DANILO LOPES LUVIZOTO

ESTUDO DE IMPLEMENTAÇÕES DE PILHAS TCP/IP PARA MICROCONTROLADORES

Trabalho de Conclusão de Curso
apresentado à Escola de Engenharia de
São Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase
em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos
2010

Dedicatória

Dedico este trabalho aos meus pais que tanto lutaram para eu poder ser o que sou e pelo amor que sempre tiveram por mim. Também a minha querida namorada que me apoiou em todos os momentos de minha graduação. Além, é claro, de todos os amigos, que juntos batalhamos e vencemos mais uma vez.

Agradecimentos

Agradeço primeiramente a Deus por me iluminar nos caminhos mais difíceis, ao Professor Evandro pela orientação e incentivo deste trabalho e a todos outros professores e colegas que me ajudaram em tudo que sei hoje.

Resumo

Este trabalho apresenta um comparativo entre duas implementações de pilhas TCP/IP para sistemas embarcados baseados em microcontrolador. O trabalho foi desenvolvido utilizando um microcontrolador MSP430 da Texas Instruments, embarcado em uma placa de desenvolvimento da Olimex. O trabalho consistiu em duas tarefas: a primeira foi o estudo de cada pilha e a segunda foi uma comparação e desenvolver uma aplicação. O objetivo foi enfatizar algumas das principais diferenças entre as pilhas sugerindo algumas aplicações.

Palavras chave: Sistemas Embarcados, Protocolo TCP/IP, Comunicação Internet, MSP430

Abstract

This paper shows a comparison between two TCP / IP stacks for embedded systems implemented based on microcontrollers. The study was conducted using a MSP430 microcontroller from Texas Instruments, included in the development kit from Olimex. The work consisted of two tasks: the first one was the study of each stack and second was a comparison and develop an application. The objective was to emphasize some differences between the stacks and suggest some applications.

Keywords: Embedded systems, TCP/IP protocol, Internet communication, MSP430

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

IP - Internet Protocol

RISC - Reduced Instruction Set Computer

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

SMTP - Simple Mail Transfer Protocol

ICMP - Internet Control Message Protocol

HTTP - HyperText Transfer Protocol

ARPA - Advanced Research Projects Agency

ACK - Acknowledgement

SACK - Selective ACK

MTU - Maximum Transfer Unit

FDDI - Fiber Distributed Data Interface

CRC - Cyclical Redundancy Check

QFP - Quad Flat Pack

LAN - Local Area Network

IETF - Internet Engineering Task Force

NAS - Network Attachment Storage

FTP - File Transfer Protocol

1. Introdução

Sistema embarcado é um sistema microprocessado no qual o ***hardware*** é dedicado ao sistema que ele controla, diferente de computadores que executam várias tarefas ao mesmo tempo com propósitos diferentes. Um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas específicas, pode-se aperfeiçoar o projeto reduzindo tamanho, recursos computacionais e custo do produto de acordo com a aplicação. Diante diss,o estão cada vez mais presentes em nosso cotidiano, controlando, monitorando e comunicando e entre diversas outras operações encontramos sistemas embarcados. O surgimento de novas aplicações é impulsionado pelo crescimento do mercado e um melhor conhecimento das necessidades.

E nesse contexto têm cada vez mais necessidades de comunicação, devido a controle, monitoração e segurança. Com isso a internet é uma alternativa viável e muitas vezes de custo reduzido. É neste contexto que este trabalho se encaixa, fazendo a integração da aplicação com o equipamento, onde cada equipamento pode ser monitorado com sensores. E com a utilização da internet pode-se controlá-lo e monitorá-lo remotamente, tendo como único pré-requisito uma conexão ativa com a internet.

As duas pilhas TCP/IP que iremos estudar são duas implementações reduzidas da pilha que é implementada para computadores. A maior restrição de ambas implementações é o hardware com menor capacidade de processamento e menor tamanho de memória RAM. Mas, mesmo diante das limitações, as duas pilhas têm que ser totalmente compatíveis com as pilhas implementadas para computadores, pois deverão se comunicar sem perdas.

1.1 Motivação

A motivação deste trabalho partiu das limitações encontradas na pilha TCP/IP de Andreas Dannenberg diante de várias aplicações. Assim, sabendo da existência da pilha de Adam Dunkels, aqui referida somente como uIP, teve-se a oportunidade de verificar as diferenças e conseqüentemente uma melhora nas possíveis aplicações com o kit de desenvolvimento Olimex EasyWeb2.

1.2 Objetivos

O objetivo deste trabalho é comparar de forma qualitativa duas pilhas de protocolos TCP/IP implementadas para sistemas embarcados com microcontroladores MSP430 da ***Texas Instruments***, e por fim exemplificar com uma aplicação que demonstre as diferenças entre as duas implementações.

1.3 Organização do texto

Este trabalho está dividido em seis capítulos, sendo que o primeiro capítulo é somente uma apresentação do tema proposto e os objetivos a serem alcançados.

O segundo capítulo faz uma introdução a todos os conceitos necessários para o entendimento, tanto do trabalho. Introduzindo conceitos de redes e protocolos, além contexto em que se enquadram.

No terceiro capítulo, é o foco do trabalho, onde são feitas as explicações das duas pilhas, além dos matérias e métodos utilizados para a comparação e logo após um comparativo de todas as diferenças que são importantes.

No quarto capítulo foi mostrado todos os resultados obtidos através do estudo das duas pilhas.

O quinto capítulo conclui o trabalho, analisando os resultados e testes realizados e também propões novas implementações e soluções.

E no último capítulo têm-se as referências bibliográficas que foram utilizadas para a composição deste trabalho.

2. Introdução Teórica

Cada um dos três séculos anteriores foi dominado por uma única tecnologia. O século XVIII foi a época dos grandes sistemas mecânicos que acompanharam a Revolução Industrial. O século XIX foi a era das máquinas a vapor e no século XX as principais conquistas tecnológicas deram-se no campo da aquisição, do processamento e da distribuição de informações [1].

Neste último século, com a invenção do transistor teve-se a revolução eletrônica, o que possibilitou várias inovações. Com isso, os processadores puderam evoluir tanto que quase todos os controles eletrônicos atualmente são microprocessados. Essa evolução foi acompanhada, nos últimos anos, pelo crescimento dos sistemas de comunicação e em especial pelas redes de computadores.

2.1 Redes de computadores

Antes do advento de computadores dotados com algum tipo de sistema de telecomunicação, a comunicação entre máquinas calculadoras e computadores antigos era realizada por usuários através do carregamento de instruções entre eles.

Em 1940, George Stibitz utilizou uma máquina de teletipo[1] para enviar instruções com um conjunto de problemas a partir de sua calculadora Model K (Figura 1) na Faculdade de Dartmouth em Nova Hampshire para a sua calculadora em Nova Iorque e recebeu os resultados de volta pelo mesmo meio.

[1] **Equipamento eletromecânico de transmissão de dados**

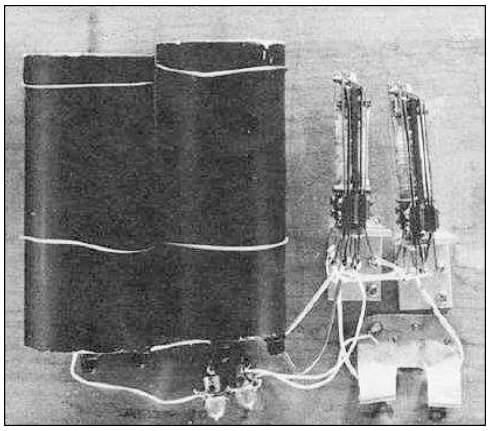


Figura 1 - Calculadora (Model K) [1]

Conectar sistemas de saída, como teletipos a computadores, era um interesse na ARPA (*Advanced Research Projects Agency*) quando, em 1962, J. C. R. Licklider foi contratado e desenvolveu um grupo de trabalho o qual ele chamou de a "Rede Intergaláctica", um precursor da ARPANET.

Durante a década de 1960, Leonard Kleinrock, Paul Baran e Donald Davies, de maneira independente, conceituaram e desenvolveram sistemas de redes, os quais usavam datagramas ou pacotes, que podiam ser usados em uma rede de comutação de pacotes entre sistemas de computadores.

Em 1969, a Universidade da Califórnia em Los Angeles, SRI (em Stanford), a Universidade da Califórnia em Santa Bárbara e a Universidade de Utah foram conectadas com o início da rede ARPANET usando circuitos com capacidade de transmissão de 50 kbits/s. Essa rede denominada de ARPANET era uma rede que interligava bases militares e departamentos de pesquisa do governo americano. Ela utilizava um ***backbone*** que passava por baixo da terra, sem ter um centro ou caminho definido. Essa técnica conferia ao sistema um segurança, pois não tinha um elemento principal, que se fosse destruído impactava no sistema.

Redes de computadores e as tecnologias necessárias para conexão e comunicação, através e entre elas, continuam a comandar as indústrias de ***hardware*** de computador, ***software*** e periféricos. Essa expansão é espelhada pelo crescimento nos números e tipos de usuários de redes, desde o pesquisador até o usuário doméstico.

Atualmente, redes de computadores são o núcleo da comunicação moderna. O escopo da comunicação cresceu significativamente na década de 1990 e essa explosão nas comunicações não teria sido possível sem o avanço progressivo das redes de computadores.

As redes em si não são somente os meios físicos de transmissão, mas um conjunto de ***software*** e ***hardware*** que comanda e controla as conexões. Por isso as regras das comunicações devem ser claras para todos os elementos envolvidos em uma conexão, com isso todos os sistemas de comunicação têm protocolos, que gerenciam toda a comunicação, desde a inicialização até o término da conexão, fazendo o gerenciamento de perdas, tempos e limites, tentando sempre maximizar a transferência e o uso do ***hardware*** e minimizar os problemas.

2.2 Protocolos e hierarquias

Para reduzir a complexidade do projeto, a maioria das redes é organizada como uma pilha de níveis ou camadas, colocadas umas sobre as outras. O número de camadas, o nome, o conteúdo e a função de cada camada diferem de uma rede para outra. No entanto, em todas as redes o objetivo de cada camada é oferecer determinados serviços às camadas superiores, isolando essas camadas dos detalhes de implementação desses recursos. De certa forma, cada camada é uma espécie de máquina virtual, oferecendo determinados serviços à camada situada acima dela.

Na realidade, esse conceito é familiar e é utilizado em toda a ciência da computação, na qual é conhecido por nomes diferentes, como ocultação de informações, tipos de dados abstratos, encapsulamento de dados e programação orientada a objetos. A idéia fundamental é que um determinado item de software (ou hardware) fornece um serviço a seus usuários, mas mantém ocultos os detalhes de seu estado interno e de seus algoritmos.

A camada n de uma máquina se comunica com a camada n de outra máquina. Coletivamente, as regras e convenções usadas nesse diálogo são conhecidas como o protocolo da camada n. Basicamente, um protocolo é um acordo entre as partes que se comunicam, estabelecendo como se dará a comunicação. A violação do protocolo dificultará a comunicação, se não a tornar completamente impossível. A Figura 2 ilustra uma rede de cinco camadas. As entidades que ocupam as camadas correspondentes em diferentes máquinas são chamadas pares (***peers***). Os pares

podem ser processos, dispositivos de hardware ou mesmo seres humanos. Em outras palavras, são os pares que se comunicam utilizando o protocolo [1].

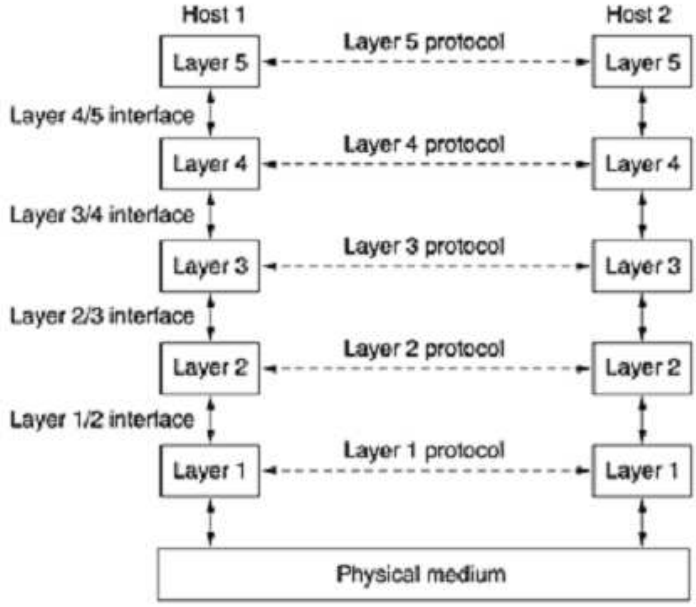


Figura 2 - Camadas, Níveis e Protocolos [1]

2.3 Protocolo TCP/IP

Com o sucesso da Internet, o modelo de protocolos TCP/IP se tornou o padrão internacional de comunicação. É o protocolo utilizado para transferência de ***web pages***, transmissões de ***email***, transferência de arquivos através da Internet. Sistemas embarcados que possuem o protocolo implementado serão elementos importantes, que podem se comunicar tanto via Internet quanto dentro de uma Intranet, que é uma versão privada da Internet.

O TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede. Seu nome vem de dois protocolos: o TCP - Protocolo de Controle de Transmissão e o IP - Protocolo de Interconexão. O protocolo é estruturado em um modelo de camadas, onde cada camada é responsável por algumas tarefas, fornecendo um conjunto de serviços definidos para o protocolo da camada superior. As camadas mais altas estão logicamente mais próximas do usuário. O modelo é estruturado de acordo com a Figura 3.

Figura 3 - Camadas Protocolo TCP/IP

Este modelo de camadas é uma idéia que surgiu com o modelo **OSI** (***Open Systems Interconnection***), que é um modelo onde se tem sete camadas estruturadas de forma semelhante ao que se tem no modelo TCP\IP. Este modelo é o precursor de quase todos os modelos de comunicação que se tem hoje. As cinco camadas do modelo TCP/IP são relacionadas com as camadas do modelo **OSI** de acordo com a Figura 4.

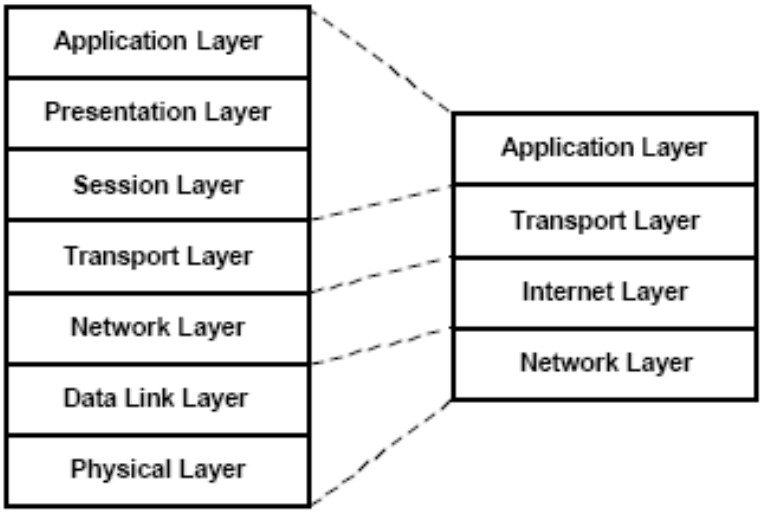
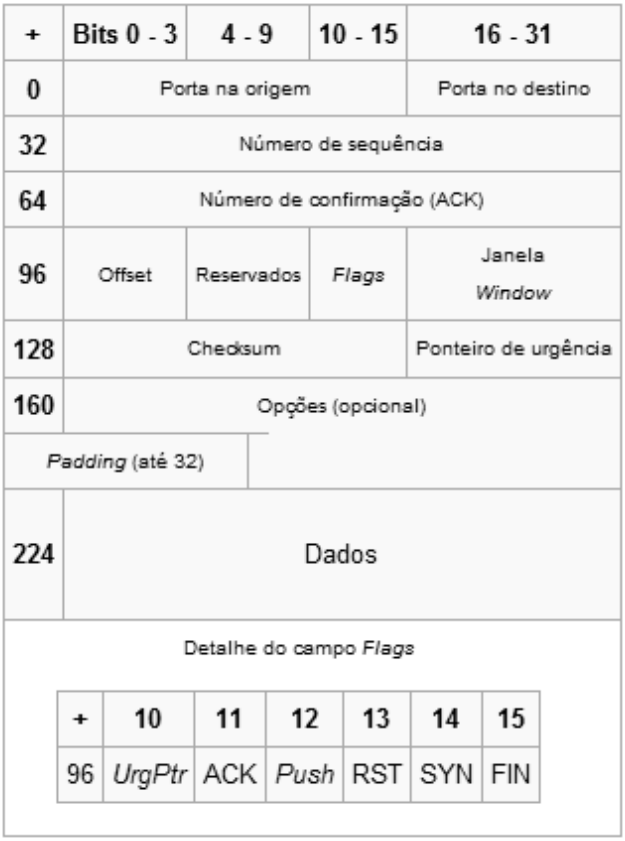


Figura 4 - Modelo OSI - TCP/IP

Os protocolos TCP\IP possuem cinco camadas, mas as pilhas implementadas estão focadas nas camadas de transporte e de rede. Todas as outras camadas abaixo serão associadas como “dispositivos de redes” e a camada acima simplesmente como camada de aplicação.

O protocolo TCP especifica três fases durante a conexão: primeiro o estabelecimento da conexão, depois a transmissão e por último o término da conexão.

O cabeçalho do protocolo TCP pode ser visto na Figura 5. Ele é composto basicamente da porta de origem e de destino, além do número de sequência que identifica a posição deste pacote diante de um fluxo de pacotes, por um número que confirma o recebimento do pacote e por alguns bits de controle, além do tamanho do cabeçalho. Possui também um campo de verificação de erros do cabeçalho e um campo em que a origem determina para indicar onde está algum dado urgente dentro do pacote. E, por último, temos os dados que são basicamente o pacote do protocolo que é utilizado pela aplicação.



+	Bits 0 - 3	4 - 9	10 - 15	16 - 31
0	Porta na origem			Porta no destino
32	Número de sequência			
64	Número de confirmação (ACK)			
96	Offset	Reservados	*Flags*	Janela *Window*
128	Checksum			Ponteiro de urgência
160	Opções (opcional)			
	Padding (até 32)			
224	Dados			

Detalhe do campo *Flags*

+	10	11	12	13	14	15
96	*UrgPtr*	ACK	*Push*	RST	SYN	FIN

Figura 5 - Cabeçalho TCP [2]

O estabelecimento da ligação ocorre de forma ordenada, onde o cliente envia um pacote em que ele pede uma conexão com o servidor. Assim que o servidor recebe o pacote, ele devolve um pacote de confirmação (ACK - ***Acknowledgement***) juntamente com um pacote informando a aceitação da conexão, mas, caso o servidor não receba o pacote, o cliente espera um tempo (***Timeout***) e reenvia o pacote. E, para finalizar o processo de estabelecimento de conexão, o servidor devolve um pacote de confirmação (ACK) conforme Figura 6.

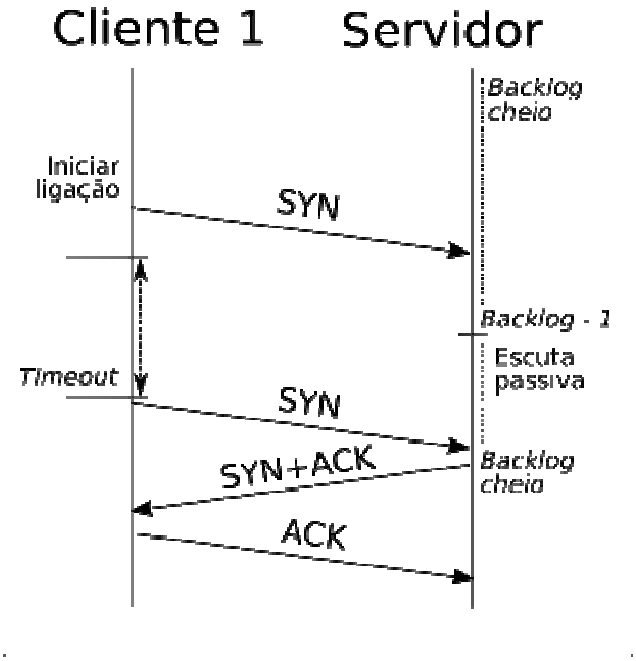


Figura 6 - Estabelecimento de conexão TCP

Depois de estabelecida a conexão começa o processo de transferência de dados, conforme Figura 7, em que o servidor envia os dados conforme solicitado pelo cliente. Esse envio ocorre de forma ordenada, e assim que o cliente recebe os dados ele devolve um pacote de confirmação contendo o número do pacote recebido para se manter uma alta confiabilidade. Uma grande vantagem do TCP, é que ele pode confirmar pacotes que chegam fora da ordem previamente estabelecida através de um pacote de confirmação (SACK - ***Selective*** *ACK*). O cabeçalho do protocolo TCP contém um campo de verificação de erros (***Checksum***), o que permite assegurar a integridade do pacote recebido.

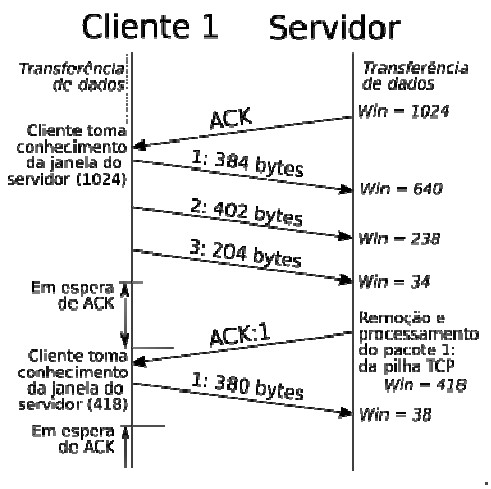


Figura 7 - Transmissão de dados TCP

Para finalizar a aplicação, qualquer uma das partes manda um pacote de finalização, e a outra parte devolve um pacote de confirmação e logo depois manda também um pacote de finalização conforme Figura 8. E por fim a primeira parte devolve um pacote de confirmação.

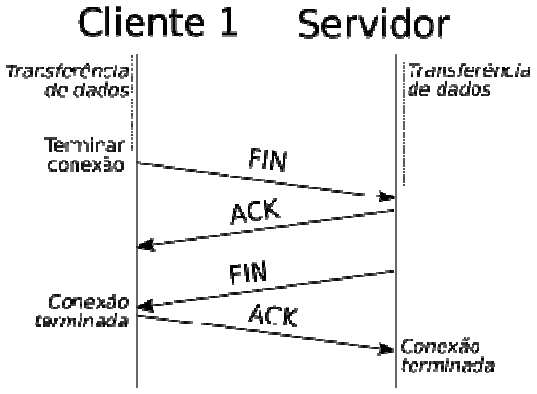


Figura 8 - Finalização da conexão TCP

O protocolo TCP introduz um conceito de porta, que está associado a um serviço da camada de aplicação. Assim cada conexão utiliza uma porta específica para o tipo de aplicação. O protocolo TCP usa o pacote IP para entrega dos

datagramas à rede. Já o pacote IP trata os pacotes TCP como dados e não interpreta qualquer conteúdo das mensagens TCP.

Representando o cabeçalho de um pacote IP temos a Figura 9. Neste cabeçalho temos basicamente, o tamanho do cabeçalho, como o tamanho do pacote inteiro. Temos também o tempo de vida antes que o pacote não seja mais retransmitido, possui também os endereços de origem e destino, além de um verificador de erros do cabeçalho e alguns bits para controle. E ocupando a maior parte do pacote temos a área de dados, que é onde será transportado o pacote TCP que vai ser transmitido.

<table>
<tr><th>+</th><th>0 - 3</th><th>4 - 7</th><th>8 - 15</th><th>16 - 18</th><th>19 - 31</th></tr>
<tr><td>0</td><td>Versão</td><td>Tamanho do cabeçalho</td><td>Tipo de Serviço (ToS) (agora DiffServ e ECN)</td><td colspan="2">Comprimento (pacote)</td></tr>
<tr><td>32</td><td colspan="3">Identificador</td><td>Flags</td><td>Offset</td></tr>
<tr><td>64</td><td colspan="2">Tempo de Vida (TTL)</td><td>Protocolo</td><td colspan="2">Checksum</td></tr>
<tr><td>96</td><td colspan="5">Endereço origem</td></tr>
<tr><td>128</td><td colspan="5">Endereço destino</td></tr>
<tr><td>160</td><td colspan="5">Opções</td></tr>
<tr><td>192</td><td colspan="5">Dados</td></tr>
</table>

Figura 9 - Cabeçalho IP [2]

O protocolo IP é usado para encaminhamento de dados. O IP oferece um serviço de datagramas não confiável, ou seja, o pacote chega sem garantias, mas isso é resolvido pelo protocolo da camada de transporte, como explicado no protocolo TCP, que faz toda a verificação de erros (***Checksum***). Vamos supor que tenham muitas redes interconectadas por ***gateways***, mas só tenham dois hosts em cada extremo da interconexão das redes, quando um ***host*** quer enviar ao outro, ele encapsula o datagrama e o envia ao ***gateway*** mais próximo. Uma vez que o quadro chega ao ***gateway***, o ***software*** de IP extrai o datagrama encapsulado, e as rotinas de

roteamento IP selecionam o próximo ***gateway*** que formará parte do caminho que levará o datagrama ao ***host*** destino, conforme representado na Figura 10.

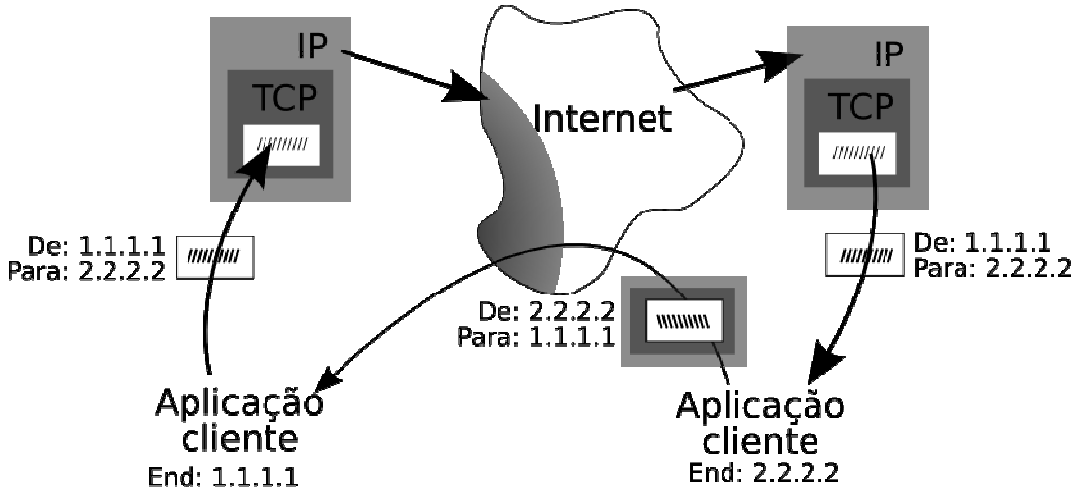


Figura 10 - Encapsulamento TCP/IP

Um problema que as pilhas TCP/IP têm que solucionar, é a fragmentação de pacotes. Por passarem por várias redes distintas, com tecnologias diferentes, muitas vezes as redes não têm capacidade de enviar pacotes IP no tamanho original que eles foram recebidos. Com isso tem-se uma fragmentação de pacotes, que normalmente é feito em ***switches*** que interligam duas redes diferentes conforme Figura 11. O ***switch*** vai dividir os pacotes e reencapsulá-los, de maneira que o receptor final consiga remontá-los. Na Tabela 1 temos os tamanhos de pacotes em diferentes tipos de redes: uma rede ***Ethernet*** padrão, uma rede ARPANET, e uma rede FDDI (***Fiber Distributed Data Interface****)* que é uma tecnologia de transmissão de dados que utiliza o protocolo de transmissão ***Token Ring*** e possui capacidade de transmissão muito elevadas, da ordem de Gigabits\s. O ***Token Ring*** funciona de forma circular, e quem quer transmitir tem que possuir o ***Token***.

Tabela 1 - Tamanho de pacotes

Tipo de rede	MTU (em bytes)
Ethernet	1500
ARPANET	1000
FDDI	4470

Figura 11 - Fragmentação de pacotes IP

3. Materiais e métodos

3.1 Kit EasyWeb 2

Para a realização deste trabalho foi utilizado o kit de desenvolvimento ***Olimex EasyWeb2*** mostrado na Figura 12. A utilização foi devido à disponibilidade do mesmo nos laboratórios da Engenharia Elétrica (LAB-SEL) da Escola de Engenharia de São Carlos (EESC) da Universidade de São Paulo (USP).

O kit de desenvolvimento possui um microcontrolador da ***Texas Instruments***, o MSP430, juntamente com uma memória EEPROM 24LC515 de 64 Kbytes, que se comunica com o microcontrolador pelo padrão I²C. O padrão I²C é um tipo de comunicação serial que se opera com dois canais, sendo um de dados e outro de ***clock*** para sincronismo. O kit é composto dos seguintes componentes:

- MSP430F149 rodando pilha TCP/IP de Andreas Dannenberg[1]
- Controlador LAN CS8900 + Transformador + Conector RJ45
- Três ***Leds*** de ***status*** da LAN
- Dois relês 10A/240VAC
- Quatro entradas foto acopladas
- Quatro chaves ***Push Button***
- Comunicação Serial RS232
- ***Buzzer***
- Display LCD 16x2
- Memória EEPROM 64k - 24LC515
- Cristal oscilador de 8Mhz
- Conector JTAG

[1] Código-livre a todos desde que citem a referência.

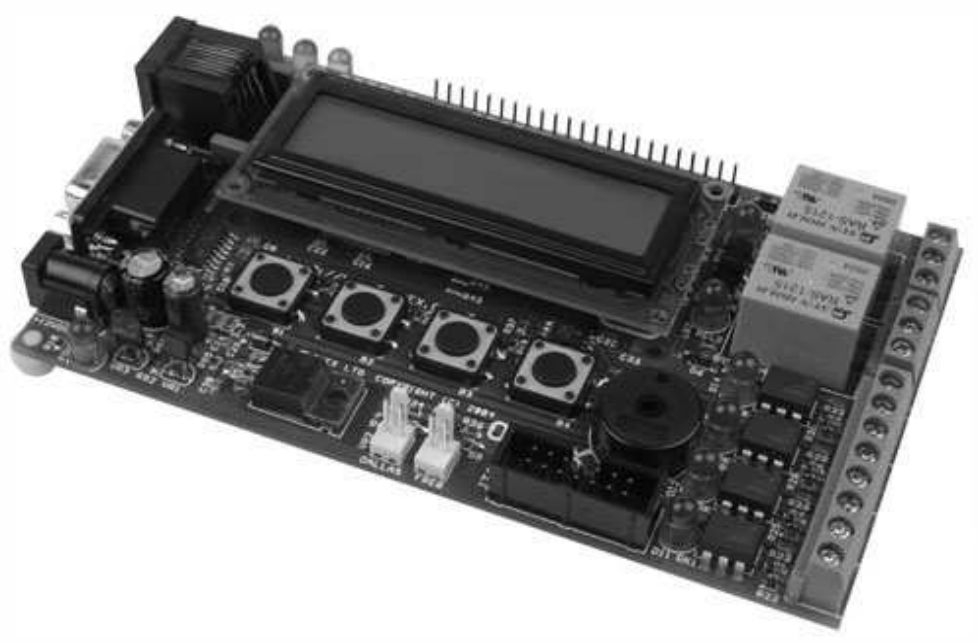


Figura 12 - Kit Olimex EasyWeb 2 [3]

3.2 MSP 430

O microcontrolador utilizado no kit de desenvolvimento é o MSP430F149 fabricado pela ***Texas Instruments***, baseados em arquitetura RISC de 16 bits. É voltado para aplicações de baixo consumo, para aplicações embarcadas e portáteis. Possuí memória de flash 60 Kbytes e 2 Kbytes de memória RAM.

Por ser de baixo consumo apresenta baixa tensão de operação, variando de 1,8 V à 3,6 V. Trabalha tipicamente com uma corrente de 280 µA à 1 MHz e 2,2V de Vcc. É bastante econômico em modo ***stand-by***, consumindo aproximadamente 1,6 µA, podendo operar em modo de retenção de memória RAM, com apenas 0,1 µA de corrente. Para reduzir consumo de energia, o MSP430 apresenta cinco modos de consumo de energia. Possuí um tempo de ***wake-up*** de 6 µs.

O MSP430 não é tolerante a entradas em nível TTL, ou seja, 5 V. Logo não é possível que dispositivos TTL transmitam informações diretamente ao microcontrolador, mas o contrario é possível, pois os padrões TTL aceitam o nível de tensão que o MSP430 suporta, ou seja, 3,6 V.

Possui conversor A/D de 12 bits com referência interna, ***timers*** de 16 bits. Possui um sistema flexível de ***clocks***, que pode ser mudado durante a execução de programas. Tem geradores de ***clock*** internos de 12 KHz, 32,768 KHz e de 16 MHz, além do ***clock*** externo.

Para programação tem uma lista contendo 51 instruções com três formatos e com sete modos de endereço. Os formatos podem ser de três tipos: ***Dual-operand***: dois operandos - fonte e destino; ***Single-operand***: apenas um operando, que pode ser uma fonte ou um destino; ***Jump***: instruções de salto no programa. Para programação tem uma lista contendo 51 instruções com três formatos e com sete modos de endereço. Os formatos podem ser de três tipos: ***Dual-operand*** que são dois operandos - fonte e destino; ***Single-operand*** que tem apenas um operando, que pode ser uma fonte ou um destino; e ***Jump*** em que possui instruções de salto no programa. Os modos de endereçamento são os seguintes: *Modo imediato; Modo registrador; Modo indexado; Modo simbólico; Modo absoluto; Modo indireto; e Modo indireto com auto incremento.*

Possui comparador interno no próprio chip, não necessitando de uso de comparadores externos. Possui cinqüenta I/Os (entrada/saída), e sessenta e quatro pinos em um chip formato QFP conforme Figura 13.

A programação pode ser realizada *on-board*, por comunicação serial, e possui um módulo JTAG para emulação. O microprocessador apresenta dois módulos USART, que podem ser operados como UART assíncrona, ou como interface SPI síncrona.

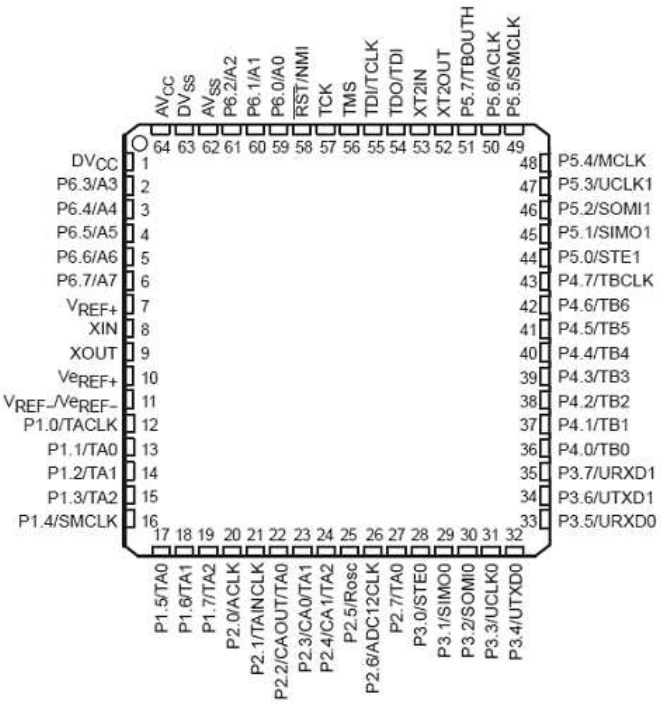


Figura 13 - Esquema Pinos MSP430F149 [4]

Temos na Figura 14 o diagrama que mostra a arquitetura interna do ***chip*** em bloco, mostrando de forma sintetizada a forma de comunicação interna.

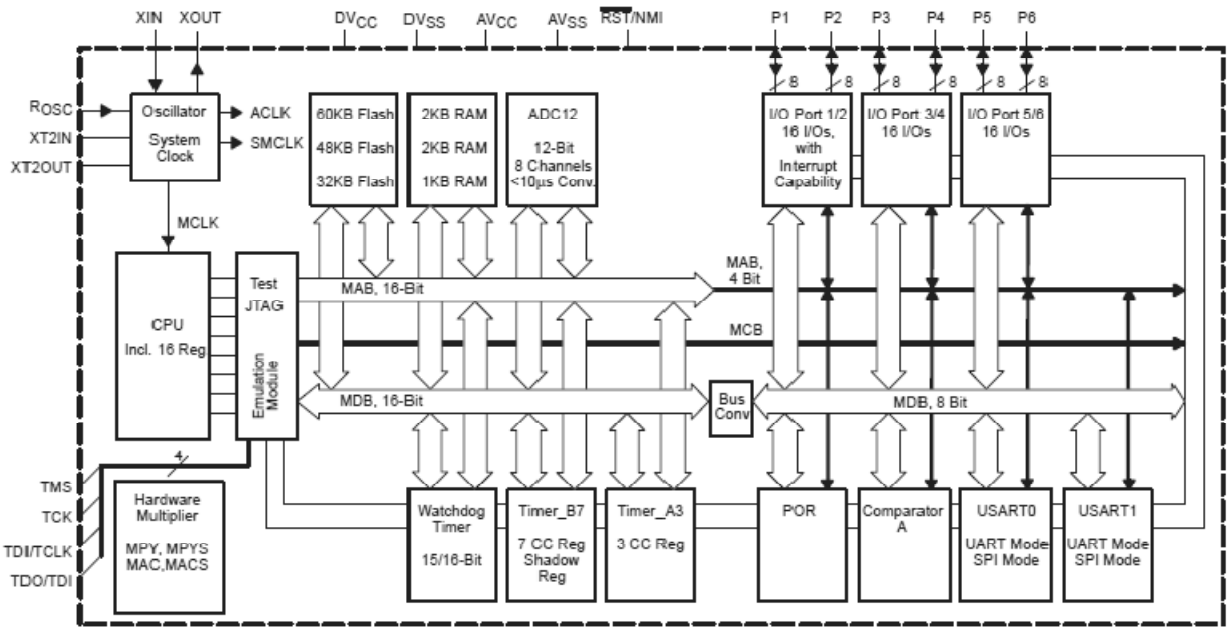


Figura 14 - Arquitetura interna do MSP430 [4]

Já na Tabela 2 tem-se as características elétricas do MSP430, como tensão de operação, consumo em determinadas freqüências, corrente de entrada e saídas suportadas, e também de condições de operação.

PARAMETER		TEST CONDITIONS		MIN	NOM	MAX	UNIT
$I_{(AM)}$	Active mode, (see Note 1) $f_{(MCLK)} = f_{(SMCLK)} = 1$ MHz, $f_{(ACLK)} = 32,768$ Hz XTS=0, SELM=(0,1)	$T_A = -40°C$ to $85°C$	$V_{CC} = 2.2$ V		280	350	µA
			$V_{CC} = 3$ V		420	560	
$I_{(AM)}$	Active mode, (see Note 1) $f_{(MCLK)} = f_{(SMCLK)} = 4\,096$ Hz, $f_{(ACLK)} = 4,096$ Hz XTS=0, SELM=(0,1) XTS=0, SELM=3	$T_A = -40°C$ to $85°C$	$V_{CC} = 2.2$ V		2.5	7	µA
			$V_{CC} = 3$ V		9	20	
$I_{(LPM0)}$	Low-power mode, (LPM0) (see Note 1)	$T_A = -40°C$ to $85°C$	$V_{CC} = 2.2$ V		32	45	µA
			$V_{CC} = 3$ V		55	70	
$I_{(LPM2)}$	Low-power mode, (LPM2), f(MCLK) = f (SMCLK) = 0 MHz, f(ACLK) = 32.768 Hz, SCG0 = 0	$T_A = -40°C$ to $85°C$	$V_{CC} = 2.2$ V		11	14	µA
			$V_{CC} = 3$ V		17	22	
$I_{(LPM3)}$	Low-power mode, (LPM3) $f_{(MCLK)} = f_{(SMCLK)} = 0$ MHz, $f_{(ACLK)} = 32,768$ Hz, SCG0 = 1 (see Note 2)	$T_A = -40°C$	$V_{CC} = 2.2$ V		0.8	1.5	µA
		$T_A = 25°C$			0.9	1.5	
		$T_A = 85°C$			1.6	2.8	
		$T_A = -40°C$	$V_{CC} = 3$ V		1.8	2.2	µA
		$T_A = 25°C$			1.6	1.9	
		$T_A = 85°C$			2.3	3.9	
$I_{(LPM4)}$	Low-power mode, (LPM4) $f_{(MCLK)} = 0$ MHz, $f_{(SMCLK)} = 0$ MHz, $f_{(ACLK)} = 0$ Hz, SCG0 = 1	$T_A = -40°C$	$V_{CC} = 2.2$ V		0.1	0.5	µA
		$T_A = 25°C$			0.1	0.5	
		$T_A = 85°C$			0.8	2.5	
		$T_A = -40°C$	$V_{CC} = 3$ V		0.1	0.5	µA
		$T_A = 25°C$			0.1	0.5	
		$T_A = 85°C$			0.8	2.5	

Tabela 2 - Características Elétricas do MSP430 [4]

3.3 CS8900

CS8900 é um controlador LAN ***Ethernet*** de baixo custo, para aplicações embarcadas. Não necessita de outros controles adicionais, sendo tudo incorporado no próprio ***chip***. O CS8900 tem incluso RAM, transmissor e receptor (10Base-T) que é uma implementação da ***Ethernet*** que suporta transmissões à taxa de 10 Mbits\s e interface via barramento ISA para comunicação com o microcontrolador, que é um barramento paralelo de 8 bits. O esquema do diagrama interno do controlador pode ser observado na Figura 15.

O controlador além da alta integração e pequeno tamanho oferece ampla gama de recursos de desempenho e opções de configuração. Sua arquitetura adapta-se aos padrões de tráfego da rede e dos recursos disponíveis. O resultado é uma maior eficiência do sistema como um todo [5].

- IEEE 802.3 Ethernet
- Modo de operação ***Full-duplex***
- Portas e Filtros 10Base-T (polarização e detecção/correção)
- Retransmissão automática quando ocorre colisão
- Rejeição automática de pacotes com erros
- Suporte à EEPROM com ***jumper***
- LED de ***link status*** e atividade da LAN
- ***Stand-by*** e modos de baixo consumo de energia
- Opera com 3 V ou 5 V
- Consumo máximo em 5 V = 120 mA; e típico em 5 V = 90 mA
- 100 pinos TQFP

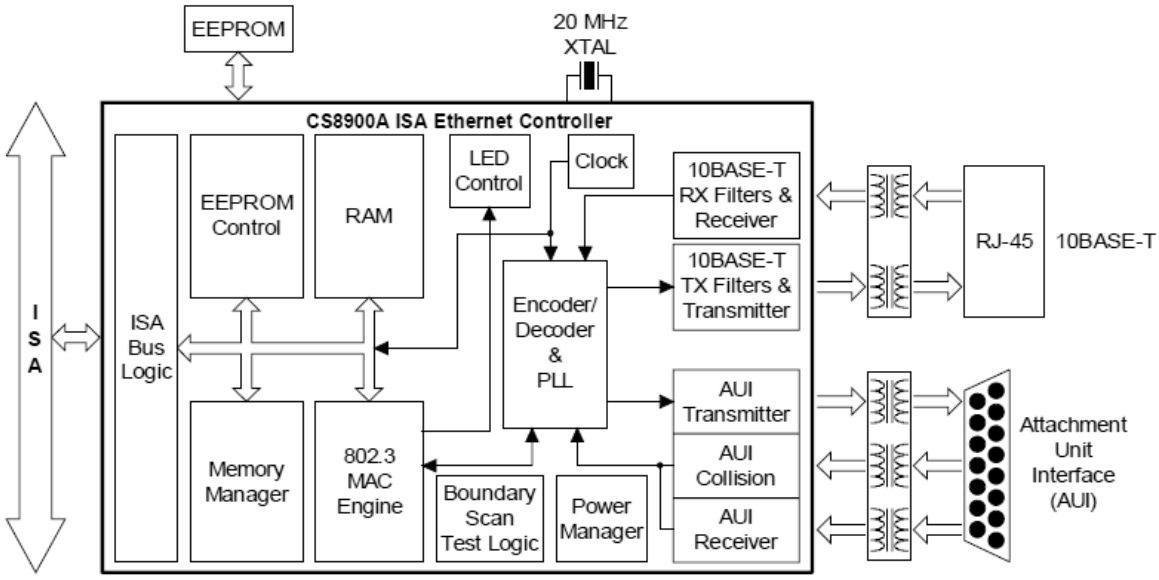


Figura 15 - Diagrama do controlador CS8900 [6]

3.4 Comparação entre as pilhas TCP/IP

O foco principal deste trabalho é a comparação das duas pilhas de protocolos TCP/IP que serão discutidas a seguir. É importante para compararmos as pilhas, entendermos cada uma individualmente. Assim, neste trabalho é realizada uma descrição de cada uma das pilhas citando cada uma de suas principais funções, e a partir desta descrição teremos o comparativo.

As implementações das duas pilhas analisadas neste trabalho utilizam pilhas simplificadas, pois a pilha de protocolo padrão de TCP/IP requer uma grande quantidade de memória, tanto de memória para o código quanto de memória RAM para abrigar suas variáveis. Como não há grande disponibilidade de memória as duas pilhas não abrangem toda complexidade do protocolo. Sendo assim as duas pilhas são desenvolvidas tendo o mínimo necessário para o protocolo funcionar, mas não deixando de ser compatível com todos os sistemas existentes.

A pilha completa TCP/IP consiste em inúmeros protocolos, que vão desde protocolo ARP que traduz endereço IP para endereços MAC, protocolos de aplicação, como SMTP (***Simple Mail Transfer Protocol***) que é utilizado para transferir texto de email. As duas pilhas são voltadas para os protocolos TCP e IP.

Basicamente o protocolo TCP fornece um fluxo confiável de bytes para os protocolos da camada superior. Ele divide o fluxo em segmentos de tamanhos adequados e cada segmento é enviado em seu próprio pacote IP. Os pacotes IP, são por sua vez enviados pelo controlado de rede. Caso o destino não esteja conectado fisicamente ao emissor, este envia o pacote a um roteador, que envia ao destino. Caso o tamanho máximo de pacotes aceito pela outra rede seja menor do que o tamanho do pacote enviado, o roteador fragmenta o pacote em pacotes com tamanho menor, com isso é necessário que o receptor consiga remontar o pacote.

As pilhas são implementadas respeitando os requisitos da RFC1122. As RFCs, são ***Request For Coments***, que são documentos que descrevem os padrões de cada protocolo de comunicação da Internet. Qualquer pessoa pode gerar uma RFC, que vai ser aprovada, ou não pelo IETF (***Internet Engineering Task Force***), e posteriormente publicada após revisão como uma RFC.

3.5 Pilha de Andreas Dannenberg

O código desenvolvido por Andreas Dannenberg segue uma estrutura mais enxuta, mas não deixando de ser eficiente. Tem-se suporte somente a aplicação de um servidor ***web***, suportando somente IPv4 e não possui suporte ao IPv6. O IPv6, surgiu diante das limitações de endereço que o IPv4 veio a apresentar. Como a Internet quando foi criada não tinha fins comerciais, o endereço com quatro octetos (IPv4) foi suficiente até os dias atuais, mas está com previsão de esgotamento para meados de 2011, então está sendo implantado o endereço com oito octetos (IPv6), em paralelo ao IPv4. Diante disso, é uma falta que poderá ter um impacto maior quando todos os endereços de internet forem migrados para o IPv6.

A pilha é desenvolvida com base na RFC793, e com isso é totalmente compatível com qualquer sistema de rede com suporte a TCP/IP, algumas funcionalidades que não são essenciais ao funcionamento da pilha TCP/IP não foram implementadas, pois é totalmente voltado pra microcontroladores de 8 bits, com baixo consumo de RAM e de memória de programa, apesar de alguns módulos não serem implementados, não é complexo o estudo para posterior implementações.

Basicamente o programa começa inicializando o microcontrolador, suas portas, além de ***clocks*** e as comunicações externas, como a serial. Depois se inicializa o controlador de rede. Logo após toda parte de inicialização concluída, entra em funcionamento a pilha.

Em funcionamento a pilha fica esperando algum chamado de conexão vindo de um cliente. Quando acontece algum evento, é feita uma verificação para saber se é um chamado para conexão. Caso seja, a pilha entra em modo de estabelecimento de conexão. Logo após a conexão estabelecida começa o envio dos pacotes contendo uma página ***web*** (essa página ***web*** foi previamente colocada na memória do microcontrolador no momento da programação), por fim finalizando a conexão.

Algumas funções e parâmetros são muito importantes para o entendimento da pilha, pois além do entendimento do protocolo TCP/IP, é importante saber como é tratado o dado pela pilha. A seguir são descritas as principais funções que compõem a pilha elaborada por Andreas Dannenberg.

void TCPLowLevelInit(void)

Essa função faz a inicialização do controlador de rede e de várias variáveis, além de configurar as portas e o Timer_A do MSP430. Ela deve ser chamada antes de transmitir qualquer dado por TCP/IP.

void Init8900(void)

Configuras as portas do CS8900 - controlador de rede - que se comunica com o microcontrolador, e faz toda configuração do controlador.

void TCPPassiveOpen(void)

Essa função coloca o controlador no modo servidor, esperando por uma conexão. O ***flag*** SOCK_ACTIVE muda para o estado ocupado. Antes de chamar a função deve-se configurar a porta através da variável global TCPLocalPort. O IP do servidor é configurado nas quatro constantes, MYIP_1 a MYIP_4, no arquivo tcpip.h.

void TCPActiveOpen(void)

Com essa função é possível se conectar a um servidor remoto. O ***flag*** SOCK_ACTIVE muda para o estado ocupado. Antes de se utilizar a função, o IP do servidor remoto e as portas local e remota devem ser configuradas, respectivamente, através das variáveis RemoteIP, TCPLocalPort e TCPRemotePort.

void TCPClose(void)

Essa função fecha uma conexão aberta. Se algum pacote estiver no ***buffer*** de saída, ele será enviado antes da conexão ser fechada. Uma nova conexão pode ser aberta após o uso desta função.

void TCPReleaseRxBuffer(void)

Essa função apaga o ***buffer*** de entrada. Deve ser utilizada após a leitura dos dados para que novos dados possam ser recebidos. Ela limpa o ***flag*** SOCK_TX_BUF_RELEASED, que indica a presença de dados novos. Para se realizar a leitura do ***buffer*** de entrada, deve-se ler a área de memória apontada pelo ponteiro TCP_RX_BUF. O número de bytes recebidos é indicado pela variável TCPRxDataCount.

void TCPTranmitTxBuffer(void)

Essa função envia dados sobre uma conexão previamente aberta. Antes de se utilizar a função, a aplicação do usuário deve checar se o ***buffer*** de saída está vazio, através do ***flag*** SOCK_TX_BUF_RELEASED do registro Socketstatus. Se estiver, deve-se escrever os dados a serem enviados na área de memória apontada por TCP_TX_BUF e o número de bytes dos dados na variável TCPTxDataCount. O número máximo de bytes que podem ser transmitidos está indicado na constante MAX_TCP_TX_DATA_SIZE.

void DoNetworkStuff(void)

Função mais importante, que faz toda a comunicação, e segue o esquema apresentado pelo fluxograma da Figura 16. Essa função deve ser chamada periodicamente pela aplicação, pois acessa várias ***flags*** no controlador de rede e no microcontrolador. Logo, quanto mais essa função for chamada freqüentemente, melhor será o desempenho da pilha TCP/IP.

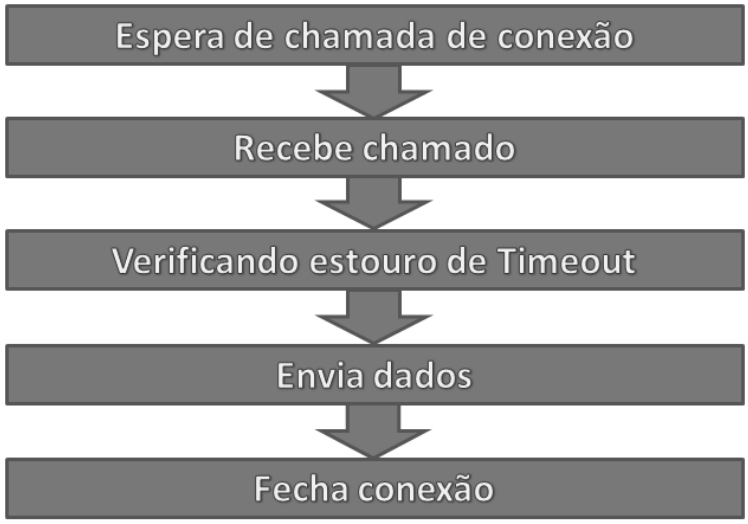


Figura 16 - Fluxograma da função DoNetworkStuff()

Esta pilha possui várias limitações. Uma delas e muito importante, é o não suporte a múltiplas conexões. Outras limitações são, a não verificação de erros, a falta de um suporte a remontagem de pacotes, que são fragmentados. Basicamente é uma

pilha para redes locais, onde a quantidade de erros é bem baixa e que a fragmentação de pacotes não acontece.

3.6 Pilha uIP (Adam Dunkels)

A pilha uIP foi elaborada por Adam Dunkels com foco em microcontroladores de 8 bits, e por isso é adaptável a praticamente qualquer microcontrolador encontrado no mercado. A pilha vem recebendo atualização constante, então sempre é possível verificar novas aplicações, e ou melhorias na pilha.

A implantação da pilha para o kit de desenvolvimento ***EasyWeb2*** foi realizada por Paul Curtis, que utilizou o software ***CrossStudio***. Curtis também adaptou a pilha uIP para outro kit com ARM. Atualmente existem diversas implementações dessa pilha para os mais diversos microcontroladores existentes atualmente.

Na arquitetura do uIP, o recurso mais escasso é a RAM. Com apenas algumas centenas de ***Bytes*** alguns mecanismos tradicionais não são aplicados. No uIP não se usa alocação dinâmica de memória e sim um único buffer global quem recebe e transmite os pacotes. O tamanho deste buffer é maximizado para suportar o máximo tamanho de pacotes. O ***buffer*** é utilizado pelo protocolo, tanto para receber quanto para enviar pacote IP ao controlador de rede que no caso é o CS8900. Outra informação importante é quanto a estrutura do código, que é distribuído em módulos fazendo que a implementação para diversos tipos de ***hardware*** seja feita de forma fácil e ágil, uma vez que pode ser parametrizado para funcionar com ***hardwares*** com pouco mais de 200 bytes de memória RAM.

O funcionamento da pilha inicializa-se com a configuração das portas e ***timers*** do microcontrolador, depois passando a configurar o controlador de rede. Quando um pacote chega, o controlador de rede coloca o pacote no ***buffer*** e aciona a pilha de protocolo. Caso o pacote tenha dados o microcontrolador irá notificar a aplicação correspondente. Como o ***buffer*** é único, logo que um pacote é recebido, ele tem que ser processado, ou então armazenado em um segundo buffer. Os pacotes não serão sobrescritos pelo próximo pacote que chegar. Caso chegue um pacote enquanto a aplicação está processando, este entrará em fila de espera. O controlador CS8900 tem um ***buffer*** que pode armazenar até quatro pacotes, criando uma fila de espera, para posteriormente enviá-lo ao microcontrolador.

Algumas funções importantes para o funcionamento da pilha serão descritos a seguir.

void uIP_appcall(void)

Esta é a função principal, que sempre que um dado é recebido, essa função é chamada. Ela define qual aplicação está sendo solicitada e automaticamente solicita a função da aplicação que trate os dados recebidos. A função identifica qual a aplicação, lendo o cabeçalho do frame e posteriormente chama a aplicação correspondente.

void uIP_newdata(void)

Esta função é que avisa a aplicação quando um novo dado é recebido, e sinaliza-se na variável global **uip_conn** que tem um dado no ***buffer***.

void uIP_send(void)

Esta função é chamada para a transmissão de dados que estão no ***buffer***.

void uIP_rexmit(void)

Esta é a função que é responsável pela retransmissão de pacotes, que ocorre quando não é recebido o ***ACK***, ou quando acontecer um erro com o pacote enviado. Apesar de que o controlador de rede do kit ***EasyWeb2*** ter suporte a isso é importante ter isso implementado quando não temos um controlador que tenha esse suporte.

void uIP_close(void)

É a função que fecha um conexão, depois que a aplicação foi executada com sucesso, e não há mais o que ser transmitido ou recebido. Já a função **void uIP_abort(void)**, tem a função de fechar a conexão imediatamente caso ocorra um erro fatal. Isso pode ocorrer de duas formas; primeiro quando houver estouro de ***timeout***, ou seja, quando já houve o número de tentativas de retransmissão que foi estipulado na variável de controle, e segundo se a conexão foi fechada pelo cliente sem o término da transmissão dos dados.

void uIP_listen(void)

É a função que faz a manutenção das portas TCP abertas. Caso uma porta esteja fechada, não terá como se comunicar com a pilha de protocolos TCP/IP.

void uIP_connect(void)

É a função que abre as portas de conexões, configurando a variável global **uIP_conn** com o status da conexão, além da porta conectada. Isso é muito importante para o controle de todas as conexões ativas.

void uIP_tcpchksum(void)

Esta é a função que faz a verificação de erros no cabeçalho da pilha TCP, caso haja algum erro ela automaticamente chama uma função para que se tenha a retransmissão do pacote com erro.

void uip_ipchksum(void)

Esta função faz a verificação de erros no cabeçalho da pilha IP, e caso haja erro ela automaticamente chama uma função para que se tenha a retransmissão do pacote com erro.

uip_conn

É a variável que controla toda conexão, controlando quais portas estão abertas, além de todo as conter todas as **flags** de controle. É basicamente onde está armazenado todas as informações referentes as conexões.

O esquema de funcionamento básico da pilha uIP pode ser observado na Figura 17. É importante entender que as funções básicas de transmissão, retransmissão, abertura de portas e as demais que fazem o controle e o estabelecimento de conexões estão sempre implementadas para que se possa colocar em funcionamento a pilha de protocolos TCP/IP, mas é importante observar que outras funções de suporte, como as de verificação de erros das pilhas TCP e IP nem sempre estão implementadas, uma vez que a necessidade é definida pela conveniência do programador e limitadas por seu ***hardware***.

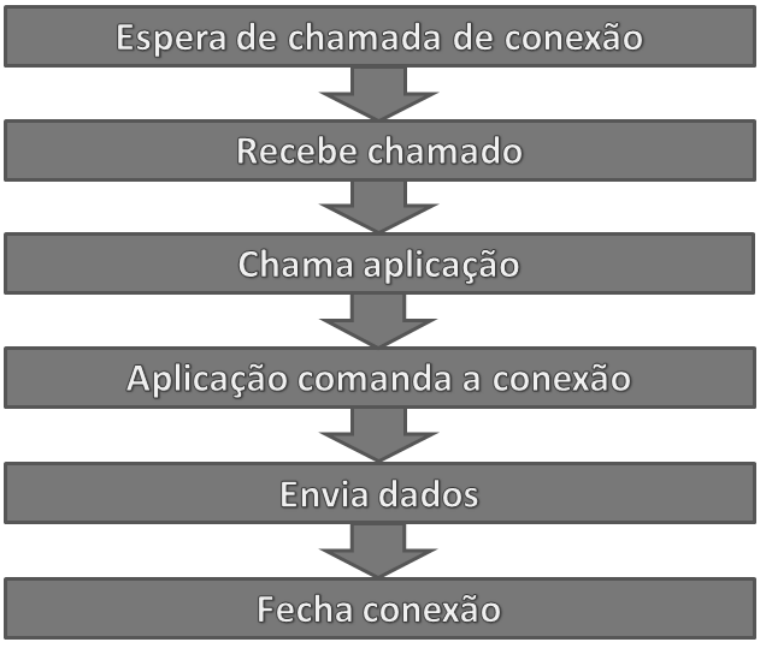


Figura 17 - Fluxograma básico do protocolo uIP

4. Resultado e discussão

Muitas diferenças de código podem ser observadas, mas o foco do trabalho é uma análise qualitativa, especificando os principais pontos de diferenças entre as duas pilhas. E por fim uma análise sobre os impactos dessas diferenças para outras aplicações.

A primeira diferença é referente à estrutura do código, em que a pilha de Andreas Dannenberg tem um código unificado, possuindo poucas funções, e as funções não possuem muitas parametrizações como se poderia de ter, dando ao código maior maleabilidade. Já a pilha uIP possui o código distribuído em módulos, aos quais podem ser implementados de acordo com a necessidade, e muitas parametrizações que contribuem para que a pilha possa ser implementada para ***hardware*** com pouquíssima quantidade de RAM (200 bytes).

A pilha uIP tem implementa o suporte ao IPv6, o que faz ela poder ser utilizada com essa nova versão do protocolo IP. Este protocolo já está sendo implantado e por enquanto está funcionando lado a lado com o Ipv4. O principal motivo para a implantação do IPv6 na Internet é a necessidade de mais endereços, porque os endereços livres IPv4 estão se esgotando.

Problemas com conexões múltiplas foram implementadas pelo uIP. O protocolo de Dannenberg tem suporte somente a uma única conexão com o servidor ***web*** por vez. Isso é bastante restrito, uma vez que hoje várias pessoas ao redor do mundo acessam a mesma página ao mesmo tempo. Considere uma indústria como exemplo, que o servidor ***web*** é a base de dados para várias máquinas funcionarem, com isso elas precisam ficar verificando o site sempre. Caso não houvesse suporte a múltiplas conexões as máquinas não conseguiriam acessar o site, e com isso não teriam acesso à base de dados, gerando uma pane que talvez pudesse resultar na parada da produção.

Um ponto de bastante atenção é a verificação de erros (***checksum***) no uIP. A pilha de Dannenberg não realiza verificação de erros. Isso para uma rede bem estruturada pode não ser relevante, mas quando se pensa que várias pessoas ao redor do mundo acessa o ***website*** tem-se de levar em conta os erros durante a transmissão e recepção dos pacotes IP. Já o uIP faz essa verificação com um código bem simples.

Apesar de não ser necessariamente uma diferença e sim uma nova implementação, o uIP tem suporte a UDP, o que pode ser favorável quando se pensa em transmissão de dados sem verificação, mas com maior agilidade e velocidade. Também tem suporte a HTTP, *Telnet* ***Server*** e ***Client***, SMTP, além de suporte a um sistema de arquivos.

A pilha de Dannenberg possui um ponto crítico que é a falta de um sistema de remontagem de pacotes. Isso é de extrema importância, pois pacotes que passam por redes diferentes ao longo do caminho podem passar por alguns pontos onde se fragmenta o pacote para se adequar a rede daquele local. Isso faz com que o servidor tenha que remontá-lo para poder entender o pacote e com isso retransmitir o que foi solicitado.

A documentação de ambos os projetos são bem elaboradas. O projeto de Andreas, não se tem documentação, mas possui um código bem comentado, mas nada além disso. Já o código uIP, tem-se uma documentação extensa, com isso torna-se fácil, o entendimento da pilha. Isso contribui para que qualquer pessoa que esteja interessada em utilizar a pilha uIP possa fazê-lo sem problemas de entendimento.

A pilha uIP por ter sido desenvolvida sem especificação de microcontroladores pode ser implementa para vários microcontroladores diferentes. Apresenta uma documentação especifica para implementação em qualquer microcontrolador. Outra diferença, está na arquitetura do código, em que o protocolo uIP é mais distribuído, onde cada aplicação tem várias funções, já no caso da pilha de Dannenberg, a pilha tem poucas funções.

As duas pilhas são de código livre, o que é citado por ambos os desenvolvedores. Com isso qualquer pessoa pode utilizar os códigos para implementarem seus projetos. As principais diferenças foram compiladas e mostradas na Tabela 3 para uma melhor visualização.

Tabela 3 - Comparação entre as duas pilhas

	Andreas	µIP
Multiplas Conexões	✗	✓
Verificação de CheckSum	✗	✓
Suporte UDP	✗	✓
Remontagem de Pacotes	✗	✓
Suporte a Sistemas de Arquivos	✗	✓
Suporte HTTP	✓	✓
Código Livre	✓	✓

Tamanho de código é algo que se deve levar bastante em consideração, devido às limitações dos microcontroladores. Apesar de serem pilhas de tamanho reduzido, são bastante consideráveis se comparadas com o tamanho total de memória de programa disponível. Além disso, um código da pilha TCP/IP muito grande, limitaria a aplicação ao qual o projeto será destinado. Os tamanhos de códigos das duas pilhas podem ser observados na Tabela 4 para efeito comparativo.

Tabela 4 - Tamanho de código das pilhas

Pilhas	Tamanho Em Bytes	Usado da memória do MSP430
uIP	11 096	18,05%
Pilha de Andreas	8 906	13,14%

Pode se ver que a pilha uIP ocupa um espaço de código maior, e isso é totalmente explicado pelas muitas funcionalidades a mais, que já foram discutidas, que ele possui. Mas mesmo assim o código é bem compacto quando comparado com pilhas TCP/IP para computadores do tipo PC.

Agora analisando o uso de memória RAM das duas pilhas, temos um cenário bem parecido, devido as duas possuir um único ***buffer*** local, com isso deve ser armazenado todo o pacote TCP/IP que está sendo recebido.

Um teste realizado foi o de colocar duas páginas com tamanhos diferentes, tentando forçar diferentes tamanhos de pacotes TCP/IP transportados. Por exemplo usando uma página com apenas 528 ***bytes*** e outra com 2567 ***bytes*** de tamanho efetivo, temos os resultados expressos abaixo na Tabela 5. Essas medidas foram feitas utilizando a função de ***DEBUG*** que os compiladores possuem. No caso o software principal que chama a função foi desprezado para não comprometer as medidas.

Tabela 5 - Uso de memória RAM

Uso de memória RAM (em bytes)		
Tamanhos de páginas	**Andreas**	**uIP**
WebPage I (528 *bytes*)	442	439
WebPage II (2567 *bytes*)	1328	1327

Um teste que visa mostrar a conectividade com o dispositivo, foi o comando *ping*. Este comando utiliza o protocolo ICMP para testar a conectividade entre equipamentos. Seu funcionamento consiste no envio de pacotes para o equipamento e de destino e na escuta das respostas. Utilizando o comando *ping* para testar as conectividades entre o computador e o dispositivo depois de implementas ambas as pilhas obteve-se os resultados das Figura 18 para a pilha de Andreas Dannenberg e a Figura 19 para a pilha uIP.

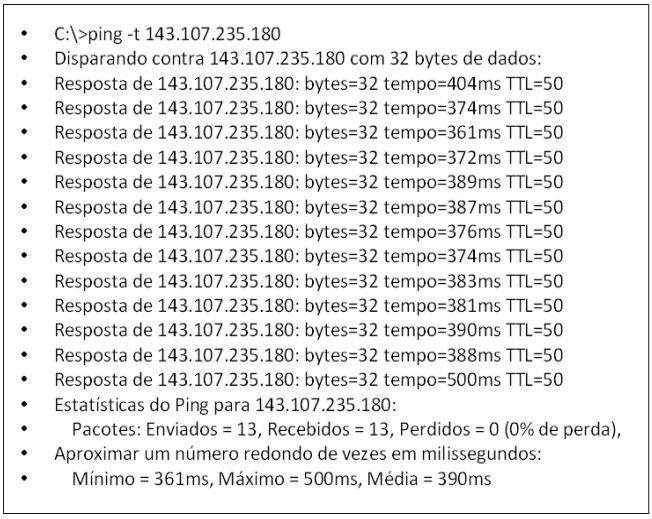


Figura 18 - Resposta da pilha de Andreas Dannenberg ao comando *ping*

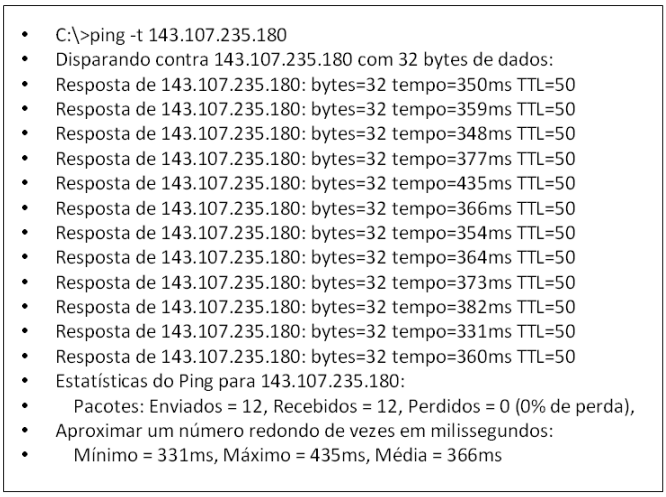


Figura 19 - Resposta da pilha uIP ao comando *ping*

Para verificar múltiplas conexões, foi utilizada uma técnica simples, que consiste em colocar uma página com um tempo de atualização na faixa de um à dois segundos, para ficar atualizando constantemente. Este tempo foi estimado utilizando um ***software*** (*Firebug v.1.4.5*), instalado juntamente com o navegador de internet Mozilla Firefox, mede o tempo de abertura de uma página, desde a solicitação até o carregamento completo. Efetuando o teste algumas vezes a média de tempo de abertura foi de 2,03 s. Logo se utilizou tempo de atualização de dois segundos.

Abrindo a página em um computador, a mesma ficou atualizando a cada dois segundos. Ao mesmo tempo em outro computador foi solicitada a página e verificada se a mesma foi recebida. Além de tentar abrir a página, foi utilizando o comando *ping* para verificar a conectividade entre o computador e o dispositivo. Para a pilha de Andreas Dannenberg a página solicitada no segundo computador não obteve resposta, e o comando *ping* teve como resposta a Figura 20. Já a pilha uIP obteve resposta ao comando *ping* conforme Figura 21.

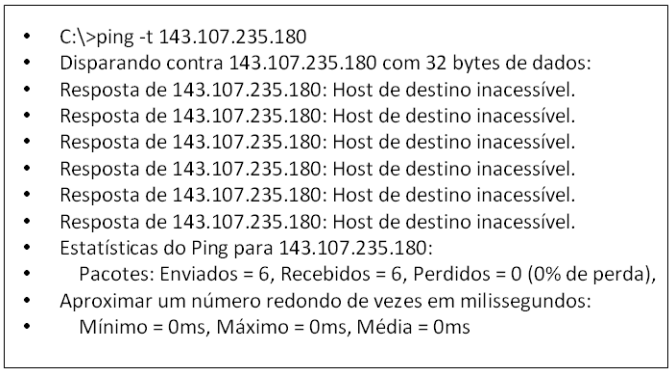


Figura 20 – Teste múltiplas conexões da pilha de Andreas Dannenberg

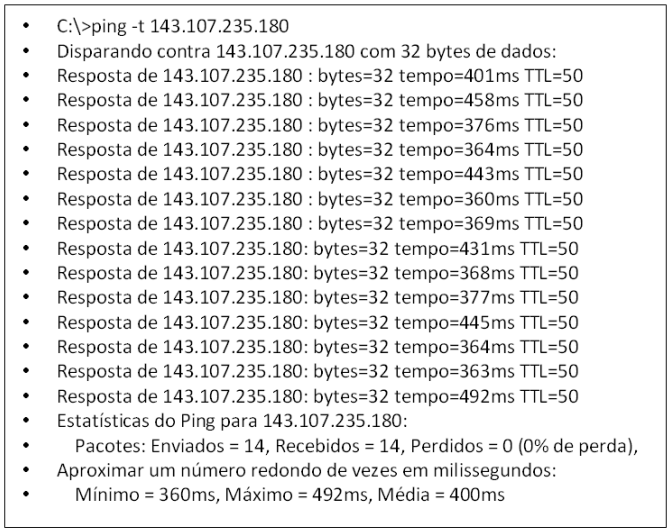


Figura 21 – Teste múltiplas conexões da pilha uIP

Uma aplicação bastante interessante e que foi estruturada para podermos exemplificar foi a implementação de um cliente SMTP utilizando a pilha uIP. O SMTP é um protocolo padrão para envio de emails através da internet. É relativamente simples de ser implementado, e tem como referências a RFC282, na qual toda a descrição é detalhada.

O protocolo é baseado em comandos enviados pelo cliente e respostas dadas pelo servidor. Sempre se espera a resposta de um comando antes de mandar um novo comando. Os comandos e resposta são sempre terminados pelo caractere "***Carriage Return***" (CR). Todas as respostas do servidor possuem duas partes: o código de retorno e a mensagem "***Human Readable***" (legível por humanos) sendo que o código é usado para identificar o tipo de mensagem por programas. Códigos começados com 2 indicam sucesso, 3 indicam sucesso, mas deve-se enviar mais dados para finalizar a operação, e 4 ou 5 são erros.

5. Conclusão

O objetivo principal deste trabalho foi a comparação das duas pilhas de protocolos TCP/IP.

Para tanto foi importante o estudo individual de ambas as pilhas de protocolos. Com isso conclui-se que diante de inúmeros fatos citados no capitulo três, a pilha uIP é mais robusta, uma vez também que é uma pilha melhorada em relação à primeira. Outros fatores são as mais diversas aplicações que são suportadas em seu código nativo.

Não se pode dizer sobre melhor ou pior. Pois depende necessariamente da aplicação foco de qualquer projeto, para se definir a melhor pilha. Uma vez que a aplicação seja somente um servidor ***web*** local, a pilha de Andreas Dannenberg suporta com tranqüilidade, pois não necessita de remontagem de pacote e a quantidade de erros é bem pequena, mas se for um servidor que será acessado de qualquer parte do mundo, prefere-se a pilha uIP, por conter remontagem de pacotes e verificação de erros.

Utilizando as pilhas, tanto de Andreas quanto a uIP pode se implementar várias aplicações desde um simples servidor ***web*** até projetos de armazenamento em massa. Desde que precise se conectar com a internet, qualquer aplicação é possível, mas sempre considerando algumas limitações.

Hoje em dia a mobilidade tornou-se algo do nosso dia-a-dia, mas as vezes estamos distraídos e nos perdemos diante de tanta mobilidade. Este é o caso de documentos importantes, que as vezes trabalhamos em vários lugares e nem sempre na mesma máquina. Com isso uma aplicação interessante a ser desenvolvida, é um servidor FTP, ou mesmo um NAS (*Network-attached storage*), onde se conecta diretamente na rede uma unidade de armazenamento (HD). Com um microcontrolador isso seria possível utilizando memórias flash. Hoje existem no mercado controladores para memórias flash que se adaptam a qualquer microcontrolador possibilitando leitura e escrita de dados. Para isso utiliza-se uma aplicação já disponível na pilha uIP para implantação de um servidor FTP.

Referencia Bibliográfica

1. **Tanenbaum, Andrew S.** *Computer Networks.* s.l. : Prentice Hall, 2002.

2. **IETF.** IETF. *IETF Web Site.* [Online] [Citado em: 02 de 11 de 2009.] http://tools.ietf.org.

3. **Olimex.** Olimex. *Olimex Web Site.* [Online] [Citado em: 02 de 11 de 2009.] http://www.olimex.com/dev/msp-easyweb2.html.

4. **Instruments, Texas.** Texas Instruments. *Texas Instruments Web Site.* [Online] [Citado em: 02 de 11 de 2009.] http://focus.ti.com/lit/ds/symlink/msp430f149.pdf.

5. **Logic's, Cirrus.** Cirrus Logic's. *Cirrus Logic's Web Site.* [Online] [Citado em: 02 de 11 de 2009.] http://cirrus.com/en/products/pro/detail/P46.html.

6. **Logic's, Cirrus.** Cirrus Logic's. *Cirrus Logic's Web Site.* [Online] http://cirrus.com/en/pubs/proDatasheet/CS8900A_F4.pdf.

7. **IANA.** IANA. *IANA Web Site.* [Online] [Citado em: 02 de 11 de 2009.] http://www.iana.org/protocols/.