Protocolos TCP/IP para Sistemas Embarcados

Felipe Massia Pereira

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Protocolos TCP/IP para Sistemas Embarcados

Felipe Massia Pereira

Junho de 2003

Banca Examinadora:

- Prof. Dr. Célio Cardoso Guimarães (Orientador)
- Prof. Dr. Maurício Ferreira Magalhães
  Faculdade de Engenharia Elétrica e Computação – UNICAMP
- Prof. Dr. Paulo Lício de Geus
  Instituto de Computação – UNICAMP
- Prof. Dr. Paulo Cesar Centoducatte
  Instituto de Computação – UNICAMP

Substitua pela ficha catalográfica

Protocolos TCP/IP para Sistemas Embarcados

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Felipe Massia Pereira e aprovada pela Banca Examinadora.

Campinas, 25 de fevereiro de 2003.

Prof. Dr. Célio Cardoso Guimarães
(Orientador)

Prof. Dr. Nelson Castro Machado
(Co-orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Substitua pela folha com a assinatura da banca

Resumo

Nos últimos anos, assistiu-se a um aumento do poder computacional dos microprocessadores, enquanto seu preço e tamanho diminuíram drasticamente. Ao mesmo tempo, observou-se um crescente número de equipamentos conectados em rede, o que resultou na ampliação da infra-estrutura de comunicações. Estas duas tendências abriram caminho para novos cenários onde sistemas computacionais embarcados em dispositivos de uso específico são ligados a uma rede local e à Internet, principalmente para fins de monitoração e controle.

Neste trabalho são analisadas as soluções existentes para a utilização dos protocolos TCP/IP em sistemas embarcados. Tais sistemas são caracterizados por sérias restrições de memória e processamento, e por isso uma implementação específica da pilha se faz necessária. Também é apresentada uma implementação dos protocolos TCP/IP e de um servidor web para um sistema baseado no processador Z80.

Abstract

Over last years, we have been watching microprocessors' computational power increase, while their price and size have decreased drastically. At the same time, a growing number of equipments being connected to networks was observed, resulting in the communications infra-structure expansion. These two trends opened way to new scenarios where computer systems embedded in devices for specific use are connected to a local network and the Internet, mainly for monitoring and control.

This work analyzes the existing solutions for using TCP/IP protocols in embedded systems. Such systems are characterized by serious limitations of memory and processing, and therefore a specific implementation of the stack becomes necessary. Also, an implementation of the TCP/IP protocols and a web server for a system based on the Z80 processor is presented.

Agradecimentos

- ao Prof. Célio, pela excelente orientação, pelo total apoio, pela confiança e pela disponibilidade;
- ao Prof. Machado, pela valiosa orientação, pelo grande volume de material que forneceu e pelo contagiante entusiasmo sobre o projeto;
- ao CNPq;
- à minha família, que acompanhou de longe este trabalho;
- ao Guilherme, amigo e colega, sem o qual eu não teria chegado até aqui;
- aos colegas do Instituto, pela sempre alegre e motivadora companhia;
- aos amigos da Pastoral Universitária, do grupo Amor ao Próximo e da Igreja N. Sra. das Dores;
- aos amigos da república onde moro;
- aos funcionários do IC, pelo suporte;
- a Deus.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Objetivos

Os objetivos propostos no início deste trabalho foram:

1. Analisar pilhas TCP/IP para sistemas embarcados;
2. Integrar uma pilha TCP/IP a um sistema de 8 ou 16 bits;
3. Analisar servidores web para sistemas embarcados.

Na presente dissertação apresentamos as soluções de conectividade TCP/IP em sistemas embarcados. Descrevemos também o transporte de uma pilha TCP/IP e de um servidor web para o processador Z80, da Zilog.

A integração da pilha não foi possível devido a dificuldades na construção de um adaptador de rede acoplável ao sistema alvo. No entanto, essas dificuldades foram contornadas através de uma simulação.

1.2 Motivação

Nos últimos anos, assistiu-se a um aumento do poder dos microprocessadores enquanto seu preço e tamanho diminuíram drasticamente [BW00, Tur99]. Ao mesmo tempo, observou-se um crescente número de equipamentos conectados em rede, ampliando a infra-estrutura de comunicações. Em especial, destaca-se a tecnologia Ethernet e a rede global Internet, hoje quase onipresentes [Web98].

Esses dois crescimentos abriram caminho para novos cenários onde os dispositivos possuem um servidor web embarcado [Web99]. Servidores web são relativamente simples, pois precisam apenas responder ao cliente transferindo os arquivos solicitados; a construção da página gráfica fica a cargo do cliente. Com a utilização de servidores web em dispositivos, é possível ter uma interface rica a um baixo custo, abandonando os LEDs, displays de LCD[1] e teclados [Qui97].

Pode-se imaginar, por exemplo, que vários termômetros localizados em pontos distantes de uma fábrica possam ser monitorados através de uma página web. Navegadores[2] possuem uma interface simples e genérica e são muito fáceis de operar exigindo pouco treinamento. Além disso, eles são gratuitos e estão disponíveis para qualquer sistema operacional. Também é desejável, por exemplo, controlar um motor através de um mecanismo semelhante utilizando scripts CGI[3]. O controle pode ser exercido a partir da própria fábrica ou até mesmo do outro lado do mundo.

Para que isso seja possível, é necessário que esses dispositivos sejam interligados à infra-estrutura de rede existente minimizando-se o custo financeiro para tal. A fim de minimizar o custo financeiro, é preciso minimizar o uso de memória RAM[4] e otimizar o poder de processamento. A economia de energia também passa a ser uma preocupação quando consideramos dispositivos *wireless*.

1.3 Trabalhos relacionados

Dentre os trabalhos acadêmicos relacionados ao assunto, vale destacar o de Dunkels e o de Padrão.

Dunkels [Dun01b] idealizou uma arquitetura de comunicação baseada em *proxy* para aliviar a carga de software sobre os sistemas embarcados. O *proxy* é um nó intermediário entre a Internet e a rede *wireless*. Na rede *wireless*, os nós rodam uma implementação reduzida da pilha TCP/IP. O *proxy* se encarrega de adequar os pacotes e conexões que fluem entre a Internet e a rede *wireless*. Por exemplo, como a pilha que roda nos nós wireless não aceita pacotes IP fragmentados, é função do *proxy* guardar e ordenar esses pacotes provenientes da Internet para entregá-los aos nós *wireless*. A pilha reduzida se chama lwIP e será descrita na seção 3.1.1.

[1] *Liquid Crystal Display.*
[2] Em Inglês, *browsers.*
[3] *Common Gateway Interface.*
[4] *Random Access Memory.*

Padrão [dCP99] desenvolveu um sistema de gerenciamento de UPS[1] baseado em SNMP[2]. O agente SNMP reside em um sistema embarcado dentro do UPS. Neste trabalho foi utilizada uma pilha comercial, a Fusion TCP/IP, descrita na seção 3.1.8.

Também foram encontradas na rede várias soluções de software e hardware para o problema da conectividade de sistemas embarcados através do protocolo TCP/IP. Essas soluções são descritas no capítulo 3.

1.4 Código aberto

Este trabalho teve como princípio o código aberto[3]. Isso significa que procuramos priorizar nesta análise sistemas que seguem essa filosofia.

Neste trabalho, o termo "código aberto" qualifica um pedaço de código que pode ser lido, redistribuído e modificado, desde que mantido o devido crédito ao autor original, e que pode ser embarcado em produtos comerciais sem necessidade de pagamento de *royalties*.

Os compiladores C utilizados na implementação foram o SDCC[4] e o GCC[5], ambos distribuídos sob os termos da GNU GPL[6].

1.5 Organização

Esta dissertação é composta dos seguintes capítulos:

Capítulo 2 Conceitos sobre sistemas embarcados e TCP/IP

Capítulo 3 Análise do que existe na área

Capítulo 4 Implementação

Capítulo 5 Conclusões

[1] *Uninterruptible Power Systems.*
[2] *Simple Network Managemente Protocol.*
[3] Em Inglês, *Open source.*
[4] *Small Devices C Compiler.* O SDCC será detalhado na seção 4.
[5] *GNU Compiler Collection.*
[6] *General Public License.*

Capítulo 2

Fundamentos

2.1 TCP/IP

O TCP e o IP são os dois protocolos base para o funcionamento da Internet. Além destes dois, vários outros protocolos constituem o que chamamos de "conjunto de protocolos TCP/IP" ou simplesmente "TCP/IP". Devido à maneira como são normalmente implementados, os protocolos também são chamados de "pilha TCP/IP", nome que utilizamos com maior freqüência neste trabalho.

Nesta seção abordamos este conjunto de protocolos. Grande parte do material é baseado em [Ste94], [WS95], [Tan96], [Com95a] e nas RFCs[1] que serão citadas oportunamente.

2.1.1 História

A Internet teve origem no fim da década de 60, quando o governo norte-americano encarregou a ARPA[2] de projetar e construir uma rede de computadores. Esta rede deveria ser robusta a ponto de continuar funcionando mesmo que um dos seus pontos parasse.

Após um tempo de pesquisas, a ARPA decidiu que a rede deveria ser chaveada por pacotes – um conceito novo na época – e formada de duas partes: uma *sub-rede* e os *hosts*. Para a construção da sub-rede, a ARPA contratou uma empresa de consultoria. A implementação do software de comunicação host-a-host foi feita por pesquisadores da área, a maioria estudantes de pós-graduação.

[1] RFC – acrônimo de *Request for Comments*. É um conjunto de documentos que contém notas técnicas sobre a Internet.

[2] *Advanced Research Projects Agency*.

A ARPANET entrou no ar em 1969 com quatro hosts. Em poucos anos a rede já tinha se espalhado pelo território dos EUA. A necessidade de um protocolo que facilitasse a intercomunicação entre redes motivou pesquisas na área, que resultaram na invenção do TCP/IP por Cerf e Kahn em 1974 [CK74]. O novo protocolo foi incluído na versão 4.2 do BSD, juntamente com a interface de programação *sockets* e vários utilitários, e foi adotado imediatamente. Em 1983, o TCP/IP se tornou o protocolo oficial e único da ARPANET. Nesta época o número de hosts era de quase 1000.

2.1.2 Implementação em Camadas

Para reduzir a complexidade, a maior parte das redes são organizadas em camadas ou níveis. O objetivo de cada camada é oferecer serviços à camada acima, ocultando os detalhes de implementação destes serviços. A comunicação entre máquinas se dá entre as camadas de mesmo nível, ou seja, a camada n de uma máquina comunica-se com a camada n de outra máquina. Esta comunicação se baseia em regras e convenções pré-definidas às quais chamamos protocolos. O conjunto de protocolos de um sistema é chamado de pilha de protocolos.

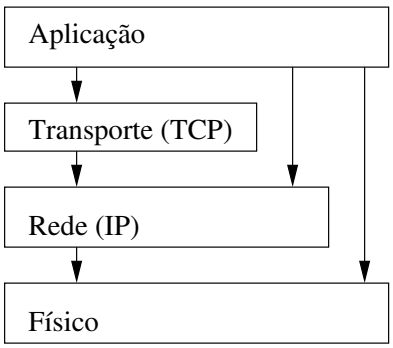

Figura 2.1: Camadas nos protocolos TCP/IP.

Entre as camadas dentro de uma mesma máquina existem as interfaces. A interface define quais serviços uma camada oferece à camada superior. A especificação das interfaces deve ser clara e deve minimizar a quantidade de informação passada entre as camadas.

Considerações de desempenho

Implementações TCP/IP para sistemas embarcados devem levar em consideração as sérias limitações de memória e processamento da arquitetura alvo. Além disso, muitas vezes não se pode contar com o suporte de um sistema operacional e suas funcionalidades, tais como gerenciamento de memória, processos e *threads*, sistema de arquivos, etc.

A RFC 817 trata da modularidade e do desempenho de uma implementação TCP/IP [Cla82]. Também [Kes97] e [Ben00] dedicam capítulos em seus livros sobre implementação e desempenho de protocolos.

Se a implementação for rígida na separação das camadas – não permitindo que elas compartilhem variáveis, por exemplo – a troca de informações entre as camadas pode degradar o desempenho. Pode-se diminuir o volume dessa troca relaxando um pouco a separação entre as camadas, permitindo que compartilhem certas informações.

A maioria das implementações de TCP/IP mantém uma fronteira bem definida entre a camada de aplicação e as camadas mais baixas. Na maioria dos sistemas operacionais, os protocolos de nível mais baixo são implementados junto ao kernel, enquanto as aplicações são implementadas como processos de usuário.

Nos sistemas operacionais utilizados em sistemas embarcados, as preocupações com quantidade de memória e desempenho antecedem as preocupações com interfaces limpas e estruturadas. Geralmente as implementações de pilhas de protocolos para sistemas embarcados também seguem esse princípio. A separação estrita de módulos é ainda mais dispensável nos sistemas pequenos dedicados a uma única função.

2.1.3 IP – *Internet Protocol*

O IP é o protocolo da camada de rede da Internet. Ele é responsável por transmitir datagramas de uma rede para outra, dentro da Internet. Cada host é ligado à Internet por uma ou mais interfaces de rede. Cada interface possui um endereço IP único, através do qual envia e recebe datagramas IP.

O IP não oferece garantia de entrega e não é orientado a conexão. Nenhuma informação de estado é mantida e os datagramas podem chegar fora de ordem.

A versão do protocolo IP mais utilizada na Internet atualmente é o IP versão 4 ou simplesmente IPv4. Seu sucessor, o IPv6, já foi projetado e há várias implementações disponíveis. Ele é compatível com a versão anterior e já vem sendo utilizado em menor

escala. Neste trabalho, analisamos apenas a versão 4[1].

Um datagrama IP é composto de um cabeçalho seguido dos dados conforme mostra a figura 2.2.

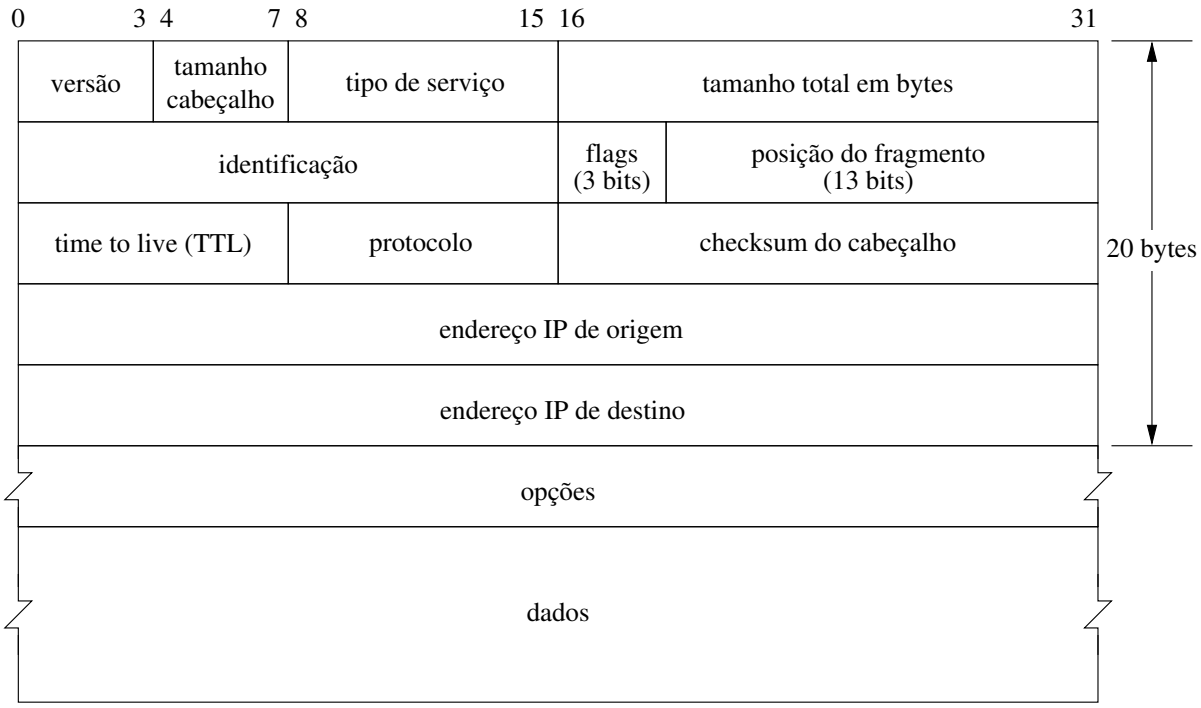

Figura 2.2: Formato de um datagrama IP.

O campo *tamanho do cabeçalho* é o número de palavras de 32 bits no cabeçalho, incluindo as opções. O campo *tipo de serviço* é composto de 3 bits de precedência, 4 bits de tipo de serviço e um bit não-utilizado. Os 4 bits de tipo de serviço são:

- minimizar atraso
- maximizar *throughput*
- maximizar confiabilidade
- minimizar custo financeiro

Através desse campo, o roteamento IP se adequa às necessidades de aplicações diferentes. O campo *tamanho total* é o tamanho total do pacote em bytes. O campo

[1]A versão 5 foi um protocolo experimental.

identificação serve para identificar unicamente cada datagrama enviado por um host e é utilizado principalmente pelas rotinas de fragmentação.

O campo *TTL* limita o tempo de vida do pacote. Na prática ele é usado como um número máximo de *hops*, ou seja, por quantos roteadores o pacote poderá passar. Ao passar por um roteador, o TTL é decrementado de um. Caso o valor chegue a zero, o pacote é descartado e uma notificação é enviada ao remetente.

O campo *protocolo* diz a qual protocolo de nível superior pertencem os dados do pacote.

O *checksum do cabeçalho* é um *checksum* calculado sobre o cabeçalho IP apenas, não incluindo os dados.

Os campos *endereço IP de origem* e *endereço IP de destino* identificam o remetente e o receptor, respectivamente.

Checksum

Um *checksum* serve para o receptor verificar a integridade dos dados. Seu valor é calculado no momento do envio pelo transmissor. Vários protocolos utilizam *checksums* como método de detecção de erros na transmissão, dentre eles o IP, o ICMP, o UDP e o TCP.

O cálculo do checksum é um dos procedimentos que mais influi no desempenho da pilha, visto que deve operar sobre cada byte transferido. Por isso ele é geralmente implementado em código de máquina. Outra técnica utilizada para melhorar o desempenho do cálculo, é realizá-lo durante a cópia dos dados de ou para a memória. Dessa forma um laço é economizado.

A RFC 1071 [BBP88] descreve técnicas de implementação do cálculo de *checksum* para arquiteturas de 8 a 32 bits, *little-endian* ou *big-endian*. A RFC 1141 [MK90] descreve como atualizar o *checksum* de forma incremental, permitindo que pequenas mudanças sejam feitas no cabeçalho IP sem que seja necessário recalcular o *checksum* sobre todo o cabeçalho.

Opções IP

O campo *opções* é uma lista de tamanho variável que pode conter as seguintes opções:

- restrições de segurança e manipulação do pacote (para aplicações militares);
- solicitar para cada roteador registrar seu IP;
- solicitar para cada roteador registrar seu IP e o momento no qual o pacote passou;

- especificar uma lista de roteadores pelos quais o pacote deve passar.

Algumas pilhas de protocolos pequenas, usadas em sistemas embarcados, normalmente não implementam as opções. Neste caso, pacotes com opções são simplesmente descartados.

Fragmentação IP

No modelo TCP/IP, entre dois pontos de comunicação pode haver uma rede formada por várias sub-redes, nas quais o tamanho máximo de uma unidade de transmissão (MTU) pode variar. Para contornar o problema e transmitir pacotes maiores que o MTU de algumas redes, roteadores IP podem dividir os pacotes IP em pacotes menores. O processo se chama fragmentação e os pacotes menores são chamados de fragmentos. O processo inverso é chamado de remontagem.

Como o IP é não-confiável, um fragmento perdido implica na perda do pacote inteiro. O pacote IP só é repassado para a camada superior quando chega por completo ao destino.

A remontagem de pacotes IP é muito custosa em termos de memória e processamento. É preciso manter na memória os fragmentos de um pacote por um certo tempo, pois não se pode prever se o restante do pacote ainda vai chegar ou se perdeu.

2.1.4 ARP – *Address Resolution Protocol*

No nível mais baixo da pilha de protocolos, os datagramas IP são manipulados por interfaces físicas de rede, como, por exemplo, Ethernet e Token Ring. Estas possuem um esquema de endereçamento próprio, diferente do endereçamento IP. O ARP é o protocolo que traduz os endereços IP para endereços de interface, também ditos endereços de enlace. O ARP é comumente utilizado para mapear endereços IP em endereços Ethernet, embora seja genérico a ponto de possibilitar o mapeamento entre dois tipos quaisquer de endereçamento.

As principais mensagens do protocolo ARP são `ARP REQUEST` e `ARP REPLY`. Quando um host deseja saber o endereço de enlace correspondente a um endereço IP de sua rede, ele envia `ARP REQUEST` em broadcast. A máquina identificada pelo endereço IP da solicitação, envia então um `ARP REPLY` contendo seu endereço de enlace. Essas duas mensagens possuem o mesmo formato, que é exibido na figura 2.3, juntamente com o cabeçalho Ethernet.

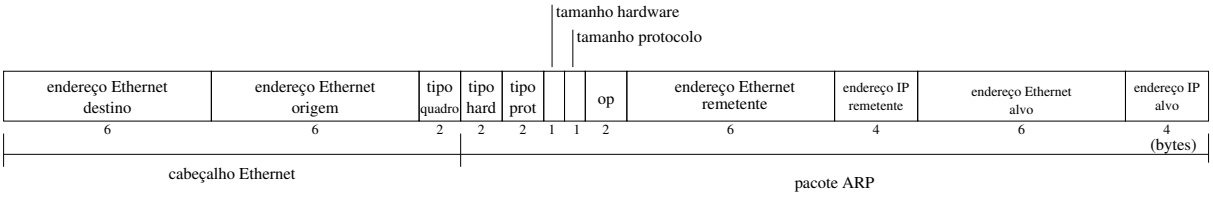

Figura 2.3: Formato de ARP request e reply e o cabeçalho Ethernet.

Os dois primeiros campos do cabeçalho Ethernet identificam o host destino e o remetente. O campo *tipo quadro* indica o tipo de dado encapsulado no quadro Ethernet.

Os campos seguintes pertencem ao pacote ARP. Os campos *tipo hard* e *tipo prot* identificam os tipos de endereçamento que se quer mapear. Os campos *tamanho hard* e *tamanho prot* contêm o tamanho em bytes de cada tipo de endereçamento. No caso do mapeamento de IP para Ethernet, tamanho hard é 6 e tamanho prot é 4. O campo *op* indica se o pacote é um request ou um reply. Os campos seguintes contêm os endereços de hardware e de protocolo para a máquina remetente e a máquina alvo.

Visto que os endereços IP das máquinas mudam raramente, a pilha TCP/IP mantém um cache de ARP. Cada entrada é composta de um endereço IP, o endereço de hardware correspondente e um campo que indica o tempo de duração daquela entrada. Normalmente, as entradas do cache de ARP expiram em 20 minutos, quando então nova mensagem `ARP REQUEST` é enviada.

2.1.5 ICMP – *Internet Control Message Protocol*

O protocolo ICMP é responsável por comunicar mensagens de erro e outras situações adversas. As mensagens ICMP são enviadas em datagramas IP. O formato genérico para mensagens ICMP é exibido na figura 2.4. Os campos *tipo*, *código* e *checksum* são comuns a todas as mensagens. O restante varia de acordo com o tipo e o código da mensagem.

As principais mensagens ICMP são descritas a seguir:

`Echo request` e `Echo reply` Utilizadas para verificar se um host está ativo. Quando um host recebe uma mensagem de `Echo request`, ele deve responder com uma mensagem `Echo reply`. O formato dessas duas mensagens, exibido na figura 2.5 é idêntico, variando apenas no valor do campo tipo, que é 0 para `reply` e 8 para `request`.

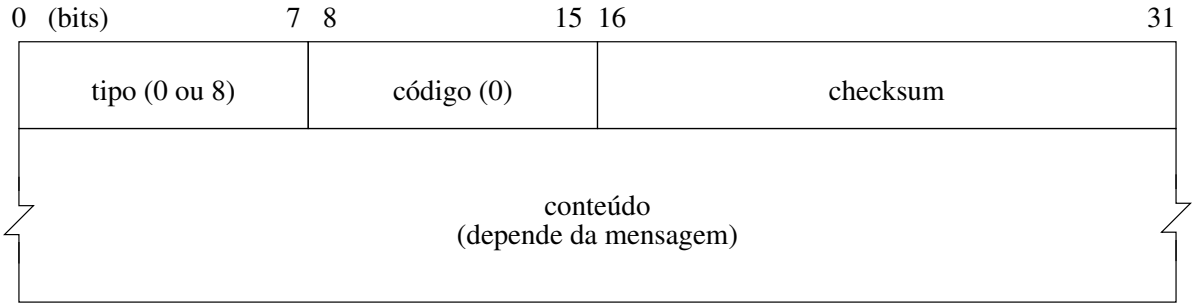

Figura 2.4: Formato genérico de uma mensagem ICMP.

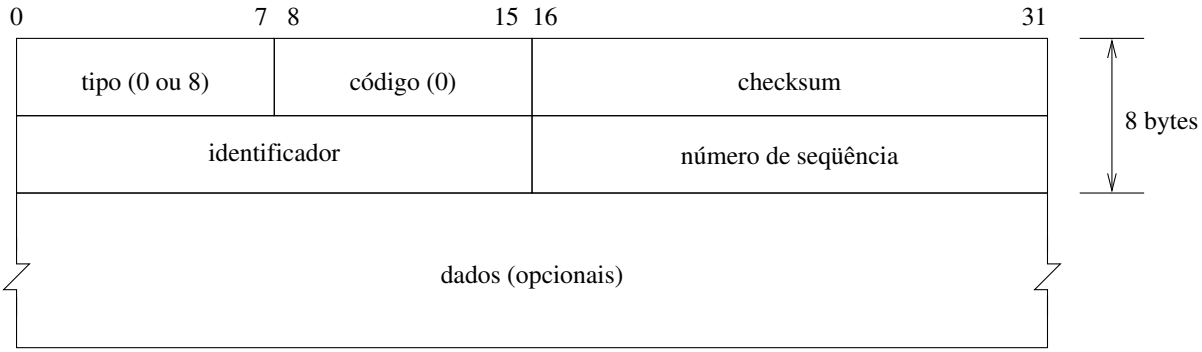

Figura 2.5: Formato das mensagens ICMP ECHO REQUEST e ECHO REPLY.

`Destination unreachable` Indica ao remetente que um datagrama enviado anteriormente não chegou ao seu destino. Isso pode acontecer, dentre outros motivos, porque o host destino está inativo, porque o datagrama é maior que o tamanho máximo suportado pela rede, ou porque não há nenhuma aplicação capaz de processar aquele datagrama.

`Source quench` Essa mensagem era utilizada para controle de congestionamento. Está em desuso.

`Redirect` Usada por roteadores para indicar a rota correta de um pacote.

`Time exceeded` Mensagem enviada quando um pacote é descartado porque seu TTL atingiu zero.

2.1.6 UDP – *User Datagram Protocol*

O UDP é um protocolo não orientado a conexão e não-confiável que transmite seus dados através de datagramas IP. Enquanto o endereço IP apenas especifica a interface de um host na Internet, com o UDP torna-se possível endereçar um processo sendo executado num host. O cabeçalho UDP é mostrado na figura 2.6.

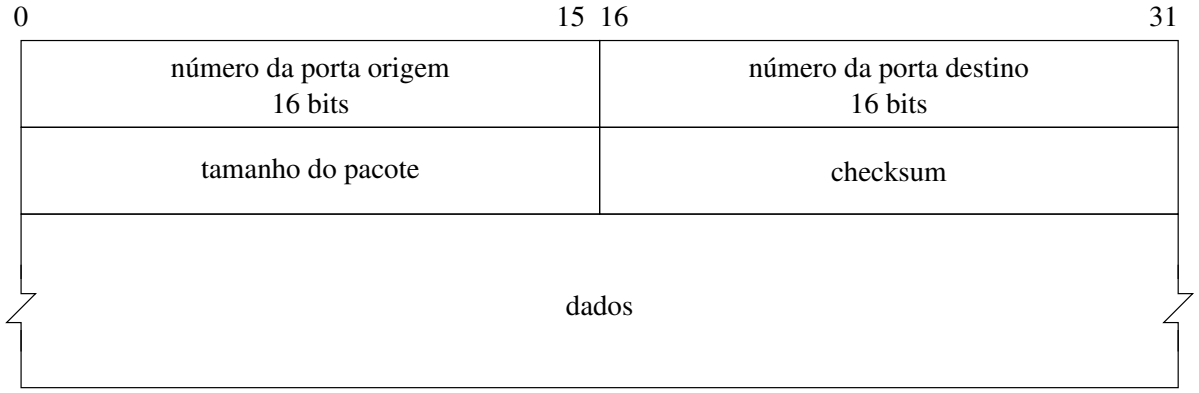

Figura 2.6: Formato do cabeçalho UDP.

Os campos *porta origem* e *porta destino* identificam o processo remetente e o processo receptor, respectivamente. O campo *tamanho do pacote* é o tamanho em bytes do datagrama UDP, ou seja, a soma dos tamanhos do cabeçalho e dos dados.

O checksum do UDP é calculado de maneira similar ao checksum do cabeçalho IP, conforme visto na seção 2.1.3. No entanto o checksum UDP cobre não só o cabeçalho, mas também os dados. Além disso, o UDP inclui um pseudo-cabeçalho e bytes no valor 0 de "enchimento" (*padding*) no cálculo do checksum.

O pseudo-cabeçalho consiste dos endereços IP da origem e do destino, o protocolo e o tamanho do datagrama UDP. Este último se repete no cabeçalho UDP e os outros se repetem no cabeçalho IP. A inclusão repetida desses campos no cálculo do checksum tem por objetivo certificar de que o datagrama chegou ao destino correto. Os bytes de enchimento são utilizados em datagramas que possuem um tamanho ímpar (tamanhos não múltiplos de 16 bits).

2.1.7 TCP – *Transmission Control Protocol*

O TCP é de longe o protocolo mais complexo da suíte que leva seu nome. Transmitindo dados encapsulados em datagramas IP, ele oferece confiabilidade na entrega dos dados e controle de fluxo, além de ser orientado a conexão. Para as camadas acima do TCP, os dados aparecem como uma seqüência de bytes. Assim como o UDP, também provê um meio para endereçar um determinado processo de um host através de números de portas. O protocolo TCP está definido na RFC 793 [Pos81].

A figura 2.7 ilustra o formato de um segmento TCP. Um segmento é uma unidade lógica de dados transferida entre um par de módulos TCP.

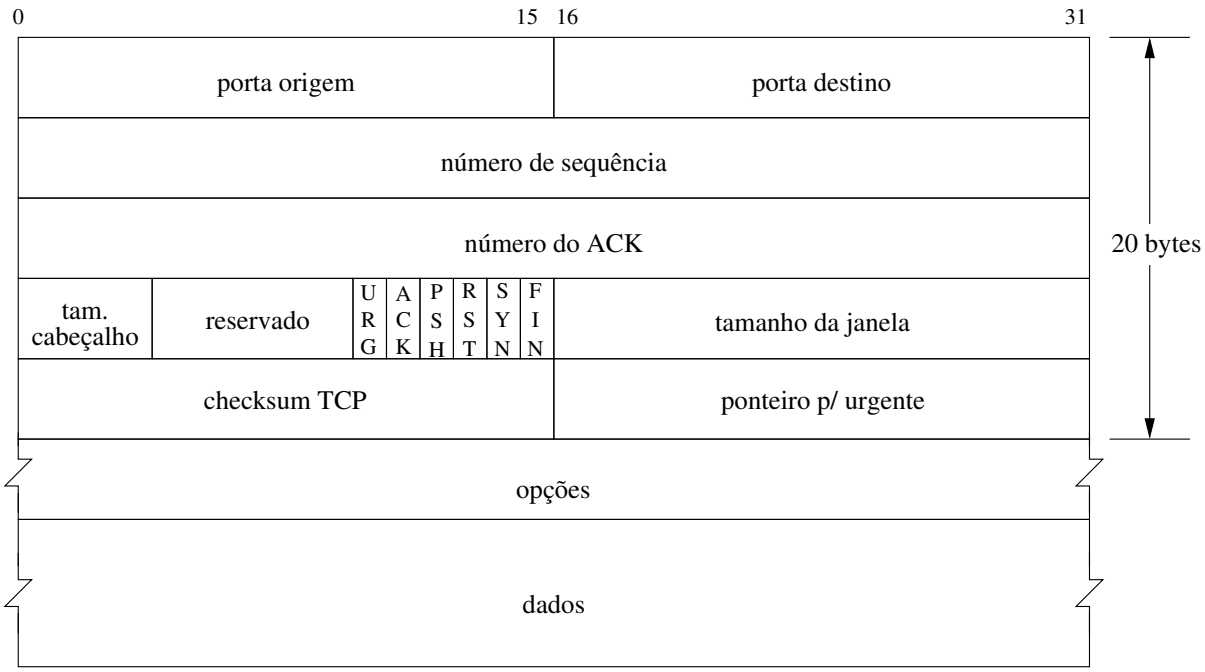

Figura 2.7: Formato do segmento TCP.

Os dois primeiros campos, porta de origem e porta de destino, identificam os processos comunicantes. A quádrupla formada por estes dois números de porta mais os números IP de origem e destino contidos no cabeçalho IP identificam uma conexão.

Cada byte transmitido no TCP tem um número de seqüência. O número de seqüência no cabeçalho TCP corresponde ao do primeiro byte do segmento. O campo *número do ACK* identifica o próximo número de seqüência sendo esperado por quem envia o ACK.

O campo *tamanho do cabeçalho* indica o tamanho do cabeçalho em palavras de 32 bits. Como o campo *opções* é de tamanho variável, o tamanho do cabeçalho também varia.

O flag SYN é utilizado durante a abertura de conexão e o flag FIN, no fechamento. O flag ACK indica que o número do ACK é válido. O flag RST é utilizado para reiniciar uma conexão, geralmente indicando uma situação adversa. O flag PSH, de *push*, indica que os dados devem ser repassados pelo receptor à aplicação tão logo possível. O flag URG indica se o ponteiro urgente é válido.

O campo *tamanho da janela* indica ao remetente quantos bytes podem ser transmitidos sem a espera de uma confirmação. Esse campo pode ser multiplicado por um fator especificado nas opções.

O *checksum* do TCP cobre todos os dados e é calculado de maneira similar ao do UDP, com um pseudo-cabeçalho. O *ponteiro urgente* indica um deslocamento relativo ao número de seqüência do segmento que aponta para o último byte de dados urgentes.

Estabelecimento e fechamento de conexões

O estabelecimento de conexões no TCP envolve três passos. Supondo que o host cliente deseja abrir uma conexão com o host servidor, o processo se dá como segue:

1. O cliente envia um segmento com o flag SYN ligado, contendo o seu número de seqüência inicial x, para comunicação no sentido cliente–servidor.

2. O servidor responde enviando um segmento com o flag SYN ligado, contendo o seu próprio número de seqüência inicial y, para comunicação no sentido servidor–cliente, e o valor $x + 1$ no campo ACK do cabeçalho TCP.

3. O cliente confirma o recebimento do SYN do servidor enviando um segmento com o valor $y + 1$ no campo ACK.

A figura 2.8 ilustra esta seqüência de conexão que é chamada de *three-way handshake*. Diz-se que o servidor executou uma abertura passiva e que o cliente fez uma abertura ativa.

O fechamento de conexões TCP pode ser parcial ou completo. No parcial, o fluxo de dados é terminado em uma direção apenas, enquanto no completo o fluxo é terminado em ambas as direções. Utilizando o fechamento parcial, uma aplicação pode configurar uma conexão de apenas escrita ou de apenas leitura. Esta modalidade também é chamada *half close*.

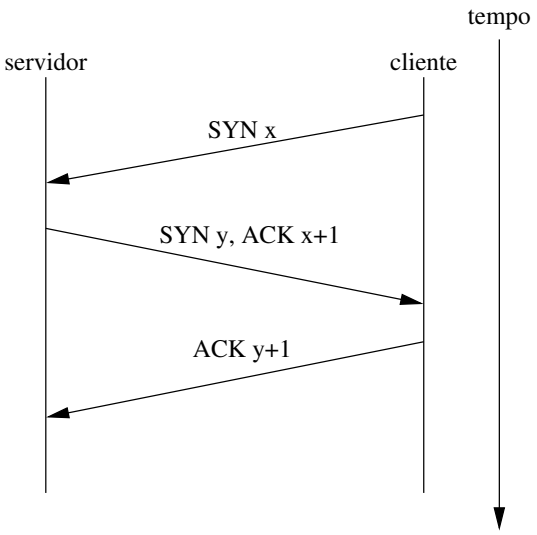

Figura 2.8: Estabelecimento de conexão TCP.

Supondo que o cliente queira fechar uma conexão estabelecida com o servidor:

1. No cliente, a aplicação fecha a conexão e a pilha envia um segmento com o flag FIN ligado. Este segmento pode conter dados.

2. O servidor reconhece o segmento FIN do cliente enviando um ACK. Um aviso de fim-de-arquivo é entregue à aplicação servidora.

3. Quando a aplicação servidora fecha a conexão, um segmento FIN é enviado ao cliente.

4. O cliente reconhece o FIN do servidor enviando um segmento com o flag ACK ligado.

Diz-se que o cliente efetuou um fechamento ativo enquanto o servidor efetuou um fechamento passivo. A figura 2.9 ilustra o processo de fechamento.

O protocolo TCP diz que após efetuar o fechamento ativo de uma conexão, a mesma deve ser mantida no estado `TIME-WAIT` por um determinado tempo. O objetivo é garantir que pacotes destinados à conexão atual não sejam aceitos por novas conexões nesta mesma porta. A duração deste estado é definido na RFC como o dobro do MSL[1], que é o tempo máximo de sobrevivência de um pacote na rede. Por causa disso, o estado `TIME-WAIT`

[1] *Maximum Segmente Lifetime.*

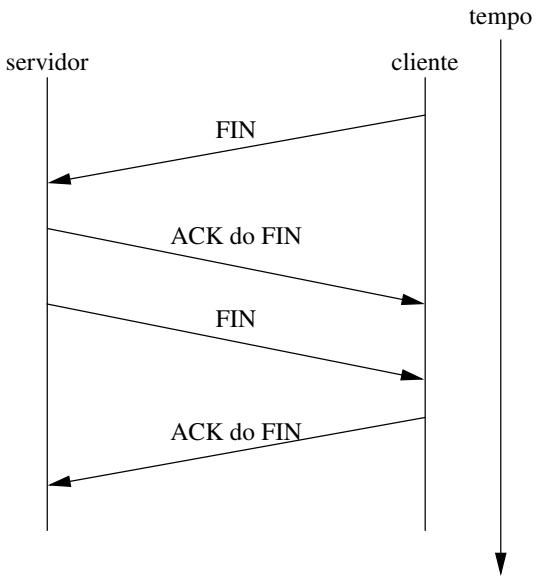

Figura 2.9: Fechamento de conexão TCP.

também é chamado de 2MSL. O MSL varia com as implementações, mas a RFC sugere um tempo de 2 min.

Esta regra do TCP nem sempre é seguida à risca. Nos sistemas embarcados, o número de conexões simultâneas é limitado devido à memória reduzida. Algumas pilhas reutilizam uma conexão mesmo antes do tempo 2MSL ter passado para aquela conexão.

Retransmissão

A fim de oferecer confiabilidade às camadas superiores, o TCP mantém informações de estado dos pacotes enviados, como momento do envio e se ele já foi confirmado. Uma rotina que é iniciada periodicamente por um temporizador percorre a lista de conexões e verifica se há pacotes que foram enviados para os quais a confirmação já deveria ter chegado. Os pacotes que se encaixam nesse perfil são reenviados.

Manter informações de estado significa mais memória RAM, a qual é escassa nos sistemas embarcados, além de ser cara. Pilhas pequenas como a uIP, não mantêm informações de estado ou mantêm informações reduzidas a fim de minimizar os gastos com memória RAM.

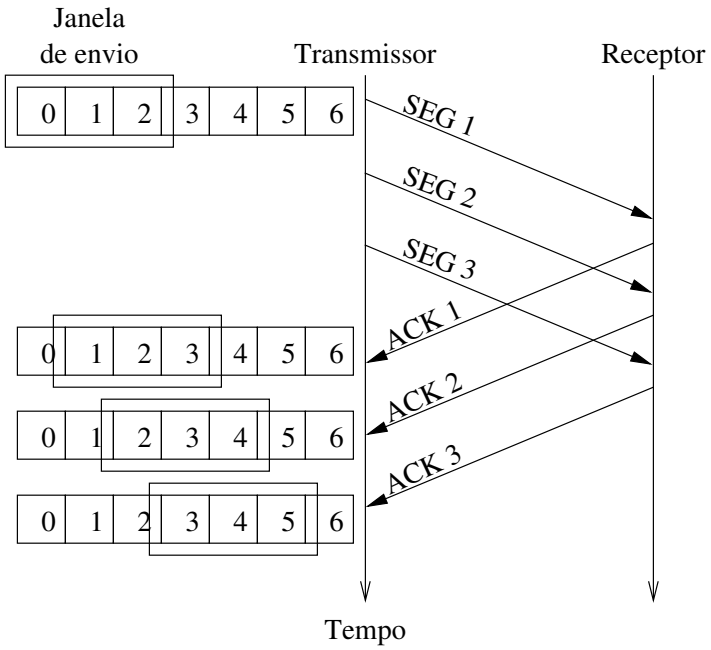

Figura 2.10: Transmissão com janelas deslizantes.

Janelas deslizantes

O TCP utiliza janelas deslizantes para tranferência massiva de dados. Significa que o remetente não espera receber o ACK de um segmento para então enviar o próximo, conforme mostrado na figura 2.10. Esse método aumenta bastante o *throughput*.

A figura ilustra uma transmissão com janela fixa de 3 segmentos. No TCP o tamanho da janela pode variar. O remetente envia os segmentos 2 e 3 mesmo sem ter recebido uma confirmação (ACK) do segmento 1, pois a janela assim o permite. Após receber as confirmações o remetente pode então enviar os segmentos de 4 a 6.

Confiabilidade

O TCP foi desenvolvido para oferecer confiabilidade operando sobre um conjunto de redes sobre as quais quase nada pode supor. Talvez o maior mérito do protocolo seja a eficiência com que trata a perda de pacotes e os timeouts para aumentar o throughput da transmissão.

Para tal, foram desenvolvidos os seguintes algoritmos, incluídos no protocolo TCP ao longo do tempo:

Medida de RTT O RTT[1] é o tempo entre o envio de um pacote e o recebimento da resposta ao mesmo. Este tempo pode variar bastante, devido à congestionamento da rede e à perda de pacotes. Ele influi diretamente na determinação do timeout para retransmissão de um pacote, o qual é decisivo para o *throughput*. Um timeout muito curto pode levar a retransmissões mesmo quando o pacote chega seguramente ao destino, ocupando a rede desnecessariamente. Um timeout muito longo causa esperas mais longas, também desnecessárias, na ocasião da perda de um pacote. O TCP possui um algoritmo eficiente de estimativa do RTT, idealizado por Jacobson [JBB92].

Slow Start Este algoritmo visa evitar que um receptor seja inundado pelo transmissor. Basicamente, o transmissor não começa enviando todos os segmentos que a janela do receptor permite. Em vez dissso, a janela de envio é inicializada em 1 e incrementada para cada ACK recebido. Isso leva a um crescimento exponencial, que vai parar ao atingir um limiar determinado pelo algoritmo de prevenção de congestionamento, que detalhamos a seguir.

Prevenção de congestionamento Quando um pacote é perdido ou um ACK duplicado é recebido, é sinal de que a rede está congestionada e que o limite da rede foi atingido. Neste momento entra em ação este outro algoritmo, o de prevenção de congestionamento. Ele trabalha conjuntamente com o slow start para evitar o afogamento da rede. Quando este algoritmo está executando o crescimento da janela passa a ser linear. Quando há perda de segmentos na rede, o slow start é iniciado novamente.

Retransmissão rápida e Recuperação rápida Quando três ou mais ACKs duplicados são recebidos é um bom indício de que houve perda de um segmento. O algoritmo de retransmissão rápida diz que nestas ocasiões o TCP deve reenviar o segmento que provavelmente é o perdido.

Normalmente também deveria iniciar o algoritmo de slow start. Mas como foram recebidos ACKs duplicados, significa que provavelmente apenas aquele segmento foi perdido. Então o algoritmo de prevenção de congestionamento é iniciado. Esse algoritmo é chamado recuperação rápida.

[1] *Round Trip Time.*

Estes algoritmos, porém, são complicados, adicionam complexidade ao código e demandam recursos excessivos de processamento e memória. A maioria das pilhas de sistemas embarcados menores não os implementam.

2.2 Servidores web

A *World Wide Web*, ou apenas web, é o serviço mais utilizado na Internet. Sua popularidade se deve à facilidade de uso e à quantidade de informações disponível.

O protocolo utilizado por clientes e servidores web é o HTTP[1]. O protocolo HTTP está na versão 1.1, definido na RFC 2616 [FGM+99]. A versão 1.1 é compatível com sua predecessora, a versão 1.0. Assim, solicitações feitas por navegadores antigos ainda podem ser respondidas por servidores mais novos. Em sistemas embarcados são mais comuns as implementações de servidores da versão 1.0, por ser mais simples. Uma das diferenças entre as versões é que na versão 1.1 é possível reutilizar a mesma conexão TCP para solicitar mais arquivos, enquanto na versão 1.0 uma conexão é necessária para cada transferência.

Enquanto nos sistemas embarcados deseja-se servidores web reduzidos, que consumam poucos recursos, as implementações mais difundidas, como o servidor Apache, por exemplo, focalizam em outros aspectos como desempenho, robustez e segurança. Alguns servidores web especialmente desenvolvidos para sistemas embarcados são revisados no capítulo 3.

É altamente desejável que servidores web em sistemas embarcados sejam capazes de gerar conteúdo dinâmico, a fim de permitir o monitoramento e o controle remotos. Acerca disso, as seguintes definições são importantes:

SSI – *Server Side Includes* são diretivas inseridas dentro do código HTML que são avaliadas e substituídas no momento em que as páginas estão sendo servidas. A vantagem em relação ao CGI é que o conteúdo dinâmico pode ser gerado sem ter que gerar a página inteira. Em sistemas embarcados pode ser utilizado para inserir numa página a temperatura lida a partir de um termômetro ligado ao sistema, por exemplo.

CGI – *Common Gateway Interface* é um padrão de interfaceamento entre servidores de informação e aplicações externas. Normalmente uma página estática é servida por

[1] *HyperText Transfer Protocol.*

um servidor web. Com CGI é possível que essa página seja gerada por uma aplicação executada pelo servidor no momento da solicitação. Essa aplicação pode ser escrita em qualquer linguagem que possa gerar um executável do sistema.

Normalmente aplicações CGI ficam em um diretório chamado cgi-bin. Quando o servidor web recebe uma solicitação de uma URL que aponta para uma aplicação CGI, essa URL é interpretada de forma especial. O sinal de '?' demarca o início da *query string*. Ela será passada como argumento de linha de comando para a aplicação CGI.

A aplicação deve retornar um documento precedido por um cabeçalho indicando o tipo de conteúdo MIME sendo retornado. Exemplos de tipos de conteúdo MIME são "text/html" para uma página HTML e "image/jpeg" para uma imagem no formato JPEG.

2.3 Sistemas embarcados

Um sistema embarcado pode ser definido como um sistema de hardware e de software integrado a um outro sistema dedicado a uma tarefa específica, que não tem a computação como fim. Podemos encontrar sistemas embarcados em vários utensílios domésticos tais como vídeo-cassete, TV e aparelho de som. Também os encontramos em automóveis, quiosques de informação, caixas eletrônicos, terminais de cartão magnético, celulares, faxes, copiadoras, máquinas fotográficas, câmeras, etc. Segundo [Tur99], os processadores utilizados em sistemas embarcados representam mais de 99% do total de processadores no mundo.

Nesta seção apresentamos tecnologias de memória e comunicação comumente utilizados em sistemas embarcados.

2.3.1 Memória

Esta seção descreve tecnologias de memória utilizadas em projetos de sistemas embarcados:

ROM – *Read Only Memory* É uma memória não-volátil de apenas leitura. Só podem ser programadas uma vez, mas são muito baratas quando compradas em grandes quantidades, duráveis e operam com pouca energia.

PROM – *Programmable ROM* A criação de chips de ROM é um processo demorado e caro para pequenas quantidades. PROM é uma ROM que pode ser programada através de um processo elétrico mais simples, com um dispositivo chamado programador, e por isso é mais adequada durante o desenvolvimento. Este processo é comumente chamado "queimar a PROM"[1]. Assim como as ROMs, as PROMs só podem ser programadas uma vez.

EPROM – *Erasable PROM* É uma ROM que pode ser apagada e reprogramada. O apagamento é feito com um dispositivo, na qual o chip é introduzido e então exposto a um raio ultra-violeta. O processo de apagamento apaga toda a EPROM, não permitindo apagar somente partes dela. A programação é realizada através de um programador que fornece vários níveis de voltagem dependendo do tipo de EPROM.

EEPROM – *Electrically Erasable PROM* Ao contrário da EPROM, a EEPROM possui um processo de apagamento elétrico. Isso permite o apagamento parcial e que o mesmo seja feito sem remover o chip do lugar. Também elimina a necessidade de equipamentos adicionais para programação e apagamento. Modificar uma EEPROM é um processo lento, pois é feito 1 byte por vez.

Flash Memórias Flash são um tipo de EEPROM. A diferença é que o apagamento da memória Flash é feito em blocos de bytes em vez de um byte por vez. Isso a torna mais rápida. Memórias Flash, porém, são mais caras.

Sistemas embarcados normalmente utilizam memórias Flash para substituir um disco rígido. Embora o custo por byte do disco rígido seja menor, as memórias Flash têm a vantagem de serem menores, de consumirem menos energia e de não possuirem partes mecânicas, o que as torna mais duráveis. Já as memórias EEPROM são utilizadas como alternativas para guardar configurações alteráveis via software, eliminando a necessidade de *jumpers*.

2.3.2 Comunicação

As tecnologias de comunicação que seguem são utilizadas para conexão de sistemas embarcados em rede.

[1] *burning*

CAN

O CAN – *Controller Area Network* – é um sistema de barramento serial. O protocolo CAN, da camada de enlace de dados, está definido no padrão ISO 11898-1 e não inclui a camada física. Dois outros padrões tratam disso: o padrão ISO 11898-2, de alta velocidade; e o padrão ISO 11898-3, de baixa velocidade com tolerância a falhas.

No padrão de alta velocidade, o meio físico é uma linha de dois fios com terminação nas duas pontas. Uma taxa de 1Mbit/s pode ser alcançada em um cabo de até 40m. O comprimento máximo do cabo é de 1Km, permitindo, neste caso, uma taxa de 50Kbit/s.

O CAN permite a interconexão de mais de um dispositivo mestre. A transmissão é feita em *broadcast*. Cada mestre tem um identificador que é utilizado também como prioridade de transmissão. O menor identificador tem maior prioridade. Como o meio é compartilhado, o protocolo *binary countdown* é utilizado para evitar colisões.

O CAN é muito usado no setor automotivo. Outras aplicações incluem automação industrial, automação predial, sistemas de alarmes, câmeras móveis para esportes e até cafeteiras.

RS232

O RS232 – *Recommended Standard 232C* – é um padrão da EIA[1] para comunicação ponto-a-ponto. Foi desenvolvido tendo em vista a conexão entre dois dispositivos seriais, denominados *Data Terminal Equipment* (DTE), geralmente um terminal, e *Data Circuit-Terminating Equipment* (DCE), geralmente um computador. Praticamente todos os computadores pessoais têm uma ou mais interfaces RS232, o que tornou o padrão popular. O padrão especifica as características elétricas, exigências do conector e funções dos sinais.

O padrão impõe um limite de 50ft (cerca de 15m) e uma taxa máxima de transmissão de 19200 bps. Mas esse limite pode ser extrapolado utilizando-se cabos de boa qualidade. Na prática taxas de até 115.200 bps são utilizadas com sucesso, desde que o comprimento do cabo seja mantido curto (1 ou 2 metros).

RS485

O RS-485, também da EIA, é um padrão para comunicação serial que permite até 32 transmissores e 32 receptores em um único barramento de dois fios.

[1] *Electronic Industries Alliance.*

A principal diferença entre este padrão e o RS232 é que neste último os sinais são representados por níveis de voltagem em relação ao terra. Para cada sinal há um fio apenas e há necessidade de um terra comum. Isso é o que limita o tamanho dos cabos e faz com que o RS232 seja adequado para comunicação entre dispositivos próximos tais como um computador e um modem ou entre um computador e um mouse.

No RS485 para cada sinal há um par trançado de fios. A transmissão do sinal é feita através da diferença entre esses dois fios. Isso permite comprimentos de cabos de até 1200m e taxas de transmissão de até 2,5MB/s.

O RS485 é utilizado para comunicação multiponto. Dispositivos adicionais podem ser conectados ao cabo. A maioria dos sistemas RS485 usam uma arquitetura mestre/escravo onde cada escravo tem um endereço único. Porém não existe padronização para os protocolos utilizados sobre a camada física. Por isso, muitos protocolos proprietários estão em uso.

IEEE 802.3

O IEEE 802.3 é um padrão para redes locais com meio compartilhado CSMA/CD[1], popularmente conhecido como Ethernet.

Nas redes Ethernet, é utilizado o seguinte protocolo para acesso ao meio compartilhado. Antes de enviar um quadro, a estação escuta no cabo. Se o cabo estiver ocupado, ela espera até que ele seja desocupado; senão ela inicia a transmissão. Se várias estações transmitirem ao mesmo tempo, ocorre colisão, a qual é detectada. As estações que estavam transmitindo param, esperam um tempo aleatório e reiniciam a operação a partir da escuta no cabo.

Os tipos de cabos mais usados nas redes Ethernet atualmente são o 10Base2 (*thin coax*) e o 10BaseT (par trançado). Ambos operam a 10Mbps. Um segmento de cabo 10Base2 pode ter no máximo 200m de comprimento e o 10BaseT pode ter no máximo 100m.

O formato do quadro Ethernet é mostrado na figura 2.11. Antes do quadro, é transmitido um *preâmbulo* para sincronização do relógio. Logo vem um *delimitador* para identificar o início do quadro. Os dois campos seguintes identificam o destino e a origem. O campo *tipo do quadro* identifica o protocolo sendo usado no quadro. Logo vêm os *dados*, que têm um tamanho entre 0 e 1500 bytes. Um quadro Ethernet não pode ter menos de

[1] *Carrier Sense Multiple Access / Collision Detection.*

64 bytes e o campo *pad* é usado como preenchimento caso não haja dados suficientes. Por fim, o campo *CRC* contém o checksum do quadro.

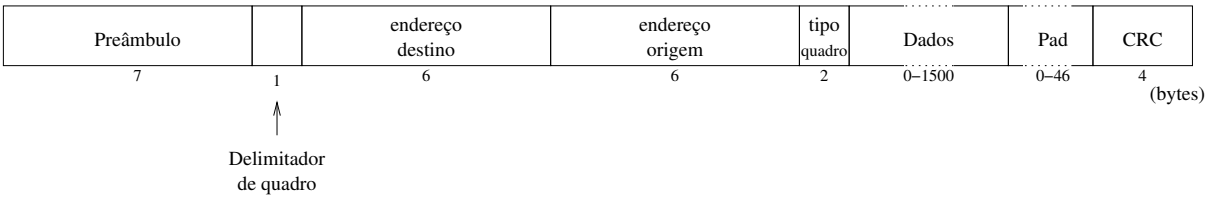

Figura 2.11: Formato do quadro Ethernet.

As redes Ethernet são as mais populares atualmente e, portanto, cada vez mais é desejável conectar dispositivos através deste padrão. Como vantagens, pode-se utilizar a infraestrutura de rede já instalada para os computadores (*hubs* e cabos). Além disso, o protocolo utilizado é aberto, o que facilita a conexão de novos dispositivos. A desvantagem é que cada dispositivo precisa de um adaptador para se conectar.

2.3.3 Chips Ethernet

Dentre as opções de conectividade, enfatizamos neste trabalho as baseadas em Ethernet. Dentre os chips Ethernet mais utilizados em sistemas embarcados de 8 e 16 bits, dois deles se destacam pela quantidade de referências e aplicações encontradas. Estes chips são uma solução de baixo custo e de fácil programação para a conexão de sistemas a redes Ethernet.

RTL8019AS da RealTek [Rea01]. É um controlador Ethernet em um chip que permite operação em 8 bits. Possui 16Kb de RAM interna. Suporta conectores RJ-45, AUI e BNC. Pode ser configurado via jumper, via EPROM ou Plug and Play. O chip opera apenas em 5V. Esse chip é utilizado em vários produtos comerciais, tais como o eZ80 e o PicoWeb (v. seções 3.3.2 e 3.4.1).

CS8900A da Cirrus Logic [Cir01]. É um controlador Ethernet em um chip, de baixo custo, otimizado para barramento ISA. O chip possui 4KB de memória RAM interna para buffers, interface para RJ-45 e AUI e filtros de recepção e transmissão para 10BASE-T. O chip CS8900A seria usado na implementação em hardware deste trabalho, seguindo o projeto de referência em [Ayr00].

O CS8900A possui um módulo que cuida do acesso ao meio. O próprio chip se encarrega da retransmissão do pacote em caso de colisão. Ele também faz geração automática do CRC e preenchimento (*padding*). Quadros que contêm erros são automaticamente rejeitados.

Quando operando no modo de 8 bits o chip não permite o uso de interrupções, e só funciona no modo *polling*.

O fabricante fornece um projeto de referência que instrui como construir um adaptador Ethernet com este chip [Ayr00]. É necessário apenas um cristal oscilador, para gerar o clock, e um transformador de isolamento, que fica entre o chip e o plug RJ-45, além de alguns componentes passivos. O tamanho da placa do adaptador fica em torno de 10cm^2.

Outra característica importante é que este chip pode operar com 3V ou 5V. A tensão de operação pode ser crucial em projetos de sistemas embarcados. Se um sistema opera em uma determinada tensão, é problemático utilizar um chip que opera em uma tensão diferente.

2.3.4 Sistemas Embarcados e a Internet

A ligação de sistemas embarcados à Internet permite inúmeras novas aplicações, como veremos na seção seguinte. Esse casamento também aponta para um mundo mais comunicativo, onde os instrumentos serão mais "inteligentes". Esta seção tenta vislumbrar como pode ser esse mundo.

Em artigo de 1999 [Mit99], Mittag faz algumas previsões sobre as inovações nos projetos de utensílios ligados à Internet. Ele categoriza a comunicação de sistemas embarcados em três níveis:

Nível 1, Porta da frente *(front-door)* É onde estamos no momento. É o nível em que o sistema embarcado acessa a Internet através de um computador pessoal (PC). Enquadram-se nessa categoria tocadores de MP3 que podem obter músicas da Internet e máquinas de costura que podem obter padrões de igual maneira. Neste nível, os aparelhos ainda são dependentes do PC, ou seja, eles funcionam como periféricos temporários do computador. A vantagem desta dependência é que operações complexas podem ser realizadas pelo PC em vez do sistema embarcado, minimizando o custo deste último.

Nível 2, Utensílios maiores Os primeiros sistemas embarcados ligados diretamente à Internet serão os utensílios maiores, tais como a televisão e o vídeo-cassete. O vídeo-cassete, por exemplo, pode ser programado à distância e mais facilmente. Em vez de especificar horários de gravação, o usuário especifica qual programa deseja gravar.

A grande diferença entre essa classe e a primeira é a conexão direta com a Internet, sem necessidade de um PC como gateway. A fim de permitir essa conexão direta, entretanto, são necessários recursos computacionais mais poderosos.

Nível 3, Comunicação presumida No momento nós temos energia presumida. Ninguém precisa se preocupar em estabelecer uma conexão com a concessionária de energia elétrica para ligar um aparelho na tomada. Este modelo deve ser seguido pelas comunicações, o que seria chamado de comunicação presumida. O DHCP[1] é um protocolo que realiza configuração de endereços, mas atua somente até o nível IP. Já há iniciativas para alcançar os níveis mais altos, como por exemplo a arquitetura Jini da Sun [Mic03]. Ela abstrai os serviços e os dispositivos de controle de forma que eles possam ser configurados e programados pelo usuário. Como exemplo, consideremos um interruptor. Normalmente ele controla uma lâmpada, mas poderia também controlar um ventilador de teto ou um ar condicionado. Quem decidiu o que o interruptor iria controlar foi o eletricista, no momento de fazer a fiação da casa. No Jini, temos um sistema centralizado de configuração que associa um ou mais serviços a dispositivos de controle. Isso permite, por exemplo, configurar a luz da sala, o som e o ar condicionado para reagirem ao interruptor.

A questão que fica é se haverá gente querendo dispositivos com essas funcionalidades.

2.3.5 Aplicações

Habilitar os protocolos TCP/IP em sistemas embarcados possibilita uma vasta gama de aplicações. Destas, algumas são comuns e até óbvias, enquanto outras parecem mais futuristas ou fúteis. Nesta seção apresentamos algumas aplicações que não deixam dúvidas de que conectar utensílios a uma rede local ou à Internet traz inúmeras vantagens[Gan98].

Máquinas de venda A máquina pode avisar ao dono que a caixa de dinheiro está cheia ou que o estoque está baixo. Assim facilita a manutenção, poupando visitas desnecessárias.

[1] *Dynamic Host Configuration Protocol.*

Firmware Qualquer aparelho com firmware embarcado pode fazer o download de atualizações via Internet. A empresa economiza em manutenção e o usuário não precisa sair de sua casa.

Aquisição de dados Dados sobre condições climáticas, número de TVs sintonizadas em um canal, ponto de funcionários, etc. podem ser enviados através da rede.

Segurança Certas câmeras podem ser conectadas diretamente a uma rede Ethernet. Sistemas de alarme inteligentes podem se ligar ao sistema da polícia.

Interface gráfica remota Há aparelhos que costumam ter uma interface pobre, de difícil utilização, com teclados pequenos e monitores de baixa resolução. Além disso, agregar esses dispositivos de interfaceamento nos aparelhos aumenta bastante o custo final do mesmo. Ao habilitar um servidor HTTP embarcado, o usuário pode acessá-lo através de um browser, que possui uma interface universalmente conhecida. Este acesso pode se limitar à inspeção, ou pode ser estendido para permitir o controle do aparelho. Além disso, o custo final é reduzido.

Manutenção remota O suporte de um produto pode ser realizado através da Internet. Clientes distantes podem contar com suporte personalizado.

Controle industrial Máquinas, robôs, medidores, etc. podem ser inspecionados e controlados através de web-browsers numa rede local ou pela Internet.

Caixas registradoras Há muito tempo as caixas registradoras comuns vêm sendo substituídas por caixas mais inteligentes conectadas à rede da loja, possibilitando um controle on-line do estoque.

Entretenimento Máquinas de fliperama podem ser mantidas e receber jogos novos através da Internet.

Vale destacar o projeto Arena, da Universidade de Luleå[Are02]. Neste projeto, jogadores de hóquei no gelo são equipados com uma câmera além de medidores de pressão sanguínea e de freqüência de pulso cardíaco. Os dados são transmitidos via TCP/IP sobre uma rede sem fio para os expectadores através de um proxy. O software de comunicação – a pilha lwIP – foi desenvolvido por Dunkels [Dun01b, Dun01a].

A TV e o vídeo-cassete são outros fortes candidatos a se conectarem à rede. A programação poderia ser baixada dos sites dos canais e o usuário poderia escolher na lista os programas que deseja gravar sem se preocupar com os horários de início e fim da gravação. Já existem no mercado conjuntos de TV nos quais um sistema embarcado usa o protocolo PPP[1] sobre linha discada para baixar novo conteúdo.

Outras aplicações mais supérfluas completam o quadro. Grande parte delas consiste em ligar um eletrodoméstico à rede global a fim de inspecioná-lo e controlá-lo a partir de qualquer lugar.

[1] *Point-to-Point Protocol.*

Capítulo 3

Soluções TCP/IP para sistemas embarcados

Ao longo do trabalho, foram encontradas várias soluções para conectar sistemas embarcados a alguma rede com o protocolo TCP/IP. As soluções podem ser classificadas nas seguintes categorias: [Gan98]

- *Stand-alone*
- Parte de um RTOS
- Integradas a um hardware

Descrevemos estas soluções a seguir, baseados nessa classificação.

3.1 Pilhas TCP/IP stand-alone

As pilhas *stand-alone* podem ser utilizadas pelo desenvolvedor como um módulo para o seu sistema operacional ou como uma aplicação *stand-alone* sem sistema operacional. Algumas delas são de código aberto, o que significa que seu código pode ser obtido gratuitamente, modificado e integrado em produtos sem cobrança de *royalties*. Outras são comerciais, com preços que atingem a cifra dos milhares de dólares. Essas pilhas *stand-alone* são objeto de análise dessa seção.

3.1.1 lwIP

A lwIP — *light-weight Internet Protocol* — é uma pilha TCP/IP completa escrita por Adam Dunkels como parte de sua tese de mestrado [Dun01b]. O objetivo da lwIP é minimizar o uso de memória RAM mantendo o protocolo TCP funcionalmente completo. Ela é adequada para sistemas embarcados com cerca de 40Kb de ROM para código e algumas dezenas de bytes de RAM.

Inicialmente, o desenvolvimento da lwIP baseou-se numa arquitetura de *proxy*, conforme mostrado na figura 3.1. O *proxy* possui uma conexão com a Internet e uma conexão com a rede *wireless*. Sua função é realizar um pré-processamento dos pacotes vindos da Internet, em nível IP e TCP, permitindo, assim, a simplificação da pilha a ser usada nos clientes *wireless*. Entre outras funções, o *proxy* se encarrega de armazenar e remontar os pacotes IP que estão fragmentados e fora de ordem destinados à rede *wireless*. O *proxy* entrega então estes pacotes à rede *wireless* sem fragmentação e na ordem correta.

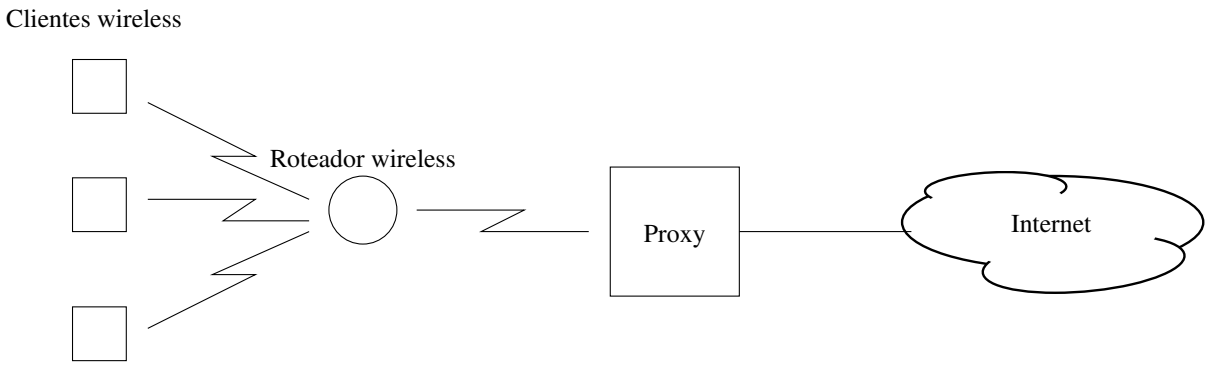

Figura 3.1: Arquitetura de proxy da lwIP.

Embora tenha sido inicialmente projetada para este ambiente, a lwIP teve seu foco mudado para operação independente de *proxy*. A primeira versão pública da lwIP foi a 0.2 e no momento da escrita desta dissertação está na versão 0.5.3. Esta última suporta pacotes IP fragmentados.

A pilha foi implementada seguindo o modelo de protocolos em camadas, separados em módulos. No entanto, algumas violações de camadas foram feitas em prol da velocidade de processamento e do uso da memória. Conforme já mencionado, essa abordagem é comum em sistemas embarcados.

Além dos módulos dos protocolos, outros módulos de suporte foram implementados, a saber:

Emulação de sistema operacional Este módulo funciona como camada intermediária entre a pilha e o sistema operacional. Consiste de várias rotinas invólucro[1] para chamadas de sistema, tais como criação de *threads*, comunicação inter-processos e temporização. Os demais módulos da pilha fazem uso destas funções no lugar de realizar as chamadas de sistema diretamente, a fim de isolar o código dependente do sistema operacional, facilitando assim o transporte da pilha para outros sistemas.

Gerenciamento de memória Os pacotes são armazenados em memória em listas de buffers chamados pbufs. Os pbufs são similares aos mbufs das implementações BSD [WS95]. O módulo de gerenciamento de memória consiste das funções de alocação e liberação de pbufs.

Interfaces de rede A estrutura de representação dos *drivers* de interfaces de rede, chamada `netif`, é também similar à do BSD. Uma lista ligada desta estrutura armazena as várias interfaces presentes no sistema. Este módulo consiste das funções de registro de interfaces e atribuição de seus parâmetros, tais como número IP, máscara e *gateway*.

API A lwIP possui duas interfaces de programação: uma interface dita "crua", e a interface "seqüencial".

Na interface crua, os eventos são notificados à aplicação através de funções *callback*. Funções callback são passadas como parâmetros para a pilha pela aplicação. Quando ocorre um evento a função correspondente é chamada pela pilha.

Já a interface seqüencial é semelhante à API sockets do BSD. Ela é implementada como um módulo em separado, com base na interface crua. Para usar a interface seqüencial, o sistema precisa ter suporte a threads.

Ao usar a interface crua, a aplicação fica mais integrada à pilha e, portanto, mais rápida. O uso de memória também é reduzido. Porém, o programador da aplicação fica mais exposto aos detalhes da implementação da pilha. Com a interface seqüencial é mais fácil escrever a aplicação e o código fica mais simples. Por isso, a

[1] Em inglês, *wrappers*.

interface crua é mais apropriada para aplicações que devem ser pequenas em código e econômicas no uso de memória.

A pilha lwIP pode funcionar com ou sem o suporte de um sistema operacional. Esta é uma opção de compilação. Com o suporte do sistema operacional, a aplicação pode utilizar a API seqüencial. São criadas *threads* para gerenciar conexões TCP e o recebimento de quadros pelas interfaces.

Sem o sistema operacional, o método de *polling* é utilizado. Ou seja, as interfaces são constantemente verificadas quanto à chegada de quadros e os eventos são comunicados à aplicação através dos callbacks.

A implementação TCP suporta *delayed ACKs*, controle de congestão, estimativa de RTT e os algoritmos de recuperação rápida e retransmissão rápida.

3.1.2 uIP

A uIP[1] é uma implementação mínima do TCP/IP escrita por Adam Dunkels do SICS – *Swedish Institute of Computer Science* [Dun02b]. O compromisso principal da uIP é a redução no uso da memória, tanto para código quanto para dados. A uIP foi projetada para ter apenas o conjunto mínimo de recursos necessários para uma pilha TCP/IP completa. Ela trabalha apenas com uma interface de rede e não implementa o protocolo UDP, mas apenas IP, ICMP e TCP. Ela foi escrita inteiramente na linguagem C, o que aumenta sua portabilidade [Dun02a]. O tamanho do código, incluindo IP, ICMP e TCP, na plataforma Intel x86, fica em 5188 bytes.

Dentre as restrições, destacamos as seguintes:

1. Suporte para uma interface de rede apenas;

2. Buffer único para recebimento de um pacote. Um pacote recebido deve ser processado pela aplicação imediatamente. O recebimento de um pacote subseqüente sobrescreve o anterior;

3. Não suporta pacotes IP fragmentados;

4. Não faz roteamento;

[1] Lê-se "micro-ip".

O núcleo da uIP é composto por funções que tratam dos protocolos IP, ARP, ICMP e TCP. O UDP não é suportado. O suporte ao ICMP é muito simples, apenas respondendo a `ECHO REQUEST`'s.

O núcleo não faz chamadas a rotinas de dispositivos. A comunicação entre a pilha uIP com o nível físico é feita através da variável global `uip_len`. No momento da recepção de um quadro ela indica o tamanho do quadro recebido ou zero caso nenhum quadro foi recebido na interface. No momento do envio de um quadro, ela indica o tamanho do quadro a ser enviado. Caso seu valor seja diferente de zero, os dados contidos na variável `uip_buf` são enviados pela interface de rede.

O laço principal da pilha pode ser descrito como segue:

loop
  if chegou pacote ($uip\_len > 0$) **then**
    processa pacote
    chama aplicação se necessário
    envia pacotes de saída
  end if
  if temporizador expirou **then**
    processamento periódico
    chama aplicação se necessário
    envia pacotes de saída
  end if
end loop

Quando um pacote chega na interface, a rotina de entrada da pilha TCP/IP é chamada. Ela realiza a demultiplexação do pacote, passando-o à aplicação se necessário. Durante o processamento do pacote recebido, um pacote de resposta pode ser gerado, e por isso a rotina de envio é chamada.

Quando um temporizador expira, um procedimento semelhante é adotado. A rotina periódica é chamada para remover as conexões que já estão fechadas depois do tempo de 2MSL, retransmitir pacotes que não foram confirmados a tempo, enviar confirmações prorrogadas, entre outras tarefas. A aplicação pode ser chamada caso aconteça uma alteração na conexão, como o fechamento por *timeout*, por exemplo. Novamente a rotina de saída de pacotes é chamada, pois também essas operações podem gerar um pacote para o exterior.

A uIP é configurável em tempo de compilação. Várias constantes podem ser ajustadas

no arquivo `uipopt.h`. Entre outras opções, as constantes definem o número máximo de conexões simultâneas abertas, o tamanho do buffer, o timeout de retransmissão, o número de entradas da tabela ARP, se a arquitetura é *little* ou *big-endian*, etc. Com isto é possível adaptar a uIP a uma grande variedade de sistemas.

3.1.3 TCP/IP Lean

O livro "TCP/IP Lean", escrito por Jeremy Bentham, inclui duas pilhas [Ben00]:

1. uma pilha TCP/IP escrita a partir do zero para a plataforma Intel x86; e
2. um mini servidor web para o microcontrolador PIC.

Ambas as pilhas foram escritas na linguagem C. O ambiente de desenvolvimento do autor é composto por ferramentas para DOS e Windows.

A pilha para a plataforma Intel pode ser compilada com os compiladores Borland C (versões 3.1 e 4.5), Microsoft Visual C (versão 6.0) e o GNU DJGPP, que é uma versão do GCC para DOS/Windows. O livro traz versões previamente compiladas da pilha com cada um dos compiladores. O tamanho da pilha varia de acordo com o compilador utilizado. Tomemos o servidor HTTP para a plataforma PC como comparação. Quando compilado com o DJGPP ele ocupa 270Kb. Já compilado com o Microsoft Visual C, ocupa 90Kb. Essa diferença deve-se a diversos fatores, entre eles, a quantidade de otimização do compilador, o tamanho do código das bibliotecas e as opções utilizadas na geração do código.

Servidor web para o PIC

O mini servidor web para o PIC faz parte de um projeto ilustrativo contido no livro. Este projeto inclui não só o desenvolvimento do código do servidor web, mas também o hardware necessário para integrá-lo. O hardware inclui:

- processador PIC (modelo 16C76 ou 16F876[1]);
- uma unidade E^2ROM;
- um relógio de tempo real;

[1]No primeiro a memória de programa é uma EPROM e no segundo é uma memória *flash*.

- um sensor de temperatura;
- 8 LEDs;
- 6 chaves;

A E^2ROM é uma memória externa não-volátil utilizada para armazenar o sistema de arquivos do servidor web. O relógio de tempo real e o sensor de temperatura são usados para exemplificar a capacidade do servidor de gerar páginas com conteúdo dinâmico. Os LEDs são utilizados para demonstrar a possibilidade de controlar dispositivos ligados ao servidor, através de um navegador web.

O software foi escrito para um compilador C do PIC: o PCM da Custom Computer Services [CCS02]. Muitas diretivas relacionadas ao hardware foram utilizadas ao longo do código. Além disso, o gerenciamento de memória é todo deixado a cargo do compilador. Como conseqüência, essa pilha tem baixa portabilidade.

O código contém uma pilha TCP/IP projetada especificamente para suportar o servidor. Esta pilha miniatura possui as rotinas básicas para permitir o funcionamento do servidor web. Também um servidor *daytime* foi implementado, com o fim de facilitar o teste da pilha miniatura, já que uma transação do servidor *daytime* é mais simples do que a de um servidor HTTP.

Os protocolos suportados pelo servidor web são o IP, ICMP, TCP e HTTP. A implementação de IP não suporta fragmentação nem opções. O ICMP limita-se a resposta de *ECHO REQUEST's* com um tamanho máximo de 32 bytes, devido à limitação de RAM no PIC. No nível de enlace, a pilha suporta apenas o protocolo SLIP.

O servidor é capaz de gerar conteúdo dinâmico e processar formulários. Isso significa que é possível, por exemplo, servir uma página que exiba a temperatura obtida de um termômetro ligado ao PIC.

Para gerar o conteúdo dinâmico, o autor implementou um esquema similar ao CGI, o qual denominou EGI – *Embedded Gateway Interface*. As páginas HTML podem conter variáveis delimitadas por marcadores especiais que acionam a substituiçao EGI. Estas variáveis são substituídas pelo servidor no momento do envio da página.

O PIC possui apenas 368 bytes de memória RAM, sendo que esta quantidade ainda é dividida em bancos. O tamanho da maior estrutura ou vetor é de 96 bytes. Essa limitação é um empecilho para a utilização de buffers de recepção e envio. Por isso, os quadros são decodificados durante a recepção, e criados durante a transmissão. Isso equivale a dizer

que no momento em que os bytes são recebidos através da linha serial, o checksum vai sendo calculado, o protocolo de nível superior é identificado e a rotina de resposta – HTTP ou *daytime* – é chamada.

Nessa abordagem, o cálculo do *checksum* do TCP, o qual cobre todos os dados, torna-se mais trabalhoso. Para quadros que estão chegando, basta manter um acumulador e depois verificar se o *checksum* calculado é válido. Para quadros a serem transmitidos, porém, isso não pode ser feito, pois o campo *checksum* do TCP encontra-se no início do quadro, e seu valor só será conhecido após o envio de todos os bytes.

Bentham optou por calcular o checksum dos arquivos previamente e armazená-los em conjunto. Isso pode ser feito porque as páginas são definidas em tempo de compilação. Para servir uma página estática (sem conteúdo dinâmico), basta enviar o *checksum* associado a ela. As páginas que contêm variáveis, entretanto, precisam de um tratamento especial, pois a mera substituição de uma variável modificará o *checksum* previamente calculado. Para contornar essa diferença devida à substituição de variáveis, as páginas devem conter um comentário HTML para cada variável. Neste comentário, serão inseridos caracteres que equilibram o valor do *checksum*. Como estes caracteres estão dentro de um comentário HTML, o usuário não percebe nada, embora eles sejam considerados no momento do cálculo do *checksum*.

3.1.4 ZSock

ZSock é uma pilha TCP/IP de código aberto, escrita para o computador portátil Cambridge Z88 [Mor02b]. Seu código está em Small C, que é um subconjunto de C idealizado por Ron Cain e descrito em [Cai80]. Alguns trechos do código, como o cálculo de checksum, está em linguagem assembly, para melhorar o desempenho. O compilador utilizado é o Z88DK, que é de código aberto e está disponível na Internet[Mor02a]. A pilha é distribuída sob os termos de uma licença estilo BSD.

A ZSock possui os seguintes protocolos: SLIP, PPP, IP, ICMP, UDP e TCP. Sua implementação do protocolo IP não suporta fragmentação. O tamanho do código fica em torno de 32Kb.

A ZSock foi projetada para possibilitar o acesso à Internet por parte dos usuários do Cambridge Z88. Daí depreende-se que seu foco é no lado cliente dos aplicativos TCP/IP em detrimento do lado servidor. Ela inclui clientes PING, FTP e TELNET.

3.1.5 TinyTCP

A TinyTCP é uma pilha de domínio público de autoria da empresa Imagen [Pad02]. Trata-se de uma implementação reduzida dos protocolos IP, TCP e FTP visando a gravação do código objeto em ROM. Ela foi projetada para o processador 68000, que é um processador *big-endian*. Por esse motivo, ela não contém suporte à troca da ordem de bytes, necessária a uma máquina *little-endian*.

No nível de enlace, a TinyTCP possui um *driver* Ethernet para um adaptador de rede 3Com. A implementação é baseada em espera ocupada (*polling*).

A implementação do TCP não suporta dados urgentes nem a opção de tamanho de janela. O tamanho da janela de recepção é fixo.

O tamanho do código objeto é de 16Kb. O código fonte da pilha é pequeno: cerca de 1400 linhas, totalizando 42Kb. Ele está organizado conforme mostra a tabela 3.1.

Tabela 3.1: Organização do código da pilha TinyTCP.

Arquivo	Descrição
sed.c, sed.h	*Simple Ethernet Driver*. Um driver para um adaptador 3Com.
arp.c	Funções do protocolo ARP.
tinytcp.c, tinytcp.h	Implementação simplificada do TCP.
tinyftp.c	Implementação simplicada do FTP. Possibilita apenas envio de arquivos.

3.1.6 uC/IP

A uC/IP é um projeto de transporte da pilha TCP/IP para sistemas embarcados de Guy Lancaster [Lan02]. O código é baseado na pilha do BSD, do qual possui a mesma licença, e na pilha KA9Q, que é uma pilha TCP/IP para uso em rádio [Kar03].

Os objetivos do projeto são:

- Portabilidade: compilar em Linux e DOS para os microcontroladores mais comuns;
- Demonstração: criação de aplicativos exemplos;
- Flexibilidade: melhorar a comunicação entre processos;
- Tamanho: reduzir ao máximo.

O projeto teve início dentro de uma empresa onde o autor trabalhava, a Global Election Systems Inc. Na época em que a empresa estava convertendo seus sistemas monolíticos com protocols proprietários para sistemas multithreads com protocolos abertos, eles avaliaram algumas pilhas comerciais e optaram por desenvolver sua própria a partir das pilhas de código aberto. Depois de terminada a pilha, a empresa permitiu que o resultado fosse colocado em domínio público.

A pilha foi projetada para funcionar em um sistema embarcado com uma linha serial. Atualmente ela possui *drivers* para os adaptadores de rede Ethernet RealTek 8139, CS8900 e NE2000 genérico, além dos protocolos SLIP e PPP.

A uC/IP é suportada nos sistemas DOS, Windows, Unix e uC/OS. O tamanho do código é de cerca de 30Kb para uma versão sem PPP. Quando inclui o PPP completo, o tamanho do código chega a 60Kb.

3.1.7 CMX Micronet

A Micronet é uma pilha comercial da empresa CMX para processadores de 8 e 16 bits [CMX02]. O sistema alvo pode utilizar Ethernet ou linha discada.

O código objeto da pilha pode ficar entre 3Kb e 20Kb dependendo do processador e da configuração escolhida. Para que coubesse em sistemas com pouca memória, as seguintes restrições foram introduzidas na CMX MicroNet:

- Não é 100% compatível com as RFCs; e
- Não suporta fragmentação IP.

A Micronet possui os seguintes protocolos: SLIP, PPP, ARP, IP, UDP, TCP, servidor HTTP, servidor FTP. A pilha é suportada nos processadores 8051, Z180, PIC (linha PIC18) e Hitachi H8s.

A CMX também fornece uma pilha mais completa para sistemas embarcados de 16 e 32 bits chamada CMX TCP/IP. Esta é compatível com as RFCs e a utilização de memória é localizada e determinística.

Na data da escrita dessa dissertação, o custo da CMX Micronet e da CMX TCP/IP, conforme informado pelo fabricante, é de respectivamente US$ 5.500,00 e US$ 10.500,00 para quase todos os processadores. A CMX não cobra *royalties* sobre produtos distribuídos que utilizam as pilhas. O desenvolvedor paga por produto por site. Isso significa que um mesmo desenvolvedor deve comprar uma licença para cada produto diferente que

incluir a pilha, não importando o número de produtos vendidos. Na compra da CMX Micronet, o desenvolvedor recebe o código fonte.

3.1.8 Fusion TCP/IP

A Fusion TCP/IP é uma pilha comercial desenvolvida pela Pacific Softworks[1], otimizada para sistemas embarcados[DSP02] e aplicações de tempo real. Ela é compatível com as RFCs e suporta dispositivos de 16, 32 e 64 bits. Uma de suas grandes vantagens é sua flexibilidade e portabilidade, pois seu código é escrito em C ANSI, tornando-a independente de processador, de sistema operacional e de compilador.

A implementação da Fusion é compatível com as RFCs. Sua versão mais recente no momento desta escrita (versão 6.0) possui suporte a vários algoritmos do TCP/IP: PAWS[2], medida de RTT, ACK seletivo, *fast retransmit/fast recovery* e *slow start*/prevenção de congestão. A pilha também suporta multicast. Estes recursos podem ser desativados pelo desenvolvedor no momento da compilação a fim de diminuir o tamanho da pilha.

A interface de programação da pilha Fusion é compatível com a API dos sockets. Entretanto ela só suporta um subconjunto das opções de sockets.

A pilha é vendida "royalty-free", ou seja, o desenvolvedor não precisa pagar royalties pelos produtos que a utilizam. A empresa dá suporte por 90 dias após a compra, a partir de quando o desenvolvedor deve pagar pelo suporte. Esta é uma das desvantagens das pilhas comerciais.

Os seguintes protocolos são suportados pela Fusion TCP/IP: ARP, RARP, IP, ICMP, UDP, TCP, DHCP (cliente/servidor), BOOTP, PPP (cliente/servidor), PPPoE, FTP, TFTP, Telnet, DNS, SNTP[3], IGMP, RIP/RIP2, OSPF, NAT e SNMP. O tamanho do código fica entre 50 e 150Kb dependendo da configuração escolhida.

3.1.9 Quadro comparativo

No quadro 3.2, sintetizamos as características das pilhas estudadas.

[1] A Pacific Softworks foi comprada pela NetSilicon.
[2] *Protection Against Wrapped Sequence.*
[3] *Simple Network Time Protocol.*

Tabela 3.2: Resumo das pilhas.

Pilhas	Tamanho	Enlace			Rede/Transporte				Servidores					Compat.
	(KB)	SLIP	Eth + ARP	PPP	IP	ICMP	UDP	TCP	BOOTP	DHCP	Login	FTP	HTTP	RFC1122
uIP	5	X	X	–	X	X	–	X	–	–	–	–	X	S
lwIP	11	X	X	X	X	X	X	X	–	X	X	–	X	S
ZSock	32	X	X	–	X	X	X	X	–	–	–	–	–	N
TinyTCP	16	–	X	–	X	–	–	X	–	–	–	X	–	N
TCP/IP Lean (PIC)	5	X	–	–	X	X	–	X	–	–	–	–	X	N
uC/IP	30–60	–	X	X	X	X	X	X	X	X	–	–	–	N
CMX Micronet	2–11	X	X	X	X	X	X	X	X	X	X	X	X	N
Fusion TCP/IP	50–150	X	X	X	X	X	X	X	X	X	X	X	X	S

3.2 Pilhas em SOs

Alguns distribuidores vendem as pilhas TCP/IP como parte de um sistema operacional. Em sistemas embarcados, normalmente são utilizados sistemas operacionais de tempo real (RTOS[1]). Descrevemos abaixo os principais sistemas operacionais para sistemas embarcados e as características de suas pilhas TCP/IP.

3.2.1 uClinux

O uClinux é o principal representante da plataforma Linux nos sistemas embarcados [JD02]. Trata-se de um derivado do kernel do Linux 2.0 para sistemas que não possuem unidade de gerenciamento de memória.

O kernel ocupa menos de 512Kb, e somando-se os aplicativos o sistema fica em 900Kb. O uClinux possui uma pilha TCP/IP completa que inclui suporte a Ethernet, PPP, IP, ICMP, UDP, TCP, e servidores FTP, HTTP e Telnet.

O uClinux foi desenvolvido originalmente para o Motorola 68000 mas hoje é suportado em várias arquiteturas: Motorola ColdFire, Motorola DragonBall, Motorola QUICC, ARM7TDMI e MC68EN302, Intel i960, NEC V850E, Atari 68K, PRISMA e ETRAX. Entre os dispositivos que adotaram uClinux estão os roteadores Cisco 2500/3000/4000, alguns telefones de VoIP e webcams.

3.2.2 eCos

O eCos – *embedded Configurable operating system* – é um sistema operacional de tempo real para sistema embarcados, da RedHat [Red03]. É um sistema de código aberto e livre de *royalties*. Ele é distribuído sob uma licença GPL modificada que permite a distribuição do sistema em aplicações comerciais. O código está disponível gratuitamente para *download*.

O eCos provê uma infraestrutura de desenvolvimento básica para suportar dispositivos com dezenas a centenas de Kbytes. Um kernel típico que inclui um escalonador, gerenciamento de memória, suporte a relógio de tempo real, várias *threads*, e tratadores de interrupção exigem em torno de 3Kb ROM e 1Kb de RAM.

Na sua versão mais atual, o eCos vem com uma pilha TCP/IP derivada da pilha do OpenBSD. Esta pilha está em desenvolvimento e suporta atualmente IPv4, UDP,

[1] *Real Time Operating System.*

TCP, BOOTP, ICMP. No nível físico ela suporta apenas Ethernet. Há drivers Ethernet disponíveis apenas para algumas placas de avaliação.

O eCos foi projetado tendo em vista arquiteturas de 16, 32 e 64 bits. Ele suporta as seguintes arquiteturas: ARM, Intel x86, Matsushita AM3x, MIPS, NEC V8xx, PowerPC, SPARC e SuperH.

A pilha lwIP, descrita na seção 3.1.1, foi transportada para o eCos.

3.2.3 QNX

O QNX é um sistema operacional comercial de tempo real para sistemas embarcados [QNX02]. A última versão do QNX, o QNX Neutrino, inclui uma pilha TCP/IP completa. Essa pilha suporta ARP, ICMP, UDP, IP e TCP, além de *forwarding*, *broadcast* e *multicast*.

Para aplicações menores, o QNX ainda provê uma pilha miniatura. Essa pilha miniatura, com suporte a IP, TCP e UDP sobre PPP e Ethernet apresenta um uso de memória de menos de 80Kb. Também é possível trocar entre as duas pilhas sem precisar recompilar as aplicações.

O QNX está disponível para as seguintes arquiteturas: Intel x86, PowerPC, ARM, StrongARM, XScale, MIPS, e SH-4.

3.2.4 RTEMS

O RTEMS – *Real-Time Executive for Multiprocessor Systems* é um sistema operacional de tempo real da OAR – On-line Applications Research[Cor02]. A OAR é uma empresa que provê serviços de pesquisa e desenvolvimento na área de defesa, em especial para o Exército americano. Ela desenvolveu, entre outras aplicações, sistemas computacionais de lançamento de mísseis[1] e simuladores para treino, e auxilia no desenvolvimento de arquiteturas avançadas para sistemas bélicos.

O RTEMS é um sistema operacional de código aberto licenciado sob os termos de uma licença GPL modificada. Não são cobrados *royalties* para os produtos comerciais desenvolvidos a partir dele. A licença modificada pemite que produtos comerciais baseados nele não precisem ser de código aberto também, ao contrário da licença GNU pura. Os usuários desenvolvedores são incentivados a enviarem as modificações feitas no sistema de volta à OAR a fim de torná-las parte da distribuição oficial. A OAR atua como gerente

[1] RTEMS costumava significar *Real-Time Executive for Missile Systems*.

do código-fonte do sistema, além de oferecer suporte, treinamento e desenvolvimento particularizado.

O RTEMS está disponível para as seguintes plataformas: Motorola MC68xxx e MC683xx, Motorola ColdFire, Hitachi SH, Intel i386 e i960, MIPS, PowerPC, SPARC, AMD A29K e Hewlett-Packard PA-RISC. Além disso, há uma versão para o UNIX que pode ser usado como ambiente de prototipação e simulação.

A pilha TCP/IP do RTEMS é baseada na pilha do FreeBSD. Possui os seguintes protocolos: IP, ICMP, UDP, TCP, DHCP, RARP, TFTP, RPC, servidor FTP e servidor HTTP.

3.3 Pilhas integradas ao Hardware

Nesta seção analisamos algumas pilhas que estão associadas a um determinado hardware ou implementadas em hardware.

Várias empresas iniciaram uma verdadeira corrida para oferecer um sistema embarcado "web-enabled". Visando desenvolvedores, que têm pressa para lançar produtos no mercado, elas disponibilizam kits de desenvolvimento que facilitam a implementação. Esses kits geralmente são compostos de:

- o sistema que será embarcado
- hardware adicional para comunicação e depuração
- software de desenvolvimento: compilador, pilha TCP/IP, ambiente de programação.

3.3.1 Rabbit

A Rabbit Semiconductors é a empresa que oferece a linha Rabbit de kits de desenvolvimento e de módulos Ethernet embarcáveis. Seus produtos são soluções integradas de hardware e software. Os kits de desenvolvimento se destinam ao projeto e à prototipação, enquanto os módulos são utilizados na fabricação do produto em larga escala.

Os módulos são baseados nos processadores Rabbit. São processadores de 8 bits baseados no Z80, mas mais poderosos. O Rabbit 3000, por exemplo, pode operar a 54MHz e possui 7 portas paralelas de 8 bits, 6 portas seriais e um espaço de endereçamento de 1Mb.

A pilha TCP/IP vem incluída com o Dynamic C, que é o ambiente de desenvolvimento distribuído com os kits. A pilha possui os seguintes protocolos: IP, UDP, TCP, BOOTP, servidor FTP, cliente FTP, servidor HTTP, POP3, SMNP, SMTP e TFTP. Drivers para o chip Ethernet da RealTek também estão incluídos. O servidor web inclui suporte a geração de conteúdo dinâmico através de scripts CGI.

O código fonte da pilha é distribuído juntamente com o Dynamic C. Não há cobrança de royalties por produto vendido. O custo de um dos módulos Ethernet, o RCM3010, composto por um processador Rabbit 3000, 512K SRAM, 512K de Flash, porta Ethernet 10BASE-T e clock de 29.4 MHz fica em torno de US$60,00 para compras acima de 1000 unidades. O kit de desenvolvimento baseado neste módulo custa US$299,00 e inclui a placa de prototipação e o ambiente Dynamic C com a pilha TCP/IP, além de um cabo serial e fonte de alimentação.

3.3.2 eZ80

O eZ80 é um produto da Zilog, baseado no processador Z80 [Zil02]. Trata-se de um sistema integrado de hardware e software que permite o desenvolvimento de aplicações TCP/IP.

O processador eZ80 é baseado no Z80, mas muito mais poderoso. Embora seja de 8 bits, ele tem um espaço de endereçamento linear de 16MB, 2 canais DMA e sua freqüência de operação é de 50MHz.

A pilha TCP/IP deste sistema é a mesma do sistema operacional XINU. O XINU é um pequeno sistema UNIX de uso didático escrito por Douglas Comer [Com84]. Seu código é aberto e livre de royalties. Segundo Comer, mesmo tendo sido projetada para uso didático, a pilha TCP/IP do XINU é completa e oferece os mesmos recursos que as pilhas comerciais [Com95b].

A pilha lwIP, descrita na seção 3.1.1, foi transportada recentemente para este sistema embarcado.

3.3.3 S7600

O chip S-7600A é uma solução TCP/IP em hardware fabricada pela Seiko [Sei02, iRe02a]. Ela se baseia na tecnologia Internet Tuner, criada pela empresa iReady. A linha S-7600 teve sua produção descontinuada pela Seiko.

O chip possui 10Kb de memória RAM. Ele suporta os protocolos PPP, IP, ICMP, UDP e TCP. O microcontrolador ligado ao S-7600 pode enviar através dele uma seqüência de bytes. O chip então se encarrega de encapsular essa seqüência de bytes nos cabeçalhos dos protocolos. O chip tem capacidade para 2 sockets, que são um ponto de comunicação entre a aplicação sendo executada e a pilha em hardware.

O S-7600 contém registradores que guardam informações como o número IP e o tamanho de um segmento TCP recebido. A aplicação pode utilizar uma biblioteca desenvolvida pela iReady, chamada iAPI, que acessa os registradores do chip através de rotinas escritas em C. Isso torna o desenvolvimento mais fácil e o código mais portável.

Um mini servidor web de exemplo foi desenvolvido com o S-7600 e um processador PIC16F84, de 8 bits. O projeto está documentado em [iRe02b].

3.3.4 IPμ8930

É um módulo controlador TCP fabricado pela Ipsil [Ips02]. O módulo consiste de um chip controlador TCP IPμ, um chip controlador Ethernet RTL8019AS e uma EEPROM de 512Kb. O módulo não vem com o plug RJ45. Além disso, o módulo possui 8 pinos de E/S, configuráveis como entrada ou saída via software, e suporte aos barramentos I^2C e SPI.

A pilha TCP/IP é implementada em hardware no chip IP*mu*. Ela é compatível com a RFC 1122[1]. O chip foi desenvolvido com uma tecnologia patenteada pela empresa como *FlowStack*. Essa tecnologia otimiza as interações entre as camadas de aplicação e as camadas de transporte/rede, possibilitando a redução do TCP/IP no silício em um baixo número de portas.

Os protocolos suportados são IP, UDP, TCP, DHCP e servidor web HTTP 1.0. A pilha não impõe restrição no tamanho do pacote. Suporta fragmentação IP e pacotes fora de seqüência. Também suporta os algoritmos do TCP de controle de fluxo e de prevenção de congestão.

Um kit de desenvolvimento para esse módulo está disponível por US$199. O módulo custa US$99. O módulo foi desenvolvido para ser usado em monitoramento remoto, mas outra aplicação sugerida é uma ponte de serial para Ethernet.

[1] A RFC 1122 trata das exigências para hosts da Internet [Bra89].

3.4 Mini Servidores Web

Quanto menor e mais barato um servidor web, mais aplicações poderão utilizá-lo. Foram encontradas várias páginas na Internet sobre servidores web miniatura. Alguns destes são projetos artesanais elaborados por hobistas e vários deles clamam ser o menor servidor web do mundo. Descrevemos a seguir esses servidores.

3.4.1 PicoWeb

O PicoWeb, da Lightner Engineering, é um mini servidor web residente numa placa de circuito impresso. O processamento fica a cargo de um microcontrolador AVR da ATMEL, o AT90S8515. O chip Ethernet utilizado é o RTL8019 da RealTek. Os diagramas de circuito estão disponíveis no site do PicoWeb [Eng02]. A placa é mostrada na figura 3.2.

Figura 3.2: O servidor web PicoWeb.

O PicoWeb é programável através de um ambiente de desenvolvimento distribuído juntamente com o produto. As ferramentas de desenvolvimento estão disponíveis para ambiente Windows e GNU/Linux.

As licenças do PicoWeb são vendidas por unidade e não por projeto. Algumas partes do código do *firmware* do PicoWeb são distribuídas apenas em formato binário.

Apesar de ser específica para o servidor web, a pilha do PicoWeb permite que uma

aplicação abra suas próprias conexões TCP ou UDP. Um exemplo do site mostra como é possível enviar um email curto a partir do PicoWeb para um servidor SMTP. A conexão, entretanto, é limitada. Após a abertura, o PicoWeb pode enviar apenas um pacote Ethernet e receber outro pacote Ethernet de resposta.

O trecho de código da figura 3.3 ilustra como é descrito um projeto simples do PicoWeb. Este exemplo foi retirado do site do produto.

```
//-------------------------------------------------------
//
// PicoWeb Project File for Simple Hello World! Web Page
//
//-------------------------------------------------------
//
// application-specific preprocessor definitions
//
#define BANNER "\r\nPicoWeb Hello World!\r\n"
#define EEPROM_IP         /* use file "ip" for default IP address
    */
#define NET_CONFIG_IP     /* allow IP address reconfiguration via
    net */
#define ENABLE_WATCHDOG   /* use Atmel watchdog timer hardware */
#define DEBUGGER          /* include debugger firmware */
//#undef DEBUGGER

//#define CLOCK 8000000     /* processor clock rate */
#define CLOCK 7372000     /* processor clock rate */

#define BAUD_RATE 19200   /* serial port baud rate */

//
// application-specific HTML and image file names
//
hello.htm       // ii00  (default Web page)
picoweb.jpg     // ii01  (PicoWeb logo image)
```

Figura 3.3: Exemplo de projeto no PicoWeb, retirado do site do produto.

No início do código alguns `defines` configuram o ambiente de execução no PicoWeb. Logo vem uma lista com os arquivos a serem incluídos no servidor. Essa lista pode conter páginas HTML, figuras JPEG e um applet Java. As últimas seções do projeto – não mostradas na figura – contêm rotinas em assembly ou pcode.

As rotinas especificadas no projeto podem ser chamadas a partir do código HTML. Essas rotinas podem retornar um valor a ser substituído na página web. Elas também podem causar a alteração do estado do PicoWeb ou de um dispositivo ligado a ele.

Aplicações no PicoWeb podem ser programadas em C. Entretanto estas ficam limitadas à memória de programa não utilizada, cerca de 2Kb. Normalmente é utilizada uma linguagem interpretada projetada especificamente para o PicoWeb, que é o "pcode". O interpretador permite que o código resida em uma memória externa, a qual oferece um espaço de endereçamento maior e, por conseqüência, aplicações mais complexas podem ser escritas.

O AT90S8515 é um microcontrolador de 8 bits da ATMEL, de arquitetura RISC [ATM99]. Ele possui 32 registradores de uso geral diretamente ligados à ULA, o que permite que dois registradores sejam acessados numa mesma instrução. O chip possui 8Kb de memória flash para programa, 512 bytes de memória EEPROM para dados não voláteis, e 512 bytes de RAM. Operando a 8MHz ele alcança um *throughput* de 8 MIPS.

3.4.2 IPic

O IPic é um servidor web contido num processador PIC, desenvolvido por H. Shrikumar da Universidade de Massachussets [Shr02]. O processador é do tamanho de uma cabeça de fósforo, conforme mostra a figura 3.4.

A conexão física é feita via protocolo SLIP a 115200 bps. Embora esteja contida em apenas 512 palavras de 12 bits, o autor afirma que a pilha do IPic é compatível com as exigências da RFC 1122 e o servidor web implementa o protocolo HTTP 1.0. O código, entretanto, não está disponível para download.

3.4.3 WebACE

O WebACE é um mini-servidor web implementado num controlador ACE1101MT8 da Fairchild [Whi02]. Segundo o autor, este é o menor servidor web do mundo, já que o controlador ACE possui pouco mais da metade do tamanho do IPic, descrito na seção anterior. O servidor é mostrado na figura 3.5.

A pilha utilizada é específica para a aplicação do servidor web e, segundo o autor, é bem limitada mas suficiente para servir páginas pequenas. Ela possui SLIP, IP, ICMP, TCP e o servidor. A conexão com a Internet é feita via SLIP numa interface RS232. O código completo ocupa cerca de 1KB e não está disponível para download.

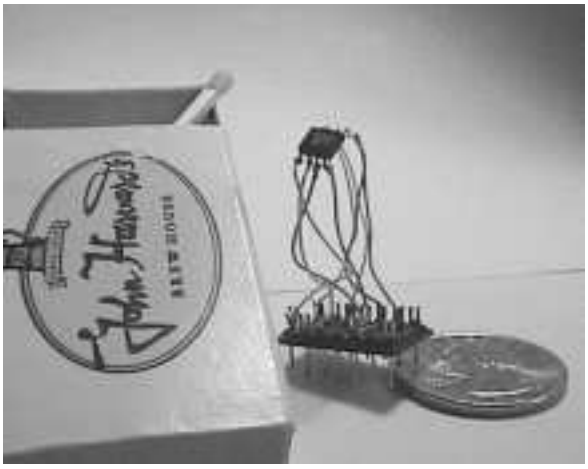

Figura 3.4: O mini servidor web IPic.

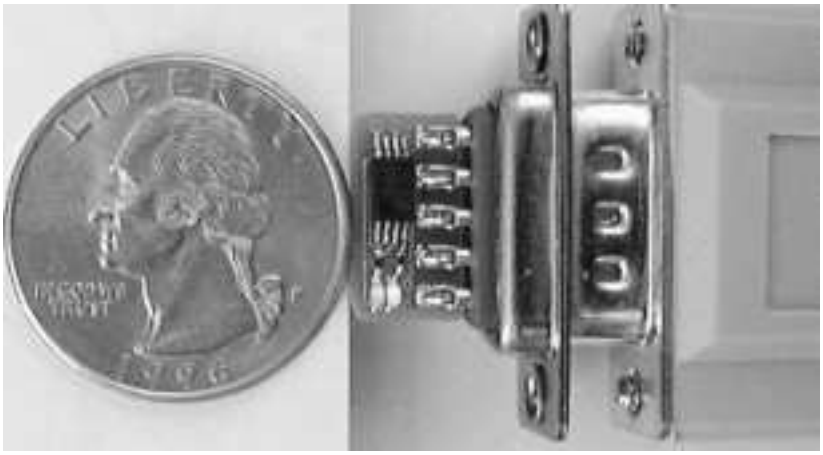

Figura 3.5: O mini servidor webACE comparado a uma moeda.

Capítulo 4

Implementação

Como parte deste projeto, trabalhamos no transporte de duas pilhas para o processador Z80, a saber: a uIP e a lwIP. Ambas foram desenvolvidas pelo mesmo autor, Adam Dunkels. São pilhas pequenas, escritas na linguagem C, projetadas para sistemas embarcados.

O processador Z80 foi escolhido devido à disponibilidade. O Instituto de Computação conta com alguns módulos de controle de acesso a portas através de cartão magnético. Estes módulos contêm um sistema baseado no controlador Z84C15, que por sua vez é baseado no processador Z80. Ao controlador seria conectada uma placa baseada no controlador Ethernet CS8900 da Cirrus, construída com base no projeto de referência do fabricante. Esta placa não ficou pronta em tempo hábil e portanto a implementação em hardware não foi possível.

O compilador utilizado foi o SDCC – Small Devices C Compiler. O SDCC é um compilador de código aberto licenciado sob a GNU GPL e está disponível gratuitamente na Internet [Dut02]. Ele suporta diversas arquiteturas alvo, dentre elas o Z80, o 8051 e o PIC, e foi escrito em C++. Embora o SDCC seja estável, em alguns momentos foram detectados erros no compilador, os quais foram relatados aos desenvolvedores e corrigidos imediatamente.

Como não dispúnhamos de *hardware* para testes, o código foi testado num simulador, o sz80, que é distribuído juntamente com o SDCC. O simulador e os testes foram executados numa máquina Intel x86 com sistema operacional GNU/Linux.

Tabela 4.1: Organização do código da uIP.

Diretório	Arquivo	Descrição
uip/	uip.c	Núcleo da pilha.
	uip_arp.c	Protocolo ARP.
sdcc/	main.c	Laço principal.
		Verifica chegada de quadros e chama rotina periódica.
	tapdev.c	Rotinas para uso do driver TUN/TAP.
	uipopt.h	Opções de compilação (ex. número de conexões possíveis).
	uip_arch.c	Código dependente de arquitetura (checksum).
apps/httpd/	httpd.c	Servidor web minimalista.
	fs.c	Rotinas do sistema de arquivos.
	fsdata.c	Dados do sistema de arquivos. Gerado por um script Perl.
	cgi.c	Rotinas para geração de conteúdo dinâmico.
	makefsdata	Script em Perl. Gera fsdata.c a partir do diretório fs/.
apps/httpd/fs/		Contém os arquivos (HTML, JPG, etc.) a serem servidos.

4.1 uIP

Nesta seção descrevemos o transporte da pilha uIP. Ela foi descrita neste trabalho na seção 3.1.2.

4.1.1 Funcionamento interno da uIP

O código da uIP se encontra subdividido conforme ilustra a tabela 4.1.

A parte nuclear da pillha se encontra no diretório `uip/`. O diretório `sdcc/` guarda os arquivos referentes ao transporte da pilha. O diretório `apps/httpd/` contém uma implementação mínima de um servidor web.

A principal função da uIP é a `uip_process()` e está no arquivo `uip/uip.c`. Ela é chamada do laço principal na função `main()` por dois motivos: quando a interface recebe um quadro, ou para processamento periódico. Este período é demarcado por um temporizador que expira de 100 em 100 ms.

Na chegada de qualquer quadro válido, a uIP atualiza sua tabela ARP. Caso o quadro contenha um pacote ARP, de solicitação ou de resposta, o mesmo é tratado pela rotina `uip_arp_arpin()`.

Na chegada de um pacote IP, são realizados os seguintes testes de consistência:

1. a versão do pacote deve ser 4;

2. o tamanho indicado no cabeçalho deve ser igual ao reportado pelo *driver*;

3. o pacote não pode ser um fragmento[1];

4. o endereço destino deve ser o da máquina em que a pilha executa; e por fim,

5. o *checksum* deve estar correto.

Logo o protocolo indicado no cabeçalho é utilizado para determinar a próxima ação. Na uIP, apenas os protocolos TCP e ICMP são aceitos, sendo que o ICMP é reduzido ao tratamento de ECHO REQUEST's.

Se o pacote é um segmento TCP, a quádrupla (IP-origem,porta-origem,IP-destino,porta-destino) é utilizada para identificar a conexão à qual ele pertence. Se for encontrada uma conexão já iniciada, o segmento é entregue a ela. Caso contrário, se o segmento possuir o flag SYN ligado e a porta estiver no estado de escuta (LISTEN), é iniciado o *three-way handshake*. Para todos os outros pacotes um RESET é enviado, salvo em resposta a outro pacote de RESET.

Muitos dos recursos do TCP/IP não foram implementados na uIP. Como apenas um pacote IP pode ser enviado e recebido por vez, não há janelas deslizantes. Também não é realizado cálculo de RTT nem *backoff* de retransmissão. A única opção do TCP aceita é a de MSS.

Juntamente com a uIP, é distribuído um pequeno servidor web como exemplo de aplicação. Embora simples, o servidor permite o oferecimento de conteúdo dinâmico. As páginas estáticas são armazenadas como constantes no código-fonte, em uma estrutura de árvore semelhante a um sistema de arquivos.

O desenvolvedor que decidir usar a uIP não precisa especificar suas páginas como constantes. Em verdade, as páginas estáticas e dinâmicas podem ser elaboradas normalmente com um editor de textos ou de HTML e colocadas em um diretório fs/ dentro do código-fonte. Um script Perl se encarrega de transformar os arquivos desse diretório em um sistema de arquivos definido como constantes em linguagem C. As constantes são guardadas no arquivo fsdata.c, que é compilado juntamente com a pilha.

O conteúdo dinâmico é organizado em arquivos texto escritos em uma linguagem script simples, própria do servidor web da uIP. A linguagem permite a inserção de código HTML estático e a chamada de funções CGI. As funções CGI devem ser escritas em C e integradas à pilha uIP.

[1]Uma nova versão da uIP foi lançada recentemente, que suporta remontagem de fragmentos.

A pilha uIP caracteriza-se por não ter uma interface de programação bem definida. O usuário desenvolvedor fica exposto a detalhes de implementação. Mas isto é compensado pela redução no tamanho da pilha.

4.1.2 uIP no Z80

O transporte da uIP para o Z80 pode ser dividido nos seguintes passos:

1. Elaboração do Makefile
2. Alterações no *driver* de interface da uIP.
3. Alterações no simulador sz80.

Além disso, também foram necessários alguns pequenos ajustes no código da pilha para que pudesse ser compilada no SDCC. Por exemplo, o compilador SDCC não aceita atribuições de tipos agregados[1] e foi necessário reescrever essas atribuições.

A uIP é distribuída originalmente com código para duas arquiteturas, que utilizam interfaces distintas: uma implementação para o computador Commodore 64[2], contendo um *driver* para uma interface serial SLIP; e outra implementação que é uma simulação para o sistema Unix, utilizando o dispositivo TAP. A vantagem de utilizar este dispositivo é que a pilha uIP pode coexistir com a pilha nativa do sistema operacional. Este dispositivo TAP foi utilizado também na simulação para o Z80 e é descrito na seção 4.1.3.

Como não tínhamos o *hardware* disponível para testar a uIP no Z80, não foi necessário implementar um *driver* para o controlador Ethernet que seria utilizado (o chip CS8900). A fim de que a uIP para o Z80 pudesse ser testada, foram feitas modificações no simulador sz80 a fim de que o programa simulado, ou seja, a pilha uIP, tivesse acesso ao dispositivo TAP. Essas modificações formam o módulo *tapz80.cc* que foi anexado ao simulador sz80 e é descrito na seção 4.1.4.

A figura 4.1 ilustra o que foi feito. Na simulação original para o Unix, a uIP utiliza diretamente o dispositivo TAP (a). Na implementação para o Z80, a uIP comunica-se com o dispositivo TAP através do módulo tapsim (b). A figura ilustra também um processo "ping" residente na mesma máquina, comunicando-se com a pilha.

[1] Tipos agregados no C são *structs* e *unions*.
[2] O processador do Commodore 64 é o 6502.

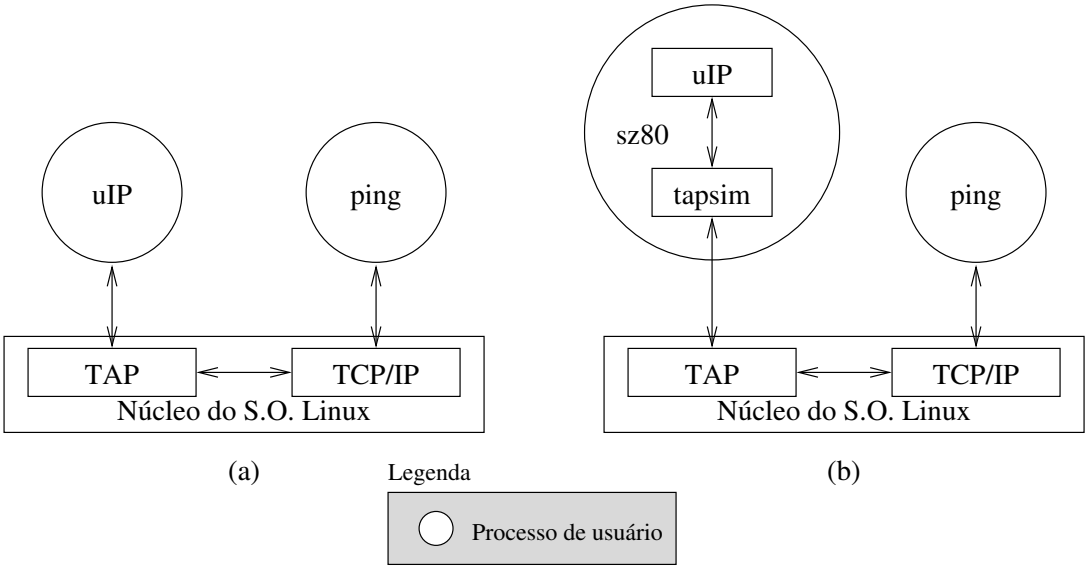

Figura 4.1: Comunicação com o dispositivo TAP: na implementação da uIP para Unix (a) e para o Z80 (b).

4.1.3 Dispositivo TAP

O dispositivo TAP [Kra02] não se trata propriamente de um adaptador de rede, mas de um mecanismo de comunicação disponível no sistema operacional no qual a camada física de uma interface Ethernet é substituída por um programa do usuário[1]. Em outras palavras, o *driver* TAP não envia um quadro para um adaptador de rede, mas para um programa do usuário. O programa do usuário, em nosso caso, é a simulação da uIP.

Para utilizar o driver TAP, o programa do usuário deve abrir o dispositivo /dev/net/tun. O *driver* registra no núcleo, então, uma interface de rede chamada tap$n$, com um endereço de Ethernet aleatório, escolhido dentre os endereços Ethernet reservados para uso local. Como podem existir vários dispositivos TAP em uma máquina, o índice n serve para identificá-los.

O endereço IP desta interface deve ser configurado pelo programa do usuário. Em nosso caso, este endereço IP serve de *gateway* entre a pilha TCP/IP do sistema e a simulação do uIP.

O *driver* TAP permite que um programa no modo usuário envie quadros Ethernet através de escritas no dispositivo /dev/net/tun. Cada quadro escrito neste dispositivo é

[1]Mais especificamente, um programa que opera em modo usuário do sistema operacional.

recebido pelo *driver*, que os redireciona para a interface de rede *tapn* conforme ilustra a figura 4.2. Logo a pilha TCP/IP do sistema operacional faz o envio destes quadros de acordo com a tabela de roteamento.

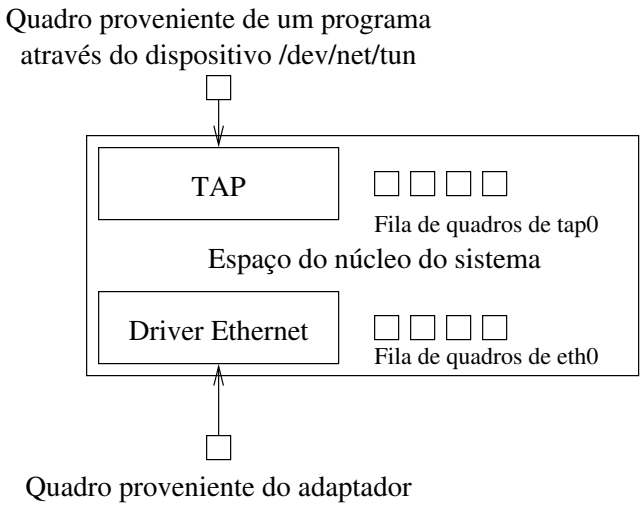

Figura 4.2: Comparação entre o driver TAP e o driver Ethernet.

A recepção de quadros pelo programa do usuário acontece de forma semelhante. Os quadros que chegam à interface tapn são redirecionados para o dispositivo /dev/net/tun, onde podem ser lidos pelo programa do usuário.

O driver TAP é distribuído em conjunto com o driver TUN em um pacote chamado "Universal TUN/TAP Driver" [Kra02]. A diferença entre os dois é que o dispositivo TAP trabalha com quadros Ethernet, enquanto o dispositivo TUN trabalha com pacotes IP. O driver TUN/TAP está disponível para os sistemas operacionais Solaris, FreeBSD e Linux, com algumas variações entre as implementações.

No Linux, o *driver* faz parte do núcleo. O arquivo /dev/net/tun é também utilizado para acessar o dispositivo TUN. O programa que desejar usar o dispositivo TAP em vez do TUN, deve indicá-lo através de uma chamada de sistema `ioctl()`. Essa é uma diferença de implementação do *driver* entre os sabores de Unix. Na implementação do FreeBSD, por exemplo, há vários dispositivos /dev/tap$n$ e /dev/tun$n$ e a chamada `ioctl()` não é necessária.

A simulação poderia também usar o dispositivo TUN. Porém, neste caso, em vez de enviar quadros Ethernet, seria necessário enviar pacotes IP. Não seria possível, então,

testar o protocolo ARP da pilha.

O código a seguir ilustra a utilização da interface tap no sistema operacional GNU/-Linux.

```
/* tapex.c
 * An example using the tap device on Linux
 * (c) Felipe Massia, 2002
 * $Id: tapex.c,v 1.3 2003/02/05 17:33:55 massia Exp $ */

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/if.h>
#include <linux/if_tun.h>

int fd, len;
unsigned char buf[1514];
struct ifreq ifr;

int
main (void)
{
  fd = open("/dev/net/tun",O_RDWR);

  /* on Linux we must tell which one we want: tun or tap */
  ifr.ifr_ifindex = 0;          /* given we got tap0 */
  ifr.ifr_flags = TUN_TAP_DEV;
  ioctl(fd,TUNSETIFF,&ifr);

  /* now configure the interface */
  system("/sbin/ifconfig␣tap0␣10.0.0.1␣netmask␣255.0.0.0");

  len = read(fd,buf,1514);
  printf("Received␣an␣Ethernet␣frame␣from␣%02x:%02x:%02x:%02x:%02
     x:%02x\n",
         buf[10],buf[11],buf[12],buf[13],buf[14],buf[15]);
    /* bytes [10..15] destination Ethernet address */

  return 0;
}
```

Após a execução deste código verifica-se a situação da interface tap0 com os utilitários

`ifconfig` e `route` conforme mostra a figura 4.3. O endereço de MAC da interface tap0 é um endereço aleatório escolhido pelo driver.

```
# ifconfig tap0
tap0      Link encap:Ethernet  HWaddr 00:FF:F8:36:31:19
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
143.106.24.64   *               255.255.255.192 U     0      0        0 eth0
10.0.0.0        *               255.0.0.0       U     0      0        0 tap0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         pascal.dcc.unic 0.0.0.0         UG    0      0        0 eth0
```

Figura 4.3: A interface tap0 e a tabela de roteamento durante a execução do programa exemplo.

O resultado do comando `ping 10.0.0.2`[1] resulta no recebimento de um quadro Ethernet pelo programa exemplo conforme mostrado nas figuras 4.4 e 4.5.

```
Received an Ethernet frame from 00:FF:F8:36:31:19
```

Figura 4.4: Saída do programa exemplo tapex.

[1]Não testamos o ping do próprio endereço do dispositivo tap, 10.0.0.1, pois o ECHO REQUEST é respondido pelo próprio núcleo, não sendo repassado à aplicação.

```
# tcpdump -i tap0
tcpdump: listening on tap0
19:26:44.843813 arp who-has 10.0.0.2 tell 10.0.0.1
```

Figura 4.5: Saída da escuta na rede com o programa tcpdump.

O quadro recebido é um pacote de requisição ARP, pois o núcleo não conhece ainda o endereço Ethernet correspondente ao número IP 10.0.0.2. Podemos notar também que o endereço Ethernet de origem é o endereço da interface tap0. Isso acontece porque o código de roteamento IP do sistema operacional verificou na tabela de rotas que a rede do número 10.0.0.2 (10.0.0.0) corresponde à rede desta interface. Mas antes de entregar o pacote ICMP de ECHO REQUEST enviado pelo programa ping ao host 10.0.0.2, o núcleo precisa saber o endereço Ethernet correspondente a este endereço IP. O núcleo envia então um ARP REQUEST. Foi o quadro deste ARP REQUEST, o primeiro a ser enviado, que foi recebido por nosso programa exemplo.

Quando se trata da simulação da uIP, esta solicitação é respondida com um endereço Ethernet fictício. Este endereço é então guardado pelo núcleo na sua tabela ARP como correspondente ao endereço 10.0.0.2. Logo o quadro contendo o pacote ICMP enviado pelo programa ping é enviado com o endereço Ethernet fictício para a interface tap0. Este pacote será entregue ao programa de simulação da uIP.

Em outras palavras, quando uma aplicação realiza uma leitura em um dispositivo tap, ela lê os bytes exatamente como chegaram da rede, incluindo o cabeçalho Ethernet. Essa aplicação pode, por exemplo, simular uma rede de computadores. O desenvolvedor pode então usar os aplicativos de rede disponíveis no UNIX como os utilitários `ping` e `tcpdump` e o navegador `links` para acessar essa rede.

Na simulação, a interface `tap`*n* é configurada com o endereço IP 192.168.0.1. No Linux, ao configurar o endereço IP da interface, uma rota para a rede classe C 192.168.0.0 já é implicitamente adicionada. Assim, a pilha TCP/IP do sistema operacional enviará todos os pacotes endereçados à rede 192.168.0.0 para a interface `tap`*n*.

4.1.4 Modificações no simulador `sz80`

Como o programa simulado não tem acesso ao sistema operacional, foram necessárias algumas modificações no `sz80` para que a uIP pudesse enviar quadros Ethernet, viabilizando, assim, o teste e a depuração da pilha de forma facilitada e a coexistência da pilha uIP com o a pilha TCP/IP do sistema operacional. O envio de quadros é feito através do dispositivo TAP e as modificações no simulador visaram permitir que a uIP tivesse acesso às chamadas de sistema relacionadas à manipulação deste dispositivo.

Três endereços de memória foram reservados no espaço de endereçamento do Z80[1]. Estes endereços permitem a troca de informações entre o programa sendo simulado (a uIP) e o módulo que agregamos ao simulador (o tapz80). O tapz80 é o módulo que realiza a interface entre a uIP e o sistema operacional Linux.

Os endereços foram definidos como constantes iniciando com o caractere sublinhado (_). Estas constantes serão acessadas apenas pelo código do simulador. O código da pilha uIP utiliza variáveis de mesmo nome, mas sem o sublinhado inicial, definidas no cabeçalho `tapsim.h` conforme segue[2]:

```
#define _TAP_BUFPTR     0xe0    // tap buffer addr
#define _TAP_COM        0xe4    // tap commands
#define _TAP_SIZE       0xe6    // tap read/write result (int)

#define TAP_BUFPTR      (*((unsigned int *) _TAP_BUFPTR))
#define TAP_COM         (*((unsigned char *) _TAP_COM))
#define TAP_SIZE        (*((int *) _TAP_SIZE))
```

`_TAP_BUFPTR` *(2 bytes)* Contém o endereço de início do *buffer* usado para transferir dados. Também é utilizado como ponteiro para o endereço IP que deve ser configurado no dispositivo TAP, no momento da inicialização.

`_TAP_SIZE` *(2 bytes)* Usado principalmente para passar e retornar o número de bytes lidos ou escritos.

`_TAP_COM` *(1 byte)* Atua como uma porta de comando. O simulador foi modificado para que uma escrita neste endereço ocasione uma chamada ao sistema operacional. O valor

[1] Este espaço de endereçamento no simulador é um vetor de 64 Kbytes.

[2] Esta listagem e as seguintes foram reformatadas para melhor visualização. Alguns `#ifdef`'s também foram removidos com o mesmo objetivo.

gravado na variável indica a chamada a ser realizada, conforme segue:

`TC_INIT` Inicializa o dispositivo TAP. Os 4 bytes localizados no endereço de memória apontado por `TAP_BUFPTR` são utilizados como endereço IP para a interface TAP. Esta operação é executada uma única vez no início da simulação.

`TC_POLL` Verifica se há dados a serem lidos do dispositivo TAP. O resultado é retornado em `TAP_SIZE`. Se há dados prontos para serem lidos, o seu valor é 1. Caso contrário é 0. A operação `TC_POLL` é executada constantemente a partir do laço principal da uIP, descrito a seguir:

```
while (1) {
  ...
  uip_len = tapdev_poll(); /* funcao que chama
     TC_POLL */
  ...

  if (uip_len > 0)
    uip_len = tapdev_read(); /* funcao que chama
       TC_READ */
  ...
}
```

`Tapdev_poll()` e `tapdev_read()` são funções "invólucro". Elas escrevem o valor da operação em `TAP_COM` e obtêm o resultado de `TAP_SIZE`. A implementação da função `tapdev_poll()` é mostrada abaixo:

```
unsigned int
tapdev_poll(void)
{
  TAP_COM = TC_POLL;
  return TAP_SIZE;
}
```

`TC_READ` Lê no máximo `TAP_SIZE` bytes do dispositivo TAP para o endereço de memória apontado por `TAP_BUFPTR`. O número de bytes efetivamente lidos é retornado em `TAP_SIZE`.

`TC_WRITE` Copia `TAP_SIZE` bytes localizados no endereço de memória apontado por `TAP_BUFPTR` para um *buffer* interno do simulador chamado `tap_buf`. Sucessivas cópias são

cumulativas, ou seja, cada chamada `TC_WRITE` anexa os bytes no *buffer*, logo após os bytes escritos em chamadas anteriores. O *buffer* é esvaziado a cada chamada `TC_SEND`.

`TC_SEND` Envia os dados contidos no *buffer* `tap_buf` através do dispositivo TAP. O número de bytes efetivamente enviados é retornado em `TAP_SIZE`. O *buffer* `tap_buf` é esvaziado.

O módulo tapz80[1] contém a implementação das operações mencionadas. Ele é composto basicamente de rotinas que manipulam um descritor do arquivo `/dev/net/tun`, controladas pelas escritas no endereço `_TAP_COM`. O simulador foi alterado para que uma escrita na memória simulada do Z80 gerasse uma chamada à função `tap_com()`, descrita abaixo:

```
void
cl_z80::tap_com (t_addr addr, int val)
{
  if (addr != _TAP_COM)
    {
      ram->set((t_addr) (addr), val);
      return;
    }

  if (val != TC_POLL)
    printf("SIM: %x <- %x\n",(int) addr,val);

  switch(val)
    {
    case TC_INIT:  tap_init();  break;
    case TC_READ:  tap_read();  break;
    case TC_WRITE: tap_write(); break;
    case TC_SEND:  tap_send();  break;
    case TC_POLL:  tap_poll();  break;
    case TC_EXIT:  state = SIM_QUIT;  break;
    default: printf("TAP_COM ERROR!\n");  break;
    }
}
```

Como esta função interceptará cada escrita na memória, ela primeiro verifica se o endereço sendo acessado é o endereço `_TAP_COM`. Se não for, a chamada original de escrita

[1]Este módulo é um arquivo C++ chamado `tapz80.cc`.

em memória é executada. Caso contrário, a função correspondente ao valor escrito é chamada.

A chamada `printf()` anterior ao comando `switch()` tem fins de depuração. Como o comando `TC_POLL` é muito frequente, ele é omitido para não poluir a saída.

Com estas cinco operações implementadas, foi possível que a pilha uIP rodando no simulador do Z80 enviasse quadros Ethernet para a rede real.

Foi tomado como base para esta implementação o código da simulação da uIP para o Unix. Foram realizadas mudanças pequenas na pilha original para o seu funcionamento no simulador.

O arquivo mais alterado foi o arquivo `tapdev.c`, que trata da comunicação da uIP com o dispositivo TAP. Na simulação original, a uIP tinha acesso ao sistema operacional enquanto na simulação no sz80 não. Este código que manipulava diretamente o dispositivo TAP foi movido para o simulador sz80 e adaptado a ele. No seu lugar, foram colocadas rotinas que utilizam as variáveis `TAP_BUFPTR`, `TAP_SIZE` e `TAP_COM` de acordo com a semântica descrita acima.

4.1.5 Tamanho do código

O código-objeto da uIP para o Z80, incluindo o servidor web, ocupou cerca de 19 Kbytes, conforme mostra a tabela 4.2. Subtraindo-se o espaço ocupado pelas páginas exemplo, temos que a pilha ocupa cerca de 11,2 Kbytes.

Tabela 4.2: Tamanho do código-objeto da uIP no Z80.

Módulo	Tamanho (em bytes)
ARP	1890
Interface (TAP)	237
Núcleo TCP/IP	5207
Servidor Web	1911
Checksum	617
Auxiliares	1808
Subtotal	**11670**
Páginas exemplo	7766
Total	**19436**

Consideramos que o resultado do tamanho do código é muito satisfatório. O Z80

possui um espaço de endereçamento de 64Kb. Isso significa que o servidor web da pilha uIP com algumas páginas exemplo ocupa apenas 30% do espaço de endereçamento do processador Z80.

Porém é importante lembrar que o *driver* da interface Ethernet representaria um considerável aumento no tamanho total da pilha. Como fizemos uma simulação, o código do *driver* foi substituído por acessos à memória conforme mencionado na seção anterior.

4.2 lwIP no Z80

Numa segunda etapa da implementação, a pilha lwIP versão 0.5.3 foi transportada para o Z80. Utilizamos o mesmo ambiente de desenvolvimento e de simulação mencionados na seção anterior (compilador SDCC e simulador sz80).

Inicialmente houve problemas com a função de cálculo do *checksum*: o compilador não gerou código correto para as expressões. O código C da função foi substituído pelo código assembly da pilha ZSock, mencionada na seção 3.1.4, com algumas modificações que tiveram a finalidade de adequar o retorno do valor da função conforme o padrão do compilador.

Porém, surgiram dois problemas. O primeiro foi a dificuldade de depuração da pilha, já que o SDCC não possui um depurador funcional. A lwIP conta com rotinas próprias para exibir os pacotes recebidos para fins de depuração, porém elas ocupam muita memória e ao incluí-las, o código extrapolava o limite de 64Kb.

O segundo problema é exatamente este: o tamanho do código-objeto. Numa configuração com suporte a TCP, IP e ICMP e um servidor de eco, ou seja, sem o servidor Web, a pilha já chegou aos 52Kb. Embora não tenha atingido os 64Kb, mesmo que o servidor web coubesse, não sobraria muito espaço para as páginas.

4.3 Implementação em hardware

Nesta seção descrevemos o hardware que seria utilizado na integração da pilha TCP/IP. Conforme mencionado no início deste capítulo, seriam utilizados módulos de controle de acesso disponíveis no Instituto. Basicamente um módulo é composto de:

- micro-controlador Z84C15;
- 32KB de RAM;

- 32KB de ROM (EPROM apagável com ultra-violeta);
- cristal oscilador de 3.6864 MHz;
- uma tela de cristal líquido.

O Z84C15 é um controlador de periféricos baseado no Z80. Ele contém em um só chip os seguintes dispositivos[Zil97]:

- CPU Z80;
- 2 entradas/saídas seriais;
- entrada/saída paralela de 16 bits;
- 4 circuitos contadores/temporizadores;
- temporizador *watch-dog*.

Para conexão à rede Ethernet, seria utilizado uma placa com o chip CS8900A, da Cirrus Logic. Este chip foi descrito na seção 2.3.3. A placa seria construída com base no projeto de referência do fabricante para operação em 8 bits [Ayr00] e conectada ao módulo através da E/S paralela do Z84C15.

Capítulo 5

Conclusões

A proposta inicial deste projeto era a de revisar o material existente na área de TCP/IP para sistemas embarcados e integrar uma pilha TCP/IP a um sistema de 8 ou 16 bits com adaptador de rede Ethernet.

No decorrer do trabalho, obtivemos:

- Um overview do estado da arte das pilhas TCP/IP para sistemas embarcados, bem como o hardware utilizado nesses sistemas;
- O transporte e a simulação da pilha uIP incluindo um servidor web para o processador Z80 da Zilog;

5.1 O overview

A enumeração e análise das soluções existentes para a integração do protocolo TCP/IP a sistemas embarcados constitui uma das contribuições deste trabalho.

A integração dos protocolos da Internet a sistemas embarcados é um tema novo e pouco explorado no meio acadêmico. Consideramos que esta dissertação também contribui no sentido de colocar o tema neste âmbito.

5.2 uIP para o Z80

A pilha uIP, de autoria de Adam Dunkels, foi transportada para o Z80. Embora a pilha não tenha sido integrada a um equipamento real, uma simulação foi realizada no sistema

operacional GNU/Linux. O simulador sz80 foi alterado a fim de que a pilha uIP pudesse se comunicar com o sistema operacional e a rede externa. Foi possível usar um navegador para visitar páginas contidas no servidor web da pilha uIP que estava sendo executada no sz80.

5.3 Dificuldades

Entre as dificuldades encontradas no decorrer do trabalho, vale ressaltar a procura de um ambiente de desenvolvimento para o Z80. Este barreira foi transposta com o achado do SDCC, da GNU. O SDCC forneceu não só o compilador mas também o simulador, fechando o ciclo de desenvolvimento. O simulador, além de executar o programa compilado, também permite visualizar o conteúdo da memória simulada em forma de dados e de código. Desde o início deste trabalho, o SDCC evoluiu bastante, inclusive com *bug reports* do autor desta dissertação.

5.4 Trabalhos futuros

Acreditamos que este trabalho possa ser estendido em várias direções. O próximo passo desse trabalho seria a integração da pilha uIP a um sistema baseado no Z80 para testes em condições reais.

A seguir citamos outras possibilidades:

Throughput Uma análise do número de *hits* e tempo de resposta de servidores web para sistemas embarcados;

Segurança Novas implementações da pilha TCP/IP, visando a ligação de sistemas embarcados a redes externas, podem conter furos de segurança. Estes furos podem ser explorados por usuários mal intencionados. Durante esse trabalho surpreendeu a falta de preocupação em relação à segurança por parte dos autores e fornecedores.

Wireless Há uma grande demanda pela utilização do TCP/IP em dispositivos *wireless*, tema que não foi o foco dessa dissertação.

Apêndice A

Ferramentas

As seguintes ferramentas foram utilizadas nos testes da uIP:

ping O programa ping serve para enviar pacotes ICMP de ECHO REQUEST a um determinado host a fim de verificar se ele está ativo. Além de mostrar se o host está ativo ou não, o ping também mostra o tempo decorrido entre a solicitação e a resposta. Está presente na maioria dos sistemas operacionais com suporte a rede.

links O links é um navegador para console (modo texto), muito similar – até no nome – ao programa **lynx**, de mesma funcionalidade. O links foi utilizado para acessar o servidor web da pilha uIP.

Site: `http://links.sourceforge.net/`

Galeon O Galeon é outro navegador web para o ambiente gráfico GNOME. Ele é baseado no Mozilla, que por sua vez é baseado no Netscape. O Galeon foi utilizado para acessar as páginas do servidor web da uIP.

Site: `http://galeon.sourceforge.net/`

tcpdump O tcpdump é um *sniffer*, ou seja, ele "escuta" os pacotes que trafegam pela rede local e os exibe num formato palatável. O **snoop** e o **iptrace** sao programas similares disponíveis para Solaris e AIX, respectivamente. O tcpdump foi utilizado para visualizar a interação da pilha uIP com outros utilitários. O apêndice B contém uma saída do tcpdump para um teste com o simulador.

Site: `http://www.tcpdump.org/`

Apêndice B

Exemplo de execução

Neste apêndice apresentamos um teste executado com a uIP. Após o início da simulação, os seguintes comandos foram executados no sistema operacional GNU/Linux:

```
$ ping -c 4 192.168.0.2
...
$ links 192.168.0.2
...
```

O parâmetro '-c' do comando ping é utilizado para especificar o número de pacotes de `echo request` a serem enviados.

A seguir apresentamos a saída do comando `tcpdump -n -i tap0` executado em paralelo ao teste[1]. O parâmetro '-n' serve para o tcpdump exibir endereços numéricos e o argumento '-i tap0' especifica a interface a ser escutada. O endereço da pilha uIP é 192.168.0.2 e o endereço utilizado pelo sistema é o 192.168.0.1.

Notamos que a primeira ação realizada pelo sistema Linux é descobrir o endereço de hardware correspondente ao endereço IP 192.168.0.2. A pilha uIP responde com o endereço 0:bd:3b:33:5:71.

```
tcpdump: listening on tap0
08:41:27.648160 arp who-has 192.168.0.2 tell 192.168.0.1
08:41:27.767087 arp reply 192.168.0.2 is-at 0:bd:3b:33:5:71
```

[1] A saída foi quebrada para permitir a intercalação dos comentários.

Nas próximas 8 linhas vemos a seqüência de 4 pings respondidos pela uIP.

```
08:41:27.767153 192.168.0.1 > 192.168.0.2: icmp: echo request (DF)
08:41:27.892181 192.168.0.2 > 192.168.0.1: icmp: echo reply (DF)
08:41:28.642763 192.168.0.1 > 192.168.0.2: icmp: echo request (DF)
08:41:28.748013 192.168.0.2 > 192.168.0.1: icmp: echo reply (DF)
08:41:29.642628 192.168.0.1 > 192.168.0.2: icmp: echo request (DF)
08:41:29.762334 192.168.0.2 > 192.168.0.1: icmp: echo reply (DF)
08:41:30.642990 192.168.0.1 > 192.168.0.2: icmp: echo request (DF)
08:41:30.750917 192.168.0.2 > 192.168.0.1: icmp: echo reply (DF)
```

Em seguida aparece o estabelecimento da conexão TCP para o navegador links. As 3 linhas de saída foram quebradas para encaixar na página. O endereço de destino do primeiro pacote é 192.168.0.2:80 indicando um acesso à porta 80 da pilha uIP. A letra 'S' logo após, indica que o flag SYN do TCP está ligado. Na segunda linha notamos que no pacote de resposta, a uIP especifica uma janela de 126 bytes e um segmento máximo também de 126 bytes.

```
08:41:49.702254 192.168.0.1.55321 > 192.168.0.2.80:
    S 229767172:229767172(0) win 5840
    <mss 1460,sackOK,timestamp 17390665 0,nop,wscale 0> (DF)
08:41:49.869577 192.168.0.2.80 > 192.168.0.1.55321:
    S 4278190629:4278190629(0) ack 229767173 win 126 <mss 126>
08:41:49.869715 192.168.0.1.55321 > 192.168.0.2.80:
    . ack 1 win 5840 (DF)
```

A seguir tem início a transferência da primeira página (/index.html). A página exemplo tem cerca de 1Kb. Notamos que as pilhas cumprem o estabelecido inicialmente em relação ao tamanho máximo de segmento e tamanho de janela.

```
08:41:49.870882 192.168.0.1.55321 > 192.168.0.2.80: . 1:64(63) ack 1 win 5840 (DF)
08:41:49.870915 192.168.0.1.55321 > 192.168.0.2.80: P 64:127(63) ack 1 win 5840 (DF)
08:41:50.349787 192.168.0.2.80 > 192.168.0.1.55321: . 1:127(126) ack 64 win 126
08:41:50.349922 192.168.0.1.55321 > 192.168.0.2.80: . 127:190(63) ack 127 win 5840 (DF)
08:41:50.791822 192.168.0.2.80 > 192.168.0.1.55321: . ack 127 win 126
08:41:50.791923 192.168.0.1.55321 > 192.168.0.2.80: . ack 127 win 5840 (DF)
08:41:51.082015 192.168.0.2.80 > 192.168.0.1.55321: . 127:253(126) ack 190 win 126
08:41:51.082165 192.168.0.1.55321 > 192.168.0.2.80: P 190:253(63) ack 253 win 5840 (DF)
08:41:51.082183 192.168.0.1.55321 > 192.168.0.2.80: . 253:316(63) ack 253 win 5840 (DF)
08:41:51.579712 192.168.0.2.80 > 192.168.0.1.55321: . 253:379(126) ack 253 win 126
08:41:51.579843 192.168.0.1.55321 > 192.168.0.2.80: P 316:379(63) ack 379 win 5840 (DF)
08:41:51.735168 192.168.0.2.80 > 192.168.0.1.55321: . ack 316 win 126
08:41:51.735230 192.168.0.1.55321 > 192.168.0.2.80: . ack 379 win 5840 (DF)
08:41:51.939363 192.168.0.2.80 > 192.168.0.1.55321: . 379:505(126) ack 379 win 126
```

```
08:41:51.939470 192.168.0.1.55321 > 192.168.0.2.80: P 379:429(50) ack 505 win 5840 (DF)
08:41:52.765636 192.168.0.2.80 > 192.168.0.1.55321: . 505:631(126) ack 429 win 126
08:41:52.802553 192.168.0.1.55321 > 192.168.0.2.80: . ack 631 win 5840 (DF)
08:41:53.066329 192.168.0.2.80 > 192.168.0.1.55321: . 631:757(126) ack 429 win 126
08:41:53.066472 192.168.0.1.55321 > 192.168.0.2.80: . ack 757 win 5840 (DF)
08:41:53.218115 192.168.0.2.80 > 192.168.0.1.55321: . 757:883(126) ack 429 win 126
08:41:53.218184 192.168.0.1.55321 > 192.168.0.2.80: . ack 883 win 5840 (DF)
08:41:53.407716 192.168.0.2.80 > 192.168.0.1.55321: . 883:1009(126) ack 429 win 126
08:41:53.407814 192.168.0.1.55321 > 192.168.0.2.80: . ack 1009 win 5840 (DF)
08:41:53.565552 192.168.0.2.80 > 192.168.0.1.55321: . 1009:1135(126) ack 429 win 126
08:41:53.565678 192.168.0.1.55321 > 192.168.0.2.80: . ack 1135 win 5840 (DF)
08:41:53.727602 192.168.0.2.80 > 192.168.0.1.55321: . 1135:1261(126) ack 429 win 126
08:41:53.727716 192.168.0.1.55321 > 192.168.0.2.80: . ack 1261 win 5840 (DF)
08:41:53.893316 192.168.0.2.80 > 192.168.0.1.55321: . 1261:1387(126) ack 429 win 126
08:41:53.893419 192.168.0.1.55321 > 192.168.0.2.80: . ack 1387 win 5840 (DF)
08:41:54.127877 192.168.0.2.80 > 192.168.0.1.55321: . 1387:1513(126) ack 429 win 126
08:41:54.127980 192.168.0.1.55321 > 192.168.0.2.80: . ack 1513 win 5840 (DF)
08:41:54.282160 192.168.0.2.80 > 192.168.0.1.55321: . 1513:1559(46) ack 429 win 126
08:41:54.282257 192.168.0.1.55321 > 192.168.0.2.80: . ack 1559 win 5840 (DF)
```

Por fim, a conexão é terminada. A letra 'F' na saída indica que o flag FIN está ligado. Quem termina a conexão é a uIP indicando fim do arquivo. A linha final é emitida quando a simulação é abortada e o dispositivo tap0, fechado.

```
08:41:54.390052 192.168.0.2.80 > 192.168.0.1.55321:
    F 1559:1559(0) ack 429 win 126
08:41:54.390444 192.168.0.1.55321 > 192.168.0.2.80:
    F 429:429(0) ack 1560 win 5840 (DF)
08:41:54.519985 192.168.0.2.80 > 192.168.0.1.55321:
    . ack 430 win 126
tcpdump: pcap_loop: recvfrom: Network is down
```

Apêndice C

uIP - Exemplo de Sistema de Arquivos

Este apêndice visa ilustrar como é construído o sistema de arquivos da pilha uIP. Este sistema é estático, pois é construído em tempo de compilação e guardado na forma de constantes em um programa C, o `fsdata.c`. O programa C é gerado automaticamente por um script Perl que é distribuído juntamente com a pilha, o `makefsdata`, a partir dos arquivos contidos no diretório `fs/`. Isso facilita a criação do site, pois o desenvolvedor só precisa colocar as páginas e outros arquivos no diretório `fs/` e então executar o script. O resultado é o programa C já mencionado, que será compilado e ligado à pilha uIP.

O arquivo `fsdata.c` é formado por três seções:

Vetores de arquivos Cada arquivo é descrito por um vetor de bytes (`char []`) inicializado com o conteúdo a ser servido em formato hexadecimal.

Na verdade, antes de codificar o arquivo em um vetor de bytes, o script adiciona o cabeçalho utilizado no protocolo HTTP 1.0. Este cabeçalho é composto pelas três seguintes linhas:

```
HTTP/1.0 200 OK
Server: uIP/0.6 (http://dunkels.com/adam/uip/)
Content-type: text/html
```

O campo `Content-type` varia de acordo com o tipo do arquivo sendo codificado. Os formatos suportados pelo script são o HTML, JPG, PNG e GIF. Outros tipos de arquivos são considerados como `text/plain`.

Estrutura O sistema de arquivos é descrita através de uma lista ligada. A lista ligada é definida pela seguinte estrutura.

```
struct fsdata_file {
  const struct fsdata_file *next; /* proximo arquivo */
  const char *f_name;   /* nome do arquivo */
  const char *f_data;   /* aponta para o vetor de dados
      */
  const int f_len;      /* tamanho do arquivo em bytes */
};
```

Raiz A constante correspondente ao primeiro arquivo da lista (raiz) é identificada através de uma diretiva `#define`. Também o número de arquivos é definido. Estes dados são usados pelo servidor para procurar um arquivo solicitado.

C.1 Exemplo

A seguir é apresentado um exemplo simples do sistema de arquivos utilizado no servidor web da uIP. O diretório `fs/` contém dois arquivo apenas: a página principal (index.html) e um texto (texto.txt).

Arquivo index.html:

```
<HTML>
<HEAD><TITLE>index.html - Exemplo</TITLE></HEAD>
<BODY>
<A HREF="texto.txt">Link para um texto</A>
</BODY>
</HTML>
```

Arquivo texto.txt:

```
Um texto de exemplo.
Fim
```

O arquivo `fsdata.c` gerado pelo script é mostrado a seguir.

Arquivo fsdata.c:

```
static const char data_index_html[] = {
/* /index.html */
0x2f, 0x69, 0x6e, 0x64, 0x65, 0x78, 0x2e, 0x68, 0x74, 0x6d, 0x6c
   , 0,
0x48, 0x54, 0x54, 0x50, 0x2f, 0x31, 0x2e, 0x30, 0x20, 0x32,
0x30, 0x30, 0x20, 0x4f, 0x4b, 0xd, 0xa, 0x53, 0x65, 0x72,
0x76, 0x65, 0x72, 0x3a, 0x20, 0x75, 0x49, 0x50, 0x2f, 0x30,
0x2e, 0x36, 0x20, 0x28, 0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f,
0x2f, 0x64, 0x75, 0x6e, 0x6b, 0x65, 0x6c, 0x73, 0x2e, 0x63,
0x6f, 0x6d, 0x2f, 0x61, 0x64, 0x61, 0x6d, 0x2f, 0x75, 0x69,
0x70, 0x2f, 0x29, 0xd, 0xa, 0x43, 0x6f, 0x6e, 0x74, 0x65,
0x6e, 0x74, 0x2d, 0x74, 0x79, 0x70, 0x65, 0x3a, 0x20, 0x74,
0x65, 0x78, 0x74, 0x2f, 0x68, 0x74, 0x6d, 0x6c, 0xd, 0xa,
0xd, 0xa, 0x3c, 0x48, 0x54, 0x4d, 0x4c, 0x3e, 0xa, 0x3c,
0x48, 0x45, 0x41, 0x44, 0x3e, 0x3c, 0x54, 0x49, 0x54, 0x4c,
0x45, 0x3e, 0x69, 0x6e, 0x64, 0x65, 0x78, 0x2e, 0x68, 0x74,
0x6d, 0x6c, 0x20, 0x2d, 0x20, 0x45, 0x78, 0x65, 0x6d, 0x70,
0x6c, 0x6f, 0x3c, 0x2f, 0x54, 0x49, 0x54, 0x4c, 0x45, 0x3e,
0x3c, 0x2f, 0x48, 0x45, 0x41, 0x44, 0x3e, 0xa, 0x3c, 0x42,
0x4f, 0x44, 0x59, 0x3e, 0xa, 0x3c, 0x41, 0x20, 0x48, 0x52,
0x45, 0x46, 0x3d, 0x22, 0x74, 0x65, 0x78, 0x74, 0x6f, 0x2e,
0x74, 0x78, 0x74, 0x22, 0x3e, 0x4c, 0x69, 0x6e, 0x6b, 0x20,
0x70, 0x61, 0x72, 0x61, 0x20, 0x75, 0x6d, 0x20, 0x74, 0x65,
0x78, 0x74, 0x6f, 0x3c, 0x2f, 0x41, 0x3e, 0xa, 0x3c, 0x2f,
0x42, 0x4f, 0x44, 0x59, 0x3e, 0xa, 0x3c, 0x2f, 0x48, 0x54,
0x4d, 0x4c, 0x3e, 0xa, };

static const char data_texto_txt[] = {
/* /texto.txt */
0x2f, 0x74, 0x65, 0x78, 0x74, 0x6f, 0x2e, 0x74, 0x78, 0x74, 0,
0x48, 0x54, 0x54, 0x50, 0x2f, 0x31, 0x2e, 0x30, 0x20, 0x32,
0x30, 0x30, 0x20, 0x4f, 0x4b, 0xd, 0xa, 0x53, 0x65, 0x72,
0x76, 0x65, 0x72, 0x3a, 0x20, 0x75, 0x49, 0x50, 0x2f, 0x30,
0x2e, 0x36, 0x20, 0x28, 0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f,
0x2f, 0x64, 0x75, 0x6e, 0x6b, 0x65, 0x6c, 0x73, 0x2e, 0x63,
0x6f, 0x6d, 0x2f, 0x61, 0x64, 0x61, 0x6d, 0x2f, 0x75, 0x69,
0x70, 0x2f, 0x29, 0xd, 0xa, 0x43, 0x6f, 0x6e, 0x74, 0x65,
0x6e, 0x74, 0x2d, 0x74, 0x79, 0x70, 0x65, 0x3a, 0x20, 0x74,
0x65, 0x78, 0x74, 0x2f, 0x70, 0x6c, 0x61, 0x69, 0x6e, 0xd,
0xa, 0xd, 0xa, 0x55, 0x6d, 0x20, 0x74, 0x65, 0x78, 0x74,
0x6f, 0x20, 0x64, 0x65, 0x20, 0x65, 0x78, 0x65, 0x6d, 0x70,
0x6c, 0x6f, 0x2e, 0xa, 0x46, 0x69, 0x6d, 0xa, };
```

```
const struct fsdata_file file_index_html =
  {NULL, data_index_html, &data_index_html[12], sizeof(
     data_index_html) - 12};

const struct fsdata_file file_texto_txt =
  {&file_index_html, data_texto_txt, &data_texto_txt[11], sizeof(
     data_texto_txt) - 11};

#define FS_ROOT file_texto_txt

#define FS_NUMFILES 2
```

Apêndice D

Código-fonte

A seguir é apresentado o código-fonte dos principais arquivos da uIP e do SDCC modificados no transporte. São eles:

`uip/tapdev.c` Este módulo da uIP contém as rotinas de escrita e leitura na memória simulada do Z80. Essas rotinas formam um "driver" de acesso à rede para a uIP.

`sdcc/tapsim.h` Este cabeçalho contém constantes que dão significado aos endereços da memória simulada e constantes dos comandos entendidos pelo dispositivo TAP na simulação.

`sdcc/tapz80.cc` Este módulo contém as rotinas que lidam diretamente com o dispositivo TAP. Elas fazem parte do simulador sz80 e são executadas de acordo com as chamadas em `uip/tapdev.c`.

Como o simulador sz80, que faz parte do compilador SDCC, precisa ser alterado para a simulação funcionar, foi elaborado um *patch* que contém todas as alterações no formato de saída do programa `diff`. Este *patch* é distribuído juntamente com a uIP para o Z80. Ele pode ser aplicado ao código-fonte do compilador com o programa `patch`. Um alvo dentro do Makefile foi criado para realizar a aplicação do *patch* automaticamente, facilitando a compilação.

D.1 uip/tapdev.c

```
/*
 * Author: Felipe Pereira <felipe.pereira@ic.unicamp.br>
 * based on UNIX tapdev.c by Adam Dunkels
 *
 * $Id: tapdev.c,v 1.1.1.1 2002/04/02 01:22:54 felipe Exp $
 */

#include "uip.h"
#include "tapsim.h"
#include <stdio.h>

static unsigned char ip_addr[4] =
  { UIP_DRIPADDR0, UIP_DRIPADDR1, UIP_DRIPADDR2, UIP_DRIPADDR3 };

/*
char *
dec2hex(int val)
{
  static char hex_buf[5];
  int c;
  hex_buf[2] = 0;
  c = val/16;
  hex_buf[0] = c>9?('a'+c-10):('0'+c);
  c = val%16;
  hex_buf[1] = c>9?('a'+c-10):('0'+c);
  return hex_buf;
}

void
dump_buf(unsigned char *buf, int len)
{
  int i;
  printf("dump_buf: len=%d\n",len);
  for(i=0; i<len; i++) {
    printf("%s ",dec2hex((unsigned int) buf[i]));
    if ((i+1)%8 == 0) {
      if ((i+1)%16 == 0){
        printf("\n");
      } else {
        printf("  ");
      }
    }
  }
  printf("\n");
}

*/
```

```
/*---------------------------------------------------------*/
void
tapdev_init(void)
{
  TAP_BUFPTR = (unsigned int) &ip_addr[0];
  TAP_COM = TC_INIT;
}
/*---------------------------------------------------------*/
unsigned int
tapdev_poll(void)
{
  TAP_COM = TC_POLL;
  return TAP_SIZE;
}
/*---------------------------------------------------------*/
unsigned int
tapdev_read(void)
{
  TAP_BUFPTR = (unsigned int) uip_buf;
  TAP_SIZE = UIP_BUFSIZE;
  TAP_COM = TC_READ;
  return TAP_SIZE;
}
/*---------------------------------------------------------*/
void
tapdev_send(void)
{
  TAP_BUFPTR = (unsigned int) uip_buf;
  if (uip_len>40+UIP_LLH_LEN)
    TAP_SIZE = 40+UIP_LLH_LEN;
  else
    TAP_SIZE = uip_len;
  TAP_COM = TC_WRITE;

  if (uip_len>40+UIP_LLH_LEN) {
    TAP_BUFPTR = (unsigned int) uip_appdata;
    TAP_SIZE = (int)uip_len-40-UIP_LLH_LEN;
    TAP_COM = TC_WRITE;
  }

  TAP_COM = TC_SEND;
}
/*---------------------------------------------------------*/
```

D.2 sdcc/tapsim.h

```
#ifndef _TAPSIM_H
#define _TAPSIM_H
```

```
#define _TAP_BUFPTR     0xe0    // tap buffer addr
#define _TAP_COM        0xe4    // tap command
#define _TAP_SIZE       0xe6    // tap read/write result (int)

#define TAP_BUFPTR      (*(unsigned int *) _TAP_BUFPTR)
#define TAP_COM         (*(unsigned char *) _TAP_COM)
#define TAP_SIZE        (*(int *) _TAP_SIZE)

// tap commands
#define TC_INIT         0  // init (nothing for now)
#define TC_READ         1  // read to buffer
#define TC_WRITE        2  // write to tap_buf
#define TC_SEND         3  // send tap_buf to tap
#define TC_POLL         4  // packet arrival?
#define TC_EXIT         10  // generates INV_INST in place of exit()

#endif /* _TAPSIM_H */
```

D.3 sdcc/tapz80.cc

```
/* Tap hack for sz80 by Felipe Pereira */

#include "tap/tapsim.h"

int tap_fd;
unsigned char tap_buf[1520]; // suffices for Ethernet

void
cl_z80::tap_com(t_addr addr, int val) {


  if (addr != _TAP_COM) {
    ram->set((t_addr) (addr), val);
    return;
  }

  if (val != TC_POLL)
    printf("SIM: %x <- %x\n",(int) addr,val);

  switch(val) {
  case TC_INIT:
    tap_init();
    break;

  case TC_READ:
    tap_read();
    break;
```

```
    case TC_WRITE:
      tap_write();
      break;

    case TC_SEND:
      tap_send();
      break;

    case TC_POLL:
      tap_poll();
      break;

    case TC_EXIT:
      state = SIM_QUIT;
      break;

    default:
      break;
    }
}

#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/if.h>
#include <linux/if_tun.h>
#include <sys/ioctl.h>
#include <fcntl.h>

#include "tap/tapsim.h"
#include "z80mac.h"

void
cl_z80::tap_init(void)
{
  char buf[1024];
  struct ifreq ifr;
  unsigned int ip_addr_addr;

  tap_fd = open("/dev/net/tun",O_RDWR);
  if (tap_fd < 0) {
    perror("tap_init:_open");
    exit(1);
  }


  memset(&ifr, 0, sizeof(ifr));
  ifr.ifr_flags = IFF_TAP|IFF_NO_PI;
```

```
  if (ioctl(tap_fd, TUNSETIFF, (void *) &ifr) < 0) {
    perror("tap_init: ioctl");
    exit(1);
  }

  ip_addr_addr = get2(_TAP_BUFPTR);
  snprintf(buf, sizeof(buf), "ifconfig tap0 inet %d.%d.%d.%d",
           (int)get1((t_addr)ip_addr_addr+0),
           (int)get1((t_addr)ip_addr_addr+1),
           (int)get1((t_addr)ip_addr_addr+2),
           (int)get1((t_addr)ip_addr_addr+3));
  system(buf);

  printf("SIM: tap initialized: %s\n",buf);
}

void
cl_z80::tap_poll(void)
{
  int maxfd;
  fd_set fdset;
  struct timeval tv;

  tv.tv_sec = tv.tv_usec = 0;
  FD_ZERO(&fdset);
  FD_SET(tap_fd, &fdset);
  maxfd = tap_fd + 1;
  if (select(maxfd, &fdset, NULL, NULL, &tv) < 0) {
    perror("tap_poll: select");
    exit(1);
  }
  store2(_TAP_SIZE, FD_ISSET(tap_fd, &fdset));
}

void
cl_z80::tap_read(void)
{
  int size,i;
  t_addr dst_addr;

  printf("SIM: tap_read\n");
  size = read(tap_fd, tap_buf, (int) get2(_TAP_SIZE));
  store2(_TAP_SIZE,(short int) size);

  printf("SIM_DUMP: size=%d byte(s)\n",size);
  for (i=0;i<size;i++) {
    if (!(i%8) && (i%16))
      printf("   ");
    if (!(i%16))
      printf("\n%3d: ",i);
```

```
    printf("␣%02x",tap_buf[i]);
  }
  printf("\n");

  dst_addr = get2(_TAP_BUFPTR);
  for (i=0;i<size;i++) {
    store1(dst_addr+i,tap_buf[i]);
  }
}

static unsigned int tap_size = 0;

void
cl_z80::tap_write(void)
{
  int size;
  unsigned int src_addr,i;

  size = get2(_TAP_SIZE);
  src_addr = get2((t_addr)_TAP_BUFPTR);
  if (size>0)
    for(i=0;i<(unsigned)size;i++) {
      tap_buf[tap_size+i] = get1(src_addr+i);
    }
  tap_size += size;
  printf(
"SIM:␣tap_write␣(%d␣bytes␣from␣%x,␣tap_size=%d)␣%x␣%x␣%x␣%x␣...\n",
         size,src_addr,tap_size,
         tap_buf[0],tap_buf[1],
         tap_buf[2],tap_buf[3]);
}

void
cl_z80::tap_send(void)
{
  short int ret;

  printf("SIM:␣tap_send␣(%d)␣%x␣%x␣%x␣%x␣...\n",
         tap_size,
         tap_buf[0],tap_buf[1],
         tap_buf[2],tap_buf[3]);

  ret = (short int) write(tap_fd,tap_buf,tap_size);

  store2(_TAP_SIZE, ret);
  tap_size = 0;
}
```

Referências Bibliográficas

[Are02] Arena, 2002. `http://www.cdt.luth.se/projects/arena/`.

[ATM99] ATMEL Corporation. *AT90S8515 Datasheet*, 1999.

[Ayr00] James Ayres. *Application Note 181 – Using the Crystal CS8900A in 8-bit Mode*. Cirrus Logic Inc., `http://www-test.cirrus.com/en/pubs/appNote/an181.pdf`, Janeiro 2000.

[BBP88] R. Braden, D. Borman, and C. Partridge. RFC 1071: Computing the Internet checksum, Setembro 1988.

[Ben00] Jeremy Bentham. *TCP/IP Lean: Web Servers for Embedded Systems*. CMP Books, 2000.

[Bra89] R. Braden. RFC 1122: Requirements for Internet Hosts – Communication Layers, Outubro 1989.

[BW00] Gaetano Borriello and Roy Want. Embedded Computation Meets the World Wide Web. *Communications of the ACM*, 43(5):59–66, Maio 2000.

[Cai80] Ron Cain. Putting C on a microcomputer: The original Small-C. *Dr. Dobbs*, 45, Julho 1980.

[CCS02] CCS, Inc. CCS, Inc. - Highly optimized PICmicro C compiler, 2002. `http://www.ccsinfo.com/picc.shtml`.

[Cir01] Cirrus Logic, Inc. *CS8900A Product Data Sheet*, Abril 2001.

[CK74] Vinton G. Cerf and Robert E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, COM-22(5):637–648, Maio 1974.

[Cla82] David C. Clark. RFC 817: Modularity and efficiency in protocol implementation, Julho 1982.

[CMX02] CMX Systems. CMX MicroNet, 2002. `http://www.cmx.com/micronet.htm`.

[Com84] Douglas Comer. *Operating system design: the XINU approach.* Prentice-Hall, Inc., 1984.

[Com95a] Douglas E. Comer. *Internetworking with TCP/IP v1: Principles, Protocols, and Architeture*, volume 1. Prentice-Hall Inc., 3rd edition, 1995.

[Com95b] Douglas E. Comer. *Internetworking with TCP/IP v2: Principles, Protocols, and Architeture*, volume 1. Prentice-Hall Inc., 3rd edition, 1995.

[Cor02] OAR Corp. RTEMS, 2002. `http://www.oarcorp.com/RTEMS/rtems.html`.

[dCP99] Wilton de Castro Padrão. Um agente *Proxy* embutido para gerência de UPS. Tese de mestrado, DCC-UFMG, Belo Horizonte, Agosto 1999.

[DSP02] DSPOS, Inc. Fusion Net - Embedded Networking Protocols, 2002. `http://www.dspos.com/DSPOSWeb/products_net.htm`.

[Dun01a] Adam Dunkels. Design and implementation of the lwIP TCP/IP stack. Technical report, Swedish Institute of Computer Science, Fevereiro 2001.

[Dun01b] Adam Dunkels. A proxy based architeture for TCP/IP. Master's thesis, Swedish Institute of Computer Science - SICS, `http://www.sics.se/~adam/thesis.pdf`, Fevereiro 2001.

[Dun02a] Adam Dunkels. Full TCP/IP in 8 bits. Abril 2002.

[Dun02b] Adam Dunkels. uIP stack, 2002. `http://www.dunkels.com/adam/uip`.

[Dut02] Sandeep Dutta. SDCC, 2002. `http://sdcc.sourceforge.net/`.

[Eng02] Lightner Engineering. PicoWeb, 2002. `http://www.picoweb.net/`.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, Junho 1999.

[Gan98] Jack G. Ganssle. Internet connectivity: Stacking the odds in your favor. *Embedded Systems Programming*, 1998. `http://www.embedded.com/98/9801fe.htm`.

[Ips02] Ipsil, Incorporated. IPu 8930, 2002. `http://www.ipsil.com/products/d8930.htm`.

[iRe02a] iReady.org. iReady Technology Forum - Home, 2002. `http://www.iready.org/`.

[iRe02b] iReady.org. iReady Technology Forum - uWebserver, 2002. `http://www.iready.org/projects/uwebserver/`.

[JBB92] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP extensions for high performance, Maio 1992.

[JD02] Michael Durrant Jeff Dione. uCLinux – Embedded Linux/Microcontroller Project, 2002. `http://www.uclinux.org/`.

[Kar03] Phil Karn. The KA9Q NOS TCP/IP Package, 2003. `http://www.ka9q.net/code/ka9qnos/`.

[Kes97] S. Keshav. *An Engineering Approach to Computer Networking: ATM networks, the Internet, and the telephone network.* Addison-Wesley, 1997.

[Kra02] Maxim Krasnyansky. TUN/TAP driver, 2002. `http://vtun.sourceforge.net/tun/`.

[Lan02] Guy Lancaster. uc/IP, 2002. `http://ucip.sourceforge.net/`.

[Mic03] Sun Microsystems. Jini network technology, 2003. `http://wwws.sun.com/software/jini/`.

[Mit99] Larry Mittag. Innovations in Internet appliance design. *Embedded Systems Programming*, 1999. `http://www.embedded.com/1999/9905/9905mittag.htm`.

[MK90] T. Mallory and A. Kullberg. RFC 1141: Incremental Updating of the Internet Checksum, Janeiro 1990.

[Mor02a] Dominic Morris. Z88DK, 2002. `http://z88dk.sourceforge.net/`.

[Mor02b] Dominic Morris. ZSock, 2002. `http://www.rst38.org.uk/zsock/`.

[Pad02] Bob Paddock. TinyTCP, 2002. `http://www.unusualresearch.com/tinytcp/tinytcp.htm`.

[Pos81] Jon Postel. RFC 793: Transmission control protocol, Setembro 1981.

[QNX02] QNX Software Systems Ltd. QNX Neutrino RTOS Home, 2002. `http://www.qnx.com/products/ps_neutrino/`.

[Qui97] Richard A. Quinnel. Web servers in embedded systems enhance user interaction. *EDN Magazine*, April 1997.

[Rea01] Realtek Semi-conductor Co., Ltd. *RTL8019AS – Realtek Full-duplex Ethernet Controller with Plug-and-Play Function*, Maio 2001. Specification.

[Red03] RedHat. eCos, 2003. `http://sources.redhat.com/ecos/`.

[Sei02] Seiko Instruments USA, Inc. S-7600A Series, 2002. `http://www.seiko-usa-ecd.com/intcir/products/rtc_assp/s7600a.html`.

[Shr02] H. Shrikumar. IPic - a match head sized web-server, 2002. `http://www-ccs.cs.umass.edu/~shri/iPic.html`.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated: the protocols*, volume 1. Addison Wesley, 1994.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, 3rd edition, 1996.

[Tur99] Jim Turley. Embedded processors by the numbers. *Embedded Systems Programming*, 12(5), Maio 1999. `http://www.embedded.com/1999/9905/9905turley.htm`.

[Web98] Warren Webb. Ethernet invades embedded space. *EDN Magazine*, pages 71–80, Setembro 1998.

[Web99] Warren Webb. Embed the web for fun and profit. *EDN Magazine*, pages 58–68, Março 1999.

[Whi02] Fredric White. webACE Server, 2002. `http://d116.com/ace/`.

[WS95] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated: the implementation*, volume 2. Addison Wesley, 1995.

[Zil97] Zilog. *Z84C15 Product Brief*, 1997.

[Zil02] Zilog, Inc. Zilog, 2002. `http://www.zilog.com/`.