UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MINIMIZAÇÃO DO CABEÇALHO DO PROTOCOLO DE COMUNICAÇÃO IPV6 VISANDO A MELHORIA DE DESEMPENHO EM REDES LOCAIS

DISSERTAÇÃO DE MESTRADO

Robert Torrel

Santa Maria, RS, Brasil
2013

MINIMIZAÇÃO DO CABEÇALHO DO PROTOCOLO DE COMUNICAÇÃO IPV6 VISANDO A MELHORIA DE DESEMPENHO EM REDES LOCAIS

por

Robert Torrel

Dissertação apresentada ao Programa de Pós-Graduação em Informática, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Computação.

Orientador: Prof. Dr. João Baptista dos Santos Martins

Santa Maria, RS, Brasil

2013

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

MINIMIZAÇÃO DO CABEÇALHO DO PROTOCOLO DE COMUNICAÇÃO IPV6 VISANDO A MELHORIA DE DESEMPENHO EM REDES LOCAIS

elaborada por
Robert Torrel

como requisito parcial para obtenção do grau de
Mestre em Computação

COMISSÃO EXAMINADORA:

João Baptista dos Santos Martins, Dr.
(Presidente/Orientador)

Rolf Fredi Molz, Dr. (UNISC)

Roseclea Duarte Medina, Dra. (UFSM)

Santa Maria, 12 de Novembro de 2013

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria, RS, Brasil

MINIMIZAÇÃO DO CABEÇALHO DO PROTOCOLO DE COMUNICAÇÃO IPV6 VISANDO A MELHORIA DE DESEMPENHO EM REDES LOCAIS

Autor: Robert Torrel
Orientador: João Baptista dos Santos Martins

Local da Defesa e Data: Santa Maria, 12 de Novembro de 2013.

Um dos principais avanços que vem ocorrendo na utilização de tecnologias de comunicação, é a substituição gradativa do protocolo IPv4 pelo IPv6. Essa mudança apresenta diversas melhorias, embora o desempenho na transmissão seja relativamente menor ao considerar o novo padrão de endereçamento, que passa de 32 bits no IPv4, para 128 bits no IPv6, aumentando o tamanho do cabeçalho resultante e o *overhead* de comunicação. Este trabalho tem como principal objetivo o desenvolvimento de um método para a minimização do cabeçalho IPv6, buscando o aumento de desemprenho na transmissão de dados em uma rede local. Testes práticos demonstraram que a solução proposta permite uma melhora de desempenho na rede, com o aumento da vazão de dados, além da diminuição da latência e da utilização de banda na transmissão de pacotes entre dois dispositivos.

Palavras-chave: IPv6; LAN; Minimização de cabeçalho; Overhead de comunicação; Desempenho de rede.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Federal University of Santa Maria, RS, Brazil

HEADER MINIMIZATION OF IPV6 COMMUNICATION PROTOCOL AIMING AT IMPROVING LOCAL AREA NETWORK PERFORMANCE

Author: Robert Torrel
Advisor: João Baptista dos Santos Martins

Place and Date: Santa Maria, November 12th, 2013.

One of the major advances that has been happening in the use of communication technologies, is the gradual replacement of IPv4 by the IPv6. This change provides several enhancements, although the transmission performance is relatively low, by considering the new addressing standard, which becomes 128 bits in IPv6, compared to 32 bits in IPv4, increasing the size of the resultant header and the communication overhead. This work has as main objective the development of a method for the minimization of the IPv6 header, seeking to increase the performance of data transmission in a local area network. Practical tests have shown that the proposed solution enables an improvement in network performance, increasing data throughput, in addition to decreasing latency and bandwidth utilization in the transmission of packets between two devices.

Keywords: IPv6; LAN; Header minimization; Communication overhead; Network performance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
BRAM	Block Random Access Memory
CMOS	Complementary Metal-Oxide Semiconductor
CRC	Cyclical Redundancy Check
DCM	Digital Clock Manager
DNS	Domain Name System
DUT	Device Under Test
EDK	Embedded Development Kit
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSL	Fast Simplex Link
Gbps	Gigabit por segundo
GMII	Gigabit Media Independent Interface
HDL	Hardware Description Language
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
IFG	InterFrame Gap
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LAN	Local area network
LLC	Logical Link Control
LUT	Look-up table
MAC	Medium Access Control

Mbps	Megabit por segundo
MII	Media Independent Interface
NAT	Network Address Translation
OSI	Open Systems Interconnection
PHY	Physical Layer
RARP	Reverse Address Resolution Protocol
RFC	Request For Comments
RGMII	Reduced Gigabit Media Independent Interface
SDK	Software Development Kit
SFD	Start Frame Sequence
TCP	Transmission Control Protocol
IETF	The Internet Engineering Task Force
TTL	Time to live
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

Conforme a demanda por recursos de rede vem aumentando a cada ano, consequentemente aumenta-se também o número de dispositivos com disponibilidade de conexão à internet. Para que estes dispositivos possam se comunicar através da rede, atribui-se um endereço IP (*Internet Protocol*) único a cada um, o qual irá identificá-lo na rede. Atualmente, a versão mais utilizada do protocolo de rede é a IPv4, a qual vem gradativamente cedendo espaço à nova versão IPv6 (DEERING; HINDEN, 1998), sendo que as duas versões funcionam paralelamente até que ocorra a total substituição. A nova versão apresenta, entre outras melhorias, um novo padrão de endereçamento, que permite um número muito maior de dispositivos conectados simultaneamente quando comparado ao IPv4, o qual já apresenta um esgotamento dos endereços disponíveis devido ao crescente número de dispositivos.

Ao oferecer um novo padrão de endereçamento, o protocolo IPv6 acaba aumentando o tamanho dos pacotes transmitidos na rede, por conta do tamanho dos endereços. No IPv4, a área destinada ao endereço, corresponde a apenas 32 bits, enquanto que no IPv6, este valor passa a ser 128 bits. A alteração no padrão de endereçamento faz com que o tamanho do cabeçalho resultante para uma transmissão padrão entre dois dispositivos seja relativamente maior na nova versão do protocolo IP. Essa diferença, faz com que a transmissão de pacotes IPv6 seja afetada, reduzindo assim o desempenho com o aumento de utilização de banda e redução na vazão de dados.

1.1 Objetivos

Este trabalho tem como principal objetivo o desenvolvimento de um método para a minimização do cabeçalho IPv6, buscando o aumento de desempenho na transmissão de dados em uma rede local. Para isso, propõe-se o desenvolvimento de uma pilha de comunicação IPv6, que realize de maneira transparente ao usuário/aplicação, a redução do cabeçalho IP durante a transmissão de dados. O desenvolvimento baseia-se em imple-

mentações em software e hardware reconfigurável. A implementação da pilha codificada na linguagem de descrição de hardware Verilog, é realizada em um dispositivo *Field Programmable Gate Array* (FPGA), visando obter um aumento significativo no desempenho quando comparado a implementações tradicionais da pilha IPv6, tanto em software como em hardware.

No contexto do modelo OSI de referência, a arquitetura proposta implementa as camadas de enlace (MAC), e rede (IP). Esta arquitetura baseia-se em uma implementação de pilha UDP/IPv4 (HERRMANN, 2010), a qual apresentou bons resultados de desempenho. Propõe-se o desenvolvimento de um módulo de adaptação entre as camadas de enlace e de rede, que permita realizar a redução do cabeçalho IPv6 ao envio de um pacote, assim como o processo contrário que deve reconstruir o pacote minimizado para o formato original IPv6 ao recebê-lo.

Para análise de desempenho, a implementação deve oferecer suporte tanto ao protocolo padrão IPv6, além da pilha IPv6 proposta neste trabalho. Assim é possível realizar uma análise comparativa entre as duas pilhas, em cada uma das plataformas utilizadas.

Além disso, este trabalho também tem por objetivo a implementação de um dispositivo que permita a realização de testes, a partir de normas estabelecidas para a certificação de equipamentos de redes. Essa implementação visa a obtenção dos resultados de desempenho das pilhas IPv6, apresentando os valores atingidos em testes de latência e vazão na transmissão de dados.

1.2 Justificativa

O protocolo IP foi concebido para atender às demandas de redes WAN (Wide Area Network), com topologias mais complexas para permitir a comunicação entre dispositivos que não fazem parte da mesma rede local. Porém este protocolo não é otimizado para redes LAN (Local Area Network), onde os dispositivos partilham do mesmo meio físico para a troca de dados. Como esta troca de dados envolve várias etapas de encapsulamento e desencapsulamento das diferentes camadas de protocolos, o desempenho de comunicação é reduzido. Diversas informações são inseridas nos cabeçalhos de protocolos, sendo que muitas dessas na verdade são irrelevantes ou redundantes para a comunicação entre dispositivos de uma mesma rede local (MURUGESAN; RAMADASS; BUDIARTO, 2009).

A Figura 1 apresenta o encapsulamento de dados entre as diversas camadas de rede como exemplo para uma aplicação UDP/IP. Entre as camadas de transporte, rede e enlace,

são adicionados cabeçalhos de controle. Esses cabeçalhos são responsáveis por inserir mais dados na rede, e por consequência, um tráfego maior é necessário para a transmissão de dados entre dois dispositivos.

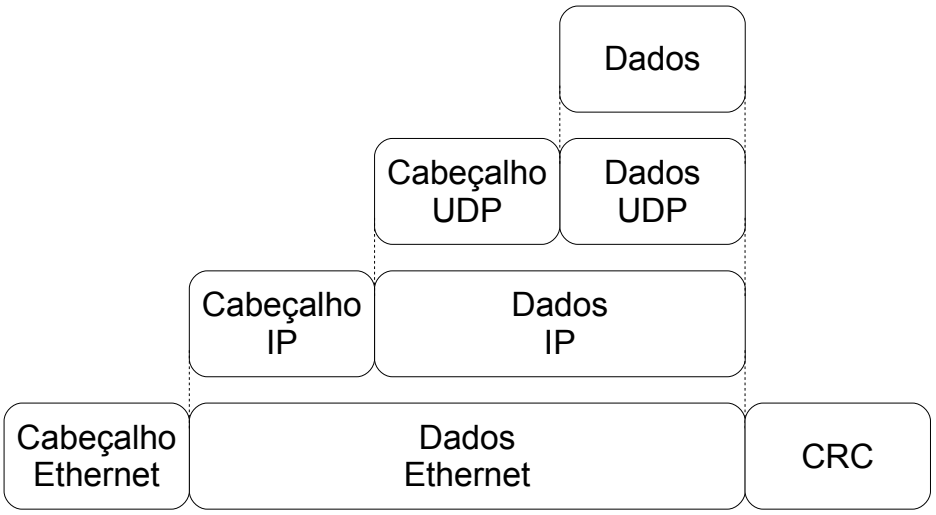

Figura 1: Encapsulamento de dados entre camadas de rede. Figura adaptada de (POSTEL, 1980)

No protocolo IPv6, o baixo desempenho quando comparado ao IPv4 fica mais evidente enquanto pacotes pequenos são transmitidos, pois um cabeçalho de 40 bytes se torna muito mais custoso para a transmissão de 1 byte de dados, do que para a transmissão de 1000 bytes de dados no pacote. A proporção do tamanho do cabeçalho com o tamanho total do pacote é reduzido à medida em que mais dados são transmitidos no mesmo pacote. Porém, a maior parte dos pacotes IP que trafegam na rede são de tamanhos pequenos, com até 100 bytes, representando 44% do total de pacotes transmitidos. Pacotes maiores, entre 1400 e 1500 bytes representam um total de 37% (JOHN; TAFVELIN, 2007).

A utilização de pacotes pequenos ocorre bastante em aplicações interativas e de tempo real, onde muitas vezes o tamanho da área de dados é menor que o próprio tamanho do cabeçalho IP. Pode-se citar como exemplo o aplicativo Skype, apresentando uma média de 83 bytes no tamanho dos pacotes em uma comunicação local (ZHANG et al., 2010). Como a maior parte do tráfego é de pacotes pequenos, o *overhead* gerado é muito grande, reduzindo assim o desempenho da rede.

No caso do tráfego para aplicações VoIP, o *overhead* causado pela inserção de cabeçalhos de protocolo é um fator limitante para definir a capacidade de uma rede local, conforme demonstrado em (KEEGAN; DAVIS, 2006). Isto implica na limitação do número de clientes VoIP simultâneos, conforme ocorre a saturação da rede pelo tráfego de dados.

Visando a redução do *overhead* no protocolo IPv6, gerado pelo excessivo tamanho do cabeçalho, foram propostas metodologias para reduzir o volume de dados do cabeçalho durante a transmissão de pacotes na rede. Em (LIM; STERN, 2000), (WESTPHAL; KOODLI, 2005) e (XUE-ZHOU; ZHI-LING; SI-MIN, 2009), os autores propõe técnicas de compressão do cabeçalho IPv6, reduzindo significativamente o seu tamanho. Já no trabalho proposto por (MURUGESAN; RAMADASS; BUDIARTO, 2009), utiliza-se uma técnica de customização do cabeçalho, onde são apresentados resultados teóricos a respeito da redução do *overhead* inserido na rede pelo cabeçalho IPv6.

Baseado em (MURUGESAN; RAMADASS; BUDIARTO, 2009), o trabalho aqui proposto explora alguns recursos avançados do IPv6, como a auto-configuração de endereçamento e redundâncias entre camadas, definindo o desenvolvimento de uma nova arquitetura que permita otimizar o tráfego de dados através do protocolo IPv6 em uma rede local. Isto é feito reduzindo-se ao máximo possível o tamanho do cabeçalho IP, criando assim um cabeçalho reduzido que transmita apenas informações relevantes durante a comunicação.

1.3 Organização

O primeiro capítulo desta dissertação realiza uma Introdução ao tema, estabelecendo os Objetivos e a Justificativa para o desenvolvimento do trabalho.

O Capítulo 2 abrange a Revisão Bibliográfica sobre protocolos de comunicação de dados, e metodologias para realização de testes em equipamentos de rede. Além disso, são apresentados os principais trabalhos relacionados à proposta da dissertação.

No Capítulo 3, define-se a arquitetura dos dispositivos implementados, detalhando-se o desenvolvimento e a implementação de cada um dos módulos propostos, tanto em software como em hardware. Nesse capítulo também é apresentada a metodologia utilizada tanto no desenvolvimento dos dispositivos como nos processos de validação e testes.

Os resultados práticos obtidos com a realização dos testes, são apresentados no Capítulo 4. É realizada uma análise a respeito das diversas arquiteturas implementadas, com as comparações de desempenho obtidas nos testes de latência e vazão. Além disso, apresenta-se também uma comparação com resultados de trabalhos relacionados.

Por fim, no Capítulo 5, são apresentadas as conclusões desta dissertação, assim como novas propostas para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados os principais fundamentos e conceitos utilizados no desenvolvimento do trabalho. Descreve-se inicialmente as quatro primeiras camadas do modelo OSI, utilizadas na comunicação de dados. Estas camadas são relevantes para um bom entendimento e estão diretamente relacionadas ao contexto deste trabalho. Após isto, apresenta-se a metodologia de testes necessários para a validação dos dispositivos de interconexão.

2.1 Protocolos de comunicação

O padrão Ethernet surgiu para permitir interconectar dispositivos através de redes de dados. Ele foi criado pela Xerox e padronizado pela IEEE (*Institute of Electrical and Electronics Engineers*) com o nome de IEEE 802.3. Este padrão engloba diversas especificações utilizadas para definir características do meio físico e modos de comunicação (IEEE, 2008).

No modelo OSI (*Open Systems Interconnection*), o padrão Ethernet IEEE 802.3 implementa as camadas 1 (nível físico) e 2 (nível de enlace). Este modelo, apresentado na Figura 2, foi desenvolvido pela ISO (*International Organization for Standardization*) e representa as funções das diversas camadas de um sistema de comunicação de dados (DAY; ZIMMERMANN, 1983).

A camada física é responsável pela troca de bits entre os dispositivos da rede, através de sinais elétricos. A camada de enlace é responsável por estabelecer um canal de transmissão entre os dispositivos da rede, permitindo a transferência de dados sobre o nível físico de maneira confiável. Acima dessas duas camadas, que fazem parte do padrão IEEE 802.3, pode-se citar ainda as camadas de rede e de transporte, pois o trabalho é desenvolvido com base nestas 4 camadas. A camada de Rede é responsável por definir os caminhos a serem seguidos pelos datagramas entre origem e destino, controle de congestionamento e interconexão de redes heterogêneas (TANENBAUM, 2002). Por fim, na camada

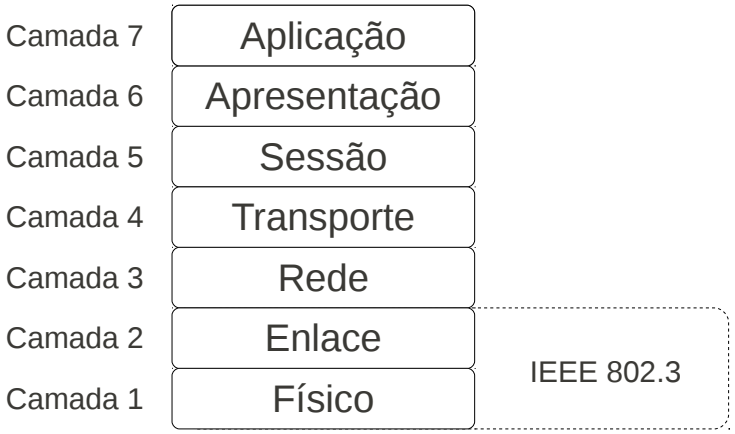

Figura 2: Modelo OSI. Figura adaptada de (DAY; ZIMMERMANN, 1983)

de Transporte os dados são transferidos de maneira eficiente e confiável entre a entre as máquinas de origem e destino, garantindo a chegada dos dados na sequência correta e livre de erros (COMER, 2005).

Entre cada uma dessas camadas, ocorre o encapsulamento e desencapsulamento de informação, para a troca de mensagens entre elas. Isso geralmente ocorre adicionando-se informações de controle relevantes dos protocolos, nos campos de cabeçalho.

A comunicação de dados à nível de enlace, se dá através de quadros Ethernet. Neste nível de comunicação, os dados são transmitidos, entre dois dispositivos conectados fisicamente entre si por um cabo de rede, configurando assim uma comunicação ponto a ponto. Na camada de enlace é definido o formato do quadro Ethernet, com os campos padronizados na norma IEEE 802.3, sendo estes descritos a seguir.

O quadro Ethernet é composto por um cabeçalho, uma área de dados e um campo de checagem de dados. No cabeçalho estão as informações necessárias para estabelecer um protocolo de transmissão, como por exemplo os endereços MAC (*Media Access Control*) origem e destino, que são descritos na seção seguinte. A área de dados, contém as informações, ou o encapsulamento de outros protocolos que irão trafegar na rede, como no caso de um datagrama IP. O último campo do quadro é utilizado para armazenar uma sequência de validação do quadro, para detectar possíveis erros que possam ocorrer durante a transmissão deste quadro.

A estrutura do quadro Ethernet está representada na Figura 3, onde são definidos os seus campos. A seguir, será feita uma descrição de cada um destes campos (IEEE, 2008).

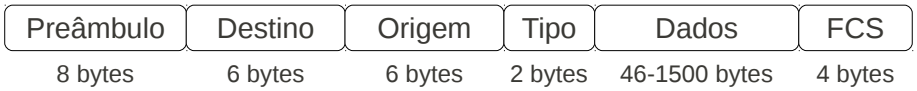

Figura 3: Estrutura de um quadro Ethernet. Figura adaptada de (IEEE, 2008)

- Preâmbulo: este campo é utilizado para estabelecer a sincronia entre o dispositivo transmissor e receptor do quadro. É formado por dois campos: O primeiro consiste de 7 bytes formados pela sequência binária de 1s e 0s alternados. O segundo campo, chamado de delimitador do início do quadro (*Start of Frame Delimiter*, ou SFD), é formado por 1 byte contendo a sequência binária "10101011". Este último byte é utilizado para indicar o início do quadro.

- Destino: campo de 6 bytes que armazena o endereço MAC referente ao dispositivo de destino do quadro. Através deste campo, o dispositivo receptor identifica se deve aceitar o quadro, ou descartá-lo.

- Origem: armazena o endereço MAC referente ao dispositivo transmissor do quadro. Da mesma forma que o campo Destino, este é composto por 6 bytes.

- Tipo: este campo pode ter duas interpretações diferentes, conforme conteúdo. Se o valor for menor que 1500 bytes, o conteúdo representa o tamanho do campo de dados do quadro Ethernet. Para valores maiores, o conteúdo deste campo identifica o protocolo encapsulado no campo de dados.

- Dados: contém as informações a serem transportadas na rede. O tamanho deste campo pode variar entre 46 e 1500 bytes. Se os dados não forem suficientes para completar o mínimo de 46 bytes, são inseridos *Pads* (bytes de preenchimento contendo valores aleatórios), uma vez que o tamanho mínimo do quadro especificado no padrão é de 64 bytes.

- FCS (*Frame Check Sequence*): campo de 4 bytes utilizado para armazenar uma sequência de verificação do quadro. Este campo é calculado através do algoritmo de CRC (*Cyclic Redundancy Check*) de 32 bits. É utilizado pelo dispositivo receptor para identificar erros no quadro, que podem ocorrer durante a transmissão no meio físico.

O CRC de 32 bits é calculado sobre todos os campos representativos do quadro Ethernet, desde o campo de endereçamento Destino até o campo Dados. Assumindo-se que a

sequência binária destes campos seja representada por D, e que o campo FCS, contendo o CRC de tamanho m seja representado por C, obtém-se a função exibida na Equação 2.1, que representa o quadro a ser transmitido com o CRC calculado.

$$T = \{DC\} = D \times 2^m + C \tag{2.1}$$

Para o cálculo do CRC, é realizada a divisão da mensagem $D \times 2^m$, por um polinômio gerador P utilizando-se aritmética de módulo 2. Isto é feito através de *shifts* e ou exclusivos (XORs). O polinômio gerador P, com grau de 32 bits ($m = 32$), é definido na Equação 2.2.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1 \tag{2.2}$$

Esse polinômio, utilizado para o cálculo do campo FCS no quadro Ethernet, pode também ser representado na forma binária, como demonstrado na Equação 2.3, onde p_{32} corresponde ao coeficiente x^{32} do polinômio *G(x)*.

$$P = \{p_{32}, p_{31}, ..., p_1, p_0\} = \{10000010011000001000111011011 0111\} \tag{2.3}$$

Para exemplificar, apresenta-se na Figura 4, a sequência de operações necessárias para o cálculo do CRC com um polinômio gerador de 4 bits ($m = 4$). Neste exemplo, a mensagem D a ser transmitida é representada por $\{11011010\}$. O polinômio gerador $G(x)x^4 + x^1 + x^0$, ou então, na forma binária $P = \{10011\}$.

Inicialmente, são adicionados 4 zeros à mensagem, e então realiza-se a divisão dessa mensagem por $P = \{10011\}$. A primeira etapa dessa divisão ocorre através da lógica XOR entre $\{11011\}$ e o divisor $\{10011\}$. Então o próximo bit da mensagem é concatenado ao resto $\{1000\}$, e outra vez realiza-se a operação XOR com o divisor. Quando o bit mais significativo do resto for igual a zero, é realizado uma operação *shift*. Seguindo-se estas operações até que todos os bits sejam calculados, o valor final obtido no resto da divisão é o código CRC, o qual será adicionado ao final da mensagem que será (WEIDONG, 2003).

Além dos campos presentes no quadro Ethernet, citados na Figura 3, existe ainda uma medida de tempo chamada *Inter Frame Gap* (IFG). Esta medida representa o tempo entre a transmissão de dois quadros consecutivos (IEEE, 2008). Para cada taxa de transmissão é definido um tempo mínimo, respeitando-se o valor equivalente à transmissão mínima de

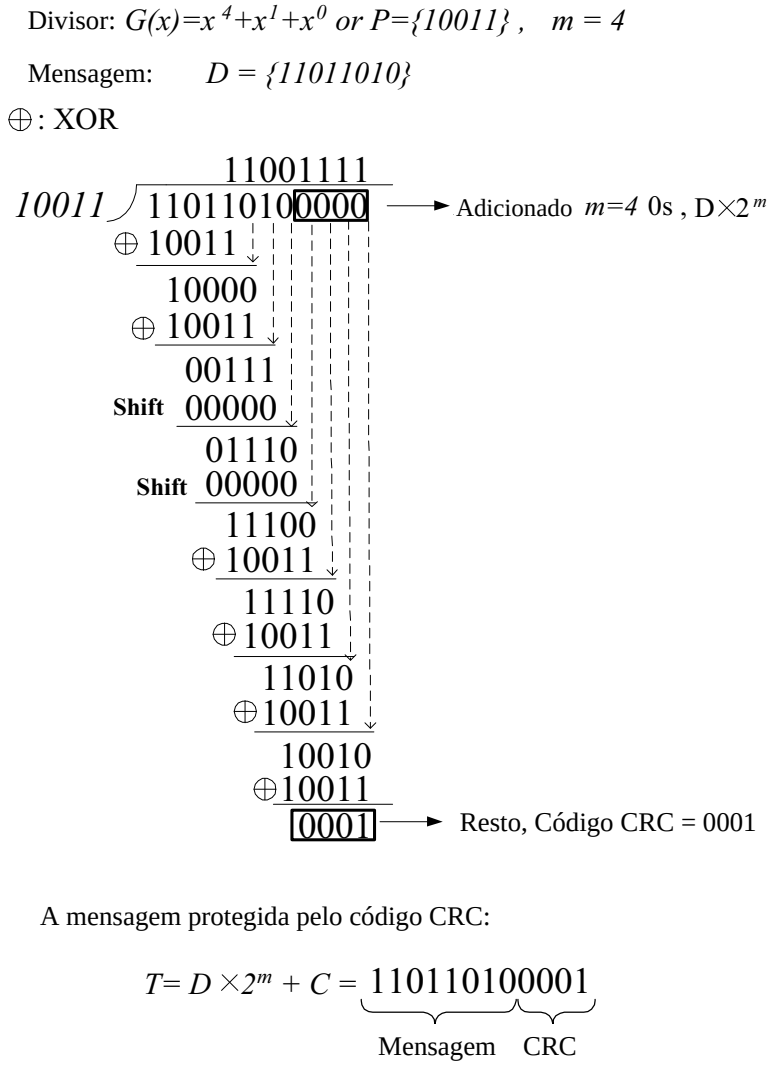

Figura 4: Exemplo de cálculo de CRC considerando um polinômio gerador de 4 *bits* (WEIDONG, 2003)

12 bytes:

- 9.600 ns para 10 Mbps *Ethernet*.
- 960 ns para 100 Mbps (*fast*) *Ethernet*.
- 96 ns para 1 Gbps (*gigabit*) *Ethernet*.

Diferentes protocolos podem ser encapsulados no campo de dados do quadro Ethernet para trafegar na rede. Um destes é o datagrama IP, o qual é formado e tratado a nível de rede.

A camada de rede possibilita a transferência de pacotes entre dispositivos de origem e destino remotos. A seguir apresenta-se a estrutura do datagrama IPv4, onde são descritos cada um de seus campos (POSTEL, 1981).

Quando um datagrama IPv4 é encapsulado no campo de dados do quadro Ethernet, o valor contido no campo Tipo, deve ser 0x0800. Este valor identifica que o protocolo IP está encapsulado na área de dados do quadro. A estrutura de um datagrama IPv4 é apresentada na Figura 5. Em seguida serão descritos cada um dos seus campos (POSTEL, 1981).

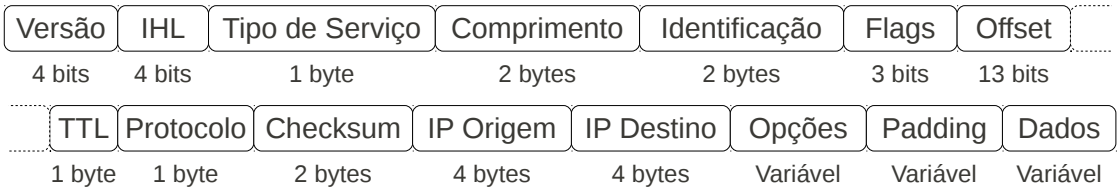

Figura 5: Estrutura de um datagrama IPv4. Figura adaptada de (POSTEL, 1981)

- Versão: informa a versão do protocolo IP com a qual foi criado o datagrama. É utilizado para verificar a compatibilidade do datagrama com o dispositivo que processa os datagramas IP.
- IHL (*Internet Header Lenght*): informa o tamanho do cabeçalho, em palavras de 32 bits. Como os campos Opções e *Padding* do datagrama não possuem tamanho fixo, faz-se necessário a utilização do campo IHL.
- Tipo de serviço: este campo é utilizado para definir qual vai ser o tratamento do datagrama na rede, identificando os datagramas por classes de serviços ou prioridade.
- Comprimento: é utilizado para informar o tamanho total do datagrama IP (cabeçalho e dados), em bytes. Como o datagrama IP não possui um campo para especificar o tamanho do campo Dados, este mesmo é calculado através da subtração entre os valores dos campos Comprimento e IHL.
- Identificação, *Flags* e *Offset*: estes campos são utilizados para controle de fragmentação e remontagem de datagramas.
- TTL (*Time To Live*): informa quanto tempo este datagrama ainda pode trafegar na rede. Cada dispositivo que processa o datagrama, altera o valor deste campo, subtraindo uma unidade de tempo do valor atual. Se o valor for igual a zero, o pacote é descartado.
- Protocolo: este campo informa o protocolo que foi utilizado para criar os dados contidos no campo Dados.

- *Checksum*: campo que armazena uma sequência de verificação, considerando apenas o cabeçalho do datagrama. Seu valor é obtido através do cálculo do complemento de 1, onde os bits de uma sequência são invertidos. Este campo deve ser recalculado pelos dispositivos que processam o datagrama, ao reencaminhá-lo, pois podem haver alterações em alguns campos, como no TTL.

- IP Origem: campo utilizado para armazenar o endereço IP do dispositivo que gerou o datagrama. O valor deste campo não muda, mesmo que o datagrama seja encaminhado por diversos dispositivos. A única exceção a isto se dá quando forem especificadas opções de NAT (*Network Address Translation*).

- IP Destino: armazena o endereço IP do dispositivo que irá receber o datagrama. Ao contrário do campo IP Origem, este valor nunca é alterado.

- Opções: o uso deste campo é opcional. Pode ser utilizado para definir os caminhos que o pacote deverá seguir na rede, através da especificação dos endereços IP. Além disso, pode-se armazenar neste campo os IPs dos dispositivos que encaminharam o datagrama.

- *Padding*: se o tamanho do cabeçalho do datagrama, em bits, não for um valor múltiplo de 32, neste campo é feito o preenchimento restante com bits 0s. Isto é necessário para garantir que o cálculo do campo IHL seja feito corretamente.

- Dados: armazena as informações a serem transportados no datagrama. Este campo geralmente contém outros protocolos encapsulados.

Além do IPv4, outra versão do datagrama pode ser encapsulada no campo de dados do quadro Ethernet. É o caso do datagrama IPv6, sendo que no momento do encapsulamento, o valor do campo Tipo deve ser igual a 0x86DD. A comunicação abordada na solução proposta deste trabalho é feita através de datagramas IPv6, sendo estes utilizados para o empacotamento e troca das informações. A estrutura de um datagrama IPv6 é apresentada na Figura 6. Em seguida serão descritos cada um dos seus campos (DEERING; HINDEN, 1998).

- Versão: assim como no IPv4, informa a versão do protocolo IP com a qual foi criado o datagrama. Neste caso, o valor é igual a 6.

- Classe de Tráfego: utilizado para definir qual vai ser o tratamento do datagrama na rede, da mesma forma como é feito no IPv4.

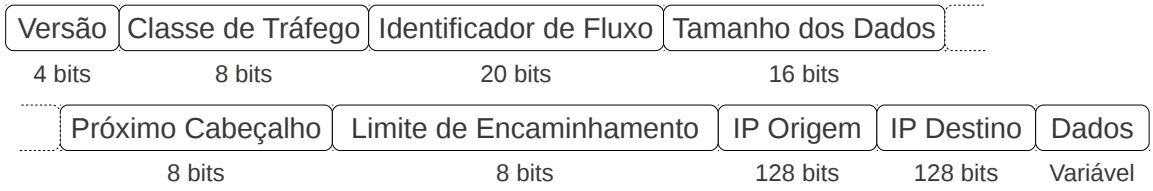

Figura 6: Estrutura de um datagrama IPv6. Figura adaptada de (DEERING; HINDEN, 1998)

- Identificador de Fluxo: informa os pacotes que pertencem ao mesmo fluxo de comunicação, separando assim os fluxos de cada uma das aplicações. Permite também que nós intermediários da rede realizem tratamentos específicos dos datagramas.
- Tamanho dos Dados: informa o tamanho do campo de dados que serão enviados após o cabeçalho, em bytes. Este campo substitui o campo Comprimento no protocolo IPv4, onde era indicado o tamanho total do datagrama, ou seja, o tamanho do cabeçalho somado ao tamanho do campo de dados. Contudo, também é somado a este campo, o tamanho dos cabeçalhos de extensão.
- Próximo Cabeçalho: similar ao campo Protocolo do datagrama IPv4, informa o protocolo que está encapsulado no campo de dados. Utiliza os mesmos valores do IPv4, além de identificar também o próximo cabeçalho de extensão.
- Limite de Encaminhamento: esse campo vai sendo decrementado de 1 a cada vez que passa por um nodo da rede. Indica o número máximo de nodos em que o datagrama pode passar antes de ser descartado.
- IP origem: indica o endereço de origem do datagrama.
- IP Destino: indica o endereço de destino do datagrama.
- Dados: assim como no IPv4, armazena as informações a serem transportados no datagrama. Este campo geralmente contém outros protocolos encapsulados, podendo ainda transportar os cabeçalhos de extensão.

Diferente do IPv4, o datagrama IPv6 oferece a opção de incluir, após o cabeçalho, alguns cabeçalhos extras chamados cabeçalhos de extensão. Com isso, alguns campos que eram fixos e obrigatórios no cabeçalho IPv4, passaram a ser opcionais ou então foram extintos. Abaixo apresenta-se os cabeçalhos de extensão existentes.

- Hop-by-Hop Options

- Routing
- Fragment
- Destination Options
- Authentication
- Encapsulating Security Payload

Um dos protocolos mais utilizados na camada de transporte, o qual é encapsulado no campo de dados do datagrama IP, é o TCP. Este protocolo, orientado à conexão, fornece uma comunicação de dados confiável, tratando o recebimento de pacotes fora de ordem ou o não recebimento de algum pacote. Outro protocolo utilizado na camada de transporte, porém não orientado a conexão, é o UDP. Esse protocolo não oferece o mesmo tratamento de dados feito no TCP, sendo portanto menos confiável. Porém, apresenta um desempenho superior em aplicações de de tempo real que necessitam uma baixa latência, sem que seja necessário um fluxo de dados livre de erros (embora as camadas superiores tratem os erros e retransmissões), como por exemplo áudio e vídeo (LAM; LIEW, 2004) (DOSTÁLEK; KABELOVÁ, 2006).

O protocolo UDP, descrito na RFC 768 (POSTEL, 1980) pode ser encapsulado em ambas versões do protocolo IP apresentadas. A estrutura de um datagrama UDP é apresentada na Figura 7, sendo cada um dos campos do cabeçalho detalhados a seguir.

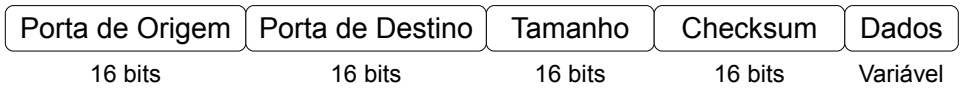

Figura 7: Estrutura de um datagrama UDP. Figura adaptada de (POSTEL, 1980)

- Porta de Origem: esse campo é utilizado para identificar os pontos extremos nas máquinas origem e destino, sendo representado por um número pré-definido que é associado a determinado tipo de aplicação.
- Porta de Destino: apresenta a mesma função do campo Porta de Origem, sendo este campo referente ao processo de destino que irá receber a mensagem. Uma listagem completa com os números das portas mais comuns existentes pode ser visualizada em (IANA, 2009).

- Tamanho: representa o tamanho, em bytes do datagrama, incluindo os tamanhos de cabeçalho e dados.
- Checksum: campo utilizado para a detecção de erros de transmissão, através da realização de operações de soma e complemento de 1 sobre os bits de cabeçalho e de dados.
- Dados: campo destinado ao transporte de dados de aplicação ou ainda outros protocolos de nível superior.

2.2 *Overhead* de protocolos de rede

Estudos demonstram (JOHN; TAFVELIN, 2007) que aproximadamente 90% a 95% do tráfego na internet corresponde aos protocolos TCP e IP, evidenciando que são os protocolos mais utilizados. Como o encapsulamento e desencapsulamento de dados é necessário para a comunicação, através de até 4 camadas do modelo OSI, são inseridos cabeçalhos de controle que geram um alto custo para a rede. A inserção destes cabeçalhos gera um consumo extra na utilização de banda, reduzindo o desempenho da rede em termos de aumento da latência e redução da vazão de dados.

A Figura 8 apresenta a relação do tamanho dos cabeçalhos com o tamanho da área de dados efetiva em uma transmissão UDP/IP. Nota-se que um grande número de bytes de controle, nos cabeçalhos, é necessário para a transmissão de dados na rede. Isso é um fator que acaba exercendo uma influência direta sobre a taxa da vazão de dados resultante na comunicação. Quanto menor é a área de dados utilizada, pior será o resultado da vazão efetiva. Ao enviar um conjunto de dados maior, a vazão resultante será maior também, pois utilizando-se o mesmo tamanho de cabeçalho, é possível enviar um número maior de informações na área de dados da camada superior.

Dependendo do conjunto de dados que a aplicação necessite transmitir, o tamanho da área de dados pode ser diversas vezes menor que o total de informações presente nos cabeçalhos. Pode-se utilizar como exemplo a aplicação Telnet, onde um pacote pode conter apenas 1 byte de dados. Nesse exemplo, em uma transmissão TCP/IPv6, o pacote resultante seria de 61 bytes, resultando em um *overhead* de aproximadamente 65% inserido pelo cabeçalho IP. Caso o tamanho da área de dados seja maior, com 1400 bytes por exemplo, o cabeçalho IP seria responsável por aproximadamente 2,7% do *overhead*, sendo este um valor pouco significativo.

Conforme observado em (JOHN; TAFVELIN, 2007), a maior parte dos pacotes IP que

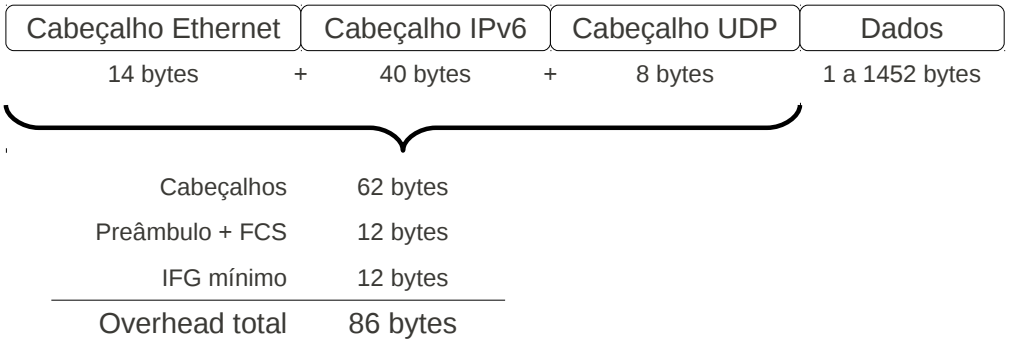

Figura 8: Overhead de protocolos

trafegam na rede são de tamanhos pequenos, até 100 bytes, representando 44% do total de pacotes transmitidos. Pacotes maiores, entre 1400 e 1500 bytes representam um total de 37%. Os demais tamanhos representam porcentagens pouco significativas. A Figura 9 apresenta a distribuição dos tamanhos de pacotes que trafegam na rede.

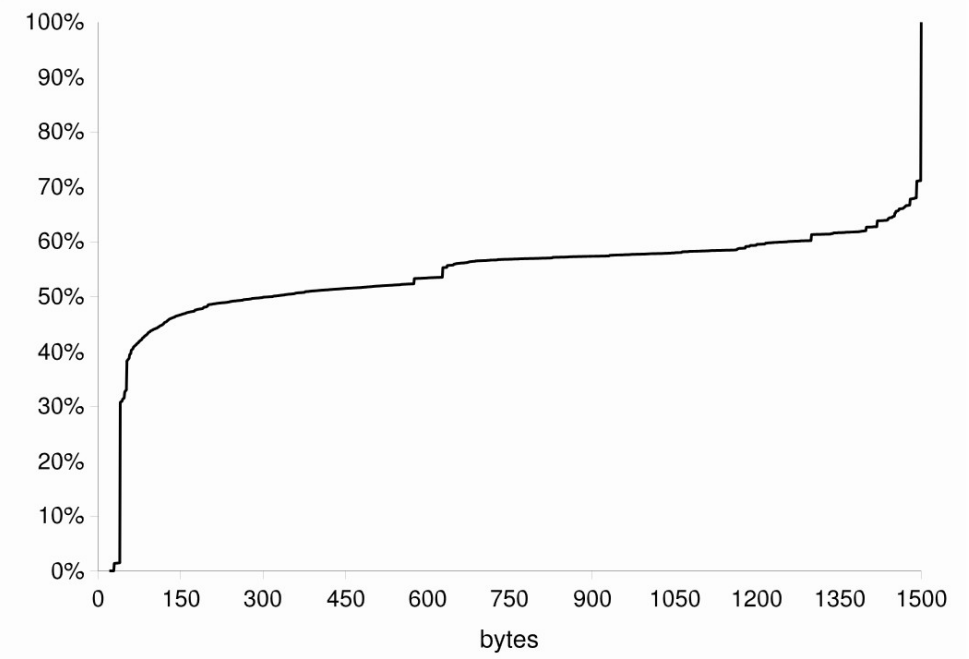

Figura 9: Distribuição dos tamanhos de pacote IP (JOHN; TAFVELIN, 2007)

Essa análise indica que muitos aplicativos utilizam pacotes pequenos para a comunicação, como no caso de aplicações interativas e de tempo real, onde muitas vezes o tamanho da área de dados é menor que o próprio tamanho do cabeçalho IP. Pode-se citar como exemplo o aplicativo Skype, apresentando uma média de 83 bytes no tamanho dos pacotes em uma comunicação local (ZHANG et al., 2010). Como a maior parte do tráfego é de pacotes pequenos, o *overhead* gerado é muito grande, reduzindo assim o desempenho

da rede.

No caso do tráfego para aplicações VoIP, o *overhead* causado pela inserção de cabeçalhos de protocolo é um fator limitante para definir a capacidade de uma rede local, conforme demonstrado em (KEEGAN; DAVIS, 2006). Isto implica na limitação do número de clientes VoIP simultâneos, conforme ocorre a saturação da rede pelo tráfego de dados.

A seguir são apresentados os cenários padrões recomentados para testes, que são utilizados para validação de dispositivos de interconexão. Estes cenários definem a estrutura necessária e interligação dos dispositivos para a realização de testes. Após isto, são descritos cada um dos testes padrões que devem ser efetuados seguindo estes cenários.

2.3 Topologia Padronizada para Testes

A fim de verificar o desempenho e conformidade dos diversos equipamentos presentes em uma rede Ethernet, devem ser realizados alguns testes. Para tanto, foram criadas padronizações (RFCs - *Requests For Comments*) de testes, a fim de definir características funcionais e metodologias de testes para os equipamentos de comunicação. A principal RFC utilizada para testar os dispositivos de interconexão é a RFC 2544 (*Benchmarking Methodology for Network Interconnect Devices*) (BRADNER; MCQUAID, 1999). Esta RFC define 6 testes diferentes, que serão descritos na próxima seção.

A realização dos testes da RFC 2544, é feita seguindo um dos cenários de testes propostos pela própria RFC 2544. Um primeiro cenário, é composto pelo dispositivo em teste (DUT - *Device Under Test*) e por por dois equipamentos testadores, sendo um responsável pela a transmissão e o outro pela a recepção, conforme demonstrado na Figura 10 [a].

Este cenário possui uma desvantagem, que é a necessidade de sincronização dos tempos dos dois testadores. Além de aumentar a dificuldade de implementação, isto pode influenciar negativamente os testes que envolvem a apuração de tempos, como por exemplo os testes de latência e reinicialização.

No segundo cenário, Figura 10 [b], apenas um equipamento de testes é responsável por fazer a transmissão (através de uma interface de rede) e a recepção dos dados (através de uma segunda interface). Com isto, a sincronia entre os tempos de quadros enviados e recebidos é garantida, oferecendo uma precisão maior nos resultados dos testes. Isto torna o segundo cenário mais vantajoso que o primeiro, e a utilização do mesmo nos testes é recomendada pela RFC (BRADNER; MCQUAID, 1999).

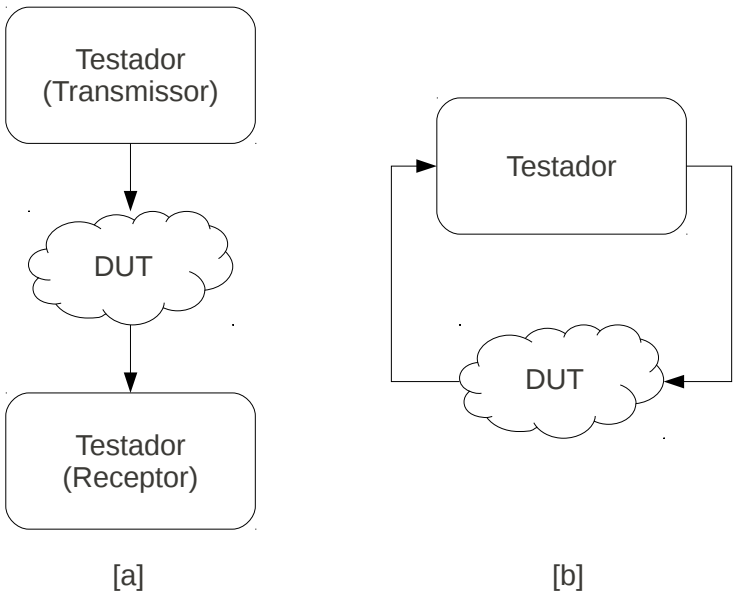

Figura 10: Cenário de testes sugerido pela RFC 2544. Figura adaptada de (BRADNER; MCQUAID, 1999)

Caso a comunicação entre o testador e o DUT esteja limitada a no máximo uma interface de rede, o cenário demonstrado na Figura 10 [b] não funcionaria corretamente, dependendo do DUT. Isto ocorre porque os quadros enviados pelo testador devem ter os endereços MAC destino e origem iguais, para que retornem ao testador pela mesma interface. Alguns equipamentos (DUT) não permitem que este tipo de quadro continue trafegando, e assim fazem a filtragem e o descarte dos quadros.

A solução para isto é a utilização de um segundo equipamento que faz o *loop* dos quadros. Na Figura 11 pode-se observar que um equipamento testador gera os quadros e um segundo, em modo *loopback* os reencaminha de volta ao equipamento gerador. Inicialmente, os quadros gerados possuem como destino o testador que está no modo *loopback*. Ao receber um quadro, este testador faz a troca dos endereços MAC e reenvia ao DUT, que por sua vez encaminha ao novo destinatário do quadro, ou seja, o equipamento gerador. Este cenário dispensa a sincronia de tempo entre os dois testadores.

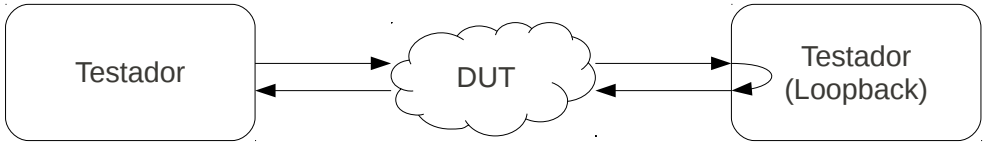

Figura 11: Cenário de testes com *loopback*. Figura adaptada de (BRADNER; MCQUAID, 1999)

A RFC 2544 recomenda que sejam realizados os seis testes descritos na seção seguinte, e para cada teste, sejam utilizados pelo menos 5 tamanhos diferentes de quadros. Nestes, devem constar os tamanhos mínimo e máximo permitido, que são respectivamente 64 e 1518 bytes. Os tamanhos de quadros recomendados para realização dos testes, são de 64, 128, 256, 512, 1024, 1280 e 1518 bytes.

2.4 Testes Ethernet

Atualmente, a grande maioria dos equipamentos utilizados para realizar testes Ethernet, seguem a recomendação da RFC 2544. A seguir, serão brevemente descritos os seis testes dessa RFC. Estes testes devem ser realizados de acordo com um dos cenários citados anteriormente (BRADNER; MCQUAID, 1999).

- Vazão: é definida como a quantidade máxima de dados que pode ser enviada em uma transmissão, sem que haja o descarte de quadros. A realização deste teste se dá com o envio de um número determinado de quadros, com tamanho e taxa específicos. Após isto, deve-se contabilizar o número de quadros corretamente recebidos, que deve ser igual ao número de quadros enviados. Se esta comparação for verdadeira, repete-se o teste, com uma taxa de transmissão maior que a anterior. Caso contrário, se algum quadro não for corretamente recebido, diminui-se a taxa, e realiza-se o teste novamente. Este procedimento deve ser repetido até que seja alcançado o resultado da maior taxa de transmissão, sem que haja qualquer perda de quadros.

- Latência: define-se com o tempo em que um quadro leva para chegar da origem ao destino. Para isto, calcula-se o atraso na recepção após o envio do quadro. Primeiramente, deve ser definida a vazão para cada tamanho de quadro, com o teste anteriormente citado. A vazão máxima é utilizada para fazer o envio de uma série de quadros (pelo menos 120 segundos de transmissão) contendo uma marca de tempo. Ao retornar ao dispositivo, calcula-se o atraso dos quadros através da subtração do tempo em que um quadro foi enviado, pelo tempo em que o quadro foi recebido, assim o resultado é obtido através da média destes tempos.

- Taxa de perda de quadros: este teste define a porcentagem de quadros que foram transmitidos pela origem, e não foram recebidos pelo destino. A execução do teste se dá com a transmissão de rajadas constantes de quadros, inicialmente a uma taxa de 100% da vazão para cada tamanho de quadro. Em seguida, diminui-se a taxa em 10%, envia-se uma nova rajada, e assim sucessivamente até que chegue aos 10% da

vazão. O resultado é obtido através do cálculo da porcentagem de perda para cada rajada transmitida na execução do teste.

- *Back-to-Back frames*: através deste teste, é possível definir a quantidade máxima de quadros contínuos suportados por um DUT em uma rajada, com o tempo mínimo definido entre cada quadro. Este teste é feito enviando-se uma rajada de quadros com intervalo de tempo entre quadros (IFG - *Inter Frame Gap*) mínimo ao DUT e contabilizando o número de quadros recebidos. Caso o número de quadros recebidos seja igual ao número de quadros transmitidos, aumenta-se a duração da rajada realiza-se o teste novamente. Caso o número de quadros recebidos seja menor do que o número de quadros transmitidos, reduz-se a duração da rajada e realiza-se o teste novamente. O resultado é definido através da maior rajada de quadros que o DUT conseguir tratar sem que ocorra a perda de quadros. Este teste deve ser realizado durante no mínimo 2 segundos e repetido pelo menos 50 vezes para cada tamanho de quadro.

- Recuperação do sistema: o objetivo deste teste é identificar a velocidade com que o DUT recupera-se de uma condição de sobrecarga. Para iniciar o teste, primeiramente determina-se a vazão para cada tamanho de quadro. Após isto, envia-se ao DUT uma rajada de quadros com aproximadamente 110% da vazão encontrada para cada tamanho de quadro. A transmissão deve durar, no mínimo, 60 segundos, para que ocorra o estouro de memória no DUT. Com isto, o DUT para de responder, e então o testador deve marcar um tempo inicial de referência. Após isto, deve ser enviada uma nova rajada de quadros ao DUT, com a taxa de quadros reduzida em 50%. No momento em que os quadros começarem a retornar ao testador, uma nova marcação de tempo deve ser feita. Calcula-se o tempo de resposta do DUT subtraindo o tempo final pelo tempo inicial. O teste deve ser realizado diversas vezes para que se tenha uma amostragem estatística confiável.

- Reinicialização: este teste busca definir o tempo que um DUT leva para recuperar-se de uma reinicialização por software ou hardware. A execução do teste é iniciada com a geração de um fluxo contínuo de quadros ao ser transmitido ao DUT. A seguir, reinicializa-se o DUT por software ou hardware, e a contagem de tempo deve iniciar quando o testador receber o último quadro. No momento em que o testador começar a receber novamente os quadros, finaliza-se a contagem do tempo, e obtém-se o resultado do teste que indica o tempo necessário ao DUT para se recuperar de uma reinicialização.

A partir dessa fundamentação teórica, é apresentado na próxima seção os trabalhos relacionados. Foi realizado um levantamento sobre as principais implementações de pilhas UDP/IP em hardware, assim como metodologias de testes e validação, para a devida avaliação de desempenho de novas arquiteturas.

2.5 Trabalhos Relacionados

Visando reduzir o *overhead* do protocolo IPv6, gerado pelo excessivo tamanho do cabeçalho, surgiram algumas propostas para reduzir o volume de dados do cabeçalho durante a transmissão de pacotes na rede. Em (LIM; STERN, 2000), (WESTPHAL; KOODLI, 2005) e (XUE-ZHOU; ZHI-LING; SI-MIN, 2009), os autores propõe técnicas de compressão do cabeçalho IPv6, reduzindo significativamente o seu tamanho.

São utilizados algoritmos de compressão para a redução do tamanho do cabeçalho, para que só entao o pacote IPv6 seja transmitido. Ao receber um pacote com o cabeçalho comprimido, a estação deve descomprimí-lo para sua forma original, para então tratar os dados originais contidos nos campos desse cabeçalho. Os algoritmos utilizam métodos de atribuição de endereçamento e codificação/decodificação aritmética para a redução do tamanho do cabeçalho IPv6. Além disso, os algoritmos de compressão baseiam-se na similaridade entre cabeçalhos de pacotes consecutivos em um fluxo de dados estabelecido, para o processamento. Com isso, a compressão e a descompressão são realizadas de maneira síncrona, baseando-se em um fluxo específico de comunicação.

Estes são métodos complexos que exigem um grande esforço de implementação e validação, além de inserir um determinado custo de processamento para os processos de compressão e descompressão dos cabeçalhos durante as transmissões dos pacotes. Além disso, é necessário que seja estabelecido um contexto de fluxo para a comunicação, o que acaba gerando um *overhead* extra.

Ao invés de utilizar técnicas de compressão e descompressão do cabeçalho, o trabalho proposto por (MURUGESAN; RAMADASS; BUDIARTO, 2009), baseia-se na customização do cabeçalho IPv6. São apresentados resultados teóricos a respeito da redução do *overhead* inserido na rede pelo cabeçalho IPv6, com o aumento da vazão de dados na rede local.

A Figura 12 apresenta os ganhos relacionados à vazão, obtidos com a redução do cabeçalho. São apresentados dois tamanhos de cabeçalho IPv6 customizado, sendo um de 20 bytes para uma comunicação externa à rede local (LAN para WAN), e outro de 1 byte utilizado para comunicação na rede local (LAN para LAN). Nota-se uma melhora

no desempenho, principalmente para pequenos pacotes. Essa melhora representa um aumento no desempenho da vazão em uma LAN de até 73% para pacotes de 128 bytes, e de 23% para pacotes de 256 bytes.

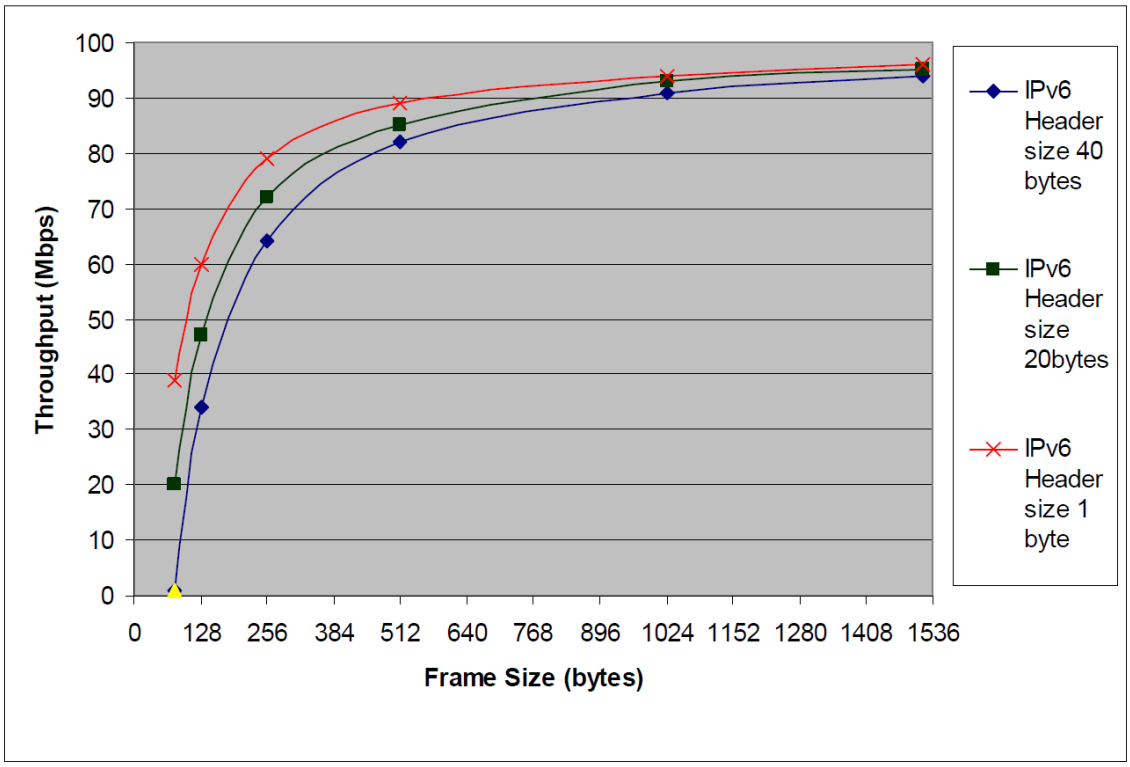

Figura 12: Relação teórica da vazão para cabeçalho customizado (MURUGESAN; RAMADASS; BUDIARTO, 2009).

A customização do pacote se dá entre as camadas de enlace e de rede, onde um módulo de customização é responsável por remover alguns campos do cabeçalho IPv6, que não são relevantes para comunicação de dados local, e então enviar o pacote IP customizado. Ao receber um pacote com o cabeçalho customizado, o processo inverso é realizado, deduzindo-se os valores relevantes de campos do cabeçalho IPv6 padrão, para só então ser tratado pelas camadas superiores.

O cabeçalho customizado resultante para uma transmissão entre duas máquinas que encontram-se na mesma rede local, possui um total de 1 byte de tamanho. Mantém-se o campo Próximo Cabeçalho, sendo os demais campos todos removidos. Para uma transmissão em que o destinatário do pacote esteja fora da rede local, o cabeçalho customizado mantém os campos Versão, Identificador de Fluxo, Próximo Cabeçalho e parte do endereço de destino. Os demais campos são removidos, sendo que o roteador ao receber este pacote, reconstrói o cabeçalho no seu formato original, para só então encaminhar ao seu destinatário na internet. A reconstrução do cabeçalho IPv6 customizado se dá com a inserção dos campos anteriormente removidos, baseando-se nas redundâncias encontradas

entre as camadas inferiores de comunicação.

Esta solução mostra-se bastante eficiente, pois reduz consideravelmente o tamanho do cabeçalho, sem a necessidade de um alto custo de processamento. O ganho apresentado está relacionado ao aumento de desempenho na transmissao de pacotes IPv6 através da LAN em termos de utilização de banda, reduzindo a latência e aumentando a vazão.

Visando reduzir o atraso de processamento em software, com o encapsulamento/-desencapsulamento de pacotes de dados, surgiram algumas propostas de implementação das pilhas de comunicação em hardware. Em (HERRMANN, 2010) foi proposta a implementação da pilha UDP/IPv4 em hardware reconfigurável, visando com isso aumentar a vazão de dados na codificação e decodificação das informações relativas aos protocolos de comunicação.

No referido trabalho foram apresentadas três implementações distintas da pilha UDP/IP em *Field Programmable Gate Array* (FPGA), utilizando-se a linguagem Verilog. As três arquiteturas implementam as camadas de de transporte, rede e enlace. Além disso, foi desenvolvido também em Verilog um módulo responsável por fazer o *loopback* dos dados, representando assim a camada de aplicação.

A primeira versão da pilha UDP/IP é a mais simples, apresenta a menor utilização de área, porém com o menor desempenho entre as três versões. A estrutura dessa versão é apresentada na Figura 13.

Além dessa, outras duas versões foram desenvolvidas. Na segunda versão, o barramento de comunicação dos pinos de dados foi incrementado de 8 para 32 *bits*, visando melhorar o desempenho da camada de transporte com a camada de aplicação. Já na terceira versão do *core*, foi adicionada à arquitetura um buffer de armazenamento de pacotes, visando um aumento na vazão de dados durante a realização dos testes.

Nos resultados apresentados por (HERRMANN, 2010), foi comprovado que apenas a terceira versão da pilha UDP/IP conseguiu atingir 100% de vazão para todos os tamanhos de quadros nos testes da RFC 2544, além de não apresentar qualquer perda de quadros. Na segunda versão foi atingida uma vazão de 45% para quadros a partir de 256 *bits*, porém na primeira versão não foi possível determinar, sendo que essa apresentou perda de quadros para todas as taxas e tamanhos de quadros.

Nos testes relacionados à latência, todas a versões apresentaram resultados muito superiores que em pilhas de comunicação utilizadas em PC, sendo que na terceira versão, os valores de latência foram de 389 a 13 vezes menores para quadros de 64 e 1518 *bytes*,

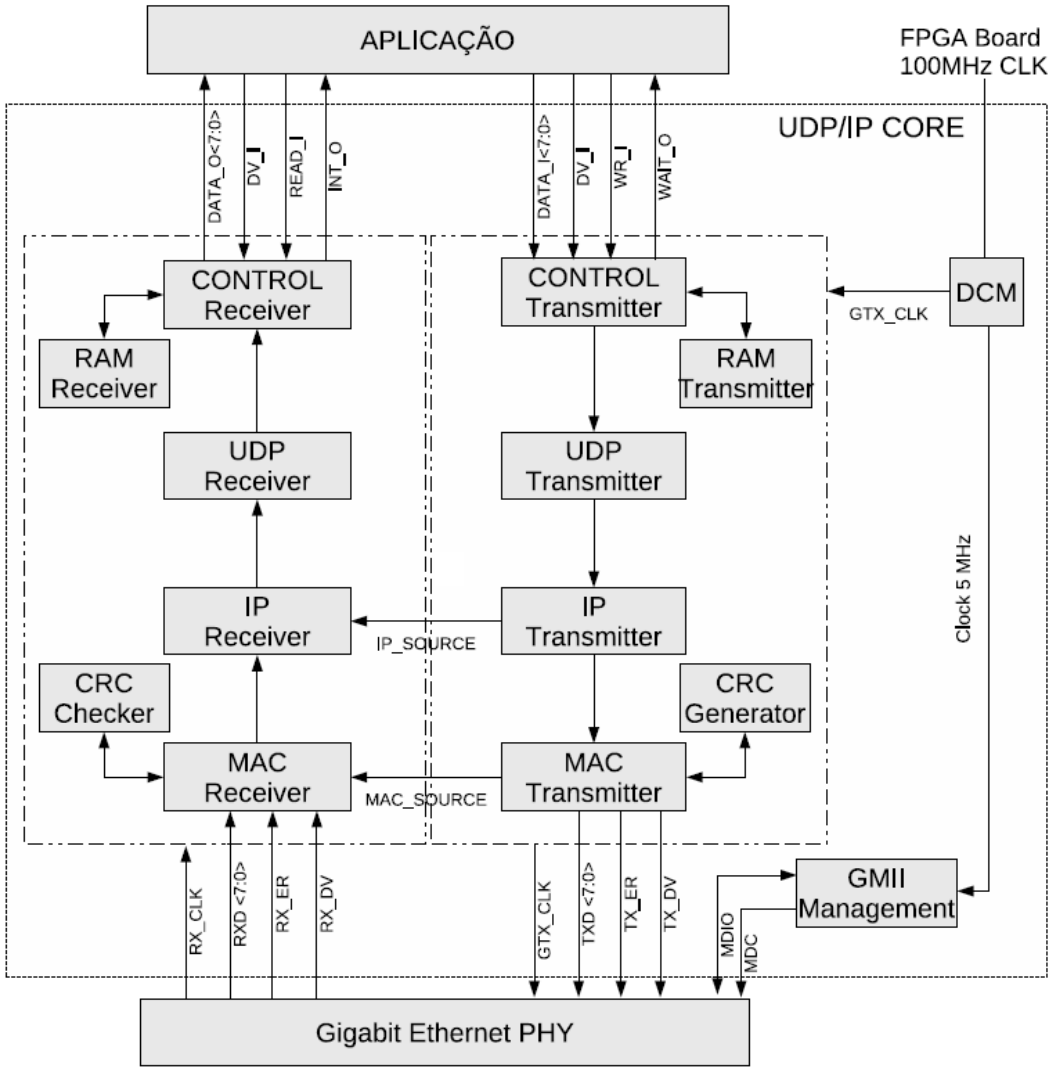

Figura 13: Estrutura da pilha UDP/IP em FPGA proposta por (HERRMANN, 2010).

respectivamente.

Em (LÖFGREN et al., 2005) são apresentadas três propostas de implementação da pilha UDP/IP num FPGA Spartan-3 xc3s200 da Xilinx (XILINX, 2010a). Nesse trabalho os autores descrevem e analisam o paralelismo de uma pilha UDP/IP implementada em FPGA para sistemas *Ethernet* embarcados. As três implementações da pilha UDP/IP são "*Minimum*", "*Medium*", "*Advanced*". Elas possuem a estrutura apresentada na Figura 14 e implementam a camada de transporte, de rede e de enlace.

Dentre as três, a implementação "*Advanced*"é a que apresenta melhores resultados de desempenho, mas também é a que ocupa mais área em relação as outras implementações. Detalhes dessas três implementações podem ser verificados na Tabela 1. (LÖFGREN et al., 2005) afirma que a implementação "*Advanced*"apresenta vazão em torno de 957 Mbps

quando excluído o processamento relacionado ao preâmbulo, cabeçalhos UDP/IP, CRC e IFG, mas afirma também que o desempenho geral do sistema está diretamente relacionado à latência presente na camada de aplicação.

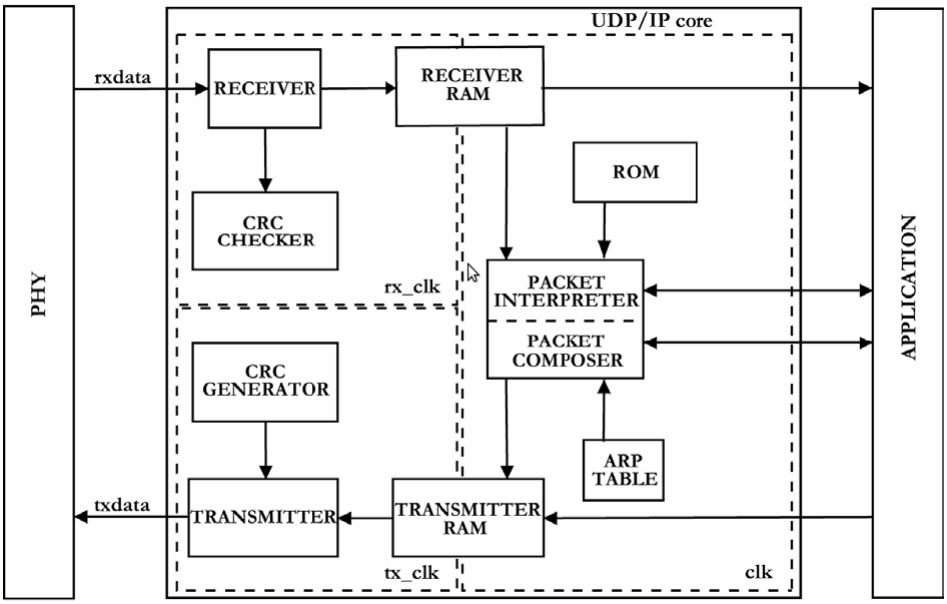

Figura 14: Estrutura da pilha UDP/IP em FPGA proposta por (LÖFGREN et al., 2005).

Tabela 1: Comparação das implementações propostas por (LÖFGREN et al., 2005).

Spartan-3, xc3s200-4ft256			
	"Minimum"	"Medium"	"Advanced"
Xilinx Slices	517	1022	1584
Xilinx BRAMs	3	3	5
Fmax (MHz)	90,7	60,3	105,6
Length (bytes) *	256	256	1518
Duplex mode	Full	Full	Full
Speed (Mbps)	10/100	10/100	10/100/1000
ARP	No	4 entries	4 entries
RARP	No	No	Yes
ICMP	No	Yes	Yes
TCP "channel"	No	No	Yes
Flexibility	Low	Medium	High

* Não considerando o preâmbulo.

As implementações apresentadas por (LÖFGREN et al., 2005), atualmente estão vinculadas à empresa *RealFast*, que apresenta UDP/IP *Cores* para comunicação *Ethernet* de tempo real (REALFAST, 2010).

Baseando-se nos resultados dos trabalhos apresentados nesta Seção, observa-se que uma implementação da pilha UDP/IP a nível de hardware, tem um ganho bastante significativo de desempenho, quando comparado a implementações em PCs, onde o processamento é feito a nível de software pelo sistema operacional. Na próxima Seção, apresenta-se

a arquitetura proposta para uma nova implementação de uma pilha de comunicação baseada no protocolo IPv6, visando otimizar o tráfego de dados em uma rede local. Isto é feito minimizando-se o tamanho do cabeçalho IPv6 durante a comunicação entre dois dispositivos, fazendo com que sejam transmitidas apenas informações relevantes para o tráfego de dados local.

3 ARQUITETURA PROPOSTA

Nesse capítulo, apresenta-se a arquitetura proposta para a obtenção de um melhor desempenho na comunicação local IPv6, bem como o seu desenvolvimento e implementação em software e hardware reconfigurável. Além disso, é apresentada a especificação da estrutura e funcionamento interno das arquiteturas. Por fim, apresenta-se a metodologia e o ambiente utilizado no desenvolvimento e na validação da arquitetura proposta.

Baseado nas informações apresentadas nos capítulos anteriores, conclui-se que o protocolo IP foi concebido para atender às demandas de redes WAN, com topologias mais complexas para permitir a comunicação entre dispositivos que não fazem parte da mesma rede local. Porém este protocolo não é otimizado para redes LAN, onde os dispositivos partilham do mesmo meio físico para a troca de dados. Como esta troca de dados envolve várias etapas de encapsulamento e desencapsulamento das diferentes camadas de protocolos, o desempenho de comunicação é reduzido. Diversas informações são inseridas nos cabeçalhos de protocolos, sendo que muitas dessas na verdade são irrelevantes ou redundantes para a comunicação entre dispositivos de uma mesma rede local. A Figura 15 apresenta os campos redundantes entre o quadro Ethernet e o cabeçalho do datagrama IPv6 padrão para uma comunicação local, além de destacar com preenchimento os campos irrelevantes.

Como o datagrama IP é encapsulado no campo de dados do quadro Ethernet, os endereços IPv6 de origem e destino de um datagrama podem ser inferidos a partir dos endereços origem e destino do cabeçalho Ethernet. Já ao campo Versão, pode-se atribuir o seu valor com base no campo Tipo do cabeçalho Ethernet. Estas são informações redundantes em uma comunicação local entre dois dispositivos, portanto podem ser omitidas durante a comunicação de dados.

Além dos campos IP Origem e IP Destino indicados na Figura 15, os campos com preenchimento, presentes no cabeçalho IPv6 apresentam informações irrelevantes para o tráfego de dados local. Estas informações, são utilizadas apenas pelos demais dispositivos que compõe a rede de internet, como roteadores e *gateways*, sendo portanto desnecessárias

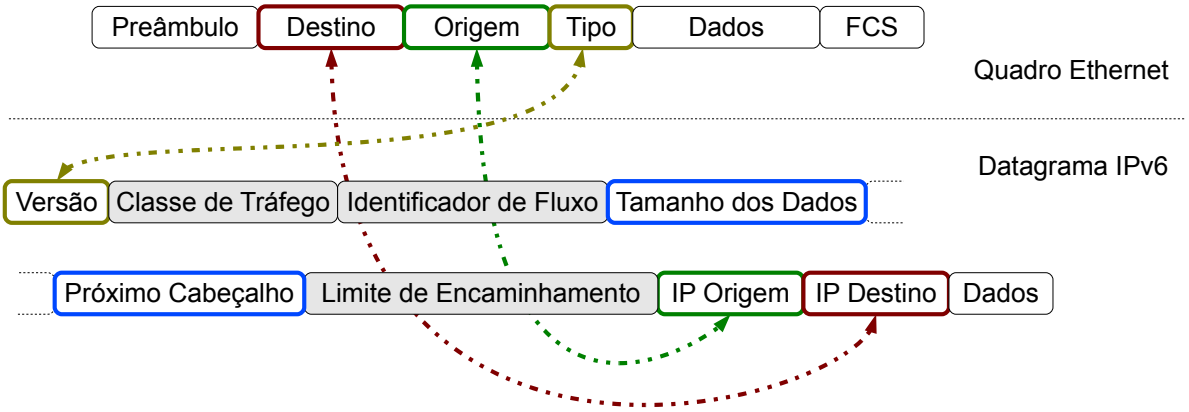

Figura 15: Redundância de dados entre cabeçalho Ethernet e cabeçalho IPv6 para comunicação local

para os dispositivos de origem e destino. Já os campos Tamanho dos Dados e Próximo Cabeçalho, são uma exceção, não constando em nenhum outro campo do cabeçalho Ethernet, além de ter suma importância para o correto tratamento do datagrama.

Partindo dessas observações, este trabalho propõe o desenvolvimento de uma arquitetura que permita otimizar o tráfego de dados através do protocolo IPv6 em uma rede local. Isto é feito reduzindo-se ao máximo possível o tamanho do cabeçalho IP, criando assim um cabeçalho reduzido que transmita apenas informações relevantes durante a comunicação. Esta abordagem explora alguns recursos avançados do IPv6, como a auto-configuração de endereçamento e redundâncias entre camadas.

Como os campos de endereçamento do cabeçalho IPv6 podem ser inferidos através dos dados contidos no cabeçalho Ethernet, estes podem ser omitidos durante a transmissão de pacotes na LAN. O mesmo ocorre com o campo Versão, no qual o valor é fixado em '6' ou inferido pelo campo Tipo do cabeçalho Ethernet. Os campos Classe de Tráfego e Identificador de Fluxo, não são utilizados quando o tráfego é exclusivamente local, portanto podem ser completamente excluídos, assim como o campo Limite de Encaminhamento, que para tráfego local vai ser de apenas 1 encaminhamento. Assim, o cabeçalho reduzido é formado apenas pelos campos Tamanho dos Dados e Próximo Cabeçalho, que são seguidos então pelo campo de dados do datagrama IP. Essa estrutura pode ser observada na Figura 16.

O datagrama com o cabeçalho reduzido é encapsulado no quadro Ethernet pelo dispositivo de origem, e então enviado para o dispositivo de destino. Ao receber e identificar o datagrama com o cabeçalho reduzido, o dispositivo de destino reconstrói o datagrama IPv6 na sua forma original antes de ser tratado pela camada de rede.

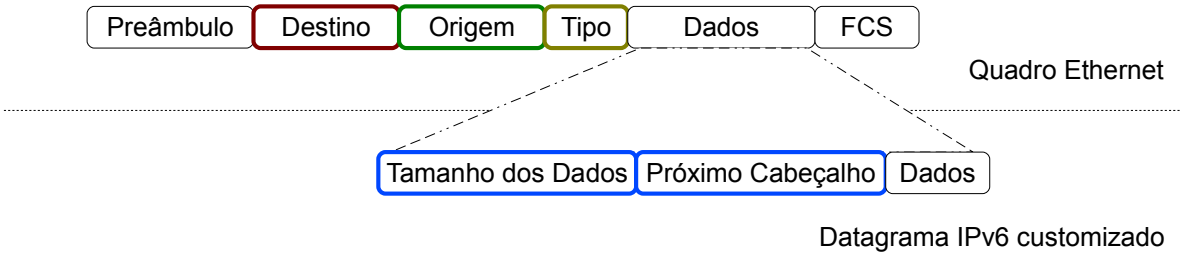

Figura 16: Datagrama com o cabeçalho reduzido

A reconstrução do cabeçalho IPv6 padrão, a partir do cabeçalho reduzido, se dá inicialmente através da geração dos endereços origem e destino. Como os dispositivos de origem e destino estão na mesma rede local, o prefixo de rede é o mesmo em ambos os casos. Este valor é inserido na primeira parte dos endereços IPv6 origem e destino, conforme padrão. A outra parte do endereçamento, que contempla o identificador de interface, é inferida a partir dos endereços de origem e destino do quadro Ethernet. Estes endereços são convertidos para o formato EUI-64, conforme definido em (CRAWFORD, 1998) e (HINDEN; DEERING, 2006). Essa conversão é exemplificada na Figura 17, onde a partir do endereço Ethernet de 48 *bits*, é gerado o interface ID no formato EUI-64. No primeiro octeto do endereço Ethernet, inverte-se o segundo bit menos significativo, que indica um escopo universal ou local para o endereço. Após o terceiro octeto do endereço Ethernet, são inseridos outros dois octetos com o valor hexadecimal fixo de 'FF FE'. Na sequência são inseridos os demais últimos três octetos do endereço Ethernet, e com isso obtém-se os endereços IPv6 de origem e destino completos.

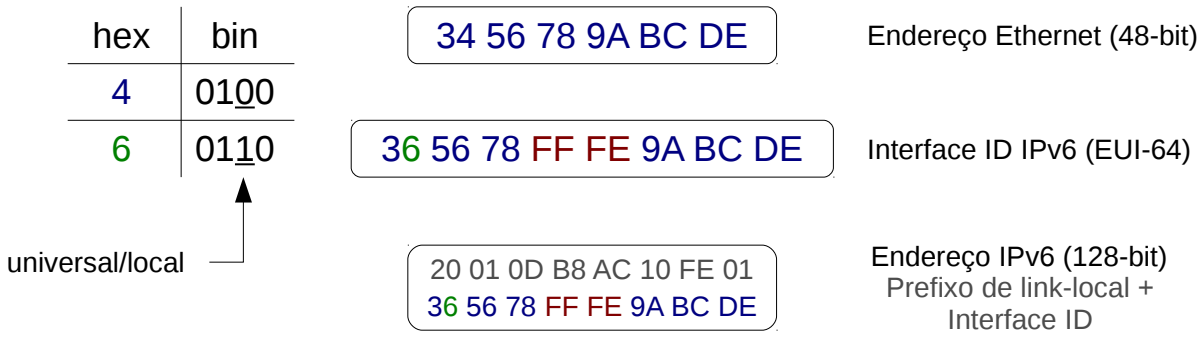

Figura 17: Exemplo de conversão de endereço Ethernet para endereço IPv6. Figura adaptada de (CRAWFORD, 1998) e (HINDEN; DEERING, 2006)

Após a obtenção dos endereços, atribui-se ao campo Versão o valor '6', e aos campos Classe de Tráfego e Identificador de Fluxo é atribuído o valor zero. Por fim, adiciona-se

os campos Tamanho dos Dados e Próximo Cabeçalho, contidos no cabeçalho reduzido. Com isso o datagrama IPv6 está restaurado para sua forma original, e pronto para ser tratado normalmente pela camada de rede.

O processo de construção do cabeçalho reduzido e reconstrução do cabeçalho IPv6 padrão é implementado de forma transparente entre as camadas de enlace e de rede dos dispositivos que compõe a rede local. Essa abordagem aumenta o desempenho de transmissão, apenas com um pequeno custo de processamento, sendo que este custo não apresentou valores significativos nos testes realizados.

Os módulos que permitem esta conversão foram desenvolvidos inicialmente em software, para uma análise a respeito da funcionalidade e eficiência do método proposto. Posteriormente foi desenvolvida uma versão em hardware reconfigurável, sobre uma pilha de comunicação IPv6, permitindo uma análise mais detalhada a respeito do ganho de desempenho quando comparado a implementações padrão do protocolo IPv6.

3.1 Pilha de comunicação IPv6 em software

A implementação da pilha de comunicação IPv6 em software, deu-se com o desenvolvimento de um aplicativo na linguagem de programação C. A principal funcionalidade desse aplicativo, é a realização do *loopback* de pacotes enviados para a interface de rede de um computador durante um teste. Esse aplicativo é executado sobre um sistema operacional Linux, sendo compilado através do GNU Compiler Collection (GCC).

A transmissão e a recepção de pacotes é realizada através de *sockets*, mais especificamente do tipo SOCK_RAW (KERRISK, 2013). Esse tipo de *socket*, permite realizar a manipulação de dados a nível de enlace. Com isso é possível gerenciar cabeçalhos e dados desde o protocolo Ethernet, até os níveis superiores, comunicando-se diretamente com o *kernel* do Linux.

Utilizando *sockets* do tipo SOCK_RAW, a compatibilidade dos protocolos IPv6 e IPv6 reduzido, é fornecida simultaneamente. A diferenciação entre os protocolos na recepção de um pacote, se dá através do campo Tipo do protocolo Ethernet. Quando um pacote IPv6 padrão é recebido, o valor contido nesse campo é igual a 0x86DD. No caso do pacote IPv6 reduzido, o valor passa a ser 0x88B6.

A arquitetura da pilha de comunicação em software, compreende duas *threads* principais. Uma delas é utilizada para a recepção e tratamento dos pacotes IPv6 padrão e reduzido. A outra é responsável pela transmissão dos mesmos pacotes, completando as-

sim o *loopback* utilizado durante a aplicação dos testes. Os pacotes são temporariamente armazenados em um *buffer*, permitindo assim a recepção e a transmissão simultânea de dados. Apresenta-se a seguir um detalhamento sobre cada uma das *threads* implementadas.

3.1.1 *Thread* de recepção

Essa *thread* é responsável pela recepção dos pacotes oriundos da placa de rede do computador. A Figura 18 apresenta um fluxograma ilustrando o processo de recepção. Inicialmente, é criado um *socket* de recepção, que mantém a *thread* bloqueada até ocorra o recebimento de um pacote. Após isso, todos os pacotes passam por um filtro, que define o caminho a ser tomado no fluxo de processamento. Se o pacote for desconhecido, o mesmo é descartado.

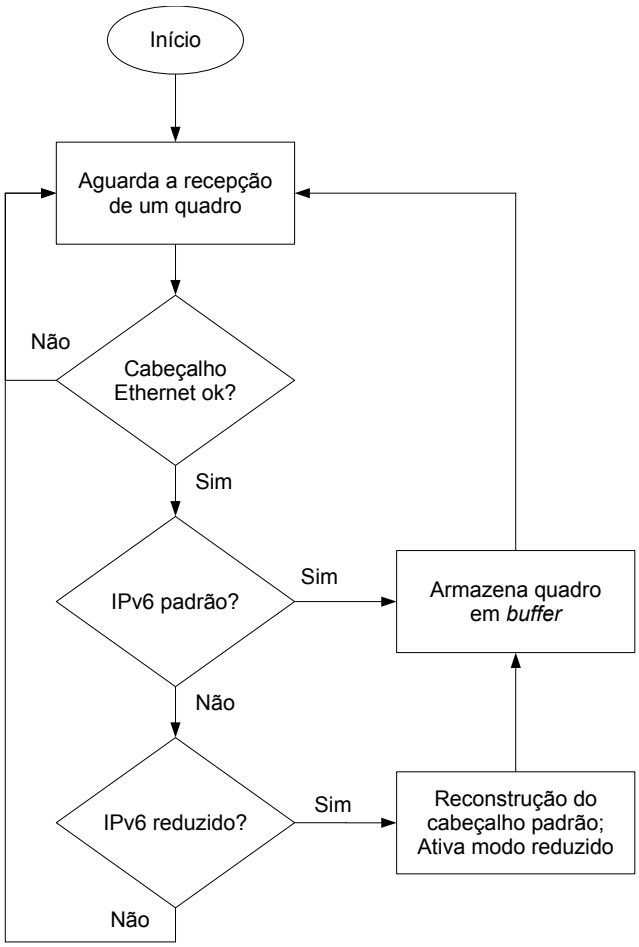

Figura 18: Fluxograma de recepção

No caso do recebimento de um pacote IPv6 reduzido, aplica-se o processo de reconstrução do cabeçalho para o formato IPv6 original, e após isso o pacote é armazenado em

buffer. Se o pacote recebido for do tipo IPv6 padrão, o mesmo passa a ser diretamente armazenado, sem sofrer qualquer tipo de tratamento.

3.1.2 *Thread* de transmissão

Na *Thread* de transmissão é onde os pacotes armazenados em *buffer* são enviados para a interface de rede do computador através do *kernel*. Num primeiro momento, a *thread* monitora o *buffer* até que um novo pacote esteja disponível para a transmissão. Assim que isso ocorrer, o pacote pode ser simplesmente enviado através de um *socket* de transmissão, ou então sofrer o processo de redução do cabeçalho IPv6, para só então ser enviado. Isso vai depender do tipo de pacote que havia sido recebido originalmente pela aplicação. Esse processo pode ser observado no fluxograma ilustrado na Figura 19.

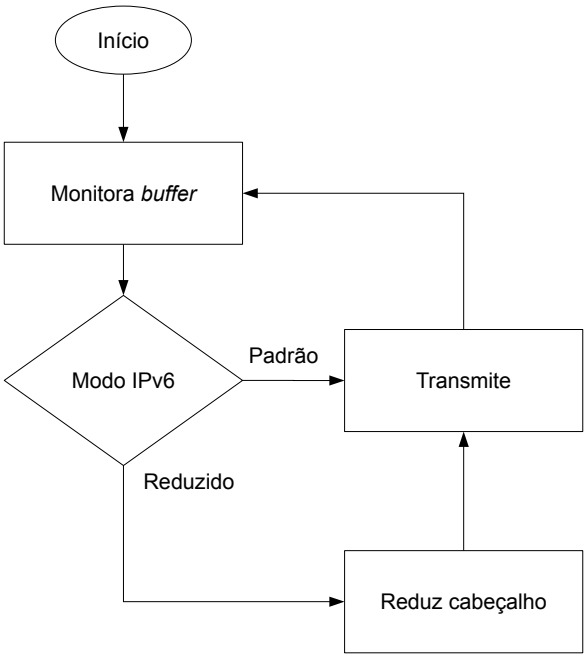

Figura 19: Fluxograma de transmissão

Após o desenvolvimento e testes preliminares da arquitetura proposta em software, deu-se início ao desenvolvimento da pilha IPv6 em hardware reconfigurável. Essa arquitetura é apresentada na próxima seção.

3.2 Pilha de comunicação IPv6 em hardware reconfigurável

A pilha de comunicação IPv6 proposta, utiliza como base a arquitetura apresentada por (HERRMANN, 2010), adicionando as funcionalidades para compatibilidade com o protocolo IPv6. A arquitetura foi desenvolvida em linguagem Verilog e prototipada na placa de desenvolvimento XUPV5-LX110T (XILINX, 2010c) dotada de um FPGA Xilinx XC5VLX110T-3ff1136 da família Virtex-5.

No desenvolvimento da pilha, foram implementadas as camadas de rede (IPv6) e enlace (MAC). A arquitetura proposta pode ser observada na Figura 20 que ilustra o *core* IPv6 contendo as camadas da pilha IP e suas interfaces. Como os blocos responsáveis pela recepção e transmissão dos dados trabalham de forma simultânea e independente, a implementação dessa pilha é considerada *full-duplex*.

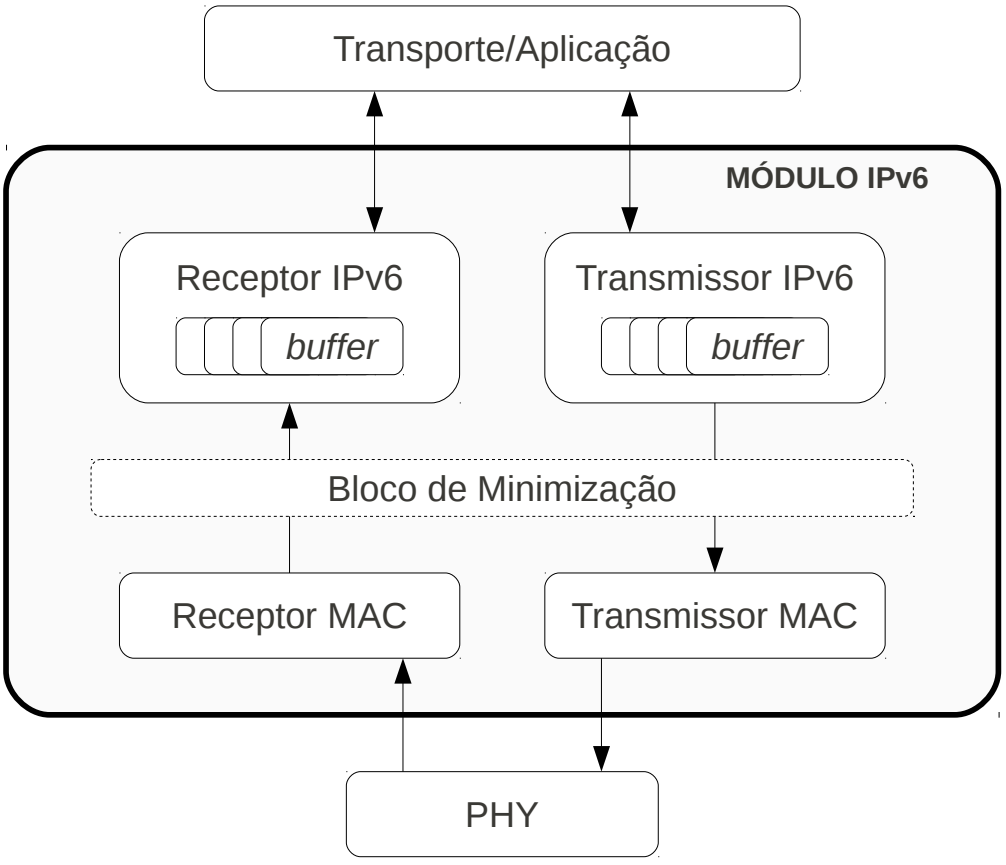

Figura 20: Estrutura da pilha IPv6 proposta

O bloco acima do módulo IPv6, que representa a camada de transporte, implementa a operação de *loopback* usada nos testes, ou seja, encaminha para os blocos de transmissão os dados recebidos através dos blocos de recepção. As próximas Seções detalham o fun-

cionamento de cada um dos blocos dessa pilha, assim como a comunicação do core IPv6 com a camada de transporte.

3.2.1 Receptor e Transmissor IPv6

Esses blocos são responsáveis por realizar a comunicação do *core* IPv6 com as camadas superiores e diretamentente com a camada de transporte, representando assim a camada de rede. O bloco Receptor IPv6 recebe o pacote do bloco Receptor MAC, realiza o desencapsulamento, ou seja, a verificação e remoção do cabeçalho Ethernet e então grava-o em um *buffer* de dados. Após recebê-lo completamente, este pacote é repassado para a camada de transporte. O bloco Transmissor IPv6 recebe o pacote da camada de transporte, armazena-o em um *buffer* de dados, realiza o encapsulamento com o cabeçalho do protocolo IPv6 e depois envia para o bloco Transmissor MAC.

Os *buffers* de dados são utilizados para armazenar os pacotes temporariamente, sendo que o *buffer* de recepção armazena o pacote até que a leitura seja efetuada pela camada superior. Da mesma forma, o *buffer* de transmissão armazena o pacote até que o envio seja realizado pelos blocos responsáveis pela transmissão. Esses *buffer* de dados são implementados por *Block Random Access Memory* (BRAMs), que são memórias internas do dispositivo FPGA (XILINX, 2009d).

A comunicação do *core* IPv6 com a camada de transporte, através dos blocos Receptor IPv6 e Transmissor IPv6, é baseada na troca de pacotes de configuração e dados. Conforme exibido na Tabela 2 existem três pacotes para realizar as configurações e um pacote de dados.

Tabela 2: Tipos de pacotes utilizados na comunicação do *core* IPv6 com a camada de transporte.

Pacote	**Tipo**	**Valor (Hexadecimal)**
IPv6	Dados	11
MAC Origem	Configuração	81
IPv6 Origem	Configuração	82
MAC *Gateway*	Configuração	88

Os pacotes de configuração são enviados da camada de transporte para o *core* IPv6 e são utilizados somente para configurar os endereços MAC de origem, IP de origem e MAC *gateway* do *core*. A estrutura desses quatro pacotes de configuração pode ser visualizada na Figura 3.

Os pacotes de dados são enviados da camada de transporte para o *core* IPv6 e também

Tabela 3: Estrutura dos pacotes de configuração do *core* IPv6.

Tipo (8 bits) hex	**Configuração**
81	MAC Origem (48 bits)
82	IPv6 Origem (128 bits)
88	MAC Gateway (48 bits)

do *core* IPv6 para a camada de transporte. Isto ocorre através do pacote de dados IPv6, o qual possui a estrutura ilustrada na Tabela 4. A Tabela 4 exibe a estrutura dos pacotes de dados que são enviados da camada de transporte para o *core* IPv6, contendo os seguintes campos:

- Tipo de pacote (1 byte): Esse campo é utilizado para diferenciar pacotes de dados e pacotes de configuração. Neste caso, para identificar pacotes de dados, este campo deve conter o valor '11' em hexadecimal para pacotes IPv6.
- Tamanho do pacote (2 bytes): Tamanho total do pacote que está sendo transmitido para o *core* IPv6. Através deste campo se torna mais simples para determinar o tamanho do pacote que a camada superior está transmitindo.
- Endereço IP de destino (16 bytes): Endereço IP de destino do pacote. Esse campo será utilizado no protocolo IP pela camada de rede. No caso do protocolo IPv6, o tamanho é de 16 bytes.
- Reservado (1 byte): Reservado para uso futuro.
- Dados (variável): Esse campo irá conter os dados da camada de transporte, sendo os dados que serão encapsulados no datagrama IP.

Pacotes de dados que são enviados do *core* IPv6 para a camada de transporte possuem a mesma estrutura exibida na Tabela 4, mas ao invés do campo endereço IP de destino esses pacotes possuem o campo endereço IP de origem.

Tabela 4: Estrutura dos pacotes de dados utilizados na comunicação da camada de transporte com o *core* IPv6.

Pacote de dados – Camada de Transporte para core IPv6					
Tamanho	**1 byte**	**2 bytes**	**16 bytes**	**1 byte**	**Variável**
Nome do Campo	**Tipo de Pacote**	**Tamanho do pacote**	**Endereço IP de destino**	**Reservado**	**Dados**

O envio de pacotes da camada de transporte para o *core* IPv6 é efetuado através dos seguintes pinos, que podem ser visualizados na Figura 21:

- *gtx_clk*: *Clock* de 125 MHz utilizado para o envio de pacotes.
- *wr_i*: Pino utilizado para iniciar o processo de escrita no *core* IPv6 (*wr_i* = '1').
- *dv_i*: Pino de *data valid*. Se este pino estiver com o valor '1' indicará a presença de um dado válido no barramento *data_i*.
- *data_i[31:0]*: Barramento de dados de 32 *bits*. O *core* IPv6 lê este campo somente quando o pino *dv_i* estiver em '1'.
- *wait_o*: Se o valor deste pino for '1', significa que o *core* está ocupado e não pode receber outro pacote no momento.

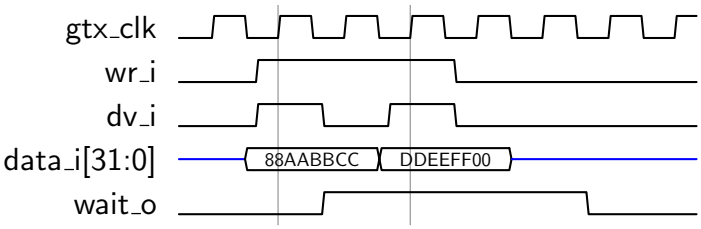

Figura 21: Diagrama de tempo - Pacotes da camada de transporte para o *core* IPv6 (HERRMANN, 2010)

O diagrama de tempo ilustrado na Figura 21 demonstra como a camada de transporte deve enviar pacotes para o *core* IPv6. Nessa Figura é possível observar o envio de um pacote de configuração MAC *gateway* para o *core*. Para realizar o envio de um pacote de dados o comportamento do diagrama permanece o mesmo, alterando somente a estrutura do pacote que será enviado.

O recebimento de pacotes do *core* IPv6 pela camada de transporte é efetuado através dos seguintes pinos, que também podem ser visualizados na Figura 22:

- *rx_clk*: *Clock* de 125 MHz utilizado para o recebimento de pacotes.
- *read_i*: Pino utilizado para iniciar o processo de leitura do *core* IPv6 (*read_i* = '1').
- *dv_i*: Pino de *data valid*. Se este pino estiver com o valor '1' indicará que o *core* deve enviar o próximo dado, através do barramento *data_o*, pertencente ao pacote.
- *data_o[32:0]*: Barramento de dados de 32 *bits*. O *core* IPv6 atualiza esse barramento após o recebimento de um *dv_i = '1'*.

- *int_o*: Se o valor deste pino for '1', significa que o *core* possui um novo pacote disponível e sua leitura pode ser realizada.

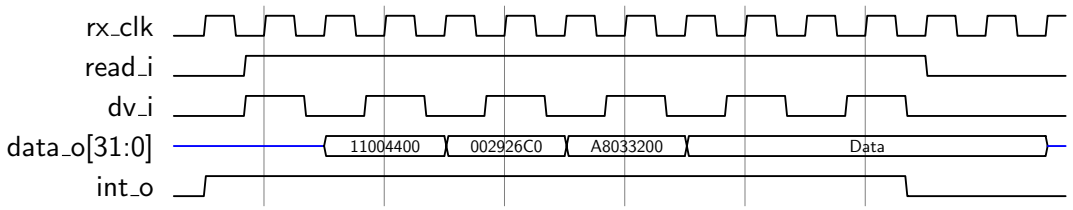

Figura 22: Diagrama de tempo - Pacotes de dados do *core* IPv6 para a camada de transporte (HERRMANN, 2010)

O diagrama de tempo ilustrado na Figura 22 demonstra como a camada de transporte procede para efetuar a leitura de um pacote de dados do *core* IPv6.

3.2.2 Bloco de Minimização

O Bloco de Minimização é inserido entre as camadas de rede e enlace do modelo OSI, e desempenha a função proposta de melhorar o desempenho de comunicação em redes locais. Quando ativado, esse bloco é responsável pela redução do cabeçalho IPv6 antes que o pacote seja encapsulado no quadro Ethernet. A redução do cabeçalho ocorre removendo-se os campos desnecessários para o tráfego de dados local.

Além da função de minimização do pacote durante uma transmissão, esse bloco também é utilizado no processo inverso, quando um pacote é recebido pelo *core*. Ao detectar o recebimento de um pacote IPv6 reduzido, antes de ser enviado à camada de rede, o mesmo sofre o processo de reconstrução para o seu formato original.

A ativação deste bloco ocorre com o recebimento de um pacote IPv6 reduzido. Assim os processos de reconstrução e minimização dos cabeçalhos IPv6 ocorre de maneira automática. Quando o tráfego de dados for de pacotes IPv6 padrão, o bloco é desativado, sem que ocorra dessa forma, qualquer interferência no fluxo de encapsulamento e desencapsulamento de dados.

3.2.3 Receptor e Transmissor MAC

O blocos Receptor MAC e Transmissor MAC representam a camada de enlace e gerenciam o recebimento e envio de quadros, respectivamente. O bloco Transmissor MAC envia o quadro para a interface do PHY. No começo é enviado o preâmbulo, seguido do *start frame*. Em seguida, o cabeçalho MAC e os dados do quadro são enviados de 8 em 8

bits. Cada byte é enviado para um bloco de geração de CRC, o qual calcula progressivamente o CRC do quadro. Este CRC é enviado para a PHY assim que o bloco Transmissor MAC terminar de enviar os dados do quadro.

O bloco Receptor MAC verifica se um novo quadro será recebido através da análise dos sinais de interface com o PHY. Se um novo quadro for detectado, um bloco de checagem é notificado e passa a calcular o CRC. Ao final do recebimento se CRC presente no quadro for diferente ao CRC calculado, o quadro será rejeitado. Somente quadros com o mesmo endereço MAC de destino que o core IPv6 e quadros com endereço MAC *broadcast* são aceitos e repassados para o bloco Receptor IP.

3.3 Testador

O desenvolvimento do dispositivo Testador, utiliza como base a arquitetura apresentada por (HERRMANN, 2010), adicionando as funcionalidades para compatibilidade com o protocolo IPv6. A arquitetura foi desenvolvida em linguagem Verilog e prototipada na placa de desenvolvimento ML402 (XILINX, 2006) dotado de um FPGA Xilinx XC4VSX35-10ff668 da família Virtex-4.

Este Testador é responsável pela realização dos testes da RFC2544, onde foram implementados os testes de Vazão e Latência, apresentados na Seção 2.4. A arquitetura oferece suporte tanto à versão IPv6 reduzida, quanto à versão IPv6 padrão.

A Figura 23 apresenta a arquitetura do Testador em hardware, contemplando também um bloco de software que é executado por um microprocessador. A seguir são apresentados detalhadamente cada um dos blocos que formam essa arquitetura.

3.3.1 Processamento RFC2544

Esse bloco é responsável pelo processamento em hardware dos testes da RFC2544 e pelo gerenciamento dos blocos Gerador e Receptor. Além disso, é realizada a comunicação com o bloco Software Embarcado através do barramento *Fast Simplex Link* (FSL) (XILINX, 2009c). Esse barramento, fornecido pela Xilinx, permite a comunicação entre dois blocos de hardware, ou seja, o bloco de Processamento e o microprocessador que executa o software.

Para a inicialização dos testes, são utilizados alguns registradores de configuração que são gerenciados pelo Software através da interface FSL. A seguir apresenta-se uma

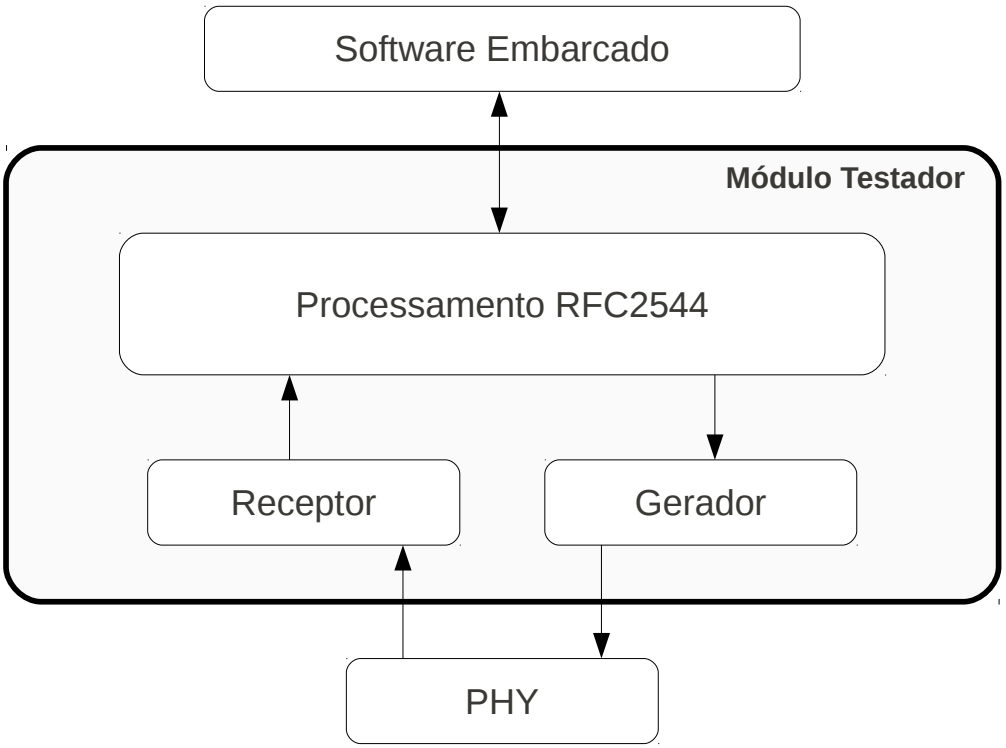

Figura 23: Estrutura do dispositivo testador proposto

descrição de cada um desses registradores:

- *frames_p_second[23:0]*: Quantidade de quadros que devem ser enviados por segundo.
- *frame_size_i[15:0]*: O tamanho de cada quadro, por exemplo, 64 bytes.
- *cycles_ifg_i[19:0]*: Quantidade de ciclos entre um quadro e outro, ou seja, o IFG.
- *repeat_test_i[7:0]*: Quantas vezes este teste deve ser executado, uma vez que os parâmetros acima são configurações para testes referentes a um segundo.
- *new_test_i*: Quando o valor deste pino for '1' indica que um novo teste deve ser iniciado.

Após o Software realizar a escrita dos registradores de configuração, o bloco de Processamento é responsável pela execução dos testes da RFC2544, de maneira independente, gerenciando os blocos Gerador e Receptor. Quando um teste é finalizado, os resultados referentes ao mesmo são disponibilizados através de outros registradores que fazem interface com o software, também através do barramento FSL. Abaixo apresenta-se detalhadamente esses registradores:

- *finish_o*: Quando valor deste pino for '1' indica que o teste está finalizado e os resultados serão enviados.
- *rcv_error_frame_o[31:0]*: Quantidade de quadros que não foram recebidos.
- *max_latency_o[31:0]*: Latência máxima encontrada durante o período de duração do teste, expressa em quantidade de ciclos de 8 ns.
- *min_latency_o[31:0]*: Latência mínima encontrada durante o período de duração do teste, expressa em quantidade de ciclos de 8 ns.
- *total_latency_o[63:0]*: Soma de todas as latências, expresso em quantidade de ciclos de 8 ns.
- *cnt_invalid_packets_o[31:0]*: Quantidade de quadros que não foram computados devido a erros de versão de protocolo, CRC ou endereços MAC e IP inválidos. Para que o teste seja válido o valor de retorno deste pino deverá ser igual a zero.

O controle dos testes é realizado através da contabilização de quadros enviados e recebidos, quadros válidos e inválidos, além dos cálculos de latência e tempo de execução de cada teste. Os valores utilizados no processamento são obtidos através da interface com os blocos Gerador e Receptor, descritos a seguir.

3.3.2 Gerador

O bloco Gerador é responsável pelo envio dos quadros de teste ao dispositivo sob avaliação. Nesse bloco, o quadro é gerado e encapsulado nas camadas de rede e de enlace. A Tabela 5 apresenta o formato do pacote do protocolo de testes desenvolvido. Após a geração desse pacote, ocorre o encapsulado em um datagrama IPv6 (padrão ou reduzido), e por fim em um quadro Ethernet antes que o mesmo possa ser enviado para a interface GMII do PHY.

Tabela 5: Formato do pacote do protocolo de testes

Protocolo de testes			
Tamanho	**1 byte**	**5 bytes**	**Variável**
Nome do Campo	**Versão**	**Contador de tempo**	**Dados**

O protocolo de testes é formado pelos os seguintes campos:

- Versão (1 byte): Versão do protocolo, deverá ser preenchido com o valor em hexadecimal '01'.

- Contador de tempo (5 bytes): Armazena o tempo no momento do envio do quadro. Quando esse quadro for novamente recebido será possível calcular a sua latência baseado no valor do contador de tempo atual implementado.

- Dados: Esse campo é utilizado para *padding*, ou seja, apenas para completar o tamanho do quadro. Por exemplo, no formato apresentado na Tabela 5, para um pacote IPv6 de 64 bytes, esse campo possui 0 bytes para *padding*. Já para um pacote IPv6 de 128 bytes, esse campo possui 64 bytes para *padding*.

3.3.3 Receptor

Ao receber um quadro da interface GMII do PHY, o bloco Receptor é responsável pelo desencapsulamento dos dados. Realiza-se a validação do CRC, das versões de protocolos e verificação de endereço MAC e IP. Caso algum erro seja detectado no quadro, este é computado como quadro inválido e o registrador *cnt_invalid_packets_o* do bloco de Processamento é incrementado. Caso a estrutura do pacote esteja correta, o bloco Receptor realiza o cálculo da latência, baseando-se no conteúdo do campo *contador de tempo* do protocolo de testes, e no contador de tempo atual do bloco de Processamento. Por fim, esse quadro é computado como válido e repassado para o bloco de Processamento, para que os valores de latência máxima, mínima e total sejam determinados.

3.3.4 Software Embarcado

Além dos blocos em hardware, o módulo Testador implementa um bloco de software, desenvolvido na linguagem C. A execução desse software se dá através do microprocessador *Microblaze* (XILINX, 2008), que permite a interação do usuário através de uma porta de comunicação serial RS232. A interface do microprocessador com o bloco de Processamento se dá através do barramento FSL, de 32 bits.

Esse bloco é responsável pela passagem de parâmetros de cada um dos testes e também pela inicialização dos testes através da interface FSL. Para tanto, o bloco implementa os algoritmos de vazão e latência da RFC2544.

A escrita das configurações dos quadros de teste no barramento FSL, assim como a leitura dos resultados é realizada por intermédio de funções de acesso ao barramento FSL.

Essas funções são definidas como *putfsl* para escritas e *getfsl* para leituras. Isso permite que através do software, seja realizada a escrita e a leitura dos registradores encontrados no bloco de Processamento.

No algoritmo do teste de vazão, faz-se uma busca pela maior taxa de transmissão suportada pelo dispositivo em teste. Inicialmente é realizado o envio de quadros à taxa de 100%. Se ocorrer a perda de algum quadro, a taxa é reduzida em 50% e o teste é realizado novamente. Repete-se esse procedimento até que seja encontrada uma taxa em que nenhum quadro é perdido. Quando isso ocorrer, incrementa-se em 50% o valor da taxa, e novamente o teste é realizado, fazendo assim, com que o valor da taxa suportada pelo dispositivo seja alcançada mais rapidamente.

Esse procedimento é realizado para todos os tamanhos de quadros sugeridos pela RFC2544, por 60 segundos para cada taxa de transmissão analisada. O tempo do teste, assim como os valores calculados para o IFG e a quantidade de quadros que serão enviados durante cada teste, são atribuídos respectivamente aos registradores *repeat_test_i*, *cycles_ifg_i* e *frames_p_second* do bloco de Processamento.

Repete-se 5 vezes o teste de vazão, e então é calculada a média dos resultados encontrados. Juntamente com esse teste, executa-se o teste de latência. Como o cálculo da latência em hardware considera apenas os quadros que foram recebidos, é possível calcular a latência para vazão de 100% mesmo em dispositivos que não suportam tal vazão. Para o cálculo da latência, utiliza-se o campo Contador de tempo do protocolo de testes apresentado na Figura 5. A média de latência retornada pelo teste é calculada através da divisão do valor contido no registrador *total_latency_o* pelo numero de quadros recebidos.

3.4 Metodologia

O desenvolvimento deste trabalho contou com o uso de uma série de ferramentas e equipamentos para simulação, síntese da arquitetura implementada e testes em laboratório. No desenvolvimento da lógica programável foram utilizadas as ferramentas ModelSim da Mentor Graphics e ISE Design Suite da Xilinx, para realizar as simulações funcionais e a síntese para FPGA. Juntamente com estas ferramentas, utilizaram-se *scripts* para automação dos processos de simulação e síntese lógica.

As simulações foram realizadas com a implementação de *testbenches*, responsáveis pela geração dos estímulos lógicos de entrada do módulo desenvolvido, além do monitoramento dos sinais de saída. Com isso, os dados transmitidos e recebidos pelo módulo são verifica-

dos automaticamente durante a simulação, sendo que quando um erro é detectado, este é imediatamente reportado pela ferramenta, permitindo assim uma breve análise e correção por parte do projetista.

O processo de síntese, que tem como entradas a descrição HDL do módulo IPv6 e o arquivo de *constraints*, é executado pela ferramenta ISE Design Suite. Nesta mesma ferramenta, são realizadas as etapas de mapeamento e roteamento para as estruturas internas do FPGA. Como saída desse conjunto de processos, é finalmente gerado um único arquivo binário, o qual é utilizado para configurar o FPGA com o módulo desenvolvido. O processo de gravação deste arquivo binário em FPGA é realizado pela ferramenta Xilinx Impact.

Após finalizada a etapa de implementação, deu-se início à etapa de testes, a qual foi viabilizada pelo cenário demonstrado na Figura 24. Este cenário foi utilizado como ambiente para a realização dos testes de implementação, bem como testes de análise de desempenho. Na Figura 24, podem ser observados os seguintes itens:

1. DUT: Placa de desenvolvimento XUPV5-LX110T dotada de um FPGA Xilinx XC5VLX110T-3ff1136 da família Virtex-5 utilizado para implementação das arquiteturas da pilha de comunicação IPv6.

2. Testador: Placa de desenvolvimento ML402 dotada de um FPGA Xilinx XC4VSX35-10ff668 da família Virtex-4 utilizado para implementação do Testador.

3. Cabo de rede: Cabo de rede que conecta a interface de rede *Gigabit Ethernet* do Testador à interface de rede *Gigabit Ethernet* do DUT.

4. Osciloscópio: Osciloscópio Agilent DSO6104A utilizado nos testes com FPGA para depuração de possíveis erros relacionados ao Testador e à pilha de comunicação IPv6. Possui dois barramentos digitais, cada um com 8 bits, através dos quais, foi possível realizar os testes funcionais das pilhas de comunicação implementadas e do Testador.

5. Barramentos digitais de teste: Barramentos digitais de teste do osciloscópio conectados às placas de desenvolvimento XUPV5-LX110T e ML402.

6. Computador: Computador com processador Intel E5700 3GHz Dual-Core com 4GB de memória RAM, utilizado nas etapas de desenvolvimento, simulação, verificação do projeto e testes. Também é utilizado na captura dos resultados oriundos do

Testador, através de um programa de monitoramento da porta RS232. Os dados capturados são armazenados em um arquivo texto para posterior verificação.

7. Cabo serial: Cabo RS232 conectado ao computador e utilizado na obtenção dos resultados reportados pelos algoritmos implementados no Testador.

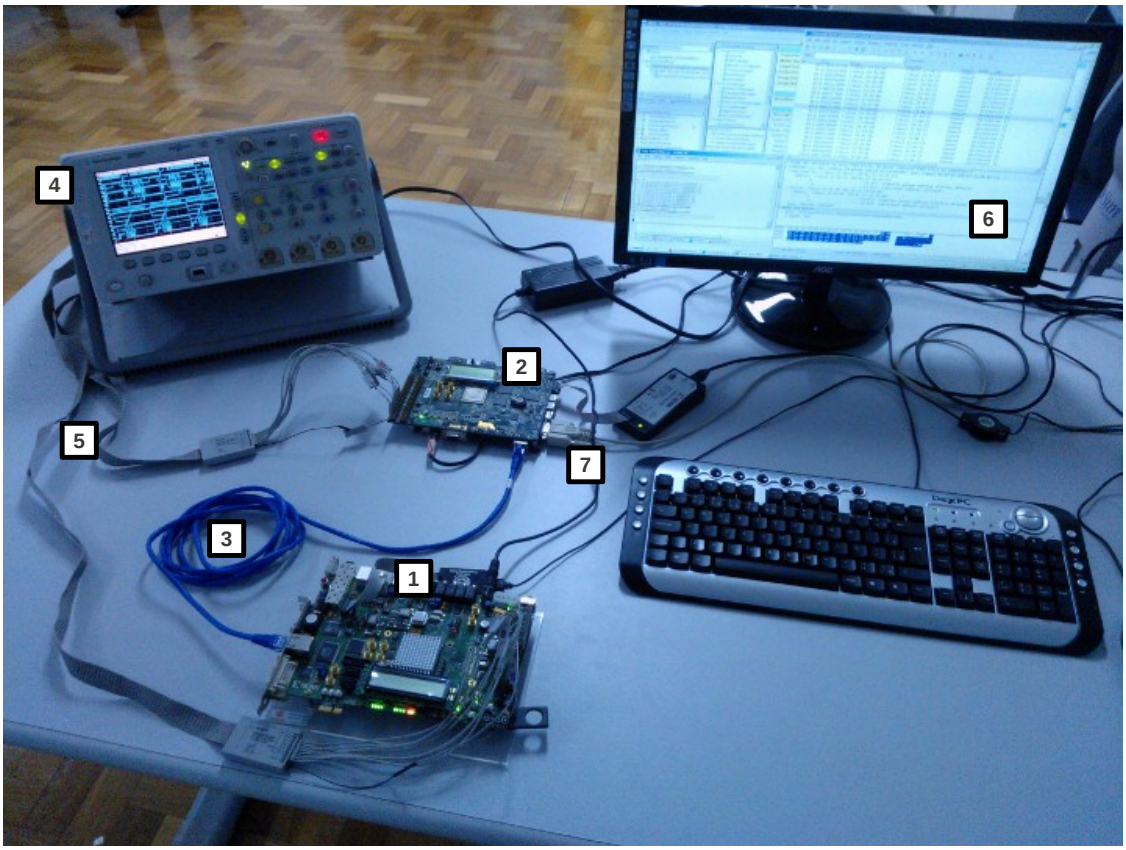

Figura 24: Cenário de testes para validação do módulo IPv6

Nota-se na imagem, que o ambiente é composto por um FPGA, que é onde está implementada a pilha IPv6 proposta neste trabalho, o qual conecta-se ao dispositivo testador e ao osciloscópio. A conexão do FPGA com o osciloscópio faz-se através do barramento digital de testes. Com o uso deste barramento, é possível verificar o correto funcionamento do módulo IPv6. Assim, se algo não funcionar ou não estiver de acordo com as especificações, pode-se identificar o erro e então corrigir na descrição HDL, sendo necessário refazer todo o fluxo de simulação e síntese novamente.

A conexão do FPGA que implementa a pilha IPv6 ao dispositivo testador é feita através das interfaces Gigabit Ethernet, utilizando-se um cabo de rede entre elas. Esta conexão permite que seja feita a comunicação entre os dois dispositivos, e com isso a

execução dos testes da RFC2544. Através da conexão do Computador com a interface Gigabit Ethernet no testador, pode-se realizar os mesmos testes no Computador ilustrado na Figura 24. Isto permite a obtenção dos resultados de desempenho da pilha IPv6 implementada a nível de software, para comparação dos resultados obtidos com o módulo desenvolvido.

4 RESULTADOS OBTIDOS

Este capítulo apresenta os resultados obtidos durante a realização dos testes da pilha de comunicação IPv6 proposta. Foram feitas comparações de desempenho entre as pilhas implementadas em hardware (FPGA) e software (Computador), avaliando a vazão, latência e taxa de perda de quadros entre a implementação da pilha padrão e a reduzida.

Para cada teste da RFC2544 realizado, foram feitas análises quanto à melhora de desempenho da pilha IPv6 reduzida quando comparada à pilha IPv6 padrão, além de apresentar uma comparação com os trabalhos relacionados, apresentados na Seção 2.5 do Capítulo 2. Nas próximas Seções são detalhadas as relações de desempenho obtidas em cada um dos testes.

4.1 Resultados de Vazão

A obtenção dos resultados de vazão deu-se através dos testes descritos na Seção 2.4, onde foi feita a análise da capacidade de cada um dos dispositivos (FPGA e Computador). Para os testes da pilha IPv6 em hardware, o Testador apresentado na Seção 3.3 foi conectado ao FPGA que implementa as pilhas IPv6 padrão e IPv6 reduzida, permitindo a inicialização dos testes. Os quadros de tamanho definidos na RFC2544, são inicialmente enviados ao DUT à taxa de 100% durante 60 segundos. Caso ocorra a perda de algum quadro, esta taxa é reduzida e o teste é realizado novamente, ocorrendo com isso também a diminuição da vazão resultante. A obtenção da vazão resultante, para cada tamanho de quadro se dá com a maior taxa suportada pelo DUT, quando não ocorre a perda de quadros.

A Figura 25 apresenta a diferença entre os resultados de vazão de dados entre a pilha IPv6 padrão, e a pilha IPv6 reduzida em hardware. Além disso, o gráfico apresenta os resultados de vazão de dados da pilha IPv4 (HERRMANN, 2010), sendo possível com isso realizar uma comparação mais ampla. As três pilhas obtiveram 100% como resultado do teste de vazão, para todos os tamanhos de quadros, porém no gáfico pode-se perceber a

diferença com relação à porcentagem de dados que podem ser transmitidos em cada um dos pacotes IP durante as transmissões.

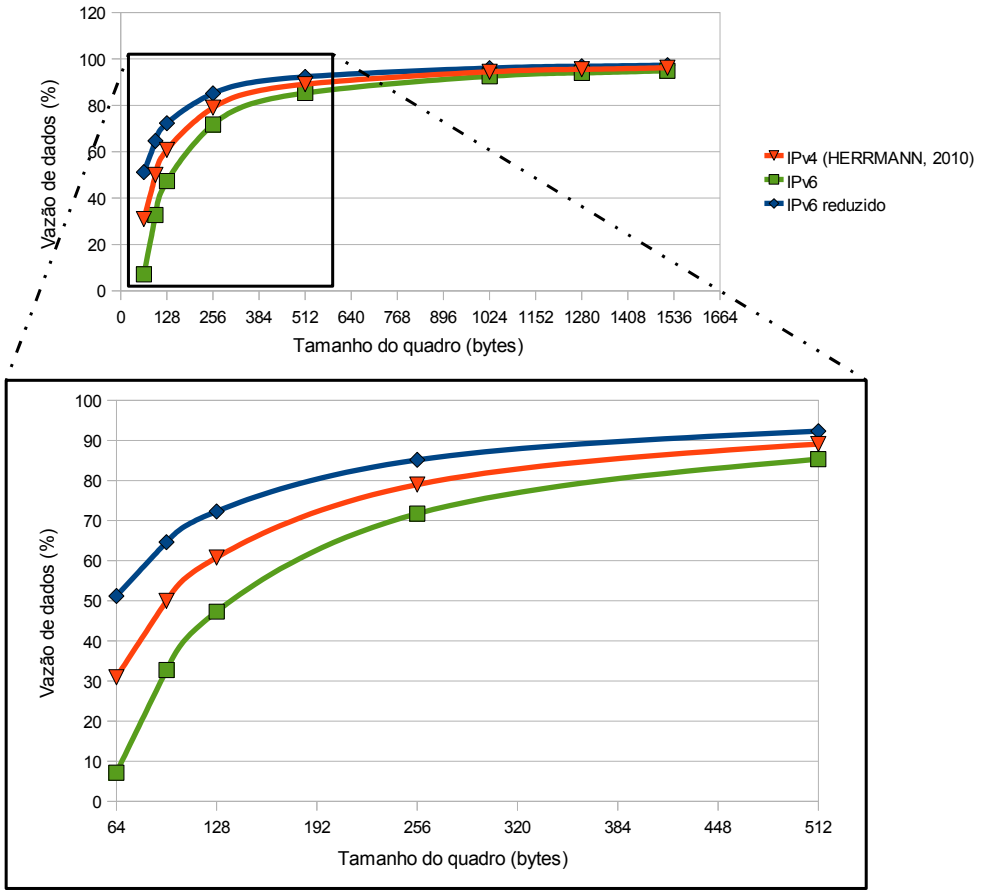

Figura 25: Resultados de vazão representando a quantidade de dados transmitida para cada tamanho de quadro

Isso ocorre devido às diferenças no tamanho dos cabeçalhos em cada uma das versões implementadas. Em um quadro de 64 bytes com o protocolo IPv6 padrão, é possível transmitir apenas 6 bytes de dados, enquanto que com o protocolo IPv6 reduzido, é possível transmitir 43 bytes com o mesmo tamanho de quadro. Mesmo na versão padrão do protocolo IPv4, pode-se observar que o *overhead* de transmissão é maior, quando comparado ao IPv6 reduzido. Esses valores podem ser observados na Tabela 6.

Nessa tabela apresenta-se a quantidade de dados que pode ser transmitida em cada um dos tamanhos de quadro, para cada uma das implementações. Além disso, é exibida a porcentagem de dados que é possível transmitir em cada quadro, considerando além do tamanho total do quadro, o preâmbulo e o tempo mínimo entre quadros. Na coluna final da tabela, apresenta-se o percentual de ganho na transmissão de dados da pilha IPv6 reduzida, quando comparada à pilha IPv6 padrão.

Aplicando-se o mesmo teste sobre as pilhas IPv6 implementadas em software, foram

Tabela 6: Relação da vazão de dados transmitidos entre as pilhas IPv6 em hardware.

Tamanho do quadro	IPv4 padrão		IPv6 padrão		IPv6 reduzido		% Ganho
	Dados	%	Dados	%	Dados	%	
64	26	31	6	7,1	43	51,2	616,7
96	58	50	38	32,8	75	64,7	97,4
128	90	60,8	70	47,3	107	72,3	52,9
256	218	79	198	71,7	235	85,1	18,7
512	474	89,1	454	85,3	491	92,3	8,1
1024	986	94,4	966	92,5	1003	96,1	3,8
1280	1242	95,5	1222	94,0	1259	96,8	3,0
1518	1480	96,2	1460	94,9	1497	97,3	2,5

observadas taxas de vazão menores que as taxas obtidas com as implementações em hardware. A Figura 26 apresenta os valores de vazão retornados no teste. Embora apresente resultados com valores menores que os apresentados nos testes da pilha em hardware, a proporção na melhora de desempenho entre as pilhas é a mesma. A Tabela 7 apresenta a porcentagem de vazão alcançada nos testes, e a relação da quantidade de dados que é transmitida em cada um dos quadros.

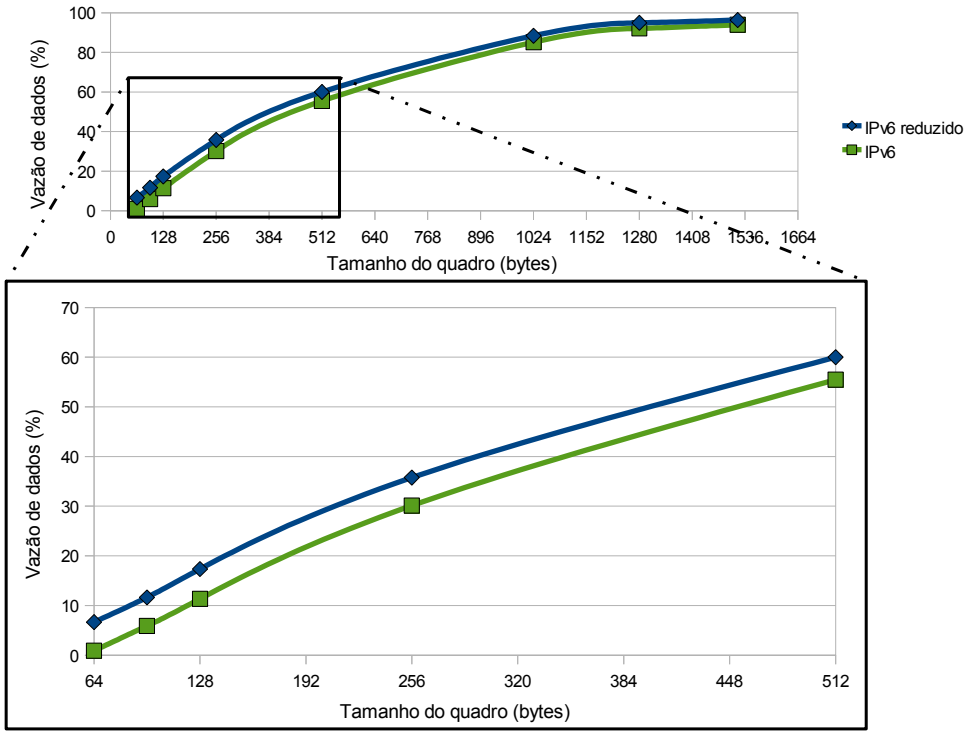

Figura 26: Resultados de vazão representando a quantidade de dados transmitida para cada tamanho de quadro

Como as taxas de vazão são menores, partindo de 13% para quadros de 64 bytes, a porcentagem de dados transmitida em cada um dos pacotes é reduzida, uma vez que o

Tabela 7: Relação da vazão de dados transmitidos entre as pilhas IPv6 em software.

Tamanho do quadro	**Vazão máxima (%)**	**IPv6 padrão**		**IPv6 reduzido**		**% Ganho**
		Dados	**%**	**Dados**	**%**	
64	13	6	0,9	43	6,7	616,7
96	18	38	5,9	75	11,6	97,4
128	24	70	11,4	107	17,4	52,9
256	42	198	30,1	235	35,8	18,7
512	65	454	55,5	491	60,0	8,1
1024	92	966	85,1	1003	88,4	3,8
1280	98	1222	92,1	1259	94,9	3,0
1518	99	1460	94,0	1497	96,4	2,5

tempo entre quadros é incrementado de acordo com a taxa de transmissão. Com isso o tempo para a transmissão de cada quadro é maior, fazendo com que a taxa efetiva da vazão de dados seja menor.

Ainda com taxas menores, a porcentagem no ganho de desempenho da pilha implementada em software permanece a mesma, quando comparada à pilha implementada em hardware. Na Figura 27, apresenta-se o percentual de ganho obtido com relação a vazão de dados, entre a pilha IPv6 padrão e a pilha IPv6 reduzida. Pode-se observar um ganho de mais de 600% para quadros de 64 bytes, seguido por 97% para quadros de 96 bytes, e um ganho mínimo de 8% para quadros de 512 bytes. As proporções de ganho para os diferentes tamanhos de pacotes podem ser observadas na Figura 27.

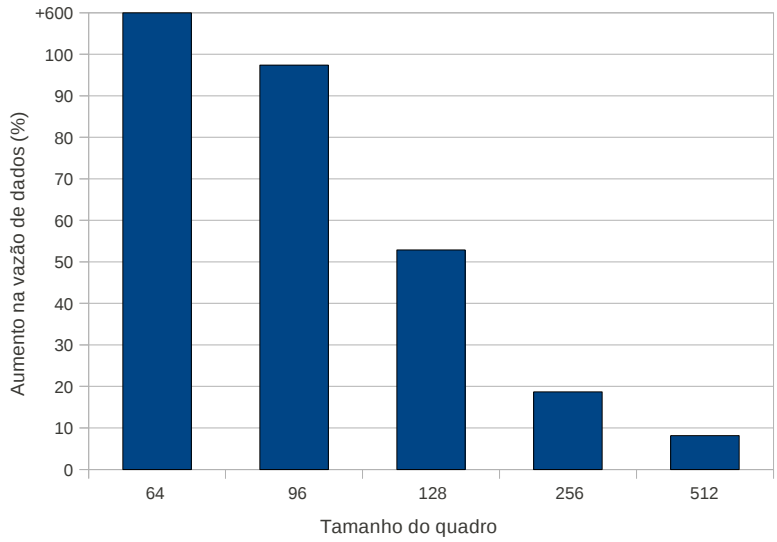

Figura 27: Percentual de ganho no desempenho de vazão dos dados

Através dos resultados de vazão obtidos e da observação do gráfico apresentado na

Figura 27, pode-se concluir que a minimização do cabeçalho do protocolo IPv6 proposta, apresenta um aumento de desempenho quando comparado à pilha IPv6 padrão, pois a vazão de dados efetiva é maior. Esse ganho de desempenho é mais significativo quando são transmitidos quadros pequenos, de 64 a 512 bytes. Para quadros maiores, com mais de 512 bytes, os ganhos não representam valores muito significativos.

4.2 Resultados de Latência

A realização do teste de latência conforme a RFC2544, foi repetida 5 vezes durante 60 segundos cada. Como resultado do teste, calcula-se a média da latência retornada para cada um dos tamanhos de quadro previstos. Como os resultados de vazão obtidos nos testes não atingiram 100% nas implementações em software, os resultados de latência neste caso somente levam em consideração os quadros recebidos.

Inicialmente nos testes das pilhas implementadas em software, não houve diferença nos resultados de latência, tanto para a pilha IPv6 padrão quanto para a pilha IPv6 reduzida. Porém, na Figura 28 é possível observar valores maiores de latência ao comparar as implementações IPv6 com uma implementação de pilha IPv4 apresentada em (HERRMANN, 2010), também em software.

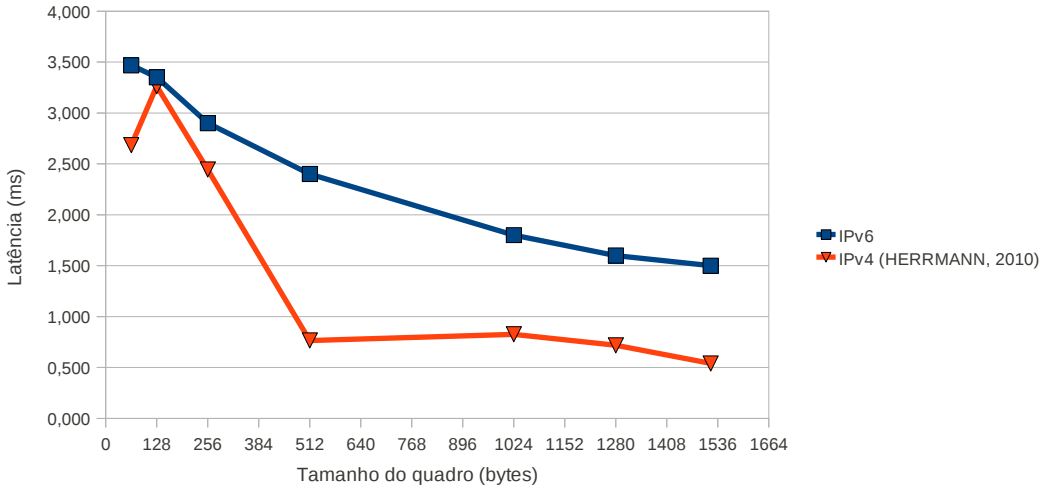

Figura 28: Latência da pilha IPv6 comparada à uma implementação da pilha IPv4 em software

Devido às diferenças de implementação das pilhas, assim como diferenças de plataforma de execução entre as pilhas testadas, observa-se uma pequena diferença na comparação dos resultados de latência. A pilha IPv6 apresentou, para quadros de 64 bytes, 3,47 ms de latência, sendo que a pilha IPv4 apresentou 2,68 ms para o mesmo tamanho

de quadro. Já para quadros de 1518 bytes obteu-se como resultado, valores de 1,5 ms e 0,5 ms com as pilhas IPv6 e IPv4, respectivamente.

Já nos testes realizados sobre as pilhas IPv6 implementadas em hardware, foram obtidos valores de latência menores ao comparar com uma pilha IPv4 padrão (HERRMANN, 2010) submetida aos mesmos tipos de testes. Na Figura 29, são representados os valores de latência obtidos nesses testes. Como as implementações das pilhas IPv6 padrão e reduzida utilizam a mesma arquitetura, a latência de transmissão dos pacotes de mesmo tamanho tende a apresentar os mesmos valores em ambas pilhas. Com isso, optou-se pela realização do teste de latência sobre a pilha IPv6 reduzida, utilizando-se tamanhos reduzidos de pacotes, sendo que a quantidade efetiva de dados a ser transmitida nos pacotes seja a mesma que na pilha IPv6 padrão. Por exemplo, um pacote IPv6 de 128 bytes contém um campo de dados de 70 bytes. Já um pacote IPv6 reduzido, com o mesmo campo de dados de 70 bytes, tem um tamanho total de 91 bytes. Assim, os pacotes utilizados nos testes para a pilha IPv6 reduzida, são de 64, 91, 219, 475, 987, 1243 e 1481 bytes, representando nos resultados de latência, tamanhos de 64, 128, 256, 512, 1024, 1280 e 1518 bytes, respectivamente. A Tabela 8 apresenta os valores de latência e os percentuais de ganho obtidos nos testes de latência realizados, entre as pilhas IPv6, para cada um dos tamanhos de quadro.

Tabela 8: Relação da latência entre as pilhas em hardware.

Tamanho do quadro	**Latência (us)**			**Ganho (%)**
	IPv4 padrão	**IPv6 padrão**	**IPv6 reduzido**	
64	6,9	4,608	4,608	0,0
96	8,75	5,12	4,608	10,0
128	10,6	5,632	5,04	10,5
256	14,8	7,68	7,088	7,7
512	17,2	11,776	11,184	5,0
1024	25,3	19,968	19,376	3,0
1280	31,4	24,064	23,472	2,5
1518	41,9	27,872	27,28	2,1

A diferença entre as latências das pilhas IPv6 indicam que é possível transmitir a mesma quantidade de dados entre dois dispositivos, sendo que na pilha IPv6 reduzida, isso é feito num menor espaço de tempo. Além disso, como os pacotes são menores, a utilização da banda disponível também é menor. O gráfico mostra ainda que a pilha IPv6 apresenta valores menores de latência, quando comparado à pilha IPv4. Para quadros de 64 bytes, a latência média foi de 4,6 us, na pilha IPv6, enquanto que na pilha IPv4 o valor foi de 6,9 us. Conforme aumenta-se o tamanho do quadro, também aumenta a latência,

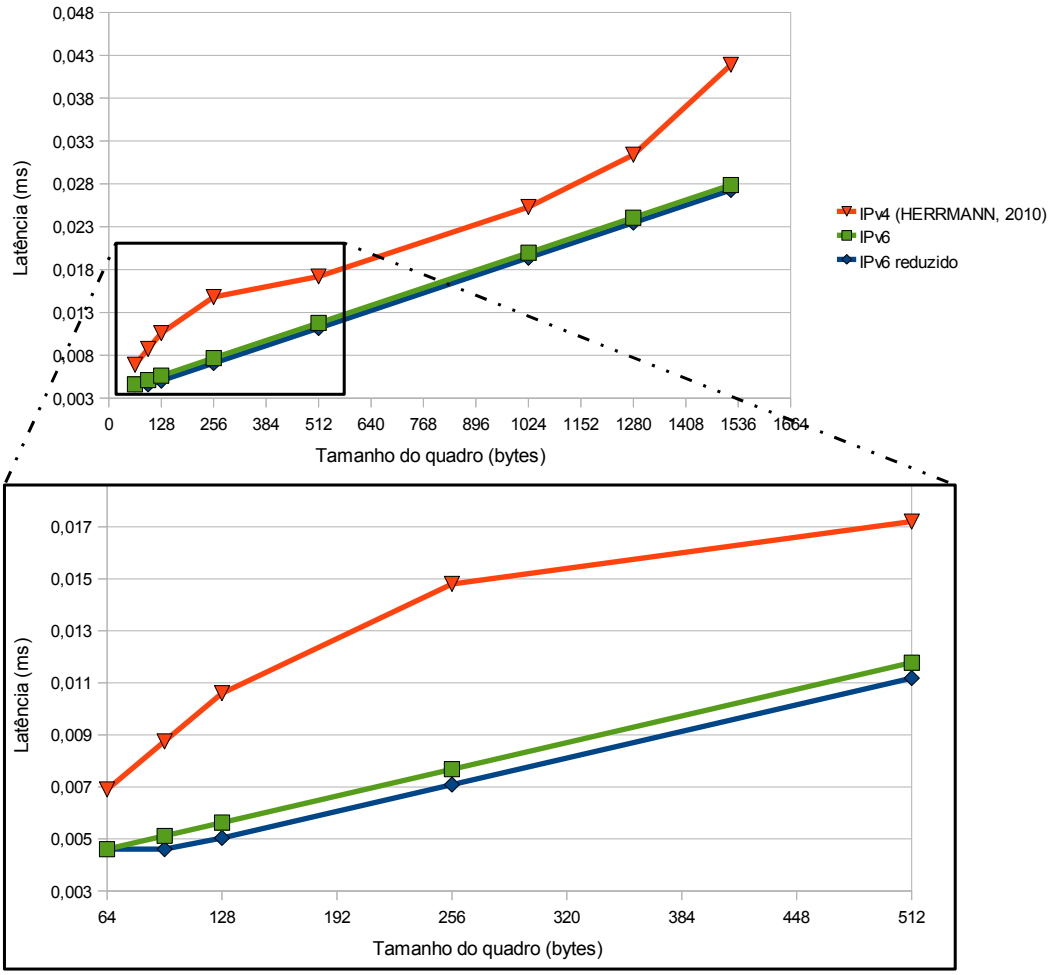

Figura 29: Latência das pilhas IPv6 padrão e reduzida, comparadas à uma implementação da pilha IPv4 em hardware

pois um tempo maior é necessário para a transmissão do quadro. Quadros de 1518 bytes apresentaram 27,9 us de latência na pilha IPv6, e 41,9 us na pilha IPv4. Nota-se ainda no gráfico, que na pilha IPv6 a latência aumenta linearmente conforme o tamanho do quadro transmitido, enquanto que na pilha IPv4 ocorrem algumas variações. Este comportamento linear é resultado do atraso de processamento do quadro, que despende um número fixo de ciclos de relógio independentemente do tamanho do quadro que é recebido e transmitido.

Os resultados de latência obtidos com os testes demonstraram, na prática que as implementações em hardware obtiveram os melhores resultados, sendo que quanto menor é o quadro, menor é o atraso na transmissão. Isso ocorre pelo fato de que o processamento do cabeçalho de cada um dos quadros é realizado ao mesmo tempo em que o quadro está sendo recebido. No caso da pilha implementada em software o efeito é contrário, uma vez que quanto menor é o quadro, maior é o atraso na transmissão do mesmo. Isto ocorre devido à sobrecarga de processamento dos cabeçalhos dos quadros, sendo que há

um numero muito mais elevado de quadros para processar durante testes com quadros pequenos, quando comparado a testes com quadros de maior tamanho.

5 CONCLUSÃO

Este trabalho apresentou como proposta um novo método e o desenvolvimento de uma nova arquitetura, visando a melhoria de desempenho na comunicação de dados. Essa arquitetura é responsável pela a redução do cabeçalho IPv6 durante a transmissão de pacotes em uma rede local, otimizando assim uma grande parte do tráfego da rede. Foram desenvolvidas pilhas de comunicação em software e hardware, com o objetivo de medir o desempenho de cada uma, e posteriormente realizar uma análise comparativa entre elas.

Os testes práticos das pilhas de comunicação foram realizados com o auxílio de um dispositivo testador, o qual foi desenvolvido com o propósito de executar um conjunto de testes definidos pela RFC2544. Para realizar a medida de desempenho da arquitetura proposta no trabalho, foram aplicados os testes de vazão e latência.

Com as análises comparativas apresentadas no Capítulo 4, é possível concluir que a arquitetura de minimização de pacotes gerou bons resultados. As principais melhorias de desempenho ocorreram para pacotes pequenos, como previsto. Foram obtidos ganhos significativos para quadros de até 512 bytes. Através dos testes realizados, é possível finalmente concluir que a arquitetura proposta permite uma melhora de desempenho na rede, com o aumento da vazão de dados, além da diminuição da latência e da utilização de banda na transmissão de pacotes entre dois dispositivos.

Os resultados obtidos com os testes de vazão demonstraram que o método proposto neste trabalho permite um aumento de desempenho de até 616,7% na transmissão de dados, quando comparado com uma implementação padrão da pilha IPv6. Este ganho foi obtido com a minimização do cabeçalho de comunicação IPv6 durante as transmissões entre dois dispositivos em uma mesma rede local. Com o método utilizado, foi possível reduzir o tamanho do cabeçalho de 40 bytes para apenas 3 bytes, obtendo assim a redução de 37 bytes para cada um dos pacotes transmitidos. Com isso foi possível transmitir uma quantidade maior de dados nos mesmos tamanhos de quadros que foram utilizados nos testes entre as implementações padrão e reduzida da pilha IPv6.

Além disso, observou-se com os testes entre as diferentes plataformas, de software e hardware, que a proporção do ganho obtido na vazão de dados é independente da taxa que define a capacidade de vazão do dispositivo. Isso indica que mesmo que um equipamento de comunicação não suporte altas taxas de transmissão, a proporção do ganho na vazão de dados se mantém a mesma que em dispositivos que suportem taxas de até 100%.

Nos testes de latência, os resultados iniciais obtidos com as implementações das pilhas em software não foram representativos, sendo que não ocorreu a redução nem o aumento do tempo total de transmissão dos quadros durante os testes. Porém foi possível observar um melhor desempenho com a pilha IPv6 reduzida em hardware, ao comparar pacotes de tamanhos reduzidos, transmitindo a mesma quantidade de dados da pilha IPv6 padrão. Com isso é possível concluir que utilizando a arquitetura proposta, o tempo de transmissão dos pacotes entre os dispositivos pode ser reduzido em até 10,5%.

5.1 Trabalhos Futuros

Além da proposta e dos resultados obtidos neste trabalho, novos esforços podem ser realizados para o aprimoramento da arquitetura desenvolvida. A seguir são apresentadas algumas ideias para complementar e dar sequência a trabalho apresentado.

- Implementação no *kernel* do Linux: otimização e implementação do módulo de minimização do cabeçalho IPv6 diretamente entre as camadas de protocolos de comunicação do *kernel* do Linux, e a realização de uma nova avaliação de desempenho. Essa proposta visa a obtenção de resultados mais satisfatórios nos testes da arquitetura em software, uma vez que a implementação utilizando *sockets* de comunicação não apresentou ganhos significativos de latência.

- Tabela ARP: *Address Resolution Protocol* (ARP) é um protocolo utilizado para encontrar um endereço MAC a partir do endereço IP. Geralmente cada máquina mantém uma tabela de resolução em cache para reduzir a latência e carga na rede, porém as arquiteturas apresentadas neste trabalho não implementam a tabela ARP. Essa pode ser uma nova abordagem utilizada nos módulos de minimização para a reconstrução dos cabeçalhos, além de apresentar um avanço no sentido de padronizar o método desenvolvido.

- Detecção de compatibilidade: desenvolvimento de uma técnica para detectar os dispositivos compatíveis com o protocolo IPv6 minimizado na rede, possibilitando até

mesmo uma integração com a tabela ARP. Fundamental para que a arquitetura seja utilizada nos mais diversos dispositivos, é a necessidade de detectar quais equipamentos da rede local suportam a transmissão dos pacotes com o cabeçalho IPv6 reduzido.

- Implementação da camada de transporte em hardware: as pilhas de comunicação foram implementadas neste trabalho visando a obtenção de resultados de desempenho na comunicação de dados na rede. Futuras implementações podem contar com camadas superiores, como por exemplo TCP ou UDP, facilitando o uso da pilha por alguma aplicação específica e aumentando com isso o desempenho de processamento na comunicação de dados.

REFERÊNCIAS

BRADNER, S.; MCQUAID, J. *RFC 2544 - Benchmarking Methodology for Network Interconnect Devices*. MarÃ§o 1999. Disponível em: <http://www.rfc-editor.org/rfc/rfc2544.txt>.

COMER, D. E. *Internetworking with TCP/IP*. 5th edition. ed. [S.l.]: Prentice Hall, 2005. Paperback. ISBN 0131876716.

CRAWFORD, M. *Transmission of IPv6 Packets over Ethernet Networks*. IETF, dez. 1998. RFC 2464 (Proposed Standard). (Request for Comments, 2464). Updated by RFC 6085. Disponível em: <http://www.ietf.org/rfc/rfc2464.txt>.

DAY, J.; ZIMMERMANN, H. The osi reference model. *Proceedings of the IEEE*, 1983.

DEERING, S.; HINDEN, R. *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*. December 1998. Disponível em: <http://tools.ietf.org/html/rfc2460>.

DOSTÁLEK, L.; KABELOVÁ, A. *Understanding Tcp/ip: A Clear And Comprehensive Guide*. [S.l.]: Packt Publishing, 2006. Paperback. ISBN 190481171X.

HERRMANN, F. L. *Implementacao de Arquiteturas de Pilha UDP/IP em Hardware Reconfiguravel Baseado no Desempenho de Vazao Latencia e Taxa de Perda de Quadros*. Dissertação (Mestrado), 2010.

HINDEN, R.; DEERING, S. *IP Version 6 Addressing Architecture*. IETF, fev. 2006. RFC 4291 (Draft Standard). (Request for Comments, 4291). Updated by RFCs 5952, 6052. Disponível em: <http://www.ietf.org/rfc/rfc4291.txt>.

IANA. *Port Numbers*. 2009. Disponível em: <http://www.iana.org/assignments/port-numbers>.

IEEE. *IEEE 802.3 CSMA/CD (Ethernet) Access Method*. 2008. PadrÃ£o Ethernet, 2008. Disponível em: <http://standards.ieee.org/about/get/802/802.3.html>. Acesso em: dezembro 2010.

JOHN, W.; TAFVELIN, S. Analysis of internet backbone traffic and header anomalies observed. In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007. (IMC '07), p. 111–116. ISBN 978-1-59593-908-1. Disponível em: <http://doi.acm.org/10.1145/1298306.1298321>.

KEEGAN, B.; DAVIS, M. An experimental analysis of the call capacity of ieee 802.11b wireless local area networks for voip telephony. In: *Irish Signals and Systems Conference, 2006. IET*. [S.l.: s.n.], 2006. p. 283–287.

KERRISK, M. *The Linux man-pages project*. [S.l.], September 2013. Release 3.54. Disponível em: <http://www.kernel.org/doc/man-pages/>.

LAM, P.-K.; LIEW, S. UDP-liter: an improved UDP protocol for real-time multimedia applications over wireless links. In: *1st International Symposium on Wireless Communication Systems, 2004.* [S.l.: s.n.], 2004. p. 314–318.

LIM, J.; STERN, H. Ipv6 header compression algorithm supporting mobility in wireless networks. In: *Southeastcon 2000. Proceedings of the IEEE.* [S.l.: s.n.], 2000. p. 535–540.

LÖFGREN, A.; LODESTEN, L.; SJOHOLM, S.; HANSSON, H. An analysis of fpga-based UDP/IP stack parallelism for embedded ethernet connectivity. In: *Proceedings of Norchip Conference, Oulu, Finland, 23rd.* [S.l.: s.n.], 2005. p. 94–97.

MURUGESAN, R.; RAMADASS, S.; BUDIARTO, R. Improving the performance of ipv6 packet transmission over lan. In: *Industrial Electronics Applications, 2009. ISIEA 2009. IEEE Symposium on.* [S.l.: s.n.], 2009. v. 1, p. 182–187.

POSTEL, J. *RFC 768 - User Datagram Protocol.* Agosto 1980. Disponível em: <http://www.rfc-editor.org/rfc/rfc768.txt>.

POSTEL, J. *RFC 791 - Internet Protocol.* Setembro 1981. Disponível em: <http://www.rfc-editor.org/rfc/rfc791.txt>.

REALFAST. *RealFast HW/SW System Design - UDP/IP Cores High Speed Ethernet Communication Cores.* 2010. Disponível em: <http://www.realfast.se/rfipp/products/udpip/udpip.shtml>.

TANENBAUM, A. S. *Computer Networks.* 4th edition. ed. [S.l.]: Prentice Hall, 2002. Paperback. ISBN 0130384887.

WEIDONG, L. *Designing TCP/IP Functions In FPGAs.* Dissertação (Mestrado), 2003.

WESTPHAL, C.; KOODLI, R. Stateless ip header compression. In: *Communications, 2005. ICC 2005. 2005 IEEE International Conference on.* [S.l.: s.n.], 2005. v. 5, p. 3236–3241 Vol. 5.

XILINX. *ML401/ML402/ML403 Evaluation Platform - User Guide.* [S.l.], Maio 2006.

XILINX. *MicroBlaze Processor Reference Guide.* 2008. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf>.

XILINX. Fast simplex link (FSL) bus (v2.11b). *Product Specification*, 2009c. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf>.

XILINX. *Block RAM (BRAM) Block (v1.00a) Data Sheet.* 2009d. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf>.

XILINX. *Xilinx, Inc. FPGA and CPLD Solutions.* 2010a. Disponível em: <http://www.xilinx.com>.

XILINX. *Xilinx University Program XUPV5-LX110T Development System.* 2010c. Disponível em: <http://www.xilinx.com/univ/xupv5-lx110t.htm>.

XUE-ZHOU, Y.; ZHI-LING, T.; SI-MIN, L. A novel mobile ipv6 header compression algorithm of wireless sensor network. In: *Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. CyberC '09. International Conference on.* [S.l.: s.n.], 2009. p. 143–147.

ZHANG, D.; ZHENG, C.; ZHANG, H.; YU, H. Identification and analysis of skype peer-to-peer traffic. In: *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*. [S.l.: s.n.], 2010. p. 200–206.