Redes de Computadores

Carlos Antônio Ruggiero

Janeiro de 1995 - Última atualização: fevereiro de 2016

Copyright (c) 2003 Carlos Antônio Ruggiero
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".

Sumário

Parte I

Teoria

Capítulo 1

Rede de Computadores - Histórico e Conceitos Gerais

Pode-se considerar que o interesse pelas redes de computadores surgiu na década de 60 nos Estados Unidos. Nessa época, o Departamento de Defesa americano resolveu financiar a pesquisa na área para viabilizar uma rede de computadores distribuída pelo território americano que conseguisse se manter operacional mesmo quando de um ataque soviético. O financiamento acabou resultando em 1969 numa conexão entre quatro pontos nos EUA: UCLA, Stanford Reasearch Institute (SRI), UC em Santa Barbara (UCSB) e Universidade de Utah em Salt Lake City. Nessa conexão, um dispositivo especial era usado para a ligação: o chamado IMP (Interface Message Processor) fabricado pela Honeywell. Essa rede ganhou o nome de ARPANET (ARPA, hoje DARPA, significa Defense Advanced Research Projects Agency do Departamento de Defesa americano). Para maiores detalhes ver gopher://gopher.isoc.org:70/00/internet/history/.

Na década de 70, muitas redes surgiram tanto academicamente quanto comercialmente. Exemplos importantes são a BITNET, financiada pela IBM mas fundamentalmente acadêmica, a DECNET, um padrão de redes desenvolvido pela Digital Equipament Corporation, a CSNet, para o pessoal de computação, a Telenet, um serviço público X.25 oferecido para a população em geral, etc. Ao mesmo tempo a ARPANET evoluía em número de conexões e em protocolos sendo que os protocolos TCP/IP, hoje utilizados, surgiriam no final da década de 70. A ARPANET se transformaria em Internet no início da década de 80 quando os antigos protocolos foram gradativamente substituídos pelo TCP/IP.

Também na década de 70 surgiram os protocolos ISO/OSI e o famoso modelo das sete camadas. Surgiu também o conjunto de protocolos X.25. Enquanto isso, na ARPANET o conceito de Internetworking se tornava cada vez mais claro e importante.

A explosão das redes ocorreu na década de 80 principalmente depois que a National Science Foundation (NSF) resolveu financiar uma grande espinha dorsal interconectando os EUA. Também por essa época, a Internet se torna internacional com conexões para a Europa e América do Sul.

O surgimento dos microcomputadores na década de 80 também estimulou muito a difusão das redes. Surgem vários protocolos específicos para microcomputadores tais como o Novell Netware, Lantastic, Banyan Vines entre outros.

Finalmente, na década de 90, com o grande apoio dado pelo vice-presidente americano Al Gore, a Internet se popularizou crescendo fantasticamente nos cinco primeiros anos da década e praticamente virando sinônimo da "Information Highway".

No Brasil, várias redes existiram antes da Internet. A primeira conexão à Internet no Brasil foi feita pela FAPESP em fevereiro de 1991. Também em fevereiro de 1991, o Centro de Computação Eletrônica (CCE) da USP se conectou à Internet através de um link de 4800 baud.

1.1 Conceitos Fundamentais

1.1.1 LAN x MAN x WAN

Um dos principais conceitos de redes de computadores diz respeito à extensão geográfica coberta pela rede. Redes restritas a um prédio ou a um campus com distâncias não superiores a 2 km são chamadas de redes locais ou, em inglês, "Local Area Network" (LAN). Redes que incluem computadores em cidades diferentes, em estados diferentes ou mesmo em diferentes países são chamadas de redes de longa distância, ou, em inglês, "Wide Area Network" (WAN). LANs são capazes de grande larguras de banda de transmissão com baixo custo enquanto as taxas de transmissão em WANs são consideravelmente mais baixas. Numa faixa intermediária, surgiu nos últimos anos o conceito de redes metropolitanas, as "Metropolitan Area Networks" (MANs) que cobrem uma área equivalente a uma cidade com desempenho comparável às LANs.

1.1.2 Rede Pública x Rede Privada

Quanto ao controle e gerenciamento das redes, estas podem ser públicas, onde qualquer pessoa, mediante pagamento, pode fazer parte da rede, ou privada que pertence a um grupo econômico privado. Exemplos de rede pública são a Internet mundial, a RENPAC no Brasil, etc. Exemplos de rede privada são as redes da IBM (VNET), da DEC (Easynet), etc.

1.1.3 NOS x SO + suporte de redes

Logo após ao surgimento do PC e devido a limitações do sistema DOS, começaram a surgir sistemas operacionais voltados especificamente a serviços de redes. Esses sistemas receberam o nome genérico de NOS do inglês "Network Operating Systems". Em máquinas maiores, a filosofia de redes foi incorporada ao sistema operacional "a posteriore". Exemplos de NOS são o Novell Netware, o Lantastic, Banyan Vines e de SO com suporte de redes, o Unix em suas várias versões (flavors), o VMS da digital etc.

1.1.4 Rede de Computadores x Rede de Teleprocessamento

Devido ao fato de a expressão "rede de computadores" ser usada em muitos contextos diferentes seu real significado pode ser facilmente confundido. No caso brasileiro é comum referir-se a um sistema com um computador e muitos terminais conectados (principalmente remotamente) como uma "rede de computadores". Apesar de a arquitetura descrita ser útil e barata em certa

situações, ela claramente não é uma rede de computador*es* já que somente *um* computador está envolvido. Nesse caso adotamos nesse curso a expressão rede de teleprocessamento para designar tal arquitetura.

1.1.5 Com conexão (Connection oriented) x Sem conexão (Connectionless)

Existem duas formas fundamentais de se trocar dados entre duas entidades:

- Com conexão: nessa forma, estabelece-se uma conexão entre as duas partes interessadas, ou seja, estabelece-se um canal entre elas de tal forma que os dados a serem transmitidos passem por esse canal. O exemplo clássico aqui é o sistema telefônico.
- Sem conexão: aqui, os dados são transmitidos através de rotas sem que a parte que envia saiba exatamente se, como e quando a outra parte receberá o que foi transmitido. A analogia aqui é com o serviço postal (correios).

A diferenciação entre os dois métodos de troca de informações é de fundamental importância no entendimento de protocolos de comunicação de dados.

1.2 O Modelo de Camadas

O modelo de camadas surgiu com as sugestões ISO/OSI da década de 70. A idéia é simplificar o projeto de redes construindo-se um conjunto de protocolos onde cada um se apóia no protoloco de nível mais baixo. Uma grande vantagem do sistema de camadas é que é possível trocar-se a implementação de uma das camadas sem que as outras sejam modificadas.

O modelo ISO/OSI apresenta sete camadas (figura 1.1), quais sejam:

1. Nível 1: nível físico (Physical Layer). Nesse nível é que ocorre realmente a comunicação de dados entre os computadores. A função dessa camada é transmitir bits de um computador a outro. Fatores importantes na definição de protocolos dessa camada são quantos volts devem ser utilizados na definição dos níveis lógicos, quantos níveis lógicos devem ser possíveis em cada instante de transmissão, quantos "fios" devem ser usados para a comunicação, etc. Exemplos de meios físicos são: cabo coaxial, par trançado, fibras óticas, comunicação via satélite, etc.
2. Nível 2: a camada de dados (Data Link Layer). Aqui, a obrigação é empacotar um conjunto de dados e transmiti-los sem erros ao outro computador. Esse conjunto de dados é normalmente chamado de "frame" e tem tamanho típico de algumas centenas de bytes. Em alguns casos, mecanismos elaborados para garantir a correção de erro são adotados nessa camada. É comum também implementar controle de fluxo nessa camada. Exemplos de protocolos dessa camada (pelo menos, aproximadamente!) são o Ethernet, o Token Ring, etc.
3. Nível 3: a camada de rede (Network Layer). A função fundamental dessa camada é *rotear* os frames ou pacotes corretamente entre os vários computadores de tal forma a estabelecer uma conexão eficaz entre o computador origem e o destino, mesmo passando por uma

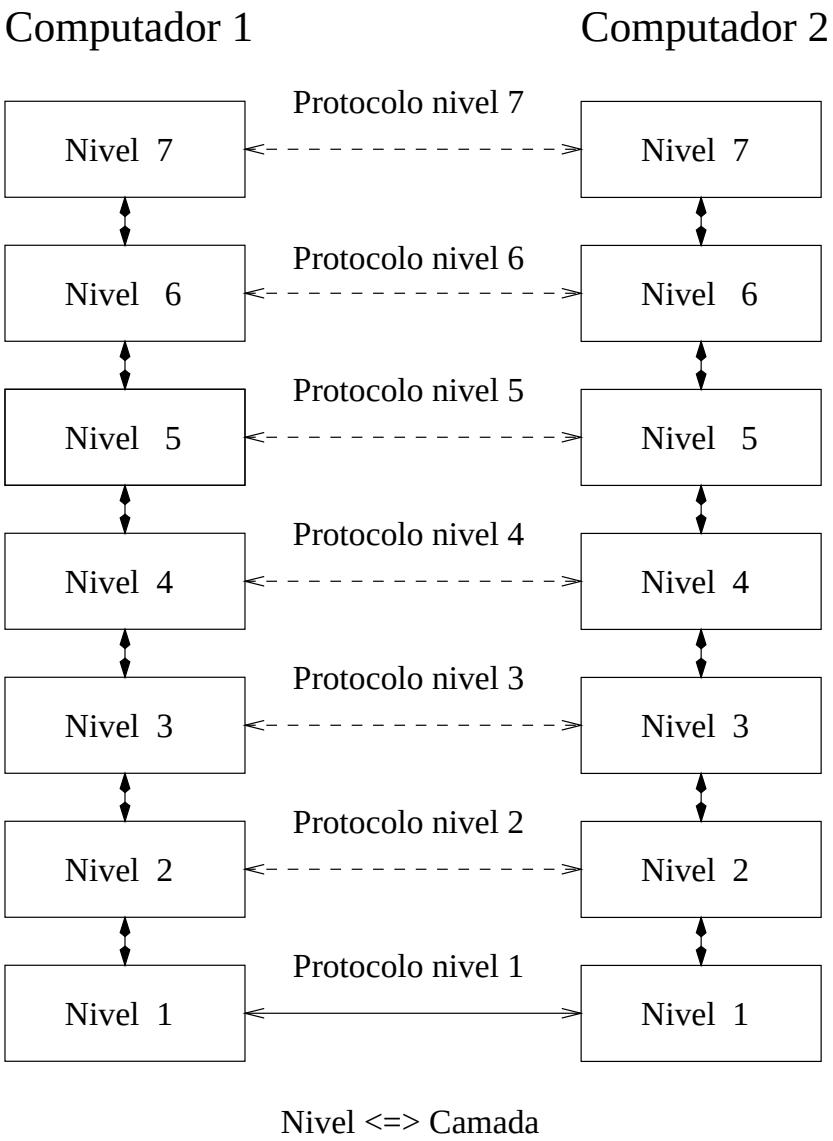


Figura 1.1: O modelo de camadas ISO/OSI

grande série de computadores intermediários. Pode ser importante determinar o caminho melhor (mais barato ou mais curto) entre dois computadores e essa é também uma tarefa para a camada de rede. Exemplos importantes de protocolos dessa camada são o IP (Internet Protocol) usado na Internet e o X.25, usado na RENPAC e em outras redes públicas de pacote ao redor do mundo. Por sua importância, a camada de rede receberá uma atenção especial nesse curso.

4. Nível 4: a camada de transporte (Transport Layer). As camadas anteriores (e de nível mais baixo) se preocupam em transmitir dados entre computadores. Não se deve esquecer porém, que, em sistemas multiusuários, várias pessoas (ou, mais precisamente, processos) podem utilizar os serviços de rede simultaneamente. É importante que a conexão de um processo não interfira nas outras conexões. Essa é uma das funções da camada de transporte. Por esse motivo, a camada de transporte é a primeira da camadas conhecidas como fim-a-fim ou "end-to-end". Outra função dessa camada é quebrar uma longa cadeia de dados a ser transmitida em várias subunidades menores que possam ser tratadas pelos níveis mais baixos e garantir que essas subunidades cheguem ao destino corretamente e na ordem esperada.

5. Nível 5: camada de sessão (Session Layer) e nível 6, camada de apresentação (Presentation Layer). Essas duas camadas estão no modelo ISO/OSI mas não são muito divulgadas e implementadas na maioria de redes existentes. Uma função da camada de sessão é a sincronização. Considere o caso da queda de conexão entre dois computadores que trocavam dados entre si. Os computadores deveriam ser capazes de, quando do reestabelecimento da conexão, continuarem do ponto onde estavam na troca de informações. Como em geral esta camada não é implementada, quando ocorre o descrito, os computadores devem recomeçar a troca de dados desde o início. A camada de apresentação tem como objetivo a conversão de dados a serem trocados para uma forma que ambos os computadores entendam. Por exemplo, um dos computadores pode usar o ASCII para a representação de caracteres enquanto outro usa o EBCDIC. Outras funções importantes da camada de apresentação são a compressão de dados e a criptografia.

6. Nível 7: camada de aplicação (Application Layer). É a última camada da "pilha" ISO/OSI. Essa camada é onde são implementados os protocolos dos aplicativos que farão uso da rede. Exemplos de aplicativos incluem a transferência virtual de terminal, correio eletrônico (*e-mail*), transferência de arquivos, etc. Exemplos de protocolos dessa camada são o Telnet, FTP, SMTP, VTAM, FTAM, etc.

É importante entender-se o correto funcionamento do modelo de camadas. Logicamente, a camada n de um computador fala diretamente com a camada n do outro. Fisicamente porém, ela passa seus dados para a camada n-1 no próprio computador, que por sua vez passa para a camada n-2 e assim por diante até a informação chegar ao nível físico. A camada física de um computador passa então a informação para a camada física do outro computador e a informação aí viaja em sentido inverso "subindo" as camadas até chegar à camada n onde os dados são interpretados. No computador origem, cada camada coloca um cabeçalho seu no pacote que é recebido "de cima" e passa o novo pacote, agora maior, para a camada inferior. Em alguns casos, a camada anexa uma informação extra ao final do pacote. No computador destino, cada camada retira o seu cabeçalho e passa o pacote (menor) para as camadas superiores.

1.3 Internetworking

Conceitualmente, o modelo de camadas pode ser considerado a maior contribuição do ISO/OSI. Da mesma forma, pode-se dizer que a maior contribuição da ARPANET, Internet e dos protocolos TCP/IP foi o conceito de Internetworking.

A idéia é conectar as inúmeras redes de computadores ao redor do mundo de tal forma que elas se comportam como uma grande rede de computadores. É importante notar que as várias redes a serem interconectadas possuem tecnologias bastante díspares; de fato, umas são projetadas para serem muito rápidas, outras para serem bastante confiáveis, outras para cobrirem uma grande área geográfica, etc. Os protocolos da ARPANET foram projetados desde o ínicio com isso em mente e a existência da Internet no mundo atual é prova definitiva do sucesso da estratégia.

O conceito do Internetworking lembra a propriedade da *transitividade* na matemática: assim, se o computador A se comunica com o computador B, e o computador B se comunica com o computador C, então é possível fazer A se comunicar com C.

1.4 Padrões

Os padrões em redes de computadores podem surgir de duas formas diferentes:

- **Organizações específicas para padronizações:** Nesse caso incluem-se os protocolos sugeridos pela ISO, pela Internet Architecture Board (IAB), o IEEE, etc.
- **Indústrias importantes na área:** Algumas empresas, com grande influência no mercado, podem criar protocolos que acabam sendo aceitos pela comunidade como um todo. Exemplos são a Ethernet, sugerida pela Xerox, Intel e Digital, o NFS sugerido pela Sun Microsystems, o Netware da Novell, etc.

A International Standards Organization (ISO) é composta por organizações encarregadas de padronização dos 89 países membros. Exemplos são a ANSI, DIN, etc.

A Internet Architecture Board é composta por uma mesa diretora (board) e duas grandes áreas: a Internet Research Task Force (IRTF) e a IETF (Internet Engineering Task Force).

1.5 O Meio Físico

A primeira camada do modelo ISO/OSI trata do meio físico que realmente transmite os bits de um computador a outro. Os meios físicos mais usados em redes de computadores são:

1. **Fios, ou Par Trançado:** nas suas várias formas, é talvez o meio físico mais comum. Muitas vezes chamado pelo nome genérico de "cobre", apresenta baixa largura de banda mas também baixo custo e, devido a enorme e abrangente rede telefônica, é encontrado praticamente no mundo todo.
2. **Cabo Coaxial:** Dois tipos de cabos são usados em redes; um é o cabo coaxial de 50 ohms usado no Ethernet (conhecido como "baseband") e o outro é o cabo coaxial usado em TV a cabo, de 75 ohms (conhecido como "broadband"). Apresenta largura de banda bem melhor do que o par trançado mas tem custo mais alto.
3. **Fibra Ótica:** é aparentemente o meio de transmissão do futuro. Apresenta enorme largura de banda com custo menor do que o "cobre". Nos EUA acredita-se que por volta de 2050,

toda a malha telefônica esteja substituída por fibras óticas. Porém, a manipulação e instalação de fibras óticas devem ser feitas com cuidado, pois as fibras óticas rompem-se facilmente.

4. **Comunicação via Satélite:** a principal vantagem aqui é a rapidez de instalação e a alta confiabilidade do sistema. A largura de banda é boa mas a latência é bem alta, podendo representar um sério problema em aplicações interativas.

5. **Enlaces de Microondas:** são bastante utilizados em telefonia para prover o serviço DDD. Também são muito usados em circuitos repetidores de televisão. O problema é que exigem visada direta o que limita a distância entre as duas antenas a menos de 50 km. Outros tipos de ligações com visada direta estão sendo pesquisados como a ligação ótica. Devem ser incluídos neste ítem, a comunicação via celulares e dispositivos sem fio ("Wireless")

6. **Meio Magnético:** Segundo Tanenbaum, não se pode desprezar a largura de banda de um caminhão trafegando cheio de fitas magnéticas. Apesar de apresentar uma latência muito grande (o tempo do caminhão levar os dados de um local para o outro!) a largura de banda é extremamente elevada. De fato, imagine-se um caminhão transportando DVDs, com capacidade de 25 GB cada, de São Paulo ao Rio de Janeiro: o caminhão tem capacidade para levar dezenas de milhares de mídias o que corresponde a dezenas de terabytes de informação sendo transferidas. Nenhum meio físico (a não ser a fibra ótica) apresenta uma largura de banda tão grande.

1.6 Recursos na Internet

Para saber mais, acesse os seguintes *sites*:

- *http://www.zakon.org/robert/internet/timeline* traz os eventos mais importantes relacionados à Internet em ordem cronológica.
- Também referente à história da Internet, veja *http://www.ocean.ic.net/ftp/doc/nethist.html*.
- Para entender a distinção entre LAN, MAN e WAN, veja a discussão de 1994 sobre "How to distinguish LAN,WAN & MAN" no grupo comp.dcom.lans.ethernet em *http://groups.google.com*.
- Veja a discussão "OSI 7 layer model" no grupo bit.listserv.novell
- O site da ISO é *http://www.iso.org*. No site do IETF (Internet Engineering Task Force), *http://www.ietf.org*, é possível encontrar a relação de RFCs, bem como fazer *download* deles.
- Para uma comparação entre fibra e *cobre*, veja a discussão "Re: SUBJECT: Fiber Versus Copper" no grupo comp.os.ms-windows.nt.admin.misc.
- Outra discussão interessante, de 2001, pode ser vista em "submarine fiber vs. satelite vs. microwave" no grupo alt.dcom.telecom.

Capítulo 2

Ethernet, FDDI - A camada 2

2.1 Ethernet

O protocolo Ethernet, hoje um padrão IEEE sob número 802.3 é talvez o MAC ("Medium Access Sublayer") mais usado no mundo. Nas seções a seguir, estudaremos as características básicas desse protocolo tais como o meio físico utilizado, o "frame", uso de repetidores e bridges, etc.

2.1.1 Histórico

O MAC Ethernet surgiu na década de 70 em trabalhos efetuados na Xerox (PARC). Posteriormente, a Xerox, a Intel e a Digital (DEC) concordaram em promover a tecnologia e criaram um padrão em 1978. Esse padrão seria adotado pelo IEEE sob o número 802.3 com pequenas modificações.

2.1.2 O Meio Físico

Vários meios físicos são utilizados para levar os frames Ethernet. O principal, pelo menos sob o aspecto do padrão é o assim chamado cabo coaxial grosso. Esse cabo é composto por um fio central envolvido por uma camada de polietileno que, por sua vez, está envolvida por uma proteção metálica coberta por um material isolante externo. O comprimento total do cabo não pode ultrapassar 500m. A impedância do cabo é 50 ohms. A figura 2.1 mostra o cabo coaxial.

A conexão com o computador é feita através de um "transceiver" ou "tap" que se conecta com o cabo e fica a ele preso. O cabo que liga o computador ao transceiver é chamado de "AUI cable" onde AUI significa "Attachment Unit Interface". A saída da placa de interface do computador é chamada de saída AUI. A arquitetura de uma rede Ethernet com cabo coaxial grosso pode ser vista na figura 2.2.

Apesar do cabo coaxial grosso ser o primeiro padrão para Ethernet, ele não é o mais comum. O problema é que ele tende a ser caro (o transceiver também não é barato), difícil de instalar por não dobrar facilmente e problemático quando se tem que mudar a topologia da

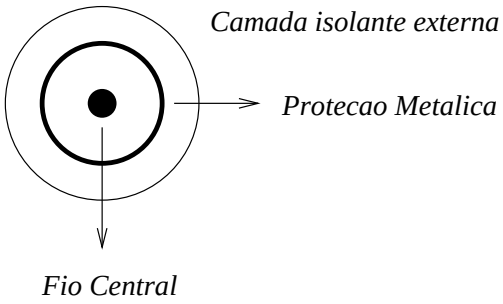


Figura 2.1: O cabo coaxial usado para Ethernet

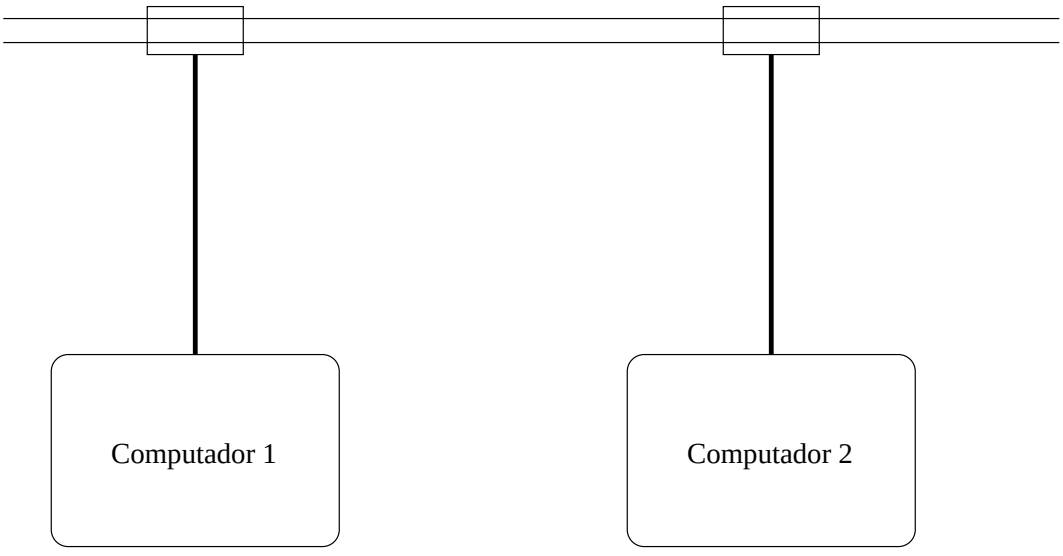


Figura 2.2: Arquitetura da Ethernet, cabo grosso

rede. Criou-se então a "cheapernet" que é a Ethernet em cabo coaxial mais fino (conhecido, em inglês, como *thinwire*) e flexível. Nesse caso, o custo é muito menor já que o cabo é mais barato, não é necessário o transceiver e é fácil modificar a topologia da rede quando necessário. Os dois principais problemas da "cheapernet" são: menor tolerância a ruído eletromagnético e menores distâncias cobertas. De fato, o cabo coaxial mais fino não pode ser colocado nas proximidades de máquinas elétricas potentes e o comprimento máximo da rede, sem repetição, é de 185 metros. A Ethernet com cabo grosso é chamada de *10base5* e a outra, *10base2*. A figura 2.3 mostra a topologia de uma conexão "cheapernet".

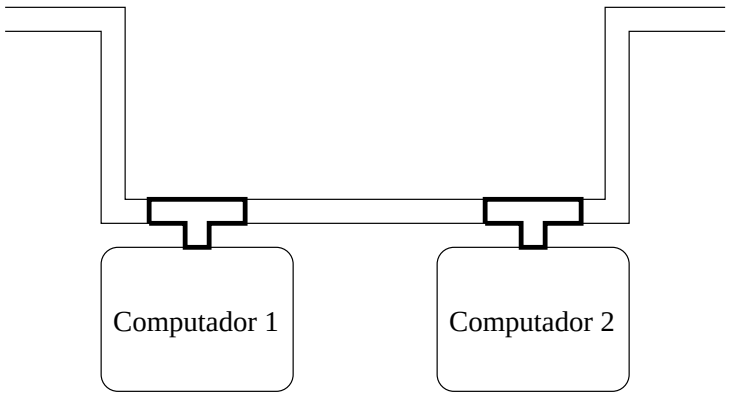


Figura 2.3: Arquitetura da Ethernet, cabo fino

Note que o cabo agora, sendo mais flexível, vai até a interface do computador e se conecta a ela através de um "T" que é um conector com três saídas do tipo BNC. Em ambas as extremidades da rede é necessário um terminador de 50Ω.

Além das duas formas de Ethernet já discutidas, foram criadas outras duas: uma baseada em fibra ótica, conhecida por *10BaseF*, e uma com par trançado (do inglês, "twisted pair") com o nome de *10BaseT*. A grande vantagem do 10BaseT é que os custos de cabeação são ainda mais baratos sendo que, em alguns casos, a própria cabeação telefônica pode ser usada. Também, o 10BaseT isola cada máquina das outras de forma que uma desconexão não prejudica o restante da rede. O 10BaseT exige porém, a presença de um "hub" para a conexão dos computadores à rede. A arquitetura da rede pode ser vista na figura 2.4.

2.1.3 Funcionamento da Ethernet

Uma rede Ethernet é composta por um meio físico (como já explicado anteriormente) que carrega os frames a uma velocidade de até 10 Mbps (dez megabits por segundo) para todos os computadores da rede (barramento). De fato, todos os computadores estão "ouvindo" todos os pacotes que transitam apesar de que, normalmente, cada computador só captura os pacotes que lhe são especificamente destinados.

Redes Ethernet pertencem à classe das redes ditas sensoras de portadora ou, do inglês, "carrier sense networks". Nessas redes, cada computador observa se o meio físico está sendo utilizado e só tenta transmitir em caso negativo. A sigla utilizada para designar essas redes é CSMA ("Carrier Sense Multiple Access"). Dentre as redes CSMA, distinguem-se duas variedades: as persistentes e as não persistentes. Redes persistentes são aquelas onde, se um computador pretende transmitir algo, ele sente o canal; se este estiver ocupado ele fica aguardando e verificando o canal *continuamente* e **tão logo** o canal desocupe, ele tenta transmitir. Redes onde, na mesma situação anterior, o computador espera um tempo aleatório depois de verificar que o

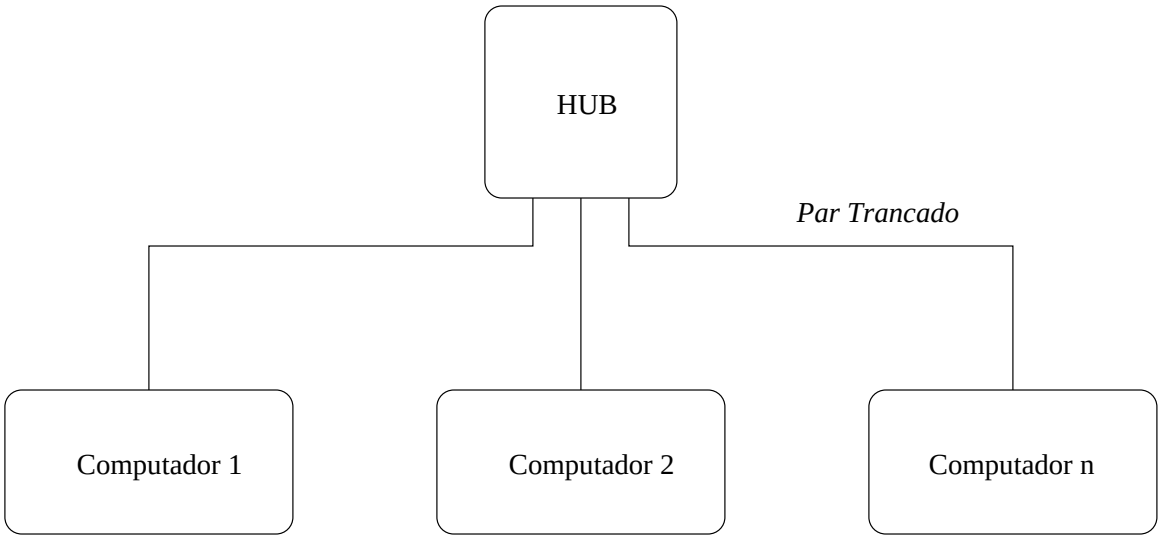


Figura 2.4: Arquitetura da Ethernet com par trançado

canal está ocupado antes de verificar novamente são chamadas de não persistentes. O Ethernet é um protocolo **persistente**.

Além de ser CSMA persistente, o Ethernet tem a possibilidade de detetar uma colisão. Colisões ocorrem no barramento Ethernet porque o sinal elétrico demora para percorrer o barramento e, assim, duas ou mais estações podem começar a transmitir pensando que o barramento está livre. Quando é detetada uma colisão, as estações transmissoras param imediatamente com a transmissão e esperam um tempo aleatório para tentar novamente. Se ocorrer novamente uma colisão, é esperado o dobro do tempo inicial e assim por diante. O Ethernet é então CSMA/CD ("Carrier Sense Multiple Access / Collision Detect).

2.1.4 Endereçamento

No Ethernet, cada **interface** tem um endereço único e invariável. Esse endereço é atribuído à interface pelo fabricante que compra conjuntos de endereços do IEEE. O endereço é formado por 48 bits, normalmente representados por uma sequência de seis bytes representados em hexadecimal e separados por dois pontos. Por exemplo, uma interface Ethernet tem o endereço:

$$00 : 80 : 48 : 84 : 8C : 82$$

2.1.5 Frame (pacote) Ethernet

O pacote Ethernet tem um tamanho mínimo de 64 bytes e, máximo de 1518 bytes. O pacote identifica os endereços da estação transmissora e da receptora, o tipo de pacote transmitido, o comprimento em bytes do campo de dados, um campo para checagem de integridade de dados (CRC) e um preâmbulo para sincronização.

A figura 2.5 mostra o formato do pacote IEEE 802.3. O formato do Ethernet original é ligeiramente diferente com o comprimento de dados substituído pelo tipo do pacote (no IEEE

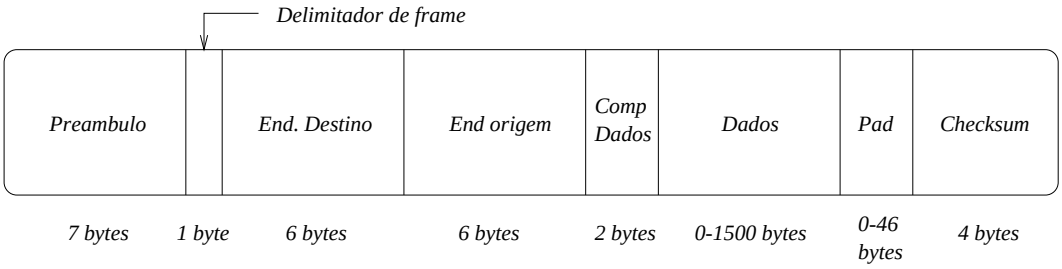


Figura 2.5: Pacote Ethernet (IEEE 802.3).

802.3, o tipo está dentro do campo de dados). O preâmbulo é composto por 7 bytes com 10101010 que produzem uma onda quadrada de 10 MHz para a sincronização do receptor. Se o campo de endereço destino contém somente uns, o pacote é de "broadcast" e todas as estações na rede recebem o pacote. O primeiro bit do campo destino ser 1 define uma transmissão para um grupo de estações o que é conhecido como "multicast".

O campo de Checksum é utilizado para se saber se o envio dos dados foi efetuado sem erro.

O tipo do pacote (dois bytes) é muito importante pois permite que redes diferentes (por exemplo, Novell e IP) trafeguem pacotes simultaneamente no barramento. Na verdade, mesmo dentro do TCP/IP, vários tipos de pacotes existem: por exemplo, o pacote IP tem tipo 0x0800, o pedido ou resposta ARP tem tipo 0x0806 e o pedido ou resposta RARP tem tipo 0x8035

2.1.6 Uso de Repetidores ("repeaters") e Pontes ("bridges")

Para se ampliar o alcance de redes Ethernet, pode-se usar de repetidores ou bridges. Repetidores simplesmente reforçam o sinal e são dispositivos aplicados no nível *físico* da rede. Não é possível mais do que 4 repetidores entre dois quaisquer transceivers resultando em um comprimento máximo de 2.5 km.

Ao contrário de repetidores, bridges agem na camada de dados da rede o que vale dizer que elas interpretam os pacotes Ethernet e só passam para o outro segmento pacotes válidos, isolando colisões, ruídos, etc. Um tipo especial de bridge (atualmente o tipo mais comum) é a "learning bridge" que aprende os endereços das interfaces que estão conectadas a cada um dos barramentos e só passa os pacotes de um segmento para outro se realmente houver a necessidade disso. Assim, as bridges acabam aumentando a largura de banda efetiva de transmissão por isolar os vários segmentos a ela conectados. É importante notar porém, que do ponto de vista dos protocolos de nível superior (por exemplo, TCP/IP), uma Ethernet com repetidores e bridges é equivalente a uma sem esses componentes já que eles são totalmente transparentes para os níveis superiores. Na verdade, isso acaba sendo mais uma vantagem desses dispositivos já que eles não precisam ser "configurados" como é o caso dos roteadores que serão vistos posteriormente.

Pontes com mais de 2 portas são chamadas de chaves ou **switches**.

2.2 FDDI

FDDI é a sigla utilizada para definir uma rede de alta velocidade em fibra ótica que se utiliza de uma topologia em anel. FDDI vem de *Fiber Distributed Data Interconnect*.

2.2.1 O meio Físico

O FDDI trabalha numa velocidade de 100 Mbps sobre fibra ótica. Fibras óticas são mais tolerantes a ruído que cabos de cobre e permitem velocidades de comunicação bem maiores. Atualmente, o custo de fibras já é menor do que o de cobre o que leva à previsão de que num futuro próximo o meio fibra será o mais comum para redes de computadores.

A topologia do FDDI é em anel com redundância de tal forma que a rede continua em funcionamento em certos casos de falha de computadores ou mesmo da conexão da fibra. O anel FDDI pode ter até 100 km de fibra e até 1000 estações. A topologia do FDDI com seus dois anéis pode ser vista na figura 2.6.

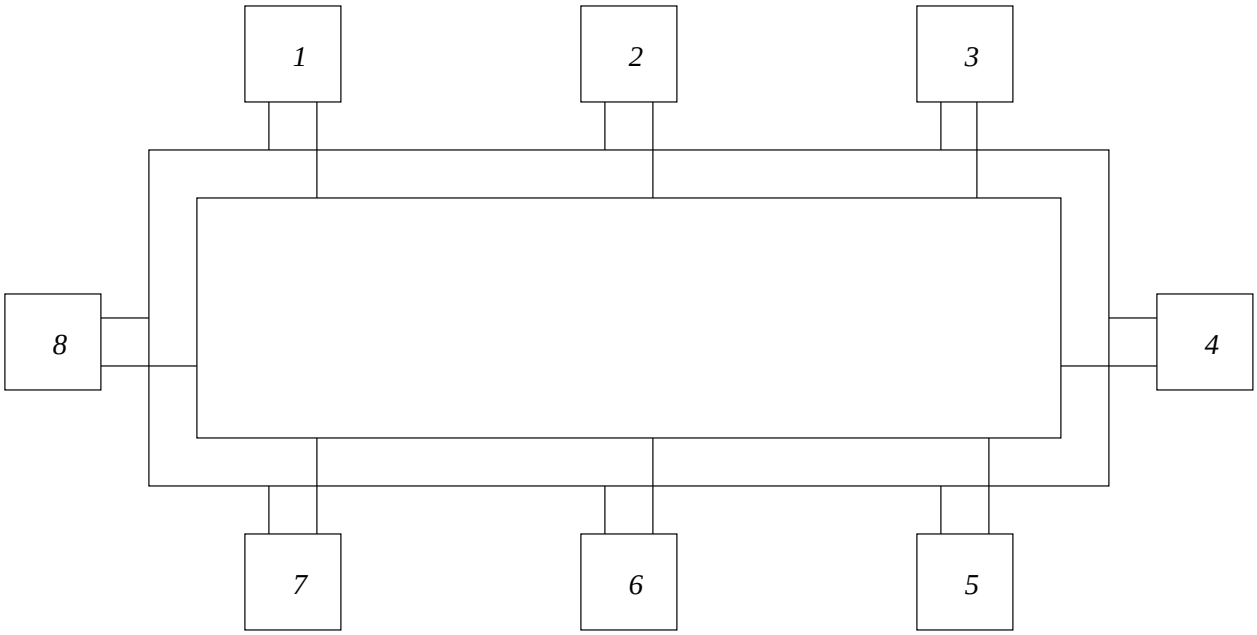


Figura 2.6: FDDI com 8 computadores conectados

2.2.2 Funcionamento

No FDDI, existe uma ficha ("token") que constantemente circula o anel. Quando uma estação quer transmitir, ela "captura a ficha" e faz a sua transmissão. Toda estação sempre pega o pacote e verifica se é para ela. Se o pacote não for seu, ela passa o pacote para a próxima estação; em caso contrário, ela mantém uma cópia do pacote e o retransmite. Depois da transmissão de um pacote, a estação transmissora passa a ficha circulante e assim concede a outras estações o direito de transmitir. Note que, enquanto tem a ficha, a estação não passa para frente os pacotes que estejam vindo pelo anel.

Quando um dos computadores falha, os outros percebem e, automaticamente, desconectam o computador quebrado da rede. Na figura 2.7 vemos a mesma topologia FDDI já apresentada depois de uma falha no computador 4.

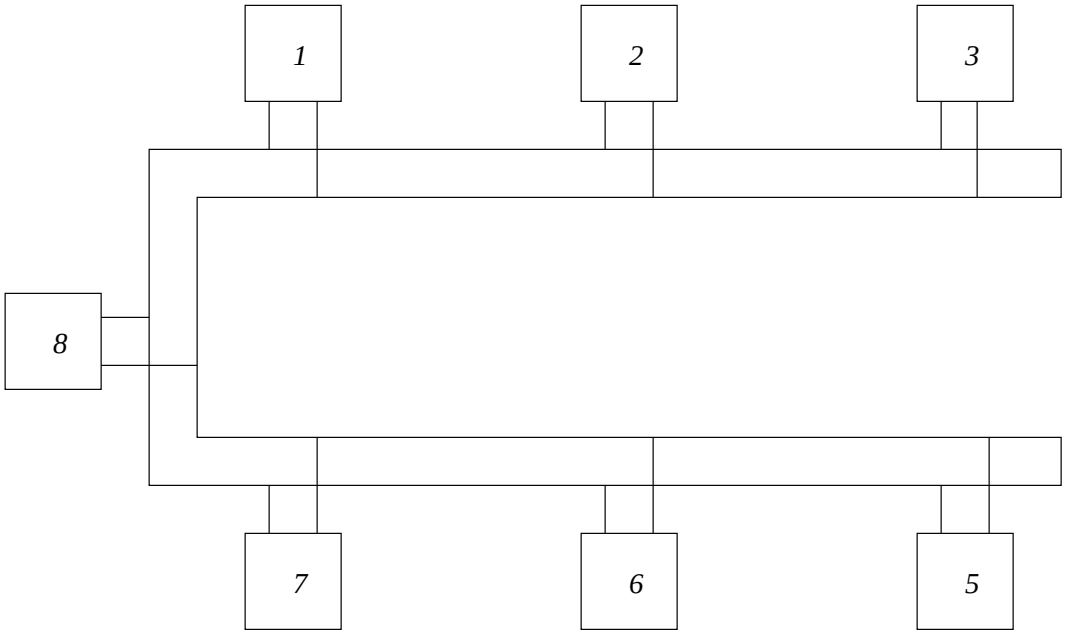


Figura 2.7: FDDI com falha em um computador

2.2.3 Frame do FDDI

O FDDI apresenta um pacote consideravelmente maior que o da Ethernet. Ele pode chegar a até 4500 bytes. O tamanho dos campos de um pacote FDDI é medido em "símbolos" de 4 bits cada um. Assim como o pacote Ethernet, ele leva o endereço destino e origem da trasmissão. O endereço FDDI pode ser de 4 ou 12 símbolos. Uma diferença importante entre o pacote FDDI e o Ethernet é que o primeiro contém informação de roteamento (até 60 símbolos) que facilitam a interconexão de mais de um anel FDDI. O campo de roteamento permite que uma estação especifique que um pacote deve primeiro ir para um determinado ponto para depois seguir para uma estação destino num outro anel.

Capítulo 3

Os Protocolos TCP/IP e Endereçamento IP

3.1 Introdução

O conjunto de protocolos da Internet ficou conhecido pelos seus dois componentes mais famosos: o *Internet Protocol*, ou IP e o *Transmission Control Protocol* ou TCP. Além deles, vários outros existem na mesma família: o *Network File System* (NFS), o *User Datagram Protocol* (UDP), o *File Transfer Protocol* (FTP), o *Simple Mail Transfer Protocol* (SMTP), o *Simple Network Management Protocol* (SNMP), o *Internet Control Message Protocol* (ICMP), o *Internet Group Management Protocol* (IGMP), o *Bootstrap Protocol* (BOOTP), o *Address Resolution Protocol* (ARP), o *Dynamic Host Configuration Protocol* (DHCP) e o telnet, originalmente *Virtual Terminal Protocol* (vtp).

A figura 3.1 mostra os protocolos TCP/IP segundo o paradigma das camadas ISO/OSI.

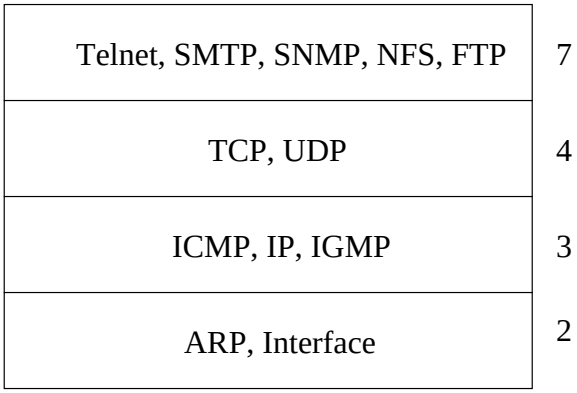


Figura 3.1: Os protocolos TCP/IP

3.2 O TCP/IP e internetworking

A filosofia de projeto mais importante para o TCP/IP é a do internetworking. O fundamental da filosofia "internetworking" é que exista uma rede lógica universal cobrindo todo o planeta. É fácil ver que um protocolo de nível 1 ou 2 do ISO/OSI não pode satisfazer esse requisito pois redes físicas locais e de longa distância são radicalmente diferentes (redes locais são rápidas e cobrem um pequeno espaço enquanto WANs são lentas mas cobrem grandes áreas). Além disso, teríamos um problema muito sério quando fosse necessário mudar a tecnologia da rede: o mundo todo deveria fazer a mudança simultaneamente!

Outra possibilidade seria adaptar os programas aplicativos para cada rede existente. Nesse caso, as diferenças entre redes seriam escondidas pelo aplicativo e teríamos uma rede universal. Essa solução porém, é problemática pois cada vez que se modificar o hardware da rede, dever-se-á modificar o aplicativo. Também, **todos** os aplicativos deverão ser programados para todas as possibilidades de redes físicas.

A melhor solução é então prover uma camada especificamente para essa interconexão entre redes. No TCP/IP, o protocolo responsável por isso é o IP.

Nesse esquema, a idéia de **roteamento** é fundamental na conexão entre duas redes.

3.3 Roteamento

Quando uma máquina deseja enviar dados para outra, o software (SO) primeiro verifica se o computador destino está na rede local; em caso positivo, os dados são transmitidos através da camada de dados; caso contrário, o pacote é enviado para uma máquina intermediária que esteja na rede local do computador origem, mas que saiba para onde mandar os dados de tal forma que esses cheguem ao destino (ou, pelo menos, se aproximem dele). Esse computador intermediário é chamado, segundo a nomenclatura original da Internet, de **gateway**. Esse termo porém, não é usado mais nesse contexto: o termo preferido hoje é **roteador** ou **router** no original inglês.

Note que o roteador deve ter, no mínimo, duas conexões diferentes de rede. A figura 3.2 mostra um esquema simples de roteamento.

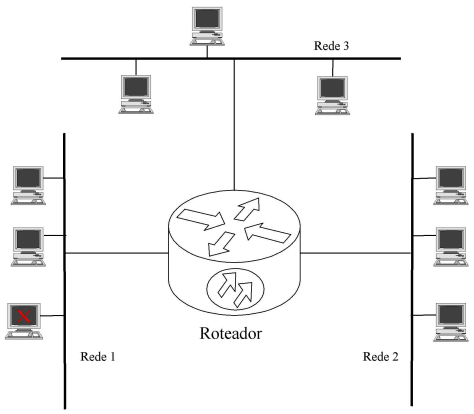


Figura 3.2: Um roteador ligando 3 redes

No exemplo da figura, o roteamento é simples já que existe apenas um roteador. Suponha porém, que uma das redes (por exemplo a rede 2) esteja conectada a uma outra rede (quatro) através de um segundo roteador. Nesse caso, pacotes da rede 1 que vão para a rede 4 tem que passar por ambos os roteadores. O roteador 1 deve saber que, para chegar a um computador da rede 4, ele precisa enviar os pacotes para o segundo roteador. Quando a internet vai crescendo, o conhecimento dos roteadores deve aumentar e seus algorítmos de roteamento se tornam cada vez mais complexos. Para facilitar o roteamento, os roteadores IP trabalham com **redes** ao invés de com computadores. A figura 3.3 ilustra o que foi explicado.

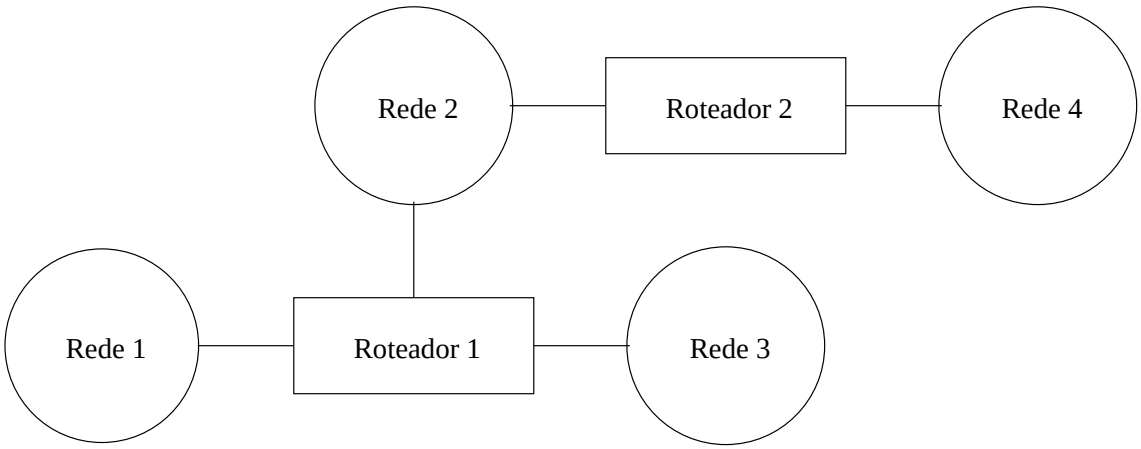


Figura 3.3: Quatro redes com dois roteadores

3.4 Endereçamento IP

Considerando que o IP é o protocolo responsável pela comunicação entre dois quaisquer computadores numa internet, seu conceito de **endereços** se torna fundamental. Apesar de podermos designar uma máquina por um nome numa internet, escolheu-se especificar um computador por um número binário de 32 bits. O problema de como transformar um nome no número correspondente será estudado posteriormente.

O endereço IP, quando usado na Internet (note o I maiúsculo), deve ser único no mundo todo. Os 32 bits do endereço são, para efeito de representação, divididos em quatro bytes. Cada byte é representado em decimal (um número entre 0 e 255) e separado do outro byte por um ponto. Assim, a ultra da Física, na Internet, tem o endereço:

10001111011010111110010000000001

que é representado por:

143.107.228.1

É normal dividir-se o endereço IP em dois: os bits do lado esquerdo são associados à **rede** enquanto os bits do lado direito representam o *host.*

3.4.1 Classes de Endereços IP

Para otimizar o roteamento, os endereços IP foram projetados para incluir tanto a rede onde está o computador como o próprio computador dentro da rede.

Para cobrir os vários tamanhos de redes, foram definidas três classes básicas: as classes A, B e C. Na classe A, o primeiro bit é 0, os sete bits seguintes designam a rede (127 possibilidades) e os restantes 24 bits designam o computador dentro da rede. Na classe B, os dois primeiros bits são 1 e 0 seguidos de 14 bits de rede e 16 bits de computador (16k redes de 64k computadores cada). Na classe C, os três primeiros bits são 110 seguidos de 21 bits de rede e 8 bits para especificar o computador. Endereços que começam com 1110 são chamados de classe D e são reservados para **multicast**. Multicasts serão vistos posteriormente. Endereços que começam com 11110 são reservados para uso futuro. A figura 3.4 exibe as classes de endereçamento IP.

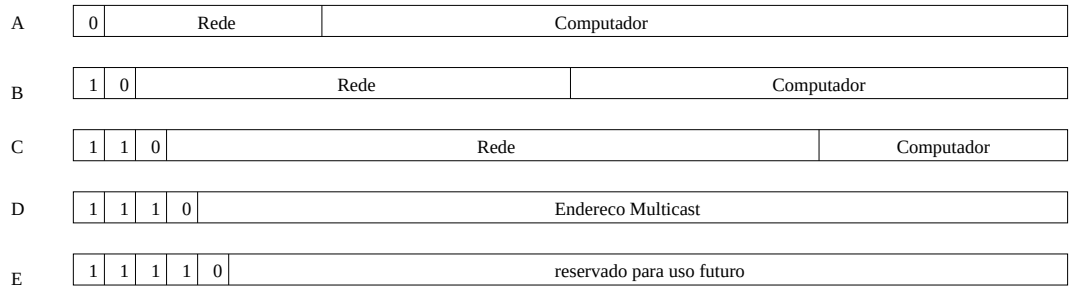


Figura 3.4: Classes de endereços IP

3.4.2 Endereços Especiais

Vários endereços especiais são definidos no endereço IP. Para se designar uma rede, deixam-se todos os bits que especificam o computador em zero. Assim, a rede da USP é especificada por 143.107.0.0. Note que não se deve atribuir o endereço zero a nenhum computador.

Se todos os bits do endereço do computador são 1, o endereço se refere a todos os computadores da rede. Fala-se então em endereço de **broadcast**. Nem todas as redes conseguem implementar o broadcast. Um caso importante é quando um computador quer mandar um broadcast mas não sabe o endereço da sua rede; nesse caso, fala-se de um broadcast local e o endereço é composto por 32 uns. Esse caso ocorre quando o computador acabou de ser ligado e não sabe ainda seu endereço IP.

Em alguns contextos bem definidos, os campos de rede e de computador serem zero especificam **esta** rede ou **este** computador.

Finalmente, a classe A 127 é reservada para **loopback**, ou seja, todo pacote enviado por um computador para um endereço 127 (por exemplo, 127.0.0.1) deve voltar para ele próprio **sem** passar por qualquer interface de rede.

Existem alguns endereços reservados para redes privadas, ou seja, redes que nunca serão conectadas à Internet (por exemplo, por segurança). Esses endereços não devem ser roteados. São os seguintes os endereços IP privados: classe A 10.0.0.0, classes B de 172.16.0.0 a 172.31.0.0 e as classes C que começam com 192.168.

3.4.3 Problemas com o Endereçamento IP

Só temos falado das vantagens do endereçamento IP. Deve-se mencionar porém, que sérios problemas existem. De fato, o maior problema de tal endereçamento é que a quantidade de números IP existentes é muito pequena para a atual demanda. No início, tinha-se a impressão que 32 bits eram mais do que suficientes mas o crescimento explosivo da Internet mostrou que essa hipótese era falsa.

Outro problema é que, se mudamos um computador de local (na verdade, de rede) seu número IP tem que ser mudado. Não existe o conceito, como na Ethernet, de um endereço associado a uma interface ou computador.

Um dos problemas mais sérios na administração de endereços IP surge quando, por qualquer motivo, os números precisam ser mudados. A causa pode ser, por exemplo, que uma rede que usava uma classe C, passa a ter mais de 255 nós e precisa de mais endereços. O tempo perdido para se modificar os endereços e tudo voltar a funcionar como anteriormente é muito grande.

Note que, apesar de termos nos referido ao número IP como um endereço de um computador, na verdade ele é o endereço de uma **interface** que liga o computador à rede. Assim, computadores com mais de uma conexão de rede (roteadores estão sempre nessa categoria) tem mais de um endereço IP. Isso causa um problema interessante: dados enviados para um computador com mais de um IP podem ter destinos diferentes, dependendo do endereço IP usado. De fato, um computador pode ser considerado **unreachable** quando se usa um número IP, e **reachable** quando se usa outro.

3.4.4 Atribuição de endereços

A entidade máxima que distribui endereços IP é a **Internet Network Information Center** ou **INTERNIC**. Note que o INTERNIC só distribui **redes** para as organizações; os endereços específicos dos computadores são atribuídos pela própria organização da maneira que melhor lhe convier. Se a organização preferir subdividir seu espaço de endereçamento usando **subnets** (subnets serão vistas posteriormente), ela não precisa de autorização do INTERNIC.

Capítulo 4

Os protocolos ARP e RARP. Os comandos tcpdump e arp

4.1 Introdução

Como foi visto anteriormente, no conjunto de protocolos TCP/IP cada máquina tem um endereço IP de 32 bits que lhe é atribuído através de configuração. Assim, o endereço IP não está associado a nenhuma máquina em particular até que esta máquina seja especificamente configurada para receber aquele número. Porém, suponha que o IP esteja "sobre" Ethernet: como saber qual o endereço físico (Ethernet) destino de um pacote quando se sabe apenas o número **IP** destino? Nas seções seguintes veremos como é possível resolver esse problema e o problema inverso, qual seja, se uma estação sabe o seu próprio endereço Ethernet, como ela poderá saber o seu número IP considerando-se que ela não tem disco (**diskless workstation**).

4.2 O Protocolo ARP

O "Address Resolution Protocol", ARP, surgiu para resolver o problema de mapeamento de endereços IP para endereços de nível de dados (ou físicos). De fato, os protocolos de nível superior, só querem tratar com um endereço: o endereço IP. Porém, na camada de dados, o número IP é completamente desconhecido. Seja por exemplo, o Ethernet: qual será o endereço destino do "frame" Ethernet quando uma máquina quer transmitir para a máquina 143.107.228.1 (por exemplo, a ultra3000 do IFSC). Qual é o endereço Ethernet da interface do computador 143.107.228.1?

A solução mais simples ocorre quando o endereço da camada de dados é pequeno e pode ser configurável. Assim, suponha que o endereço da interface seja composto de apenas um byte: pode-se configurar esse endereço como sendo o último byte do número IP (numa classe C) e assim, o mapeamento é óbvio. Em algumas redes físicas, esse tipo de solução é possível como por exemplo, na proNET.

No caso da Ethernet, Token Ring, FDDI e ATM, uma solução simples como a descrita acima não funciona pois os endereços envolvidos são muito grandes. Também, manter uma tabela

de mapeamento de endereços causaria sérios problemas de manutenção. A solução sugerida foi a criação de um protocolo específico para fazer o mapeamento que recebeu o nome de ARP. O funcionamento do ARP é simples: quando um computador quer transmitir um pacote para um outro computador cujo endereço Ethernet é desconhecido, ele emite um pacote de "broadcast" na Ethernet com o seu IP e o IP do computador destino. Esse "frame" é, na verdade, uma requisição para o computador destino devolver seu endereço físico. Todos os computadores da rede local recebem o pedido mas somente aquele cujo IP é especificado como destino no pacote, responde. O mapeamento provido pelo ARP é dito **dinâmico** pois é executado automaticamente quando for necessário, sem a intervenção da aplicação e sem configuração da máquina. Note que o ARP só é possível em redes com capacidade de "broadcast". O ARP é especificado pelo RFC826 escrito por Plummer em 1982 (ver http://www.faqs.org/rfcs/rfc826.html).

Depois do primeiro "broadcast", o computador origem armazena o endereço físico do computador destino numa "cache" de tal forma que, numa futura troca de pacotes, ele já saiba o endereço e não precise utilizar o ARP novamente. Também, o computador destino já armazena o endereço físico do computador origem, de forma que ele não precise utilizar o ARP quando precisar enviar pacotes para o computador origem. Na verdade, como todas as máquinas receberam o pacote ARP, **todas** podem armazenar o seu endereço físico.

O "frame" ARP tem um tipo especial na Ethernet: 0x0806. Esse valor é o mesmo tanto para o **pedido** ARP quanto para a **resposta**.

4.3 O formato do pacote ARP

Tanto o pedido ARP quanto a resposta ARP tem o mesmo formato. Esse formato é mostrado na figura 4.1.

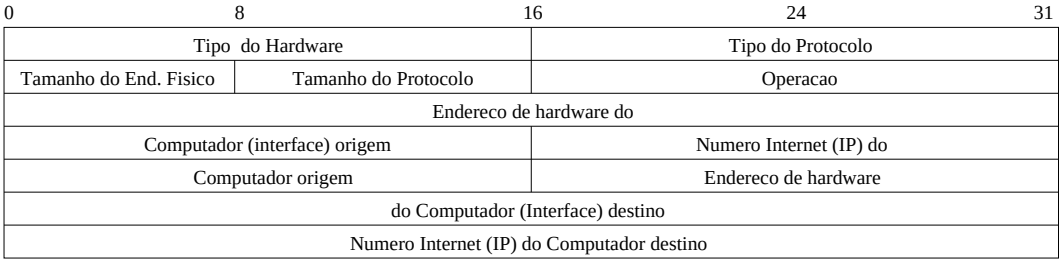


Figura 4.1: O pacote ARP/RARP

Note que esse pacote está no campo de dados do "frame" Ethernet sendo precedido pelos endereços destino e origem (12 bytes) e pelo tipo (0x0806) do "frame", e sendo seguido pelo CRC.

O pacote mostrado na figura 4.1 é utilizado pelo pedido ARP, resposta ARP, pedido RARP e resposta RARP. O campo do **Tipo do Hardware** especifica o tipo de endereço físico que está sendo pedido ou respondido. No caso do Ethernet, o valor desse campo é 1. O **Tipo de Protocolo** especifica o tipo de endereço lógico (rede) que será utilizado. No caso do IP, este campo vale 0x800 (que é o mesmo do tipo IP no "frame" Ethernet).

O campo **Tamanho do Endereço Físico** especifica o tamanho em bytes do endereço da camada de dados: para o Ethernet, esse valor é 6. O campo **Tamanho do Protocolo**

especifica o tamanho em bytes do endereço da camada de redes. No caso de IP, esse campo tem o valor 4.

O campo **operação** mostra que tipo de operação o pacote representa: 1 para pedido ARP, 2 para resposta ARP, 3 para pedido RARP e 4 para resposta RARP.

Os campos seguintes representam os endereços de origem (físico e de rede) e os endereços destino (físico e rede). Note que, na figura, são reservados 6 bytes para os endereços físicos e 4 para os lógicos. Isso vem do fato que o exemplo é baseado no ARP sendo utilizado com IP sobre Ethernet. Na verdade, o ARP pode ser utilizado com outras redes físicas e com outros protocolos da camada de rede.

4.4 O comando tcpdump

Um comando muito importante para o aprendizado de redes é o tcpdump, que mostra o que trafega pela rede através de uma determinada interface. O tcpdump coloca a interface em modo promíscuo e faz com que ela pegue todos os pacotes que trafegam pela rede. Em sistemas UNIX do tipo BSD, o tcpdump se utiliza de um dispositivo no kernel chamado de BPF ("BSD Packet Filter").

Nesse curso, vários exemplos serão exibidos utilizando-se da saída do **tcpdump**. A saída desse programa não é fácil de ser lida já que os dados vem quase que sem processamento. Porém, eles dão informações completas sobre o que se passa na rede.

O formato do comando em máquinas do tipo UNIX BSD é

```
ixtoto# man tcpdump

TCPDUMP(1)                                                TCPDUMP(1)

NAME
       tcpdump - dump traffic on a network

SYNOPSIS
       tcpdump [ -deflnNOpqStvx ] [ -c count ] [ -F file ]
               [ -i interface ] [ -r file ] [ -s snaplen ]
               [ -w file ] expression
```

Algumas opções importantes do programa são:

- **-c count**: pára depois de **count** pacotes terem sido recebidos.
- **-i interface**: ouve a interface especificada.
- **-x**: dá os bytes em hexadecimal.
- **-e**: imprime o cabeçalho da camada de dados em cada linha.

4.5 Exemplo do Funcionamento do ARP

Abaixo é reproduzida a saída do comando tcpdump da máquina **ixtoto**. Era feito, simultaneamente um **ping** da ixtoto para a máquina 200.1.1.2 localizada na mesma rede Ethernet.

```
ixtoto# tcpdump -e -x -c 8 -i ed0
tcpdump: listening on ed0
00:17:31.534685 0:80:48:84:91:78 Broadcast arp 42: arp who-has
200.1.1.2 tell  200.1.1.1
                         0001 0800 0604 0001 0080 4884 9178 c801
                         0101 0000 0000 0000 c801 0102
00:17:31.536473 0:80:48:84:8c:82 0:80:48:84:91:78 arp 60: arp reply
200.1.1.2  i-at 0:80:48:84:8c:82
                         0001 0800 0604 0002 0080 4884 8c82 c801
                         0102 0080 4884 9178 c801 0101 0101 0000
                         305c b97c 0000 0000 6978 746f 746f
00:17:31.536696 0:80:48:84:91:78 0:80:48:84:8c:82 ip 98: 200.1.1.1 >
200.1.1.2:icmp: echo request
                         4500 0054 35eb 0000 ff01 f3b7 c801 0101
                         c801 0102 0800 07e2 e408 0000 9bba 5c30
                         2127 0800 0809 0a0b 0c0d 0e0f 1011 1213
                         1415 1617 1819
00:17:31.538458 0:80:48:84:8c:82 0:80:48:84:91:78 ip 98: 200.1.1.2 >
200.1.1.1:icmp: echo reply
                         4500 0054 35eb 0000 ff01 f3b7 c801 0102
                         c801 0101 0000 0fe2 e408 0000 9bba 5c30
                         2127 0800 0809 0a0b 0c0d 0e0f 1011 1213
                         1415 1617 1819
```

Note que, no pedido ARP, o endereço destino Ethernet não é fornecido. Na resposta, o computador destino preenche o valor que falta, inverte os campos de destino e origem, muda a operação e envia o "frame" de volta.

4.6 O Comando arp

Existe um comando no Unix para se verificar o endereço Ethernet de uma máquina conectada à rede local: é o comando arp. Sua descrição é:

```
ARP(8)                   UNIX System Manager's Manual                   ARP(8)

NAME
     arp - address resolution display and control

SYNOPSIS
     arp hostname
     arp -a
     arp -d hostname
```

```
arp -s hostname ether_addr [temp] [pub]
arp -f filename
```

No modo mais simples, faz-se arp hostname para se saber o endereço Ethernet de "hostname".

Por exemplo:

```
ixtoto# arp 200.1.1.2
? (200.1.1.2) at 0:80:48:84:8c:82
```

A opção **-a** mostra todos os endereços na "cache". A opção **-d hostname** apaga o endereço de **hostname** da "cache". A opção **-s hostname ether_addr** cria uma entrada na tabela *cache*.

4.7 O Protocolo RARP

O problema do RARP (Reverse Address Resolution Protocol) é, aproximadamente, o inverso do ARP, ou seja, sabe-se o endereço Ethernet e quer-se saber o endereço IP. Esse caso ocorre em máquinas que não tem disco ("diskless workstations"). Elas não sabem seu endereço IP mas precisam desse endereço para pegar de uma máquina servidora o código necessário para fazer o "boot". A única informação que essas máquinas tem é o seu próprio endereço Ethernet já que esse endereço é fixo para sua interface. Um software local está habilitado a descobrir sozinho, esse endereço. Na verdade, considere o comando abaixo que configura uma interface Ethernet com um endereço IP na máquina ixtoto:

```
ixtoto#ifconfig ed0 inet 200.1.1.1 netmask 0xffffff00
```

Se depois executarmos o comando para verificar o estado da interface ed0, teremos:

```
ixtoto# ifconfig ed0
ed0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 200.1.1.1 netmask 0xffffff00 broadcast 200.1.1.255
        ether 00:80:48:84:91:78
```

Note que o endereço Ethernet, que não foi especificado, já é atribuído automaticamente à interface ed0.

A máquina sem disco gera um "broadcast" fazendo um pedido RARP: ela fornece seu endereço Ethernet e pede que um servidor qualquer lhe forneça o seu endereço IP. Uma vez obtido o endereço IP, ela guarda esse endereço e não faz mais um pedido RARP até o próximo "boot".

O formato do RARP é o mesmo visto para ARP com as modificações no campo de **operação**. Também, o tipo do "frame" Ethernet é 0x8035. A especificação do RARP é o RFC 903 de 1984.

Ao contrário do ARP, um pacote RARP pode ser respondido por várias máquinas, ou seja, as servidoras. É preciso cuidado numa rede Ethernet para que o esquema funcione sem problemas. O RARP fornece apenas um endereço IP; existem outros protocolos para máquinas sem disco que retornam mais informações tais como o BOOTP ("Boot Protocol") e o DHCP (RFC2131 e RFC3315).

Capítulo 5

O Datagrama IP, a base da Internet

5.1 Introdução

Os protocolos vistos anteriormente (ARP e RARP) são considerados protocolos da camada de dados. Passamos agora à camada de rede e discutimos o principal protocolo usado hoje nesse nível que é a base da comunicação na Internet: o **Internet Protocol** conhecido simplesmente como **IP**.

5.2 Características do IP

Pode-se dizer que o IP tem três características fundamentais:

1. **Sem conexão:** De fato, os pacotes IP são enviados separadamente e independentemente uns dos outros. Alguns podem, inclusive, seguir por caminhos diferentes de outros. Não é necessário o estabelecimento de conexão antes do início da transmissão dos pacotes IP. A analogia aqui é com o serviço de correios onde os pacotes IP são as cartas.

2. **Não confiável:** a entrega de pacotes IP não é garantida sendo possível haver perda, mudança de ordem, duplicação, demora excessiva na entrega do pacote sem que o sistema avise o emissor ou o receptor.

3. **Máximo esforço:** Apesar dos problemas que caracterizam o serviço como **não confiável**, o software sempre tenta enviar os pacotes da melhor maneira possível; nunca um pacote é propositalmente jogado fora. Falhas ocorrem só quando as camadas inferiores falham ou quando não existe mais capacidade de transmissão devido ao excessivo uso de recursos.

Como podem, porém, as aplicações tolerarem o fato do IP ser um serviço não confiável? É importante lembrar que falamos aqui da camada 3. A maioria das aplicações espera que a camada 4 (no caso do TCP/IP seria o protocolo TCP) introduza a confiabilidade necessária.

O protocolo IP define o formato do pacote, que é chamado de **datagrama**, a forma como os datagramas devem ser roteados através da internet e como os datagramas devem ser processados (como e quando gerar mensagens de erro, quando descartar um pacote, etc).

5.3 O Datagrama IP

Assim como na Ethernet (frame), o datagrama IP é composto de duas partes: um cabeçalho contendo os endereços IP envolvidos bem como outras informações importantes para o correto funcionamento do protocolo e a parte dos dados propriamente dita. A figura 5.1 ilustra o datagrama IP.

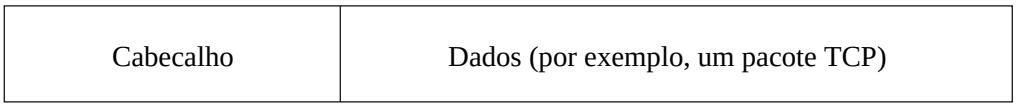


Figura 5.1: O datagrama IP

Note que o datagrama completo é enviado encapsulado no frame da camada de dados. Por exemplo, na Ethernet teríamos:

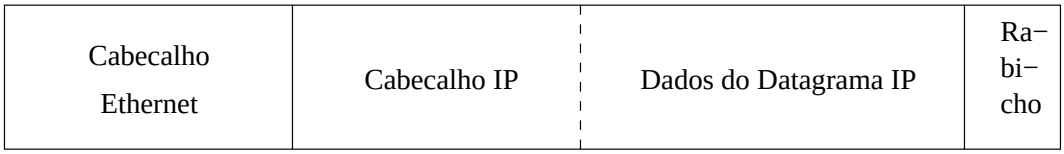


Figura 5.2: O datagrama IP dentro do frame Ethernet

O formato do datagrama IP pode ser visto na figura 5.3.

Note que a representação do datagrama vai do bit 0 mais à esquerda até o 31 na direita. O byte de 0 a 7 é transmitido primeiramente sendo seguido do bytes que compreende os bits de 8 a 15 e assim, sucessivamente. Essa é a ordem recomendada pela Internet e todos os computadores devem segui-la.

Os 4 primeiros bits do datagrama revelam a versão do protocolo IP sendo utilizado. A versão atual é 4. O campo seguinte revela qual o comprimento do **cabeçalho** IP em múltiplos de 32 bits. Como ele é composto de apenas 4 bits, o comprimento máximo do cabeçalho IP é de 15 palavras de 32 bits (60 bytes no total). O único campo do cabeçalho que apresenta tamanho variável é o de opções com o correspondente enchimento (para completar os 32 bits). Se não houver opções, o valor desse campo é 5 como pode ser visto na figura 5.3.

O campo de **tipo de serviço** é composto por três bits de precedência, que são normalmente desprezados, 4 bits de tipo de serviço propriamente dito e de 1 bit que deve ser sempre zero. O formato desse campo pode ser visto na figura 5.4.

A precedência vai de 0 (menor precedência) até 7 (maior precedência) permitindo a diferenciação de importância entre os datagramas. Como já foi dito, esse campo não é normalmente implementado. Os 4 bits de tipo são (na ordem): mínima latência, máxima largura de banda, máxima confiabilidade e mínimo custo financeiro. Só um desses bits deve estar ligado

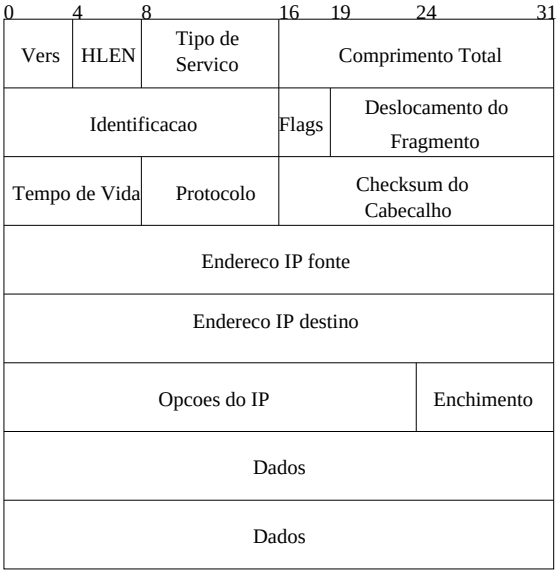


Figura 5.3: Formato do datagrama IP

Precedencia	Tipo de servico	0

Figura 5.4: O formato do **tipo de serviço**

revelando como o datagrama deve ser tratado. Se o tipo do datagrama for **normal**, os quatro bits devem ficar em zero.

Os valores recomendados de tipo de serviço para as várias aplicações são:

- **Telnet ou rlogin:** 1000.
- **FTP controle:** 1000.
- **FTP dados:** 0100.
- **TFTP:** 1000.
- **SMTP comando:** 1000.
- **SMTP dado:** 0100.
- **DNS UDP:** 1000.
- **DNS TCP:** 0000.
- **ICMP:** 0000.
- **SNMP:** 0010.
- **BOOTP:** 0000.
- **NNTP:** 0001.

Apesar de muitas implementações do IP não usarem o tipo de serviço, esses bits podem funcionar como uma espécie de sugestão para os algorítmos de roteamento que podem se basear neles para tomadas de decisão de melhor rota.

O **comprimento total** dá o tamanho do datagrama IP em bytes. Com esse campo, juntamente com o campo de **comprimento de cabeçalho** é possível saber onde começa e onde acaba o campo de dados. O tamanho máximo de um datagrama IP é de 65535 bytes. Se o tamanho do datagrama for maior do que o tamanho máximo que pode ser transmitido pelo frame na camada de dados é necessário quebrar o datagrama em vários pedaços menores. Isso é conhecido como **fragmentação** e será visto em detalhes a seguir.

5.3.1 MTU e Fragmentação

Cada rede física impõe um limite máximo no tamanho de um frame que ela pode transportar. Assim, o tamanho dos dados carregados por um único frame fica limitado. A esse limite dá-se o nome de **Maximum Transfer (Transmission) Unit** ou MTU. O RFC 1191 (Mogul e Deering, 1990) dá alguns valores de MTU para redes comuns. Alguns desses valores podem ser vistos na tabela 5.1.

Rede	MTU em bytes
Ethernet	1500
IEEE 802.3	1492
Hyperchannel	65536
FDDI	4352
X.25	576
4 Mbps Token Ring	4464
16 Mbps Token Ring	17914
Point-to-point	296

Tabela 5.1: MTUs para várias redes

Muitas vezes, é necessário transmitir-se datagramas IP cujo tamanho total é maior que o tamanho máximo permitido pelo frame da camada de dados. Nesse caso, o IP prevê a quebra do datagrama em pedaços menores que sejam menores ou iguais ao MTU. A essa quebra dá-se o nome de fragmentação.

A fragmentação ocorre, tipicamente, quando um roteador recebe um datagrama por uma rede física com alto MTU e tem que retransmiti-lo por uma conexão com baixo MTU (por exemplo, de uma rede Ethernet para uma conexão ponto a ponto). Nesse caso, o roteador pega o datagrama, quebra-o em pedaços que sejam menores que o MTU da rede de saída e os envia. A única restrição que é imposta é que o tamanho dos fragmentos seja múltiplo de 8 bytes com exceção do último fragmento, é claro. Ao chegar ao seu destino, os fragmentos são montados e o datagrama original é reconstruído.

É exigido dos computadores e roteadores que eles sejam capazes de tratar de datagramas com tamanho até de 576 bytes.

É importante notar que, cada fragmento, tem o mesmo formato do datagrama IP já apresentado na figura 5.3.

Uma vez que um datagrama é fragmentado, os pedaços menores não são mais remon-

tados até chegarem ao seu destino final. Isso é feito para que roteadores intermediários não precisem armazenar fragmentos a espera dos outros. Uma grande desvantagem porém, é que a transmissão dos datagramas pode se tornar ineficiente se, no caminho entre origem e destino, houver uma conexão com MTU baixo, pois, depois dessa conexão, os datagramas que trafegarão serão necessariamente pequenos. Também, se um dos fragmentos se perder, os outros prosseguem seu caminho até o destino final sem que seja possível a regeração do datagrama original.

5.3.2 Fragmentação e o datagrama IP

Três campos do datagrama IP estão diretamente relacionados com a fragmentação: o da **identificação**, o de **flags** e o de **deslocamento de fragmento**.

O campo de **identificação** é único para cada datagrama gerado. Normalmente, a máquina origem tem um contador que é incrementado a cada envio de um datagrama e é utilizado no campo de identificação. Todos os fragmentos de um datagrama devem conter o **mesmo** número de identificação.

Os dois bits menos significativos do campo **flags** (que tem 3 bits) também tem relação com a fragmentação: o primeiro (mais significativo) é o bit **não fragmente**. Se um roteador precisar fragmentar um datagrama que contenha esse bit em 1, ele não o faz e retorna uma mensagem de erro de volta à origem. O bit menos significativo é o bit de **mais fragmentos**. Ele sinaliza se o fragmento é o último a ser enviado. Esse flag é necessário para que o computador destino saiba quando já pode remontar o datagrama. Note que o campo **comprimento total**, no caso de fragmentos, dá o comprimento total do fragmento e não do datagrama original.

O campo de **deslocamento de fragmento** dá o deslocamento do início do fragmento em relação ao início do datagrama original em múltiplos de 8 bytes. Essa informação é fundamental para a remontagem do datagrama original.

5.3.3 Time to live

O campo de **tempo de vida** ou *time to live* (TTL) especifica qual o tempo máximo que pode levar o percurso desde a origem até o destino. Originalmente, esse campo é dado em segundos mas, na maioria das implementações, ele fornece o número máximo de roteadores pelo qual o pacote deve passar. Cada roteador, antes de retransmitir o datagrama, decrementa o campo de 1. Se o campo chegar a zero o datagrama é descartado.

O TTL é extremamente útil para evitar que roteadores mal configurados acabem inundando a rede com datagramas que ficam circulando eternamente.

5.3.4 O Campo de Protocolo e de Checagem de Cabeçalho

Assim como o campo de **tipo** no frame Ethernet, o campo de **protocolo** no datagrama IP é utilizado para identificar o protocolo de nível superior que está utilizando o serviço IP. Alguns protocolos e seus números são mostrados na tabela 5.2. Esses números foram tirados de Postel, 1981 (RFC 790).

Número em Decimal	Protocolo
1	ICMP
6	TCP
17	UDP

Tabela 5.2: Exemplos de protocolos no datagrama IP

O campo de checagem de cabeçalho tenta garantir a integridade do cabeçalho do datagrama. Ele **não** inclue o campo de dados. Isso é feito para facilitar o trabalho dos roteadores que só precisam se preocupar com a integridade do cabeçalho. É responsabilidade das camadas superiores, garantir que os dados chegaram corretamente ao seu destino. O número é calculado considerando-se o cabeçalho como uma sequência de inteiros de 16 bits. Esses inteiros são somados em complemento de um e toma-se como "checksum", o complemento de 1 do resultado.

O código (em C) para cálculo do *checksum* é apresentado no RFC 1071 e pode ser visto a seguir:

```
RFC1071 - Section 4.1
The following "C" code algorithm computes the checksum with an inner
loop that sums 16-bits at a time in a 32-bit accumulator.

/* Compute Internet Checksum for "count" bytes
 *         beginning at location "addr".
 */
unsigned short GenerateChecksum(unsigned char* addr, int count)
{
   register long sum = 0;

   while( count > 1 )  {
      /*  This is the inner loop */
      sum += * (unsigned short) addr++;
      count -= 2;
   }

   /*  Add left-over byte, if any */
   if( count > 0 )
      sum += * (unsigned char *) addr;

   /*  Fold 32-bit sum to 16 bits */
   while (sum>>16)
      sum = (sum & 0xffff) + (sum >> 16);

   checksum = ~sum;
```

5.3.5 Campos de endereço, opções e dados

O datagrama carrega o endereço IP de origem e o de destino (ambos de 32 bits). Esses endereços não são alterados de forma alguma pelos roteadores pelos quais passa o datagrama.

O campo de **opções** fornece algumas opções úteis para gerenciamento de redes. Esse

campo, se existir, deve ter tamanho múltiplo de 32 bits. Algumas opções são o "Record Route" e o "Source Routing".

Finalmente o campo de **dados** carrega o dado propriamente dito. Por exemplo, ele pode carregar um pacote **TCP**.

5.4 Exemplo de Datagramas IP

A listagem a seguir mostra a saída do programa tcpdump executado na máquina ixtoto (200.1.1.1) quando, simultaneamente, era executado "ftp 200.1.1.2" da ixtoto para uma outra máquina na mesma Ethernet.

```
ixtoto# tcpdump -e -x -c 8 -i ed0
tcpdump: listening on ed0
00:48:07.251851 0:80:48:84:91:78 Broadcast ip 150: 200.1.1.1.who
> 200.1.1.255.who: udp 108
                         4500 0088 a440 0000 4011 4322 c801 0101
                         c801 01ff 0201 0201 0074 6579 0101 0000
                         3065 fc47 0000 0000 6978 746f 746f 0000
                         0000 0000 0000
00:48:13.314945 0:80:48:84:91:78 Broadcast arp 42: arp who-has
200.1.1.2 tell 200.1.1.1
                         0001 0800 0604 0001 0080 4884 9178 c801
                         0101 0000 0000 0000 c801 0102
00:48:13.316810 0:80:48:84:8c:82 0:80:48:84:91:78 arp 60: arp
reply 200.1.1.2 is-at 0:80:48:84:8c:82
                         0001 0800 0604 0002 0080 4884 8c82 c801
                         0102 0080 4884 9178 c801 0101 0101 0000
                         3065 fb93 0000 0000 6978 746f 746f
00:48:13.316989 0:80:48:84:91:78 0:80:48:84:8c:82 ip 58:
200.1.1.1.1209 > 200.1.1.2.ftp: S 1030935041:1030935041(0) win 16384 <mss 1460>
                         4500 002c a442 0000 4006 4484 c801 0101
                         c801 0102 04b9 0015 3d72 d201 0000 0000
                         6002 4000 b1de 0000 0204 05b4
00:48:13.318889 0:80:48:84:8c:82 0:80:48:84:91:78 ip 60: 200.1.1.2.ftp
> 200.1.1.1.1209: S 4007460864:4007460864(0) ack 1030935042 win 4096 <mss 512>
                         4500 002c 05b8 0000 3c06 e70e c801 0102
                         c801 0101 0015 04b9 eedd 0000 3d72 d202
                         6012 1000 f6a3 0000 0204 0200 2032
00:48:13.319199 0:80:48:84:91:78 0:80:48:84:8c:82 ip 54: 200.1.1.1.1209
> 200.1.1.2.ftp: . ack 1 win 16384
                         4500 0028 a443 0000 4006 4487 c801 0101
                         c801 0102 04b9 0015 3d72 d202 eedd 0001
                         5010 4000 daac 0000
00:48:13.427741 0:80:48:84:8c:82 0:80:48:84:91:78 ip 115: 200.1.1.2.ftp
> 200.1.1.1.1209: P 1:62(61) ack 1 win 4096
                         4500 0065 05b9 0000 3c06 e6d4 c801 0102
                         c801 0101 0015 04b9 eedd 0001 3d72 d202
                         5018 1000 5eb9 0000 3232 3020 7874 6f74
                         6f2e 6966 712e
```

É extremamente instrutivo tentar entender o porque desses números em hexadecimal nos pacotes IP.

Capítulo 6

Os Protocolos SLIP e PPP

6.1 Introdução

Apesar de não serem tratados normalmente como pertencentes ao conjunto de protocolos TCP/IP (pelo menos o SLIP), esses protocolos ganharam enorme popularidade entre os usuários e devem necessariamente ser conhecidos por todos aqueles que trabalham seriamente com redes de computadores. Eles podem ser considerados protocolos da camada 2, ou seja, equivalentes à Ethernet, Token Ring, etc.

6.2 A Idéia

Desde que redes de computadores se tornaram importantes, grande esforço tem sido feito no sentido de tornar o acesso a redes o maior possível. Para aumentar o número de pessoas que acessam as redes, é fundamental que se diminua o custo das conexões. O problema é particularmente forte quando se fala em WANs já que, aqui, o custo tende a ser bastante alto mesmo para conexões de pequena velocidade. De fato, considere como exemplo, o custo de uma linha dedicada com velocidade de 2 MB/s interligando duas cidades próximas do Estado de São Paulo; ele é de alguns milhares de reais mensais mais os custos de instalação e infra-estrutura.

Assim, o método clássico de acesso remoto tem sido o uso de uma linha telefônica comum, que a maioria dos estabelecimentos e mesmo indivíduos já possuem naturalmente. A conexão é feita através de um dispositivo chamado "modem" (de "modulador-demodulador") que se conecta ao computador via uma linha serial padrão do tipo RS232C. O custo desse arranjo é muito baixo pois só envolve a linha telefônica (que normalmente já existia), uma porta serial que existe na maioria dos computadores, e um modem em cada lado da conexão.

Apesar da pouca confiabilidade do arranjo, derivado do fato que linhas telefônicas não foram projetadas para transmitir dados e de linhas telefônicas serem de baixa qualidade, ele se tornou bastante popular devido ao baixo custo.

O funcionamento clássico é o seguinte: normalmente se liga um usuário remoto a um centro de computação que dispõe de várias linhas telefônicas dedicadas especificamente para

esse fim. O usuário se utiliza de um programa chamado de "emulador de terminal" que lhe dá acesso à porta serial e, consequentemente, ao modem. O usuário dá então os comandos para que o modem disque o número do centro de computação. Quando o modem do centro atende, estabelece-se a conexão e o computador do usuário se torna um terminal remoto do computador central.

Note que o cenário descrito **não** configura uma rede de computadores; de fato, o computador do usuário não toma parte da comunicação a não ser como um terminal "burro" do computador central. Nessa configuração a versatilidade é muito baixa sendo que o computador central não "vê" o computador do usuário. Talvez a única coisa possível, além é claro da funcionalidade de terminal, é a transferência de arquivos, desde que bastante **guiada** pelo usuário. Apesar de ser uma forma rudimentar de comunicação, devido a seu baixo custo, ela se tornou bastante popular e deu origem a literalmente milhões de sistemas específicos para essa finalidade chamados de BBS (do inglês, Bulletin Board System). Atualmente, os sistemas de BBS caíram em desuso, com a popularização da Internet.

Dois fatores porém, fizeram com que os usuários exigissem uma funcionalidade maior do que a das BBSs: os computadores pessoais se tornaram muito mais poderosos e a utilização deles como simples terminais parece ser uma enorme perda de recursos; em segundo lugar, a crescente popularidade da Internet fez com que os usuários desejassem cada vez mais ter a funcionalidade completa da Internet nos seus computadores pessoais.

Surge então o **SLIP** ou **Serial Line IP**. A idéia do SLIP é fornecer uma base para que se trafeguem pacotes IP sobre linhas seriais de tal forma que o computador do usuário se torne efetivamente um membro da rede com a qual ele se conecta. O SLIP foi definido (ou sugerido) no RFC 1055 escrito em 1988. Apesar de ser uma solução relativamente rudimentar para o problema de redes de longa distância, ele se tornou **extremamente** popular nos anos 90 até ser substituído pelo PPP.

Os vários problemas do SLIP (descritos posteriormente), e ao mesmo tempo, sua enorme popularidade, levaram ao desenvolvimento de um protocolo que também apresentasse baixo custo mas que fosse um pouco mais **sólido** e apresentasse maior funcionalidade. Surge então o **Point to Point Protocol** ou **PPP** que mantém a mesma necessidade mínima de recursos do SLIP (uma linha serial assíncrona do tipo RS232C, um modem e uma linha telefônica) mas com recursos extras, que permitem, inclusive, trafegar pacotes não IP.

Durante muitos anos, o SLIP foi mais popular que o PPP, mas, devido às tendências verificadas nos últimos anos, esse panorama se inverteu. O suporte ao PPP em sistemas operacionais populares em meados da década de 90 como o Windows95 e o OS/2 Warp fizeram do PPP o método preferido de conexão remota barata às WANs. Antes de entrarmos em detalhes dos dois protocolos, é importante que se tenha alguns conhecimentos básicos de modens.

6.3 O MODEM

O modem é, em sua forma original, um equipamento que transforma sinais digitais oriundos de uma interface computacional para sinais analógicos que são transmitidos através de uma linha telefônica comum. Os sinais digitais devem então ser **modulados** para serem transmitidos analogicamente. No destino, o modem retorna a informação contida no sinal analógico para sua forma digital original (a **demodulação**). A modulação e demodulação são necessárias para que não haja perda de sinal na linha telefônica, que foi projetada para transmitir sinais analógicos, no caso, a voz humana. O grande limitante na largura de banda conseguida pelos modens vem

do fato que as linhas telefônicas comuns só permitem a passagem de frequências até 4khz (não é necessário mais do que isso para se manter uma conversa entre dois seres humanos de forma que o entendimento de ambas as partes seja razoável; note porém que muitas vezes não é possível se reconhecer a voz da outra pessoa).

Com a evolução da tecnologia e dos métodos de modulação, os modens têm se tornado cada vez mais rápidos sendo velocidades de até 33600 bits/segundo (56K) possíveis atualmente. Não é o objetivo desse curso dar ao aluno uma idéia profunda do funcionamento dos modens (o que, por si só, seria material para um curso completo). É importante porém, que o aluno esteja familiarizado com alguns padrões e comandos básicos associados aos modems. Alguns padrões muito populares atualmente são:

- **V.22:** Padrão CCITT que especifica como dois modens devem se comunicar em velocidades de até 1.200 bps (bits/segundo).
- **V.22bis:** Padrão CCITT que especifica como dois modens devem se comunicar em velocidades de até 2.400 bps.
- **V.32:** Padrão CCITT que especifica como dois modens devem se comunicar em velocidades de até 9.600 bps.
- **V.32bis:** Padrão CCITT que especifica como dois modens devem se comunicar em velocidades de até 14.400 bps.
- **V.34:** Padrão CCITT para velocidades de 28.800 bps.
- **V.90:** Protocolo para conexões até 56Kbps.
- **MNP2-4:** Protocolos da firma americana Microcom, que fornecem métodos de correção de erros.
- **V.42:** Protocolo CCITT para correção de erro. Deve ser usado ao invés dos protocolos proprietários MNP2-4.
- **MNP5:** Protocolo de compressão de dados da firma americana Microcom. Permite compressão de até 2:1. Se o dado já estiver comprimido, o MNP5 pode até **aumentar** o tamanho dos dados.
- **V.42bis:** Protocolo de compressão do CCITT. Deve ser utilizado ao invés do MNP5 pois apresenta uma taxa de compressão maior (4:1) e nunca aumenta o tamanho dos dados.

6.3.1 Comandos Hayes

O fabricante americano de modens Hayes criou um conjunto de comandos que definem a comunicação entre o computador e o modem. Esses comandos se tornaram bastante populares e acabaram se transformando num padrão utilizado por praticamente todos os fabricantes. O conjunto de comandos Hayes é enorme e, infelizmente, varia de fabricante para fabricante. Os comandos sempre começam com as letras **at** do inglês **attention** e são as vezes chamados de comandos *at*. Os comandos mais importantes são (em geral!!):

- **atdt**< *numero* > disque o número telefônico a seguir utilizando-se da discagem por **tom**. O número pode conter vírgulas que indicam que o modem deve aguardar um certo tempo para continuar a discar.

- **atdp**< *numero* > o mesmo anterior, só que, agora, use a discagem por **pulso**.
- **ata** Coloque o modem imediatamente em modo **resposta**.
- +++ Cada mais deve ser separado do outro por aproximadamente um segundo. Retorne ao modo comando (quando, por exemplo, um modem já se conectou ao outro e se encontra no modo de dados).
- **ato** Volta ao modo de dados.
- **at&c1** Leve em consideração o estado da portadora (ligado ou desligado).
- **at** Só requisita uma resposta do modem (tipicamente, OK).
- **at&v** Dá a configuração atual do modem.
- **ath** Encerra a conexão.
- **ats0=002** Coloque o valor 2 no registrador zero. Esse registrador controla o número de chamadas antes que o modem atenda ao telefone. No caso do exemplo, ele atenderá o telefone depois do segundo sinal.
- **at&d1 e at&d0** Leva em consideração (ou não) o estado do sinal DTR (*Data Terminal Ready*).

6.4 SLIP

As seguintes regras são utilizadas pelo SLIP:

- Cada datagrama IP é transmitido precedido e seguido por um byte 0xc0 que é considerado o byte END do pacote SLIP.
- Se um dos bytes do datagrama é 0xc0, ele é transmitido como a sequência de dois bytes 0xdb 0xdc. O byte 0xdb é chamado de "escape" do SLIP ou ESC SLIP.
- Se o datagrama contém o byte 0xdb, este é transformado na sequência 0xdb 0xdd.

É de se esperar que um protocolo tão simples deva ter deficiências. As principais deficiências do SLIP são:

1. Não contempla correção de erros. Note que não existe no SLIP o CRC que é encontrado em outros protocolos. Assim, o SLIP deixa para os protocolos de nível superior a responsabilidade de fazer a correção de erro. Isso é particularmente problemático com linhas telefônicas comuns se o modem não fizer a correção de erro (V.42 ou MNP2-4) pois linhas telefônicas são muito ruidosas.
2. Não contemplam compressão de dados.
3. Não permitem o tráfego simultâneo de protocolos outros que não o IP já que não contempla o **tipo** do protocolo.
4. Cada lado da comunicação deve saber, previamente, o número IP que lhe está reservado nessa particular conexão.

O problema da compressão, pelo menos do cabeçalho IP, é resolvido com uma variação do SLIP chamada de CSLIP (para "compressed SLIP"). O CSLIP consegue reduzir o tamanho do cabeçalho, de 40 bytes para 5 bytes ou algo similar. Isso é muito importante para aplicações interativas em linhas seriais lentas. O CSLIP também é conhecido como van Jacobson SLIP em homenagem ao seu idealizador.

6.5 PPP

O PPP é especificado nos RFC 1331 e 1332. O protocolo PPP tenta corrigir as deficiências do SLIP. Ele permite conexões assíncronas como o SLIP e síncronas. O PPP também define dois tipos de protocolos: o LCP ("Link Control Protocol") e o NCP ("Network Control Protocol"). O LCP estabelece, configura e testa a conexão o que permite a negociação de **opções** entre as duas partes. NCPs existem para diferentes famílias de protocolos tais como IP, X.25, DECNET, etc. No caso do IP, o NCP permite a compressão do cabeçalho como no CSLIP. Também permite a negociação de números IP entre os computadores. O formato do pacote PPP é visto na figura 6.1.

Como no SLIP, o pacote PPP é precedido e seguido de um byte fixo, 0x7E. Após o primeiro 0x7E, segue-se um byte de endereço que é sempre 0xFF e um de controle que é 0x03. O campo de protocolo especifica que tipo de pacote é transmitido como dado. Por exemplo, o pacote IP vem com tipo 0x0021, o NCP vem como 0x8021 e o LCP, 0xC021. O campo de CRC serve para verificar se a transmissão se deu sem erros.

0x7E 1 byte	end. 0xFF 1 byte	contr 0x03 1 byte	Protocolo 2 bytes	Dados Ate 1500 bytes	CRC 2 bytes	0x7E 1 byte

Figura 6.1: O formato do pacote PPP

O byte 0x7E é substituído por 0x7D 0x5E. O byte 0x7D é transmitido como 0x7D 0x5D. Também, todos os bytes entre 0 e 32 ASCII são transmitidos com o caracter de "escape" (0x7D). Isso é muito útil para permitir que os "drivers" das portas seriais usem os caracteres de controle ASCII (por exemplo, o de controle de fluxo).

Capítulo 7

Interface com Sistema Operacional, parte 1

7.1 Introdução

Neste capítulo, veremos como é feita a interface do software de rede com o sistema operacional do ponto de vista do usuário (ou do gerente do sistema). A interface, do ponto de vista do implementador, será vista no futuro.

O sistema básico para onde foi primeiro desenvolvido o TCP/IP foi o Unix (Berkeley). Assim, os comandos apresentados serão sempre para o UNIX a menos que se diga que se referem a outro sistema. Note que, como o TCP/IP foi implementado primeiro para o UNIX, muitos dos comandos foram implementados com o mesmo nome em outros sistemas ou com nomes bastante parecidos. A funcionalidade de tais comandos é também, bastante similar nos vários S.O.

7.2 Comandos Fundamentais

Vários são os comandos utilizados para o correto funcionamento do TCP/IP em máquinas UNIX. Nas seções a seguir, veremos vários desses comandos.

7.2.1 ifconfig

Um dos primeiros comandos que devem ser utilizados para configurar o TCP/IP numa máquina é o **ifconfig**. O nome do comando deriva-se da expressão "Interface Configuration".

O ifconfig é utilizado para se associar um número IP a uma interface de rede do computador. Além do número IP, outras carecterísticas de rede podem ser atribuídas à interface tais como a **máscara da rede** que define a que subrede o computador pertence, "flags" para ajustar certos atributos da conexão, etc.

A sintaxe do **ifconfig** no UNIX BSD (na verdade, no FreeBSD) é:

```
IFCONFIG(8)              UNIX System Manager's Manual              IFCONFIG(8)

NAME
     ifconfig - configure network interface parameters

SYNOPSIS
     ifconfig interface address_family [address [dest_address]] [parameters]
     ifconfig interface [protocol_family]
     ifconfig -a
     ifconfig -au
     ifconfig -ad

DESCRIPTION
     Ifconfig is used to assign an address to a network interface and/or con-
     figure network interface parameters.  Ifconfig must be used at boot time
     to define the network address of each interface present on a machine; it
     may also be used at a later time to redefine an interface's address or
     other operating parameters.
```

A seguir são apresentados alguns exemplos de utilização do comando:

#ifconfig ed0 10.0.0.1 netmask 255.255.255.0

Neste exemplo (utilizado na máquina ixtoto para a configuração da rede local), configura-se a interface **ed0** para o endereço IP 10.0.0.1 e classe C. Para desligar a interface pode-se fazer:

#ifconfig ed0 down

Outro exemplo é:

#ifconfig ed0 143.107.228.49 netmask 0xffffffc0

Esse comando configura a interface da máquina **rapper** para funcionar na rede do DFI-IFSC. Note que, neste exemplo, a máscara de rede é especificada em hexadecimal e, apesar de 143.107 ser originalmente uma classe B, o mascaramento de subrede indica uma rede menor que uma classe C (na verdade, um quarto de classe C!)

Considere finalmente, o exemplo a seguir:

ifconfig sl0 143.107.228.51 143.107.228.49

Este comando configura a interface sl0 com o endereço IP 143.107.228.51 (na **ixtoto**). A interface agora é a sl0, que é uma interface do tipo **slip**. O endereço 143.107.228.49 é o IP da máquina diretamente conectada a ixtoto pelo slip, que é a **rapper**.

O **ifconfig** é um comando simples e importante. A maior dificuldade na utilização desse comando é conseguir-se o nome da interface que se quer configurar. De fato, alguns sistemas usam nomes difíceis de se memorizar. Alguns nomes lembram os fabricantes da interface. Assim, no SunOS 4.1.3, a interface Ethernet mais comum é a le0, que lembra **Lance Ethernet**. Note que, se tivermos mais de uma interface do mesmo tipo, teremos le0, le1, le2, etc. É comum uma interface do tipo Novell (placas NE1000 e NE2000) receberem o nome de ne0 de Novell Ethernet. Se a placa é a Intel EtherExpress, a interface pode se chamar ix0. No FreeBSD, alguns nomes usados são:

- **ed0:** nome da interface para NE1000, NE2000, WD8003, WD8013, 3COM503.
- **ix0:** Intel EtherExpress.
- **de0:** DEC DC21x40 PCI que inclui uma placa para Ethernet 100.
- **eg0:** 3C505.
- **ep0:** 3C509.
- **ie0:** AT&T Startlan 10 e EN100; NI5210.
- **le0:** DEC Etherworks 2 e 3.
- **lnc0:** Lance PC/Net cards (ISOLAN, NE2100, NE32-VL).

Normalmente, o comando localiza-se em **/sbin/ifconfig**.

No Linux, um exemplo de **ifconfig** seria, para uma interface Ethernet do tipo Novell 2000:

#/sbin/ifconfig eth0 200.1.1.1

Note que, a única diferença do comando no Linux em relação ao FreeBSD é o nome da interface.

O sistema OS/2 Warp Connect (que contém o pacote de TCP/IP 3.0 da IBM) vem com o comando **ifconfig** que fica localizado normalmente em `C:\TCPIP\BIN`. O comando apresenta uma sintaxe similar àquela encontrada no Unix.

7.2.2 route

Dois tipos de roteamento existem no TCP/IP: o roteamento **estático** onde informações de roteamento ficam armazenadas numa tabela **fixa** e **dinâmico**, onde informações de roteamento são trocadas entre os vários roteadores. Roteamento dinâmico é muito importante no TCP/IP e será visto posteriormente. No roteamento estático, as rotas são manuseadas na tabela através do comando **route** (se pronuncia "raute" no inglês americano e "ruute" no inglês britânico).

A sintaxe do programa **route** é:

```
ROUTE(8)                UNIX System Manager's Manual                ROUTE(8)

NAME
     route - manually manipulate the routing tables.

SYNOPSIS
     route [-nqv] command [[modifiers] args]

DESCRIPTION
     Route is a utility used to manually manipulate the network routing ta-
     bles.  It normally is not needed, as a system routing table management
     daemon such as routed(8),  should tend to this task.

     The route  utility supports a limited number of general options, but a
     rich command language, enabling the user to specify any arbitrary request
     that could be delivered via the programmatic interface discussed in
     route(4).
```

As opções de comando mais importantes são:

- **add:** adiciona a rota à tabela de roteamento.
- **delete:** remove a rota da tabela.

Um exemplo do comando é dado a seguir:

route add default 143.107.228.49

Esse comando é utilizado na máquina ixtoto para dizer que, se não houver nenhuma outra rota específica, os datagramas IP devem ser enviados ao roteador 143.107.228.49 que é a **rapper**. Os datagramas são enviados pela interface sl0 que liga a **ixtoto** à **rapper**.

Outro exemplo seria:

#route add 143.107.228.51 143.107.228.49 2

Esse comando pode ser dado na máquina **zeppelin**, cujo IP é 143.107.228.5, para se acessar a **ixtoto**. Isso é necessário pois a **ixtoto** não está na rede local (ethernet) da zeppelin e sim conectada via LP até a **rapper**. O número 2 indica que dois "hops" devem ser percorridos para se chegar até a **ixtoto**. O uso desse dígito está, gradativamente, sendo descontinuado.

No Linux, o comando de roteamento é muito parecido com aquele encontrado no FreeBSD. Veja:

#route add default sl0

Note o uso do nome da interface diretamente no comando (sl0).

7.2.3 slattach

O comando **slattach** é utilizado para se definir uma interface de rede do tipo slip sobre uma interface serial. Esse comando não é padrão no UNIX mas muitas versões desse sistema operacional já incluem o comando.

A sintaxe do comando **slattach** pode ser vista a seguir:

```
SLATTACH(8)              UNIX System Manager's Manual              SLATTACH(8)

NAME
     slattach - attach serial lines as network interfaces

SYNOPSIS
     Slattach [-a] [-c] [-e exit-command] [-f] [-h] [-l] [-n] [-r
              redial-command] [-s baudrate] [-u unit-command] [-z] ttyname

DESCRIPTION
     Slattach is used to assign a tty line to a network interface, and to de-
     fine the network source and destination addresses.  The following
     operands are supported by slattach:

     -a            Autoenable the VJ header compression option, if the other
                   end of the link is capable of VJ header compression then it
                   will be used otherwise normal headers will be used.

     -c            Enables the VJ header compression option.  Note that both
                   ends of the link must be able to use VJ header compression
                   for this to work.

     -e exit-command
                   Specifies a command to be invoked within a shell (sh -c ex-
                   it-command) before slattach exits.

     -f            Disables the invocation of daemon() to run slattach in the
                   background.

     -h            Turn on cts/rts style flow control on the slip port, by de-
                   fault no flow control is done.

     -l            disable modem control (CLOCAL) and ignore carrier detect on
                   the slip port.  By default the redial-command is invoked
                   upon carrier drop and slattach aborts if no redial-command
                   is specified.

     -n            Throw away ICMP packets.  The slip interface will ignore
                   ICMP packets to prevent slow lines being saturated by ICMP
                   responses.

     -r redial-command
                   Specifies a command to be invoked within a shell (sh -c re-
                   dial-command) whenever carrier is lost on the line.
```

```
     -s baudrate   Specifies the speed of the connection. If not specified,
                   the default of 9600 is used.

     -u unit-command
                   When the line is switched to slip discipline, run ‘sh -c
                   unit-command <last> <current>’ where <last> and <current>
                   are the slip unit numbers when the line was last opened and
                   the unit number of the current slip connection respecitve-
                   ly.  The unit number can change after redialing if you are
                   using more than one slip line.  Slattach will abort if the
                   unit number changes and ‘-u unit-command’ was not speci-
                   fied.

     -z            forces redial redial-cmd upon startup irrespective of car-
                   rier.

     ttyname       Specifies the name of the tty device.  Ttyname should be a
                   string of the form ‘ttyXX or’ ‘/dev/ttyXX.’

     Only the super-user may attach a network interface.

     To detach the interface, use ‘‘ifconfig interface-name down’’ after
     killing off the slattach process using ‘kill -INT’. Interface-name is the
     name that is shown by netstat(1)

     To setup slattach to redial the phone when carrier is lost, use the ‘-r
     redial-cmd’ option to specify a script or executable that will reconnect
     the serial line to the slip server.  For example, the script could redial
     the server and log in, etc.

     To reconfigure the network interface in case the slip unit number
     changes, use the ‘-u unit-cmd’ option to specify a script or executable
     that will be invoked as ‘sh -c unit-cmd old new,’ where old and new are
     the slip unit numbers before and after reconnecting the line.  The unit
     number can change if you have more than one line disconnect at the same
     time.  The first to succeed in reconnecting will get the lowest unit num-
     ber.

     To kill slattach use ‘kill -INT’ (SIGINT) which causes it to close the
     tty and exit.

     To force a redial, use ‘kill -HUP’ (SIGHUP) which causes slattach to
     think carrier was lost and thus invoke ‘sh -c redial-command’ to recon-
     nect to the server.

     If you use a hard-wired connection rather than a modem, invoke slattach
     with the ‘-l’ option in order to ignore carrier on the slip line.

EXAMPLES
           slattach ttyh8
           slattach -s 4800 /dev/tty01
           slattach -c -s 38400 /dev/sio01
           slattach -r 'kermit -y dial.script >kermit.log 2>&1'

DIAGNOSTICS
```

```
     Look for error messages in /var/log/messages (slattach is a daemon).
     Messages indicating the specified interface does not exit, the requested
     address is unknown, the user is not privileged and tried to alter an in-
     terface's configuration are logged there.  Slattach also logs failure to
     set the controlling terminal or failure to install signal handlers.  Upon
     connection and redial the ttyname and baud rate are logged and on shut-
     down the ttyname is logged.

FILES
     /var/run/slattach.<tty>.pid,

     with <tty> replaced by the terminal path name component of ttyname. This
     file contains the numerical process ID of the slattach process and can be
     examined by scripts in oder to send a signal to slattch.

SEE ALSO
     netstat(1),  netintro(4),  ifconfig(8),  rc(8)

HISTORY
     The slattach command appeared in 4.3BSD.

4.3 Berkeley Distribution        April 4, 1993                              2
```

Um exemplo de utilização desse comando pode ser visto a seguir:

#slattach -s 115200 -h /dev/cuaa1

Esse comando é usado no FreeBSD para configurar o protocolo slip na linha serial 1 (/dev/cuaa1). A opção -s indica a velocidade da porta serial. A opção -h exige o *handshake* de hardware. O dispositivo serial 1 é a segunda porta serial do hardware do PC, que em DOS é chamado de COM2.

No Linux, o comando será:

#/usr/sbin/slattach -s 115200 -p slip /dev/cua1 &

Note que o protocolo slip é especificado e que é preciso colocar o sinal de "background" para o processo.

7.2.4 Resumo

O conjunto de comandos para configurar uma interface, com um exemplo do **ixtoto** executando FreeBSD é:

```
#slattach -s 115200 -h /dev/cuaa1
#ifconfig sl0 143.107.228.51 143.107.228.9 netmask 255.255.255.0
#ifconfig ed0 200.1.1.1 netmask 255.255.255.0
#route add default 143.107.228.9
```

7.3 O kernel

No UNIX, o programa que executa o sistema operacional é tratado pelo nome geral de kernel. Por exemplo, no FreeBSD esse programa está no diretório raiz e tem o nome **kernel**. Nomes como **vmunix** ou **unix** são comuns.

É importante saber se o kernel já tem os assim chamados **device drivers** para as interfaces de rede. Se não tiver, pode ser preciso recompilar o kernel incluindo o controlador de dispositivo. Sistemas operacionais modernos permitem que certos controladores possam ser "linkados" dinamicamente.

Mesmo no caso de um controlador de dispositivo que já exista no kernel, é importante verificar quais os parâmetros para os quais ele está configurado. Por exemplo, o kernel padrão do FreeBSD para a interface de rede ethernet ed0 (em barramento ISA - não há problema quando o barramento é PCI) vem configurado para endereço de I/O 0x280 e interrupção 5. É possível mudar esses os valores sem recompilar o kernel mas isso pode não ser possível em muitos sistemas operacionais.

7.4 Arquivos Importantes

Inúmeros arquivos são importantes na configuração do TCP/IP no UNIX; agora porém mencionaremos apenas dois. O primeiro é o arquivo de **hosts** que normalmente se localiza em **/etc/hosts**. Esse arquivo contém algumas ligações estáticas entre nomes de computadores e seus respectivos números IP. O arquivo a seguir mostra um exemplo de **hosts** simples.

```
127.0.0.1               localhost
143.107.180.179         ixtoto.ifsc.usp.br ixtoto
```

Os computadores desse arquivo podem ser acessados diretamente pelo nome. Outro arquivo importante na resolução de nomes é o **resolv.conf**, normalmente localizado no subdiretório **/etc/resolv.conf**. A seguir mostramos um exemplo simples desse arquivo:

```
domain  ifsc.usp.br
nameserver      143.107.228.5
```

Capítulo 8

DNS

8.1 O *Domain Name System*

Apesar de sua enorme importância no conjunto de protocolos TCP/IP, os mecanismos de resolução de nomes são raramente entendidos corretamente pelos iniciantes em redes de computadores. Trata-se também de um tópico não abordado (ou abordado apenas superficialmente) em cursos introdutórios de redes de computadores.

Toda interface de máquina ligada à Internet (ou, mais geralmente, ligada a qualquer rede que usa os protocolos TCP/IP) tem um número IP único a ela associado. Toda a comunicação entre duas máquinas envolve os seus números IP que estarão presentes nos pacotes IP por elas trocados. Lembre-se que o número IP é um inteiro de 32 bits normalmente representado como uma sequência de 4 bytes separados por um ponto.

Um problema óbvio do endereçamento IP para os usuários é que é muito difícil se aprender e memorizar os números IP de mais do que uma dúzia de máquinas. O problema é parecido com aquele de se utilizar o telefone: sempre que queremos ligar para alguém, é preciso saber qual é o seu número telefônico. A solução clássica da telefonia é a utilização de listas contendo, em ordem alfabética, todos os assinantes do serviço. Essa solução é inconveniente por vários motivos:

1. Mesmo se o número for achado na lista é preciso que ele seja anotado e discado no aparelho telefônico já que este não entende o conceito de nome. Esse processo leva a erros frequentes e se o usuário quiser fazer a mesma ligação duas horas depois, ele deverá novamente procurar na lista telefônica (a não ser que ele tenha uma excelente memória ou deixe um "post-it" próximo ao telefone com o número desejado o que por sua vez cria um outro problema: o excesso de papel ao redor do aparelho telefônico).

2. Não é fácil localizarem-se números telefônicos que não estejam na localidade de onde deverá partir a chamada. De fato, é impossível manter-se um conjunto de listas de todas as cidades do país e muito menos do mundo. A solução que as TELEs brasileiras adotam de fornecer números 102 em várias localidades ameniza o problema mas ainda ficamos numa situação insatisfatória já que é preciso contactar um operador humano com a consequente perda de tempo e é preciso também saber qual o DDD a ser colocado antes do 121. Por exemplo, para se obter informações de telefone na cidade de Blumenau em Santa Catarina, é preciso

discar-se 048-121 apesar do DDD de Blumenau ser 047. O problema se agrava em ligações internacionais onde tal serviço não existe e onde o usuário se defronta com o problema de falar uma língua estrangeira. Com a existência de mais de uma operadora telefônica no país, o problema se agrava ainda mais.

A solução da comunidade do TCP/IP foi a criação de nomes que devem ser, de alguma maneira, associados ao número IP já que só esses tem validade nos níveis mais baixos de protocolo. No início, esses nomes eram compostos apenas de uma sequência de caracteres que eram associados centralmente (no Network Information Center — NIC — hoje, INTERNIC) com o número IP da máquina a que ele se referia. Esse esquema de nomes rapidamente se tornou inviável com o crescimento explosivo de máquinas ligadas a Internet já que o problema de memorização e o de colisão de nomes acaba se tornando muito grave. Surge então o esquema hierárquico de nomes de computadores. Nesse esquema, o nome de uma máquina é dado como uma sequência de nomes separados por ponto; o nome mais a direita representa o nível mais alto da hierarquia (nível mais abrangente) enquanto o nome mais a esquerda representa o nível menos abrangente (normalmente o nome da máquina) e mais baixo na hierarquia.

Para se entender melhor essa nomenclatura, considere o nome da ultra3000 do IFSC:

uspfsc.ifsc.usp.br

O nome mais a direita, ou seja o "br" refere-se ao domínio mais amplo, no caso, o domínio Brasil. Dentro do "br" tem-se o domínio "usp" que se refere às máquinas da Universidade de São Paulo. Dentro do domínio (ou sub-domínio) usp, tem-se o sub-domínio sc que se refere aos computadores da USP em São Carlos. O "ifsc" refere-se ao Instituto de Física de São Carlos. Finalmente, o nome da ultra3000 é uspfsc.

A autoridade sobre os domínios amplos (mais a direita) ainda é do INTERNIC que delega os sub-domínios para as instituições de direito. Por exemplo, o domínio "br" é controlado pela FAPESP em São Paulo que por sua vez deu o domínio usp para a Universidade de São Paulo (aqui representada pelo Centro de Computação Eletrônica — CCE). O sub-domínio ifqsc é controlado no Instituto de Física e Química de São Carlos.

Note que o sistema hierárquico de nomes permite que diferentes organizações usem o mesmo nome para alguns computadores: de fato, praticamente todas as grandes empresas americanas tem uma máquina chamada (ou com um alias) www.nomedaempresa.com já que, se o nome da empresa muda, o nome é diferente.

Dentro da Internet, alguns nomes de domínio já estão definidos. Exemplos são: "edu" para o sistema educacional nos EUA, "com" para as empresas comerciais americanas, "gov" para instituições governamentais nos EUA, "mil" para o setor militar americano e "sigla" para os países fora dos EUA. Assim, o domínio para o Brasil é "br", para o Reino Unido é "uk" (United Kingdom), para a Alemanha é "de" (Deutschland) etc. No Brasil foi decidido que sob o "br", as universidades e centros de pesquisa ganhariam um espaco diretamente abaixo de "br": por exemplo temos "unicamp.br". Já as instituições comerciais ficariam todas no sub-domínio "com". O Banco Itaú, por exemplo, é autoridade no domínio itau.com.br enquanto a UNIMED responde por unimed.com.br.

O sistema de nomes descrito acima é conhecido como DNS do inglês "Domain Name System".

8.1.1 Resolução de nomes no DNS

Apesar do DNS ser um sistema intuitivo e fácil de se compreender, existem muitos detalhes na resolução de nomes que podem ser bastante confusos para o iniciante. Considere com atenção, portanto, os conceitos descritos abaixo.

Em primeiro lugar, deve-se ter em mente que a Internet pode funcionar perfeitamente **sem** a resolução de nomes já que os protocolos todos funcionam com o **número** IP e não com o nome. Assim, se queremos ter acesso a um computador remoto através do serviço de transferência de terminal, basta fazer *telnet IP* onde IP é o número IP do computador remoto e nenhuma resolução de nomes acontecerá (na verdade, alguns servidores telnet poderão tentar fazer uma resolução reversa mas deixemos esse detalhe para seções subsequentes).

Outro conceito que se deve ter em mente é que o DNS é um **serviço** que está conceitualmente no nível de aplicação do ISO/OSI, ou seja, na camada 7. Ele se utiliza do UDP para transportar as requisições e respostas. É comum para o iniciante em rede achar que o DNS é um protocolo de camada 3 e confundir o processo de resolução de nomes com o de roteamento com o qual o DNS nada tem a ver.

Em terceiro lugar, é importante lembrar que, assim como as demais aplicações do TCP/IP, o DNS é implementado como um sistema cliente-servidor. A diferença com aplicações como *telnet* e *ftp* é que a maioria das máquinas da Internet **não** está configurada para rodar o servidor de nomes enquanto praticamente todas executam o servidor *telnet* e o *ftp* (é claro que máquinas que rodam sistemas operacionais monousuários tais como DOS e Windows 3.1 têm mais dificuldades em executar esses servidores). As máquinas que rodam o servidor do DNS são chamadas de *name servers* ou *servidores de nome*. A maioria das máquinas porém, são *clientes* do DNS.

Finalmente, não se deve confundir o DNS com o Network Information Service (NIS) anteriormente conhecido como “yellow pages” ou YP. Apesar de ambos tratarem com bases de informações distribuídas, eles tem objetivo bastante distintos.

Analisemos então o funcionamento da resolução de nomes numa rede IP. O objetivo fundamental desta resolução é associar-se o nome conhecido ao número IP correspondente. Na verdade, é importante lembrar que o DNS também é usado para auxiliar no envio de mensagens eletrônicas (*e-mails*) em cujo caso, mais de um endereço IP pode estar envolvido. Consideremos primeiramente, porém, o caso mais simples onde simplesmente se deseja descobrir o endereço IP de uma determinada máquina cujo nome sabemos ser x.y.z.

O software cliente DNS da máquina que quer saber o número IP (conhecido por “resolvedor”) verifica na configuração local qual é o número IP do servidor de nomes que deve ser utilizado; uma requisição é então mandada para o servidor perguntando qual o endereço IP da máquina x.y.z. O servidor então verifica se ele é “autoridade” no domínio z. Se for, ele verifica em suas tabelas qual o IP da máquina x.y.z e retorna ao requisitante. Em caso negativo, ele envia uma requisição para um “servidor raiz” da Internet para descobrir quem é autoridade no domínio z. O servidor raiz então responde para o requisitante o endereço de um servidor de nomes que é autoridade no domínio z. O servidor de nomes inicial pergunta para o servidor que é autoridade em z qual o endereço de x.y.z. O novo servidor ou responde com o IP de x.y.z ou responde com o endereço de ainda um outro servidor de nomes que é autoridade em “y.z”. O processo continua até que o endereço IP de x.y.z seja determinado.

Alguns pontos devem ser ressaltados no processo descrito acima:

1. Todo servidor de nome deve ter uma lista de endereços dos "servidores raízes" para que ele saiba onde começar sua busca. Essa lista normalmente está localizada num arquivo chamado named.root e tem o formato mostrado a seguir:

```
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC registration services
;       under anonymous FTP as
;           file                /domain/named.root
;           on server           FTP.RS.INTERNIC.NET
;       -OR- under Gopher at    RS.INTERNIC.NET
;           under menu          InterNIC Registration Services (NSI)
;              submenu          InterNIC Registration Archives
;           file                named.root
;
;       last update:    Nov 8, 1995
;       related version of root zone:   1995110800
;
;
; formerly NS.INTERNIC.NET
;
.                       3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.                       3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000      A     128.9.0.107
;
; formerly C.PSI.NET
;
.                       3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.     3600000      A     192.33.4.12
;
; formerly TERP.UMD.EDU
;
.                       3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.     3600000      A     128.8.10.90
;
; formerly NS.NASA.GOV
;
.                       3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.     3600000      A     192.203.230.10
;
; formerly NS.ISC.ORG
;
.                       3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.     3600000      A     192.5.5.241
;
; formerly NS.NIC.DDN.MIL
;
.                       3600000      NS    G.ROOT-SERVERS.NET.
```

```
G.ROOT-SERVERS.NET.      3600000      A     192.112.36.4
;
; formerly AOS.ARL.ARMY.MIL
;
.                        3600000      NS    H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.      3600000      A     128.63.2.53
;
; formerly NIC.NORDU.NET
;
.                        3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.      3600000      A     192.36.148.17
; End of File
```

2. No caso normal, é sempre o servidor de nomes cujo endereço é configurado no cliente que se responsabiliza por perguntar aos outros servidores de nome até finalmente chegar ao número IP. Na verdade, existem dois tipos de requisições a um servidor de nomes: as recursivas e as iterativas (ou não recursivas). O cliente ("resolver") sempre faz requisições recursivas para o servidor de nomes que se vê então obrigado a responder com o número IP desejado. Por sua vez, a maioria dos servidores de nome faz requisições iterativas aos servidores raízes e aos subsequentes o que significa que ele só recebe ponteiros para servidores "mais próximos" da autoridade no IP que ele busca. Assim, ele faz o trabalho difícil fazendo requisições iterativas para *vários* servidores de nome.

3. Os vários servidores de nomes estão espalhados pela Internet não havendo nenhuma garantia que, a medida que se aproximam do nome desejado, os servidores fiquem mais próximos fisicamente do computador requisitado. Não existe relação entre a rota seguida por pacotes IP e a sequência de computadores consultados para se associar um nome a um número IP.

4. Uma consequência desagradável do ítem anterior é que, para se descobrir o endereço IP de uma máquina que esteja fisicamente próxima a máquina requisitante (por exemplo, numa mesma universidade) é muitas vezes necessário percorrer-se o mundo todo. Se as conexões com servidores distantes não estiverem disponíveis, é as vezes impossível se descobrir o número IP de uma máquina que está localizada num prédio do outro lado da rua com a qual se tem conectividade total.

5. Se por algum motivo, um "servidor raiz" não responder a uma requisição do servidor de nomes, este tentará perguntar a um outro servidor raiz. Ele só desistirá se nenhum servidor raiz estiver disponível.

6. Para aumentar a eficiência do processo, o servidor de nomes armazena os resultados de suas requisições numa estrutura temporária chamada "cache". Assim, quando uma nova requisição sobre o mesmo domínio lhe for efetuada, ele não precisará percorrer os diversos servidores novamente.

8.1.2 O DNS e o Sistema Operacional Unix

O cliente DNS, ou seja, o resolvedor, é implementado por um conjunto de rotinas que são chamadas por aplicações que necessitam da resolução de nomes para seu correto funcionamento. Essas rotinas se utilizam da configuração de um arquivo, normalmente localizado em */etc* chamado *resolv.conf*. Um exemplo de uma resolv.conf típico pode ser visto a seguir:

```
domain  ifqsc.sc.usp.br.
search ifqsc.sc.usp.br totolocal.sc.usp.br
nameserver      143.107.228.51
```

Note a utilização do diretivo "search" que indica o que deve ser adicionado ao nome para se ter o nome completo do "host".

O servidor do DNS, ou seja, o servidor de nomes, é normalmente um programa chamado *named* de "name daemon". Esse programa se utiliza de um arquivo de configuração que é um dos parâmetros quando *named* é executado. Por exemplo, para se iniciar o servidor de nomes, chama-se:

/usr/sbin/named -b /etc/named.boot

Um exemplo de arquivo named.boot é:

```
;       $Id: named.boot,v 1.3 1995/03/23 08:43:02 rgrimes Exp $
;       From: @(#)named.boot    5.1 (Berkeley) 6/30/90

; boot file for secondary name server
; Note that there should be one primary entry for each SOA record.

; example sortlist config:
; sortlist 128.3.0.0

directory       /etc/namedb

; type    domain                source host/file                backup file

cache     .                                                     named.root
primary   0.0.127.IN-ADDR.ARPA  localhost.rev

; example secondary server config:
; secondary Berkeley.EDU        128.32.130.11 128.32.133.1      ucbhosts.bak
;secondary 32.128.IN-ADDR.ARPA 128.32.130.11 128.32.133.1 ucbhosts.rev.bak

; example primary server config:
; primary  Berkeley.EDU         ucbhosts
; primary  32.128.IN-ADDR.ARPA  ucbhosts.rev

primary ifqsc.sc.usp.br.        db.ifqsc
primary 228.107.143.IN-ADDR.ARPA db.143.107.228
primary totolocal.sc.usp.br.    db.toto_local
primary 1.1.10.IN-ADDR.ARPA db.10.1.1
```

8.1.3 nslookup

Para se entender e depurar o funcionamento de um servidor de nomes, uma ferramenta bastante interessante é o programa *nslookup*. Esse programa executa requisições de resolução de nomes interativamente.

Considere a transcrição abaixo que ilustra o potencial do *nslookup*:

```
toto@ixtoto:ttyp3 ~ (166)> /usr/sbin/nslookup
```

```
Default Server:  ixtoto.ifqsc.sc.usp.br
Address:  143.107.228.51

> r7.cs.man.ac.uk.
Server:  ixtoto.ifqsc.sc.usp.br
Address:  143.107.228.51

Name:    r7.cs.man.ac.uk
Address:  130.88.192.72

> r7.cs.man.ac.uk.
Server:  ixtoto.ifqsc.sc.usp.br
Address:  143.107.228.51

Non-authoritative answer:
Name:    r7.cs.man.ac.uk
Address:  130.88.192.72

> set norecurse
> www.ford.com.
Server:  ixtoto.ifqsc.sc.usp.br
Address:  143.107.228.51

Name:    www.ford.com
Served by:
- A.ROOT-SERVERS.NET


- B.ROOT-SERVERS.NET


- C.ROOT-SERVERS.NET


- D.ROOT-SERVERS.NET


- E.ROOT-SERVERS.NET


- F.ROOT-SERVERS.NET


- G.ROOT-SERVERS.NET
          192.112.36.4

- H.ROOT-SERVERS.NET


- I.ROOT-SERVERS.NET

> server G.ROOT-SERVERS.NET
Default Server:  G.ROOT-SERVERS.NET
Address:  192.112.36.4
```

```
> www.ford.com.
Server:  G.ROOT-SERVERS.NET
Address:  192.112.36.4

Name:    www.ford.com
Served by:
- DNS003.FORD.com
          198.111.80.21
          FORD.com
- DNS.MERIT.NET
          35.1.1.42
          FORD.com

> server DNS003.FORD.com.
Default Server:  DNS003.FORD.com
Address:  198.111.80.21

> www.ford.com.
Server:  DNS003.FORD.com
Address:  198.111.80.21

Name:    www.ford.com
Address:  198.108.89.236

> set recurse
> set debug
> set d2
> www.att.com.
Server:  DNS003.FORD.com
Address:  198.111.80.21

;; res_mkquery(0, www.att.com, 1, 1)
------------
SendRequest(), len 29
    HEADER:
        opcode = QUERY, id = 38577, rcode = NOERROR
        header flags:  query, want recursion
        questions = 1,  answers = 0,  authority records = 0,  additional = 0

    QUESTIONS:
        www.att.com, type = A, class = IN

------------
------------
Got answer (160 bytes):
    HEADER:
        opcode = QUERY, id = 38577, rcode = NOERROR
        header flags:  response, auth. answer, want recursion, recursion avail.
        questions = 1,  answers = 1,  authority records = 3,  additional = 3

    QUESTIONS:
        www.att.com, type = A, class = IN
    ANSWERS:
    ->  www.att.com
        type = A, class = IN, dlen = 4
```

```
        internet address = 198.152.185.65
        ttl = 1200 (20 mins)
    AUTHORITY RECORDS:
    ->  att.com
        type = NS, class = IN, dlen = 8
        nameserver = kcgw1.att.com
        ttl = 86400 (1 day)
    ->  att.com
        type = NS, class = IN, dlen = 8
        nameserver = cagw1.att.com
        ttl = 86400 (1 day)
    ->  att.com
        type = NS, class = IN, dlen = 8
        nameserver = hogw1.att.com
        ttl = 86400 (1 day)
    ADDITIONAL RECORDS:
    ->  kcgw1.att.com
        type = A, class = IN, dlen = 4
        internet address = 192.128.133.1
        ttl = 86400 (1 day)
    ->  cagw1.att.com
        type = A, class = IN, dlen = 4
        internet address = 192.128.52.89
        ttl = 86400 (1 day)
    ->  hogw1.att.com
        type = A, class = IN, dlen = 4
        internet address = 204.179.186.33
        ttl = 86400 (1 day)

------------
Name:    www.att.com
Address:  198.152.185.65
> exit
toto@ixtoto:ttyp3 ~ (167)>
```

Capítulo 9

FDDI, FDDI II, 100 Base T

9.1 FDDI

FDDI é a sigla utilizada para definir uma rede de alta velocidade em fibra ótica que se utiliza de uma topologia em anel. FDDI vem de *Fibre Distributed Data Interconnect*. Posteriormente, para acomodar melhor os serviços de multimídia, foi proposta uma extensão ao protocolo que ficou conhecida como FDDI II.

9.1.1 O Meio Físico

O FDDI trabalha numa velocidade de 100 Mbps sobre fibra ótica. Fibras óticas são mais tolerantes a ruído que cabos de cobre e permitem velocidades de comunicação bem maiores. Atualmente, o custo de fibras já é menor do que o de cobre o que leva à previsão de que num futuro próximo o meio fibra será o mais comum para redes de computadores.

As seguintes fibras podem ser utilizadas no FDDI: multimodo 62,5/123 μm e multimodo 50/125 μm. O comprimento de onda utilizado na transmissão é o de 1300 nm.

A codificação de Manchester, que é aquela utilizada também na Ethernet e no Token Ring, estabelece que para se representar um bit, é preciso duas transições. No Ethernet, isso implica que, para termos um fluxo de 10 Mbps, é preciso um relógio de 20 MHz. Se fosse utilizado o mesmo esquema no FDDI, seria preciso um relógio de 200MHz o que é complicado e caro. Ao invés disso, o FDDI utiliza uma codificação conhecida como 4B/5B (4 bits para cada 5 baud). Assim, para um relógio de 125 MHz, tem-se os desejados 100 Mbps. A cada 8 ns, é verificado se o estado ótico se alterou (se a luz estava ligada e agora está desligada ou vice-versa). Em caso afirmativo, considera-se que o bit é 1. Se o estado da luz não se alterou (continua apagada ou continua acesa), considera-se que o bit é 0. Além do FDDI, o 100 Base T e o ATM 100 Mbps se utilizam da codificação 4B/5B.

Na codificação 4B/5B, são utizados 16 "símbolos de dados" (16 seriam 4 bits — 4B) derivados de 5 bits (5B). Assim, tem-se mais 16 "símbolos" que são divididos em símbolos de controle e símbolos de violação. Os símbolos de dados são escolhidos de forma que nunca ocorra a recepção de 4 ou mais bits zeros na cadeia de bits de dados. Para exemplificar, o símbolo de

dados "D" (binário 1101) é representado por 11011. O símbolo de controle J é 11000, o K é 10001, o T é 01101, o Q é 00000, o I é 11111 etc.

Os "frames" e a "ficha" do FDDI são especificados em termos de "campos" designados por siglas de 2, 3 ou 4 letras. Por exemplo, a ficha é especificada como um conjunto sequencial de 4 campos: PA, SD, FC, ED. PA é o preâmbulo que envolve a transmissão de uma série de símbolos "Idle" (I) com a finalidade de comparar os relógios das várias estações. SD ("starting delimiter") indica o início da ficha e é composto por um símbolo J seguido de um K. Esses símbolos não são utilizados em nenhuma outra situação. FC ("Frame Control") indica o tipo da ficha e é composto por dois símbolos: se os símbolos forem 80, trata-se de uma ficha não restrita e se forem C0, de uma ficha restrita. Finalmente ED ("ending delimiter") marca o final da ficha e é composto por dois símbolos T. O formato de um "frame" no FDDI é a sequência de campos: PA, SD, FC, DA, SA, INFO, FCS, ED, FS. PA e SD tem o mesmo significado já visto para fichas. FC indica qual o tipo de informação que será transmitido no campo INFO. Alguns valores comuns são: C2 e C3 para "frame" MAC, 40 para "frame" nulo, 41 e 4F para "frame" de gerenciamento de estação (SMT — Station Management), etc. DA e SA são respectivamente os endereços destino e origem. Esses campos são compostos por 12 símbolos cada. O campo INFO carrega os dados propriamente ditos e podem transportar de 0 a 4478 bytes (lembre-se: dois símbolos são necessários para cada byte). O FCS ("Frame Control Sequence") é utilizado para que a estação receptora verifique a integridade dos dados recebidos. O campo ED é constituído de um único símbolo T que indica final de "frame". O campo FS ("Frame Status") é composto por 3 símbolos que indicam o estado do "frame". Por exemplo, um dos estados é o de erro quando algum erro de transmissão é detectado.

A topologia do FDDI é em anel com redundância (na forma de um segundo anel) de tal forma que a rede continua em funcionamento em certos casos de falha de computadores ou mesmo da conexão da fibra. Uma falha única de conexão é automaticamente corrigida pela reconfiguração do anel (com a utilização do segundo anel) sem nenhuma perda de funcionalidade. Falhas subsequentes também serão corrigidas mas podem provocar a criação de subredes desconexas. O anel FDDI pode ter até 100 km de fibra e até 1000 estações. A topologia do FDDI com seus dois anéis pode ser vista na figura 9.1.

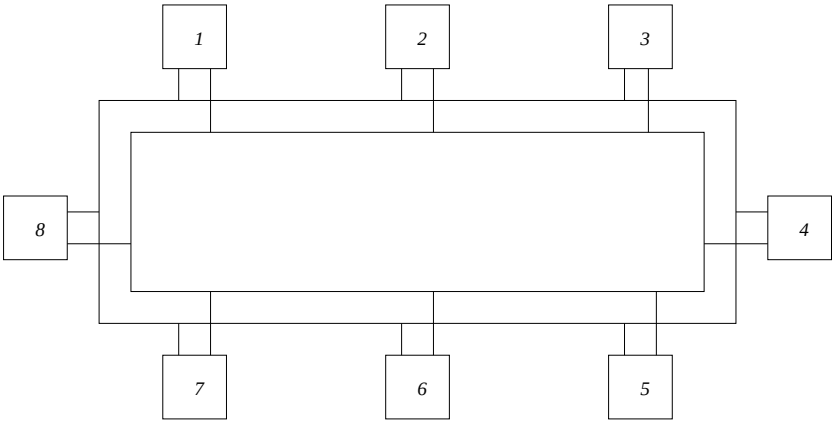


Figura 9.1: FDDI com 8 computadores conectados

9.1.2 Funcionamento

No FDDI, existe uma ficha ("token") que constantemente circula o anel. Quando uma estação quer transmitir, ela "captura a ficha" e faz a sua transmissão. Toda estação sempre pega o pacote

e verifica se é para ela. Se o pacote não for seu, ela passa o pacote para a próxima estação; em caso contrário, ela mantém uma cópia do pacote e o retransmite. Depois da transmissão de um pacote, a estação transmissora passa a ficha circulante e assim concede a outras estações o direito de transmitir. Note que, enquanto tem a ficha, a estação não passa para frente os pacotes que estejam vindo pelo anel.

Quando um dos computadores falha, os outros percebem e, automaticamente, desconectam o computador quebrado da rede. Na figura 9.2 vemos a mesma topologia FDDI já apresentada depois de uma falha no computador 4.

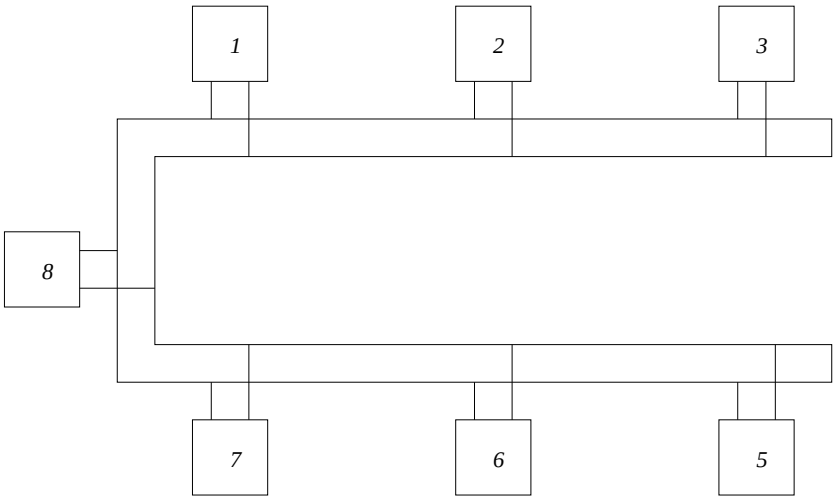


Figura 9.2: FDDI com falha em um computador

9.1.3 Frame do FDDI

O FDDI apresenta um pacote consideravelmente maior que o da Ethernet. Ele pode chegar a até 4500 bytes. O tamanho dos campos de um pacote FDDI é medido em "símbolos" de 4 bits cada um. Assim como o pacote Ethernet, ele leva o endereço destino e origem da trasmissão. O endereço FDDI pode ser de 4 ou 12 símbolos. Uma diferença importante entre o pacote FDDI e o Ethernet é que o primeiro contém informação de roteamento (até 60 símbolos) que facilitam a interconexão de mais de um anel FDDI. O campo de roteamento permite que uma estação especifique que um pacote deve primeiro ir para um determinado ponto para depois seguir para uma estação destino num outro anel.

9.1.4 Utilização e Aspectos de Mercado

O FDDI é talvez a primeira das tecnologias de redes de alta velocidade. Ela começou a ser especificada no início da década de 80 e no início da década de 90 já era encontrada nas principais universidades americanas e em algumas empresas grandes dos EUA. Sua principal utilização no início era a interconexão entre redes locais departamentais em grandes campi ou em empresas com instalações multi-prediais. Apesar de ser possível projetarem-se redes com cobertura municipal com a tecnologia FDDI (na verdade, surgiu até a sigla MAN — Metropolitan Area Network — para se designar tal rede), isso foi raramente feito devido a dificuldade política de instalar fibras nas cidades.

Um grande problema para a aceitação e popularidade do FDDI, se deve ao fato das interfaces para FDDI serem caras. Também, o surgimento de redes chaveadas prejudicou muito a aceitação do FDDI que é uma rede baseada em um meio compartilhado. Finalmente, o surgimento do 100 Base T e 100 VG, com suas facilidades de migração para quem já possui Ethernet também ajudou a diminuir o interesse pelo FDDI.

Apesar dos problemas apontados no parágrafo anterior, o FDDI ainda é bastante utilizado na interconexão de redes locais. No campus de São Carlos da USP, está disponível, já há alguns anos, uma rede FDDI que interliga os vários institutos e departamentos. Apesar do tráfego entre as unidades da USP não ser muito significativo, a rede FDDI garante que não exista estrangulamento interno ao campus no acesso às linhas da Internet que partem do CISC.

9.2 FDDI II

As redes FDDI apresentam uma largura de banda bastante superior a da Ethernet. Também, o aproveitamento dessa largura de banda é bem melhor já que não existem perdas devido a colisões. Mesmo assim porém, o FDDI ainda supõe um meio compartilhado o que torna difícil a boa utilização da rede para aplicações de multimídia que exigem boa largura de banda e latência fixa. De fato, não é possível garantir-se uma latência constante no FDDI.

Para resolver esse problema, foi proposto em 1994 um novo protocolo derivado do FDDI: o FDDI II. O FDDI II é bastante similar ao FDDI com uma única diferença: ele engloba o conceito de chaveamento de circuito que garante latência baixa e constante. A largura de banda de 100 Mbps é dividida em 16 circuitos de 6.144 Mbps. Cada um desses circuitos pode ser subdividido em 96 canais de 64 kbps. Circuitos estabelecidos entre duas estações para transmitir voz, por exemplo, sofrem de uma latência constante de 125 μs entre nós independente do tráfego da rede. No FDDI comum, esse atraso pode chegar a vários milisegundos.

Apesar da inovação, o FDDI II não foi bem recebido pela indústria e poucas são as empresas que implementam interfaces FDDI II.

9.3 100 Base T

9.3.1 Introdução

Talvez a tecnologia de rede local mais bem sucedida tenha sido a Ethernet, criada na década de 70 e hoje encontrada em todas as partes do mundo. A despeito de seu sucesso, é claro hoje que a Ethernet (ou IEEE 802.3) tem limitações sérias que não podem mais ser aceitas em ambientes onde um bom desempenho seja fundamental.

Alguns problemas da Ethernet são:

1. A largura de banda de 10 Mbps já pode ser considerada pequena para os avanços tecnológicos da área na década de 90.
2. A largura de banda disponível deve ser dividida por todas as estações que queiram transmitir dados num determinado instante o que faz com que cada uma tenha uma largura de

banda efetiva pequena quando existirem muitas estações conectadas.

3. Em situações onde a carga da rede é "pesada", a largura de banda disponível cai muito (não é incomum cair abaixo de 3 Mbps). Em redes carregadas (como a do IFSC), o usuário pode esperar segundos para conseguir enviar seus dados.
4. A distribuição de largura de banda pelas estações não é previsível e pode ser bastante injusta. Assim, é difícil se utilizar uma rede Ethernet em transmissões que envolvam multimídia.

Para tentar aliviar alguns desses problemas, o IEEE definiu uma norma que ficou conhecida como "fast ethernet" ou 100 Base T.

9.3.2 Características do 100 Base T

Três são os tipos de meios físicos possíveis no 100 Base T (note a ausência do cabo coaxial): par trançado não blindado (UTP) categoria 5 (2 pares de cobre) especificado em 100 Base TX, 4 pares trançados categoria 3 especificados em 100 Base T4 e a fibra ótica, em 100 Base FX. Sem o cabo coaxial, fica impossibilitada a topologia de barramento e torna-se essencial a presença de um hub ou de uma chave.

A norma é bastante parecida com a do 10 Base T. A diferença fundamental é que o tempo de um bit é diminuido em 10 vezes com o consequente aumento de 10 vezes da largura de banda. O formato do pacote é idêntico ao da ethernet original o que implica que o usuário pode se utilizar de suas aplicações atuais sem modificações. Outras características da ethernet original são mantidas o que leva, assim como na ethernet, a problemas de baixa eficiência com carga alta, dificuldade de transmissão de multimídia, etc. A grande vantagem do 100 Base T é que ele dá ao projetista uma forma de aumentar a velocidade de uma rede 10 Base T com relativamente poucas modificações na topologia, no cabeamento e, principalmente, nos aplicativos (software) que se utilizam da rede. É claro que é necessária a compra de interfaces e hubs (ou chaves) novos que implementem o novo protocolo.

Duas outras características facilitam a migração de 10 Base T para 100 Base T: a existência de placas com as duas interfaces por um preço um pouco maior que o da interface comum 10 Base T e uma nova característica (que infelizmente é opcional) da norma conhecida como "Nway". A existência de placas com as duas interfaces permite que o gerente da rede compre gradativamente tais placas e as use, a princípio, como interfaces comuns 10 Base T. Quando ele tiver todas as interfaces de uma sub-rede com interface dupla, basta mudar o hub ou chave para uma com suporte 100 Base T que a rede passa a funcionar a 100 Mbps. Se os equipamentos 100 Base T implementarem o Nway, a migração torna-se ainda mais simples. De fato, pelo Nway, os dispositivos de 100 Base T passam a funcionar como 10 Base T se pelo menos um outro dispositivo da mesma rede funcionar na velocidade mais baixa. Quando todos os dispositivos forem capazes de se comunicar a 100 Mbps, a rede passa automaticamente para a velocidade mais alta.

A utilização de chaves 100 Base T pode aliviar o problema que a tecnologia tem de ainda ser baseada em um meio compartilhado. Se uma interface apenas for alocada por porta da chave, elimina-se completamente o problema de colisão. Também, a norma permite que, nessa situação, se utilize a configuração "full duplex" que permite que dados sejam passados em ambas as direções simultaneamente, efetivamente dobrando a largura de banda. Mesmo com tudo isso porém, ainda não se pode garantir que tráfego do tipo multimídia seja convenientemente transmitido já que não existe garantia que a chave será "justa" na transmissão de pacotes.

9.3.3 Utilização e Aspectos de Mercado

Apesar de ser um avanço em relação ao ethernet convencional, não é claro que o 100 Base T seja ou será um sucesso de mercado.

Como principais competidores, ele enfrenta:

1. FDDI: o FDDI é uma tecnologia mais antiga e mais madura apesar de mais cara. A alocação de largura de banda no FDDI é intrinsicamente mais justa e mais eficiente o que faz com que muitos projetistas o prefiram ao 100 Base T.
2. 100 VG AnyLAN: talvez o maior competidor do 100 Base T, o 100 VG tem como objetivo o mesmo mercado do 100 Base T, qual seja, o das redes locais de alto desempenho. Maiores detalhes bem como vantagens e desvantagens do 100VG serão vistos na próxima apostila.
3. Redes Chaveadas: para muitos gerentes de sistema, mais simples do que adotar um novo padrão, por mais que ele seja parecido com o antigo, é introduzir um conjunto de chaves na topologia atual. Desta forma é possível reduzir as colisões na rede existente com grande ganho em largura de banda efetiva sem a introdução de qualquer interface ou hub novos. Maiores detalhes desta solução serão vistos posteriormente.
4. ATM ou Fibre Channel: muitos projetistas e gerentes de rede acreditam que, se for necessário se mudar a rede atual, deve-se ir para um protocolo ou filosofia que permitam expansões consideráveis na largura de banda efetiva. Está claro que isto não é possível com o 100 Base T. Com o ATM e Fibre Channel, expansões de centenas de vezes na largura de banda efetiva são possíveis.

A grande vantagem do 100 Base T em relação a todas essas soluções é que ele apresenta um considerável ganho de velocidade com mínimo de transformação da rede ethernet já existente. Também, por ser uma solução barata, pode ser uma escolha interessante para quem está começando agora com sua rede local e quer algo mais do que a ethernet tradicional.

Capítulo 10

Introdução ao Frame Relay

10.1 Introdução

Esta apostila apresenta uma breve introdução aos aspectos técnicos do protocolo Frame Relay. A apresentação é fortemente baseada no livro de Daniel Minoli "Enterprise Networking".

10.2 Aspectos Técnicos: O Frame Relay

Como já foi visto na apostila anterior, o Frame Relay pode ser encarado como uma simplificação do protocolo X.25. De fato, o Frame Relay perde a capacidade de correção de erros na camada dois (e também a capacidade de retransmissão), e toda a funcionalidade da camada 3.

A figura 10.1 apresenta uma comparação dos dois protocolos através de um diagrama de estado.

Originalmente, o Frame Relay foi projetado para suportar Permanent Virtual Circuit (PVC) e Switched Virtual Circuit (SVC), mas, atualmente, somente o PVC está sendo implementado. Assim, não existe mecanismo para estabelecimento de conexão, sendo estas predefinidas pelo gerente da rede. O roteamento não pode ser mudado dinamicamente. A consequência é uma conexão rápida, com baixa latência, ideal para conexões entre LANs.

O protocolo Frame Relay permite a multiplexação simultânea de 1024 canais sobre uma única ligação física.

Assim como no caso do X.25, o Frame Relay especifica a UNI (user network interface) que conecta o equipamento do usuário ao nó processador Frame Relay. Três são as hipóteses em que se baseia o Frame Relay segundo Minoli:

1. Equipamentos inteligentes do lado do usuário para a execução do protocolo.

2. Linhas de transmissão com melhor qualidade (menor ruído e menor possibilidade de erros).

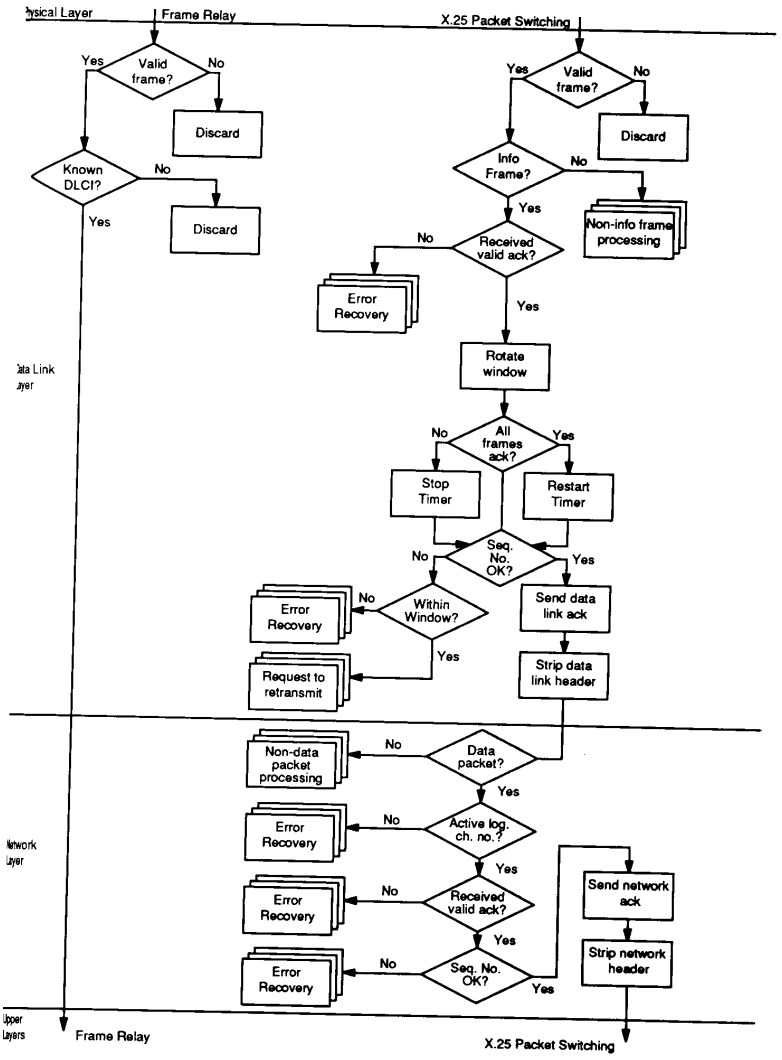


Figure 11.5 Comparison of frame relay protocol state diagram with the one for X.25.

Figura 10.1: Comparação entre Frame Relay e X.25. Minoli, pg 593

3. Detecção e correção de erros devem ser implementadas em níveis superiores do protocolo.

O nível físico do Frame Relay é derivado do LAP-D (LAP é Link Access Protocol) e recebeu o nome de LAP-F. O LAP-F inclui campos de controle de fluxo, janelas e correção de erros. O Frame Relay adota o que é conhecido como "núcleo" do LAP-F, o LAP-F-core que elimina as janelas e a correção de erros.

10.3 O Problema dos Erros de Transmissão

No Frame Relay, a correção de erros e a retransmissão, quando necessárias, são efetuadas pelos equipamentos das pontas. A rede só consegue detectar os erros e , quando isso ocorre, o pacote é simplesmente descartado. Assim, toda retransmissão é feita pelo gerador do frame e vai até o equipamento receptor. No X.25, cada nó da rede pode retransmitir o pacote no caso de erro de transmissão. São considerados erros: pacotes com dados errados, pacotes perdidos, pacotes duplicados, e pacotes fora da sequência correta.

A idéia do Frame Relay é a seguinte: quando as linhas de transmissão apresentavam alta probabilidade de erro, era economicamente justificável investir-se em equipamentos mais complexos que tivessem a capacidade de correção e retransmissão de pacotes como no caso do X.25. De fato, se a probabilidade de erro for alta, não faz sentido esperar-se que seja feita a retransmissão entre os pontos finais (principalmente no caso de o pacote ter que passar por vários nós intermediários) já que isso poderia introduzir grandes atrasos e faria mal uso da largura de banda disponível. Por outro lado, se a probabilidade de erro for baixa, não se justifica a maior complexidade (com o consequente maior custo) dos equipamentos intermediários. Nesse último caso, o atraso introduzido por cada nó seria inevitável devido a maior complexidade no tratamento do pacote.

Para ilustrar o compromisso entre as duas alternativas, considere o caso de comunicação entre dois pontos unidos por três segmentos de rede. Considere que a probabilidade de transmissão correta em cada link seja s, a mesma para os três links para simplificar a análise. Na tabela a seguir, são dados o atraso esperado e a largura de banda utilizada nos casos de $s = 0.9$, $s = 0.99$ e $s = 0.999$. Note que é utilizado o conceito de largura de banda esperada que dá a largura de banda realmente utilizada na transmissão. Note que esse valor é dado em "unidades de links" e portanto, é sempre maior que 3 (já que cada transmissão envolve pelo menos três links).

Prob	L.B. (por conexão)	L.B. (fim a fim)	Atraso (p.c.)	Atraso (faf)
0.9	3.30674	4.02831	0.66135	0.53711
0.99	3.03027	3.09182	0.60606	0.41224
0.999	3.00300	3.00902	0.60060	0.40120

Tabela 10.1: L. B esperada e atraso nos caso de por conexão e fim a fim. Minoli pg 596

Note que o atraso é sempre menor no caso da retransmissão ser feita entre os pontos finais. Essa é uma grande vantagem do Frame Relay: baixa latência. Por outro lado, a largura de banda necessária é sempre menor no caso da retransmissão por conexão que faz melhor uso dos recursos disponíveis. Porém, quanto menor a probabilidade de erro, menor é a diferença na eficiência de utilização o que torna o uso de Frame Relay bastante conveniente com linhas confiáveis. Note que, no exemplo da tabela (com 3 conexões), se a probabilidade de sucesso da

transmissão for de 0.999, a largura de banda esperada no caso por conexão, 3.00300 é praticamente a mesma do caso fim a fim, 3.00902 (a diferença é de apenas 0.2 %).

Não entraremos aqui no cálculo dos valores da tabela. É fácil ver porém, porque a largura de banda esperada é maior num caso do que no outro. Suponha que um frame ande dois links sem problema mas seja perdido no último link. No caso de correção e retransmissão por conexão, teremos uma largura de banda necessária de 4, enquanto no caso fim a fim, teremos o valor 6.

É importante notar que os cálculos dependem do número de links existentes entre os pontos que se quer conectar. Na verdade, nada se ganha com Frame Relay se os pontos estiverem ambos conectados ao mesmo nó processador Frame Relay.

10.4 Exemplo de Redes Frame Relay

Deve-se lembrar que a idéia do Frame Relay é permitir uma conexão entre todos os locais distantes de uma corporação de forma que cada local tenha somente uma conexão à rede.

A arquitetura de tal conexão pode ser vista na figura 10.2 que mostra quatro redes locais interconectadas através de uma rede Frame Relay.

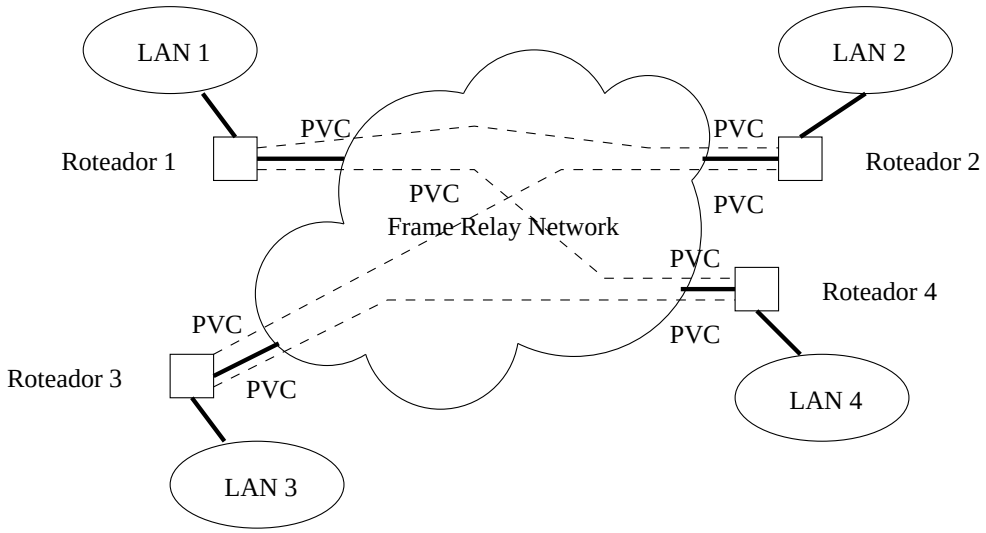


Figura 10.2: Uma rede Frame Relay

Note que só existe **uma** conexão física entre cada local e a rede. Essa conexão liga o roteador local com um nó processador de Frame Relay. Note que, nada impede que dois locais diferentes se conectem a um mesmo nó processador.

Os frames com destinos diferentes são sempre mandados pelo mesmo canal físico mas através de PVCs diferentes.

A rede Frame Relay da figura 10.2 pode ser pública, privada ou híbrida. No caso brasileiro, o primeiro fornecedor de redes Frame Relay é a Embratel, com equipamentos da Northern Telecom. Nesse caso, a rede é dita pública pois todos tem acesso se pagarem as tarifas

cobradas. Internamente à rede, os pacotes podem ser roteados e transmitidos como for mais conveniente (entre os nós processadores). A tendência atual parece ser a utilização de células (ATM) apesar de que frames podem também ser utilizados.

O Frame Relay é um protocolo orientado a conexão. Porém, o PVC utilizado no Frame Relay é configurado na hora da instalação pelo gerente da rede. Não é necessário o estabelecimento da conexão cada vez que se desejar transmitir algo; ela está sempre "ligada".

10.5 Mecanismo de Transmissão

A figura 10.3 mostra a arquitetura de uma conexão entre dois pontos finais através das camadas ISO/OSI.

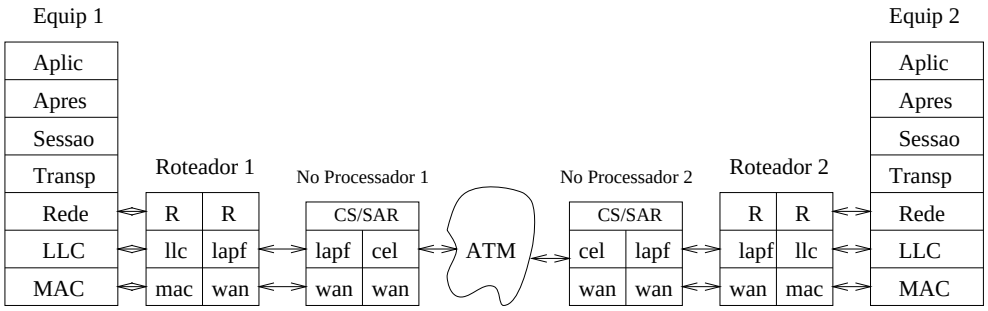


Figura 10.3: Conexão via Frame Relay – Protocolos

Note que a interconexão dos nós processadores Frame Relay pode ser implementada com qualquer protocolo (no caso ATM).

Suponhamos que o equipamento 1 (por exemplo, uma estação de trabalho Unix) queira se comunicar com o equipamento 2 (por exemplo, um PC). Ele descobre, através de sua tabela de roteamento que os dados devem ser enviados para o roteador. Este determina que o equipamento 2 é externo e pode ser alcançado através da rede Frame Relay. Ele então envia os dados através de um determinado PVC que, ele sabe, é o caminho para se chegar ao roteador 2. Para isso, ele determina o **DLCI** (Data Link Connection Identifier) correspondente ao PVC. Não confundir o DLCI com algum tipo de endereço do roteador 2! O DLCI só tem validade local e pode mudar até chegar à outra extremidade.

O roteador manda o frame então ao nó processador 1 com o DLCI correto. Idealmente o frame deve ser do mesmo tamanho do frame da rede local para que se minimizem os atrasos e overheads. O nó 1 transforma o DLCI num **VCI** (já que estamos supondo uma rede ATM) e o frame, agora segmentado em células (daí a necessidade do SAR – segmentation and reassembly layer), é transmitido pela rede ATM até o nó processador 2. O nó 2 transforma o VCI num DLCI conveniente (que pode ser diferente do primeiro DLCI) e envia o frame até o roteador que o entrega ao PC destino.

Para ilustrar o funcionamento do DLCI, considere a figura 10.4

Note como é feito o mapeamento entre DLCI, PVCs e VCIs.

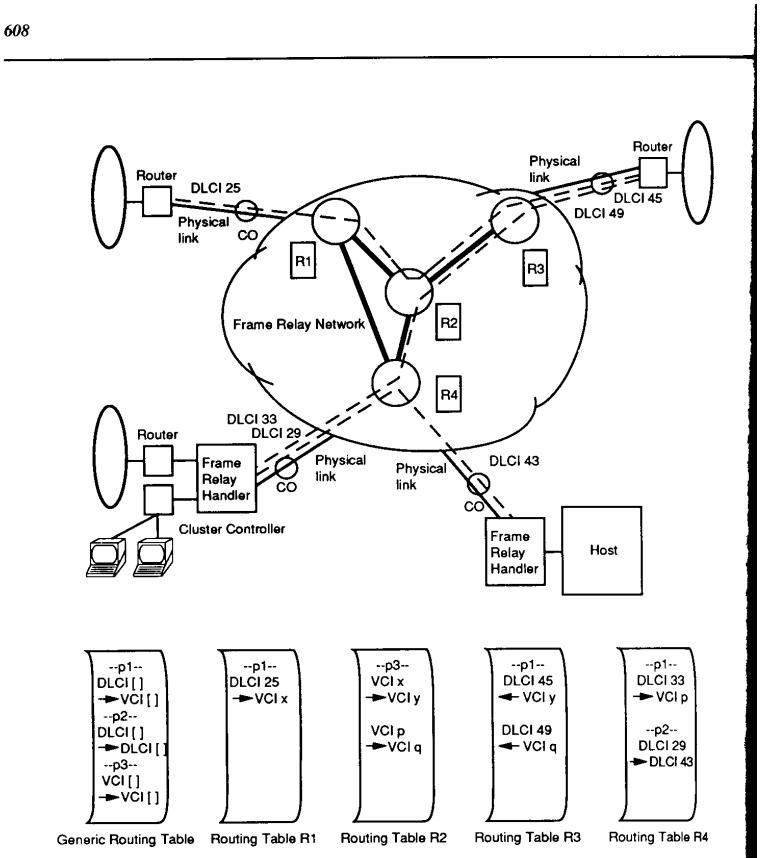


Figure 11.8 Mapping of DLCIs in a frame relay network.

Figura 10.4: O DLCI em redes Frame Relay. Minoli, pg 608

10.6 Benefícios do Frame Relay

Para finalizar vejamos as vantagens conseguidas pela nova tecnologia Frame Relay.

Minoli dá várias vantagens para a rede Frame Relay em ambientes privados (mas muitas valem também para redes públicas). As principais são:

1. **Compartilhamento de Portos:** de fato, frames que se destinam a destinos diferentes tem uma única saída através do roteador frame relay. Esse roteador tem também habilidades de multiplexação estatística fazendo uso eficiente do canal de saída.
2. **Largura de Banda sob Demanda:** a conexão Frame Relay está disponível, em sua velocidade máxima, para cada usuário da conexão. É possível limitar-se o acesso de cada usuário em redes com alto grau de utilização.
3. **Uso eficiente de Largura de Banda:** a utilização do Frame Relay, ao invés dos tradicionais esquemas TDM pode tornar o uso dos canais muito mais eficiente. Fatores de 20% foram detectados em algumas redes.
4. **Alto "Throuput" e Baixo Atraso:** Como já foi dito, o atraso no Frame Relay é muito menor do que o do X.25 e a largura de banda total está disponível para cada usuário.
5. **Facilidade de Expansão:** Adicionar um ponto a uma rede Frame Relay é muito simples bastando adquirir-se um novo roteador com interface Frame Relay e conectá-lo à rede. Só é necessário uma reconfiguração de software para se especificar as novas conexões lógicas.
6. **Transição fácil de redes atuais:** Roteadores existentes normalmente podem ser transformados em roteadores Frame Relay com pequeno custo adicional.
7. **Boa convivência com LANs:** A camada de dados do Frame Relay não é muito diferente da camada de dados das LANs existentes o que torna simples a convivência entre elas.
8. **Facilidade de Gerenciamento:** com a redução de linhas externas, o gerenciamento da WAN fica bastante simplificado. Esse é um fator importante pois acredita-se que até 50% do custo de uma rede atualmente vem do seu gerenciamento.
9. **Padrões:** O Frame Relay é todo baseado em padrões estabelecidos internacionalmente e tem tido aceitação tanto nos EUA quanto na Europa e Japão.
10. **Vendedores:** existe atualmente um grande número de fabricantes de equipamentos para Frame Relay.
11. **Redução de Custos:** o Frame Relay reduz consideravelmente os custos de comunicação em relação a maioria das tecnologias mais antigas de projeto de WANs

Capítulo 11

Introdução ao ISDN e Detalhes de Operação do Frame Relay

11.1 Introdução

O ISDN (*Integrated Services Digital Network*) surgiu na década de 80 com o objetivo de integrar diferentes formas de comunicação. Com o advento da transmissão digital, tornou-se cada vez mais viável a substituição de linhas analógicas de voz, as populares linhas telefônicas discadas, também conhecidas por *POTS – Plain Old Telephone Service*, por linhas digitais de melhor qualidade. Como a maioria dos serviços de comunicação de dados já era baseada em tecnologia digital há bastante tempo, surge a idéia de unificar todos esses serviços num serviço único com redução de custos em equipamentos, operação e manutenção dos serviços de comunicação.

Essa unificação parecia ser o futuro das telecomunicações na década de 80, principalmente do ponto de vista empresarial pois a maioria das firmas tinha uma variedade de serviços de comunicação tais como: diversas linhas de voz, algumas vezes conectadas através de PABX, linhas privativas para comunicação de dados, linhas para o serviço de TELEX, linhas de comunicação de dados via redes de pacotes, etc. Essa variedade de serviços aumentava consideravelmente os custos de gerência e manutenção além, é claro, do próprio custo de comunicação.

Surge então o ISDN com objetivo de unificar todos esses serviços reduzindo o número de interfaces, protocolos, conectores e equipamentos de comunicação em geral. A qualidade da comunicação também melhora já que, agora, não existem mais conexões analógicas como no antigo POTS.

Apesar de ser uma idéia aparentemente boa, o ISDN foi um fracasso comercial, nitidamente nos Estados Unidos, onde ele foi ferozmente rejeitado. É difícil entender os motivos de tal rejeição: alguns críticos dizem que o problema é que o ISDN não integrava serviços de imagem o que seria essencial para sua aceitação no mercado; outros diziam que o custo da migração para o ISDN não era justificado já que ele não trazia nenhuma funcionalidade extra; outros ainda diziam que o mercado não estava preparado para o ISDN.

De qualquer forma, o estudo do ISDN é importante por dois motivos:

- **Aceitação parcial do protocolo no mercado de acesso Internet**: apesar do atraso de vários anos, finalmente o ISDN está sendo fornecido pelas grandes companhias telefônicas dos EUA principalmente para usuários que desejam acesso mais rápido do que aquele conseguido por modems. De fato, com o ISDN, é possível conectar-se a 128 kbps à Internet.

- **Importância do ISDN para o B-ISDN**: O ISDN, agora conhecido como "narrowband" ISDN é a base para o B-ISDN ("broadband") que por sua vez é o modelo adotado pelo ATM. Como o ATM pode ser considerado o protocolo mais importante de redes no presente, o entendimento do ISDN acaba sendo necessário.

11.2 ISDN

11.2.1 Características Técnicas

A "pilha" ISDN implementa as 3 primeiras camadas do modelo de referência OSI (com 7 camadas).

No nível físico (camada 1 OSI), dois protocolos são possíveis no ISDN. Eles são explicitados nas recomendações I.430 e I.431 que fazem referência a documentos da série Q do ITU-T. O I.430 refere-se à *taxa básica* ("Basic Rate") enquanto o I.431 refere-se à *taxa primária* ("Primary Rate") que serão explicadas abaixo. No nível de dados, a especificação é a I.440/I.441 conhecida como LPAD. Na camada de redes (camada 3), o documento é o I.450/I.451 que se referem ao Q.931 da ITU-T.

O equipamento básico do usuário que se conecta ao ISDN é chamado de **TE1**, de "terminal equipment". O TE1 se conecta a um nó ISDN através de 4 fios (2 pares) por onde passam, multiplexados, 3 canais digitais: 2 canais chamados **B** de "bearer", e um canal chamado **D**, ou "delta". Esses 3 canais são conhecidos como **2B + D**. O 2B + D constitui o que é conhecido como **BRI** de "Basic Rate Interface".

Os canais B operam em velocidade de 64 Kbps (apesar de que, nos EUA, muitas vezes eles são de 56 Kbps. Nesses canais passam as informações do usuário (voz, dado e até mesmo imagens de baixa qualidade). O canal D, de 16 Kbps, é usado para identificação do usuário, estabelecimento de conexão e controle. Em algumas aplicações, o canal D pode ser usado para transmitir dados digitais em redes de pacotes (estilo RENPAC).

Além do BRI, o ISDN prevê também o **PRI** ou "Primary Rate Interface" que é composto por 23 canais B e um canal D nos EUA e Japão o que equivale a uma conexão T1. Na Europa, o PRI é composto por 30 canais B e 1 canal D transportados num canal E1.

Além do TE1, que é um equipamento que se conecta diretamente ao ISDN, é definido o TA (de "terminal adapter") que pode conectar à rede ISDN, um equipamento que não esteja preparado para isso. Esse tipo de equipamento recebe o nome de TE2 na nomenclatura do ISDN. A conexão entre o TE2 e o TA é feita, por exemplo, através de uma conexão RS-232. Outro equipamento é o NT1 (de "network termination") que transforma os 4 fios do ISDN em apenas um par trançado convencional de linha telefônica e faz a conexão da casa ou escritório do usuário até a companhia telefônica. Finalmente, existe o NT2 que é um equipamento capaz de multiplexar vários canais B e transformá-los, por exemplo, num acesso PRI. A figura 11.1 mostra uma típica conexão ISDN.

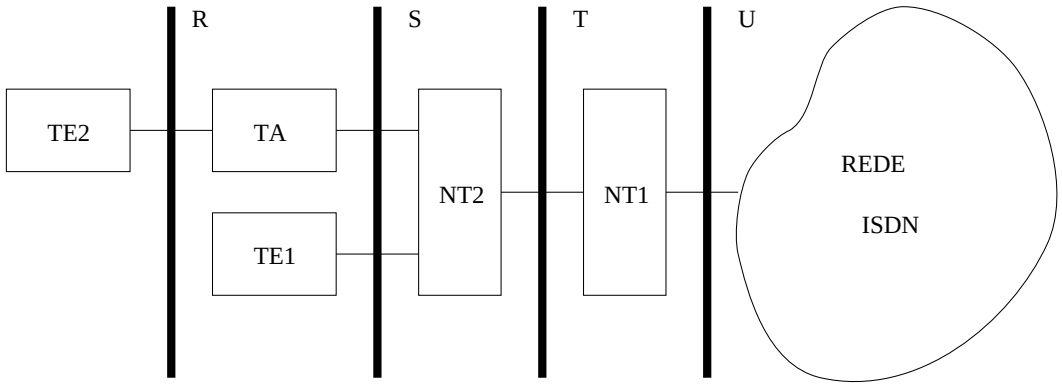


Figura 11.1: Uma conexão ISDN

Note que as interfaces entre os equipamentos recebem nomes especiais: R é a conexão não ISDN (RS-232 e V.34) que conecta PCs, FAXs etc; as interfaces S/T são a 4 fios enquanto a interface U é a 2 fios.

Além de BRI e PRI, existem canais intermediários: H0 (384 Kbps), H11 (1536 Kbps) etc.

No nível de dados (camada 2), o ISDN se utiliza do protocolo LAPD ("Link Access Protocol on the D-channel"). O frame LAPD é baseado no LAPB usado no X.25 que por sua vez é similar ao HDLC. Ele é usado no canal D do ISDN.

O frame LAPD pode ser visto na figura 11.2.

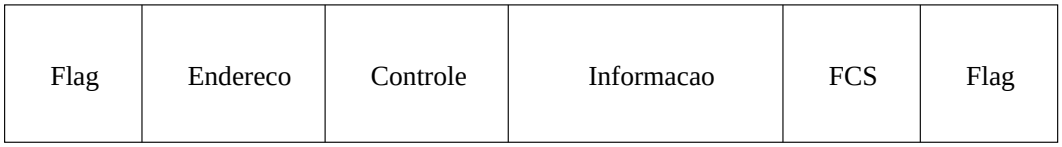


Figura 11.2: O pacote LAPD

O campo de flag é composto por um byte que é sempre 01111110. Ele delimita o pacote marcando seu início e fim. O campo de endereçamento é composto normalmente por 2 bytes. O primeiro flag de cada byte é o indicador de fim de endereço (para permitir mais do que 2 bytes de endereçamento). O segundo bit do primeiro byte é um indicador de comando ou resposta. A interpretação desse bit é oposta entre o usuário e a rede: o usuário manda comandos com esse bit em 0 e responde a comandos com o bit 1. A rede faz exatamente o inverso. O restante dos bits é reservado para os campos *SAPI* e *TEI* que, conjuntamente podem ser considerados como um *data link connection identifier*, DLCI, tal qual no frame relay (que, por sinal, usa um protocolo muito parecido com o LAPD). O campo de FCS ("frame check sequence") é utilizado para se detectar se o dado foi transmitido corretamente.

A camada de redes do ISDN é muito similar àquela do frame relay, que será estudada posteriormente.

11.2.2 Situação no Mercado

Como dito na introdução, o ISDN começa agora a ser utilizado nos EUA como um meio de acesso relativamente rápido e barato à Internet. Para ilustrar, considere os seguintes preços praticados atualmente nos EUA.

Uma companhia, a TELALINK do estado americano do Tennesse, cobra US$ 35,00 por mes o acesso Internet através de um canal B ISDN (64 Kbps). Se forem usados os dois canais (2B), o custo sobe para US$ 55 mensais. No estado, a Bell South cobra de US$ 29 a 35 dólares mensais para manter a linha ISDN. Apesar de mais caro que o acesso padrão através de modem e linha discada (que é facilmente encontrado nos EUA por US$ 19 mensais), o custo do ISDN tem se mostrado atraente para muitos usuários mais exigentes da Internet. É claro que aos preços mencionados, devem ser somados os preços dos equipamentos que devem ser adquiridos e da instalação da linha ISDN.

Os valores de custo mencionados dizem respeito ao ano de 1997 e são apresentados em caráter meramente ilustrativo.

11.3 Frame Relay, Detalhes de Operação

11.3.1 Introdução

Em apostila anterior, já foram vistos os aspectos gerais da tecnologia Frame Relay, suas vantagens e desvantagens em relação a outras tecnologias e seu modo de funcionamento básico. Agora que já discutimos o ISDN em algum detalhe, é possível nos aprofundarmos em alguns aspectos técnicos importantes do Frame Relay que ainda não foram abordados.

11.3.2 PVC e SVC

Um conceito fundamental em muitas redes WAN que contemplam ligações orientadas à conexão é aquele dos circuitos permanentes e dos chaveados. Tanto no X.25 como no Frame Relay, (e também no ATM, como será visto posteriormente), os circuitos são ditos *virtuais* pois eles são estabelecidos ou cancelados de acordo com a vontade dos operadores e/ou usuários não tendo relação fixa com os circuitos físicos, ou seja, com o meio físico sobre os quais eles são implementados.

Tanto os PVCs como os SVCs (as siglas derivam do inglês: “permanent virtual circuit” e “ switched virtual circuit”) são circuitos virtuais como o próprio nome revela. Vários PVCs ou SVCs podem conviver sobre um mesmo meio físico compartilhando sua largura de banda através de técnicas de multiplexação.

Existe porém uma diferenciação muito importante entre PVCs e SVCs. O PVC é um circuito estabelecido pelos operadores de rede, antes que o circuito possa ser utilizado. Uma vez em operação, o PVC funciona praticamente como uma linha privativa conectando sempre os mesmos dois pontos. Quando o circuito não é mais necessário, o (s) usuário (s) pede a companhia que lhe fornece o serviço, que desative o PVC. Isto é então feito novamente por um operador da rede. Pode-se dizer que o PVC é um circuito virtual *estático*.

Já o SVC é um circuito dito *chaveado*, ou seja, quando um ponto A deseja se comunicar com um ponto B, uma requisição à rede é efetuada. A rede então, automaticamente estabelece um circuito entre A e B (se B concordar é claro) e então A e B podem trocar dados até que terminem suas transações. Então, um pedido de desconexão é feito à rede, que cancela o SVC. Note que aqui, ao contrário do caso do PVC, não existe a necessidade de operador humano estabelecer ou cancelar a conexão. Pode-se dizer portanto que o SVC implementa uma conexão virtual *dinâmica*.

11.3.3 O "Frame" do Frame Relay

O "frame" ou pacote do Frame Relay é muito parecido com o LAPD de onde ele foi derivado. Na figura 11.3, o formato do frame é mostrado.

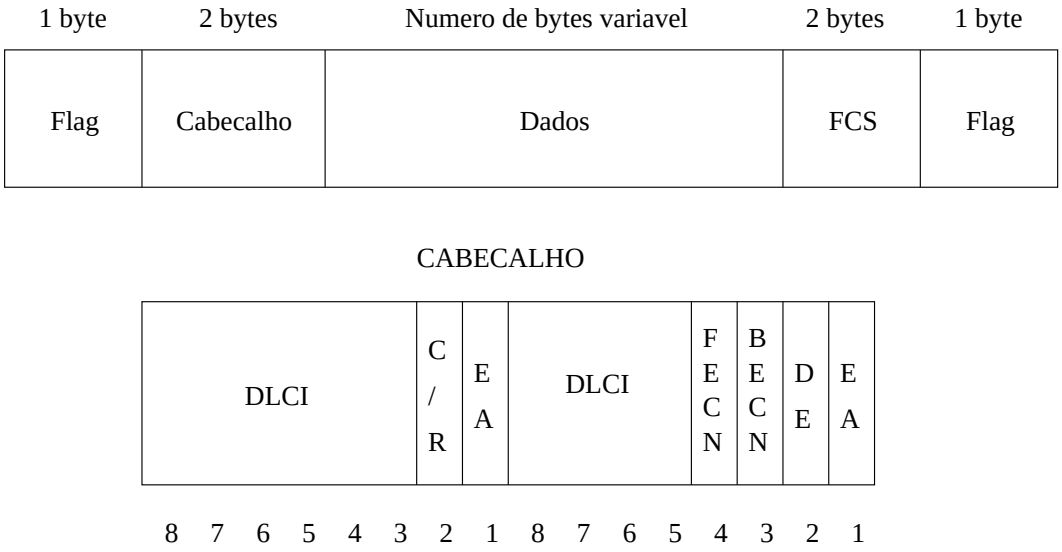


Figura 11.3: O pacote Frame Relay

Note que, o campo de endereçamento e o de controle são unificados num único campo chamado de "header" ou cabeçalho.

Os flags tem exatamente o mesmo significado que no LAPD.

Dentro do cabeçalho, o subcampo DLCI ("Data Link Connection Identifier") já foi visto anteriormente. Ele identifica o canal virtual seja ele PVC ou SVC. Cada equipamento da rede Frame Relay tem uma tabela que diz exatamente qual o caminho que está associado a cada DLCI. Existem 10 bits para o DLCI o que implica que existem apenas 1024 valores possíveis para os circuitos (na verdade, os valores disponíveis de DLCI são menores já que alguns valores são reservados — é o caso do valor 0 reservado para sinalização e do valor 1023 reservado para gerênciamento). Esse valor não é tão pequeno como pode parecer pois os DLCI tem significado apenas local (podem ser atribuídos valores globais de DLCI mas não entraremos nesse detalhe nesse curso). Isso significa que, quando o circuito passa pelos vários roteadores da rede, o número DLCI varia para um mesmo circuito. Mesmo assim, algumas vezes termos apenas 10 bits para o DLCI é limitante. Através do bit EA, é possível estender-se o campo do cabeçalho de tal forma que o número de bits para o DLCI é maior. A figura 11.4 ilustra as extensões de tamanho de frame. Não entraremos em detalhes do "DL-core". Note na utilização do campo EA ("address

extension") para determinar a extensão do pacote.

DLCI			C/R	EA=0
DLCI	FECN	BECN	DE	EA=1

DLCI			C/R	EA=0
DLCI	FECN	BECN	DE	EA=0
DLCI ou controle DL-core			D/C	EA=1

DLCI			C/R	EA=0
DLCI	FECN	BECN	DE	EA=0
DLCI				EA=0
DLCI ou controle DL-core			D/C	EA=1

Figura 11.4: Extensão do pacote Frame Relay

Os campos FECN ("forward explicit congestion notification") e BECN ("backward explicit congestion notification") são utilizados para remediar situações de congestionamento na rede. De certa forma, eles implementam um mecanismo simples de controle de fluxo. Se um equipamento da rede Frame Relay tem problemas de rotear um determinado tráfego, ele coloca o bit BECN do pacote em um e envia o pacote para a fonte da transmissão. A fonte, ao receber um pacote com BECN igual a um pode adotar uma política de reduzir a velocidade de transmissão de dados naquele circuito virtual em particular. Da mesma forma, o FECN é usado para avisar ao destino que um equipamento no caminho está com problemas de congestionamento.

O campo de C/R tem a mesma função que no LAPD utilizado no ISDN e já visto anteriormente.

O bit DE ("discard eligibility") indica se o pacote pode ser descartado, em caso de congestionamento ou não. Na verdade, ele indica que pacotes são mais importantes (DE em 0). Em caso de congestionamento, os pacotes com DE em 1 são os primeiros a serem descartados. É claro que, em caso de congestionamento severo, ou na falta de pacotes com o bit DE em 1, os pacotes com bit DE em zero também serão descartados.

Em algumas redes Frame Relay, o bit DE é usado pela operadora de serviço para tentar manter uma qualidade de serviço (QoS) pré-estabelecida em contrato. Em redes Frame Relay, um fator que pode ser especificado no contrato entre usuário e provedor é o CIR ("committed information rate"). Se a quantidade de tráfego utilizada pelo usuário num determinado intervalo de tempo for menor que o CIR contratado, a operadora não altera o DE. Se, por outro lado, o tráfego for maior, a operadora começa a alterar bits DE de frames em excesso para 1. Desta forma, o usuário até pode utilizar mais largura de banda do que a contratada mas até o ponto que isso não congestione a rede da operadora.

O conceito de QoS é muito importante em redes modernas. Em particular, trata-se de um conceito fundamental no B-ISDN (ATM).

11.3.4 CLLM

Como já foi dito anteriormente, redes Frame Relay eliminam o controle de fluxo existente em outros protocolos tais como o X.25. Assim, é preciso tratar-se o problema de congestionamento de rede de forma diferente. Para isso foi criado o protocolo CLLM ("consolidated link layer management")

Suponha que o tráfego entre usuário e rede atinja valores muito altos e comece a provocar congestionamento em alguma chave da rede. Pode acontecer que o tráfego esteja tão intenso que não seja possível mandar um pacote através do próprio circuito virtual para avisar à fonte que o congestionamento está ocorrendo.

O CLLM especifica então que o circuito virtual com DLCI 1023 pode ser utilizado para esse fim se necessário. O DLCI do circuito congestionante é detectado e seu número enviado atrave's de mensagens especiais para a fonte para que esta possa tomar as providências necessárias.

11.3.5 Estabelecimento de SVC

No Frame Relay atualmente (1997), praticamente só são utilizados os PVCs. Acredita-se porém que no futuro próximo cresça o interesse por SVCs. Assim, os SVCs já começam a ser oferecidos por operadores nos EUA.

Para o estabelecimento de um circuito virtual chaveado, é necessária a troca de mensagens entre os pontos que querem se conectar e a própria rede de conexão. A essa troca de mensagens dá-se o nome de *sinalização* e é um conceito muito importante na telefonia. Em redes de computadores, não era muito utilizado até o advento do X.25, ISDN, Frame Relay e ATM.

A sinalização do Frame Relay é especificada num documento chamado DSS1 ("Digital Signaling System"). Não entraremos em detalhes do DSS1, que, inclusive, é bastante complexo. Basta aqui dar uma idéia geral do que ocorre.

O ponto que origina a chamada envia para a rede uma mensagem de *setup* que contém campos que especificam o endereço do ponto com o qual ele deseja estabelecer conexão, características deseja'veis da conexão, DLCIs, etc. A rede envia de volta para o ponto origem uma mensagem de *call proceeding* e manda o pedido de conexão para o ponto destino que por sua vez pode mandar uma mensagem para a rede do tipo *call proceeding*. Se resolver aceitar a conexão, o destino envia de volta para a origem uma mensagem do tipo *connect*. Quando essa mensagem chega a rede, ela pode enviar uma mensagem do tipo *connect acknowlegded* para o destino enquanto envia a mensagem *connect* para a origem. Quando a origem recebe a mensagem de *connect* ela envia de volta para a rede uma mensagem do tipo *connect acknowlegded* e começa a se comunicar com o destino através do circuito que foi estabelecido no processo. Ao final da troca de dados, a origem envia uma mensagem de *disconnect* para o destino que responde com uma mensagem *release*. Ao receber a mensagem *release* do destino a rede envia para o destino uma mensagem de *release complete*. Finalmente quando a origem recebe a mensagem *release*, ela envia à rede uma mensagem de *release complete*

Capítulo 12

TLI, Transport Layer Interface

12.1 Transport Layer Interface (TLI)

12.1.1 Introdução e Generalidades

O sistema TLI, Transport Layer Interface, surgiu no Unix System V, release 3 como uma alternativa para o mecanismo de soquetes do BSD Unix [Stevens90]. A idéia fundamental do TLI é prover uma interface do sistema operacional com o nível de transporte de rede, em especial com o nível de transporte do modelo OSI (TP0 a TP4).

Ao contrário dos soquetes do BSD, o TLI não apareceu com nenhum transporte incluído, até que surgiu o release 4 onde foi incorporado o TCP/IP. Esse fato apresenta pontos positivos e negativos: uma grande vantagem é que o TLI acabou sendo bastante independente do protocolo de transporte; do lado negativo, destaca-se a dificuldade de se utilizar o TLI já que é necessário recorrer ao software de "third parties" para o transporte.

Uma importante diferença com os soquetes de Berkeley é que o TLI se constituiu de um conjunto de funções e subprogramas que ficam armazenados numa biblioteca e não de um conjunto de chamadas do sistema. Na verdade, o TLI foi projetado para trabalhar juntamente com os *streams*, que são uma generalização do sistema de entrada e saída do Unix tendo aparecido também no System V release 3. Para maiores detalhes sobre streams, veja [Ritchie84,Bach86,Stevens90]. Normalmente, a biblioteca de funções está em /usr/lib/libnsl.a ou /usr/lib/libnsl_s.a (nsl significa *Network Services Library*) . Uma compilação típica com TLI (nas estações Sun executando SunOS 4.1.1) seria:

```
$ cc -o tli_program tli_program.c -lnsl
```

Uma outra diferença com os soquetes de Berkeley é que em TLI, os dois processos que se comunicam são chamados de *pontos finais de transporte* sendo que os termos *soquete* ou *meia associação* não são utilizados.

12.1.2 As chamadas de biblioteca do TLI

Na figura 12.1, é apresentado um diagrama mostrando as chamadas do TLI num esquema cliente-servidor orientado a conexão.

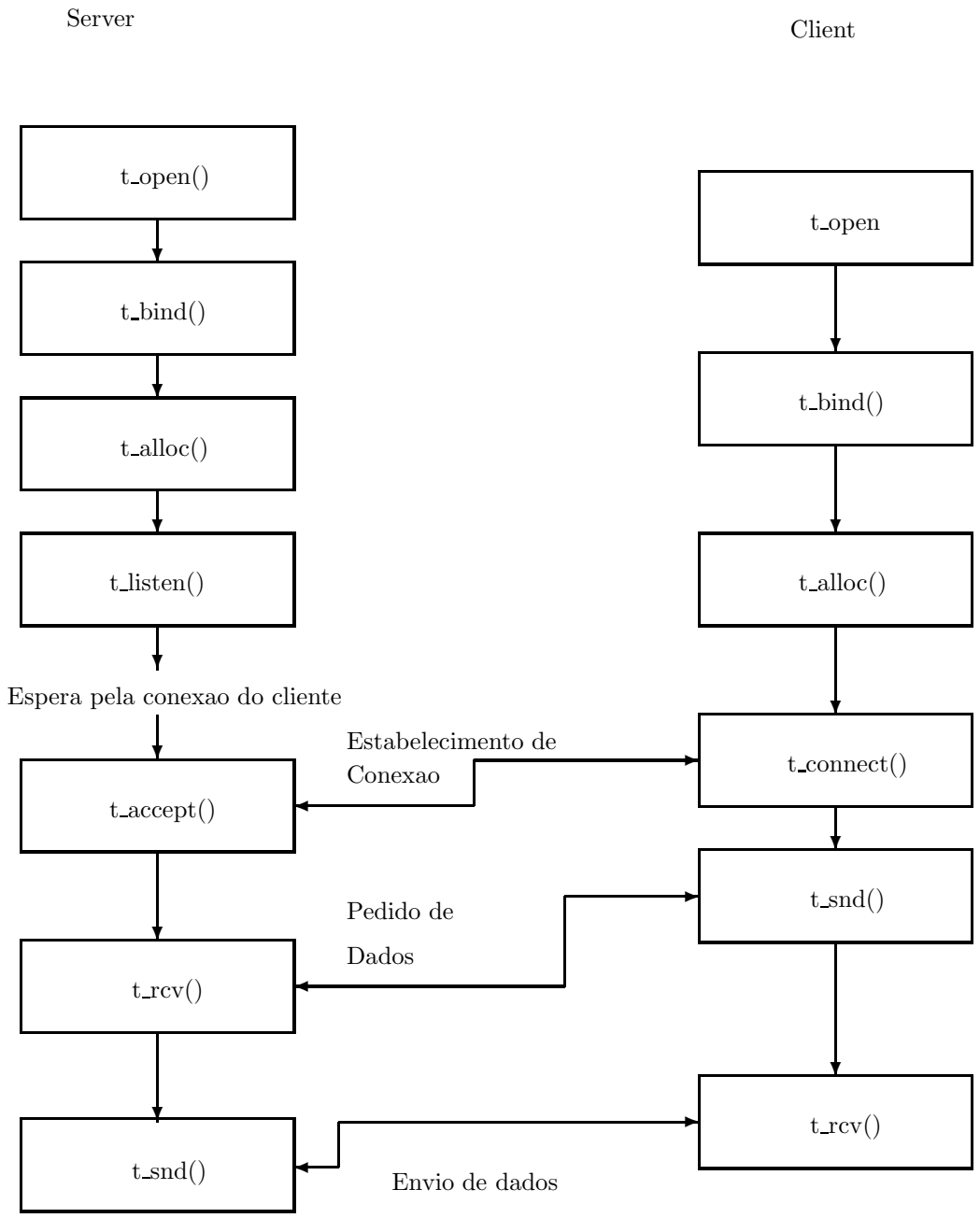


Figura 12.1: Chamadas TLI num cliente-servidor com conexão

A função **t_open** é a primeira a ser chamada e retorna um descritor de arquivo para o dispositivo que fornecerá o transporte. Arquivos típicos são `/dev/tcp`, `/dev/udp` `/dev/ip`. O descritor retornado é então utilizado nas outras funções TLI. O formato de t_open é:

int t_open(char **pathname*, int *oflag*, struct t_info **info*);

oflag são os flags similares ao da chamada **open**. t_info é uma estrutura retornada com valores referentes ao protocolo de transporte.

A função **t_bind** associa um endereço aos pontos finais de transporte. O formato é:

int t_bind(int *fd*, struct t_bind **request*, struct t_bind **return*);

A estrutura t_bind especifica o endereço e o número máximo de conexões permitidas.

A função **t_alloc** auxilia no processo de alocação de espaço das várias estruturas utilizadas no TLI. Seu formato é:

char *t_alloc(int *fd*, int *structtype*, int *fields*);

structtype especifica uma das várias estruturas utilizadas nas diversas funções. *fields* diz à função quais os campos para os quais se deve alocar espaço.

A função **t_connect** é usada pelo cliente para estabelecer uma conexão com o servidor. Seu formato é:

int t_connect(int *fd*, struct t_call **sendcall*, struct t_call **recvcall*);

A estrutura *sendcall* fornece o endereço do servidor ao qual o cliente tenta se conectar bem como opções e dados usados na conexão. *recvcall* é a resposta do servidor.

A função **t_listen** é a espera do servidor por um pedido de conexão do cliente. Seu formato é:

int t_listen(int *fd*, struct t_call **call*);

Para aceitar um pedido de conexão, o servidor executa a função **t_accept**. Seu formato é:

int t_accept(int *fd*, int *connfd*, struct t_call **call*);

O valor *connfd*, no caso de um servidor *concorrente*, é um novo descritor por nós criado para atender ao pedido de conexão. Assim, o descritor original *fd* fica liberado para esperar outros pedidos de conexão. Se o servidor for *iterativo*, *fd* e *connfd* são idênticos.

As funções **t_snd** e **t_rcv** são usadas para transmitir e receber dados. Os formatos são:

int t_snd(int *fd*, char **buff*, unsigned int *nbytes*, int *flags*);

int t_rcv(int *fd*, char **buff*, unsigned int *nbytes*, int *flags*);

Os três primeiros parâmetros são similares aos das chamadas de sistema **read** e **write**. *flags* especifica se existe mais dado a ser enviado (ou recebido) e se os dados devem ser trafegar "fora de banda".

12.1.3 Exemplo

Nessa sessão apresentamos um exemplo de um par cliente-servidor em TLI, baseado num exemplo de Stevens [Stevens90]. O exemplo é bastante simples: o cliente lê do terminal uma frase qualquer e a envia ao servidor que ecoa a frase de volta ao cliente.

A seguir apresenta-se o programa servidor:

```
/*
 * Example of server using TCP protocol
 */

#include       "tli_header.h"

main(argc, argv)
int     argc;
char    *argv[];
{
        int                         tfd, newtfd, clilen, childpid;
        struct sockaddr_in          cli_addr, serv_addr;
        struct t_bind               req;
        struct t_call               *callptr;

        pname = argv[0];


        /*
         * Create a TCP transport endpoint
         */

        if ( (tfd = t_open(DEV_TCP, O_RDWR, (struct t_info *) 0)) < 0)
                printf("server: can't t_open %s", DEV_TCP);
        printf("Ja fiz t_open tfd= %d \n",tfd);
        /*
         * Bind our local address so that the client can send to us.
         */

        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
        serv_addr.sin_port = htons(SERV_TCP_PORT);

        req.addr.maxlen = sizeof(serv_addr);
        req.addr.len = sizeof(serv_addr);
        req.addr.buf = (char *) &serv_addr;
        req.qlen = 5;
```

```
        if (t_bind(tfd, &req, (struct t_bind *) 0) < 0 )
               printf("server: can't t_bind local address");

        /*
         * Allocate a t_call structure for t_listen() and t_accept().
         */

     if ( (callptr = (struct t_call *) t_alloc(tfd, T_CALL, T_ADDR)) == NULL)
         printf("server: t_alloc error for T_CALL");

        for ( ; ; ) {
                /*
                 * Wait for a connection from a client process
                 * This is an example of a concurrent server
                 */

        if (t_listen(tfd, callptr) < 0)
                 printf("server: t_listen error");

        if ( (newtfd = accept_call(tfd, callptr, DEV_TCP, 1)) < 0)
                printf("server: accept_call error");
        printf("Aceitei %d \n",newtfd);
        if ( (childpid = fork ()) < 0)
                printf("server: fork error");

        else if (childpid == 0) {      /* child process */
             t_close(tfd);             /* close original endpoint */
             str_echo(newtfd);         /* process the request */
             exit(0);
           }

        close(newtfd);       /* parent process */
      }
      }
```

Esse servidor é concorrente, ou seja, atende a vários clientes simultaneamente. Note que ele chama a função accept_call para cada cliente que tenta estabelecer a conexão. A função accept_call pode ser vista a seguir.

```
/*
 *
 * Accept an incoming connection request
 * Return the new descriptor that refers to the newly accepted connection,
 * or -1. The only time we return -1 is if a disconnect arrives before
 * the accpet is performed.
 */

#include <stdio.h>
#include <tiuser.h>
#include <fcntl.h>
#include <stropts.h>
```

```
int
accept_call(listenfd, callptr, name, rwflag)
int             listenfd;     /* the descriptor caller used for t_listen() */
struct t_call   *callptr; /* from t_listen(), passed to t_accept90 */
char            *name;    /* name of transport provider */
int             rwflag;   /* if nonzero, push read/write module */
{
       int          newfd;
       extern int   t_errno;

       /*
        * Open the transport provider to get a new file descriptor.
        */

        if ( (newfd= t_open(name, O_RDWR, (struct t_info *) 0)) < 0)
               printf("t_open error");

       /*
        *  Bind any local address to the new descriptor. Since this function
        *  is intended to be called by a server after a
        *  connection request has arrived, any local address will suffice.
        */

        if (t_bind(newfd, (struct t_bind *) 0, (struct t_bind *) 0) < 0)
                printf("t_bind error");

       /*
        * Accept the connection request on the new descriptor.
        */

        if (t_accept(listenfd, newfd, callptr) < 0) {
                if (t_errno == TLOOK) {
                        /*
                         * An asynchronous event has occurred. We must have
                         * received a disconnect. Go ahead and call t_rcvdis()
                         * then close the new file descriptor that we opened
                         * above
                         */

                        if (t_rcvdis(listenfd, (struct t_discon *) 0) < 0)
                                printf("t_rcvdis error");
                        if (t_close(newfd) < 0)
                                printf("t_rcvdis error");
                        return(-1);   /* return error to caller */
                       }
                       printf("t_accept error");
              }

              /*
               * If the caller requests, push the stream module "tirdwr"\ onto
               * the new stream , so that the read(2) and write(2) system calls
               * can be used. We first have to pop the "timod"\ module (the
               * default).
               */
```

```
        if (rwflag) {
              if (ioctl(newfd, I_POP, (char *) 0) <0)
                      printf("I_POP of timod failed");
              if (ioctl(newfd, I_PUSH, "tirdwr") < 0)
                      printf("I_PUSH of tirdwr failed");
            }
        return(newfd);
        }
```

No código acima, o arquivo *tli_header.h* contém as inicializações de constantes utilizadas no servidor (e também no cliente). O arquivo utilizado foi:

```
/*
 *  Definitions for TCP client/server programs
 */

#include     <stdio.h>
#include     <fcntl.h>
#include     <tiuser.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>

#define DEV_TCP    "/dev/tcp"

#define SERV_TCP_PORT    6000
#define SERV_HOST_ADDR   "143.107.54.5"\   /* host address for server */

#define MAXLINE 255

char *pname;
```

O cliente pode ser visto a seguir.

```
/*
 * Example of client using TCP protocol
 */

# include      "tli_header.h"

main(argc, argv)
     int      argc;
     char     *argv[];

{
     int          tfd;
     char         *t_alloc();      /*TLI function */
```

```
    struct  t_call   *callptr;
    struct  sockaddr_in    serv_addr;

    pname = argv[0];

    /*
     * Create a TCP transport endpoint and bind it.
     */

       if ( (tfd = t_open(DEV_TCP, O_RDWR,  0)) < 0)
               printf("client: can't t_open %s", DEV_TCP);

       if (t_bind(tfd, (struct t_bind *) 0, (struct t_bind *) 0) < 0 )
              printf("client: t_bind error");

    /*
     * Fill in the structure "serv_addr"\ with the address of the
     * server that we want to connect with.
     */

      bzero((char *) &serv_addr, sizeof(serv_addr));
      serv_addr.sin_family   = AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
      serv_addr.sin_port      = htons(SERV_TCP_PORT);

    /*
     * Allocate a t_call structure, and initialize it.
     * Let t_alloc() initialize the addr structure of the t_call structure.
     */

     if ( (callptr = (struct t_call *) t_alloc(tfd, T_CALL, T_ADDR)) == NULL)
           printf("client: t_alloc error");
     callptr->addr.maxlen = sizeof(serv_addr);
     callptr->addr.len    = sizeof(serv_addr);
     callptr->addr.buf    = (char *) &serv_addr;
     callptr->opt.len     = 0;              /* no options */
     callptr->udata.len   = 0;              /* no user data with connect */

    /*
     * Connect to the server.
     */

    if (t_connect(tfd, callptr, (struct t_call *) 0) < 0)
           printf("client: can't t_connect to server");

    doit(stdin, tfd);      /* do it all */

    close(tfd);
    exit(0);
  }

/*
 * Read the contents of the FILE *fp, write each line to the
 * transport endpoint (to the server process), then read a line back from
 * the transport endpoint and print it on the standard output.
```

```
 */

   doit(fp, tfd)
   register FILE *fp;
   register int  tfd;
{
      int   n, flags;
      char  sendline[MAXLINE], recvline[MAXLINE + 1];

      while (fgets(sendline, MAXLINE, fp) != NULL) {
              n = strlen (sendline);
              if (t_snd(tfd,sendline, n, 0) != n)
                    printf("client: t_snd error");

              /*
               * Now read a line from the transport endpoint and write it
               * to out standard output.
               */

              n = t_rcv(tfd, recvline, MAXLINE, &flags);
              if (n < 0)
                       printf("client: t_rcv error");
              recvline[n] = 0;
              fputs(recvline, stdout);
            }

      if (ferror(fp))
            printf("client: error reading file");
    }
```

Finalmente apresentamos as rotinas de entrada e saída que implementam o eco realizado pelo servidor.

```
/*
 * Read "n"\ bytes from a descriptor
 * Use in place of read() when fd is a stream socket.
 */

int
readn(fd, ptr, nbytes)
register int    fd;
register char  *ptr;
register int nbytes;
{

      int   nleft, nread;

      nleft = nbytes;
      while (nleft > 0) {
        nread = read(fd, ptr, nleft);
        if (nread < 0)
```

```
                return(nread);          /* error, return < 0 */
        else if (nread == 0)
                break;                  /* EOF */

        nleft -= nread;
        ptr   += nread;
      }
      return(nbytes - nleft);          /* return >= 0 */
    }

    int
    writen( fd, ptr, nbytes)
    register int     fd;
    register char    *ptr;
    register int     nbytes;
    {

         int nleft, nwritten;

         nleft = nbytes;
         while (nleft > 0) {
                nwritten = write(fd, ptr, nleft);
                if (nwritten <= 0)
                      return(nwritten);        /* error */

                nleft -= nwritten;
                ptr   += nwritten;
              }
         return(nbytes - nleft);
       }

   int
   readline(fd, ptr, maxlen)
   register int     fd;
   register char    *ptr;
   register int     maxlen;
{
        int n, rc;
        char c;

        for ( n = 1; n < maxlen; n++) {
                if ( (rc = read(fd, &c, 1)) == 1) {
                        *ptr++ = c;
                        if (c == '\n')
                              break;
                 } else if (rc == 0) {
                         if (n == 1)
                                 return(0);     /* EOF, no data read */
                         else
                                 break;         /* EOF, some data was read */
                 } else
                         return(-1);            /* error */
               }

              *ptr = 0;
```

```
                return(n);
        }


/*
 * Read a stream socket one line at a time, and write each line back
 * to sender.
 *
 * Return when the connection is terminated.
 */

#define MAXLINE 512

str_echo(sockfd)
int        sockfd;
{
        int       n;
        char      line[MAXLINE];

        for ( ; ; ) {
                n = readline(sockfd, line, MAXLINE);
                if (n == 0)
                        return;       /* connection terminated */
                else if (n < 0)
                       printf("str_echo: readline error");
                if (writen(sockfd, line, n) != n)
                       printf("str_echo: writen error");
              }
      }
```

Os comandos utilizados para a compilação dos programas acima foram, no SunOS 4.1.1:

% cc -o tli_server tli_server.c tli_accept.c str_echo.c -lnsl

% cc -o tli_client tli_client.c -lnsl

No Interactive System V Unix para 386 (agora chamado de Solaris), os comandos são:

$ cc -o tli_server tli_server.c tli_accept.c str_echo.c -linet -lnsl_s

$ cc -o tli_client tli_client.c -linet -lnsl_s

Parte II

Prática

Capítulo 13

Prática 1

13.1 Introdução

Nesta prática, o aluno instala sistemas operacionais em um computador IBM PC compatível e faz experimentos com placas de rede.

13.2 Roteiro da Prática

1. Instale (ou verifique que estão instalados) os sistemas operacionais DOS, Windows e FreeBSD.
2. Instale algum selecionador de boot para ser possível carregar qualquer dos sistemas.
3. Verifique que os sistemas reconhecem as interfaces de rede.
4. Configure o TCP/IP no seu micro.
5. Mostre que existe conectividade entre seu PC e o PC do outro grupo.
6. Configure um PC extra com duas placas de rede e sistema FreeBSD 4.3.
7. Configure tal PC como um roteador.
8. Coloque um PC em uma subrede (192.168.10.1) e outro em outra subrede (192.168.12.1). Mostre conectividade entre os dois PCs passando pelo PC extra (roteador).

Capítulo 14

Prática 2

14.1 Introdução

Nesta prática, o aluno aprende a trabalhar com o ipfw (IP firewall) e com o NAT (Network Address Translation) no sistema FreeBSD.

14.2 Roteiro da Prática

1. Leia atentamente os manuais de ipfw(8) e natd(8).
2. Compile um novo kernel com suporte para firewall e "diversion" na máquina "gateway". Para isso, va ao diretório /usr/src/sys/i386/conf e edite a configuração (com nome, por exemplo, ROTEADOR) do seu micro. Inclua as opções `options IPFIREWALL` e `options IPDIVERT`. Faça `config ROTEADOR`, `cd /usr/src/sys/compile/ROTEADOR`, `make depend`, `make` e `make install`.
3. Reinicialize o "gateway" e defina as regras para o firewall. Em particular, tente:
   (a) Nao permita ping de nenhum computador para o gateway.
   (b) Permita ping apenas dos computadores 1 e 2 para o gateway. Nenhum outro computador da Internet deve poder fazer ping para o gateway.
   (c) Permita ping de todos os computadores para o gateway mas permita telnet somente dos computadores 1 e 2.
4. Permita o acesso dos computadores 1 e 2 à Internet através do gateway usando NAT.
5. Faça um servidor de telnet no computador 1 e um de ftp no computador 2 que sejam acessados por computadores da Internet através do gateway usando NAT.

Capítulo 15

Prática 3

15.1 Introdução

Nesta prática, o aluno aprende a trabalhar com dois servidores de arquivos: o NFS, muito usado em ambientes Unix e o *samba*, utilitário que implementa um servidor de arquivos compatível com os sistemas Windows.

15.2 Roteiro da Prática

1. Leia atentamente os manuais de `mount_nfs(8)`, `exports(5)`, `nfsd(8)`, `mountd(8)` e `showmount(8)`.
2. Em um servidor, crie um subdiretório /fora e coloque alguns arquivos nele.
3. Crie um arquivo `/etc/exports` que permita a um cliente acesso ao diretório `/fora`.
4. Verifique que o servidor está executando `portmap`.
5. Execute `nfsd` e `mountd` no servidor com as devidas opções.
6. De uma máquina cliente, monte o diretório `/fora` do servidor em `/mnt` e verifique que voce tem acesso, a partir do cliente, a todos os arquivos em `/fora`.
7. Instale o pacote samba no FreeBSD do servidor.
8. Leia atentamente os manuais de `nmbd(8)`, `smbd(8)`, `smb.conf(5)`, `lmhosts(5)` e `smbclient(1)`.
9. Edite `/usr/local/etc/smb.conf` criando um diretório público que possa ser lido e vasculhado ("browseable") por todas as máquinas na rede local.
10. Dispare o `nmbd` e o `smbd`. Teste a configuração com o `smbclient`.
11. Coloque a máquina servidora no arquivo `lmhosts` do máquina cliente Windows. Verifique que você pode ter acesso aos arquivos pelo Windows.

Capítulo 16

Prática 4

16.1 Introdução

É pedido ao aluno que escreva um "stack" IP simplificado para DOS. O programa deverá ser capaz de enviar e responder "echo requests" para computadores na mesma subrede

16.2 Roteiro da Prática

1. Leia com atenção os RFCs referentes ao ICMP e ao ARP.
2. Escreva um programa em turbo C 2.01 que implementa pedido e resposta **arp** e pedido e resposta ICMP (**echo request e echo reply**)

Capítulo 17

RFC0792 ICMP

Network Working Group
Request for Comments: 792

J. Postel
ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable.

There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragemented datagrams. (Fragment zero has the fragment offeset equal zero).

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation, see [1].

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

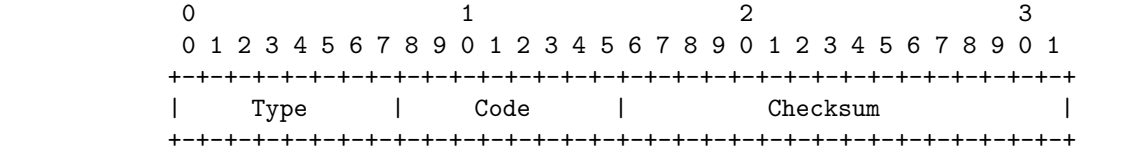
Source Address

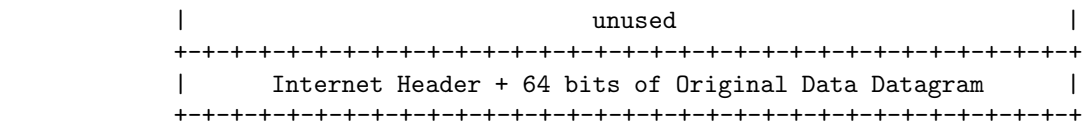
The address of the gateway or host that composes the ICMP message. Unless otherwise noted, this can be any of a gateway's addresses.

Destination Address

The address of the gateway or host to which the message should be sent.

Destination Unreachable Message





IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

3

Code

0 = net unreachable;

1 = host unreachable;

2 = protocol unreachable;

3 = port unreachable;

4 = fragmentation needed and DF set;

5 = source route failed.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original

datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

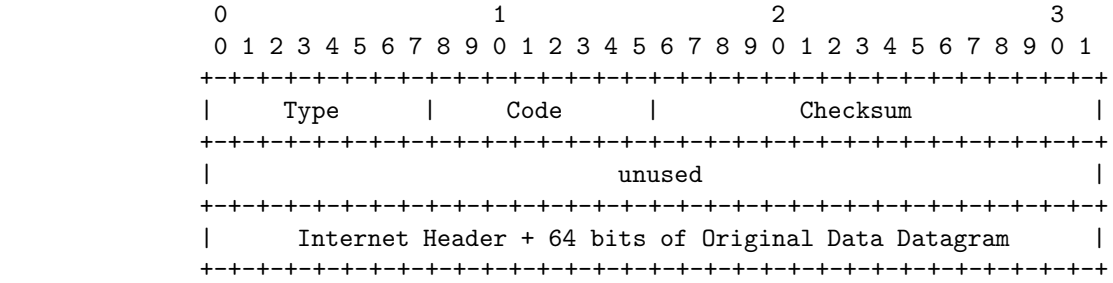
If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

Another case is when a datagram must be fragmented to be forwarded by a gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway. Codes 2 and 3 may be received from a host.

Time Exceeded Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

11

Code

0 = time to live exceeded in transit;

1 = fragment reassembly time exceeded.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If the gateway processing a datagram finds the time to live field

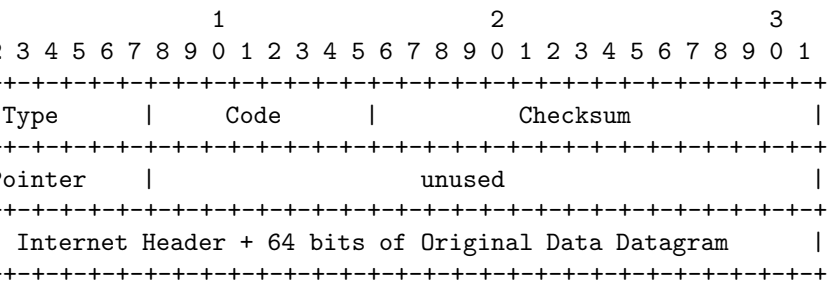
is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message.

If fragment zero is not available then no time exceeded need be sent at all.

Code 0 may be received from a gateway. Code 1 may be received from a host.

Parameter Problem Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

[Page 8]

September 1981
RFC 792

Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

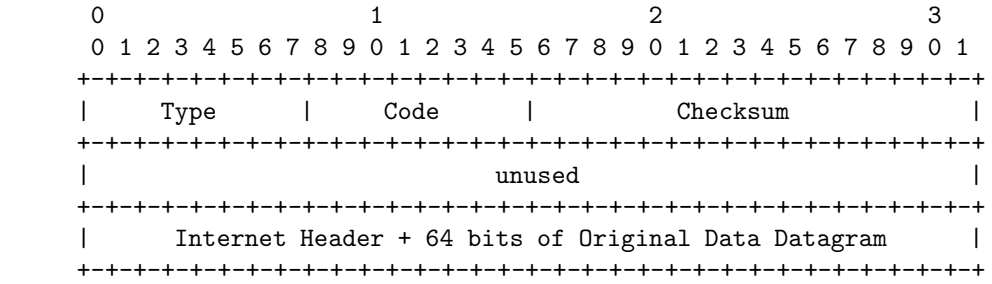
The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 may be received from a gateway or a host.

[Page 9]

September 1981
RFC 792

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway

discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which

triggered the source quench message may be delivered.

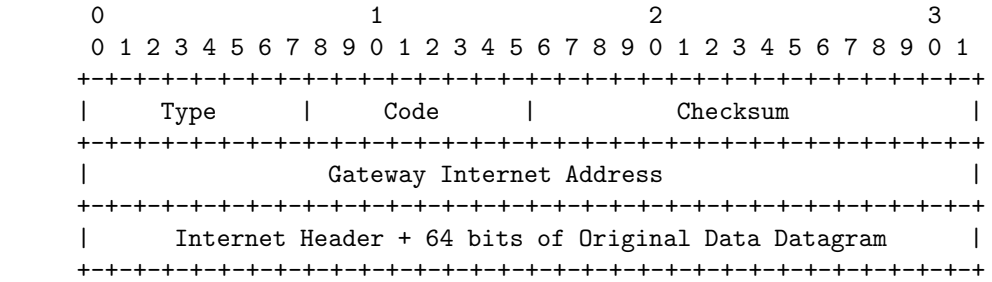
Code 0 may be received from a gateway or a host.

[Page 11]

September 1981

RFC 792

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.

1 = Redirect datagrams for the Host.

2 = Redirect datagrams for the Type of Service and Network.

3 = Redirect datagrams for the Type of Service and Host.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

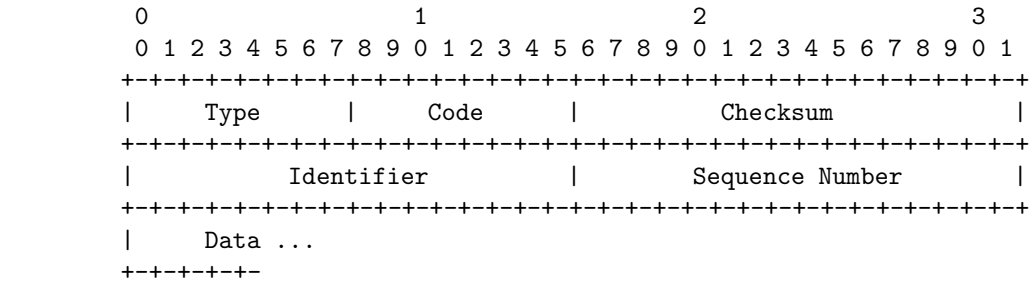
Description

The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

Echo or Echo Reply Message



IP Fields:

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

IP Fields:

Type

8 for echo message;

0 for echo reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

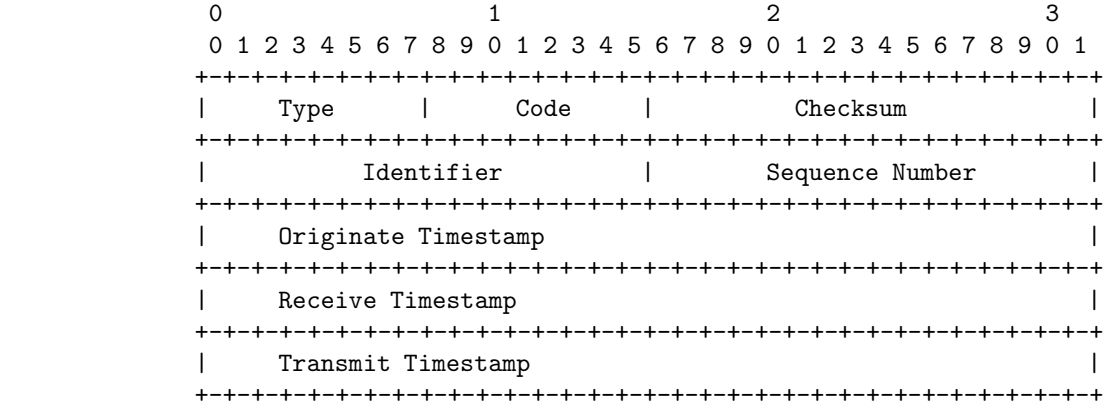
Description

The data received in the echo message must be returned in the echo reply message.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

Timestamp or Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply

reversed, the type code changed to 14, and the checksum
recomputed.

IP Fields:

Type

13 for timestamp message;

14 for timestamp reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's
complement sum of the ICMP message starting with the ICMP Type.
For computing the checksum , the checksum field should be zero.
This checksum may be replaced in the future.

Identifier

[Page 16]

September 1981
RFC 792

If code = 0, an identifier to aid in matching timestamp and
replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and
replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the
reply together with an additional timestamp. The timestamp is 32
bits of milliseconds since midnight UT. One use of these
timestamps is described by Mills [5].

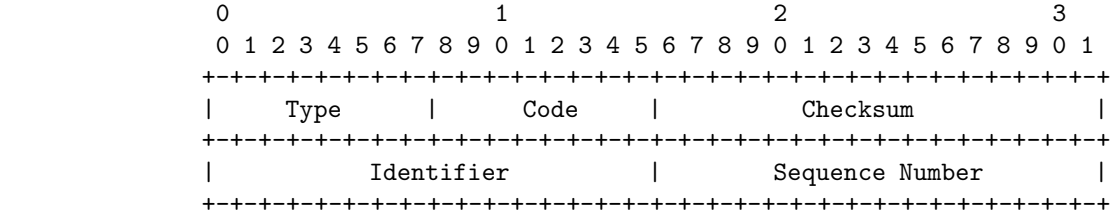
The Originate Timestamp is the time the sender last touched the
message before sending it, the Receive Timestamp is the time the
echoer first touched it on receipt, and the Transmit Timestamp is
the time the echoer last touched the message on sending it.

If the time is not available in miliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Information Request or Information Reply Message



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

IP Fields:

Type

15 for information request message;

16 for information reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

[Page 18]

September 1981
RFC 792

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

[Page 19]

September 1981
RFC 792

Summary of Message Types

0 Echo Reply

3 Destination Unreachable

4 Source Quench

5 Redirect

8 Echo

11 Time Exceeded

12 Parameter Problem

13 Timestamp

14 Timestamp Reply

15 Information Request

16 Information Reply

References

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Information Processing Techniques Office, Defense Advanced Research Projects Agency, July 1978.

[3] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Beranek and Newman, April 1979.

[4] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

[5] Mills, D., "DCNET Internet Clock Service," RFC 778, COMSAT Laboratories, April 1981.

Capítulo 18

RFC0826 - ARP

Network Working Group
Request For Comments: 826

David C. Plummer
(DCP@MIT-MC)
November 1982

An Ethernet Address Resolution Protocol
-- or --
Converting Network Protocol Addresses
to 48.bit Ethernet Address
for Transmission on
Ethernet Hardware

Abstract

The implementation of protocol P on a sending host S decides, through protocol P's routing mechanism, that it wants to transmit to a target host T located some place on a connected piece of 10Mbit Ethernet cable. To actually transmit the Ethernet packet a 48.bit Ethernet address must be generated. The addresses of hosts within protocol P are not always compatible with the corresponding Ethernet address (being different lengths or values). Presented here is a protocol that allows dynamic distribution of the information needed to build tables to translate an address A in protocol P's address space into a 48.bit Ethernet address.

Generalizations have been made which allow the protocol to be used for non-10Mbit Ethernet hardware. Some packet radio networks are examples of such hardware.

```
The protocol proposed here is the result of a great deal of
discussion with several other people, most notably J. Noel
Chiappa, Yogen Dalal, and James E. Kulp, and helpful comments
from David Moon.



[The purpose of this RFC is to present a method of Converting
Protocol Addresses (e.g., IP addresses) to Local Network
Addresses (e.g., Ethernet addresses).  This is a issue of general
concern in the ARPA Internet community at this time.  The
method proposed here is presented for your consideration and
comment.  This is not the specification of a Internet Standard.]

Notes:
------

This protocol was originally designed for the DEC/Intel/Xerox
10Mbit Ethernet.  It has been generalized to allow it to be used
for other types of networks.  Much of the discussion will be
directed toward the 10Mbit Ethernet.  Generalizations, where
applicable, will follow the Ethernet-specific discussion.

DOD Internet Protocol will be referred to as Internet.

Numbers here are in the Ethernet standard, which is high byte
first.  This is the opposite of the byte addressing of machines
such as PDP-11s and VAXes.  Therefore, special care must be taken
with the opcode field (ar$op) described below.

An agreed upon authority is needed to manage hardware name space
values (see below).  Until an official authority exists, requests
should be submitted to
        David C. Plummer
        Symbolics, Inc.
        243 Vassar Street
        Cambridge, Massachusetts  02139
Alternatively, network mail can be sent to DCP@MIT-MC.

The Problem:
------------

The world is a jungle in general, and the networking game
contributes many animals.  At nearly every layer of a network
architecture there are several potential protocols that could be
used.  For example, at a high level, there is TELNET and SUPDUP
for remote login.  Somewhere below that there is a reliable byte
stream protocol, which might be CHAOS protocol, DOD TCP, Xerox
BSP or DECnet.  Even closer to the hardware is the logical
transport layer, which might be CHAOS, DOD Internet, Xerox PUP,
or DECnet.  The 10Mbit Ethernet allows all of these protocols
(and more) to coexist on a single cable by means of a type field
in the Ethernet packet header.  However, the 10Mbit Ethernet
```

requires 48.bit addresses on the physical cable, yet most
protocol addresses are not 48.bits long, nor do they necessarily
have any relationship to the 48.bit Ethernet address of the
hardware. For example, CHAOS addresses are 16.bits, DOD Internet
addresses are 32.bits, and Xerox PUP addresses are 8.bits. A
protocol is needed to dynamically distribute the correspondences
between a <protocol, address> pair and a 48.bit Ethernet address.

Motivation:

Use of the 10Mbit Ethernet is increasing as more manufacturers
supply interfaces that conform to the specification published by
DEC, Intel and Xerox. With this increasing availability, more
and more software is being written for these interfaces. There
are two alternatives: (1) Every implementor invents his/her own
method to do some form of address resolution, or (2) every
implementor uses a standard so that his/her code can be
distributed to other systems without need for modification. This
proposal attempts to set the standard.

Definitions:

Define the following for referring to the values put in the TYPE
field of the Ethernet packet header:
        ether_type$XEROX_PUP,
        ether_type$DOD_INTERNET,
        ether_type$CHAOS,
and a new one:
        ether_type$ADDRESS_RESOLUTION.
Also define the following values (to be discussed later):
        ares_op$REQUEST (= 1, high byte transmitted first) and
        ares_op$REPLY   (= 2),
and
        ares_hrd$Ethernet (= 1).

Packet format:

To communicate mappings from <protocol, address> pairs to 48.bit
Ethernet addresses, a packet format that embodies the Address
Resolution protocol is needed. The format of the packet follows.

    Ethernet transmission layer (not necessarily accessible to
         the user):
        48.bit: Ethernet address of destination
        48.bit: Ethernet address of sender
        16.bit: Protocol type = ether_type$ADDRESS_RESOLUTION
    Ethernet packet data:
        16.bit: (ar$hrd) Hardware address space (e.g., Ethernet,
                         Packet Radio Net.)
        16.bit: (ar$pro) Protocol address space. For Ethernet
                         hardware, this is from the set of type
                         fields ether_typ$<protocol>.

```
         8.bit: (ar$hln) byte length of each hardware address
         8.bit: (ar$pln) byte length of each protocol address
        16.bit: (ar$op)  opcode (ares_op$REQUEST | ares_op$REPLY)
        nbytes: (ar$sha) Hardware address of sender of this
                         packet, n from the ar$hln field.
        mbytes: (ar$spa) Protocol address of sender of this
                         packet, m from the ar$pln field.
        nbytes: (ar$tha) Hardware address of target of this
                         packet (if known).
        mbytes: (ar$tpa) Protocol address of target.


Packet Generation:
------------------

As a packet is sent down through the network layers, routing
determines the protocol address of the next hop for the packet
and on which piece of hardware it expects to find the station
with the immediate target protocol address.  In the case of the
10Mbit Ethernet, address resolution is needed and some lower
layer (probably the hardware driver) must consult the Address
Resolution module (perhaps implemented in the Ethernet support
module) to convert the <protocol type, target protocol address>
pair to a 48.bit Ethernet address.  The Address Resolution module
tries to find this pair in a table.  If it finds the pair, it
gives the corresponding 48.bit Ethernet address back to the
caller (hardware driver) which then transmits the packet.  If it
does not, it probably informs the caller that it is throwing the
packet away (on the assumption the packet will be retransmitted
by a higher network layer), and generates an Ethernet packet with
a type field of ether_type$ADDRESS_RESOLUTION.  The Address
Resolution module then sets the ar$hrd field to
ares_hrd$Ethernet, ar$pro to the protocol type that is being
resolved, ar$hln to 6 (the number of bytes in a 48.bit Ethernet
address), ar$pln to the length of an address in that protocol,
ar$op to ares_op$REQUEST, ar$sha with the 48.bit ethernet address
of itself, ar$spa with the protocol address of itself, and ar$tpa
with the protocol address of the machine that is trying to be
accessed.  It does not set ar$tha to anything in particular,
because it is this value that it is trying to determine.  It
could set ar$tha to the broadcast address for the hardware (all
ones in the case of the 10Mbit Ethernet) if that makes it
convenient for some aspect of the implementation.  It then causes
this packet to be broadcast to all stations on the Ethernet cable
originally determined by the routing mechanism.


Packet Reception:
-----------------

When an address resolution packet is received, the receiving
Ethernet module gives the packet to the Address Resolution module
which goes through an algorithm similar to the following.
Negative conditionals indicate an end of processing and a
```

```
discarding of the packet.

?Do I have the hardware type in ar$hrd?
Yes: (almost definitely)
  [optionally check the hardware length ar$hln]
  ?Do I speak the protocol in ar$pro?
  Yes:
    [optionally check the protocol length ar$pln]
    Merge_flag := false
    If the pair <protocol type, sender protocol address> is
        already in my translation table, update the sender
        hardware address field of the entry with the new
        information in the packet and set Merge_flag to true.
    ?Am I the target protocol address?
    Yes:
      If Merge_flag is false, add the triplet <protocol type,
          sender protocol address, sender hardware address> to
          the translation table.
      ?Is the opcode ares_op$REQUEST?  (NOW look at the opcode!!)
      Yes:
        Swap hardware and protocol fields, putting the local
            hardware and protocol addresses in the sender fields.
        Set the ar$op field to ares_op$REPLY
        Send the packet to the (new) target hardware address on
            the same hardware on which the request was received.

Notice that the <protocol type, sender protocol address, sender
hardware address> triplet is merged into the table before the
opcode is looked at.  This is on the assumption that communcation
is bidirectional; if A has some reason to talk to B, then B will
probably have some reason to talk to A.  Notice also that if an
entry already exists for the <protocol type, sender protocol
address> pair, then the new hardware address supersedes the old
one.  Related Issues gives some motivation for this.

Generalization:  The ar$hrd and ar$hln fields allow this protocol
and packet format to be used for non-10Mbit Ethernets.  For the
10Mbit Ethernet <ar$hrd, ar$hln> takes on the value <1, 6>.  For
other hardware networks, the ar$pro field may no longer
correspond to the Ethernet type field, but it should be
associated with the protocol whose address resolution is being
sought.


Why is it done this way??
-------------------------

Periodic broadcasting is definitely not desired.  Imagine 100
workstations on a single Ethernet, each broadcasting address
resolution information once per 10 minutes (as one possible set
of parameters).  This is one packet every 6 seconds.  This is
almost reasonable, but what use is it?  The workstations aren't
generally going to be talking to each other (and therefore have
100 useless entries in a table); they will be mainly talking to a
mainframe, file server or bridge, but only to a small number of
```

other workstations (for interactive conversations, for example).
The protocol described in this paper distributes information as
it is needed, and only once (probably) per boot of a machine.

This format does not allow for more than one resolution to be
done in the same packet. This is for simplicity. If things were
multiplexed the packet format would be considerably harder to
digest, and much of the information could be gratuitous. Think
of a bridge that talks four protocols telling a workstation all
four protocol addresses, three of which the workstation will
probably never use.

This format allows the packet buffer to be reused if a reply is
generated; a reply has the same length as a request, and several
of the fields are the same.

The value of the hardware field (ar$hrd) is taken from a list for
this purpose. Currently the only defined value is for the 10Mbit
Ethernet (ares_hrd$Ethernet = 1). There has been talk of using
this protocol for Packet Radio Networks as well, and this will
require another value as will other future hardware mediums that
wish to use this protocol.

For the 10Mbit Ethernet, the value in the protocol field (ar$pro)
is taken from the set ether_type$. This is a natural reuse of
the assigned protocol types. Combining this with the opcode
(ar$op) would effectively halve the number of protocols that can
be resolved under this protocol and would make a monitor/debugger
more complex (see Network Monitoring and Debugging below). It is
hoped that we will never see 32768 protocols, but Murphy made
some laws which don't allow us to make this assumption.

In theory, the length fields (ar$hln and ar$pln) are redundant,
since the length of a protocol address should be determined by
the hardware type (found in ar$hrd) and the protocol type (found
in ar$pro). It is included for optional consistency checking,
and for network monitoring and debugging (see below).

The opcode is to determine if this is a request (which may cause
a reply) or a reply to a previous request. 16 bits for this is
overkill, but a flag (field) is needed.

The sender hardware address and sender protocol address are
absolutely necessary. It is these fields that get put in a
translation table.

The target protocol address is necessary in the request form of
the packet so that a machine can determine whether or not to
enter the sender information in a table or to send a reply. It
is not necessarily needed in the reply form if one assumes a
reply is only provoked by a request. It is included for
completeness, network monitoring, and to simplify the suggested
processing algorithm described above (which does not look at the
opcode until AFTER putting the sender information in a table).

The target hardware address is included for completeness and network monitoring. It has no meaning in the request form, since it is this number that the machine is requesting. Its meaning in the reply form is the address of the machine making the request. In some implementations (which do not get to look at the 14.byte ethernet header, for example) this may save some register shuffling or stack space by sending this field to the hardware driver as the hardware destination address of the packet.

There are no padding bytes between addresses. The packet data should be viewed as a byte stream in which only 3 byte pairs are defined to be words (arhrd, arpro and ar$op) which are sent most significant byte first (Ethernet/PDP-10 byte style).

Network monitoring and debugging:

The above Address Resolution protocol allows a machine to gain knowledge about the higher level protocol activity (e.g., CHAOS, Internet, PUP, DECnet) on an Ethernet cable. It can determine which Ethernet protocol type fields are in use (by value) and the protocol addresses within each protocol type. In fact, it is not necessary for the monitor to speak any of the higher level protocols involved. It goes something like this:

When a monitor receives an Address Resolution packet, it always enters the <protocol type, sender protocol address, sender hardware address> in a table. It can determine the length of the hardware and protocol address from the ar$hln and ar$pln fields of the packet. If the opcode is a REPLY the monitor can then throw the packet away. If the opcode is a REQUEST and the target protocol address matches the protocol address of the monitor, the monitor sends a REPLY as it normally would. The monitor will only get one mapping this way, since the REPLY to the REQUEST will be sent directly to the requesting host. The monitor could try sending its own REQUEST, but this could get two monitors into a REQUEST sending loop, and care must be taken.

Because the protocol and opcode are not combined into one field, the monitor does not need to know which request opcode is associated with which reply opcode for the same higher level protocol. The length fields should also give enough information to enable it to "parse" a protocol addresses, although it has no knowledge of what the protocol addresses mean.

A working implementation of the Address Resolution protocol can also be used to debug a non-working implementation. Presumably a hardware driver will successfully broadcast a packet with Ethernet type field of ether_type$ADDRESS_RESOLUTION. The format of the packet may not be totally correct, because initial implementations may have bugs, and table management may be slightly tricky. Because requests are broadcast a monitor will receive the packet and can display it for debugging if desired.

```
An Example:
-----------

Let there exist machines X and Y that are on the same 10Mbit
Ethernet cable.  They have Ethernet address EA(X) and EA(Y) and
DOD Internet addresses IPA(X) and IPA(Y) .  Let the Ethernet type
of Internet be ET(IP).  Machine X has just been started, and
sooner or later wants to send an Internet packet to machine Y on
the same cable.  X knows that it wants to send to IPA(Y) and
tells the hardware driver (here an Ethernet driver) IPA(Y).  The
driver consults the Address Resolution module to convert <ET(IP),
IPA(Y)> into a 48.bit Ethernet address, but because X was just
started, it does not have this information.  It throws the
Internet packet away and instead creates an ADDRESS RESOLUTION
packet with
        (ar$hrd) = ares_hrd$Ethernet
        (ar$pro) = ET(IP)
        (ar$hln) = length(EA(X))
        (ar$pln) = length(IPA(X))
        (ar$op)  = ares_op$REQUEST
        (ar$sha) = EA(X)
        (ar$spa) = IPA(X)
        (ar$tha) = don't care
        (ar$tpa) = IPA(Y)
and broadcasts this packet to everybody on the cable.

Machine Y gets this packet, and determines that it understands
the hardware type (Ethernet), that it speaks the indicated
protocol (Internet) and that the packet is for it
((ar$tpa)=IPA(Y)).  It enters (probably replacing any existing
entry) the information that <ET(IP), IPA(X)> maps to EA(X).  It
then notices that it is a request, so it swaps fields, putting
EA(Y) in the new sender Ethernet address field (ar$sha), sets the
opcode to reply, and sends the packet directly (not broadcast) to
EA(X).  At this point Y knows how to send to X, but X still
doesn't know how to send to Y.

Machine X gets the reply packet from Y, forms the map from
<ET(IP), IPA(Y)> to EA(Y), notices the packet is a reply and
throws it away.  The next time X's Internet module tries to send
a packet to Y on the Ethernet, the translation will succeed, and
the packet will (hopefully) arrive.  If Y's Internet module then
wants to talk to X, this will also succeed since Y has remembered
the information from X's request for Address Resolution.

Related issue:
--------------

It may be desirable to have table aging and/or timeouts.  The
implementation of these is outside the scope of this protocol.
Here is a more detailed description (thanks to MOON@SCRC@MIT-MC).

If a host moves, any connections initiated by that host will
work, assuming its own address resolution table is cleared when
```

it moves. However, connections initiated to it by other hosts will have no particular reason to know to discard their old address. However, 48.bit Ethernet addresses are supposed to be unique and fixed for all time, so they shouldn't change. A host could "move" if a host name (and address in some other protocol) were reassigned to a different physical piece of hardware. Also, as we know from experience, there is always the danger of incorrect routing information accidentally getting transmitted through hardware or software error; it should not be allowed to persist forever. Perhaps failure to initiate a connection should inform the Address Resolution module to delete the information on the basis that the host is not reachable, possibly because it is down or the old translation is no longer valid. Or perhaps receiving of a packet from a host should reset a timeout in the address resolution entry used for transmitting packets to that host; if no packets are received from a host for a suitable length of time, the address resolution entry is forgotten. This may cause extra overhead to scan the table for each incoming packet. Perhaps a hash or index can make this faster.

The suggested algorithm for receiving address resolution packets tries to lessen the time it takes for recovery if a host does move. Recall that if the <protocol type, sender protocol address> is already in the translation table, then the sender hardware address supersedes the existing entry. Therefore, on a perfect Ethernet where a broadcast REQUEST reaches all stations on the cable, each station will be get the new hardware address.

Another alternative is to have a daemon perform the timeouts. After a suitable time, the daemon considers removing an entry. It first sends (with a small number of retransmissions if needed) an address resolution packet with opcode REQUEST directly to the Ethernet address in the table. If a REPLY is not seen in a short amount of time, the entry is deleted. The request is sent directly so as not to bother every station on the Ethernet. Just forgetting entries will likely cause useful information to be forgotten, which must be regained.

Since hosts don't transmit information about anyone other than themselves, rebooting a host will cause its address mapping table to be up to date. Bad information can't persist forever by being passed around from machine to machine; the only bad information that can exist is in a machine that doesn't know that some other machine has changed its 48.bit Ethernet address. Perhaps manually resetting (or clearing) the address mapping table will suffice.

This issue clearly needs more thought if it is believed to be important. It is caused by any address resolution-like protocol.

Capítulo 19

GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

19.1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following

pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

19.2 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 19.3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

19.3 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than

100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

19.4 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 19.2 and 19.3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

3. State on the Title page the name of the publisher of the Modified Version, as the publisher.

4. Preserve all the copyright notices of the Document.

5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

8. Include an unaltered copy of this License.

9. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

19.5 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 19.4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections

with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

19.6 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

19.7 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 19.3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

19.8 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 19.4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the

translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 19.4) to Preserve its Title (section 19.1) will typically require changing the actual title.

19.9 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

19.10 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.