Revista da Graduação

Vol. 4	No. 2	2011	14

Seção: FACULDADE DE ENGENHARIA

Título: Múltiplos serviços de internet com o uIP: desenvolvimento de uma aplicação embarcada para redes TCP/IP

Autor: Marco Túlio Gonçalves Martins

Este trabalho está publicado na Revista da Graduação.
ISSN 1983-1374
http://revistaseletronicas.pucrs.br/ojs/index.php/graduacao/article/view/10075/7110

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA

ENGENHARIA DE COMPUTAÇÃO

MARCO TÚLIO GONÇALVES MARTINS

MÚLTIPLOS SERVIÇOS DE INTERNET COM O uIP:

DESENVOLVIMENTO DE UMA APLICAÇÃO EMBARCADA PARA REDES TCP/IP

ORIENTADOR: MARCOS AUGUSTO STEMMER

PORTO ALEGRE

2011

MARCO TÚLIO GONÇALVES MARTINS

MÚLTIPLOS SERVIÇOS DE INTERNET COM O uIP:
DESENVOLVIMENTO DE UMA APLICAÇÃO EMBARCADA PARA REDES TCP/IP

Trabalho de conclusão de curso de graduação apresentado nas Faculdades de Engenharia e Informática da Pontifícia Universidade Católica do Rio Grande do Sul, como requisito parcial para obtenção do grau de Engenheiro de Computação.

Orientador: Marco Augusto Stemmer

Porto Alegre

2011

Dedico este trabalho a toda minha família.

Aos meus pais, pelo apoio, compreensão, paciência nos períodos complicados do fim de cada semestre e pelo incentivo para o meu crescimento profissional.

Ao meu Deus, acima de tudo, por dar-me conhecimento e sabedoria. Pelas orações de todos que me acompanharam até aqui.

Aos meus amigos e colegas de graduação pelo companheirismo, disponibilidade e amizade.

Aos meus colegas de serviço pelo apoio e pelas trocas de conhecimento.

Ao professor Mestre Marcos Augusto Stemmer pela sua orientação, dedicação e oportunidades de conhecimento. Ao mestrando Felipe Lavratti pelo apoio na elaboração deste trabalho.

“Eis que Deus é a minha salvação; confiarei e não temerei, porque o Senhor Deus é a minha força e o meu cântico; ele se tornou a minha salvação.” (Isaias 12:2)

RESUMO

O atual avanço das tecnologias de comunicação e dos sistemas embarcados resultou no aumento da interação entre estes sistemas usando protocolos de comunicação. O modelo TCP/IP usado atualmente, é pesado demais para ser usado em sistemas embarcados pequenos devido as altas taxas de processamento dos protocolos e uso de memória. Como alternativa a este modelo, foi desenvolvida a pilha uIP (*micro Internet Protocol*) para sistemas embarcados com limitações de memória e poder de processamento.

Contudo, o modelo uIP, possui algumas limitações, como a disponibilidade de apenas um serviço na sua camada de aplicação, a não reportagem de erros de comunicação, a limitação de *buffers,* entre outros. O principal objetivo deste trabalho é superar uma dessas limitações, desenvolvendo um sistema onde múltiplas aplicações possam interagir em um mesmo ambiente dedicado.

A limitação de apenas uma aplicação no uIP pôde ser superada através da criação de uma sub-camada dentro da camada de transporte da sua pilha. Esta sub-camada tem a função de gerenciar os pacotes recebidos e encaminhá-los para a aplicação correta. A porta de destino que o protocolo TCP tem no seu cabeçalho é utilizada para decidir o destino dos pacotes vindos da rede.

Duas novas aplicações foram desenvolvidas para a camada de aplicação da pilha uIP-TCP/IP. Uma delas é um servidor para troca de arquivos usando o protocolo SFT (*Secure File Transfer)* e a outra é um *chat* de comunicação usando *sockets* TCP. Ambas as aplicações suportam interações com sistemas que utilizem o modelo TCP/IP padrão. As principais funções do uIP, as APIs (*Application Programming Interfaces*) para geração de *threads* e *sockets* no uIP, bem como as novas aplicações e suas funcionalidades foram documentadas ao longo do projeto.

Palavras chaves: sistemas embarcados, redes TCP/IP, pilha uIP-TCP/IP, múltiplas aplicações.

ABSTRACT

Current advance in communication technology and the field of embedded systems have increased the interaction of such systems using communication protocols. Although the TCP/IP is standard today, it is usually too heavy to be used in small embedded systems due to the overheads associated to processing and memory usage. As an alternative the Micro Internet Protocol (uIP) stack was designed to be portable to small embedded system with limited memory and processing power.

However, the uIP model has limitations, such as the availability of only one service in its application layer, the absence of communication error reports, the buffers limitation, among others. The main goal of this work is to overcome some of these limitations, developing a system where multiple applications are able to interact in the same dedicated environment.

The limitation of accepting only one application in uIP has been overcome by the creation of a sub-layer within the stack´s transport layer. This sub-layer manages the incoming packets and routes them to the correct application. The destination port included in the TCP´s header is used to indicate the packet´s destination in the network.

Two new applications for the application layer of the uIP-TCP/IP stack have been developed. One of them is a server for file exchange using Secure File Transfer (SFT) protocol and the other is *chat* communication using TCP *sockets*. Both applications support interactions with systems that use the TCP/IP standard. The uIP´s main functions, the Application Programming Interfaces (APIs) for thread and *socket* creation in the uIP, as well as the new applications created and their functionalities have been documented.

Keywords: embedded systems, TCP/IP, networks, uIP-TCP/IP stack, multiple applications.

LISTA DE ILUSTRAÇÕES

Figura 1 – Diagrama de blocos do LPC2378 da NXP [3] 15
Figura 2 – Kit de Desenvolvimentos PUCRS[4] 16
Figura 3 – Modelo OSI vs Modelo TCP/IP 17
Figura 4 – Processo de entrada de dados em redes TCP/IP[2] 17
Figura 5 – Processo de saída de dados em redes TCP/IP[2]..... 18
Figura 6 – Laço principal de controle do modelo uIP.[2] 21
Figura 7 – Diagrama de fluxo dos pacotes em redes TCP/IP[8]..... 31
Figura 8 – Sub-camada *mult* inserida na camada de transporte.[8] 32
Figura 9 – Cabeçalho do protocolo TCP[9] 33
Figura 10 – Sub-camada *mult* – Comunicação direta com as aplicações DHCP, TELNET, HTTPD, SFT e o uIP.[10]..... 34
Figura 11 – Cabeçalho protocolo SFT..... 41
Figura 12 – Pagina Web presente no uIP com múltiplas Aplicações. 54

LISTA DE TABELAS

Tabela 1 – Características da pilha uIP TCP/IP[1] 19
Tabela 2 – Protocolos disponíveis na pilha uIP TCP/IP.[1] 20
Tabela 3 – Tradução e adaptação do manual do uIP – opções de configuração dos protocolos [1]. 27
Tabela 4 – Estrutura do uip_conn no uIP[1] 27
Tabela 5 – Protocolos presentes na camada de aplicação 34
Tabela 6 – Estrutura da camada *mult*. 35
Tabela 7 – Estrutura de uma *socket* da biblioteca *protosocket*.[1, p. 102] 36
Tabela 8 – Elementos da estrutura *chat*_state usado pelo serviço CLI_WEBSOCK 37
Tabela 9 – Estrutura cli_str - Responsável em tratar os comandos da CLI 50
Tabela 10 – Estrutura str_t - Responsável em tratar as funcionalidades da CLI. 51

LISTA DE SIGLAS

API – *Application Programming Interface*

ARM – *Advanced Risc Machine*

ARP – *Address Resolution Protocol*

ARPA – *Advanced Research Projects Agency*

CLI – *Command-line Interface*

DMA – *Direct Memory Access*

DNS - *Domain Name System*

FreeRTOS –*free Real Time Operating System*

FTP – *File Transfer Protocol*

HTTP – *Hypertext Transfer Protocol*

ICMP – *Internet Control Message Protocol*

IGMP – *Internet Group Management Protocol*

IMAP – *Internet Message Access Protocol*

IP – *Internet Protocol*

MAC – *Media Access Control*

MTU – *Maximum Transmission Unit*

NXP – *Next eXPerience*

OSI – *Open System Interconnection*

PDA – *Personal Digital Assistants*

POP3- *Post Office Protocol*

PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul

RAM – *Random-access Memory*

RFC – *Request For Comments*

ROM – *Read-only memory*

SFP – *Simple File Transfer*

SLIP – *Serial Line Internet Protocol*

SMTP – *Simple Mail Transfer Protocol*

SRAM – *Static Random-access Memory*

SSH – *Security Shell*

TCP – *Transmission Control Protocol*

UDP – *User Daragram Protocol*

uIP – micro Internet Protocol

USB – *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO 12

1.1 PROPOSTA DE DESENVOLVIMENTO 12

1.2 ORGANIZAÇÃO DA PROPOSTA 13

2 SISTEMAS EMBARCADOS 14

2.1 KIT DE DESENVOLVIMENTO COM ARQUITETURA ARM7 14

3 REDES TCP/IP 17

4 PILHA uIP-TCP/IP 19

4.1 SIMPLIFICAÇÕES DO MODELO TCP/IP 20

4.2 PROTOCOLOS IMPORTADOS 22

4.3 APIs EMBARCADAS 23

4.3.1 *PROTOTHREADS* 23

4.3.2 *PROTOSOCKETS* 24

4.4 IMPORTANDO O uIP NO ARM7 24

5 FUNÇÕES DO uIP 25

5.1 CONFIGURAÇÕES DE PROJETO PARA O uIP 25

5.1.1 DEFINIÇÕES DO UIPOPT.H 25

5.2 PRINCIPAIS FUNÇÕES DO uIP 27

6 NOVAS FUNCIONALIDADES 30

6.1 uIP COM MULTIPLAS APLICAÇOES 30

6.1.1 SUB-CAMADA *MULT* 32

6.1.2 TRATAMENTO DE PACOTES NA CAMADA *MULT* 33

6.2 COMUNICAÇÃO COM *SOCKETS* 35

6.2.1 BIBLIOTECA *PROTOSOCKETS* 36

6.2.2 APLICAÇÃO CLI_WEBSOCK NO uIP 37

6.2.2.1 CONFIGURAÇÃO DA CLI_WEBSOCK 37

6.2.2.2 FUNÇÃO *CHAT*_APPCALL NA CAMADA DE APLICAÇÃO 38

6.2.3 APLICAÇÃO *CHAT* NO *HOST* DESTINO 39

6.2.4 DESCRIÇÃO DO USO DA APLICAÇÃO 39

6.3 PROTOCOLO SFT – *Secure File Transfer* 41
6.3.1 CARACTERÍSTICAS DA APLICAÇÃO SFTD 42
6.3.1.1 COMANDOS SUPORTADOS NA APLICACAO SFTD 42
6.3.1.2 FUNCIONAMENTO DA APLICAÇÃO SFTD 43
6.3.2 CARACTERISITICAS DO SERVIDOR SFT 44
6.3.2.1 COMANDOS DO SERVIDOR SFT 44
6.3.2.2 FUNCIONAMENTO DO SERVIDOR SFT 45
6.3.2.3 SENDSEQ – ENVIO DE ARQUIVOS SEQUENCIAIS 46
6.3.2.3.1 CRIANDO IMAGENS PARA ENVIAR PELO SERVIDOR 48
6.3.3 DESCRIÇÃO DO USO DO SERVIDOR / CLIENTE SFT 48
6.4 CLI – INTERFACE DE LINHA DE COMANDO 49
6.4.1 COMANDOS DE CONFIGURAÇÃO DO uIP PELA CLI 49
6.4.2 INSERINDO FUNCIONALIDADES NA CLI 50
7 APLICAÇÕES DO uIP 52
7.1 SERVIÇOS DE TELNET 52
7.2 SERVIÇOS DE WEBSERVER 53
8 CONCLUSÃO 55
9 REFERÊNCIAS 56

1 INTRODUÇÃO

Em muitos projetos de sistemas embarcados a comunicação com a Internet é um ponto fundamental para a aplicação. Atendendo a estas necessidades, a pilha uIP TCP/IP foi desenvolvida para prover a conexão entre a rede de Internet e os microcontroladores. Esta pilha oferece alguns serviços na sua camada de aplicação que podem ser usados em sistemas embarcados de acordo com as necessidades que o projeto precisa atender.

A pilha TCP/IP completa utiliza muito recurso do processador e muita memória, o que inviabiliza o seu uso em microcontroladores. O uIP tem as quatro camadas do modelo TCP/IP: aplicação, transporte, rede e física. Essas camadas no uIP são bem resumidas para que seja possível utilizá-los em sistemas embarcados.

O uIP foi desenvolvido por Adam Dunkels [1], podendo ser usado em sistemas dedicados, com recursos de hardwares limitados, como microcontroladores de 8 bits. O uso de memória com está pilha é bastante pequeno, requerendo apenas 30 *bytes* de memória por conexão[2].

A pilha uIP oferece alguns serviços que foram adaptados de acordo com as RFCs correspondentes. Estes serviços ficam disponíveis na camada de aplicação do modelo TCP/IP, como: TELNET, HTTP entre outros. A versão atual do uIP tem a limitação de permitir que apenas um serviço seja executado por vez.

A proposta deste trabalho é testar o uIP em uma arquitetura especifica e fazer melhoramentos de modo a superar algumas das suas limitações.

1.1 PROPOSTA DE DESENVOLVIMENTO

Em determinados projetos é importante que se tenha mais de um serviço disponível para o meio de comunicação entre o sistema e a rede de Internet. Com o

objetivo de aprimorar a camada de aplicação do uIP para projetos embarcados, este trabalho propõe-se:

- Incorporar e testar a pilha uIP TCP/IP na plataforma com o processador LPC2378;
- Superar limitação do uIP TCP/IP para um único serviço na camada de aplicação. Desenvolvendo um sistema que suporte múltiplos serviços simultaneamente.
- Desenvolver um comunicador usando *sockets* TCP/IP do uIP;
- Implementar um protocolo para transferência de arquivos;
- Criar uma interface de linha de comando para funcionalidades do sistema;
- Documentar as aplicações desenvolvidas e os serviços incorporados para que novos projetos possam utilizá-lo como base.

1.2 ORGANIZAÇÃO DA PROPOSTA

Este trabalho está organizado da seguinte forma:

- Seção 1 – Introdução do uIP e da proposta de desenvolvimento;
- Seção 2 – Apresentação do conceito de sistemas embarcados;
- Seção 3 – Resumo do funcionamento do modelo TCP/IP.
- Seção 4 – Apresentação da pilha uIP-TCP/IP, das simplificações realizadas para que pudesse ser embarcada e das principais API;
- Seção 5 – Apresentação das funções essenciais do uIP e as configurações para um sistema que utiliza a pilha como base;
- Seção 6 – Apresentação das novas funcionalidades, abordando o desenvolvimento de uma camada para atender múltiplos serviços, o protocolo SFT e a comunicação com *sockets* TCP;
- Seção 7 – Abordagem das alterações feitas nas aplicações já existentes na versão 1.0 do uIP.
- Seção 8 – Conclusão do desenvolvimento das novas aplicações.
- Seção 9 – Bibliografia utilizada.

2 SISTEMAS EMBARCADOS

Cada vez mais presentes em nosso dia-a-dia, os sistemas embarcados são sistemas destinados a atender a conjunto de tarefas específicas. Estes sistemas possuem características muito parecidas com os computadores atuais, com processadores, memórias, interfaces e meios de interação com o ambiente externo, porém em uma escala bastante inferior, pois o seu objetivo é atender apenas a um fim específico de forma segura, garantindo uma maior confiabilidade[3].

Os processadores utilizados em sistemas embarcados são geralmente de baixo custo e baixo consumo de energia. Dentre as várias arquiteturas usadas em sistemas embarcados, destaca-se a arquitetura ARM. O *core* do ARM é licenciado para vários fabricantes de microcontroladores embarcados, estando presente em telefones celulares, PDA, impressoras, câmeras, etc.

A aplicação desenvolvida neste projeto utilizou a arquitetura ARM7 com o microprocessador LPC2378 da NXP que executa instruções de 32 e 16 bits, inserido no kit de desenvolvimento da PUCRS.

2.1 KIT DE DESENVOLVIMENTO COM ARQUITETURA ARM7

No desenvolvimento do projeto de múltiplas aplicações para o uIP foi utilizado o kit de desenvolvimento da PUCRS com o microprocessador LPC2378 da família ARM.

O microprocessador LPC2378 da NXP é da arquitetura ARM7 e executa instruções de 32 e 16 bits. Dentre as suas muitas características, as de maior importância e necessárias para o desenvolvimento deste projeto, destacam-se [4]:

- Interface Ethernet 10/100Mbps;

- Suporte a drive USB 2;
- 512kB de memória flash;
- 32kB de memória RAM;
- Interface de rede Ethernet MAC com DMA;
- 16kB de SRAM para interface Ethernet;
- Quatro Interfaces para comunicação serial.

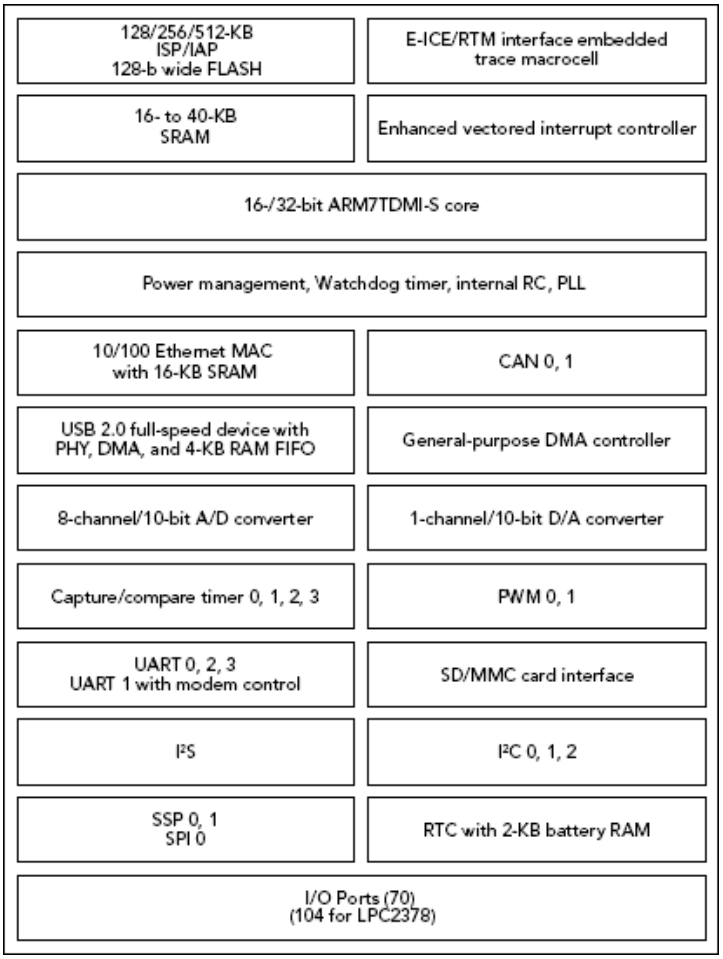

Figura 1 – Diagrama de blocos do LPC2378 da NXP [3]

Junto ao microprocessador LPC2378 o Kit de desenvolvimento PUCRS oferece outros recursos que foram utilizados para execução deste trabalho, destacando-se:

- LCD Nokia 6100
- *Display* de LCD 16x2
- Sensor de temperatura

- USB device
- Conversor serial USB
- Interface Ethernet

Figura 2 – Kit de Desenvolvimentos PUCRS[4]

3 REDES TCP/IP

O modelo TCP/IP utilizado na rede de Internet atual foi proposto pela ARPA (*Advanced Research Projects Agency*). Ele é divido em uma pilha com quatro camadas: Aplicação, Transporte, Rede e Física / Enlace [5]. Esta divisão de camadas foi feita tendo como base nas sete camadas presente no modelo OSI de comunicação.

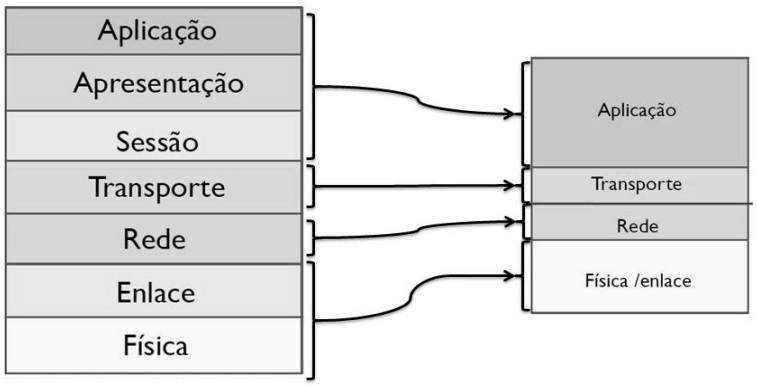

Figura 3 – Modelo OSI vs Modelo TCP/IP

Quando um pacote chega da rede, o modelo TCP/IP irá demultiplexar este pacote. Primeiramente do meio físico para uma interface de rede, passando pela camada de transporte (protocolo TCP/UDP) sendo direcionado para a aplicação (Figura 4).

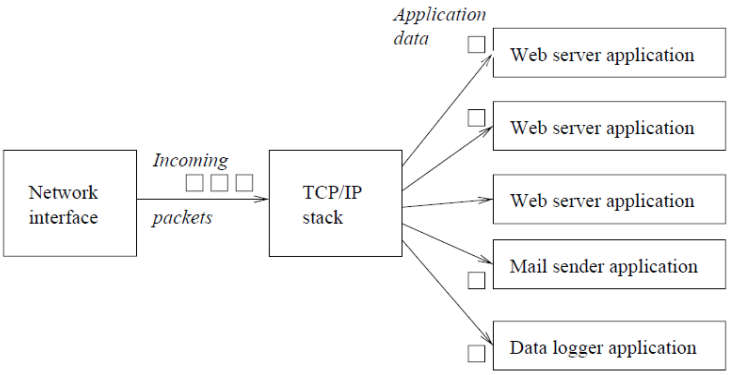

Figura 4 – Processo de entrada de dados em redes TCP/IP[2]

Após o pacote ser tratado pela aplicação, ele é enviado ao seu destino por um processo inverso de multiplexação dos pacotes. A aplicação gera um pacote de resposta para o *host*, que passa pela camada transporte, é encaminhado para interface de rede e lançada no meio físico (Figura 5).

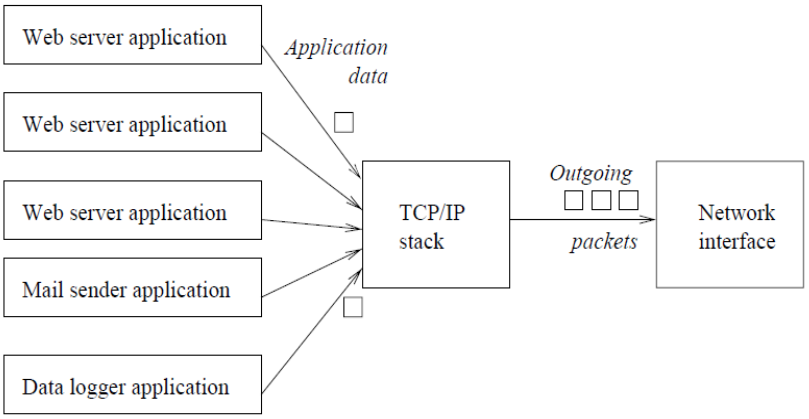

Figura 5 – Processo de saída de dados em redes TCP/IP[2]

Os principais protocolos utilizados nas redes TCP/IP é o protocolo IP (*Internet Protocol*) usado na camada de rede e o protocolo TCP (*Transmission Control Protocol*) usado na camada de transporte. Além destes protocolos, muitos outros estão presentes nas redes de Internet para garantir segurança e confiabilidade na troca de informações.

4 PILHA uIP-TCP/IP

Com o sucesso das redes de Internet, os sistemas embarcados evoluíram da comunicação serial para poder atender também a comunicação com estas redes. O modelo TCP/IP tradicional requer muitos recursos, tanto no tamanho do código como em memória. Isto torna o uso das implementações tradicionais inviáveis para o uso em pequenos sistemas embarcados com recursos limitados de memória [2]. Assim, foram desenvolvidos modelos de protocolos TCP/IP mais simples para atender esta demanda.

CARACTERÍSTICAS uIP TCP/IP	
PARA O DESENVOLVEDOR	Projeto bem documentado.
	Código Fonte comentado.
	Diversas RFCs já implementadas e disponíveis para aplicação.
	Livre para uso comercial e não comercial.
PARA A APLICAÇÃO	Código Pequeno.
	Baixo uso de memória RAM.
	Suporte aos protocolos ARP, SLIP, IP, UDP, ICMP e os protocolos TCP.
	Aplicações já disponíveis: servidor web, SMTP, TELNET, DNS
	Aceita múltiplas conexões TCP ativas simultaneamente.
	RFCs TCP e IP implementadas, incluindo controle de fluxos, remontagem de fragmentos e retransmissão de informações de acordo com *timeout*.

Tabela 1 – Características da pilha uIP TCP/IP[1]

A pilha de comunicação uIP TCP/IP é um projeto desenvolvido por Adam Dunkels[1] do Instituto Sueco de Ciência da Computação (*Swedish Institute of Computer Science*) em 2006. O uIP procura implementar a pilha de protocolos TCP/IP de forma simplificada, minimizando o uso de memória.

Na Tabela 1 são apresentadas as características em dois aspectos importantes, uma do ponto de vista do desenvolvedor, apresentando os benefícios do uso do uIP TCP/IP para projetos que precisam se comunicar com a rede, outra do ponto de vista do quanto à aplicação em desenvolvimento pode ter a acrescentar utilizando-o como base no projeto.

CAMADA	PROTOCOLOS IMPORTADOS	PROTOCOLOS NÃO IMPORTADOS
Rede	IP, ICMP e ARP	RARP, IGMP, etc.
Transporte	TCP e UDP	SCTP, DCCP, etc.
Aplicação	HTTP, SMTP, TELNET e DNS.	SSH, RDP,NNTP, POP3, IMAP, etc.

Tabela 2 – Protocolos disponíveis na pilha uIP TCP/IP.[1]

Com os principais protocolos implementados no uIP (Tabela 2), é possível realizar a comunicação direta com redes TCP/IP internas (intranet) ou até mesmo com a Internet global, podendo oferecer serviços como: paginas Web, transferências de e-mails e comunicação ponto-a-ponto com a internet.

4.1 SIMPLIFICAÇÕES DO MODELO TCP/IP

A RFC1122 define os requisitos de *software* necessários para a comunicação de um *host* com a Internet, abordando um conjunto de protocolos que devem ser implementados para que este possa se comunicar na rede [6].

Para poder atender as especificações da RFC e ao mesmo tempo poder ter um modelo simplificado da pilha TCP/IP, muitos serviços disponíveis no modelo padrão foram removidos, abordando apenas os essenciais para a aplicação.

Dentre as principais alterações realizadas para poder atender a sistemas com recursos limitados estão [2]:

- Eliminação do suporte a remontagem de pacotes IP fragmentados. Estes pacotes são bastante raros de ocorrerem, as redes costumam assumir o tamanho de 1500 para a MTU[1] dos pacotes enviados.
- Remoção do mecanismo para reportar erros.

[1] MTU é a unidade máxima de transmissão que um protocolo de comunicação pode transmitir [16].

- Remoção da configuração dinâmica de tipo de serviços para conexões TCP. Uma vez que poucos aplicativos utilizam este recurso.
- Eliminação da alocação de memória dinâmica. Uso de apenas uma memória global.
- Inserção de uma pequena janela de recepção de conexões. Apenas um único segmento TCP pode estar na rede por conexão.

O funcionamento do modelo uIP baseia-se em um laço infinito verificando dois fatores essenciais, se um pacote chegou da rede e se ocorreu *timeout* dos pacotes enviados.

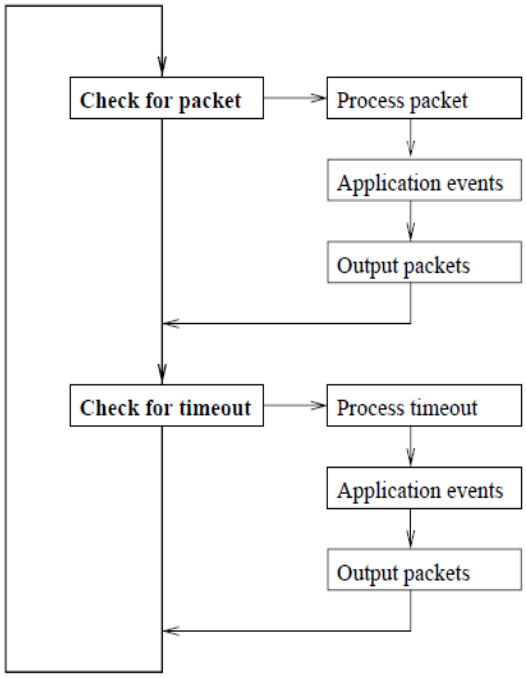

Figura 6 – Laço principal de controle do modelo uIP.[2]

No campo "*check for packet*" da Figura 6, é feito o monitoramento dos pacotes vindos da rede. Quando um pacote chega da rede ele é encaminhado diretamente para a aplicação. Depois de tratado pela aplicação correspondente o pacote de resposta é retransmitido para a rede. O campo "*check for timeout*" é usado pelo protocolo TCP para controle de envio de pacotes e latências da rede.

4.2 PROTOCOLOS IMPORTADOS

Os protocolos implementados no uIP atendem os requisitos essenciais presentes em suas RFCs correspondentes, passando por algumas alterações para atender a sistemas críticos.

No protocolo IP o dado do pacote é mapeado na estrutura do uIP para ser remontado após ter sido tratado pela aplicação. Como a implementação possui apenas um buffer para os pacotes, não é possível realizar a remontagem de mais de um pacote simultaneamente para envio na rede.

O protocolo ICMP foi simplificado oferecendo suporte apenas para as mensagens de respostas, não sendo possível a sua utilização para descoberta de caminhos.

No protocolo TCP as conexões são identificadas de acordo com a porta de entrada dos pacotes, permitindo apenas um segmento por conexão. Neste protocolo foi removido o mecanismo de “janela deslizante”, onde vários dados são enviados seqüencialmente com espera de confirmação no ultimo pacote da seqüência. Esta remoção não altera a confiabilidade do sistema e reduz o uso de memória.

Quando um pacote é enviado para rede externa, o “*buffer”* do uIP fica livre para novas transmissões, caso o pacote necessite ser retransmitido, o uIP avisa através de *flags* a aplicação correspondente ao pacote. A aplicação responsável irá criar um novo pacote reproduzindo os mesmos dados do pacote anterior e encaminhá-lo na rede novamente.

Para controle do fluxo de transmissão de pacotes o uIP não envia pacotes maiores do que o receptor pode receber. Desta forma, antes de um pacote ser enviado ao seu destino, é verificado qual o tamanho máximo que destino suporta.

O uIP suporta apenas uma conexão TCP por vez. Desta forma não é necessário implementar um controle de congestionamento para conexões simultâneas.

Com estas simplificações nos protocolos das redes TCP/IP foi possível reduzir o tamanho do código utilizado para implementar a pilha uIP TCP/IP e a quantidade de memória destinada a atender comunicações da Internet com sistemas embarcados. O uIP requer apenas 30 bytes de memória por conexão[2].

4.3 APIs EMBARCADAS

O modelo uIP TCP/IP necessita principalmente das funcionalidades de duas API's, *protothreads* para geração de multi-tarefas e *protosockets* uma API de *sockets*, para comunicação da aplicação com a interface TCP/IP.

4.3.1 *PROTOTHREADS*

Sistemas multi-tarefas exigem uma grande quantidade de memória para poder atender às funções em execução. *Protothread*s foi desenvolvido por Adam Dunkels especificamente para sistemas embarcados, provendo múltiplos serviços em uma mesma aplicação com baixo custo.

*Protothread*s é uma biblioteca voltada a atender sistemas orientados a eventos, sem gerar sobrecargas ao sistema. É extremamente leve com a execução do código seqüencial permitindo bloquear funções para atender aos eventos.

De acordo com Dunkels [1]

> "Como *protothread*s não guarda pilha de contexto através de uma chamada de bloqueio, variáveis locais não são preservadas quando ocorre *protothread*s. Variáveis locais devem ser usadas com cuidado, em caso de duvidas, não use variáveis locais dentro de *protothread*s."

Uma das grandes qualidades das *protothread*s é que cada *protothread* requer entre dois e doze bytes de memória apenas. Diferente de sistemas que fazem alocação de pilha para cada nova tarefa criada, em *protothread*s existe apenas uma pilha onde as tarefas são empilhadas.

4.3.2 *PROTOSOCKETS*

A biblioteca *Protosockets* fornece uma interface para a pilha uIP semelhante as interfaces de *sockets* BSD[2] tradicionais.[1]

Protosockets foi escrito para trabalhar com conexões TCP e utiliza *protothread*s para o fluxo de execuções de suas tarefas, herdando as mesmas limitações para as suas variáveis.

A biblioteca apresenta as seguintes limitações:

- Envio de dados sem a possibilidade de retransmissão de pacotes,
- Leitura de dados sem que eles estejam em pacotes fragmentados.

As especificações mais detalhadas de suas funções e o modo de como utilizá-las em uma aplicação serão abordados nos próximos capítulos.

4.4 IMPORTANDO O uIP NO ARM7

A importação do uIP para o LPC2378 foi baseada em aplicação desenvolvida para o ARM7 LPC2368 da NXP [7]. Nesta aplicação, é apresentado um *Webserver* Demo utilizando o uIP em conjunto com o FreeRTOS.

O kit de desenvolvimento utilizado neste projeto possui o processador ARM7 LPC2378 da NXP. Como ambos possuem a mesma compatibilidade de pinos e são da mesma família LPC 23xx, o funcionamento de um sistema no outro é bastante similar. Não tendo problemas para migrar de um sistema para o outro.

[2] Sockets BSD – Interface padronizada de sockets originada no sistema operacional UNIX BSD (Berkeley Software Distribution), conhecido também como Berkeley Sockets. [15]

5 FUNÇÕES DO uIP

O uIP TCP/IP foi desenvolvido para atender a comunicação com redes TCP/IP. Entretanto, para que esta comunicação possa ser possível, algumas configurações devem ser definidas conforme a necessidade da aplicação.

5.1 CONFIGURAÇÕES DE PROJETO PARA O uIP

As principais configurações do uIP para um projeto estão presentes no arquivo **uipopt.h**. Este arquivo não deve ser alterado, pois possui as definições necessárias para o uIP[1].

O arquivo ***uip-conf.h*** serve para configurar o uIP para ser adaptado em um projeto. Algumas definições devem ser feitas em tempo de projeto, para serem usadas pelo compilador e atender as exigências de projeto.

5.1.1 DEFINIÇÕES DO UIPOPT.H

As configurações específicas de um projeto para o uIP são feitas com as definições iniciada em “UIP_”. Dividido conforme os protocolos IP, UDP, TCP, ARP e configurações gerais.

CONF.	DEFINIÇÃO	FUNCIONALIDADE
Estáticas	UIP_FIZEDADDR	Determina se o endereço de IP será fixo ou dinâmico. Para protocolos como DHCP ele deve ser fixado em 1.
	UIP_PINGADDRCONF	Habilita endereço para responder a requisições de PING.
	UIP_FIXEDETHADDR	Especifica se o módulo ARP deve ser configurado com o endereço MAC fixo ou não.
IP	UIP_TTL	Configuração do TTL (time to live) para os pacotes do uIP

CONF.	DEFINIÇÃO	FUNCIONALIDADE
	UIP_REASSEMBLY	Ativa o suporte a remontagem de pacotes.
	UIP_REASS_MAXAGE	Tempo máximo que um fragmento IP deve esperar no buffer de montagem antes de ser descartado.
UDP	UIP_UDP	Configuração para habilitar suporte a UDP ou não.
	UIP_UDP_CHEKSUMS	Cálculo do checksum dever ser usado para pacotes UDP.
	UIP_UDP_CONNS	Quantidade de conexões UDP
TCP	UIP_ACTIVE_OPEN	Determina se o suporte a abertura de conexões uIP deve ser compilado.
	UIP_CONNS	Determina o máximo de conexões TCP simultâneas.
	UIP_LISTENPORTS	Determina o máximo de portas que podem ser ouvidas pelo TCP.
	UIP_URGDATA	Determina se deve ser compilado o suporte a dados urgente do TCP.
	UIP_RTO	Tempo limite para retransmissão.
	UIP_MAXRTX	Número máximo que um pacote deve ser retransmitido antes da conexão ser abortada.
	UIP_MAXSYNRTX	Número máximo de vezes que um segmento SYN deve ser retransmitido antes da conexão ser considerada mal sucedida.
	UIP_TCP_MSS	Tamanho máximo do segmento TCP.
	UIP_RECEIVE_WINDOW	Tamanho da janela de recepção.
	UIP_TIME_WAIT_TIMEOUT	Quanto tempo uma conexão pode ficar no estado *TIME_WAIT*
ARP	UIP_ARPTAB_SIZE	Tamanho da tabela ARP.
	UIP_ARP_MAXXAGE	O tempo máximo dos dados na tabela ARP medidas em 10ths de segundos.
GERAIS	UIP_BUFSIZE	Tamanho do pacote do uIP.
	UIP_STATISTICS	Determina se a estatísticas devem ser inseridas no projeto.
	UIP_LOGGING	Determina se o login para alguns eventos devem

CONF.	DEFINIÇÃO	FUNCIONALIDADE
		ser compilados no projeto.
	UIP_BROADCAST	Suporte a mensagens em broadcast.
	UIP_LLH_LEN	Tamanho do cabeçalho do *link*
	UIP_LOG	Habilitar impressão uma mensagem de log do uIP.

Tabela 3 – Tradução e adaptação do manual do uIP – opções de configuração dos protocolos [1].

5.2 PRINCIPAIS FUNÇÕES DO uIP

Para que o serviço de uIP possa ser utilizado e para criação de novas funcionalidades ou aplicações é necessário observar e entender algumas funções essenciais de projetos com o uIP.

A principal estrutura do uIP é chamada de *uip_conn* e é utilizada pelos serviços de conexão TCP.

ELEMENTO	FUNCIONALIDADE
Ripaddr	Endereço IP do *host* remoto
Lport	Porta TCP local
Rport	Porta TCP remota
rcv_nxt[4]	Número da próxima seqüência que esperamos receber
snd_nxt[4]	Número da seqüência que foi enviada pelo uIP.
Len	Tamanho dos dados que foi previamente enviado.
Mss	Tamanho máximo do segmento para atual conexão.
initialmss	Tamanho máximo do segmento para a conexão inicial.
As	Cálculo de *time-out* para o estado de retransmissão.
Sv	Cálculo de *time-out* para o estado de retransmissão.
Rto	*Time-out* para retransmissão.
tcpstateflags	Estado das flags de TCP
Timer	Tempo de retransmissão.
Nrtx	Número de retransmissões para o ultimo pacote enviado.
Appstate	Estrutura do tipo tcp_appstate_t, aponta para a aplicação em execução.

Tabela 4 – Estrutura do uip_conn no uIP[1]

Quando uma conexão TCP é realizada, o uIP aponta para estrutura *uIP_conn*, que possui como um dos elementos principais, a estrutura tcp_*appstate_t appstate*. Esta estrutura é dinâmica e aponta para o serviço que será usado na camada de aplicação, através da função *UIP_APPCALL*.

Quando um evento ocorre no uIP, a estrutura apontada por UIP_APPCALL é chamada para tratar o evento. Em sistemas que utiliza o uIP para apenas uma aplicação, a estrutura desta aplicação é diretamente apontada para esta definição.

No desenvolvimento deste trabalho, foi criada a sub-camada *mult*[3] para atender múltiplas aplicações para todas as vezes que ocorrerem eventos em conexões TCP.

Para criação dos serviços na camada de aplicação, algumas funções são essenciais, dentre elas estão:

- **UIP_newdata –** usada para verificar se existe um novo dado vindo da rede.
- **UIP_datalen –** retorna o tamanho dos dados recebidos da rede.
- **UIP_send –** função para envio de dados para rede.
- **UIP_rexmit –** função para configurar a *flag* de retransmissão de pacote.
- **UIP_close –** função para fechar a conexão corrente de uma aplicação.
- **UIP_closed –** Se a conexão foi encerrada com o *host* destino, esta função retorna verdadeiro.
- **UIP_abort –** função para aborta a conexão corrente, útil para casos de fatal erro por exemplo.
- **UIP_timeout –** função para verificar se o tempo de limite de conexão foi atingido.
- **UIP_poll –** função que faz sondagem da aplicação de tempo em tempo para verificar se ela esta sendo monitorada pelo uIP.
- **UIP_connect –** função que retorna um ponteiro para a estrutura *uip_conn.*
- **UIP_connected –** função que retorna verdadeiro se o pedido foi feita por uma nova conexão TCP.

[3] Camada mult está detalhada na seção 6.1.1

- **UIP_listen –** função para adicionar uma nova porta na fila de portas a serem monitoradas.

Tendo estas funções como base é possível desenvolver uma nova aplicação para um projeto com o uIP e a comunicação com a pilha TCP/IP padrão, presente em um *host* destino.

6 NOVAS FUNCIONALIDADES

Com o objetivo de superar a limitação do uIP de atender apenas uma aplicação, foi desenvolvido um sistema onde é possível ter múltiplas aplicações rodando em conjunto. Foram criadas duas novas aplicações que atuam em conjunto com o uIP, um *chat* que utiliza os serviços disponíveis na API de *protosockets* e o protocolo SFP para transferências de arquivos.

6.1 uIP COM MULTIPLAS APLICAÇOES

Com a versão 1.0 do uIP[4] era possível executar apenas uma aplicação no sistema. Desta forma, o projeto limitava-se a atender apenas um tipo de serviço vindo da rede, seja ele TELNET, HTTP, SMTP, etc. Com a funcionalidade de múltiplos serviços em execução, é possível, em um mesmo sistema, ter vários protocolos sendo atendidos.

Quando um pacote vem da Internet, na **camada física** é validado o MAC, verificando se o destino do pacote é o *host* local. Depois ele passa para a **camada de rede** para verificação dos IP's e do tipo de conexão, podendo ser TCP ou UDP. Em seguida o pacote é encaminhado para a estrutura de tratamento do pacote na **camada de transporte**. Nesta camada, se o protocolo é TCP, é validada a conexão e encaminhado para a **camada de aplicação**, em caso de pacotes UDP, não é feita validação da conexão já que este protocolo não utiliza esta segurança.

[4] uIP versão 1.0 – projeto - http://www.sics.se/~adam/uIP/index.php/Main_Page

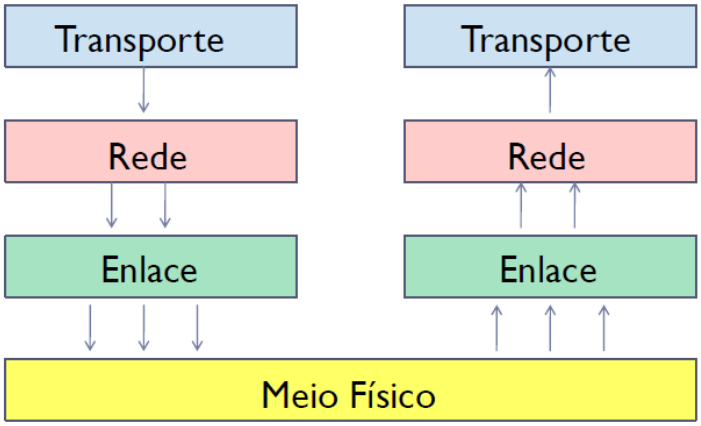

Figura 7 – Diagrama de fluxo dos pacotes em redes TCP/IP[8]

Para os pacotes de serviços UDP, não é feito nenhum processamento da informação, pois o uIP não implementa este serviço[2], mas foca-se nos protocolos de IP, ICMP e TCP.

Uma aplicação que utiliza os serviços de TCP deve, em sua inicialização no uIP, adicionar a sua porta para leitura. Esta inicialização se dá através da função *uIP_listen*(PORTA), onde o campo PORTA refere-se a porta que a aplicação utiliza para realizar o seu serviço.

Como a camada de aplicação atende apenas um serviço, o protocolo TCP desenvolvido para o uIP não faz verificação do tipo de porta para chamar a camada de aplicação. Desta forma, um pacote vindo da rede, se a sua porta estiver habilitada, é encaminhado diretamente para a aplicação sem verificação do tipo de serviço.

Como exemplo, suponhamos um *webserver* de aplicação (porta 80) em execução, com a porta 21 (de serviços de FTP) habilitada para verificação. Quando a minha aplicação receber serviços de FTP ela irá encaminhá-los para serem tratados pelo *webserver*. Como o *webserver* não está habilitado para tratar os serviços de FTP o pacote não será reconhecido pela a aplicação.

6.1.1 SUB-CAMADA *MULT*

Para criar múltiplas aplicações executando na mesma plataforma, foi desenvolvida uma sub-camada para refinamento dos pacotes dentro da camada de transporte, chamada “*mult”*. Desta forma, a camada de transporte irá encaminhar os pacotes TCP para ela.

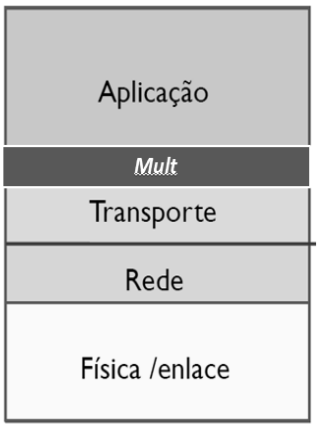

Figura 8 – Sub-camada *mult* inserida na camada de transporte.[8]

A sub-camada *mult* funciona como um gerenciador de pacotes TCP. Quando a camada de transporte envia os pacotes para ela, é feita uma verificação do tipo de serviço que este protocolo pretende executar. Esta verificação é feita através do campo *destination_port*. Após a verificação do tipo de serviço, a sub-camada *mult* chama a aplicação responsável em tratar o pacote.

Além de gerenciar os pacotes TCP e encaminhá-los para a devida aplicação, a camada *mult* é responsável por habilitar as portas que devem ser “ouvidas” pela aplicação uIP.

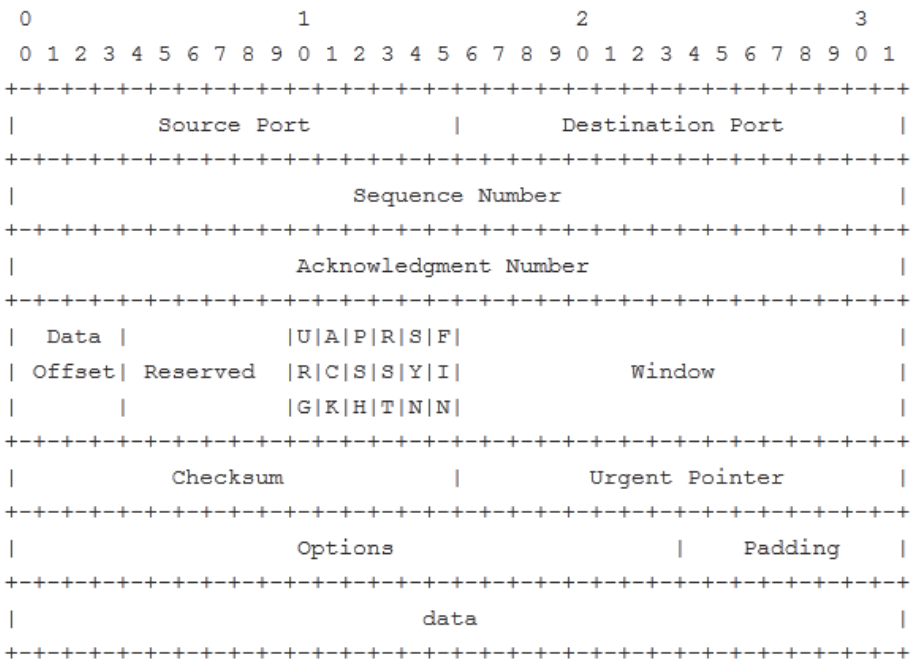

Figura 9 – Cabeçalho do protocolo TCP[9]

Para validação desta sub-camada, foram habilitados quatro serviços na camada de aplicação, sendo: TELNET, *WebServer, Chat* e protocolo SFT.

6.1.2 TRATAMENTO DE PACOTES NA CAMADA *MULT*

Para inicializar as aplicações foi criada a função *services_init* que coloca as portas referente as aplicações na pilha de portas a serem monitoradas. Quando uma nova aplicação é inserida no uIP, a sua porta deve ser adicionada na pilha através desta função.

Desta forma, todos os pacotes TCP que chegarem da rede, se a porta de destino estiver inclusa nesta função ele será encaminhado para a sub-camada *mult*, para verificação do tipo de aplicação.

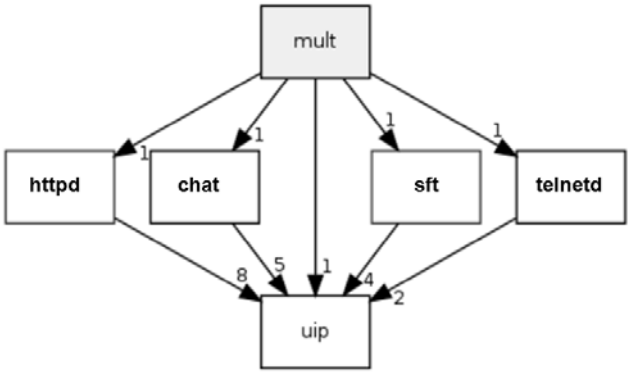

Figura 10 – Sub-camada *mult* – Comunicação direta com as aplicações DHCP, TELNET, HTTPD, SFT e o uIP.[10]

Após a inicialização das portas de monitoramento, a camada *mult* começa atuar em sua principal função, de gerenciar os serviços e encaminhá-los para aplicação correspondente. Este controle é feito através da função *services_appcall.* O pacote ao chegar nesta camada já possui a estrutura TCP (*uip_conn*) preenchida, com o campo *destination_port* já mapeado.

A chamada da função para tratar um pacote é feita de acordo com a porta que a aplicação utiliza para se comunicar na Internet. Desta forma, a função *services_appcall* monitora o campo *destination_port* do protocolo TCP e, se o valor corresponder a um valor da lista de monitoração, é chamado a aplicação para aquela porta.

PROTOCOLO	PORTA
TELNET	23
HTTP	80
CHAT	Porta dinâmica
SFT	45

Tabela 5 – Protocolos presentes na camada de aplicação

As funções para destino de cada porta devem estar adicionados a estrutura da sub-camada *mult*, como pode ser observado na Tabela 6. A estrutura *mult* não possui nenhum campo destinado a controle de dados, ela utiliza a primitiva *union* para atender apenas a aplicação que deseja acesso, economizando memória para a

aplicação. Desta forma, em vez de alocar espaço em memória para todas as estruturas que atende a camada *mult*, apenas a de maior tamanho é alocada.

Elemento	Funcionalidade
httpd_state * http_st	Aponta para a estrutura de tratamento de pacotes HTTP.
telnetd_state telnet_st	Aponta para a estrutura de tratamento de pacotes TELNET.
***chat*_state * *chat*_st**	Aponta para a estrutura de tratamento de pacotes da CLI_WEBSOCK.
Sftd_state * sft_st	Aponta para a estrutura de tratamento de pacotes SFT.

Tabela 6 – Estrutura da camada *mult*.

Na validação desta sub-camada foram adicionados quatro serviços, com as suas respectivas portas, como mostra a Tabela 5. Quando um destes tipos de serviços chega da rede, ele é encaminhado para aplicação que irá tratar o pacote.

É importante observar que está sub-camada é usada apenas para pacotes que chegam da rede.

No modelo TCP/IP a verificação da porta e do tipo de serviço que o pacote deseja ter acesso deve ser feita pela camada de transporte. Por esta razão a sub-camada *mult* é classificada como um serviço da camada de transporte e não como um serviço da camada de aplicação.

6.2 COMUNICAÇÃO COM *SOCKETS*

A pilha uIP utiliza a API *protosockets* para realizar alguns serviços, como o WebServer e o WebClient. Esta API foi desenvolvida junto com o uIP e oferece a possibilidade de comunicação com um *host* remoto através de *sockets* TCP.

Com o objetivo de desenvolver uma aplicação que utilize este serviço e validá-lo na execução de mult-aplicações, foi criado um sistema de comunicação entre cliente e servidor usando *sockets* TCP.

6.2.1 BIBLIOTECA *PROTOSOCKETS*

A biblioteca *protosockets* utiliza a mesma interface de *sockets* BSD. A estrutura usada para definição de uma *socket* é chamada *psock*.

Elementos da Estrutura	Funcionalidade
PT	Thread usado para executar as funções da estrutura *psock.*
psockpt	Thread usado para executar as funções existentes dentro da estrutura *psock*.
***sendptr**	Ponteiro para o próximo dado para ser enviado.
***readptr**	Ponteiro para o próximo dado a ser lido.
***bufptr**	Ponteiro usado pelo buffer de entrada de dados.
sendlen	Número de bytes para ser enviado.
Readlen	Número de bytes para ser lido.
Psock_buf	Estrutura para saber o estado do buffer de entrada.
bufsize	Tamanho do buffer de entrada.
State	Estado da *protosocket*.

Tabela 7 – Estrutura de uma *socket* da biblioteca *protosocket*.[1, p. 102]

As principais funções utilizadas da API *protosockets* e que foram úteis para criação das trocas de mensagens são[11]:

- **psock_init –** Inicializa a *protosocket*.
- **psock_begin(psock) –** Inicializa as *protothread* para a função.
- **psock_send –** Envia dados na rede.
- **psock_send_str –** Envia uma *string* por *socket*.
- **Psock_readto –** Lê dados vindos da *socket* passada por parâmetro até que chegue o caractere especificado para limitar recebimento.
- **psock_close –** Fecha a *protosocket***.**
- **psock_end –** Finaliza a *protothread* criada para a *protosocket*.
- **psock_close_exit –** Fecha a *protosocket* e a *protothread* usadas pela estrutura.

Além das funções apresentadas aqui existem outras para diferentes condições, e que podem servir para atender determinadas aplicações, variando conforme o tipo de serviço a ser desenvolvido [1].

6.2.2 APLICAÇÃO CLI_WEBSOCK NO uIP

Para criar um comunicador entre um *host* origem e a aplicação embarcada foram desenvolvidos dois serviços, um serviço para a camada de aplicação do uIP, chamado *cli_websock*, usando a API *protosockets*, e outro serviço para a camada de aplicação de um computador com a pilha TCP/IP padrão, utilizando *sockets* TCP.

6.2.2.1 CONFIGURAÇÃO DA CLI_WEBSOCK

A aplicação *cli_websock* é inicializada com a porta 5555 como default. Sua inicialização é feita pela sub-camada *mult* que chama a função *chat_init*. Uma característica importante deste serviço é o uso de porta dinâmica, ou seja, depois do serviço ser inicializado, a sua porta pode ser alterada de acordo com a porta que será usada pelo *host* destino.

A estrutura para tratar pacotes usados no *chat* é chamada *chat_state* e possui a estrutura apresentada na Tabela 8.

Elemento	Funcionalidade
State	Monitora o estado do *chat* para determinar se o *chat* está habilitado para receber ou enviar mensagem.
Buff	Campo usado para armazenar a mensagem que será usada para envio ao *host* destino.
Bufflen	Campo com o tamanho da mensagem a ser enviada para o destino.
Inbuff	Campo usado para receber as mensagens de entrada. Limitado a mensagens de até 200 caracteres.

Tabela 8 – Elementos da estrutura *chat*_state usado pelo serviço CLI_WEBSOCK

Este serviço é síncrono e é monitorado pelo elemento *state* que é composto por três estados: *send_chat*, quando o *chat* está destinado ao envio de uma mensagem, *recv_chat*, quando a aplicação irá enviar uma mensagem para o destino e *quit_chat*, para finalizar a aplicação.

6.2.2.2 FUNÇÃO *CHAT*_APPCALL NA CAMADA DE APLICAÇÃO

O pacote TCP chega à camada *mult* com a porta *CHAT*_PORT e é encaminhado para a função *chat_appcall*, responsável em fazer o tratamento dos pacotes.

Caso seja a primeira conexão do serviço, a função *chat_appcall* irá inicializar uma *protosocket* com a função *PSOCK_INIT* passando como parâmetro o elemento *inbuff* e o seu tamanho. É importante ressaltar que ao passar o elemento *inbuff* como parâmetro ele será usado sempre para receber as mensagens do *host* conectado.

Feita a inicialização da *protosocket*, é chamada a função *chat_communication* que monitora o estado atual do *chat* e chama as configurações necessárias.

Toda vez que uma mensagem vai ser enviada ou recebida, a aplicação coloca a *protosocket na su*a função inicial através da função *psock_begin* passando a *protosocket* inicializada anteriormente como parâmetro.

Quando no estado de recebimento de mensagens (*recv_chat*), a aplicação *chat* fica bloqueada a espera de uma mensagem vinda da rede pela *protosocket*, usando a função *psock_readto*, é passada como parâmetro o caractere ‘\n’ para receber as mensagens do destino até a quebra de linha.

A porta serial do kit é usada como dispositivo de entrada para escrita de mensagens. A mensagem recebida do *host* destino é enviada diretamente para a saída da serial e o estado do *chat* é alterado para o envio de uma mensagem.

No estado de envio de mensagem (*send_chat*), o terminal fica aguardando uma mensagem vinda da serial até o caractere ‘\n’. Depois de recebida a

mensagem, é feita uma verificação para validar se é uma mensagem para fechar a conexão, ou se é uma mensagem para o *host* destino.

Caso a mensagem seja ***quit*** a mensagem é enviado para o *host* destino e a aplicação passa para o estado de *quit_chat* onde será encerrada a conexão. As funções *psock_close* e *psock_end*, finalizam os serviços da *protosocket* e das *protothread*s usadas na aplicação.

Caso a mensagem recebida seja diferente, ela é enviada para o destino pela função *psock_send_str* e o estado da mensagem é alterado para o recebimento de mensagem.

6.2.3 APLICAÇÃO *CHAT* NO *HOST* DESTINO

Para o *host* destino foi desenvolvida uma aplicação chamada *chat* usando *sockets* TCP configurada para receber o endereço de destino e a porta por parâmetros na sua inicialização.

Após estabelecida a conexão, a aplicação fica aguardando uma mensagem vinda do destino, como o *chat* desenvolvido é síncrono, a aplicação só estará apta para enviar uma mensagem, após ter recebido uma mensagem do *host* conectado pelo *socket*.

Quando a mensagem recebida é ***quit*** o sistema reconhece que houve o fim da comunicação com o *host* destino e encerra a aplicação. Outra forma de finalizar a aplicação pelo sistema é enviando apenas um caractere para o destino sendo **‘.’**.

6.2.4 DESCRIÇÃO DO USO DA APLICAÇÃO

Para poder ter acesso a aplicação é necessário que se tenha um barramento serial conectado a porta USB do kit de desenvolvimento. A aplicação *cli_websock* é

interpretada como umas das funções da aplicação CLI. Desta forma, quando o terminal serial está aberto, para chamar a aplicação basta informar o comando *chat* e a porta para acesso. Para este exemplo vamos utilizar a porta 1234 e o IP do aplicativo será 192.168.1.13. Sendo:

TCCOS> *chat* 1234

Como a aplicação CLI utiliza a porta serial para acesso a suas funções, o *chat* passa a ser um processo concorrente neste barramento. Desta forma, quando a aplicação *chat* entra em execução, ela bloqueia o funcionamento da CLI, dedicando o barramento para a comunicação entre o *host* origem e o *host* destino.

Bem Vindo ao *chat*.

1234-*CHAT*>

Após a inicialização do *chat* pela CLI, no *host* onde se pretende ter a execução deve se chamar a aplicação *chat* passando o endereço da conexão e a porta 1234. Como a aplicação foi desenvolvida usando o sistema Linux, a chamada da aplicação fica:

Sintaxe: ./*chat* ip_destino porta

./*chat* 192.168.1.13 1234

O aplicativo irá inicializar e tentar conexão com o IP destino, ao obter sucesso ele retorna a seguinte mensagem:

Conexao Estabelecida.

:>

Ficando no aguardo de uma mensagem vinda da aplicação *chat* embarcado.

Após receber uma mensagem do *host,* a aplicação poderá enviar sua mensagem de resposta. Para sair do *chat* é necessária que aplicação embarcada envie a mensagem ***quit***.

É importante ressaltar que para o funcionamento do *chat*, a aplicação deve ser inicializada primeiramente no sistema embarcado, indicando a porta usada para se comunicar, seguido da chamada da aplicação no *host*.

6.3 PROTOCOLO SFT – *Secure File Transfer*

A comunicação com a Internet possibilita a troca de dados em uma velocidade bastante alta. O protocolo SFT foi desenvolvido especificamente para realizar a transferências de arquivos entre um *host* e a aplicação atual do uIP.

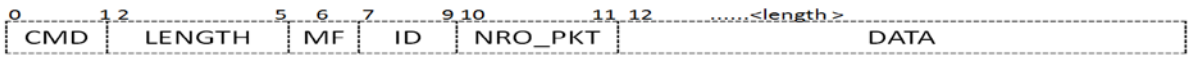

Figura 11 – Cabeçalho protocolo SFT

Como mostra a Figura 11 o cabeçalho do protocolo SFT é bastante simples, tendo os campos necessários para que a aplicação possa interpretar os dados e tratar. Sendo:

- **CMD –** Tipo do comando para tratamento do serviço, conforme mostra a seção 6.3.1.1.
- **LENGTH –** Tamanho do pacote de dados que será usado**.**
- **MF –** 0 caso não existe mais fragmentos do pacote e 1 caso esteja enviando partes de um arquivo.
- **ID –** Número identificador do pacote.
- **NRO_PKT –** Numero do pacote.
- **DATA –** Dados do arquivo que está sendo enviado.

A comunicação do SFT com o uIP é direta, com uma conexão TCP. Sendo gerenciado pela sub-camada *mult* e encaminhada para a aplicação responsável em tratar este protocolo, a aplicação *SFTD*.

6.3.1 CARACTERÍSTICAS DA APLICAÇÃO SFTD

A aplicação SFTD foi desenvolvida usando as funções do uIP para envio e recebimento de pacotes. Ela possui duas características importantes: a sua comunicação é síncrona e os pacotes recebidos não são fragmentados, ambas as características foram inseridas para garantir segurança e confiabilidade na troca de arquivos, pois a pilha uIP não reporta erros[2].

A comunicação síncrona inserida na aplicação faz com que todos os pacotes recebidos sejam respondidos com uma mensagem de confirmação. O servidor necessita desta mensagem de confirmação para poder enviar o próximo pacote.

Como os pacotes SFT não podem ser fragmentados, por medidas de segurança, foi inserido no cabeçalho do protocolo um campo MF para verificação se existe mais pacotes de um mesmo arquivo que está sendo recebido.

O kit de desenvolvimento utilizado não possui memória externa, logo a aplicação foi desenvolvida para receber arquivos do servidor e enviá-los para o *display* Nokia6100 conectado ao kit de desenvolvimento.

6.3.1.1 COMANDOS SUPORTADOS NA APLICACAO SFTD

O protocolo SFT foi criado para realizar envio de arquivos do *host* para o uIP. Os comandos da aplicação SFTD são:

- CMD_RECV – indica recebimento de um pacote de imagem da rede.
- CMD_RECV_ANS – resposta da aplicação para o servidor informando que já recebeu e processou os dados vindos do pacote.
- CMD_STOP – informa ao servidor para parar o envio de pacotes.
- CMD_RECV_SEQ – Indica que o servidor ira enviar vários arquivos seqüenciais.

- CMD_FINESHED – resposta da aplicação para o servidor informando que recebeu o ultimo pacote de um arquivo.

6.3.1.2 FUNCIONAMENTO DA APLICAÇÃO SFTD

A aplicação SFTD é ativada pela camada *mult*. Quando um pacote vem da rede e é direcionado para a aplicação, a SFTD irá receber o pacote e verificar para que tipo de comando o pacote destina-se.

A versão atual da aplicação está habilitada apenas para o recebimento de arquivos vindos do servidor. Como o microprocessador utilizado pela aplicação é o LPC2378 sem memória externa, teria muito pouco espaço para armazenamento de arquivos na memória dele. Para contornar esta limitação e validar o protocolo, foi alterada a forma como os arquivos são armazenados.

Quando um pacote chega da rede, a aplicação interpreta o cabeçalho, e os dados são direcionados diretamente para o *display* Nokia6100 conectado ao kit. Desta forma eles não são armazenados na memória do microprocessador.

Após o pacote com uma parte da imagem ser transferida para o *display*, o *buffer* dela é apagada para armazenamento do próximo pacote. Como a velocidade da troca de dados com a Internet é muito alta, o olho humano não consegue perceber os recebimentos dos pacotes separadamente, apenas a imagem num todo.

A aplicação foi desenvolvida para um funcionamento síncrono, após o recebimento do pacote, ela verifica o campo de M.F. procurando se existem mais pacotes daquele arquivo, se sim, ela envia uma mensagem de resposta informando ao servidor que aguarda o próximo pacote. Caso não tenha mais nenhum pacote e a imagem já tenha chegado ao fim, a aplicação envia uma mensagem de fim para o servidor, informando que todos os pacotes foram recebidos com sucesso.

Quando o comando de recebimento é do tipo *cmd_recv*, a aplicação ira interpretar o protocolo recebido e enviar o campo de dados diretamente para o *display*. Após todo o pacote de dados ser enviado, a aplicação envia uma

mensagem para o servidor informando-o para envio do próximo pacote, passando para o estado de “*wait_recv*” onde ela ira ficar aguardando o próximo pacote da imagem.

Quando o comando de recebimento de arquivos é do tipo *cmd_recv_seq*, significa que vários arquivos serão enviados para a aplicação. Este comando foi criado para receber frames de vídeos através da rede. Para finalizar o recebimento das imagens seqüenciais, basta pressionar o terceiro botão do kit. A aplicação ira interromper o recebimento e enviar um pedido de pare para o servidor.

Caso a aplicação receba um comando que não está habilitada para atender ela retorna um comando de erro para o servidor.

Quando a conexão com o servidor é finalizada, a aplicação libera a memória usada pela estrutura para tratamento dos pacotes e salta para o estado de fechado, aguardando uma nova conexão.

6.3.2 CARACTERISITICAS DO SERVIDOR SFT

Para atender a aplicação SFTD foi desenvolvido um servidor que se comunica usando o protocolo SFT. Este servidor foi projetado para ser usado em maquinas com o sistema operacional Linux e possui comandos para envio de arquivos para um sistema que interprete protocolos SFT.

O servidor se comunica com o *host* destino usando *socket* TCP. Depois de estabelecida a conexão entre um cliente e o servidor, é oferecida uma interface para o usuário interagir com o servidor e realizar as transferências de arquivos.

6.3.2.1 COMANDOS DO SERVIDOR SFT

O servidor SFT suporta os seguintes comandos:

- CMD_SEND – comando para envio de um arquivo.
- CMD_SEND_WAIT – comando interno que deve receber de resposta do cliente para enviar novo pacote.
- CMD_STOP – comando interno que deve receber do cliente para parar a transmissão de pacotes.
- CMD_SEND_SEQ – comando para envio de uma seqüência de pacotes.
- CMD_FINISHED – comanda interno que deve receber do cliente quando for enviado o ultimo pacote.
- CMD_ERROR – comando interno que deve ser recebido do cliente quando ocorrer algum tipo de erro.

6.3.2.2 FUNCIONAMENTO DO SERVIDOR SFT

Para iniciar o uso do servidor SFT no Linux é necessário passar por parâmetro na inicialização o endereço do cliente. Não é necessário nenhum comando no cliente. O servidor encontra-se junto com a documentação do projeto [10] no diretório *tools/SFT_tools* .

Os comandos aceitos no servidor são:

- **send** – envia um arquivo para o cliente.
- **sendseq** – envia um conjunto de arquivos seqüenciais.
- **quit** – encerra a conexão com o cliente e finaliza o servidor.

Exemplo de inicialização do servidor, supondo que o endereço do cliente seja 192.168.1.13:

./SFT 192.168.1.13

SFT Conectado com sucesso...

SFT>

Ao iniciar o servidor, é criado um identificador (ID) de mensagem com valor 0, a cada arquivo enviado este ID e incrementado, desta forma, o cliente pode controlar o recebimento de pacotes e saber que se refere a um mesmo arquivo.

Quando uma solicitação de envio de um arquivo é feita, o servidor busca por este arquivo no seu repositório para enviá-lo. Ao localizar, ele reparte os dados do arquivo em pacotes com 1000 caracteres cada. Assim, todos os pacotes recebem um tamanho padrão variando apenas o ultimo pacote que contem os restos dos caracteres do arquivo.

Para cada pacote do arquivo enviado, é inserido um número de pacote em ordem crescente. Desta forma, o cliente pode controlar o recebimento de pacotes para junta-los na ordem correta sem que pacotes sejam perdidos.

A *flag* MF (mais fragmentos) serve para informar ao cliente que existe mais pacotes daquele arquivo que está sendo enviado (MF = 1), ou informar que está sendo enviado o ultimo pacote do arquivo (MF = 0).

Após todos os campos do protocolo ser preenchidos o pacote é enviado para o cliente. O servidor fica no aguarda de uma confirmação para poder enviar o próximo pacote. Caso o cliente informe que houve erro no recebimento do pacote, pois o número está descasado com o pacote recebido anteriormente, o servidor faz o reenvio do mesmo pacote. Ao receber a mensagem de confirmação, é enviado o próximo pacote, repetindo até o fim do arquivo.

6.3.2.3 SENDSEQ – ENVIO DE ARQUIVOS SEQUENCIAIS

O servidor foi desenvolvido para enviar arquivo de imagem para o cliente. Para o envio de uma única imagem basta usar o comando *send*. Para enviar uma animação com imagens seqüenciais é necessário enviar várias imagens em seqüência, o comando *sendseq* foi desenvolvido com este propósito.

Quando é feita uma solicitação de arquivos seqüenciais, o servidor fica constantemente enviando os arquivos. Para criar uma seqüência de arquivos para ser enviado é necessário os seguintes passos:

- Deve ser passado o nome do diretório onde estão os arquivos;
- Os arquivos devem estar nomeados em seqüência (arquivo 1, 2, 3, ...n).
- Deve existir um arquivo com o nome: <nome_diretorio>.conf. Neste arquivo deve ter as seguintes informações:
    - **Tempo –** informa quanto tempo deve ser aguardado para envio de outro arquivo.
    - **Quantidade de imagens –** informa quantas imagens existem no diretório.
    - **Tipo –** informa qual o tipo de seqüência esta sendo enviado, podendo ser:
        - 1 – para envio de uma seqüência de “apresentação” – desta forma o tempo de troca das imagens será medido em segundos.
        - 2 – para envio de uma seqüência de “animação” – desta forma o tempo de troca das imagens será medido em micro-segundos.
    - O arquivo ficara no formato:

        <tempo>, <quantidade de imagens>, <tipo>

Suponha que exista um conjunto de 120 imagens de uma seqüência para uma animação com o nome “dance” e que a troca de frames das imagens deva ser em 3 microssegundos. O arquivo de configuração para este conjunto ficaria:

3, 120, 2 >> dance.conf

Após ler as configurações o servidor entra em um laço infinito enviando constantemente as imagens. Quando o envio é de uma animação, como a troca de pacotes é muito rápida, não é possível perceber que se trata de imagens isoladas seqüenciais.

Para que possa ser finalizado o envio dos arquivos é necessário que o cliente envie uma mensagem de finalização.

É importante observar que o envio de uma animação gera um alto trafego de pacotes. O uIP aceita só uma conexão TCP por vez, podendo prejudicar os outros serviços que estejam sendo usados.

6.3.2.3.1 CRIANDO IMAGENS PARA ENVIAR PELO SERVIDOR

As imagens enviadas para o cliente serão encaminhadas diretamente para o *display* da Nokia. Desta forma, o arquivo com a imagem deve conter apenas os *pixels* da imagem.

As imagens enviadas devem ser do tipo bitmap com 132x132 de tamanho. Para convertê-las no formato aceito pelo *display* da Nokia6100 foi utilizado o programa *bmp2* desenvolvido por Jimmy Stelzer[5] e alterado para criar automaticamente uma seqüência de imagens e deixá-las no formato usado pelo comando *sendseq.*

O programa bmp2, bem como o arquivo *readme* com as explicações de como utilizá-los, encontram-se junto com os arquivos de implementação do projeto[10].

6.3.3 DESCRIÇÃO DO USO DO SERVIDOR / CLIENTE SFT

O servidor é inicializado no Linux, solicitando uma conexão com o endereço do cliente passado por parâmetro. Depois de conectado, o servidor abre uma interface de linha de comando para o usuário interagir com o cliente, enviando os comandos aceitos pelo servidor (seção 6.3.2.2).

Quando é feito o envio de um arquivo, após ser finalizado com sucesso, o servidor fica disponível para envio de um novo arquivo. Ao solicitar o envio de arquivos seqüenciais, a interrupção do serviço deve ser feita pelo cliente com uma mensagem de finalização.

[5] Jimmy Stelzer – Estudante de engenharia de computação na PUCRS.

Quando o cliente deseja interromper um serviço, basta clicar o botão 3 ao lado do *display* presente no kit. Este é o mecanismo abordado para a interrupção de imagens seqüenciais enviadas para o *display*. Ao ser pressionado é enviado uma mensagem finalizadora de serviço para o servidor.

Como não existe uma interface de comandos por parte do cliente, para sair da aplicação basta finalizar o serviço junto ao servidor.

6.4 CLI – INTERFACE DE LINHA DE COMANDO

A comunicação por linha de comando CLI (*Command Line Interface*) foi desenvolvida para comunicar-se através de uma interface serial. O objetivo é poder interagir com as configurações do sistema e alterar parâmetros que não são possíveis de se realizar através da comunicação Ethernet.

Quando o sistema é iniciado a CLI é chamada para interação com o usuário. Para verificar quais comandos estão disponíveis na CLI basta digitar o comando de ***help***, será listado todas as funções existentes e como utilizá-las.

As funcionalidades presentes na cli podem ser acessadas através de comandos de ***get,*** para que seja apresentado o resultado de uma função, ou por comandos de ***set***, para alterar uma aplicação.

6.4.1 COMANDOS DE CONFIGURAÇÃO DO uIP PELA CLI

O uIP foi configurado para ter endereços de rede fixo. Desta forma, os endereços de IP, *gateway* e mascara, são fixados em tempo de projeto. Para que estes endereços possam ser alterados na aplicação é necessário o uso da CLI.

A CLI permite ler estes endereços através dos comandos:

- GET IP – retorna o endereço de IP da aplicação.
- GET GATEWAY – retorna o endereço de *gateway* da aplicação.
- GET MASCARA - retorna a mascara da aplicação.

Para alterar estes endereços os comandos são:

- SET IP <endereço> - altera o endereço de IP da aplicação.
    - exemplo: set ip 192.168.1.12
- SET GATEWAY <endereço> - altera o endereço de gateway da aplicação.
    - exemplo: set gateway 192.168.1.1
- SET MASCARA <endereço> - altera a mascara da aplicação.
    - Exemplo: set mascara 255.255.0.0

Outro comando útil do uIP, e que pode ser acessado pela CLI é o endereço de MAC.

- GET MAC – retorna o valor de MAC do kit de desenvolvimento.

Este comando só pode ser lido. Não é possível alterar o endereço de MAC do equipamento.

6.4.2 INSERINDO FUNCIONALIDADES NA CLI

Uma funcionalidade na CLI é formada por uma estrutura que aponta para a função responsável em tratar o comando solicitado, como mostra a Tabela 9.

COMANDO	FUNCIONALIDADE
Str_t* tp	Aponta para a estrutura tratadora do comando. A estrutura deve ser do tipo str_t.
String	Nome da função a ser chamada pela CLI.
Help	Mensagem para orientação de como tratar o comando.

Tabela 9 – Estrutura cli_str - Responsável em tratar os comandos da CLI

Quando uma nova funcionalidade é criada, ela deve ser inserida na lista de comandos da CLI, presentes no arquivo *menu_cli.h*. A função tratadora do comando é do tipo *str_t* que serve para tratar e armazenar a resposta da CLI para a funcionalidade.

Comando	Funcionalidade
tipo	Define o tipo de função, podendo ser: GET ou SET.
Cmd_name	Nome da função tratadora do comando.
value	Contém as informações a serem usadas pelo comando de set.
resp	Contém a mensagem de resposta da função tratadora.

Tabela 10 – Estrutura str_t - Responsável em tratar as funcionalidades da CLI.

Após o comando ser inserido, deve-se criar a função tratadora do comando no arquivo *cli_func.c*.

Os arquivos de projetos foram documentados e estão disponíveis no endereço: **http://marcotuliogm.github.com/mult-UIP/docs/html/index.html**[10]**.**

7 APLICAÇÕES DO uIP

Foram adaptadas duas aplicações já existentes no projeto raiz da pilha uIP para as novas funcionalidades criadas. Sendo, o serviço de TELNET e o serviço de *WebServer*.

7.1 SERVIÇOS DE TELNET

O serviço de TELNET permite que um cliente possa ter acesso remoto a um terminal de comandos, o projeto para sistemas embarcados foi criado por Dunkels [1, p. 116]. Sua aplicação original possui um *shell* de comunicação bastante simples, contendo apenas uma ilustração de sua funcionalidade e validação para atender os requisitos das RFCs 854-861 que normalizam o serviço de TELNET.

Este serviço foi inserido no projeto com múltiplas aplicações, passando a ser monitorado através da sub-camada *mult*. A porta utilizada pelo serviço de TELNET é a 23 e é inicializada pela função *mult_init*.

O *shell* exemplo utilizado pelo TELNET foi removido da aplicação inserida neste projeto. Em seu local foi colocada a saída da CLI apresentada na seção 6.4. Desta forma a interface do serviço é a mesma do usuário da porta serial.

É importante observar que o cliente ao ter acesso a CLI pelo serviço de TELNET, passa a ter direito a troca de IP, Gateway e mascara, porém uma vez que um destes endereços é alterado o cliente perde a conexão com o uIP pelo endereço que ele foi conectado, necessitando uma nova conexão pelo novo endereço.

O uIP não envia pacotes fragmentados, logo o envio de informações para o serviço de TELNET não pode passar do tamanho da MTU do projeto. Comandos como HELP não podem ser acessados por este serviço, uma vez que a resposta deste comando é bastante grande.

7.2 SERVIÇOS DE WEBSERVER

A aplicação WebServer permite utilizar a pilha uIP como um servidor Web podendo ser acessado por um usuário externo. O HTTP foi desenvolvido para sistemas embarcados por Dunkels [1, p. 120]. Junto ao serviço foi desenvolvido um exemplo com uma pagina Web fazendo acesso aos serviços e APIs do uIP.

O *Webserver* utiliza as APIs de *protosockets* e *protothread*s na sua implementação. Para cada nova funcionalidade adicionada no serviço é criada uma nova *protothread*.

O serviço de *WebServer* foi adicionado no projeto de múltiplas aplicações. A inicialização e gerencia dos pacotes destinados a ela é feita através da sub-camada *mult*. A porta 80 é utilizada para monitorar os serviços do *WebServer* sendo inicializada pela função *mult_init.*

A pagina Web desenvolvida para aplicação oferece recursos para controle do kit, podendo ser alterada pelo cliente que estiver acessando. Para adicionar funcionalidades no WebServer foi seguido o tutorial escrito por Felipe Lavratti[12].

Opções inseridas na pagina da aplicação:

- Opção de configuração do relógio do Kit de desenvolvimento, através do RTC do LPC2378,
- Configuração da mensagem do LCD 16x2,
- Verificação da temperatura ambiente.

Figura 12 – Pagina Web presente no uIP com múltiplas Aplicações.

A Figura 12 mostra como ficou a pagina Web presente no Kit. Para acessar a pagina disponível basta abrir o *browser* do *host* conectado na mesma rede do equipamento e acessar o endereço IP da aplicação.

8 CONCLUSÃO

O uIP é uma implementação simplificada do modelo TCP/IP, abordando apenas os protocolos essenciais para a comunicação com a Internet. Ele foi desenvolvido especificamente para ser usado em pequenos sistemas embarcados com recursos de memória limitada.

Este trabalho procurou suprir umas das limitações existentes na versão atual do uIP, criando uma camada para tratar os serviços vindos da camada de transporte para a aplicação, sendo possível ter múltiplos serviços em execução simultaneamente.

Foi desenvolvido um *chat* de comunicação utilizando *sockets* TCP e *protosockets*, para validar os testes com múltiplos serviços. Além deste, foi criado o protocolo SFT para realizar a transferência de arquivos entre um servidor com a pilha TCP/IP padrão e um cliente utilizando a pilha uIP.

As novas funcionalidades do uIP foram testadas e validadas em um microcontrolador embarcado com arquitetura ARM7 tipo LPC2378 da NXP. Para testar o protocolo de transferência de arquivos foram usadas imagens de mapas de bits apresentadas em um display gráfico tipo Nokia 6100 conectado à interface SPI do microcontrolador. A transferência foi suficientemente rápida para exibir animações em tempo real. Simultaneamente foi possível estabelecer comunicações com a aplicação *chat*, *webserver* e *telnet*.

Todas as novas funcionalidades do uIP, o uso das APIs e das funções principais do uIP foram documentadas auxiliando para a criação de novos projetos e adição de funcionalidades ao sistema.

9 REFERÊNCIAS

1. DUNKELS, A. The uIP Embedded TCP/IP Stack. **Swedish Institute of Computer Science**, 2006. Disponivel em: <http://www.sics.se/~adam/uip/index.php/Main_Page>. Acesso em: 20 Fevereiro 2011.
2. DUNKELS, A. **Full TCP/IP for 8-Bit Architectures**. Swedish Institute of Computer Science. [S.l.], p. 14. 2007.
3. CARLOSE.MORIMOTO. Entendendo os Sistemas Embarcados. **Guia do Hardware**, 2007. Disponivel em: <http://www.hardware.com.br/artigos/entendendo-sistemas-embarcados/>. Acesso em: 20 mar. 2011.
4. STEMMER, M. A. Lab. Processadores: Programação do LPC2378 usando o gcc. **Laboratório de processadores**, 2011. Disponivel em: <http://www.ee.pucrs.br/~stemmer/labproc/apostila2011/lpc-01.html>. Acesso em: 10 maio 2011.
5. COMER, D. E. **Interligação em Rede com TCP/IP**. Rio de Janeiro: tradução de ARX Publicações, v. 1, 1998.
6. BRADEN, R. Requirements for Internet Hosts -- Communication Layers. **Internet Engineering Task Force**, 1989. Disponivel em: <http://www.ietf.org/rfc/rfc1122.txt>. Acesso em: 17 maio 2011.
7. BARRY, R. FreeRTOS Web Server Demo. **FreeRTOS-A Free professional grade RTOS**, 2011. Disponivel em: <http://www.freertos.org/index.html?http://www.freertos.org/portlpc2368_Eclipse.html>. Acesso em: 06 Abril 2011.
8. PERALTA, K. **Camada de Rede e Protocolo IP**. PUCRS. Porto Alegre, p. 16. 2011.
9. POSTEL, J. RFC 793 - Transmission Control Protocol. **Internet FAQ Archives**, 1981. Disponivel em: <http://www.faqs.org/rfcs/rfc793.html>. Acesso em: 21 maio 2011.
10. MARTINS, M. T. G. Mult-UIP: Multiplas Aplicações com o UIP. **GitHub**, 2011. Disponivel em: <http://marcotuliogm.github.com/mult-UIP/docs/html/index.html>. Acesso em: 15 junho 2011.
11. DUNKELS, A. Protosockets library. **uIP 1.0**, 2006. Disponivel em: <http://www.sics.se/~adam/uip/uip-1.0-refman/a00158.html>. Acesso em: 20 maio 2011.
12. LAVRATTI, F. D. A. N. uIP + Webserver – Como customizar o Webserver-uIP-Demo para a sua aplicação. **Sistemas Embarcados Livre**, 2011. Disponivel em: <http://selivre.wordpress.com/2011/01/26/uip-webserver-como-customizar-o-webserver-uip-demo-para-a-sua-aplicacao/>. Acesso em: 03 fevereiro 2011.
13. SEMICONDUCTORS, N. **LPC2378 - Preliminary data sheet**. NXP. [S.l.], p. 53. 2007.

14. SEMICONDUCTORS, N. NXP Semiconductors - Microcontrollers [Products - LPC23xx Series Device Highlight - LPC2378]. **NXP Semiconductors - Microcontrollers**, 2011. Disponivel em: <http://ics.nxp.com/products/lpc2000/lpc23xx/~LPC2378/>. Acesso em: 17 maio 2011.

15. COOLER. Linux: BSD Sockets em linguagem C [Artigo]. **Viva o Linux**, 2010. Disponivel em: <http://www.vivaolinux.com.br/artigo/BSD-Sockets-em-linguagem-C?pagina=3>. Acesso em: 18 maio 2011.

16. WIKIPÉDIA, C. D. MTU. **Wikipédia, a enciclopédia livre.**, 2010. ISSN 18643546. Disponivel em: <http://pt.wikipedia.org/w/index.php?title=MTU&oldid=18643546>. Acesso em: 02 junho 2011.

17. LTDA, C. T. **eCOG1X FTP Server with USB File Storage**. Cambridge: [s.n.], 2008. ISBN AN068.

18. FELIPE. Webserver-uIP Demo. **SELivre**, 2011. Disponivel em: <http://selivre.googlecode.com/files/Webserver-uIP-Demo-fixed.tar.gz>. Acesso em: 20 Janeiro 2011.