para outros. Em uma rede, entretanto, um mecanismo simples de senha está suscetível a roubo. Se um usuário em uma localização envia sua senha através da rede para um computador em outra localização, qualquer um que “grampeie” a rede pode obter uma cópia da senha. Observar mensagens em uma rede é especialmente fácil quando os pacotes viajam através de uma LAN porque muitas tecnologias de LAN permitem que uma estação anexada à rede capture uma cópia de todo o tráfego. Em tais situações, passos adicionais devem ser tomados para prevenir que a senha seja reusada.

29.7 Encriptação e Privacidade

Para assegurar que o conteúdo de uma mensagem permaneça confidencial mesmo com tentativas de interceptação, ela deve ser *encriptada*. Na essência, a encriptação embaralha os bits de uma mensagem de tal maneira que apenas o seu receptor consiga desembaralhá-la. Alguém que intercepte uma cópia da mensagem encriptada não será capaz de extrair suas informações.

Existem muitas tecnologias de encriptação. Em algumas tecnologias, quem envia e quem recebe devem ter uma cópia de uma *chave de encriptação*, a qual é mantida em segredo. Quem envia utiliza a chave para produzir a mensagem encriptada, a qual é então enviada através da rede. O receptor utiliza a chave para decodificar a mensagem encriptada. Isto é, a função *encriptar* tem dois argumentos: a chave, *K* e a mensagem a ser encriptada, *M*. A função produz uma versão encriptada da mensagem, *E*.

- $E = \text{encriptar}(K, M)$
- A função *decriptar* reverte o processo a fim de obter a mensagem original:
- $M = \text{decriptar}(K, E)$
- Matematicamente, a decriptar é o inverso de encriptar:
- $M = \text{decriptar}(K, \text{encriptar}(K, M))$

29.8 Encriptação por Chave Pública

Em muitos esquemas de encriptação, a chave deve se mantida em segredo para evitar comprometer a segurança. Uma técnica de encriptação interessante designa para cada usuário um par de chaves. Uma delas, chamada de *chave privada*, é mantida em segredo, enquanto que a outra que se chama *chave pública* é publicada junto com o nome do usuário, então todo mundo sabe o seu valor. A função de encriptação tem a propriedade matemática que a mensagem encriptada com a chave pública não pode ser facilmente decriptada exceto com a

chave privada, e a mensagem encriptada com a chave privada não pode ser decriptada exceto com a chave pública.

As relações entre a encriptação e a decriptação com as duas chaves podem ser expressas matematicamente. Denotemos a mensagem por M , a chave pública do usuário 1 por $pub-u1$ e a chave privada do usuário 1 por $prv-u1$. Então

$$M = \text{decriptar}(pub-u1, \text{encriptar}(prv-u1, M))$$

E

$$M = \text{decriptar}(prv-u1, \text{encriptar}(pub-u1, M))$$

Revelar a chave pública é seguro porque as funções usadas para encriptar e decriptar têm uma *propriedade de só uma via*. Isto é, dizer a qualquer um a chave pública não permite que a pessoa forje uma mensagem que pareça estar encriptada com a chave privada.

A encriptação por chave pública pode ser utilizada para garantir a confidencialidade. Quem envia a mensagem que deseja que ela permaneça em sigilo utiliza a chave pública do receptor para encriptar a mensagem. Obter uma cópia da mensagem quando ela passar pela rede não permite que alguém leia seu conteúdo porque a decriptação requer a chave privada do receptor. Por isso, esse esquema assegura que os dados permaneçam confidenciais porque apenas o receptor pode decriptar a mensagem.

29.9 Autenticação com Assinaturas Digitais

Um mecanismo de encriptação pode também ser utilizado para autenticar quem está enviando uma mensagem. Essa técnica é conhecida como uma *assinatura digital*. Para assinar uma mensagem, quem a envia encripta a mensagem usando uma chave conhecida apenas por essa pessoa. O receptor utiliza a função inversa para decriptar a mensagem. O receptor sabe quem enviou a mensagem porque apenas quem enviou a mensagem tem a chave necessária para realizar a encriptação. Para assegurar que as mensagens encriptadas não estão sendo copiadas e reenviadas mais tarde, a mensagem original pode conter um tempo e data que a mensagem foi criada.

Considere como um sistema de chave pública pode ser usado para prover uma assinatura digital. Para assinar uma mensagem, o usuário encripta a mensagem usando sua chave privada. Para verificar a sua assinatura, o receptor verifica a sua chave pública e a utiliza para decriptar a mensagem. Devido a só o usuário saber a chave privada, apenas o usuário pode encriptar uma mensagem que pode ser decodificada com sua chave pública.

Interessantemente, dois níveis de encriptação podem ser utilizados para garantir que a mensagem será autêntica e privada. Primeiro, a mensagem é assinada através da utilização da chave privada do usuário que a envia para encriptá-la. Segundo, a mensagem encriptada é encriptada novamente usando a chave pública do seu receptor. Matematicamente, a dupla encriptação pode ser expressa assim:

$$X = \text{encriptar}(pub-u2, \text{encriptar}(prv-u1, M))$$

Onde M é a mensagem original, X é a mensagem duplamente encriptada, $prv-u1$ é a chave privada do usuário que envia a mensagem e $pub-u2$ é a chave pública do seu receptor.

Quando ela é recebida, o processo de decriptação é reverso ao processo de encriptação. Primeiro, o receptor utiliza a sua chave privada para decriptar a mensagem. A decriptação remove um nível de encriptação, mas ainda deixa a mensagem digitalmente assinada. Segundo, o receptor usa a chave pública de quem enviou a mensagem para decriptá-la novamente. O processo pode ser expresso assim:

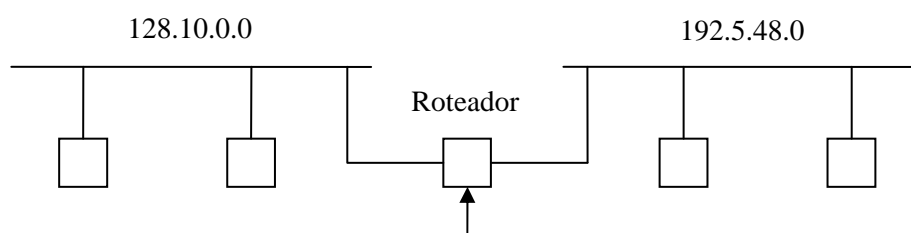
$$M = \text{descriptar}(\text{pub-}u1, \text{descriptar}(\text{prv-}u2, X))$$

Onde X é a mensagem que viajou pela rede, M é a mensagem original, $\text{prv-}u2$ é a chave privada do receptor e $\text{pub-}u1$ é a chave pública de quem a enviou.

Se uma mensagem com sentido resultar da dupla decifração, é verdade que a mensagem era confidencial e autêntica. A mensagem deve alcançar o seu receptor desejado porque somente ele tem a chave privada correta necessária para remover a primeira encriptação. A mensagem deve ter sido autenticada, porque apenas quem a enviou tem a chave privada necessária para encriptá-la para que a sua chave pública decifre-a corretamente.

29.10 Filtragem de Pacotes

Para prevenir que cada computador em uma rede acesse arbitrariamente computadores ou serviços, muitos sites utilizam a técnica conhecida como *filtragem de pacotes*. Como ilustra a figura abaixo, um filtro de pacotes é um programa que opera em um roteador. O filtro consiste em um software que pode prevenir que pacotes passem pelo roteador de uma rede para a outra. O administrador deve configurar o filtro de pacotes para especificar quais pacotes podem passar pelo roteador e quais deveriam ser bloqueados.



Filtragem de pacotes no roteador

O filtro de pacotes opera examinando campos no cabeçalho de cada pacote. O filtro pode ser configurado para especificar quais campos do cabeçalho serão examinados e como interpretar seus valores. Para controlar quais computadores em uma rede podem se comunicar com computadores em outra, um administrador especifica que o filtro deve examinar os campos *origem* e *destino* no cabeçalho de cada pacote. Na figura, para prevenir que o computador com endereço IP 192.5.48.27 na rede da direita se comunique com qualquer computador da rede da esquerda, o administrador deve especificar que o filtro bloqueie todos os pacotes com o endereço de origem igual a 192.5.48.27. Similarmente, para evitar que um computador com o endereço 128.10.0.32 na rede da esquerda receba qualquer pacote da rede da direita, o administrador especifica que o filtro bloqueie todos os pacotes com o endereço de destino igual a 128.10.0.32.

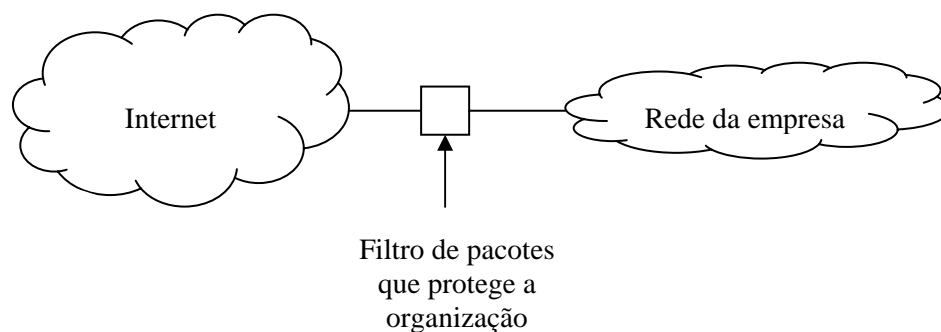
Em adição a utilizar os endereços de origem e destino, um filtro de pacotes pode examinar o protocolo em um pacote ou o serviço de nível mais alto ao qual o pacote corresponde. A habilidade de bloquear pacotes seletivamente para um serviço em particular significa que o administrador pode evitar tráfego para um só serviço, enquanto permite o tráfego para outros. Por exemplo, um administrador pode configurar um filtro de pacotes para bloquear todos os pacotes que carregam comunicação da World Wide Web, enquanto permite pacotes que carregam tráfego de e-mail.

Um mecanismo de filtro de pacotes permite que um administrador especifique combinações complexas de endereços de origem e destino, e serviços. Tipicamente, o software de filtro de pacotes permite que o administrador especifique combinações booleanas de origem, destino e tipo de serviço. Por isso, um administrador pode controlar o acesso a serviços específicos em computadores específicos. Por exemplo, o administrador pode escolher bloquear todo o tráfego destinado para o serviço FTP no computador 128.10.2.14, todo o tráfego da World Wide Web que deixa o computador 192.5.48.33 e todo o e-mail do computador

192.5.48.34. O filtro bloqueia apenas as combinações especificadas – o filtro deixa passar o tráfego destinado para outros computadores e tráfego para outros serviços nos computadores especificados.

29.11 Conceito de Firewall de Internet

Um filtro de pacotes é muito utilizado para proteger computadores de uma organização e redes de tráfego da Internet não desejado. A figura abaixo mostra o filtro colocado em um roteador que conecta a organização ao resto da Internet:



Um filtro de pacotes configurado para proteger uma organização contra tráfego do resto da Internet é chamado de um *firewall de Internet*; o termo vem do limite físico a prova de fogo colocado entre duas estruturas para evitar que o fogo se mova entre elas. Como um firewall convencional, um firewall de Internet é feito para evitar que os problemas na Internet se espalhem pelos computadores da organização.

Os firewalls são as ferramentas de segurança mais importantes usadas para lidar com conexões de rede entre duas organizações que não confiam uma na outra. Colocar um firewall em cada conexão de rede externa, uma organização pode estabelecer um *perímetro de segurança* que evita que pessoas de fora interfiram nos computadores da rede interna. Em particular, limitando o acesso a um conjunto pequeno de computadores, um firewall pode prevenir que pessoas de fora vejam todos os computadores em uma organização, inundando as redes da empresa com tráfego não desejado ou atacando um computador enviando uma sequência de datagramas IP que se sabe que causam o mau funcionamento do sistema computacional (por exemplo, derruba-o).

Um firewall pode reduzir o custo da segurança. Sem o firewall para prevenir acesso, pessoas de fora podem enviar pacotes para computadores arbitrários dentro da empresa. Consequentemente, para prover a segurança, uma organização deve tornar todos os computadores seguros. Com o firewall, entretanto, um administrador pode restringir a entrada de pacotes a um pequeno grupo de computadores. No caso mais extremo, o conjunto pode conter apenas um computador. Embora tal conjunto de computadores precise ser seguro, os outros computadores da empresa não precisam ser. Por isso, a organização pode economizar dinheiro porque é mais barato instalar um firewall do que tornar todos os sistemas operacionais seguros.