UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JEAN CARLO HAMERSKI

Desenvolvimento de uma Arquitetura Parametrizável para Processamento da Pilha TCP/IP em Hardware

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profa. Dra. Fernanda Lima Kastensmidt
Orientadora

Porto Alegre, julho de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hamerski, Jean Carlo

Desenvolvimento de uma Arquitetura Parametrizável para Processamento da Pilha TCP/IP em Hardware / Jean Carlo Hamerski. – Porto Alegre: PPGC da UFRGS, 2008.

80 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Orientadora: Fernanda Lima Kastensmidt.

1. TCP/IP Offload Engine. 2. Throughput. 3. Desempenho TCP/IP. I. Kastensmidt, Fernanda Lima. II. Título.

“Para ganhar conhecimento, adicione coisas todos os dias. Para ganhar sabedoria, elimine coisas todos os dias.”

LAO TSÉ

AGRADECIMENTOS

A todos os meus parentes, principalmente aos meus pais, Jorge e Arlete Hamerski, pelo encorajamento e apoio, que tornam o fruto do meu esforço ainda mais saboroso.

Aos meus amigos, pelo apoio e companheirismo.

À minha companheira, Paula Pedone, pela compreensão e comprometimento.

À minha orientadora, Fernanda Lima Kastensmidt, pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

A todos os professores do Programa de Pós-Graduação em Computação pelos seus ensinamentos e aos funcionários, que, durante esses anos, contribuíram de algum modo para o meu enriquecimento pessoal e profissional.

A todos os colegas de estudo que de alguma forma fizeram parte dessa empreitada, em especial ao colega Chris Dennis Tomás Horna pela cessão do MAC 10/100Mb (Media Access Control - 10/100 Megabit), utilizado nesse trabalho.

Aos órgãos de apoio à pesquisa, em especial ao Conselho Nacional de Desenvolvimento Científico e Tecnológico e à Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul, pelo excelente auxílio que oferecem àqueles que buscam, através do seu trabalho, agregar conhecimento à nação brasileira.

A Deus, a grande força que me encoraja cada vez mais a superar meus limites mantendo a dignidade e o respeito ao próximo na dianteira de meus passos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction Set Processors
CPU	Central Processing Unit
CRC	Cycle Redundancy Check
Gbps	Gigabits por segundo
ICMP	Internet Control Message Protocol
I/O	Input/Output
IP	Internet Protocol
iSCSI	Internet Small Computer System Interface
ISO	International Standards Organization
FIFO	First In, First Out
LANS	Local Area Network
MAC	Media Access Control
Mbps	Megabits por segundo
MSS	Maximum Segment Size
NAS	Network Attached Storage
NIC	Network Interface Card
OSI	Open Systems Interconnection
PDU	Packet Data Unit
RISC	Reduced Instruction Set Computer
RTT	Round Trip Time
SAN	Storage Area Network
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TOEs	TCP/IP Offload Engine

TTL	TTL - Time to Live
UDP	User Datagram Protocol
VHDL	VHSIC Hardware Description Language
WANS	Wide Area Network

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O aumento da popularidade da Internet e a criação de novos meios de transmissão estimulam um explosivo crescimento da taxa de transmissão de dados sobre a Internet. Assim, o processamento TCP/IP baseado em software torna-se um gargalo por não processar os pacotes na velocidade das linhas de transmissão, em especial os pacotes da camada de transporte. Conseqüentemente, surge a necessidade de implementação em hardware do processamento TCP/IP, o que traria vantagens como aceleração do processamento do fluxo de dados.

Neste sentido, este trabalho apresenta a arquitetura do iNetCore, descrita em VHDL, para processamento dos protocolos das camadas de rede e transporte em hardware. Duas implementações desta arquitetura foram elaboradas, buscando explorar o espaço de projeto e analisar os resultados obtidos na síntese para a tecnologia ASIC e FPGA, e o desempenho no processamento de pacotes.

Uma arquitetura HW/SW contendo o iNetCore foi prototipada sobre a placa Virtex-II Pro Development System. Em conjunto com essa arquitetura, foi implementada uma interface de comunicação com o barramento OPB, tornando possível a implementação de softwares da camada de aplicação que queiram usar a pilha TCP/IP desenvolvida em hardware.

Por fim, foram efetuados experimentos para avaliar o desempenho da arquitetura HW/SW no processamento de segmentos TCP. A arquitetura HW/SW em conjunto com o iNetCore alcançou um throughput de até 1,45 Gbps, possibilitando o uso da arquitetura para processamento de pacotes TCP/IP na plenitude de banda disponíveis em redes gigabit.

Palavras-chave: TCP/IP Offload Engine, throughput, desempenho TCP/IP.

Development of a Customizable Architecture to TCP/IP Stack Processing in Hardware

ABSTRACT

The advent of new transmission lines stimulates an explosive increase of the Internet data-transmission rate. Thus, the TCP/IP processing based on software became a bottleneck, because it cannot reach the transmission line speed required, specially in the transmission of transport layer packets. This limitation brings the necessity of implementation of the TCP/IP processing in hardware, what it would bring advantages in the acceleration of data flow processing.

In this way, this work presents the iNetCore architecture, described in VHDL, able to process the transport and network layers protocols in hardware. Two implementations of this architecture were implemented. The objective is to explore the design space and to analyze the results in ASIC and FPGA technology synthesis. Also, a simulation environment was built to analyze the performance in the packets computation.

A HW/SW architecture containing the iNetcore was prototyped on Virtex-II Pro Development System board. In conjunction with this architecture, it was implemented a communication interface with OPB bus, which makes possible the development of application layer softwares that may use the hardware TCP/IP stack developed.

Finally, experiments were realized in order to evaluate the HW/SW architecture performance in the TCP segments processing. The HW/SW architecture together with the iNetCore reached a throughput of about 1.45 Gbps in the TCP/IP packets processing. It proves its potential to use available bandwidth in gigabit networks.

Palavras-chave: TCP/IP Offload Engine, throughput, TCP/IP performance.

1 INTRODUÇÃO

Com o avanço da tecnologia de semicondutores, a velocidade nas linhas de transmissão Ethernet tem aumentado de 3Mbps para 10Gbps, assim como a capacidade de processamento dos processadores também tem aumentado na mesma proporção. Porém, a capacidade de processamento da pilha de protocolos TCP/IP por parte desses processadores não cresceu na mesma taxa (WANG et al, 2005).

Pesquisas indicam que o maior overhead no processamento TCP/IP é causado pelo gerenciamento de I/O e de *buffers* nos sistemas operacionais (CLARK, 1989). Quando é necessária uma alta taxa de transmissão, e conseqüente processamento, essas tarefas de I/O podem requerer um tempo de uso da Unidade de Processamento Central (CPU) muito grande para processar os dados dos pacotes TCP/IP. Com isso, outras aplicações que estejam rodando sobre a mesma plataforma poderiam não dispor do tempo de CPU necessário para rodar suas tarefas (MARKATOS, 2002).

Neste sentido, o tempo de processamento da pilha de protocolos TCP/IP vem sendo largamente discutido nos últimos anos. Entre as abordagens propostas, algumas apontam no sentido de otimizar o desempenho TCP para protocolos da camada de aplicação. Outra proposta é otimizar o software de processamento TCP/IP. A última abordagem se baseia no processamento "offload"da pilha de protocolos TCP/IP por um hardware dedicado. Este hardware dedicado pode ser, por exemplo, um dispositivo de rede inteligente contendo em sua arquitetura um ASIC específico para processamento da pilha TCP/IP, ou ainda um processador com um conjunto de instruções específico para esse processamento, no caso, um ASIP. Cada solução possui suas vantagens e desvantagens que serão discutidas posteriormente.

O processamento "offload"da pilha TCP/IP por um hardware dedicado visa dispensar a CPU do sistema da tarefa de processar os protocolos de comunicação, passando essa tarefa para a placa de rede (NIC - Network Interface Card, em inglês). Deste modo, a placa de rede pode interagir com a rede sem a participação da CPU, diminuindo custos como processamento de interrupções, cópias de memória e outras tarefas do modelo tradicional de processamento em software. Porém, para a placa de rede poder realizar a tarefa de processamento da pilha TCP/IP, faz-se necessário um hardware específico, chamado de TCP/IP Offload Engine.

No decorrer da dissertação é apresentada uma visão geral da técnica denominada "TCP/IP Offload", seguido da planificação das possíveis implementações dessa técnica. Após, são apresentadas algumas arquiteturas ASIP e ASIC existentes para processamento da pilha TCP/IP. Em seguida, é apresentado o *iNetCore*, a arquitetura modular parametrizável desenvolvida para realizar o processamento da pilha TCP/IP, com todos os detalhes específicos dessa implementação, explicitando também toda a interface com as camadas de nível superior e inferior do modelo TCP/IP. O objetivo deste tipo de abor-

dagem é superar as limitações do processamento tradicional TCP/IP em software e minimizar os recursos de sistema necessários para este processamento.

O protocolo TCP foi criado para garantir uma transmissão confiável de pacotes sobre o protocolo IP. Como o protocolo IP pode reordenar ou descartar alguns pacotes entre o transmissor/receptor de um pacote, é o protocolo TCP que garante a entrega confiável desses pacotes ao receptor de maneira ordenada. Em virtude desse controle na ordenação do pacote, controle de retransmissões, entre outras funcionalidades, o protocolo TCP é o mais complexo da pilha TCP/IP e sua implementação em uma determinada plataforma dependerá dos recursos que essa dispõe e quais serão os propósitos para o uso do TCP como protocolo de transporte de dados.

Tradicionalmente, a pilha TCP/IP requer uma grande quantidade de recursos em termos de tamanho de código e uso de memória, quando a sua implementação é efetuada em software (DUNKELS, 2007). Quando se fala em implementação da pilha TCP/IP em hardware, deve-se levar em consideração todas essas questões e, por se tratar de uma plataforma que, em geral, apresenta poucos recursos em relação à quantidade de memória e elementos lógicos, a escolha sobre quais funcionalidades devem ser implementadas em hardware torna-se decisiva na caracterização de uma arquitetura que seja, ao mesmo tempo, compacta e eficiente.

Em virtude disso, definiu-se uma metodologia de trabalho que buscasse identificar uma implementação da pilha TCP/IP em software que apresentasse um conjunto mínimo de funcionalidades, mas suficientes para uma completa implementação em relação aos serviços que fossem utilizá-la para comunicação com a Internet.

A implementação escolhida para estudo foi a pilha uIP (DUNKELS, 2007), uma arquitetura de 8 bits em software que apresenta os protocolos IP, ICMP, TCP e ARP, e características de pouco uso de memória e completa gama de funções para comunicação com a maioria dos serviços que utilizam a pilha TCP/IP, tais como: servidores/clientes dhcp, web, smtp e telnet. Essa implementação é considerada a menor do mundo em relação ao tamanho de código e uso de memória, e por isso, é aconselhada para sistemas que apresentam poucos recursos de memória e não requerem um desempenho muito alto em relação à taxa de processamento dos dados.

Por apresentar todas as funcionalidades necessárias para uma completa implementação de serviços de rede sobre a pilha TCP/IP, a implementação uIP mostrou-se muito interessante para ser mapeada e ter suas funcionalidades implementadas em hardware. Porém, o estudo efetuado sobre o código do uIP, desenvolvido em linguagem C e implementado buscando uma alta compactação, revelou uma baixa modularização em relação às funções implementadas em virtude da utilização massiva da diretiva "goto". Por esse motivo, o uIP serviu apenas como referência em relação a quais funcionalidades deveriam ser implementadas, entre as quais podem ser citadas:

- *IP* e *TCP checksum*, que garante a integridade dos dados que compõem o pacote;
- *IP fragment reassembly*, que remonta o pacote para ser entregue às camadas de transporte e aplicação;
- *Multiple TCP connections*, possibilitando a execução simultânea de vários serviços TCP;
- *TCP Options*, permitindo a caracterização dos segmentos TCP para serviços especiais;

- *Variable TCP MSS*, permitindo o controle do tamanho dos segmentos TCP em razão de características da transmissão;

- *RTT Estimation*, que controla o timer para retransmissão de um segmento em caso de não recebimento de resposta (ack) ou perda de segmento;

- *TCP flow control*, que controla o tamanho do pacote, não permitindo o envio de um segmento maior do que o receptor possa armazenar;

- *TCP urgent data*, provê um mecanismo de notificação entre aplicações remotas, que podem setar um segmento como sendo mais urgente que um segmento normal.

Outras funcionalidades também foram implementadas e serão detalhadas no decorrer da dissertação.

A metodologia de trabalho contém a implementação do algoritmo da pilha TCP/IP em hardware sob diversos aspectos quanto à composição da arquitetura do módulo funcional a ser desenvolvido: tamanho da palavra (16 e 32 bits), módulos em hardware/software e modularização interna da arquitetura (seu impacto).

As diversas descrições do mesmo módulo funcional foram fornecidas como entrada para ferramentas de síntese ASIC e VHDL. Nestas ferramentas, diversas diretivas de síntese foram utilizadas de forma a explorar o espaço de projeto (menor área, maior desempenho, etc). Assim, as descrições foram sintetizadas usando essas diferentes diretivas e os resultados foram avaliados em relação à área e desempenho.

Ao final da etapa de avaliação das sínteses efetuadas sobre as diversas implementações do algoritmo, foram analisados aspectos relativos ao custo, desempenho e eficiência das arquiteturas. Essa análise foi feita no intuito de definir uma arquitetura que possuísse características de:

- *configurabilidade*, através da configuração de parâmetros a respeito dos requisitos que se deseja alcançar (velocidade de processamento, por exemplo);

- *flexibilidade*, levando-se em conta a característica modular da arquitetura final, possibilitando assim, por exemplo, a adição de novos módulos que poderiam processar outros tipos de pacotes e remoção de módulos que não interessam no caso de uso da pilha TCP/IP para uma aplicação específica.

Para o desenvolvimento do projeto foram utilizadas ferramentas de síntese ASIC e VHDL (Mentor Leonardo Spectrum, Xilinx ISE), uma ferramenta de projeto de sistemas com suporte ao co-design hardware/software (Xilinx Plataform Studio) e uma ferramenta de depuração de protótipos em FPGA (Chipscope).

A dissertação apresenta, no capítulo 2, o Modelo TCP/IP usado para implementação da arquitetura proposta no trabalho. Neste capítulo são detalhados os protocolos da pilha TCP/IP, em especial as características do protocolo TCP e uma discussão sobre a viabilidade da implementação das principais funcionalidades da pilha TCP/IP.

No capítulo 3 é apresentada a técnica "TCP/IP Offload", que consiste na idéia de processar todo o fluxo de pacotes da pilha TCP/IP em um hardware específico. Além disso, são discutidas as principais causas do overhead de processamento da pilha TCP/IP e como solucionar os possíveis gargalos desse processamento.

O capítulo 4 mostra algumas arquiteturas existentes que implementam a técnica "TCP/IP Offload"em ASIP ou ASIC.

No capítulo 5 é apresentado todo o desenvolvimento, decisões de projeto e implementação do iNetCore e ainda os experimentos efetuados e resultados de síntese da arquitetura do iNetCore em tecnologia ASIC e FPGA.

Já no capítulo 6 é relatado o desenvolvimento da interface de comunicação entre o iNetCore e a camada de aplicação da pilha TCP/IP, detalhando o ambiente de integração do iNetCore com uma arquitetura hardware/software contendo diversos elementos presentes em um sistema de computação tradicional, tal como processador, barramento de memória, bloco de memória, entre outros. Também é relatado um experimento para avaliar o throughput alcançado pelo iNetCore no processamento de pacotes TCP/IP.

Por fim, o capítulo 7 apresenta as considerações finais deste trabalho, dificuldades encontradas, principais contribuições do trabalho desenvolvido e questões que ficaram em aberto no desenvolvimento do iNetCore que podem ser exploradas em trabalhos futuros.

2 O MODELO TCP/IP

Surgida nos anos 60 como uma iniciativa conjunta entre o departamento de defesa americano, universidades e fabricantes de sistemas computacionais, para ser um meio de comunicação robusto a eventuais quedas de nodos, e independente das várias plataformas então existentes, a rede ARPANET (assim chamada devido ao grupo responsabilizado pela sua criação, o *Advanced Research Projects Agency*) introduziu o modelo de camadas de protocolos, o qual foi aprimorado nos anos 70 pela Universidade da Califórnia e serve como base para o atual TCP/IP. Este modelo, embora extraído de sua implementação, é a base da Internet tal como ela é conhecida nos dias de hoje. Tal situação resulta da derivação histórica desta rede a partir daquela implementada pela ARPA (Advanced Research Projects Agency), a qual teve êxito justamente por poder ser facilmente portada entre as plataformas integrantes do mecanismo de comunicação.

Embora TCP e IP dêem nomes a protocolos, o modelo TCP/IP não se refere especificamente ao seu emprego na implementação da pilha de camadas. Neste modelo, entre as aplicações e o meio físico existem dois níveis, designados como nível de transporte (entre os quais se enquadra o protocolo TCP) e nível de rede (do protocolo IP). É bom frisar que embora existam versões do modelo reduzidas, que descartem uma ou outra funcionalidade, o modelo original não pode ser caracterizado sem a presença de outros protocolos, como o UDP, o ARP e o ICMP, alguns deles com atuações intermediárias entre os níveis. O protocolo ARP, por exemplo, prevê uma tradução entre o endereçamento usado no nível de rede para referenciar outras máquinas para aquele efetivamente compreendido pelo meio físico. O modelo TCP/IP, por definição, não se preocupa com os detalhes concernentes à comunicação física dos seus agentes.

2.1 Modelo OSI x Modelo TCP/IP

O modelo de camadas foi assumido pela *International Standards Organization* (ISO), que definiu um padrão teórico baseado em sete camadas bem delineadas e especializadas (conforme Tabela 2.1). Este modelo, chamado OSI, tem servido apenas como referência para melhor compreensão dos processos de comunicações nas redes. A idéia principal por trás de tal tipo de modelo é a presença de uma implementação individual deste em cada um dos elementos da rede. Camadas iguais comunicam-se entre si de um agente para a outro. Dentro da mesma entidade computacional, as camadas comunicam-se com as adjacentes. Para uma informação que está sendo enviada, a cada nova transição de nível, que estará se dando para baixo, considerando uma ordem de acordo com os níveis de abstração, um cabeçalho é inserido, de modo que a camada equivalente na outra entidade possa interpretar corretamente do que se trata e o que fazer com o dado que está recebendo. No recebimento, portanto, após o tratamento, esses cabeçalhos são descartados

e então passados para o nível superior, até chegarem na forma de dados compreensíveis pela aplicação receptora. À medida que vão sendo acrescidos de informações, os dados recebem denominações distintas. Diz-se que a camada de aplicação interpreta mensagens, a de transporte, segmentos, enquanto a camada de rede lida com datagramas e a de enlace com quadros, até chegar ao nível físico onde somente bits são manipulados.

Tabela 2.1: Camadas do modelo OSI

7	Aplicação	Esta camada funciona como uma interface de ligação entre os processos de comunicação de rede e as aplicações utilizadas pelo usuário.
6	Apresentação	Aqui os dados são convertidos e garantidos em um formato universal.
5	Sessão	Estabelece e encerra os enlaces de comunicação.
4	Transporte	Efetua os processos de seqüenciamento e, em alguns casos, confirmação de recebimento dos pacotes de dados.
3	Rede	O roteamento dos dados através da rede é implementado aqui.
2	Enlace	Aqui a informação é formatada em quadros (frames). Um quadro representa a exata estrutura dos dados fisicamente transmitidos através do fio ou outro meio.
1	Física	Define a conexão física entre o sistema computacional e a rede. Especifica o conector, a pinagem, níveis de tensão, dimensões físicas, características mecânicas e elétricas, etc.

O modelo TCP/IP, anterior ao modelo OSI, possui a característica de não oferecer a mesma rigidez para o delineamento de camadas. Por isso, muitas vezes sua descrição é feita de acordo com uma outra abordagem que enfatiza as relações entre os protocolos que o compõem (Figura 2.1(a)). No entanto, por não ser um modelo tão abrangente como o modelo OSI, que descreve padrões do início ao fim do processo de comunicação, o modelo TCP/IP apresenta uma maior flexibilidade, admitindo que existam aplicações e dispositivos de rede, e que estes precisam se comunicar de alguma maneira. Para isso, o modelo propõe dois módulos: um cuja função principal seja começar e terminar uma conexão, e ainda controlar o fluxo de dados e de efetuar processos de correção e verificação de erros, de maneira equivalente à camada de transporte do modelo OSI; e um segundo módulo responsável pelo roteamento da informação para a rede correta, usado ainda para atribuir endereço de rede (IP) ao sistema e ligar as camadas superiores aos protocolos de hardware, de maneira equivalente às camadas de rede e, em algumas funcionalidades, enlace do modelo OSI. Dessa maneira, uma estrutura em quatro camadas para o TCP/IP pode ser aproximada de acordo com a Figura 2.1(b).

2.2 Comunicação entre as camadas do Modelo Funcional TCP/IP

Em um ambiente de comunicação pela rede via protocolo TCP/IP, os dados originados da camada de aplicação são transmitidos no *host* de origem por meio das diferentes camadas que compõem o modelo funcional TCP/IP. No destino os dados são recebidos através das mesmas camadas, e entregues à camada de aplicação. Durante este procedimento, tanto no *host* origem como no *host* destino, os dados são processados em cada uma

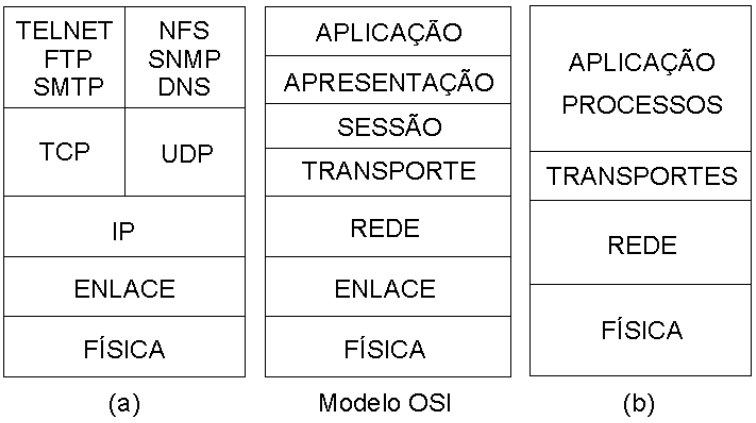

Figura 2.1: Perspectivas para o modelo TCP/IP em relação ao modelo OSI: (a) funcional e (b) em camadas

dessas camadas. Em adição, as informações de controle dos protocolos são adicionadas (na origem) ou retiradas (no destino) em cada camada para garantir que os dados sejam transmitidos de forma confiável. A figura 2.2 demonstra todo esse processo.

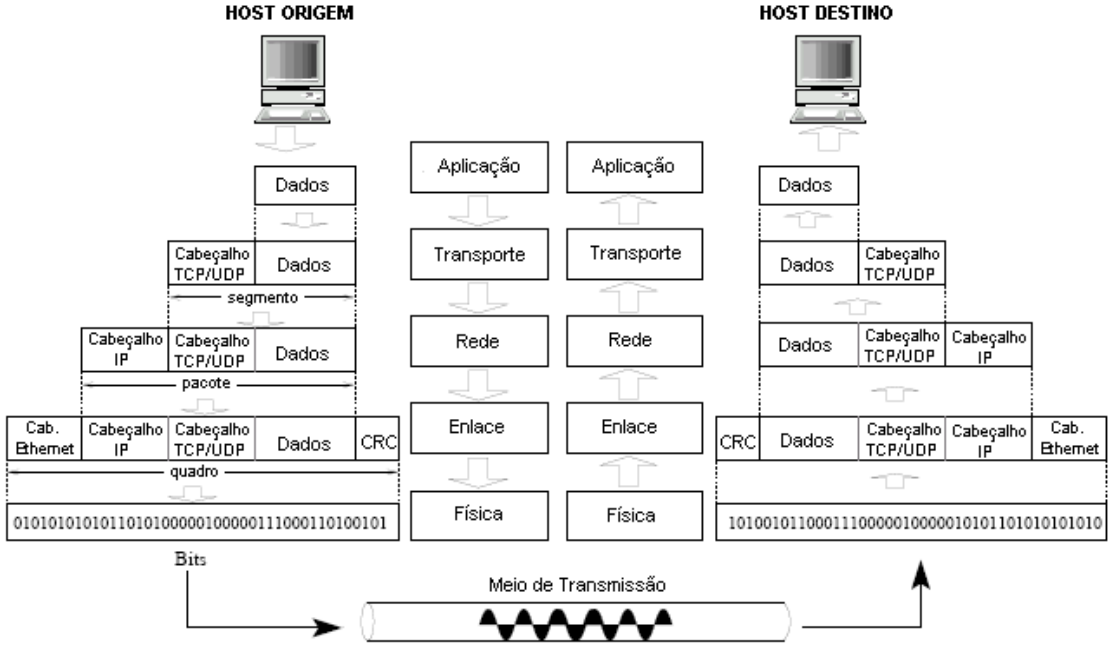

Figura 2.2: Comunicação de dados entre as camadas do Modelo Funcional TCP/IP

Na origem, os dados a serem transmitidos são gerados pela camada de aplicação, passados para as camadas inferiores, até chegar ao meio de transmissão. Os dados são transmitidos em forma de PDUs (*Packet Data Unit*) através das camadas. Cada camada define seu próprio PDU, nominalmente, Segmento para a camada de transporte, Pacote para a camada de rede e Quadro para a camada de enlace. As PDUs contém os dados recebidos da camada posterior, juntamente com as informações de controle do protocolo em uso (chamado de cabeçalho). O processo de inserção de cabeçalho é usualmente chamado de encapsulamento.

Na camada de transporte, um cabeçalho TCP/UDP é adicionado aos dados. O cabeçalho

contém informações como número de porta, o qual identifica um processo que está executando na camada de aplicação e utiliza essa porta para realizar a comunicação com a camada de transporte, e algumas outras informações de controle especificadas pelo protocolo de transmissão (TCP, por exemplo - ver seção 2.3.4).

Na camada de rede, um cabeçalho IP é adicionado à PDU recebida da camada de nível posterior. O cabeçalho contém os endereços IP origem e destino, este último usado para especificar o destinatário do pacote na Internet.

Na camada de enlace, um cabeçalho Ethernet (ou outro cabeçalho de um protocolo da camada de enlace) é inserido ao quadro, além de um campo de 32 bits, chamado de CRC (*Cycle Redundancy Check*) que é inserido ao final do quadro que busca garantir a integridade do conteúdo do quadro durante sua transmissão.

Já a camada de mais baixo nível no modelo funcional TCP/IP, a camada física, irá transmitir o quadro formado na camada de enlace pela rede.

Essas mesmas operações são realizadas no *host* destino, por exemplo, o quadro será desencapsulado, o código CRC será verificado a fim de garantir a integridade do quadro durante a transmissão, além das outras operações realizadas até os dados chegarem à camada de aplicação no *host* destino.

2.3 Alguns protocolos do Modelo TCP/IP

Nesta seção, alguns protocolos de especial interesse para o escopo deste trabalho serão discutidos. Serão apresentadas suas funcionalidades, posições em relação ao modelo TCP/IP e, quando conveniente, seus formatos de dados que, em determinada instância, serão tratadas como entradas de um sistema que o implemente.

2.3.1 Internet Protocol (IP)

Definido pelo RFC 791 (POSTEL, 1981), o *Internet Protocol* (IP) compõe a camada 3 do modelo OSI (camada de rede). Esta camada é responsável pelo endereçamento dos pacotes de informação dos dispositivos origem e destino e possível roteamento entre as respectivas redes, se diferentes. Este roteamento é executado através de endereços IP, os quais são compostos de 4 octetos, que identificam logicamente fonte e destino da informação. Este protocolo, usando a parte identificador de rede do endereço, normalmente designada pelos seus primeiros bytes, pode definir a melhor rota através de uma tabela de roteamento mantida e atualizada pelos roteadores.

Este protocolo recebe os dados da camada superior (transporte) na forma de segmentos. Ocorre então o processo de fragmentação e os conjuntos de dados passam a se chamar datagramas, os quais são então codificados para envio à camada inferior (física) para encaminhamento no meio físico. Um datagrama IP possui os seguintes campos, nesta ordem (conforme Figura 2.3):

- Cabeçalho (32 bits): contém informações sobre a versão do número IP, e o tipo de serviço que está sendo executado (ou solicitado);
- Comprimento (16 bits): informa o comprimento do datagrama incluindo dados e cabeçalho;
- Fragmentação (16 bits): instrui ao protocolo como recuperar datagramas quando chegam após um processo de fragmentação muito comum em interfaces defeituosas e de tráfego intenso;

- Time to Live (8 bits): TTL, informa o número de roteadores que podem redirecionar o datagrama. O valor é decrementado até zero a cada roteador, quando então é descartado;
- Seleção de protocolo (8 bits): diz respeito a qual protocolo deverá receber o datagrama na próxima camada, se TCP ou UDP;
- Verificação de erro (16 bits): seleciona qual processo será utilizado na detecção de erros (verificação de redundância cíclica ou checagem na seqüência de quadros);
- Endereços fonte e destino (ambos com 32 bits): caracteriza por completo toda informação sobre endereçamento, necessária ao processo de roteamento;
- Dados (tamanho livre, definido pelo tipo de rede): contém o dado que está sendo comunicado, em si.

	1	2	3	4	5	6	7	8	9
Bits	32	16	16	8	8	16	32	32	xxx
Descrição	Cabeçalho	Comprimento	Fragmentação	TTL	Protocolo TCP ou UDP	Verificação de Erros	Endereço Fonte	Endereço Destino	Dados

Figura 2.3: O datagrama TCP/IP

Embora todas as informações necessárias para que o IP possa se comunicar com o resto da rede estejam distribuídas nestes campos, é importante observar que a camada de rede utiliza endereços lógicos de 32 bits, mas para que os dados cheguem aos *hosts*, é necessário um outro tipo de endereçamento, que designa individualmente cada dispositivo de rede. Esse endereço, chamado de Media Access Control (MAC) é gerenciado pelo protocolo ARP, conforme será visto a seguir.

2.3.2 Address Resolution Protocol (ARP)

Definido pelo RFC 826 (PLUMMER, 1981), o protocolo ARP é responsável por mapear um endereço IP para um endereço do tipo MAC, através de uma tabela que associa a localização lógica de uma determinada máquina com a sua estrutura física de comunicação. Na construção do datagrama, a aplicação conhece os endereços MAC e IP da fonte "A"e somente o endereço IP do destino "B". Para descobrir o endereço MAC de "B"o protocolo ARP envia um broadcast a todos os dispositivos do segmento perguntando ao dono do IP "B"o seu endereço MAC. Por sua vez, o dispositivo dono do IP, envia também por broadcast, ou seja, para todos, o seu endereço MAC. Todos os dispositivos do segmento acrescentam na sua tabela ARP (IP x MAC), também chamada de proxycache ARP, este registro relativo a "B", que permanece durante um certo tempo. Finalmente, o dispositivo "A"envia o quadro destinado ao dispositivo "B". Neste exemplo, o mesmo quadro é enviado para "B"e a interface do roteador deste segmento, porém somente o dispositivo "B"irá abrir o quadro até a última camada pois somente ele tem o endereço MAC

destino. Se houvesse outros dispositivos no segmento, eles passariam a conhecer também o endereço MAC de "B"de maneira que se quiserem enviar algo este *host* posteriormente, não seria mais necessário um broadcast ARP.

2.3.3 Internet Control Message Protocol (ICMP)

Definido pelo RFC 792 (POSTEL, 1981), o ICMP é um protocolo de mensagens de controle usado para informar outros dispositivos de importantes situações das quais se podem citar como exemplo: fluxo de mensagens maior que a capacidade de processamento de um dispositivo, que se caracteriza quando um *host* destino de capacidade inferior manda uma mensagem pedindo à origem para que diminua ou pare seu fluxo de transmissão; comunicação de eventos relacionados ao parâmetro "time to live", como por exemplo a informação à fonte de que um determinado pacote foi descartado; e mensagens de redirecionamento, quando o roteador determina que um caminho melhor que o praticado atualmente existe para o pacote que acabou de ser enviado. A aplicação PING utiliza esse mecanismo de troca de mensagens para sua operação.

2.3.4 Transmission Control Protocol (TCP)

Definido pelo RFC 793 (POSTEL, 1981), o protocolo TCP encontra-se um nível acima da camada de rede, e é dito um protocolo de transporte. Define a maneira para tratar datagramas perdidos ou corrompidos. Além disso, o TCP é responsável pela segurança na transmissão e chegada dos dados ao destino e também define todo o processo de início de conexão e multiplexação de múltiplos protocolos da camada de aplicação em uma única conexão, otimizando assim a ligação de várias aplicações com o mesmo destino.

Ao contrário do UDP (User Datagram Protocol), um protocolo de mesmo nível, o TCP é orientado à conexão, ou seja, estabelece um caminho negociado entre os *hosts* fonte e destino antes de enviar os dados, possuindo mecanismos de confirmação de envio e recebimento de informações. Uma sincronização em três níveis é executada para o estabelecimento de uma conexão. O *host* fonte envia um dado de sincronização que será interpretado e reconhecido pelo destino, que acusará este evento com um sinal apropriado, contendo uma seqüência numérica que o identifica. Por fim, a fonte envia um último sinal de sincronização dizendo que, a partir de então, começará a enviar dados.

A estrutura do cabeçalho TCP, mostrado na figura 2.4, usa um par de 16-bit para endereçamento de porta origem e destino. Os próximos 32 bits representam o número de seqüência (sequence number), que identifica o segmento na rede e também é usado para reordenar o segmento na chegada ao destino.

O campo "acknowledgment sequence number"é usado para informar ao *host* remoto que o segmento em questão foi recebido com sucesso. O "acknowledgment sequence number"é sempre um valor acima do número de seqüência do segmento TCP recebido. O campo "data offset"indica o número de quatro octetos presentes dentro do cabeçalho TCP. Os seis bits seguintes são flags usados para indicar uma série de condições:

- URG: indica se o pacote é setado como urgente;
- ACK: indica se o campo acknowledgment é válido;
- PSH: é setado quando o transmissor quer que os dados do segmento seja repassados para a aplicação remota;

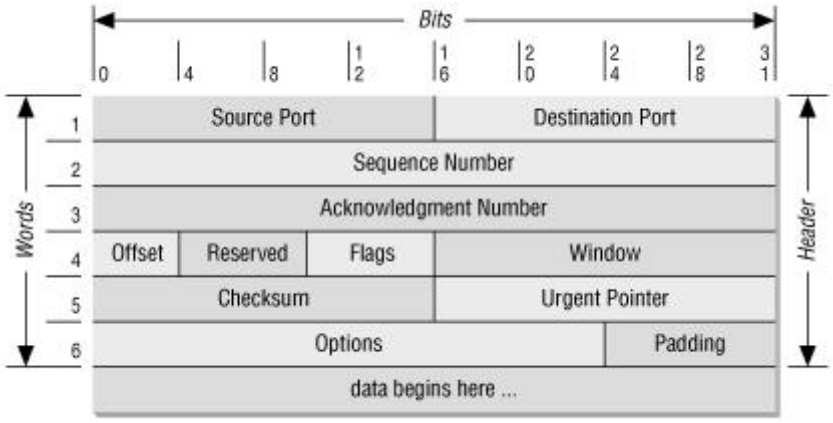

Figura 2.4: Formato do Segmento TCP

- RST: é usado para resetar a conexão TCP;
- SYN: é usado para sincronizar o início de uma conexão;
- FIN: é usado para fechar uma conexão.

O campo "window"é um contador 16-bit que é usado para indicar o número de pacotes que o receptor pode receber em seu *buffer* em um determinado período de tempo. O campo "TCP checksum"é usado para assegurar a integridade do segmento. O campo "urgent pointer"indica o número de seqüência do octeto final quando a flag "urgent"está setada. Por fim, o campo "TCP DATA"contém os dados que estão sendo transportados no segmento em questão.

O protocolo TCP, em um nível superior, é responsável por direcionar os dados para a aplicação correta. Isso é feito através dos campos portas origem e destino no cabeçalho TCP. O valor presente nesse campo é relacionado dinamicamente pelas aplicações. A exceção são as 1024 primeiras portas, previamente definidas, que só podem ser usadas por aplicações específicas. Assim, haverá uma conexão estabelecida na porta 80, por exemplo, somente se um vínculo HTTP entre um cliente e um servidor estiver ativo.

2.3.4.1 Máquina de Estados do Protocolo TCP

Nesta seção é apresentada a Máquina de Estados do protocolo TCP. A representação é feita com base nas interações realizadas no lado servidor (figura 2.5) e no lado cliente (figura 2.6) de uma comunicação realizada pelo protocolo TCP.

Nas figuras, as setas representam as interações entre os estados, realizadas a partir de um evento ocorrido pela chegada (*recv*) e envio (*send*) de um pacote TCP, ou através da solicitação de uma operação pela aplicação (*appl*). O estado inicial é representado pelo estado *CLOSED*.

Na máquina de estados do servidor TCP, os eventos possíveis que geram as transições de estados são detalhados a seguir:

1. *Passive open* - Operação solicitada pela aplicação;
2. Recebeu um pacote SYN do cliente e lhe envia um pacote SYN-ACK;
3. Recebeu um pacote ACK do cliente (não envia nada);

4. Recebeu um pacote FIN do cliente e lhe envia um pacote ACK;

5. Operação solicitada pela aplicação, prosseguida pelo envio de uma pacote FIN para o cliente;

6. Recebeu um pacote ACK do cliente, finalizando a conexão TCP.

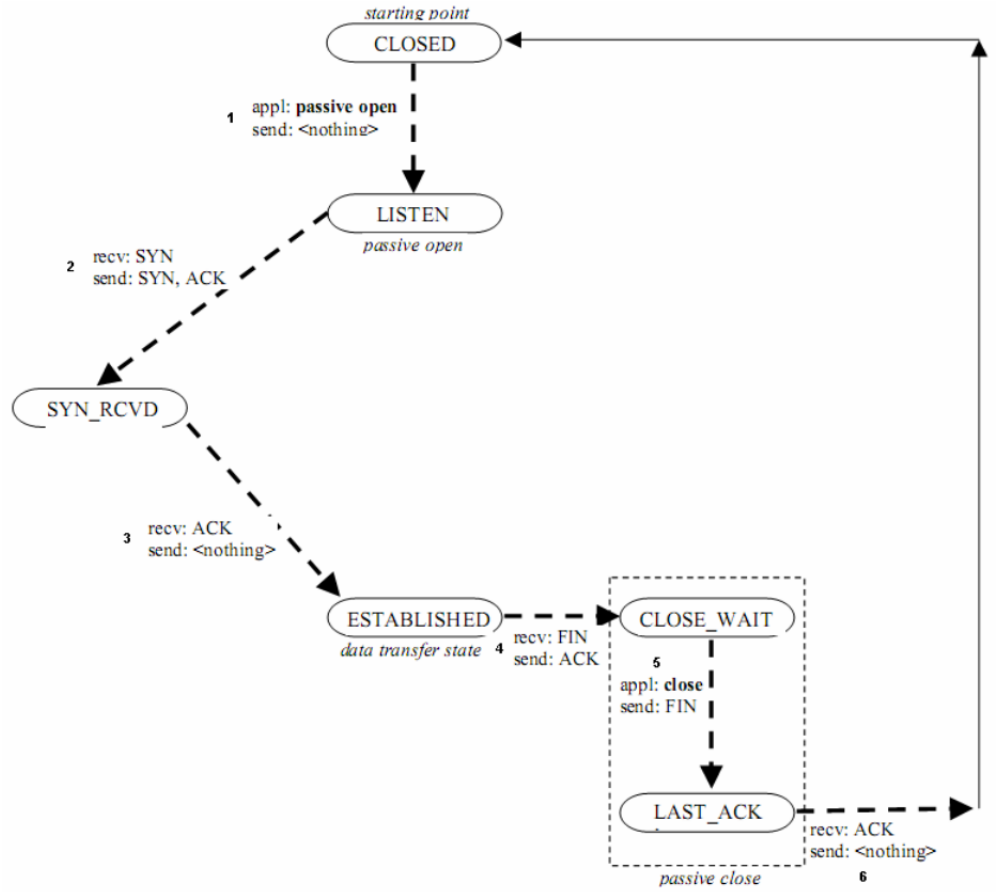

Figura 2.5: Máquina de estados de um servidor TCP

Na máquina de estados do cliente TCP, os eventos possíveis que geram as transições de estados são detalhados a seguir:

1. *Active open* - Operação solicitada pela aplicação para conexão à um servidor, prosseguida pelo envio de um pacote SYN para o servidor;

2. *Close* - Operação solicitada pela aplicação;

3. *Timeout* - Excedido tempo limite de espera;

4. Recebeu um pacote SYN e um ACK do servidor e lhe envia um pacote ACK;

5. Operação solicitada pela aplicação, prosseguida pelo envio de um pacote FIN para o servidor;

6. Recebeu um pacote ACK do servidor (não envia nada);

7. Recebeu um pacote FIN do servidor e lhe envia um pacote ACK;

8. Temporizador de 2 ms.

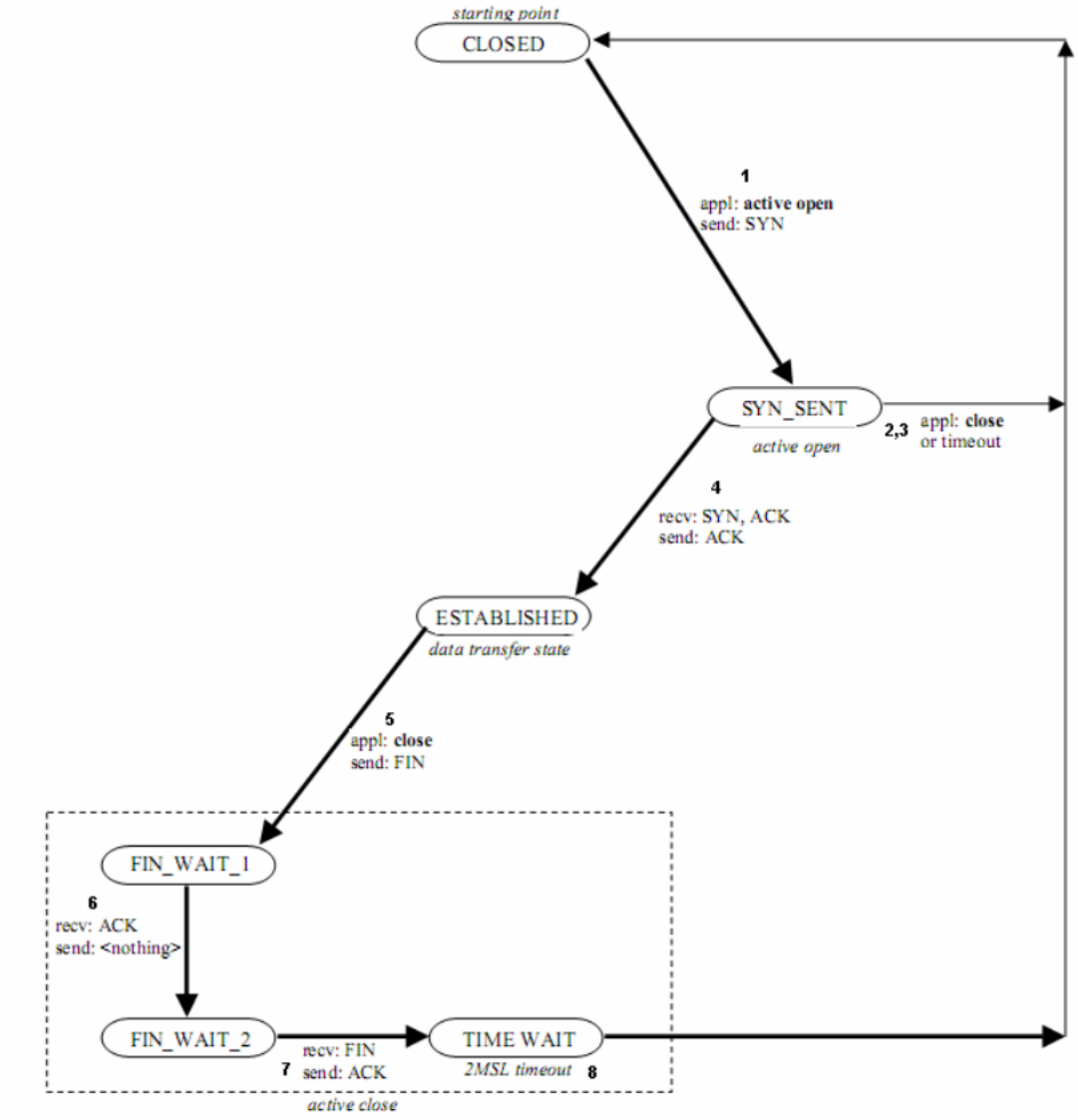

Figura 2.6: Máquina de estados de um cliente TCP

2.3.4.2 Estabelecimento de uma Conexão TCP

O estabelecimento de uma conexão TCP é efetuado através da negociação de certos parâmetros do cabeçalho TCP, visando garantir que a sessão utilizará o mais largo pacote IP possível sem fragmentação e que o tamanho da janela de transmissão usada na transferência é adequada para o produto banda-atraso (bandwitdh-delay) do caminho da rede em que o pacote ser transportado.

A primeira fase de uma seção TCP é o estabelecimento da conexão (figura 2.7). Isto requer um pequeno protocolo de "handshake", onde são transferidos três pacotes TCP, definido pelas seguintes etapas:

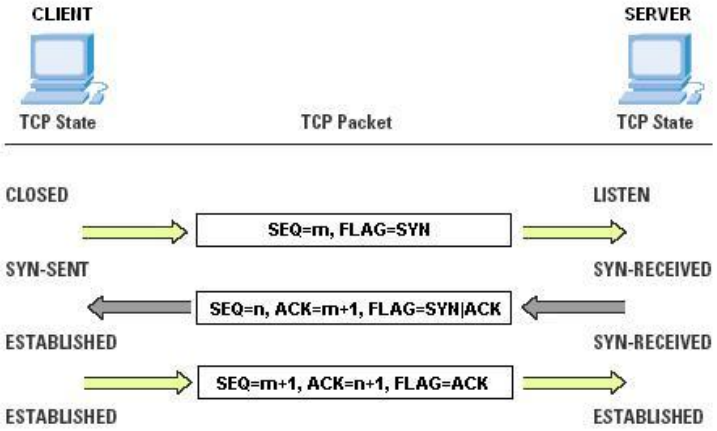

Figura 2.7: Protocolo de estabelecimento de uma Conexão TCP

1. O transmissor (cliente) envia um pacote para o receptor (servidor) em uma determinada porta (destination port) com um número de seqüência inicial no campo "Sequence Number"usando para isso um pacote SYN (flag SYN habilitada);

2. O receptor responde através de um pacote SYN+ACK (flags SYN e ACK habilitadas) com o campo "Acknowledgement Sequence Number"contendo o número de seqüência inicial do pacote recebido mais um, além do número de seqüência inicial por parte do receptor no campo "Sequence Number";

3. O transmissor responde através de um pacote ACK (flag ACK habilitada) com o campo "Sequence Number"contendo a confirmação do último número de seqüência do transmissor e o campo "Acknowledgement Sequence Number"contendo número de seqüência do pacote recebido mais um.

Com a conexão estabelecida tanto no lado do cliente como no lado do servidor, os dados podem começar a serem transferidos entre o cliente e o servidor.

2.3.4.3 Finalização de uma Conexão TCP

Já a finalização de uma conexão é realizada pela transferência de quatro pacotes entre os *hosts* cliente e servidor (2.8), definida pelas seguintes etapas:

1. O transmissor (cliente) envia um pacote para o receptor (servidor) com o campo "destination port"contendo o nº da porta que se deseja finalizar a conexão, usando para isso um pacote com as flags FIN e ACK habilitadas;

2. O receptor responde com um pacote ACK (flag ACK habilitada) contendo no campo "Acknowledgement Sequence Number"o número de seqüência inicial do pacote recebido mais um −, além do número de seqüência inicial por parte do receptor no campo "Sequence Number";

3. O receptor envia um pacote para o transmissor com as flags FIN e ACK habilitadas;

4. O transmissor responde através de um pacote ACK (flag ACK habilitado) com o campo "Sequence Number"contendo a confirmação do último número de seqüência recebido do receptor, além da informação no campo "Acknowledgement Sequence Number"contendo número de seqüência do pacote recebido mais um.

Após realizada essas etapas, as conexões estarão finalizadas tanto no servidor como no cliente.

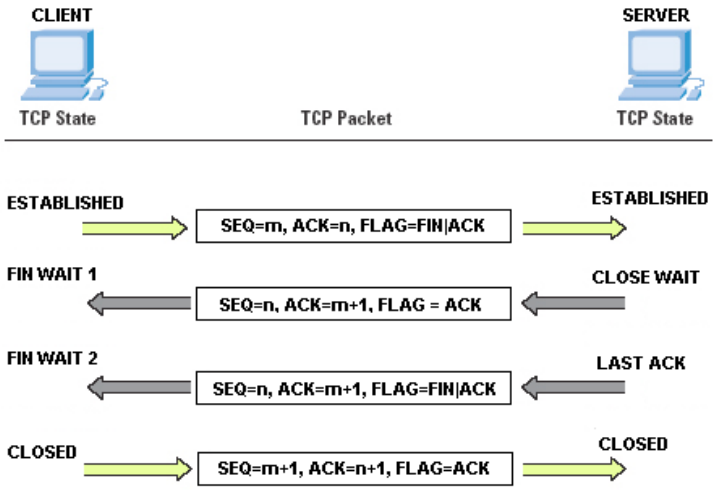

Figura 2.8: Protocolo de finalização de uma Conexão TCP

2.4 O Processamento TCP/IP

Com o constante crescimento da disponibilidade de banda de Internet, a pilha de protocolos TCP/IP passou a ser um gargalo no processamento do tráfego gerado por aplicações que utilizam essas altas bandas para se comunicarem.

Dentre os protocolos que compõem a pilha TCP/IP citados na seção 2.3, o protocolo TCP é o que requer maior carga de processamento. Este protocolo foi criado em 1981, onde taxas de transmissão na ordem de 10Mbps eram consideradas altíssimas. Porém, com o surgimento de novas tecnologias que permitiram o aumento da banda de passagem nas linhas de transmissão, aquela taxa, considerada altíssima na concepção do procotolo TCP, foi crescendo chegando atualmente a taxas de 10Gbps, tornando inconcebível a utilização do protocolo TCP como fora implementado em sua concepção. Em virtude disso, o protocolo TCP vem sofrendo constantes atualizações durante todos esse anos, o que possibilitou sua utilização até os dias de hoje.

2.4.1 Números de seqüência dos segmentos TCP

Apesar das constantes melhorias no protocolo TCP visando sobrepujar suas limitações, atualmente não é possível alcançar taxas suficientes para utilizar plenamente um link de uma rede gigabit realizando o processamento do protocolo através de uma plataforma de software (GUERRA, 2006). Uma das características do protocolo TCP que podem comprometer sua utilização em redes gigabit são os números de seqüência dos segmentos TCP. Durante a transmissão de um segmento (pacote) pela rede, este pode ter sua chegada atrasada ou até mesmo a entrega não ser efetuada. Para tratar dessas questões são utilizados números de seqüência para identificar cada segmento e assim poder reordenar os segmentos no caso de chegada atrasada ou solicitar uma retransmissão no caso de perda do segmento pela rede.

O número de seqüência está presente no cabeçalho do segmento TCP e, por definição, deve ser único para cada segmento que está em trânsito na rede. Pardridge (PARTRIDGE,

1998) analisa que em uma rede 10Mbps são necessários 1700 segundos para que o contador do número de seqüência reinicialize. Uma vez que um pacote IP tem um tempo de vida na rede em torno de 120 segundos, não há possibilidade de dois segmentos com o mesmo número de seqüência estarem na rede ao mesmo tempo. Já em uma rede 1Gpbs, o contador é reiniciado a cada 17 segundos e, em uma rede 10Gpbs, a cada 1,7 segundos. Sendo assim, nessas redes de alta velocidade há possibilidade de dois segmentos com o mesmo número de seqüência estarem na rede ao mesmo tempo, acarretando problemas na ordenação dos pacotes pelo receptor/transmissor.

2.4.2 Tamanho dos segmentos TCP

Outro fator que pode influenciar a taxa de envio/recebimento em uma rede gigabit é o tamanho dos segmentos TCP. Na chegada de um segmento TCP, este deve ter seu cabeçalho processado. Portanto, cada segmento possui um overhead devido ao processamento do cabeçalho, não importando o tamanho do segmento.

Por exemplo: uma aplicação deve enviar 9.000 bytes. No caso de se utilizar o tamanho mínimo de pacote (60 bytes) seriam necessários, no mínimo, 150 segmento para enviar todo o conteúdo especificado. Levando em conta que o overhead de processamento do cabeçalho de cada segmento é de *X segundos*, o overhead total, nesse caso, seria de *150*X segundos*. Em outro caso, utilizando-se pacotes do tipo Jumbo (DYKSTRA, 1999), onde cada pacote tem até 9.000 bytes, seriam necessários somente um segmento para enviar o mesmo conteúdo e o overhead de processamento do cabeçalho seria de somente *X segundos*.

Vale salientar que o overhead no processamento de um segmento TCP não se deve somente ao processamento do cabeçalho. Funções como o cálculo do checksum e o gerenciamento de conexões TCP também requerem uma carga de processamento considerável, mas esse overhead não é influenciado pelo tamanho do segmento TCP.

3 TCP/IP OFFLOAD

A pilha de protocolos TCP/IP foi desenvolvida para ser processada em software executando sobre um processador central presente no servidor, uma vez que a capacidade de processamento alcançada era suficiente na época da sua concepção. Porém, nos últimos anos, pelo surgimento das conexões de rede com velocidade na ordem de Gbps (Gigabit por segundo), a CPU ficou sobrecarregada com a alta carga de processamento exigido pela pilha TCP/IP (FENG et al, 2005). Atividades como cópias de memória, computação de checksum, tratamento de interrupções geradas pelo dispositivo de rede e reordenamento de pacotes que chegam fora de ordem, ocasionam essa sobrecarga da CPU.

Em redes de alta velocidade, a CPU acaba dedicando mais tempo de processamento para o tráfego de rede do que para outras aplicações que dependem dos seus recursos de processamento. Para limitar o uso da CPU no processamento de tráfego de rede, surgiu a técnica chamada "TCP/IP offload", que é implementada por um dispositivo chamado "TCP/IP offload engine"(TOE). A idéia básica é implementar o processamento dos protocolos, antes executada pela CPU, em um hardware dedicado que pode ser implantado no adaptador de rede, por exemplo. Essa implementação pode ser realizada através de um processador de rede (em tecnologia ASIP), um ASIC ou uma combinação dos dois.

3.1 Capacidade da rede, oferta e demanda

Antes de entrar nos detalhes da técnica "TCP/IP offload", vale a pena comentar sobre a capacidade das redes sobre as quais os protocolos TCP/IP operam. Redes locais (LANS) são predominantemente implementadas usando Ethernet, a qual atingiu uma taxa de transferência de dados desde 10 Mbps (Megabits por segundo) em 1990, passando para 100 Mbps em meados de 1990 e hoje em dia já é comum encontrar redes locais operando à velocidade de 1 Gbps (Gibagits por segundo). Em 2002, nos enlaces entre redes de grande porte (WANS), tornou-se essencial a operação de redes em velocidades na casa de 10 Gbps para o transporte de uma demanda de dados que só tenderia a crescer cada vez mais.

As tecnologias que permitiam um aumento da velocidade das linhas de transmissão estavam disponíveis, porém, historicamente, ela sempre teve um atraso de um ou dois anos para ser aplicada. Isso ocorreu porque os pontos da rede não eram capazes de processar os dados nessa mesma velocidade e os projetistas tinham que implementar novas soluções em software da pilha TCP/IP para alcançar o desempenho desejado. Por exemplo, a tecnologia Gigabit Ethernet foi introduzida em 1998, mas só foi difundida dois anos mais tarde.

A demanda por banda de rede tem crescido continuamente. É comum hoje em dia se encontrar computadores pessoais com dispositivos de rede (placas de rede) que operam na

velocidade de até 1 Gbps. A demanda por essa tecnologia se torna evidente também, por exemplo, em uma organização de porte médio que possui um sistema de armazenamento de arquivos centralizado e um servidor de banco de dados que requer conectividade gigabit para prover respostas ao cliente no tempo correto. O desafio que surge é como operar estas interfaces na capacidade especificada e ainda prover suficiente poder computacional para o servidor rodar outras aplicações necessárias para seu funcionamento, e tudo isso sem um preço abusivo.

3.2 Aumento poder CPU x Processamento TCP/IP

O poder de processamento das CPUs que executam o software da pilha de protocolos TCP/IP tem crescido continuamente. Analisando o avanço alcançado por processadores Intel Pentium, por exemplo, a velocidade do clock aumentou cerca de 5.000 por cento, avançando de 60 MHz para em torno de 3 GHz.

Pesquisas indicam que o aumento do poder de processamento das CPUs tem comportamento equivalente ao crescimento da velocidade das linhas de transmissão, porém a habilidade dessas CPUs para processarem o tráfego TCP/IP não cresceu na mesma taxa.

Currid (CURRID, 2004) analisa o custo relativo do processamento TCP em razão do uso de CPU, velocidade da CPU e throughput alcançado através da seguinte equação:

$$RelativeCost = (CPUutilization * CPUspeed)/Throughput$$

onde, $CPUutilization$ é o fator de utilização da CPU durante o processamento da pilha de protocolo TCP/IP (medido em percentagem), $CPUspeed$ é a freqüência de processamento do processador (medido em MHz) e $Throughput$ é a quantidade de dados processados em um determinado espaço de tempo (medido em $Mbps$ - Mega bits por segundo).

No mesmo trabalho, Currid (CURRID, 2004) efetua uma análise onde o "TCP throughput"e a utilização da CPU são medidos através de diferentes tamanhos de segmento TCP, usando CPUs de 800 MHz e 2.4 MHz, rodando sobre a mesma configuração de hardware. Essa modelagem objetiva mostrar o quanto é necessário de poder de computação para processar uma quantidade determinada de tráfego de rede TCP.

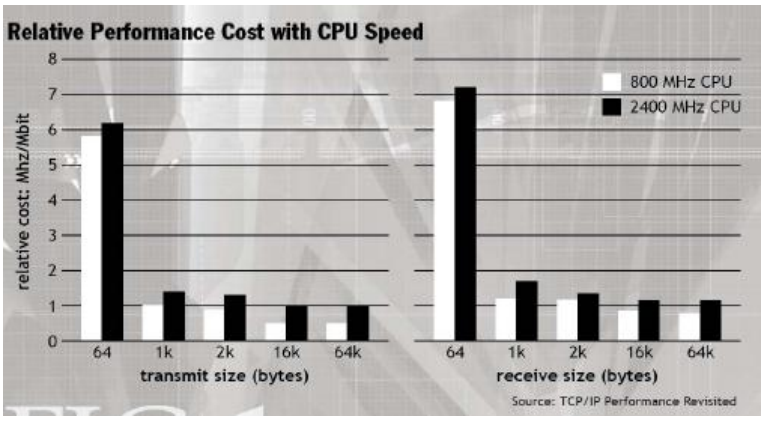

Figura 3.1: Custo relativo em MHz/Mbit para processamento de tráfego TCP

Como pode ser visualizado na figura 3.1 (CURRID, 2004), comparando-se os resultados alcançados pela duas CPUs, o aumento da velocidade da CPU em três vezes (de 800

MHz para 2.4 GHz) não foi suficiente para melhorar o desempenho da transmissão de pacotes TCP na rede, na mesma proporção. Em todos os casos, o máximo que se alcançou foi o dobro do desempenho para pacotes de tamanhos maiores do que 16 KB. A principal razão dessa disparidade é a divergência entre o desempenho da CPU e o desempenho de outros componentes como memória e dispositivos de entrada e saída. Uma outra observação que pode ser efetuada sobre a modelagem realizada é que o custo relativo para processamento de 1 Mb de informação TCP é de 1 MHz de CPU, para pacotes maiores do que 1 KB. Isso equivale a dizer que seria necessário o uso exclusivo, por parte da máquina de estados TCP, de um processador de 1 GHz para alcançar o throughput de 1 Gbps.

3.3 Overhead TCP/IP

Em um ambiente webserver comum, a maior porção de utilização da CPU é atribuída à pilha TCP/IP (FOONG et al, 2003). Isso se deve grande parte ao processamento TCP dos pacotes, que por si só torna-se o grande overhead do sistema (CLARK, 1989). Muitas das causas do overhead TCP é atribuída às suas implementações em Sistemas Operacionais e conseqüentes ineficiências, que podem ser mapeadas segundo Kant (KANT, 2003), em:

- Interrupções causadas pela invocação/término do processamento por parte da pilha TCP/IP;
- Múltiplas trocas de contexto entre os modos de usuário e *kernel* do Sistema Operacional;
- Múltiplas camadas intermediárias que precisam ser processadas antes da entrega do pacote TCP;
- Uma ou mais cópias de memória-para-memória;
- Tratamento de códigos de erro/exceção.

A seguir, são detalhadas as principais causas do overhead causado pelo processamento da pilha TCP/IP em software.

3.3.1 Processamento de Interrupções de CPU

Uma aplicação que deseja comunicar-se com um *host* remoto através da rede produz uma série de requisições ao sistema operacional em forma de interrupções para segmentar os dados a serem enviados em pacotes, além das interrupções geradas pelas operações sobre pacotes que realizam a sincronização de uma conexão TCP (ver subseção 2.3.4.2). Cada transmissão/recebimento de um pacote gera uma série de cópias de dados do *buffer* da camada de aplicação (espaço do usuário) para o *buffer* do *kernel* do sistema (espaço do *kernel*), e do *kernel* para o adaptador de rede. Em conseqüência a essas diversas cópias de dados de *buffer* para *buffer*, cada evento desses gera uma interrupção a ser tratada pelo sistema operacional, ocasionando uma grande quantidade de trocas de contexto − um tipo de multitarefa que direciona o foco da CPU de um processo para outro − neste caso, do processo da aplicação corrente para o *kernel* do sistema operacional e vice-versa (SENAPATHI et al, 2004).

A figura 3.2 demonstra um exemplo para melhor entendimento (região hachurada refere-se ao processamento efetuado em hardware, região não hachurada refere-se ao processamento efetuado em software). Para esse exemplo, pensa-se em uma rede gigabit (um

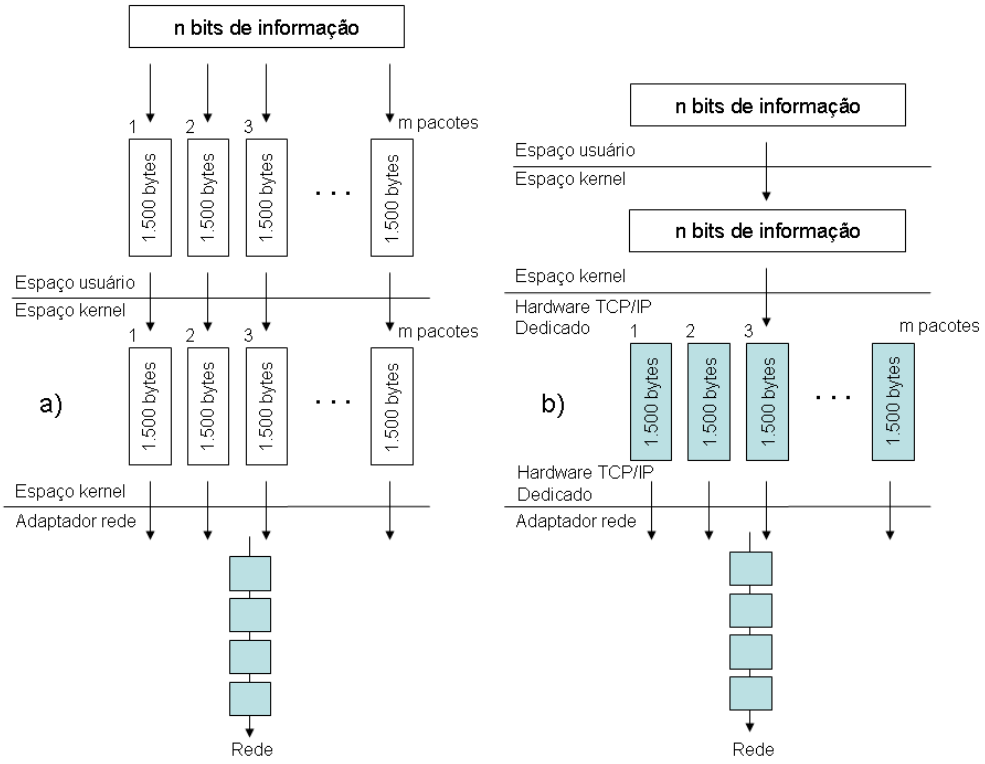

Figura 3.2: Comportamento de transmissão de pacotes em uma abordagem da pilha TCP/IP em a)software e em b)hardware

bilhão de bits por segundo), operando na plenitude de sua largura de banda. A CPU de um *host* nessa rede necessita processar em torno de 83.000 pacotes por segundo (ou um pacote a cada 12 microsegundos). Isso para o caso dos pacotes serem todos de tamanho em torno de 1.500 (um mil e quinhentos) bytes, o que dificilmente acontece no processo de formação dos pacotes em um ambiente de rede normal, aumentando ainda mais os recursos necessários para o processamento. Cada pacote de 1.500 bytes vai gerar uma interrupção para a cópia de dados do pacote entre os *buffers* do sistema. Com isso, o número de interrupções a serem processadas e conseqüentes trocas de contexto pelo sistema operacional tornam esse processamento praticamente inviável em uma arquitetura onde a pilha TCP/IP é processada por um software do *kernel* do sistema operacional.

Através do processamento da pilha TCP/IP em hardware, o processo de transmissão de dados pela camada de aplicação para um *host* remoto, não importando a quantidade de dados a ser transmitida, pode ser efetuado gerando-se apenas uma interrupção para copiar esses dados do *buffer* da camada de aplicação (espaço do usuário) para o *buffer* do *kernel* (espaço do *kernel*). A partir daí, o *kernel* irá transferir os dados do seu *buffer* para o *buffer* do hardware TCP/IP dedicado. O hardware específico para essa tarefa iria então receber esses dados, processá-los em uma arquitetura de alto desempenho, e dividi-los em pacotes de até 1500 bytes para serem enviados pela rede gigabit, isso sem gerar mais nenhuma interrupção para o sistema operacional. Portanto, o uso de um módulo em hardware, específico para esse processamento, poderia reduzir o número de interrupções de milhares para apenas uma ou duas por transação de entrada e saída.

3.3.2 Cópias de Memória

Em uma transação de envio de pacotes por parte de adaptadores de rede tradicionais (NICs), os dados são copiados do espaço de usuário (camada de aplicação) para o espaço do *kernel* do sistema operacional. O adaptador de rede então copia os dados do *kernel* para o *buffer* interno do adaptador. Essas tarefas requerem múltiplas trocas de dados através do barramento da memória do sistema. Quando pacotes são recebidos da rede, o NIC copia o pacote do *buffer* interno para o *buffer* residente no espaço de memória do sistema. Os pacotes são então copiados para o *buffer* TCP e finalmente para a aplicação, num total de três cópias de memória (SENAPATHI et al, 2004).

Uma arquitetura com TCP/IP Offload pode reduzir o número de cópias de memória para duas: O NIC copia os pacotes para o *buffer* TCP e então para o *buffer* da camada de aplicação.

3.3.3 Processamento do Protocolo

O processamento tradicional da pilha TCP/IP por um sistema operacional em software requer uma carga muito grande para o processamento de pacotes de sincronização na tarefa de estabelecimento de uma conexão TCP, os quais são de tamanho igual a 64 bytes. Em adição, o protocolo TCP deve manter armazenado informações sobre cada conexão criada, tal como o tamanho da janela de transmissão do pacote, tanto para o transmissor como para o receptor. Cada vez que um pacote é recebido ou enviado, a posição e o tamanho da janela é trocada e o sistema deve manter registrado esses valores.

O recebimento de um pacote consome mais carga de CPU do que uma transmissão (SENAPATHI et al, 2004). Um NIC tradicional deve armazenar os pacotes recebidos e então notificar o sistema operacional usando interrupções. Após a troca de contexto para tratamento da interrupção, o sistema processa a informação contida no pacote para que possa ser associada com uma conexão TCP aberta. Os dados do segmento TCP são então correlacionados com uma aplicação e copiados para a área de memória correspondente à aplicação.

Outro processamento necessário é o cálculo do checksum em cada pacote que o *host* envia ou recebe, visando determinar se o pacote está livre de erro. O protocolo TCP também armazena uma informação de *acknowledgment* para cada pacote recebido. Cada uma dessas operações resultam em uma interrupção gerada para o sistema operacional. Como resultado, a CPU do *host* pode ser saturada pela freqüente geração de interrupções e processamento do protocolo.

Usando uma abordagem de processamento da pilha TCP/IP em hardware, todo esse processamento é efetuado por um hardware dedicado, permitindo à CPU do *host* uma carga maior para processamento de outras tarefas.

3.4 Soluções em Hardware ou Software

Como citado na seção anterior, o maior overhead no processamento TCP/IP é causado pelo gerenciamento de I/O e de *buffers* nos sistemas operacionais (CLARK, 1989). Para sanar esses problemas, os trabalhos na área enfocam dois tipos de soluções:

- soluções em software: identificam os "gargalos"do processamento em software que podem ser otimizados, tais como operações do *kernel* do sistema e seus *drivers* de dispositivos;

- soluções em hardware: fazem uso das vantagens da velocidade de processamento em hardware para solucionar os problemas de "gargalos"na comunicação.

Os principais gargalos do processamento em software são: operações de interrupção, cópia de dados, entre outros. Alguns trabalhos atacam essas questões. Em Steenkiste (STEENKISTE, 1998), o gargalo é identificado no tráfego de memória existente entre a área destinada ao usuário e a área destinada ao *kernel*, além do tráfego entre a área do *kernel* e o *buffer* dos adaptadores de rede. Então é proposta uma arquitetura para dispositivos de rede, juntamente com uma interface de comunicação com o *kernel* do sistema operacional, onde as tarefas de checksum e armazenamento dos pacotes são feitas por um hardware dedicado.

Jacobson (JACOBSON, 1990) apresenta um algoritmo em software otimizado para processamento de pacotes TCP.

A metodologia mais freqüentemente utilizada é reduzir o overhead através do processamento do checksum dos pacotes por um hardware dedicado. Clark (CLARK, 1982), propõe uma técnica que combina operações de cópia e checksum para minimizar o custo da computação de checksum. A idéia é eliminar uma cópia na memória fazendo a cópia e o cálculo do checksum em uma única rotina. Outra técnica proposta é um mecanismo de "zero-pass checksum"(FINN et al, 1996), onde os pacotes são movidos diretamente da memória do sistema para os adaptadores de rede. Uma outra estratégia é implementar o checksum Internet diretamente nos adaptadores de rede (KLEINPASTE et al, 1995).

No próximo capítulo são detalhadas soluções existentes em hardware, já apresentadas na comunidade científica e, por suas características, podem ser consideradas semelhantes à arquitetura do iNetCore desenvolvida nesta dissertação.

4 ARQUITETURAS EXISTENTES

Neste capítulo serão relatadas algumas arquiteturas em desenvolvimento ou já implementadas pela comunidade científica que visam realizar o processamento de pacotes da pilha de protocolos TCP/IP.

As primeiras implementações da pilha TCP/IP foram desenvolvidas em software. Porém, a necessidade de processamento da pilha na mesma velocidade das linhas de transmissão, que hoje estão na casa de 10Gbps, levou à criação de novas implementações. Essas novas implementações, quanto à arquitetura alvo sobre a qual é projetada, podem ser divididas em dois tipos: ASIP ou ASIC.

As implementações em ASIP ou ASIC, conhecidas como "TCP/IP Offload Engines – TOE", visam abranger segmentos de mercado que estão surgindo impulsionados pelas novas tecnologias de rede. Esses segmentos têm algumas características em comum, mas diferem em relação aos parâmetros Ethernet tais como latência de rede e número de conexões simultâneas a serem gerenciadas. Os principais segmentos de mercado objetivados são:

- HPC (High-Performance Computing) e plataformas de "supercomputação";
- Sistema de entrega de conteúdo multimídia de alta qualidade;
- Próxima geração de armazenamento baseado em IP incluindo SAN (Storage Area Network), NAS (Network Attached Storage), e iSCSI (Internet SCSI);
- Sistemas de processamento de transações eletrônicas, tais como servidores de "e-commerce".

A seguir serão detalhadas as características das soluções ASIP e ASIC e algumas arquiteturas já implementadas de cada tecnologia.

4.1 Arquiteturas ASIP

As implementações em ASIP utilizam um processador de uso geral RISC com um conjunto específico de instruções para processamento dos protocolos da camada de rede e transporte do modelo TCP-IP. Também conhecidas como processadores de rede (em inglês, *network processors*), caracterizam-se por apresentar uma maior flexibilidade que as implementações ASIC, porém perdem na capacidade de processamento do tráfego de rede (throughput). Por ventura, podem fazer uso de módulos específicos em hardware para aumentar a capacidade de processamento.

Grande parte dos TOEs baseados em uma arquitetura ASIP são aplicados no desenvolvimento de dispositivos de rede que são usados no segmento de mercado iSCSI. Como os ASIPs são baseados em um microprocessador RISC, estes requerem um grande número de instruções para conduzir cada uma das conexões a serem gerenciadas pelo dispositivo de rede. Por isso sua aplicação na tecnologia iSCSI torna-se viável, visto que esta possui a característica de gerenciamento de um número pequeno de conexões simultâneas. Por outro lado, sua aplicação no segmento de mercado dos sistemas de entrega de conteúdo multimídia torna-se inviável, pois esses sistemas necessitam de gerenciamento de um número grande de conexões simultâneas.

Uma vez explicitado as características em comum desse tipo de implementação, será apresentado a seguir uma arquitetura ASIP que possui a característica de processamento de tráfego TCP/IP.

4.1.1 PRO3 System (PAPAEFSTATHIOU et al, 2004)

O PRO3 System é uma arquitetura ASIP baseada em um microprocessador RISC que promete acelerar o processamento de protocolos das camadas 1, 2 e 3 do modelo TCP/IP em interfaces de rede de alta velocidade. A figura 4.1 exibe detalhes internos de sua arquitetura.

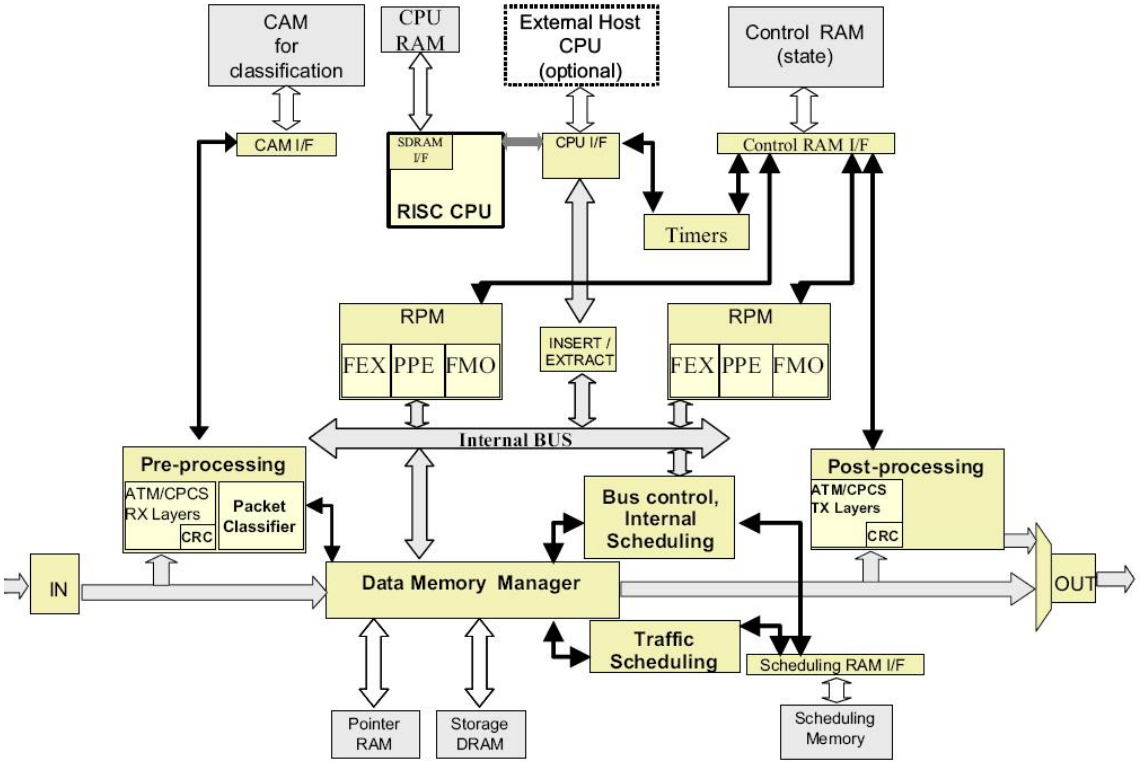

Figura 4.1: Diagrama de blocos da arquitetura do PRO3 System

Essa arquitetura possui características de paralelismo e pipeline, mesclando blocos dedicados em hardware (como os blocos Pre-processing e Post-processing) com um microprocessador de propósito geral RISC. Os blocos dedicados são responsáveis pelo processamento de características específicas de rede, tais como: processamento dos cabeçalhos, classificação dos pacotes, segmentação e reordenamento dos pacotes, armazenamento dos pacotes em *buffers*, etc.. O microprocessador RISC, que possui um conjunto

específico de instruções para o processamento do tráfego da rede, é responsável pela computação de tarefas de alto nível, tais como: cálculos matemáticos simples, atribuições de determinados itens de dados da rede aos valores constantes baseados em um outro valor buscado dentro do pacote, etc..

O PRO3 System foi fabricado em um chip de tecnologia 18nm ocupando cerca de 37mm2 de área, possuindo 836 mil gates e 1 Mbit de memória SRAM interna. Em testes de avaliação de desempenho, o dispositivo chegou a taxa de 2,5 Gbps (Giga bit por segundo) no processamento de pacotes TCP de tamanho, em média, de 128 bytes. Mais detalhes podem ser encontrados em (PAPAEFSTATHIOU et al, 2004).

4.2 Arquiteturas ASIC

As implementações em ASIC são baseadas em máquinas de estados e caracterizam-se por realizar todo o processamento TCP/IP em um hardware dedicado, implementada por meio de um VLSI ASIC. Isso garante um alto rendimento quanto à capacidade de processamento dos protocolos que compõem a pilha. Como os protocolos de rede são padronizados no tocante à definição do seu comportamento por parte dos grupos de trabalho que os desenvolvem, a característica de flexibilidade garantida por implementações em software e até mesmo em ASIP não se torna um fator muito importante, o que viabiliza ainda mais a implementação em ASIC. Ela torna-se ainda mais interessante quando sua aplicação é em larga escala.

A seguir serão detalhadas algumas arquiteturas ASIC já implementadas pela comunidade científica, sendo que todas foram prototipadas em componentes de FPGA.

4.2.1 TOE baseado em um microprocessador NIOS II (BOKAI et al, 2005)

A arquitetura do TOE apresentada por Bokai (BOKAI et al, 2005) apresenta os módulos exibidos na figura 4.2. Todo o processamento da pilha TCP/IP é realizado pelo TOE, que processa pacotes do tipo ARP, ICMP, IP, UDP (8 conexões simultâneas) e TCP (8 conexões simultâneas, funcionalidade não finalizada no protótipo apresentado).

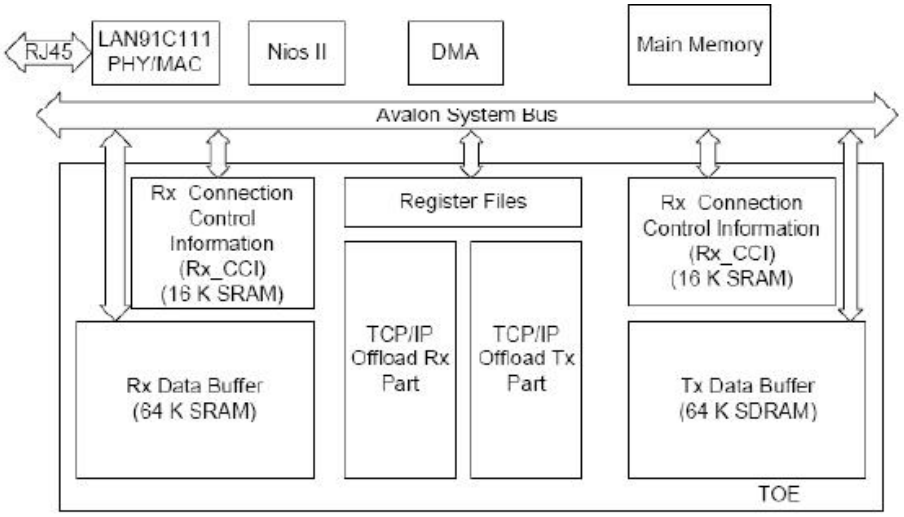

Figura 4.2: Visão geral da arquitetura do TOE baseado em um microprocessador NIOS II

A arquitetura como um todo apresenta os seguintes componentes:

- Nios II CPU

- Gerenciador DMA
- Memória Principal SRAM
- Adaptador PHY/MAC Lan91c111 tri-state
- ASIC TCP/IP Offload Engine

O processador Nios II controla a comunicação dos dados entre o MAC, o TOE e a memória principal. No recebimento de pacotes, o adaptador MAC recebe os pacotes da rede e armazena no *buffer* interno do TOE, que processa o pacote e o envia para a memória principal. No envio, o MAC primeiro envia os dados do pacote para o *buffer* interno do TOE, o qual adiciona os cabeçalhos e o transfere de volta para o MAC transmitir para a rede.

A arquitetura foi desenvolvida sobre a plataforma de FPGA da Altera e promete processar tráfego TCP/IP a uma taxa de 100 Mbps. O trabalho não apresenta resultados de ocupação do circuito gerado. Mais informações podem ser encontradas em (BOKAI et al, 2005).

4.2.2 Open Source TCP/IP Core (DOLLAS et al, 2005)

Dollas (DOLLAS et al, 2005) apresenta uma arquitetura compreendendo um "open-source IP core"capaz de processar pacotes da pilha TCP/IP. Os pacotes que são processados por essa implementação são os seguintes: ARP, ICMP, UDP, IP e TCP.

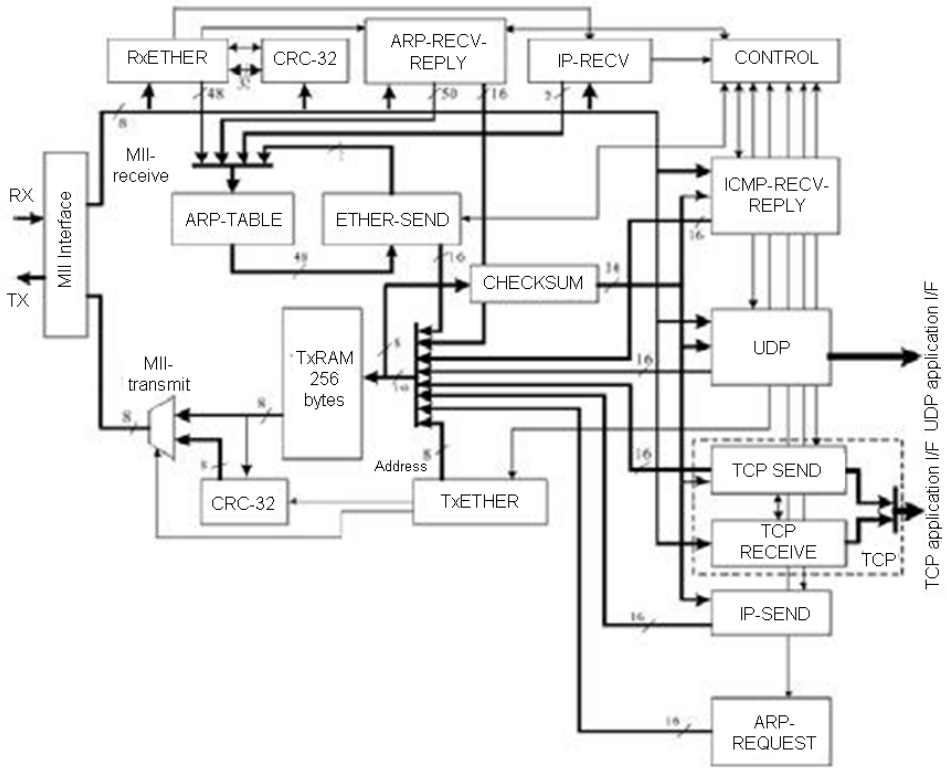

Figura 4.3: Arquitetura do TCP/IP Core

A arquitetura do TCP/IP Core é composta pelos seguintes módulos internos, que podem ser visualizados na figura 4.3 (DOLLAS et al, 2005):

- MII-INTERFACE – interface MII para controle dos sinais e recebimento/envio dos pacotes pela rede;
- RxETHER – aguarda o recebimento de um datagrama;
- ARP-RECV-REPLY – monta um pacote ARP Reply, quando acionado;
- ARP-RECV-REPLY – monta um pacote ICMP Reply, quando acionado;
- IP-RECV – identifica qual tipo de pacote está encapsulado no datagrama IP;
- CRC-32 – calcula o CRC (em inglês, Cyclic Redundancy Check) do datagrama de entrada;
- TCP – realiza o processamento de um segmento TCP;
- UDP – realiza o processamento de um segmento UDP;
- ARP-TABLE – mantém uma tabela que relaciona um endereço IP com um endereço físico;
- TxRAM – memória onde o pacote a ser transmitido é armazenado;
- TxETHER – implementa a subcamada MAC;
- IP-SEND – encapsula o datagrama a ser transmitido em um cabeçalho IP;
- ARP-REQUEST – monta um datagrama do tipo Arp Request e o transmite;
- CHECKSUM – realiza o CHECKSUM de uma certa quantia de dados do datagrama;
- CONTROL – responsável por sincronizar todos os módulos durante a recepção de um datagrama e também supervisiona o acesso à TxRAM;

A arquitetura é capaz de gerenciar 15 conexões UDP e 31 conexões TCP simultaneamente. Foi desenvolvida através de um fluxo de projeto ASIC, mas para fins de validação foi prototipada em FPGA no componente Xilinx Virtex 2 - XCV8000. O projeto total ocupou 10.000 (dez mil) slices, 10 Block RAMs e atingiu uma freqüência de 37,5 MHz. Através desse clock, a arquitetura apresentada promete trabalhar a uma taxa de 700 Mbps em modo full-duplex. Mais informações podem ser encontradas em (DOLLAS et al, 2005).

4.2.3 Measurement Engine (YUSUF et al, 2005)

Yusuf (YUSUF et al, 2005) apresenta uma arquitetura composta de módulos em software e hardware para a tarefa específica de análise de fluxo de rede. O objetivo principal da arquitetura, apresentada na figura 4.4, é coletar estatísticas da rede para diagnosticar problemas de desempenho como perdas de pacotes, além de possibilitar a aplicação de políticas de segurança através da análise dos pacotes. Para realizar essa análise, é necessário inspecionar pacotes do tipo TCP e usar pacotes UDP para transportar os dados das estatísticas para o cliente que usará esses serviços.

Essa arquitetura não permite o uso da pilha TCP/IP por parte de um projetista de hardware, e sim faz uso das suas características para proporcionar um outro tipo de serviço especializado - a análise do fluxo.

A arquitetura foi desenvolvida em FPGA no componente Virtex II Pro XC2V8000 da Xilinx. A arquitetura pode ser configurada para vários fluxos de pacotes. Na configuração máxima, onde podem ser processados até 500.000 fluxos de pacotes ao mesmo tempo,

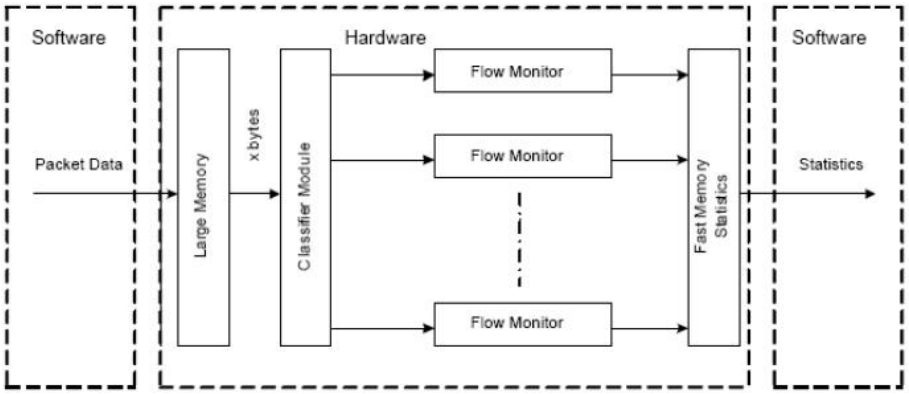

Figura 4.4: Arquitetura dos componentes de hardware do Measurement Engine

a síntese gerou um circuito com 2.000 (dois mil) slices e 885 BRAMs. A análise de desempenho foi realizada independentemente da tecnologia de prototipação. Em testes, a arquitetura chegou a um throughput de 24,75 Mbps por fluxo de pacotes. Mais detalhes podem ser encontrados em (YUSUF et al, 2005).

4.2.4 TCP/IP Offload Engine System over Gigabit Ethernet (WU et al, 2006)

Uma outra proposta de hardware para processamento da pilha TCP/IP é apresentada por WU (WU et al, 2006). Ele apresenta um TCP/IP Offload Engine baseado na tecnologia de FPGAs, unindo a velocidade de processamento alcançada pelo hardware desenvolvido com a flexibilidade de parte do processamento em software. Em hardware são realizadas funções como o cálculo do checksum, identificação dos tipos dos pacotes e máquinas de estados para processamento de pacotes específicos. como ARP e ICMP echo/request. Em software, são realizadas tarefas como o gerenciamento de conexões TCP, gerenciamento dos *buffers* e remontagem de pacotes que chegam fora de ordem. O software é implementado na forma de um firmware que pode ser atualizada para a inserção de novas funcionalidades.

A figura 4.5 apresenta o digrama de blocos do TCP/IP Offload Engine. O hardware é implementado sobre dois domínios de clock, um clock de 125MHz usado pela interface com o MAC Gigabit Ethernet e outro clock de 100MHz utilizado pelos outros módulos que compõem o hardware do TOE. Os principais módulos do TOE são descritos abaixo:

- GMAC RX e GMAC TX − interface GMII para controle dos sinais e recebimento/envio dos pacotes pela rede;
- Packet Identification − encaminha os frames recebidos para os módulos ARP ou Rx Buffer de acordo com o tipo de protocolo do frame em processamento;
- ARP − gerencia a tabela ARP e processa os pacotes do tipo ARP;
- Tx e Rx Buffer − memória de 32KB para armazenar pacotes do tipo IP na transmissão (Tx Buffer) e recepção (Rx Buffer);
- IP/ICMP Rx Data Engine − verifica a informação contida no pacote IP e pode descartar o pacote caso haja algum erro (tal como erro de checksum), encaminhar o pacote para o módulo ICMP caso o pacote for do tipo ICMP ou então transmitir para um *buffer* externo onde o firmware irá buscar o pacote para processá-lo no caso do pacote ser do tipo TCP;

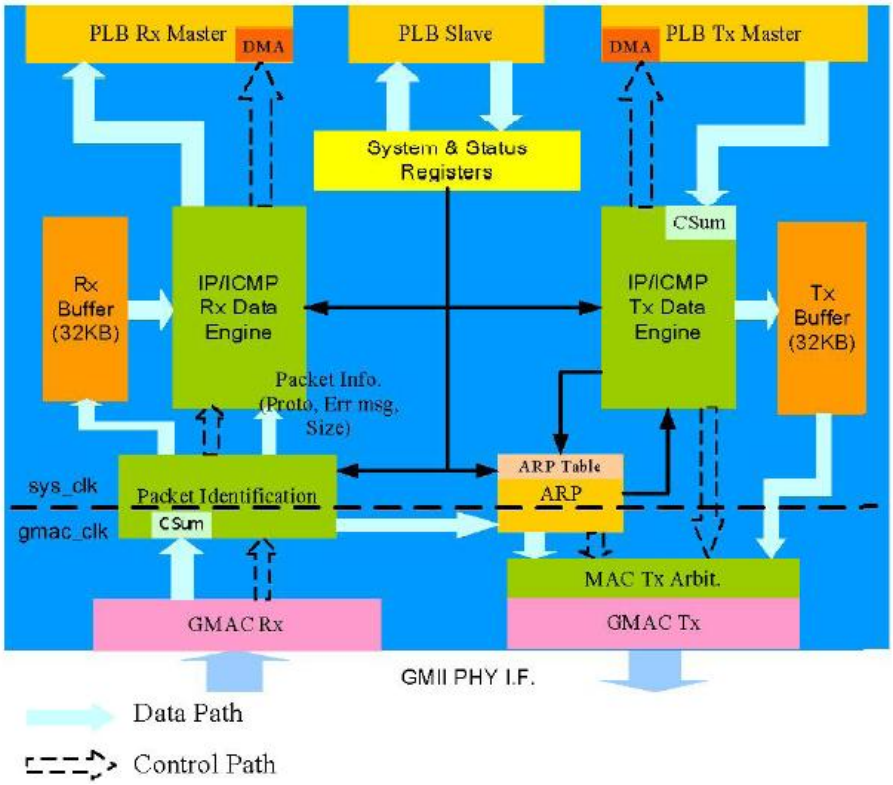

Figura 4.5: Diagrama de blocos do hardware do TCP/IP Offload Engine

- IP/ICMP Tx Data Engine – automaticamente gera informações de checksum e identificação do pacote que está sendo transmitido, sendo que o firmware gera o restante da informação contida nos cabeçalhos TCP e IP;
- Gigabit MAC Tx Arbiter – gerencia as requisições de envio de pacotes pela interface Gigabit Ethernet;
- System and Status Registers – armazena informações do sistema tais como endereço MAC, IP, gateway e máscara de rede, entre outras informações;
- PLB Master/Slave – realiza a interface entre o TOE e o barramento PLB que compõem o sistema em conjunto com um processador PowerPC onde o firmware é processado;

A arquitetura apresentada foi implementada sobre a tecnologia FPGA e sintetizada para o FPGA Virtex II PRO (XC2VP50) da Xilinx, ocupando 116 BRAMs e 5219 SLICEs. Foi efetuado um experimento com pacotes de tamanho igual à 1.500 bytes e o throughput alcançado foi de 239,34 na transmissão e 296,32 no recebimento. Foram realizados experimentos com outros tamanhos maiores de pacotes, mas os melhores resultados foram alcançados nos pacotes de 1.500 bytes.

4.3 Comparativo dos trabalhos correlatos

A tabela 4.1 resume as principais características dos trabalhos correlatos apresentados neste capítulo.

Cabe salientar que, dentre os trabalhos apresentados nesse capítulo, o trabalho apresentado por WU (WU et al, 2006) possui características muito parecidas quanto à implementação e tarefas realizadas em relação à arquitetura do iNetCore apresentado nessa

Tabela 4.1: Comparativo dos trabalhos correlatos

Referência	Tecnologia	Processador	Hardware + Software	Prototipação	Throughput (Mbps)
PAPA	ASIP	RISC	SIM	18nm	2.500
BOKAI	ASIC/FPGA	NIOS II	SIM	Altera	100
DOLLAS	ASIC/FPGA	-	NÃO (somente HW)	XCV8000	700
YUSUF	ASIC/FPGA	-	NÃO (somente HW)	XC2V8000	24,75
WU	ASIC/FPGA	PowerPC-405	SIM	XC2VP50	296,32

dissertação. O modelo de arquitetura, combinando elementos em hardware com um software específico para processamento do restante das tarefas da pilha TCP/IP, também é utilizado no iNetCore, diferenciando-se do TOE apresentado por WU (Wu:Giga-06) no que tange às tarefas realizadas em hardware ou em software, e algumas outras características específicas que serão detalhadas no restante da dissertação. Sendo assim, o trabalho apresentado por WU pode ser usado como comparativo quanto aos recursos de FPGA utilizados e ao throughput alcançado pelas duas implementações. Estes resultados são relatados na seção 6.3 .

5 DESENVOLVIMENTO DO INETCORE

Como visto no capítulo 4, os trabalhos citados fazem uso de diversas técnicas para diminuir o custo de processamento da pilha TCP/IP. No entanto, as arquiteturas propostas em hardware já existentes não são configuráveis para diferentes requisitos de aplicações. Uma das inovações desse trabalho diz respeito à customização da pilha TCP/IP em hardware em relação ao número de conexões TCP a serem gerenciadas, tamanho da tabela ARP e tamanho da memória compartilhada. Todos esses parâmetros são configuráveis pelo usuário antes da síntese da arquitetura para FPGA.

Devido à sua origem histórica, até pouco tempo atrás o modelo TCP/IP era realizado apenas em software rodando sobre sistemas com recursos computacionais suficientes para sua operação. No entanto, com o aumento na escala de integração de circuitos eletrônicos digitais, a possibilidade de encapsular as suas funcionalidades em hardware tornou-se presente, o que permitiria aos dispositivos desta espécie estarem em contato com a Internet. Naturalmente, esse tipo de abordagem traz novos problemas, como a pouca disponibilidade ou mesmo a indisponibilidade de certos recursos, principalmente memória e suporte do sistema operacional. Tais dificuldades devem ser levadas em conta para a discussão da viabilidade e dos ganhos da implementação, ou seja, a conexão de um sistema embarcado com a Internet deve ter vantagens que compensem seu esforço de projeto e concepção.

Em primeiro lugar, uma necessidade criada pela própria evolução da tecnologia é a comunicação entre os sistemas embarcados, dado que, com o aumento do número de funcionalidades, os dados processados por estes sistemas têm valor simbólico cada vez maior, ou seja, mais informação aglutinada, sendo relevantes para serem compartilhadas com outros sistemas. Além disso, circuitos de maior complexidade são capazes de disponibilizar serviços e, adicionalmente, solicitá-las.

Dentro desta mesma linha, está a comunicação do circuito com seus operadores, importante principalmente para o controle de sistemas de atuação crítica. Em Satish (2001), é ilustrado um sistema que gerencia um gerador de energia elétrica, utilizado para monitorá-lo, configurá-lo e operá-lo de maneira segura, através de um mecanismo de login de dois níveis. Essas operações podem ser feitas através da Internet utilizando-se como interface universal um simples navegador web. Um outro exemplo diz respeito a impressoras, hoje em dia já equipado com interface de rede e servidores HTTP embarcados, podendo ser diretamente acessadas através de um IP próprio por uma rede local.

Há ainda equipamentos cujo projeto possui a finalidade de conexão com a Internet, e cujas exigências de desempenho e tamanho tornam implementações por software em microcontroladores inadequadas. O exemplo mais palpável desse tipo de exigência é a telefonia celular, que hoje se torna um terminal de convergência de informações, oriundas quase sempre da Internet.

Uma segunda abordagem já discutida, diz respeito ao aumento das taxas de transmis-

são. As implementações em software do modelo TCP/IP para processadores de propósito geral em breve não conseguirão lidar com grandes velocidades nos meios físicos de rede. Para combater esta situação, é proposto um módulo funcional TCP/IP (a exemplo do que já acontece com processadores que possuem unidade específica para tratamento de ponto flutuante, por exemplo) dirigido a ganho de desempenho, para acelerar os processamentos sem sobrecarregar a unidade central.

Neste sentido, a presente dissertação apresenta a arquitetura do iNetCore, uma arquitetura da pilha TCP/IP implementada totalmente em hardware, diferenciando-se de outras arquiteturas devido à configurabilidade de características internas da pilha TCP/IP em relação a certos parâmetros que se tornam essenciais para alcançar os requisitos exigidos por certas aplicações que venham utilizar a pilha para comunicação com a Internet.

5.1 O Mapeamento para Hardware

O TCP/IP é um modelo elaborado com fins de comunicação. Assim sendo, pressupõe a existência de pelo menos duas entidades computacionais dentro de um mesmo sistema. Para fins de construção, no entanto, o módulo pode ser isolado levando-se em conta que este será capaz de receber dados tanto de um nível superior (aplicações) como de um nível inferior (meio físico). Em (DUNKELS, 2007), dois esquemas são apresentados: um chamado de processamento de entrada (pacotes entrantes), onde o dado está vindo do meio físico (Figura 5.1(a)) e outro designado como processamento de saída, onde os dados provêm da aplicação (Figura 5.1(b)) (DUNKELS, 2007). Dependendo do esquema, as entradas e saídas do iNetCore podem ser de uma ou outra natureza.

Para que seja modelado em hardware, uma abstração dessa ambigüidade precisa ser feita, o que não é muito complexo. Como os protocolos, os quais definem a forma como os pacotes devem ser montados e processados para serem transmitidos pela rede, apresentam uma característica seqüencial, a implementação desses protocolos em hardware através da manipulação de máquinas de estados se torna bastante viável. E como cada protocolo que compõe a pilha TCP/IP tem características bem definidas, a concepção de uma arquitetura modularizada em hardware em razão dos protocolos que serão processados também se torna bastante interessante. Pensou-se então em um hardware dedicado, denominado *iNetCore*, que recebe dados da camada física e da camada de aplicação, dividido internamente em submódulos definidos pela própria característica modularizada da pilha TCP/IP, onde cada submódulo seria composto pelos elementos necessários para realizar o processamento de um dos protocolos da pilha TCP/IP.

Assim, a entrada do iNetCore será uma informação dividida em pacotes, em um formato reconhecido pelo conjunto de protocolos que o compõem. O iNetCore recebe um desses pacotes, o qual contém a informação sobre seu tipo e propósito, ou seja, se deve subir ou descer na pilha. Tais características, detectadas com a leitura dos seus cabeçalhos pelas máquinas de estados, em um nível de abstração mais elevado, determinam como o pacote deve ser manipulado ou, em um outro nível já mais inferior, o submódulo para o qual deve ser direcionado.

Com base nesse mecanismo, optou-se pelo uso de um *buffer* (memória compartilhada) onde o pacote entrante é armazenado para que os submódulos possam acessá-lo e efetuar o processamento. O resultado desta manipulação será um novo dado em um formato específico para outra camada (camada de aplicação, por exemplo), ou uma resposta a uma solicitação na forma de um novo pacote, o qual será gravado neste *buffer*, caracterizando assim a saída do sistema.

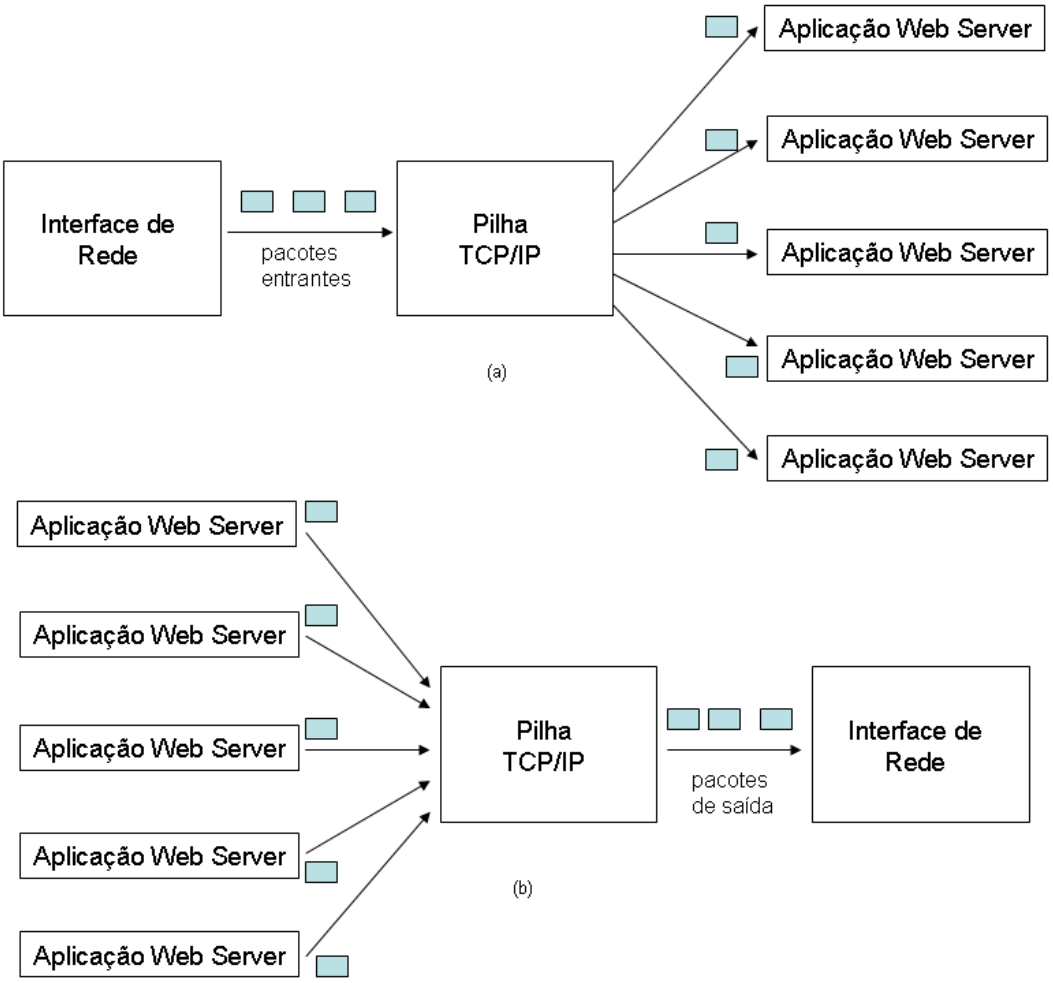

Figura 5.1: (a) Processamento de entrada e (b) saída de fluxo de rede

5.2 A arquitetura do iNetCore

Como já foi implicitamente levantado, cada protocolo que compõe o modelo TCP/IP pode ser implementado por um submódulo de hardware, invocado de acordo com o tipo de pacote que é buscado do *buffer*. Além disso, módulos combinacionais adicionais, como o verificador de checksum, também entram na arquitetura global do sistema. Ainda, em nível de hardware, um *buffer* pode ser tratado diretamente como uma memória, compartilhada entre submódulos e, em princípio, acessível às camadas física (através da interface MAC) e de aplicação (através da interface da camada de aplicação).

Como citado, o objeto de estudo e desenvolvimento é o iNetCore, para o qual foi definida toda uma arquitetura específica. Porém, para integrá-lo a um determinado sistema de computação, é necessária a implementação das interfaces fazendo com que ele possa ser facilmente integrado com qualquer sistema que siga o Modelo TCP/IP.

Além disso, durante o projeto da arquitetura do iNetCore devem ser tomadas uma série de decisões de projeto em relação aos elementos de hardware que irão compor a arquitetura final, levando-se em conta as funcionalidades presentes em uma pilha TCP/IP convencional e quais dessas funcionalidades são devidamente viáveis de serem implementadas em hardware. A próxima seção aborda essas questões.

5.2.1 Limitações de Projeto

Pensando na implementação da pilha TCP/IP em hardware, há diversas características no projeto da pilha TCP/IP que devem ser levadas em consideração devido à limitação dos recursos de hardware disponíveis, tais como memória e recursos de hardware (número de portas lógicas ou elementos configuráreis, no caso do hardware em questão ser um FPGA). A seguir, são detalhadas algumas dessas limitações e as decisões de projeto que foram tomadas para a caracterização da arquitetura final do iNetCore.

5.2.1.1 Número de conexões TCP

O número de conexões TCP que podem ser gerenciado influencia diretamente na quantidade de informações que devem ser armazenadas para o gerenciamento de cada conexão. Logo, deve-se analisar a quantidade de módulos de memória disponível no sistema, sejam estes módulos internos ou externos. Para facilitar esse entendimento, segue abaixo, na tabela 5.1, as informações armazenadas para cada conexão TCP e o número de bits necessários para o registro dessas informações.

Tabela 5.1: Dados que devem ser armazenados para cada conexão TCP

Dado	**Descrição**	**Nº de bits**
ripaddr	Endereço IP do dispositivo remoto.	32
lport	Número da porta TCP no dispositivo local.	14
rport	Número da porta TCP no dispositivo remoto.	14
rcv_nxt	Número de seqüência do próximo segmento TCP.	32
snd_nxt	Número de seqüência do último segmento TCP recebido.	32
len	Quantidade de dados enviados no último segmento TCP.	11
mss	Tamanho atual máximo de segmento da conexão.	11
initialmss	Tamanho inicial máximo de segmento da conexão.	11
sa	Variável utilizada no cálculo do tempo de retransmissão de segmento.	11
sv	Variável utilizada no cálculo do tempo de retransmissão de segmento.	11
rto	Tempo de retransmissão de segmento.	10
tcpstateflags	Valores de flag da conexão TCP.	4
timer	Relógio do tempo de retransmissão de segmento.	11
nrtx	Número de retransmissões do último segmento enviado.	11

Portanto, para cada conexão TCP, é necessário o armazenamento de 215 bits (soma dos valores citados na terceira coluna da tabela 5.1), seja em registradores ou em módulos de memória (BRAMs, por exemplo). Na seção 5.3 são apresentados os resultados de síntese da arquitetura variando-se a quantidade de conexões TCP gerenciadas pelo iNetCore.

5.2.1.2 Tamanho da Tabela ARP

Analogamente ao número de conexões TCP, a quantidade de memória utilizada para armazenar as informações da tabela ARP depende da profundidade dessa tabela em relação ao número de endereços que se quer gerenciar. Abaixo, na tabela 5.2, seguem os dados armazenados para cada registro na Tabela ARP.

Tabela 5.2: Dados armazenados para registro na Tabela ARP

Dado	**Descrição**	**Nº de bits**
ip	Endereço IP.	32
eth	Endereço físico.	48
timer	Relógio do tempo de vida do registro.	14

Portanto, para cada registro na tabela ARP, é necessário o armazenamento de 94 bits. Em uma pilha TCP/IP tradicional costuma-se utilizar uma tabela ARP de até 10 registros.

5.2.1.3 Tamanho da memória para armazenamento de pacotes

A falta de espaço de memória também torna-se uma preocupação se múltiplos datagramas devem ser recebidos da rede e remontados em caso de chegada fora de ordem.

Por decisão de projeto, optou-se por utilizar uma memória (*buffer*) que pudesse comportar a quantidade máxima de dados que um pacote Ethernet pode ter, ou seja, 1524 (um mil, quinhentos e vinte e quatro) bytes. Assim, o reordenamento de pacotes não é realizado pelo iNetCore. Essa decisão foi tomada devido ao fato de que, em um ambiente de rede gigabit, a transferência de dados na taxa de 1 Gbps dificilmente vai ser alcançada em um ambiente de conexão que não seja ponto-a-ponto. Visto que um ambiente de rede ponto-a-ponto não possui a característica permanente de perda de pacotes ou recebimento fora de ordem, característica esta pertencente a um ambiente de rede com múltiplos roteadores pelo caminho, os pacotes não precisarão ser reordenados, pois raramente chegarão fora de ordem. Mesmo que esporadicamente cheguem fora de ordem, serão descartados e os protocolos de comunicação implementados no iNetCore buscarão estabilizar a conexão novamente através da solicitação de reenvio de pacotes perdidos.

A memória utilizada no iNetCore, além de armazenar temporariamente o pacote que está em processamento, também é utilizada para receber dados vindos da camada de aplicação. Cabe salientar que o tamanho dessa memória é configurável para o projetista, ou seja, ele pode alterar o tamanho da memória para o caso do tamanho máximo de pacote ser maior do que 1.524 (um mil, quinhento e vinte e quatro) bytes, como em aplicações que utilizam pacotes do tipo Jumbo (DYKSTRA, 1999), que podem chegar até 9.000 (nove mil) bytes. Como todo o projeto do iNetCore foi implementado sobre a tecnologia Fast Ethernet, o tamanho máximo de pacote, e por conseqüência, da memória de armazenamento, é de 1.524 (um mil, quinhento e vinte e quatro) bytes.

5.2.2 Projeto da Arquitetura do iNetCore

Nesta seção serão relatados detalhes da arquitetura do iNetCore e como os dados são manipulados.

A figura 5.2 exibe as diferenças existentes entre uma abordagem de execução da pilha TCP/IP na forma de um software e a abordagem adotada pela arquitetura do iNetCore. Na abordagem em software, realizada nos sistemas operacionais convencionais, as aplicações e todas as operações da pilha TCP/IP são executadas sobre o sistema operacional em uma plataforma de software. Já na abordagem adotada no desenvolvimento da arquitetura do iNetCore, somente as aplicações são executadas em uma plataforma de software, com todas as tarefas da pilha TCP/IP realizadas por um hardware dedicado.

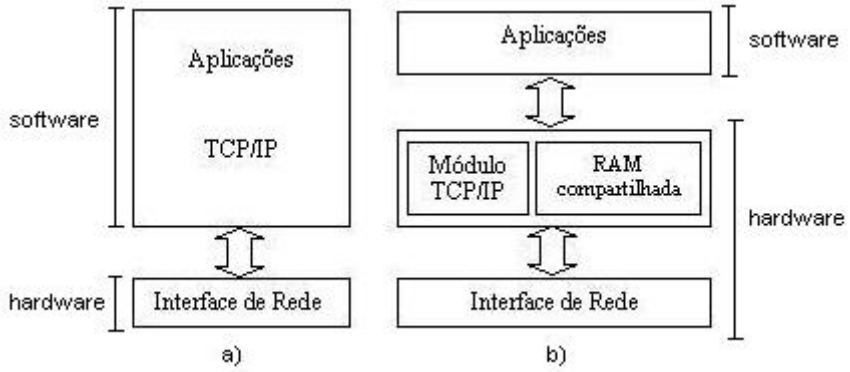

Figura 5.2: (a) TCP/IP em software e (b) Arquitetura Proposta

A arquitetura do iNetCore é baseada na idéia de um espaço de armazenamento compartilhado entre uma interface de rede, o iNetCore e as aplicações que estão rodando sobre uma plataforma de software.

A interface de rede repassa os pacotes vindos da camada física para o íNetCore, os quais são armazenados na memória compartilhada. Uma observação importante é que o iNetCore é independente da interface de rede que está sendo usada, por exemplo: Ethernet, Wireless, etc.

O iNetCore realizará o processamento dos protocolos IP, ICMP, ARP e TCP. Tarefas como estabelecimento de conexão, recepção/transformação de dados, checksum, gerenciamento de conexões, entre outras, são todas realizadas pelo iNetCore, acelerando o processamento e reduzindo a carga de CPU necessária comparando-se com a abordagem onde todo o processamento TCP/IP era efetuado em software.

Nesta arquitetura, as interrupções causadas pelo envio/recebimento de pacotes, ou seja, operações de I/O, não existem mais. Todo o gerenciamento do fluxo de dados Internet é realizado pelo iNetCore e apenas as funções de mais alto nível, mais especificamente dos serviços que estão na camada de aplicação, são realizadas em software.

A característica modular da composição da pilha de protocolos do modelo TCP/IP permitiu a divisão da arquitetura proposta em módulos conforme o tipo de protocolo. Com isso, uma importante peculiaridade aparece: a configurabilidade da arquitetura conforme o serviço que se deseja rodar sobre a pilha TCP/IP. Por exemplo, serviços que não desejam servir-se de funções implementadas pelo protocolo ICMP, poderão dispensar a presença do módulo na arquitetura e com isso o projetista poderia simplesmente extrair o módulo ICMP de forma a configurá-la especificamente para a aplicação em uso.

A arquitetura interna do iNetCore pode ser representada de acordo com a Figura 5.3.

O módulo de controle principal (MCP) é o responsável pela coordenação e atuação efetiva do iNetCore. Um fluxograma que descreve o funcionamento básico do bloco de controle principal é demonstrado na figura 5.4. A máquina de estados do MCP possui 57 estados. No MCP são feitas as primeiras verificações dos dados, como o checksum do

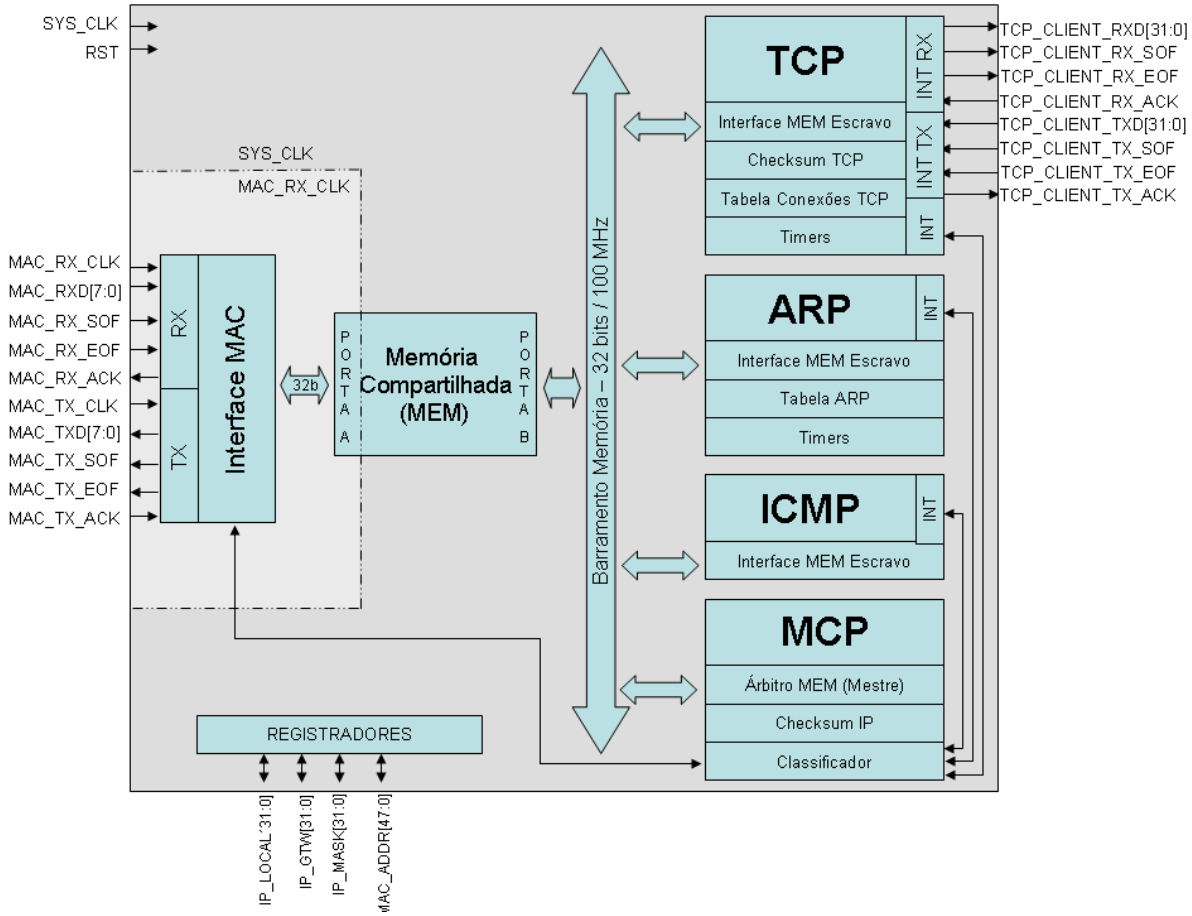

Figura 5.3: Arquitetura do iNetCore

cabeçalho IP, verificação da versão do protocolo e, em se tratando de um pacote válido, identificação do tipo de protocolo (Classificador) e direcionamento para os respectivos submódulos (ou o seu descarte, em caso contrário). Para isto, este bloco se vale da propriedade de que um pacote com determinada identidade segue seqüencialmente passos correspondentes até o fim do seu processamento, ou seja, transita entre estados guiados por certas condições dependentes da especificação do módulo que o está processando. O módulo MCP também controla o acesso ao barramento da memória compartilhada.

Os outros módulos (ICMP, ARP e TCP) são responsáveis pelo processamento de seus respectivos protocolos.

Particularmente, o Módulo ARP, mantém registrada uma tabela, que mapeia um endereço físico (MAC) com um endereço lógico (IP). Além disso, possui internamento um temporizador que controla o tempo em que um registro da Tabela ARP é válido. O fluxograma que modela a máquina de estados do Módulo ARP pode ser visualizado na figura 5.5. A máquina de estados do Módulo ARP possui 51 estados. O Módulo ARP possui basicamente três operações: atualização da Tabela ARP com dados dos pacotes entrantes; montagem de pacotes do tipo "ARP Request", que são usados para requisitar à rede um endereço MAC com base em um endereço IP; e processamento de pacotes do tipo "ARP Reply", que são os pacotes retornados após um pedido feito através do pacote "ARP Request".

O Módulo TCP é formado por uma máquina de estados implementada com base nas máquinas de estados de um cliente TCP (figura 2.6) e de um servidor TCP (figura 2.5).

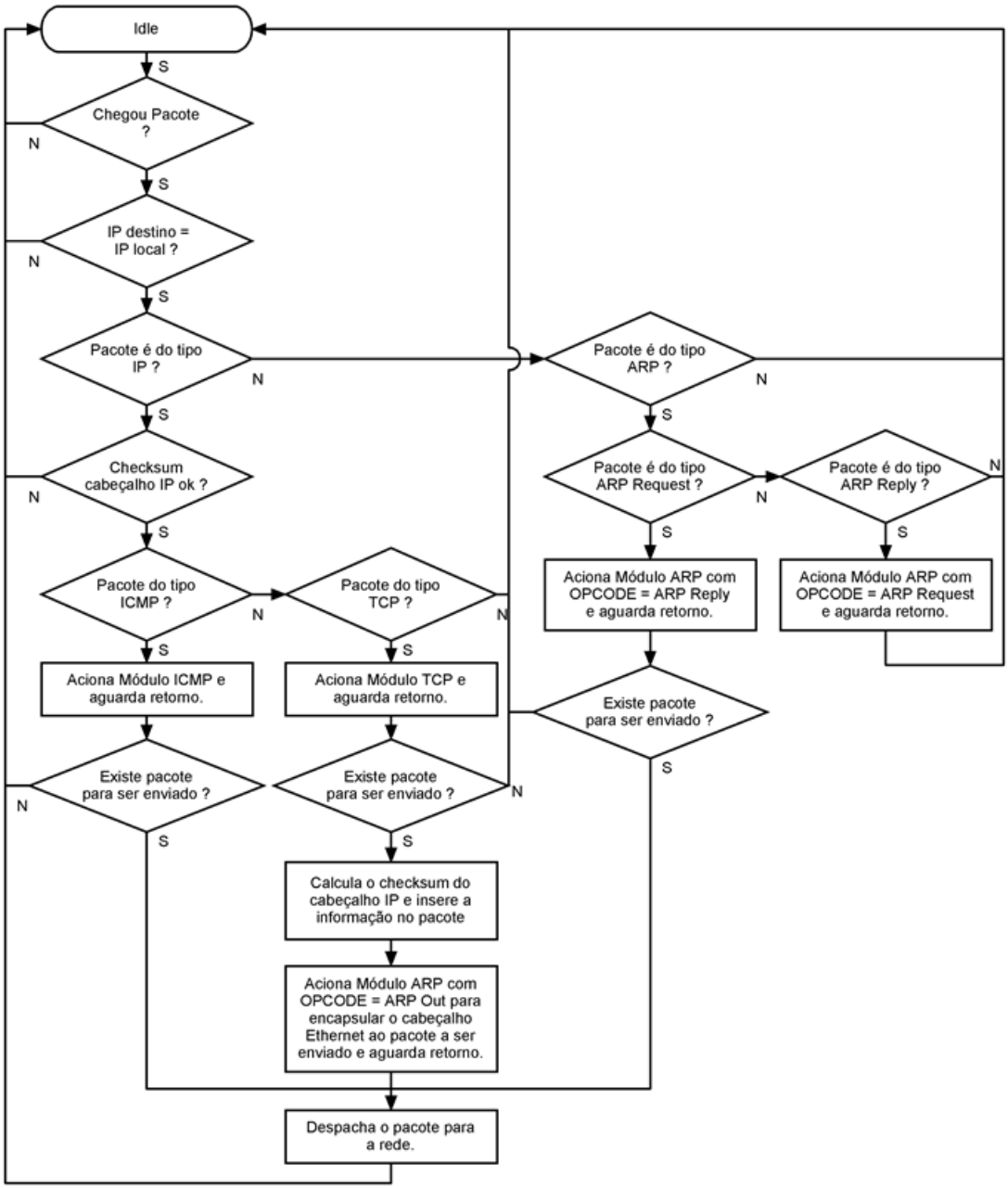

Figura 5.4: Fluxograma do Módulo de Controle Principal - MCP

Com isso, o iNetCore pode ser usado por diversos serviços na camada de aplicação, por exemplo: servidor ou cliente HTTP, servidor ou cliente FTP, entre outros. A máquina de estados implementada no Módulo TCP possui 154 estados.

O Módulo TCP é responsável por todo o controle de estabelecimento e finalização de conexões, assim como a passagem de dados para a camada de aplicação por meio das interfaces INT TX e INT RX. Para controlar as conexões TCP, o módulo TCP mantém o estado das conexões em registradores internos. Também possui temporizadores internos para controlar o tempo de vida de um pacote TCP na rede, além do tempo de vida das conexões TCP. É o módulo TCP que verifica o checksum de um pacote entrante, descartando-o em caso de erro ou processando o conteúdo do pacote se o checksum estiver correto. Também calcula o checksum TCP de um pacote que está sendo montado

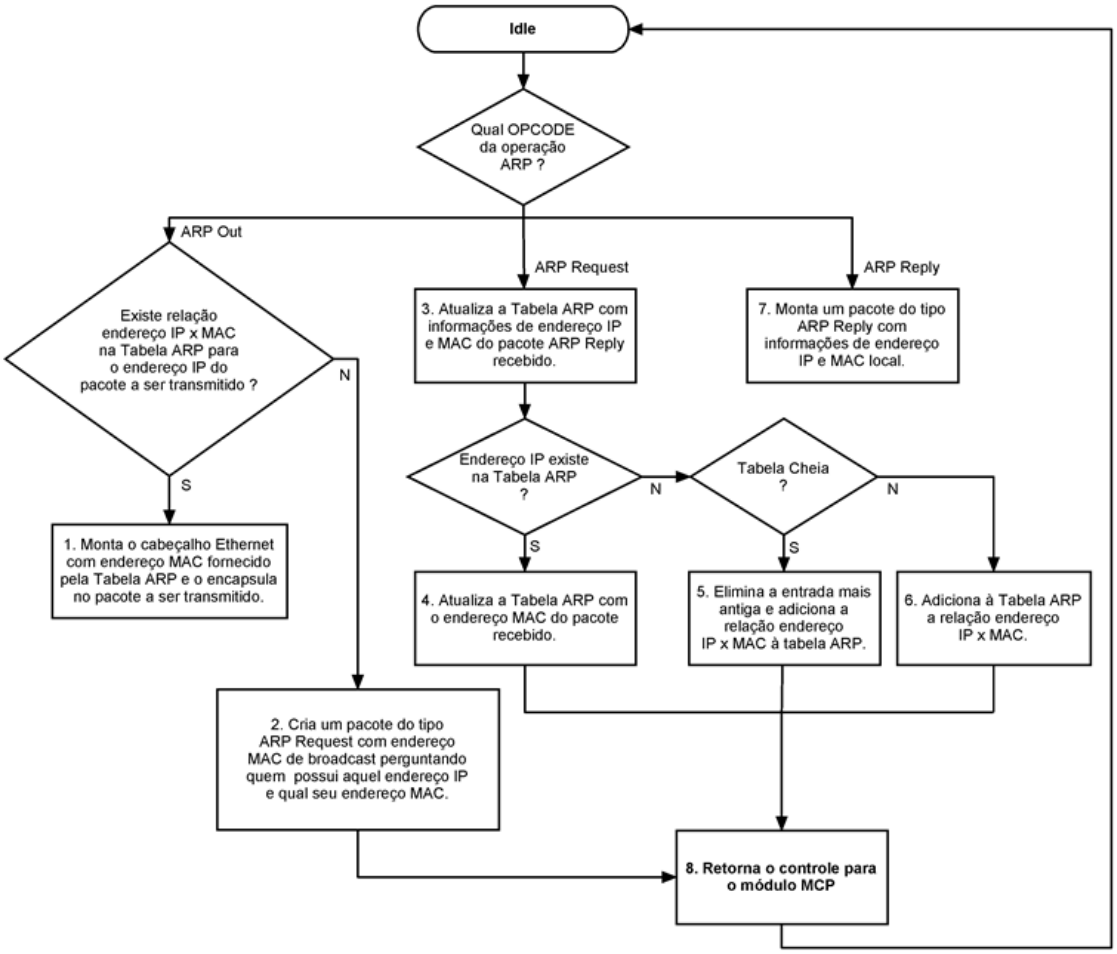

Figura 5.5: Fluxograma do Módulo ARP

para ser enviado para a rede.

O Módulo ICMP é capaz de montar pacotes do tipo "ICMP Echo Response"para ser retransmitido à rede. O fluxograma que modela a máquina de estados do Módulo ICMP pode ser visualizado na figura 5.6. A máquina de estados do Módulo ICMP possui 10 estados.

Outra propriedade bastante interessante para a implementação do TCP/IP em hardware é a manutenção do caráter seqüencial mesmo com a descendência no nível hierárquico, ou seja, a invocação de um submódulo é uma conveniência de projeto que permite interpretar que o próximo estado a partir da sua chamada está inserido em um outro nível de abstração, sem que, no entanto, haja quebras de continuidade em relação à entrada do pacote no módulo.

Por fim, as operações elementares para uma arquitetura embarcada TCP/IP, apesar da complexidade presente no gerenciamento de conexões pelo módulo TCP, são relativamente simples, se resumindo a operações de escrita e leitura na memória. Visto que os submódulos são implementados através de máquinas de estados, que guiam o processo de envio e recebimento de pacotes, as operações são realizadas durante a evolução através dos estados das máquinas. Como a memória compartilhada se trata do único meio de armazenamento disponível, dados de cabeçalho são buscados separadamente de acordo com as necessidades, e também as escritas que fornecem os resultados, alterando o estado inicial da memória.

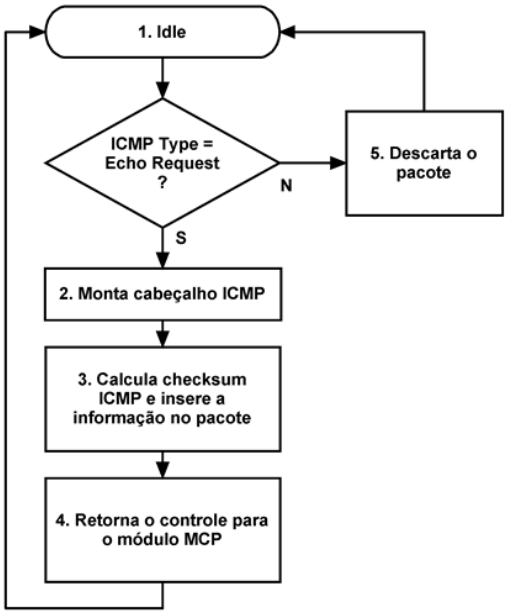

Figura 5.6: Fluxograma do Módulo ICMP

O iNetCore foi descrito em linguagem VHDL e apresenta parâmetros de configuração que devem ser setados antes da síntese da arquitetura para a tecnologia desejada. Entre as configurações possíveis, constam:

- número de conexões TCP simultâneas a serem gerenciadas;
- tamanho do *buffer* (memória compartilhada) que irá armazenar o pacote em processamento;
- tamanho da tabela ARP que realiza o mapeamento entre endereço físico (MAC) e endereço de rede (IP);
- configuração do endereço IP, gateway, endereço físico (MAC) e máscara de sub-rede do *host*.

Como já citado, o tamanho do *buffer*, o número de conexões TCP e o tamanho da tabela ARP são parâmetros que devem ser setados antes da síntese do módulo para a tecnologia de hardware escolhida, seja ela FPGA ou ASIC. Já as configurações de endereço IP, gateway, endereço físico e máscara de sub-rede são setadas através de portas de entrada do iNetCore.

Uma observação que deve ser feita é em relação ao número de conexões que o módulo TCP deverá gerenciar, definido pelo usuário da arquitetura antes da síntese da mesma. As informações de cada conexão são armazenadas em registradores internos ao módulo TCP. Portanto, quanto maior for o número de conexões simultâneas a serem gerenciadas pelo módulo TCP, maior será o gasto em área. O mesmo vale para o tamanho da tabela ARP a ser usada, onde o tamanho em área do módulo ARP será proporcional ao número de entradas da tabela.

5.2.2.1 *Estrutura de Memória*

Na arquitetura do iNetCore, a interface de rede é responsável por montar os pacotes vindos da camada física e colocá-los na memória compartilhada. O modo como os dados do pacote são escritos na memória, em relação ao tamanho da palavra, é que vai definir a estrutura das máquinas de estados que definem o comportamento de cada submódulo interno da arquitetura do iNetCore. Neste sentido, duas implementações diferentes do iNetCore foram desenvolvidas. O que as diferencia é o tamanho da palavra escrita ou lida da memória (16 e 32 bits) e conseqüente descrição das máquinas de estados.

A memória compartilhada presente na arquitetura do iNetCore é usada para armazenar o pacote recebido da rede. A interface de rede é responsável por receber o fluxo do pacote vindo do MAC e armazená-lo na memória compartilhada.

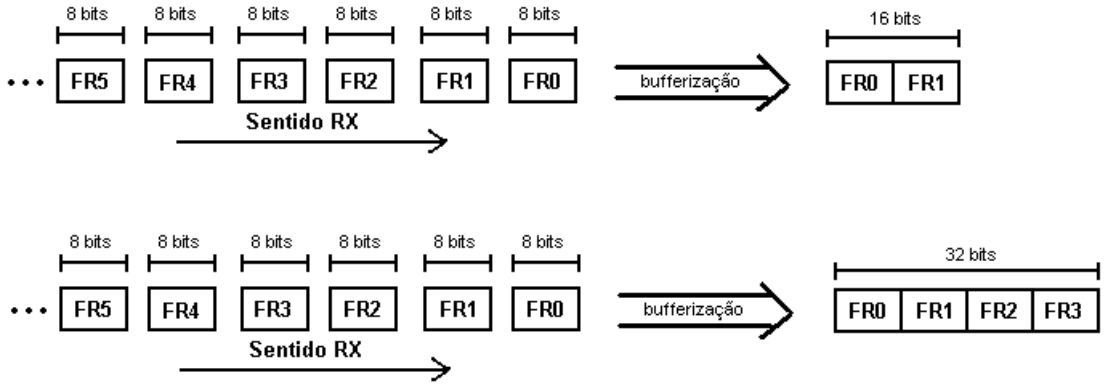

Figura 5.7: Esquema de bufferização dos dados do pacote para escrita na memória compartilhada

A arquitetura usada do iNetCore - 16 ou 32 bits - é que define a forma como os dados do pacotes são escritos na memória. Para formar a palavra a ser escrita na memória, os dados vindos do MAC (na forma de byte) são concatenados e escritos na memória. A figura 5.7 demonstra a operação realizada pela interface MAC sobre o fluxo do pacote recebido nas implementações da arquitetura em 16 bits (figura 5.7 (a)) e 32 bits (figura 5.7 (b)). Na figura, FR0, FR1, FR2, FR3, FR4 e FR5, representam os primeiros 6 bytes do pacote (frame) que está sendo recebido. As palavras de 16 ou 32 bits, à medida em que vão sendo formadas pelo submódulo Interface MAC em suas respectivas versões de arquitetura, são então armazenadas na memória compartilhada para serem acessadas pelos outros submódulos do iNetCore. Os submódulos mantêm a informação da posição exata de cada byte do pacote na memória compartilhada e, dessa maneira, realizam o processamento desse pacote.

5.2.2.2 *Cálculo do Checksum - Módulo Checksum*

Como já citado, o cálculo do Checksum é um fator crítico que afeta a velocidade de processamento dos pacotes. O checksum é executado para verificar a integridade dos dados nos pacotes da camada de rede (protocolo IP) e da camada de transporte (protocolo TCP e ICMP). Na camada de rede, o checksum é calculado sobre o cabeçalho IP, enquanto na camada de transporte ele é calculado sobre todo os dados que compõem o pacote referente ao tipo de protocolo, TCP ou UDP, sendo "varrido"todo o pacote (cabeçalho + dados). As principais etapas do algoritmo do checksum são os seguintes:

- Octetos adjacentes dos dados (a serem usados no cálculo do checksum) são concatenados para formar uma palavra de 16 bits;
- É efetuada uma soma binária com "complemento de um"sobre estas palavras de 16 bits;
- O complemento de um do resultado da soma de todos os dados cobertos são escritos no campo correspondente do cabeçalho do pacote;

No sentido inverso, para checar o checksum de um determinado protocolo, a soma é realizada sobre os dados que se deseja cobrir, inclusive o campo de checksum. Se o resultado consistir em "todos 1", ou "FFFF"em hexadecimal, a checagem está correta e indica que não há erro naqueles dados.

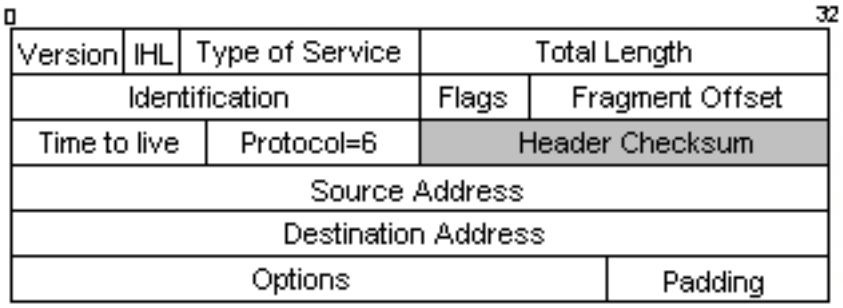

Figura 5.8: Formato do cabeçalho IP

Um exemplo de cálculo de checksum é demonstrado na figura 5.9. Neste exemplo, é efetuado o cálculo de checksum do cabeçalho IP (figura 5.8) de um pacote. Na figura 5.9, cada linha do cabeçalho contém 32 bits. Para o cálculo do checksum, esses bits são agrupados de 16 em 16 bits. O resultado final é adicionado ao campo "Header Checksum"do cabeçalho IP.

Na arquitetura do iNetCore, o checksum é efetuado sobre dois fluxos distintos. No fluxo de dados dos pacotes recebidos pela interface de rede, o checksum é efetuado pelo submódulo Checksum RX, que realiza o cálculo do checksum paralelamente à escrita dos dados na memória compartilhada. Já no fluxo dos dados recebidos pela interface denominada INT TX, presente no submódulo TCP, o checksum é efetuado diretamente pelo submódulo TCP calculado paralelamente à escrita dos dados na memória compartilhada.

5.2.2.3 Interface do iNetCore com o MAC e o cliente TCP

Como pode ser visualizado na figura 5.3, o iNetCore apresenta duas interfaces para comunicação com o MAC e com o cliente TCP. A tabela 5.3 mostra os sinais que formam essas duas interfaces.

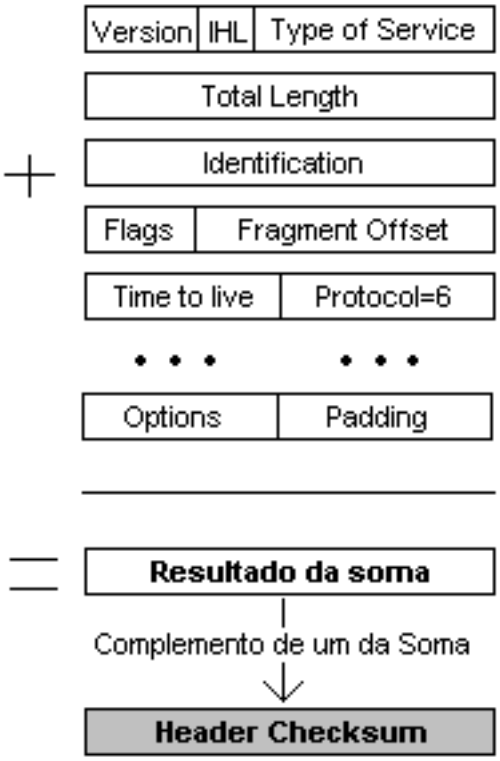

Figura 5.9: Exemplo de cálculo do checksum do Cabeçalho IP

Tabela 5.3: Sinais externos do iNetCore

Sinal	**Direção**	**Gerador**	**Descrição**
MAC_RX_CLK	Entrada	MAC	Relógio de 12,5 MHz para recepção de quadros vindos do MAC
CLK_SYS	Entrada	TCP_CLIENT	Relógio do sistema (mesmo clock do barramento onde o iNetCore está liga)
RST	Entrada	TCP_CLIENT	Reset do iNetCore
MAC_RXD[7:0]	Entrada	MAC	Canal que recepciona os dados em formato de bytes vindo do MAC
MAC_RX_SOF	Entrada	MAC	Indica o início do recebimento de um quadro vindo do MAC
MAC_RX_EOF	Entrada	MAC	Indica o fim do recebimento de um quadro vindo do MAC

continua na próxima página

Tabela 5.3: continuação da página anterior

Sinal	Direção	Gerador	Descrição
MAC_RX_ABORT	Entrada	MAC	Indica que o quadro recebido do MAC contém erros
MAC_TX_CLK	Entrada	MAC	Relógio de 12,5 MHz para transmissão de quadros ao MAC
MAC_TXD[7:0]	Saída	iNetCore	Canal de transmissão de dados em formato de bytes para o MAC
MAC_TX_SOF	Saída	iNetCore	Indica o início da transmissão de um quadro para o MAC
MAC_TX_EOF	Saída	iNetCore	Indica o fim da transmissão de um quadro para o MAC
MAC_TX_ACK	Entrada	MAC	Indica ao iNetCore iniciar a transmissão do primeiro byte do quadro
TCP_CLIENT_TXD[31:0]	Entrada	TCP_CLIENT	Canal que recepciona os dados transmitidos pelo TCP_CLIENT
TCP_CLIENT_TX_SOF	Entrada	TCP_CLIENT	Indica o início da transmissão de um quadro pelo TCP_CLIENT
TCP_CLIENT_TX_EOF	Entrada	TCP_CLIENT	Indica o fim da transmissão de um quadro pelo TCP_CLIENT
TCP_CLIENT_TX_ACK	Saída	iNetCore	Indica ao TCP_CLIENT iniciar a transmissão do primeiro byte do quadro
TCP_CLIENT_RXD[31:0]	Saída	iNetCore	Canal de transmissão de dados para o TCP_CLIENT
TCP_CLIENT_RX_SOF	Saída	iNetCore	Indica o início da transmissão de um quadro para o TCP_CLIENT
TCP_CLIENT_RX_EOF	Saída	iNetCore	Indica o fim da transmissão de um quadro para o TCP_CLIENT

continua na próxima página

Tabela 5.3: continuação da página anterior

Sinal	Direção	Gerador	Descrição
TCP_CLIENT_RX_ACK	Entrada	TCP_CLIENT	Indica ao iNetCore iniciar a transmissão do primeiro byte do quadro

Para realizar a tarefa de recepção de quadro Ethernet vindo do MAC, a interface implementada através do submódulo Interface MAC respeita o diagrama temporal mostrado na figura 5.10.

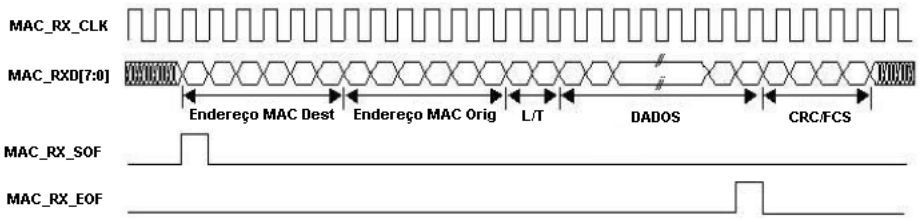

Figura 5.10: Diagrama Temporal para recepção de quadro Ethernet do MAC

Da mesma forma, para transmissão de um quadro Ethernet através do MAC, a interface implementada através do submódulo Interface MAC respeita o diagrama temporal mostrado na figura 5.11.

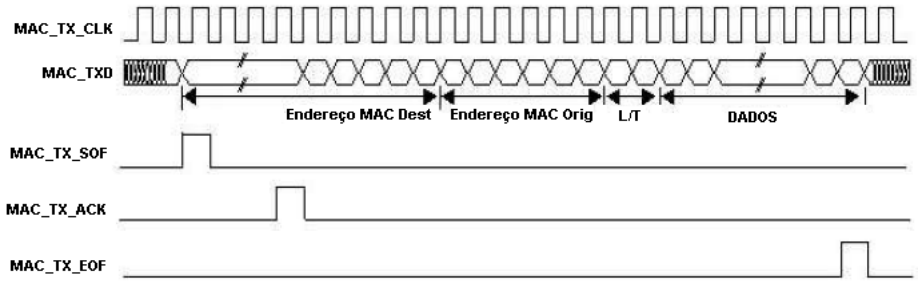

Figura 5.11: Diagrama Temporal para transmissão de quadro Ethernet para o MAC

Já para realizar a tarefa de recepção de dados vindo do cliente TCP, a interface implementada através do submódulo INT TX respeita o diagrama temporal mostrado na figura 5.12.

Da mesma forma, para transmissão de dados para o cliente TCP, a interface implementada através do submódulo INT RX respeita o diagrama temporal mostrado na figura 5.13.

5.3 Síntese da arquitetura do iNetCore para hardware ASIC e FPGA

Como discutido na seção anterior, foram desenvolvidas duas versões de arquitetura do iNetCore, em relação ao tamanho da palavra escrita ou lida da memória (16 e 32 bits) e conseqüente descrição das máquinas de estados. Essas duas versões foram analisadas

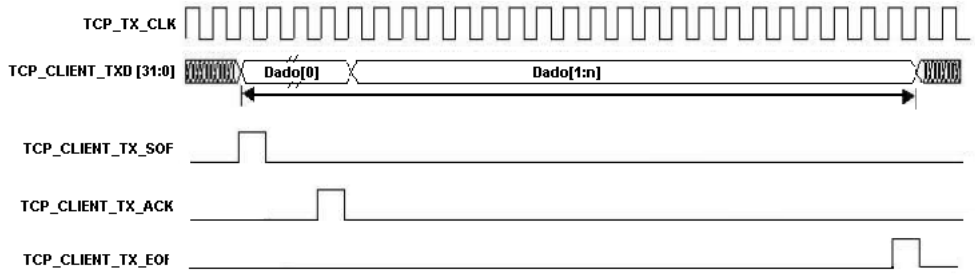

Figura 5.12: Diagrama Temporal para recepção de dados do cliente TCP

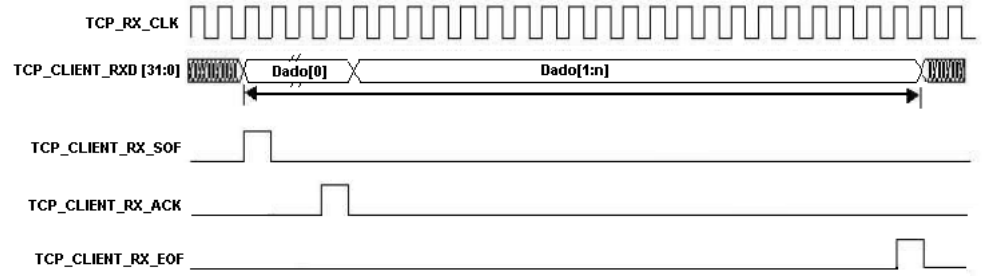

Figura 5.13: Diagrama Temporal para transmissão de dados ao cliente TCP

também variando o número de conexões simultâneas TCP que cada implementação é capaz de gerenciar. Como citado anteriormente, quanto maior for o número de conexões simultâneas a serem gerenciadas pelo módulo TCP, maior será o gasto em área. Com isso, seis descrições diferentes da arquitetura proposta foram analisadas.

Com o objetivo de comparar essas seis implementações, todas descritas em linguagem VHDL, elas foram sintetizadas para duas diferentes plataformas de hardware: ASIC e FPGA. Na síntese para ASIC foi utilizada a ferramenta Leonardo Spectum da Mentor Graphics na tecnologia de 0.18um. Na síntese para FPGA foi utilizada a ferramenta Xilinx ISE 8.1i da Xilinx, tendo como placa de prototipação a Virtex II-Pro XC2VP30-5ff896. Os resultados de área e freqüência de cada submódulo e do iNetCore por completo são exibidos nas tabelas 5.4 e 5.5, para a plataforma ASIC, e tabelas 5.6 e 5.7, para a plataforma FPGA (todas as sínteses para FPGA ocuparam uma BRAM).

Tabela 5.4: Resultados de síntese do iNetCore em uma arquitetura de 16 bits - ASIC

Conexões TCP	Fator	MCP	ICMP	ARP	TCP	TOTAL
10	Área (n.º Gates)	3.187	1.014	3.956	18.685	27.094
	Freqüência (MHz)	162,5	470,5	304,1	142,8	142,8
5	Área (n.º Gates)	3.187	1.014	3.956	14.084	22.493
	Freqüência (MHz)	162,5	470,5	304,1	142,8	142,8
1	Área (n.º Gates)	3.187	1.014	3.956	10.740	19.150
	Freqüência (MHz)	162,5	470,5	304,1	142,8	142,8

Em ambas as plataformas de hardware, os resultados alcançados pela descrição de 32

Tabela 5.5: Resultados de síntese do iNetCore em uma arquitetura de 32 bits - ASIC

Conexões TCP	Fator	MCP	ICMP	ARP	TCP	TOTAL
10	Área (n.º Gates)	2.513	504	3.038	19.348	25.670
	Freqüência (MHz)	147,4	948,6	176	92,4	92,4
5	Área (n.º Gates)	2.513	504	3.038	14.884	21.205
	Freqüência (MHz)	147,4	948,6	176	92,4	92,4
1	Área (n.º Gates)	2.513	504	3.038	10.306	16.627
	Freqüência (MHz)	147,4	948,6	176	92,4	92,4

Tabela 5.6: Resultados de síntese do iNetCore em uma arquitetura de 16 bits - FPGA

Conexões TCP	Fator	MCP	ICMP	ARP	TCP	TOTAL
10	LUTs	758	208	836	6.035	7.936
	Slices	443	120	450	3.371	4.448
	FFs	248	113	385	2.556	3.346
	Fmax(MHz)	77,5	207,7	222,8	70,9	71,2
5	LUTs	758	208	836	4.565	6.459
	Slices	443	120	450	2.541	3.612
	FFs	248	113	385	1.578	2.368
	Fmax(MHz)	77,5	207,7	222,8	70,9	71,2
1	LUTs	758	208	836	2.555	4.480
	Slices	443	120	450	1.441	2.532
	FFs	248	113	385	758	1.547
	Fmax(MHz)	77,5	207,7	222,8	70,9	71,2

bits se mostraram melhores em relação à área ocupada pelo circuito gerado, porém, como esperado, a freqüência máxima de operação alcançada foi menor que na descrição de 16 bits. Uma análise sobre o algoritmo que implementa a pilha TCP/IP revelou que a maioria das buscas de dados na memória segue um padrão de tamanho de palavra de 32 bits, caso como o endereço de rede (32 bits). Em virtude disso, a complexidade da máquina de estados dos módulos foi reduzida e, conseqüentemente, a área ocupada por esses módulos se tornou menor. Em relação à freqüência, o fato das operações efetuadas na descrição de 32 bits serem sobre palavras de 32 bits acarretou na diminuição da freqüência máxima de operação alcançada pelo circuito gerado (HAMERSKI et al, 2007).

5.4 Análise de Desempenho

Com o objetivo de verificar o desempenho da arquitetura do iNetCore sobre um conjunto específico pacotes, foi desenvolvido um ambiente de simulação de fluxo de rede. Este ambiente é composto pelo iNetCore, além de um módulo gerador de pacotes (que simula a Interface MAC num ambiente de rede real). A figura 5.14 apresenta o diagrama

Tabela 5.7: Resultados de síntese do iNetCore em uma arquitetura de 32 bits - FPGA

Conexões TCP	Fator	MCP	ICMP	ARP	TCP	TOTAL
10	LUTs	728	165	672	5.873	7.551
	Slices	424	89	373	3.289	4.448
	FFs	181	56	302	2.491	3.015
	Fmax(MHz)	59,8	414,3	159,3	58,7	58,7
5	LUTs	728	165	672	4.074	5.735
	Slices	424	89	373	2.283	3.229
	FFs	181	56	302	1.527	2.047
	Fmax(MHz)	59,8	414,3	159,3	58,7	58,7
1	LUTs	728	165	672	2.278	3.968
	Slices	424	89	373	1.282	2.240
	FFs	181	56	302	648	1.172
	Fmax(MHz)	59,8	414,3	159,3	58,7	58,7

de blocos deste ambiente.

Neste ambiente de simulação, o Gerador de Pacotes possuiu um conjunto de pacotes de comportamento conhecido. A simulação de fluxo de rede inicia através do envio, por parte do gerador, do pacote a ser processado. O iNetCore armazena o pacote na memória compartilhada e realiza o processamento através de uma série de verificações no pacote, como endereço de rede destino, versão do pacote e checksum. Ao término do processamento, se um pacote foi gerado pelo iNetCore e deve ser enviado, a interface MAC interna ao iNetCore irá despachar o pacote para a rede, no caso, enviando o pacote criado para o Gerador de Pacotes. O Gerador de Pacotes, por sua vez, irá comparar o conteúdo do pacote recebido e validará o pacote caso ele seja igual ao esperado.

O Gerador de Pacotes desenvolvido gera um conjunto de quatro pacotes, como segue:

- ArpReply (60 bytes): define um pacote do tipo ARP como retorno de uma solicitação ARPRequest à rede;
- ICMPEcho (74 bytes): Pacote de solicitação de ICMPEcho por um *host* externo;
- Syn (62 bytes): Solicitação (sync) de conexão a um serviço TCP;
- Synack (60 bytes): Confirmação (sync + ack) de uma solicitação de conexão a um serviço TCP.

Esses quatro pacotes definem um pequeno fluxo de pacotes realizado em um ambiente de rede real. As tabelas 5.8 e 5.9 exibem os resultados obtidos na simulação do fluxo de pacotes.

Como pode ser observado nas tabelas 5.8 e 5.9, os resultados da descrição de 16 bits são melhores que os da descrição de 32 bits, em relação ao tempo de computação dos quatro pacotes que simulam o fluxo de rede real. A análise foi efetuada sobre as duas descrições e para três configurações diferentes em relação ao número de conexões simultâneas TCP. O número de conexões TCP irá afetar apenas o processamento do pacote Syn, que é um pacote do tipo TCP. Embora o número de ciclos de relógio necessários

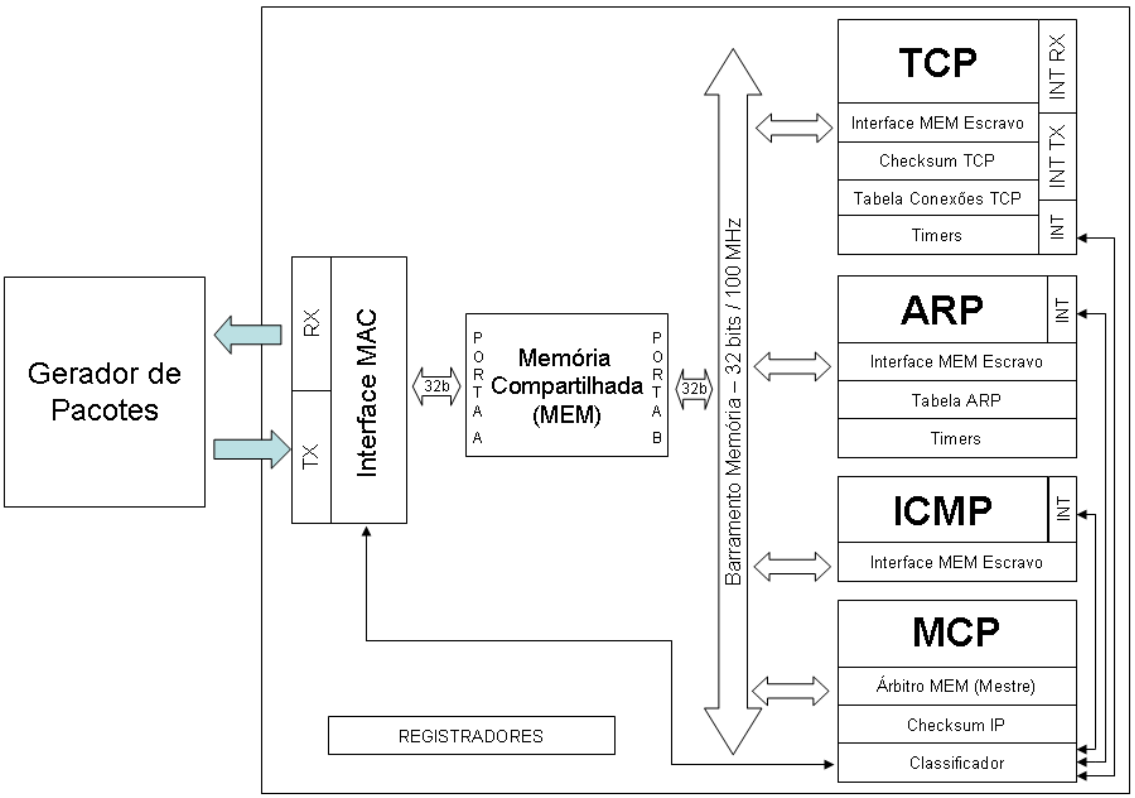

Figura 5.14: Diagrama de Blocos do Ambiente de Simulação

para processar um pacote na descrição de 16 bits ser maior, essa diferença não se traduz no tempo de computação do pacote. Isso ocorre porque a freqüência da descrição de 16 bits é maior que a freqüência da descrição de 32 bits.

Tabela 5.8: Resultados do Tempo de Computação dos Pacotes (us) para arquitetura de 16 bits

Conexões TCP	ARPReply	ICMPEcho	Syn	SynAck
10	0,2031	0,5252	1,3515	0,7073
5	0,2031	0,5252	1,2815	0,7073
1	0,2031	0,5252	1,2254	0,7073

Uma análise comparativa dos resultados obtidos nos dois experimentos detalhados nas tabelas 5.4 à 5.9 revela que a área ocupada pelos circuitos gerados a partir da descrição de 16 bits é em média 9% maior, porém, o tempo de computação é em média 18% menor do que a descrição de 32 bits (média calculada para as três configurações de número de conexões TCP).

A partir desses dados, a escolha sobre qual descrição usar vai depender das restrições e requisitos para a qual a presente arquitetura será aplicada. Ou seja, desejando-se aplicar a arquitetura com uma restrição em relação à área ocupada pelo circuito do iNetCore a

Tabela 5.9: Resultados do Tempo de Computação dos Pacotes (us) para arquitetura de 32 bits

Conexões TCP	ARPReply	ICMPEcho	Syn	SynAck
10	0,2706	0,6061	1,6234	0,7251
5	0,2706	0,6061	1,5152	0,7251
1	0,2706	0,6061	1,4286	0,7251

solução mais viável seria a descrição de 32 bits. Porém, se há um requisito de desempenho, a descrição de 16 bits se mostra a de maior poder de computação para o fluxo de pacotes analisados.

6 INTERFACE DE COMUNICAÇÃO HARDWARE/SOFTWARE DA ARQUITETURA DO INETCORE

A arquitetura do iNetCore, como apresentado no capítulo 5, possui uma interface para comunicação com a camada de aplicação. Neste capítulo, será relatado o desenvolvimento dessa interface e o ambiente de integração do iNetCore com uma arquitetura hardware/software contendo diversos elementos presentes em um sistema de computação tradicional, tal como processador, barramentos, bloco de memória, entre outros.

6.1 Arquitetura Hardware/Software com o iNetCore

Em um sistema de comunicação via Internet existente nos computadores atuais, é necessária a presença da pilha de comunicação TCP/IP, própria de cada sistema operacional, para gerenciar o processamento de todos os protocolos da pilha de maneira transparente para o usuário do sistema operacional.

A integração da pilha TCP/IP, agora implementada em hardware através do iNetCore, nesse sistema de comunicação apresenta dois desafios principais:

- apresentar a flexibilidade existente em uma pilha TCP/IP que tradicionalmente foi executada por meio de um software do sistema operacional;
- tornar o uso da pilha TCP/IP, agora em hardware, transparente ao usuário.

A flexibilidade é alcançada através da customização do iNetCore, apresentado no capítulo 5, em relação ao número de conexões TCP, tamanho da tabela ARP e tamanho da memória compartilhada.

Já a transparência é alcançada através da implementação de uma interface de comunicação entre o iNetCore e um serviço da camada de aplicação rodando em um sistema operacional qualquer. Na arquitetura do iNetCore, esse ambiente de comunicação é implementado através de uma interface existente entre esses dois níveis (INT RX e INT TX da figura 5.3). Essa interface irá gerenciar possíveis eventos, tais como:

- Inicialização e configuração inicial do iNetCore;
- Solicitação de envio de dados para encapsulamento em pacotes por parte de um determinado serviço da camada de aplicação;
- Entrega de conteúdo contido em um segmento TCP por parte do iNetCore para um determinado serviço da camada de aplicação;

Nesse ambiente de comunicação, alguns dados a respeito dos detalhes da conexão TCP que está sendo tratada devem ser passados para o serviço que está rodando na camada de aplicação. Sendo assim, o iNetCore disponibiliza através dessa interface dados como o número da porta que irá acionar o respectivo serviço, e os dados recebidos em um segmento TCP que devem ser passados para o serviço correspondente.

A camada em software é responsável por inicializar e configurar o iNetCore, além de sincronizar a chegada e saída de dados de segmentos TCP, e acionar o serviço correspondente ao pedido efetuado no iNetCore através do desencapsulamento de um segmento TCP.

Um exemplo de utilização do iNetCore por um serviço da camada de aplicação pode ser a implementação de um Web Server em software. A comunicação entre o iNetCore e o serviço que está rodando em software é efetuado sempre que há um pedido de requisição de página (get http). Quando isso acontece, o serviço é acionado e dados são repassados para o Web Server que irá processá-los. O resultado desse processamento poderá ser o envio de dados pelo serviço para o iNetCore, por exemplo, o conteúdo de uma página html. Nesse caso, o Web Server irá transferir o conteúdo da página requisitada (ou parte dela) e retornará o controle para o iNetCore que irá encapsular o cabeçalho do pacote e realizar o checksum para envio do pacote. Uma vez terminado o processamento, o iNetCore acionará o MAC para que o pacote seja enviado através da interface de rede presente na arquitetura.

A arquitetura HW/SW foi prototipada na placa *Virtex-II Pro Development System* da Xilinx (XILINX, 2005). A ferramenta EDK 8.1 da Xilinx foi utilizada para integração dos módulos em hardware e software, valendo-se de características presente na plataforma de desenvolvimento em questão, como *drivers* de dispositivos, inserção de módulos de controle de clock, microprocessadores, memória, entre outras funcionalidades.

A arquitetura HW/SW com todos os componentes da plataforma em uso é apresentada na figura 6.1.

Como pode ser visualizada na figura 6.1, o iNetCore é utilizado em conjunto com alguns componentes para formar um periférico, denominado aqui Interface OPB, que é conectado ao barramento OPB da arquitetura HW/SW. Para acesso à rede, é utilizado um core desenvolvido por Horna (HORNA et al, 2007), denominado MAC - Media Access Controle - 10/100M, o qual realiza o processamento da camada de enlace, transformando os bits recebidos do PHY 10/100M em pacotes ethernet entregues ao iNetCore. O MAC, por sua vez, recebe dados vindos do PHY 10/100M LXT972, da Intel (INTEL, 2000). O PHY 10/100M é um chip externo ao FPGA, presente na placa de desenvolvimento, fazendo a ligação da porta RJ45 (onde é conectado o cabo de rede) com o dispositivo que deseja acessá-lo, no caso, o MAC. O periférico Inteface OPB apresenta também duas fifos, TCP RX e TCP TX, as quais o iNetCore irá utilizar para transferir o fluxo de dados de um pacote. Os dados transferidos através das fifos são mapeados no bloco de memória (BRAM) presente no barramento OPB, onde o serviço em software irá buscar e armazenar os dados do pacote em processamento.

Além do periférico Interface OPB, a arquitetura HW/SW é formada por um microprocessador Microblaze (Microblaze Processor), um bloco de memória (BRAM BLOCK), um periférico para acesso à porta serial (UART LITE) e um módulo gerenciador de clock digital (DCM). Todos esses elementos fazem parte da arquitetura HW/SW e são sintetizados pela ferramenta EDK da Xilinx para um só sistema (comumente chamado de SOC - System On a Chip) dentro do FPGA presente na placa de desenvolvimento. A placa de desenvolvimento em questão é o *Virtex-II Pro Development System*, da Digilent (XILINX,

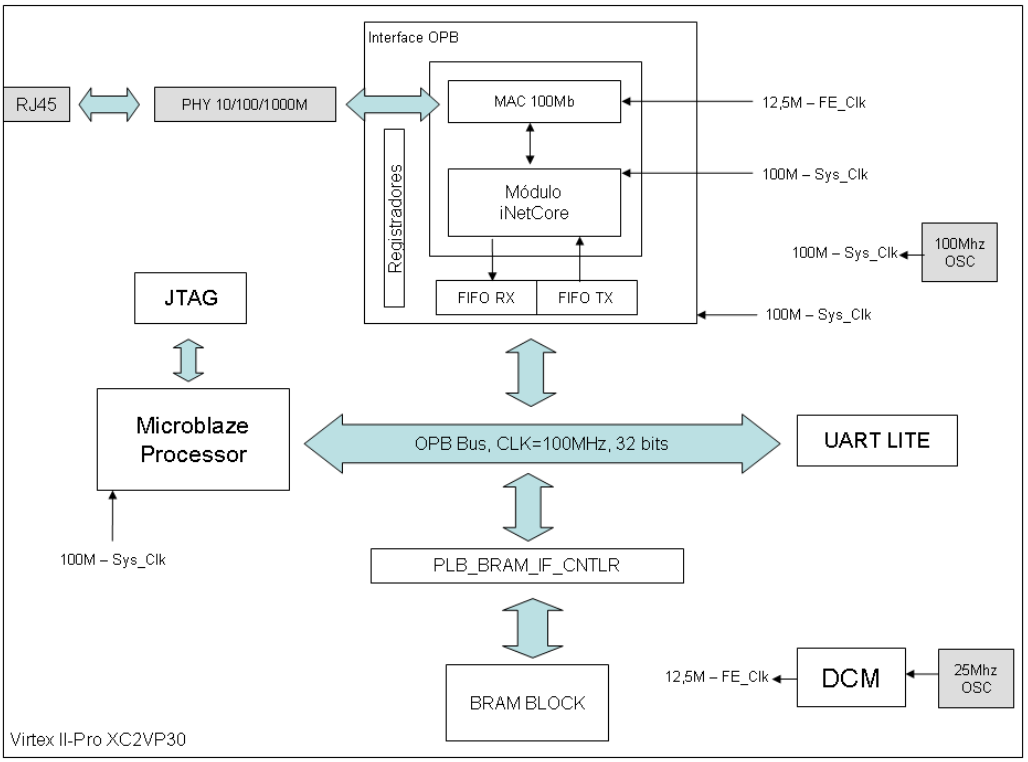

Figura 6.1: Arquitetura HW/SW para a plataforma Virtex-II Pro Development System

2005).

6.1.1 Interface de comunicação HW/SW

O iNetCore, em conjunto com outros componentes já citados, formam um periférico que é conectado ao barramento OPB, denominado Interface OPB.

O recebimento e o envio de dados por meio deste periférico é realizado de modo transparente para o serviço que queira utilizar o iNetCore, não se importando com questões de endereço de registradores que realizam a comunicação com o iNetCore em hardware, tampouco com as FIFOs que compõem o barramento e são usadas para transferir/receber os dados. Essa transparência é alcançada por meio de diretivas de software que o programador usa para realizar a comunicação com o iNetCore. Essas diretivas visam facilitar a comunicação do iNetCore com os serviços da camada de aplicação de forma transparente. As diretivas que podem ser usadas pelo programador são exibidas na tabela 6.1.

Tabela 6.1: Diretivas para acesso ao iNetCore via software

Diretivas	**Parâmetros**	**Descrição**
int transmit_data (unsigned char *p_buff)	p_buff = ponteiro para região de memória onde está armazenado o conjunto de dados a ser transmitido	Deve ser usada quando se deseja transmitir dados para o iNetCore a serem enviados para a rede, retornando zero se a transmissão foi bem sucedida.

continua na próxima página

Tabela 6.1: continuação da página anterior

Diretivas	**Parâmetros**	**Descrição**
int receive_data (unsigned char *p_buff)	p_buff = ponteiro para a região de memória onde o conjunto de dados recebidos deve ser armazenado	Deve ser usada quando se deseja receber dados do iNetCore após a indicação de chegada de pacote pela camada de aplicação, retornando zero se a recepção foi bem sucedida.
int packet_rcv	–	Deve ser usada para verificar a indicação de chegada de pacote, retornando "1"se um novo pacote foi recebido. Esta operação deve ser prosseguida pela diretiva receive_pack para recebimento do pacote.
int listen_connection (int port)	port = número da porta que se deseja colocar em modo de "escuta"	Deve ser usada para abrir uma conexão TCP no iNetCore para comunicação com aplicações remotas, retornando zero quando o pedido foi efetuado com sucesso; na ocasião de falha na solicitação, as possíveis causas são: a porta está em uso por outro processo ou não há mais conexões disponíveis na tabela de conexões do iNetCore.
int reset	–	Deve ser usada para reiniciar o iNetCore, acarretando na reinicialização das tabelas ARP e conexões TCP.
int set_ip (unsigned char ipnumber)	ipnumber = endereço IP local	Deve ser usada para setar o endereço IP local, retornando 0 se a operação foi bem sucedida.
get_ip (unsigned char *ipnumber)	ipnumber = ponteiro para a região de memória onde o endereço IP lido será armazenado	Deve ser usada para ver qual endereço IP local está setado no iNetCore.
int set_gw (unsigned char ipnumber)	ipnumber = endereço IP do gateway em formato hexadecimal	Deve ser usada para setar o endereço IP do gateway da rede, retornando 0 se a operação foi bem sucedida.
get_gw (unsigned char *ipnumber)	ipnumber = ponteiro para a região de memória onde o endereço IP lido será armazenado	Deve ser usada para ver qual endereço IP do gateway está setado no iNetCore.

continua na próxima página

Tabela 6.1: continuação da página anterior

Diretivas	Parâmetros	Descrição
int set_mask (unsigned char ipnumber)	ipnumber = endereço IP da máscara de rede em formato hexadecimal	Deve ser usada para setar a máscara de rede do endereço IP setado pela diretiva set_ip, retornando 0 se a operação foi bem sucedida.
get_mask (unsigned char *ipnumber)	ipnumber = ponteiro para a região de memória onde o endereço IP lido será armazenado	Deve ser usada para ver qual a máscara de rede está setada no iNetCore.

6.2 Síntese da Arquitetura HW/SW

No capítulo 5 é apresentado o iNetCore e as duas implementações desenvolvidas para a sua arquitetura: 16 e 32 bits. Foi realizada a síntese das duas implementações, além de uma análise de desempenho em relação ao processamento de pacotes do tipo "ARP Reply", "ICMP Echo", "Syn TCP"e "Syn Ack TCP". A análise sobre os resultados revelou que a descrição de 16 bits se mostrou mais eficiente em relação ao tempo de processamento dos pacotes, enquanto a descrição de 32 bits obteve um resultado melhor em relação à área ocupada do circuito nas tecnologias ASIC e FPGA.

Procurou-se definir qual das duas descrições implementadas do iNetCore seria utilizada na arquitetura HW/SW prototipada em FPGA, levando-se em consideração duas características determinísticas da arquitetura HW/SW:

- o barramento OPB onde o iNetCore é conectado apresenta uma largura de dados de 32 bits, assim como as fifos utilizadas para passagem dos dados para a camada de usuário, acessadas pelo serviço que utilizará o iNetCore para comunicação com a Internet;
- a freqüência do barramento é de 100MHz.

A primeira característica pôde ser compatibilizada com a utilização da implementação do iNetCore de 32 bits. Porém, a freqüência alcançada na síntese para o FGPA utilizado na placa de desenvolvimento foi de 58,7MHz, como demonstrado na seção 5.3. Foi preciso otimizar a implementação para alcançar a freqüência de operação barramento OPB. Entre as modificações realizadas na arquitetura do iNetCore, a determinante foi a otimização da operação de checksum dos pacotes. Passou-se a utilizar um somador de 16 bits para a realização do checksum, e conseqüentemente, a implementação de uma lógica para controle desse somador nos submódulos MCP e TCP que realizam o checksum dos protocolos IP e TCP, respectivamente. Os resultados alcançados na síntese do iNetCore juntamente com os demais componentes da arquitetura HW/SW são relatados na tabela 6.6.

A seguir é detalhado o ambiente de desenvolvimento para utilização da arquitetura HW/SW. Esse ambiente foi implementado utilizando a ferramenta Embedded Development Kit (EDK) 8.1.02i da Xilinx (EDK, 2005). A estrutura de diretório do projeto é demonstrada na figura fig:estr_dir. O diretório raiz é o xps_tcp.

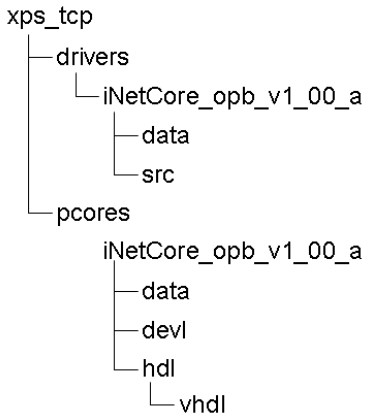

Figura 6.2: Estrutura de diretório do projeto no EDK

Os arquivos VHDL do módulo iNetCore e todos os outros arquivos instanciados ficam armazenados no diretório *pcores/iNetCore_opb_v1_00_a/hdl/vhdl*. Neste local também ficam armazenados os dois arquivos que realizam a interface com o barramento OPB da arquitetura HW/SW (iNetCore_opb.vhd e user_logic.vhd). A descrição de cada arquivo deste diretório é exibida na tabela 6.2.

Tabela 6.2: Arquivos que compõem o periférico Interface_OPB

Arquivo	**Descrição**
iNetCore_opb.vhd	Arquivo topo do projeto. Realiza a interface entre o barramento OPB e a lógica do usuário (user_logic).
user_logic.vhd	Realiza o mapeamento entre o iNetCore e os registradores, além de determinar a lógica do usuário definida por máquinas de estados para escrita e leitura nas fifos do barramento OPB.
iNetCore.vhd	Realiza a interligação entre os módulos que compõem a arquitetura do iNetCore.
mcontrol.vhd	Define a máquina de estados do submódulo MCP, o qual controla toda a sincronização de chegada e saída de pacotes vindos do MAC, assim como o acionamento dos submódulos TCP, ARP e ICMP, além do controle de checksum do cabeçalho IP.
icmp.vhd	Máquina de estados para processamento dos pacotes do tipo ICMP.
arp.vhd	Máquina de estados para processamento dos pacotes do tipo ARP.
tcp.vhd	Define a máquina de estados para processamento dos pacotes do tipo TCP e interface com o serviço TCP da camada de aplicação, além do controle de checksum do segmento TCP.
chksum.vhd	Módulo combinacional do somador de 16 bits usado no cálculo de checksum.

continua na próxima página

Tabela 6.2: continuação da página anterior

Arquivo	Descrição
interface_mac.vhd	Define a máquina de estados da interface com o MAC.
global_constants.vhd	Pacote de tipos e constantes utilizados no projeto.

No diretório *pcores/iNetCore_opb_v1_00_a/data* ficam armazenados dois arquivos que são modificados à medida em que novos módulos de lógica são adicionados à arquitetura HW/SW. Esses dois arquivos são descritos na tabela 6.3.

Tabela 6.3: Arquivos de configuração do iNetCore_OPB

Arquivo	Descrição
iNetCore_opb_v2_1_0.pao	Lista todos os arquivos que devem ser sintetizados pela ferramenta de síntese. Obs.: Caso novos arquivos sejam adicionados ao projeto, sua correspondente informação deve ser incluída nesse arquivo.
iNetCore_opb_v2_1_0.mpd	Entre outras informações, descreve o mapeamento das portas do arquivo topo do projeto (iNetCore_opb.vhd).

Os arquivos do serviço TCP que está rodando em software ficam armazenados no diretório *drivers/iNetCore_opb_v1_00_a/src*. Neste diretório ficam dois arquivos importantes: iNetCore_opb.c e iNetCore_opb.h. Estes arquivos contém as diretivas de software com a função de realizar a interface com o iNetCore de forma transparente ao usuário. A descrição desses arquivos é exibida na tabela 6.4.

Tabela 6.4: Arquivos de interface com o iNetCore_OPB

Arquivo	Descrição
iNetCore_opb.c	Implementa as funções de reset do PHY, iNetCore e MAC por meio de escrita/leitura nos registradores HW/SW. Também apresenta as implementações das diretivas de software, que serão detalhadas a seguir. Em suma, o arquivo tcpip_opb.c realiza a interface em software entre o serviço que está rodando na camada de aplicação e o iNetCore_OPB.
iNetCore_opb.h	Arquivo de cabeçalho das funções e constantes do iNetCore_opb.c. Obs.: Este arquivo deve ser incluído na implementação do serviço TCP que está rodando na camada de aplicação para acesso às diretivas da tabela 6.1, através da seguinte cláusula: "#include iNetCore_opb.h"

Como o MAC utiliza o PHY 10/100M para comunicação com a rede, é necessário o

mapeamento de portas do periférico com portas externas do FPGA que fazem a conexão física com o PHY 10/100M. Essas portas externas são exibidas na tabela 6.5.

Tabela 6.5: Pinos I/O externos do FPGA

Porta	**Direção**	**Descrição**
clk_100_pin	I	Sinal de clock de 100MHz utilizado como clock de referência pelo barramento OPB, pelo microprocessador Microblaze e pelo iNetCore.
mii_rx_clk_pin	I	Sinal de clock de 25MHz utilizado como clock de referência pelo MAC para recebimento de pacotes.
mii_tx_clk_pin	I	Sinal de clock de 25MHz utilizado como clock de referência pelo MAC para transmissão de pacotes.
sys_rst_pin	I	Sinal de reset do sistema, utilizado pelo microprocessador Microblaze.
fpga_0_RS232_RX_pin	I	Sinal de dados para recebimento de dados pela porta serial da placa de prototipação.
fpga_0_RS232_TX_pin	O	Sinal de dados para transmissão de dados pela porta serial da placa de prototipação.
mii_rx_dv_pin	I	Sinal que indica um dado válido no barramento mii_rxd_pin (utilizado pelo MAC para recebimento de dados pela interface MII da placa de prototipação).
mii_rxd_pin [0:3]	I	Sinal de dados para recebimento de dados pela interface MII da placa de prototipação.
mii_tx_en_pin	O	Sinal gerado pelo MAC que indica à interface MII que há dados válidos no barramento mii_txd_pin para serem transmitidos.
mii_txd_pin	O	Sinal de dados para transmissão de dados pela interface MII da placa de prototipação.
mii_txerr_pin	O	Sinal gerado pelo MAC que indica que os dados em transmissão estão com erro e devem ser descartados.
n_phy_reset_pin	O	Sinal de reset que deve é passado à interface MII na inicialização do sistema.

A arquitetura HW/SW que pode ser visualizada na figura 6.1 foi sintetizada para o FPGA Virtex II-Pro XC2VP30-5ff896 da Xilinx. Os resultados de síntese são exibidos na tabela 6.6.

Como pode ser visualizado na tabela 6.6, a síntese do periférico composto, entre outros elementos, pelo iNetCore (configurado para uma conexão TCP e duas entradas na tabela ARP), obteve uma freqüência de operação de 101,93MHz suficiente para sua conexão ao barramento OPB. Cabe salientar que o microprocessador Microblaze utilizado

Tabela 6.6: Dados de síntese da arquitetura HW/SW no EDK

Componente	LUTs	Slices	FFs	BRAMs	Frequência (MHz)
Microblaze	1091	817	553	–	130,38
DLMB	1	1	1	–	303,67
ILMB	1	1	1	–	303,67
MP_OPB	169	99	11	–	303,67
LMB_BRAM	–	–	–	32	–
DLMB_CNTLR	5	3	1	–	–
ILMB_CNTLR	5	3	1	–	–
RS232_UART_1	86	48	58	–	209,46
iNetCore_OPB	4069	2305	1722	6	101,93
TOTAL	5427	3277	1704	38	101,93

na arquitetura HW/SW é definido como um *softcore* que também é sintetizado, ocupando elementos de lógica em FPGA.

6.3 Experimentos

Para avaliar o desempenho em relação ao throughput alcançado pelo iNetCore no processamento de pacotes TCP, foi realizado um experimento que simula uma conexão sendo estabelecida em uma determinada porta e dados sendo repassados de uma aplicação remota para um aplicação rodando sobre a arquitetura HW/SW na placa de desenvolvimento, utilizando o iNetCore para realizar a comunicação TCP/IP com a aplicação remota.

Antes de relatar o resultado deste experimento, é importante introduzir o conceito de latência. O submódulo TCP, assim como os outros submódulos do iNetCore, é modelado através de uma máquina de estados, onde as transições entre os estados da máquina irão definir o fluxo dos dados através da arquitetura do iNetCore. A transição entre os estados é realizada de maneira seqüencial e, em virtude disso, ocupam um certo tempo, contados em número de ciclos de clock, equivalente à quantidade de transições efetuadas até o momento onde o pacote é decodificado. A decodificação do pacote é realizada através de uma série de operações: a conexão correspondente é identificada, o conteúdo é validado através do checksum e a área de dados do pacote é repassada para a camada de aplicação. Portanto, o tempo levado pelo iNetCore para realizar todas essas operações sobre o pacote até o momento onde a área de dados do pacote é repassada para a camada de aplicação é definido como latência. A figura 6.3 demonstra esse conceito, onde o tempo considerado no processamento de desencapsulamento (sentido RX) e encapsulamento (sentido TX) dos pacotes é definido como latência.

Um ponto crítico na decodificação de um segmento TCP é o cálculo do checksum sobre os dados trafegados pelo segmento TCP. O checksum deve cobrir o cabeçalho e a área de dados do segmento TCP. Para evitar uma latência muito grande, o checksum é calculado para todo pacote que é recebido da rede, no momento em que ele está sendo escrito na memória compartilhada. O checksum calculado é comparado com a informação constante no cabeçalho do segmento TCP recebido e, no momento em que a máquina de

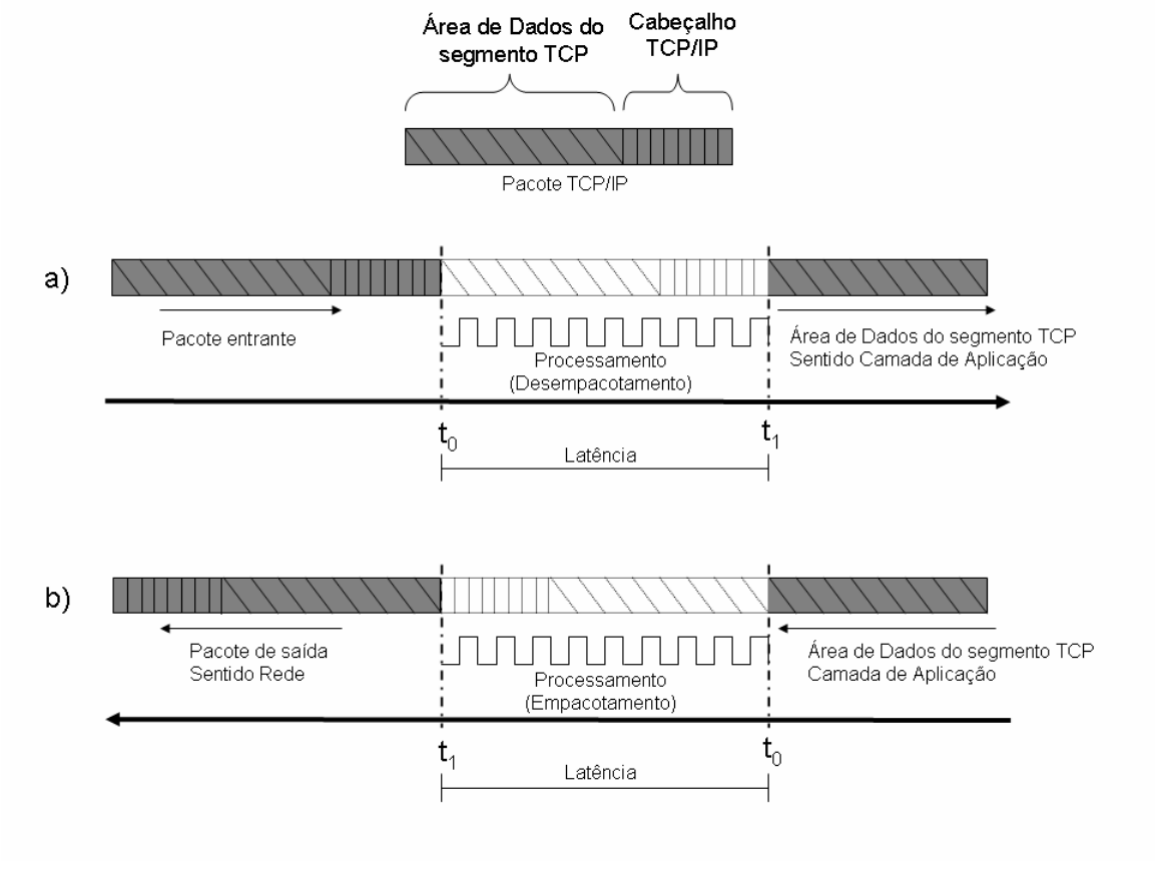

Figura 6.3: Esquema de processamento dos pacotes no sentido RX (a) e TX (b)

estados do módulo TCP necessita da informação do valor do checksum calculado, ele já está disponível.

No cálculo da latência, o fator determinante para a variação do valor da latência entre diferentes pacotes é o tamanho da área de dados do segmento TCP, uma vez que todo o conteúdo da área de dados do segmento TCP é repassado da memória compartilhada para as fifos do barramento OPB. O tempo despendido nessa tarefa é diretamente proporcional ao tamanho da área de dados do segmento TCP. O cálculo da latência para cada segmento TCP pode ser definido pela seguinte expressão:

$$L = C + T$$

onde:

- L = Latência (em segundos);
- C = tempo despendido pelo iNetCore para decodificar o cabeçalho do protocolo TCP/IP – este valor é constante para diferentes segmentos TCP;
- T = tempo despendido para repassar os dados do pacote no sentido iNetCore para a Camada de Aplicação, através das fifos do barramento OPB.

Outro termo muito utilizado na literatura é o "throughput", que mede a capacidade de um dispositivo de trafegar fluxo de dados em um determinado sentido. Neste experimento, o throughput está diretamente ligado aos valores de latência e tamanho de

um determinado pacote TCP/IP. Como a latência é definida pelo tempo de encapsulamento/desencapsulamento de um pacote pelo iNetCore, cada pacote terá um valor de latência específico, diretamente proporcional ao seu tamanho. Sendo que o valor de latência para encapsulamento/desencapsulamento do cabeçalho TCP é fixo, o que modifica o valor de latência de um pacote para outro é o tamanho do pacote. Quanto maior o pacote, maior a latência. No mesmo sentido, quanto maior o pacote, maior a quantidade de dados que o pacote trafega na área de dados. Essa relação "tamanho x latência"é que vai definir o throughput alcançado pelo iNetCore.

Os dados possuem fluxo diferentes dentro do iNetCore. No recebimento (sentido RX), o processamento de um pacote leva mais tempo do que na transmissão (sentido TX). Isso acontece porque são necessárias mais tarefas para a decodificação de um cabeçalho no recebimento de um pacote do que na montagem de um cabeçalho para transmissão.

A tabela 6.7 exibe os valores de latência e throughtput alcançados para diferentes tamanhos de pacote no sentido RX, onde o fluxo de dados é o seguinte: MAC –> iNetCore –> Serviço TCP. Na tabela, "N"é o tamanho, em bytes, do pacote TCP/IP, "D"é o tamanho, em bytes, da área de dados do segmento TCP, "L"é a latência em microsegundos (us) e "Th"é o throughput alcançado, em gigabit por segundo (Gbps).

Tabela 6.7: Latência e Throughput no tratamento de segmentos TCP no sentido RX

N(bytes)	D(bytes)	L(us)	**Th(Gbps)**
64	10	0,905	**0,088**
128	74	1,225	**0,483**
256	202	1,865	**0,866**
512	458	3,145	**1,165**
1024	970	5,705	**1,360**
1524	1470	8,205	**1,433**

A tabela 6.8 exibe os valores de latência e throughtput alcançados para diferentes tamanhos de pacote no sentido TX, onde o fluxo dos dados é o seguinte: Serviço TCP –> iNetCore –> MAC. Na tabela, "N"é o tamanho, em bytes, do pacote TCP/IP, "D"é o tamanho, em bytes, da área de dados do segmento TCP, "L"é a latência em microsegundos (us) e "Th"é o throughput alcançado, em gigabit por segundo (Gbps).

Tabela 6.8: Latência e Throughput no tratamento de segmentos TCP no sentido TX

N(bytes)	D(bytes)	L(us)	**Th(Gbps)**
64	10	0,805	**0,099**
128	74	1,125	**0,526**
256	202	1,765	**0,915**
512	458	3,045	**1,203**
1024	970	5,605	**1,384**
1524	1470	8,105	**1,451**

Como pode ser observado nos dados do experimento, quanto maior o tamanho do pacote TCP/IP, maior a latência de processamento pelo iNetCore. O cálculo do throughput

é efetuado com base na latência e no tamanho da área de dados do segmento TCP, podendo ser definido através da seguinte expressão:

$$Th = (8*D)/L$$

onde:

- Th = Throughput (em bits por segundo − bps);
- D = Tamanho em bytes da área de dados do segmento TCP, sendo definido pelo tamanho total do pacote TCP/IP (N) menos o tamanho do cabeçalho TCP/IP (54 bytes);
- L = Latência de processamento de um pacote(em segundos).

Cabe salientar que o throughput medido no experimento leva em conta somente a taxa de transferência dos dados que compõem a área de dados do segmento TCP, e não considera os dados do cabeçalho TCP/IP.

O throughput alcançado pelo iNetCore em ambos os sentidos − RX e TX − aumenta na medida em que o tamanho do pacote também aumenta. Isso acontece porque um pacote de tamanho maior transporta mais dados na área de dados do segmento TCP e, como a latência para a decodificação do cabeçalho TCP/IP é fixa para diferentes tamanhos de pacote, quanto mais dados puderem ser transportados em um único pacote, maior será o throughput alcançado.

No sentido RX, o throughput varia de 88,4Mbps (para pacotes menores) à 1,43Gpbs para pacotes de tamanho máximo de um pacote ethernet (1.524 bytes). Já no sentido TX, o throughput aumenta um pouco, variando de 99,4Mbps (para pacotes menores) à 1,45Gpbs para pacotes de tamanho máximo para um pacote ethernet (1.524 bytes).

Comparando-se os resultados alcançados nesse experimento com os resultados relatados na tabela 4.1, o iNetCore alcançou um throughput seis vezes maior que o alcançado por WU (WU et al, 2006). Já a área ocupada no FPGA pela arquitetura HW/SW, apresentada nesse capítulo, foi de 3717 Slices contra 5219 Slices ocupados pela arquitetura apresentada por WU (WU et al, 2006), 28% menor. Os melhores resultados alcançados pelo iNetCore em relação a WU (WU et al, 2006) se devem principalmente ao fato de que todo o gerenciamento de conexões TCP no iNetCore é efetuado em hardware, eliminando as limitações acarretadas pelo processamento em software relatadas nessa dissertação.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Esta dissertação apresentou a arquitetura do iNetCore para processamento da pilha de protocolos TCP/IP em hardware. A arquitetura proposta é composta por um módulo específico para processamento de pacotes das camadas de rede (protocolos IP, ARP e ICMP) e transporte (protocolo TCP) do modelo TCP/IP. Todo o processamento que deve ser realizado sobre esses pacotes é realizado pelo hardware dedicado, acelerando assim o throughput e livrando a carga de CPU que seria necessária para o processamento da pilha TCP/IP em software. A característica diferencial da arquitetura do iNetCore é a customização de algumas funcionalidades presentes nas tradicionais pilhas TCP/IP em software: a definição do número de conexões TCP a serem gerenciadas, o tamanho da tabela ARP e o tamanho da memória que irá armazenar o pacote em processamento. Essa customização adiciona uma maior flexibilidade à arquitetura ao mesmo tempo que permite um alto desempenho de processamento em hardware.

Foram desenvolvidas duas implementações da arquitetura do iNetCore em relação à largura do barramento de dados do meio de armazenamento utilizado, o que acarreta também a mudança das máquinas de estados que definem o comportamento dos submódulos. Essas duas implementações foram sintetizadas para tecnologias ASIC e FPGA, variando-se as configurações da arquitetura referente ao número de conexões TCP gerenciadas.

Além disso, foi desenvolvido um ambiente de integração do iNetCore com um serviço rodando em software. Esse ambiente foi elaborado sobre a plataforma de FPGA Virtex II-Pro da Xilinx, representando um sistema de computação tradicional, com microprocessador, memória, barramento e módulos específicos de processamento, como o iNetCore. Para uso do iNetCore, foi definido uma série de diretivas que devem ser usadas para acesso às funcionalidades do iNetCore por um software que queira utilizá-lo para comunicação com um dispositivo remoto via Internet. Foi efetuado um experimento sobre esse ambiente com o objetivo de verificar o desempenho do iNetCore no processamento de fluxo de segmentos TCP. O throughput alcançado foi de 99,4 Mbps - Mega bits por segundo - para pacotes de tamanhos menores (64 bytes) e chegou à 1,45 Gbps - Giga bits por segundo - para pacotes de tamanhos maiores (1524 bytes).

7.1 Principais Realizações

As principais realizações do trabalho desenvolvido são relatadas a seguir:

1. Estudo e mapeamento das principais funcionalidades da pilha TCP/IP.
2. Exploração do projeto em relação à melhorias no desempenho do processamento dos pacotes, principalmente naquelas funções que requerem uma maior carga de processamento, tal como o checksum.

3. Completa definição, implementação e verificação da interface tanto no lado do cliente MAC como no lado da camada de aplicação.

4. Definição, implementação e verificação das diretivas de acesso ao iNetCore pelo software da camada de aplicação, por meio de uma compacta biblioteca de funções.

5. Definição, implementação e verificação de uma arquitetura que possibilita ao usuário configurá-la conforme as necessidades da aplicação que deseja utilizar a pilha TCP/IP em hardware para acesso à Internet.

A seguir, são relatadas as dificuldades encontradas e algumas questões que ficaram em aberto no desenvolvimento do trabalho apresentado nessa dissertação e que podem ser exploradas em trabalhos futuros.

7.2 Trabalhos Futuros

A idéia do trabalho foi desenvolver uma arquitetura que pudesse ser completa o bastante para que uma aplicação pudesse comunicar-se com a Internet de forma confiável, através da utilização do protocolo TCP, e com possibilidade de utilizar o módulo desenvolvido nas novas tecnologias de acesso à rede, com velocidades acima de 100Mbps.

A versão desenvolvida da arquitetura utiliza componentes de memória disponíveis na lógica programável, como Block RAMs ou RAMs distribuídas, para armazenar o pacote. Já as informações da tabela de conexões TCP e da tabela ARP são armazenadas por meio de registradores internos da arquitetura. Em virtude disso, a utilização de um grande número de conexões TCP acarretaria na utilização de uma grande quantidade de lógica programável, na forma de memória distribuída ou *BRAMs*, mesmo para componentes com um grande número de elementos de memória disponíveis. Como alternativa, essas informações poderiam ser armazenadas em módulos externos de memória, muitas vezes disponíveis na plataforma de prototipação, como, por exemplo, memórias do tipo SDRAM, com a adição de uma interface à arquitetura do iNetCore que possibilitasse a comunicação com essas memórias.

Outra dificuldade encontrada na implementação do iNetCore foi a impossibilidade do armazenamento de pacotes para a reordenação no caso da chegada fora de ordem. Em virtude da escassez de recursos de memória, a arquitetura é capaz de armazenar o conteúdo de apenas um pacote (de tamanho parametrizável). Esse pacote não pode ser sobreposto enquanto não for efetuado seu processamento por parte do iNetCore. Outros pacotes que chegam durante esse processamento serão descartados. Para evitar isso, pode ser utilizada uma FIFO na chegada dos pacotes da rede.

Um outro esquema de armazenamento também poderia ser criado para armazenar mais de um pacote por vez na arquitetura do iNetCore, por meio de uma abordagem da arquitetura com uma FIFO paginada, onde cada página da FIFO poderia conter um pacote e o iNetCore acessaria os pacotes através da incrementação das páginas da FIFO.

Todas essas questões foram levantadas a partir dos resultados finais alcançados. Com certeza outras questões devem surgir com o tempo e devem ser exploradas em futuros aperfeiçoamentos da arquitetura do iNetCore apresentado nessa dissertação.

REFERÊNCIAS

BOKAI, Z.; CHENGYE, Y. TCP/IP Offload Engine (TOE) for an SOC System. In: NIOS II EMBEDDED PROCESSOR DESIGN CONTEST: outstanding designs 2005. [S. l.: s. n., 2005]. p. 306–322.

CLARK, D.; JACOBSON, V. et al. An Analysis of TCP Processing Overhead. **IEEE COMM.**, New York, v. 27, n. 6, p. 23–29, 1989.

CLARK, D. **Modularity and Efficiency in Protocol Implementation**: RFC 817. [S. l.: s. n.], 1982.

CLARK, T. **IP SANs**: A Guide to iSCSI, iFCP and FCIP Protocols for Storage Area Networks. Boston: Addison-Wesley, 2002. 288p.

COMER, D. E. **Interligação em Rede com TCP/IP**. 3. ed. Rio de Janeiro: Campus, 1999. 2v.

CURRID, A. TCP Offload to the Rescue, **Queue**, [S. l.], v. 2, n. 3, p. 58–65, May 2004.

DOLLAS, A. et al. An Open TCP/IP Core for Reconfigurable Logic. In: IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, FCCM, 13., 2005, Napa, CA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2005. p. 297-298.

DUNKELS, A. **The uIP Embedeed TCP/IP Stack**. Disponível em: <http://www.sics.se/adam/uip/>. Acesso em: nov. 2007.

DYKSTRA, P. **Gigabit Ethernet Jumbo Frames**. Disponível em: <www.wareonearth.com/whitepapers/GigabitEthernetJumboFrames.pdf>. Acesso em: nov. 2007.

FENG, W. et al. Performance Characterization of a 10Gigabit Ethernet TOE. In: HIGH PERFORMANCE INTERCONNECTS, FCCM, 13., 2005, Napa, CA. **Proceedings...** Los Alamos, NM: IEEE Computer Society, 2005. p. 58-63.

FINN, G.; HOTZ, S.; METER, R. V. The Impact of a Zero-Scan Internet Checksumming Mechanism. **ACM SIGCOMM Computer Communication Review**, New York, v. 26, n. 5, p. 27-39, Oct. 1996.

FOONG, A. et al. TCP Performance Re-Visited. In: IEEE INTERNACIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE, ISPASS, 2003, Austin, Texas. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2003. p. 70-79.

GUERRA, F. T. **Protocolos de Transporte Para Redes de Alta Velocidade: um estudo comparativo**. 2006. 80p. Dissertação (Mestrado em Engenharia de Telecomunicações) - Centro Tecnológico, Universidade Federal Fluminense, Niterói.

HAMERSKI, J.; RECKZIEGEL, E.; KASTENSMIDT, F. Evaluating Memory Sharing Data Size and TCP Connections in the Performance of a Reconfigurable Hardware-based Architecture for TCP/IP Stack. In: VERY LARGE SCALE INTEGRATION, ISVLSI, 2007, Atlanta, GA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2007. p. 212-217.

HORNA, C. D. T.; RAMOS, F. L.; BARCELOS, M. B.; REIS, R. A. L. Implementação e Validação de IP Soft Cores para Interfaces Ethernet 10/100 e 1000 Mbps Sobre Dispositivos Reconfiguráveis. In: WORKSHOP IBERCHIP, 13., 2007, Lima, Perú. **XII Workshop IBERCHIP**. Lima: HOZLO S. R. L., 2007. p. 276–281.

INTEL COMPANY. **LXT972 - Dual-Speed Single Port Fast Ethernet Transceiver**: datasheet - Revision 1.1. [S. l.], 2000.

JACOBSON, V. 4BSD Header Prediction. **ACM SIGCOMM Computer Communication Review**, New York, v. 20, n. 1, p. 13-15, 1990.

KANT, K. TCP Offload Performance for Front-End Servers. In: GLOBAL TELECOMMUNICATIONS CONFERENCE, GLOBECOM, 2003, San Francisco, USA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2003. v. 6, p. 3242-3247.

KLEINPASTE, K.; STEENKISTE, P.; ZILL, B. Software Support for Outboard Buffering and Checksumming. **ACM SIGCOMM Computer Communication Review**, New York, v. 25, n. 4, p. 87–98, 1995.

MARKATOS, E. Speeding up TCP/IP: Faster Processors are not Enough. In: PERFORMANCE, COMPUTING, AND COMMUNICATIONS CONFERENCE, 21., 2002, Phoenix, USA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2002. p. 341-345.

PAPAEFSTATHIOU, I.; KORNAROS, G.; ZERVOS, N. Software Processing Performance in Network Processors. In: CONFERENCE ON DESIGN, AUTOMATIONS AND TEST IN EUROPE, 2004. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2002. v. 3, p. 186-191.

PARTRIDGE, C. **Gigabit Networking**. 6th ed. Reading, MA: Addison Wesley Longman, 1998.

PLUMMER, D. C. **Address Resolution Protocol**: RFC 826. [S. l.]: Network Working Group, 1982.

POSTEL, J. B. **Transmission Control Protocol**: RFC 793. [S. l.]: USC/Information Sciences Institute, 1981.

ROMANOW, A.; BAILEY, S. **An Overview of RDMA over IP**. [S. l.]: The Internet Society, 2002.

SENAPATHI, S.; HERNANDEZ, R. **Introduction to TCP Offload Engines**. [S. l.]: Dell Computers - Power Solutions, 2004. p. 103-107.

STEENKISTE, P. Design, Implementation and Evaluation of a Single-Copy Protocol Stack. **Software - Practice and Experience Journal**, New York, v. 28, n. 7, p. 749-772, 1998.

STEVENS, W. R. **TCP/IP Illustrated**. Reading, MA: Addison-Wessley, 1998.

XILINX. **Xilinx University Program Virtex-II Pro Development System**: hardware reference manual - UG069. v. 1.0. [S. l], 2005.

XILINX. **Embedded Development Kit EDK 8.1i**: embedded system tools reference manual - UG111. v. 5.0. [S. l.], 2005.

WANG, W; WANG, J.; LI, J. Study on Enhanced Strategies for TCP/IP Offload Engines. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED SYSTEMS, ICPADS, 11., 2005, Washington, USA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2005. p. 398–404.

WU, Z.; CHEN, H. Design and Implementation of TCP/IP Offload Engine System over Gigabit Ethernet. In: INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS AND NETWORKS, 1., 2006, Arlington, USA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society, 2006. p. 245-250.

YUSUF, S. et al. A Combined Hardware-Software Architecture for Network Flow Analysis. In: INTERNATIONAL CONFERENCE ON ENGINEERING OF RECONFIGURABLE SYSTEMS AND ALGORITHMS, 2005, Los Angeles, USA. **Proceedings...** [S. l.: s. n.], 2005. p. 149–155.