UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

André Weizmann

PROJETO DE UM EMULADOR DE COMUTADOR, PARA UMA REDE LOCAL ETHERNET, UTILIZANDO REDES DE PETRI COLORIDAS.

Dissertação submetida ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Santa Catarina, como parte dos requisitos, para obtenção do título de Mestre em Ciências da Computação.

Orientador: Prof. João Bosco Mangueira Sobral, Dr.

Florianópolis, outubro/2002.

PROJETO DE UM EMULADOR DE COMUTADOR, PARA UMA REDE LOCAL ETHERNET, UTILIZANDO REDES DE PETRI COLORIDAS

André Weizmann

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Fernando Álvaro Ostumi Gauthier, Dr.
Coordenador do Curso

Banca Examinadora

João Bosco Mangueira Sobral, Dr.
Orientador

Jorge Muniz Barreto, Dr.
Membro

Bernardo Gonçalves Riso, Dr.
Membro

Em memória, a todos que morreram por uma causa que acreditavam.

" A imaginação é o nosso limite "

Autor Desconhecido

AGRADECIMENTOS

A Deus;

A minha esposa Leila, pelo apoio e amor dedicados;

Aos meus familiares, que sempre estão ao meu lado e confiam em meu potencial;

À Universidade Federal de Santa Catarina - UFSC;

Ao amigo Msc. Alexandre Broleze , pelo apoio e incentivo;

Ao orientador Prof. Dr. João Bosco Mangueira Sobral pelo acompanhamento pontual, competente e crítico;

Ao Prof. Dr. Paulo da Silva Borges, pelo apoio e exemplo crítico;

Ao Sr. José Ivo Fernandes de Oliveira, Romeo de Oliveira e André da Silva Abade, pela atenção e conhecimentos compartilhados;

Aos demais professores e colaboradores do curso de Pós-Graduação em Ciências da Computação da UFSC;

A empresa West Line Integradora de Sistemas na pessoa do Sr. João Ribeiro, pelo apoio na busca de material técnico e consultoria;

A todos que de uma maneira ou outra contribuíram para a realização deste trabalho.

SUMÁRIO

LISTA DE TABELAS E FIGURAS

LISTA DE ABREVIATURAS

ONG – Organização Não Governamental.

TI – Tecnologia da Informação.

LAN – Local Área Network.

WAN – World Área Network.

TCP – Transfer Control Protocol.

IP – Internet Protocol.

IEEE – Institute of Electrical and Electronics Engineers

CSMA – Carrier Sense Multiple Acces

CD – Collision Detect

MBPS – Mega Bits por Segundo

GBPS – Giga Bits por Segundo

DARPA – Department of Defense Advanced Research Projects Agency.

NCT – Network Control Protocol

IETF – Internet Engineering Task Force

IRTF – Internet Research Task Force

MAC - Media Access Control

DNS – Domain Named System

SAT – Source Address Table

FSC - Frame Check Sequence

CRC - Cyclic Redundancy Check

NIC - Network Interface Card

OUI - Organization Unique Identifier

ARP – Address Resolution Protocol

RARP – Return Address Resolution Protocol

DHCP – Domain Host Control Protocol

MIT – Militar Institute Tecnology

HP – Hewlett Packward

IBM – Internacional Bussines Machine

NAT – Network Address Translation

RESUMO

A administração dos dados que trafegam em redes de computadores, dá-se através da análise dos quadros (*frames*) de dados. Encontram-se nestes quadros os endereçamentos de máquina-destinatária e máquina-remetente, o protocolo utilizado, o tamanho total de bytes de dados no quadro e a seqüência de verificação à falhas CRC.

Várias implementações em *hardware* e *software* provêem estas informações estatísticas, dentre o *hardware* pode-se citar o comutador (*switch*). Equipamento este que efetua a leitura do datagrama do quadro de dados, extraindo informações quanto ao número de quadros danificados, qual a máquina que mais remete quadros, qual a mâquina que mais recebe quadros, os protocolos que trafegam dados ou o protocolo principapal em uso, entre outros.

Através do uso das Redes de Petri Coloridas, implementou-se um algoritmo que emula o funcionamento de um comutador em um computador, fornecendo dados estatísticos sobre o tráfego dos dados na rede, ao qual o emulador estiver conectado, outrossim, possibilita a sua utilização em substituição aos *hubs*. Como sistema operacional tem-se o Linux, que por ser de domínio público, não tem-se que pagar pela licença de utilização do mesmo e como linguagem de programação o C, por ser uma linguagem robusta, portável e pela sua capacidade de trabalhar em baixo nível. Este algoritmo foi implementado, simulado e validado utilizando o programa DaNAMiCS, dentro do conceito de Redes de Petri Coloridas.

ABSTRACT

The management of data that flow through a network ocurrs with a frame analisys. These frames have addresses of target and source computers, the protocol used, the total size of data bytes and a sequence of CRC failures verification .

Many implementations in hardware and in software have these estatistical informations, especifically the hardware we could say about switches. A switch reads a datagram from the frame, extracting informations as number of crashed frames, wich computer sends more frames, wich computer receives more frames, wich protocols are used or the more used protocol.

Using Colored Petri Net, an algorithm was implemented whose function is to simulate how a switch works in a computer. The simulation receives statical data about the its network. This job makes possible the substitution of hubs. The operational system used is the Linux. This choice was made because Linux is open source and free and also because the C language would be used at this environment too. The implementation and the validation of this algorithm was done in DaNaMics software and with this software was used the thecnic of Colored Petri Net.

1.INTRODUÇÃO

Este trabalho versa sobre a construção de um algoritmo para uma ferramenta que emula o funcionamento de um comutador(*switch)* em um microcomputador que interconecta redes locais Ethernet, no sentido de encaminhar quadros(*frames)* de dados, que circulam entre redes, de forma otimizada e gerenciável. Tal ferramenta proporciona a obtenção de dados estatísticos do tráfego na rede, através da leitura dos endereços Ethernet constantes dos quadros, interpretando os endereços de destinatário, remetente e o protocolo de alto nível que está manipulando os dados contidos no quadro.

1.1MOTIVAÇÃO

Devido à necessidade do controle do fluxo de quadros que trafegam por redes dos mais variados tipos e tecnologias, originou-se a necessidade de administrar e otimizar os investimentos em soluções, para a interconexão de redes. Neste cenário pode-se destacar, a ascensão das redes que utilizam a tecnologia Ethernet, devido a sua popularização e baixo custo de implantação comparadas a outras tecnologias, tais como ATM ou Token Ring.

Ao mesmo tempo, deve-se verificar, se uma rede supre o fluxo de dados que a ela compete. Surge então, a necessidade de gerenciamento da rede. Segundo Paul LaSalle [LAS02] o melhor servidor do mundo não fará nada bem feito se a rede estiver tão ocupada que ninguém possa acessá-la, ou se a rede é tão propensa a erros que os sistemas têm de retransmitir constantemente. Ante a isto, quanto mais ferramentas de monitoração implantadas em uma rede, melhor será o controle de quadros que trafegam pela mesma.

Neste cenário de ferramentas de controle e monitoração podemos citar uma comparação entre o *hub* e o comutador (*switch*), Como o *hub* compartilha o meio de comunicação e funciona apenas como um repetidor de sinais, torna-se notável que a adição de mais computadores ao *hub* irá decrementar a performance da rede como um todo. O *hub* não tem "inteligência" suficiente para diferenciar endereços de origem e destino dentro de um quadro e todos os computadores que estão conectados ao *hub* recebem este sinal.

Enquanto, um *hub* de 24 portas e um *link* de 100 Mbps, levando-se em consideração que todas as portas estão sendo usadas devido ao compartilhamento da velocidade por todas as portas, teríamos uma velocidade aproximada por porta de 4,5 Mbps, com um comutador, cada porta, trabalhará a uma velocidade de 100 Mbps, de forma dedicada, não gerando *broadcast* para todas as portas. A máquina-remetente comunica-se com uma máquina-destino, diretamente. Uma outra vantagem de um comutador é que pode-se efetuar uma comunicação múltipla entre os computadores conectados ao mesmo, ao contrário do *hub*, onde apenas uma máquina, por vez, pode transmitir.

A segurança da rede também é um ponto importante que se ganha com o uso de comutadores. Em uma rede baseada em *hubs* todos os dados estão disponíveis para todas as estações conectadas a um segmento. Qualquer pessoa com acesso ao *hub* e com um software de monitoramento, pode capturar dados que estejam trafegando no meio de comunicação. Isto não é possível em uma rede baseada em comutador porque uma máquina-remetente envia dados para uma máquina-destino de forma direcionada e não disponível para os outros computadores conectados ao mesmo comutador.

Um outro fator motivante deste trabalho, diz respeito ao custo de um comutador de boa qualidade e com suporte a gerenciamento. As marcas mais conhecidas, tais como: 3COM, Cisco, Hewlet Packard, Enterasys, entre outras) custam em torno de U$ 2.500 (Dois mil e quinhentos dólares) aproximadamente, encarecendo e tornando pouco acessível à aquisição deste equipamento, por pequenas e médias empresas, organizações não governamentais, universidades e escolas.

Este projeto visa implementar um emulador de comutador que será instalado em um microcomputador. A implantação do emulador pode custar aproximadamente

1.000 (mil reais), dado que o microcomputador utilizado para este fim, pode ser até mesmo, um microcomputador 486 DX4-100, máquina esta que estando fora de linha, pode ser adquirida por um valor de baixo custo. A configuração a ser utilizada deverá ser considerada conforme o tamanho da empresa e a quantidade de computadores que a mesma possui, ou mesmo o número de serviços que poderão ser habilitados nesta máquina. O *hardware* utilizado pode ser igualmente compartilhado com outras aplicações como: roteador de quadros, *firewall*, servidor-*proxy*, servidor DNS, entre outros.

Como um exemplo, este projeto, pode ser implantado em uma estrutura de rede com um número aproximado de 120 estações de trabalho, interligadas através de 5 *hubs*, de 24 portas, cascateados pelas portas 23 e 24, e trabalhando a uma velocidade de conexão de 10 Mbps. Assim, pode-se melhorar a velocidade de conexão para 100 Mbps e diminuir o número de colisões e perdas de quadros. Ao mesmo tempo pode-se melhorar o nível de segurança da rede. Supondo-se uma reestruturação da rede, na qual os *hubs* sejam trocados por comutadores de 10/100 Mbps, isto acarretaria uma alta demanda financeira, e neste caso, ao invés de serem instalados comutadores, pode-se considerar a instalação do emulador proposto neste trabalho, interligando os *hubs* e eliminando os cascateamento desses.

1.2 OBJETIVOS

Os objetivos do presente trabalho são: escrever o algoritmo de funcionamento do emulador, conhecer os conceitos e os aspectos teóricos das Redes de Petri Coloridas, modelar e validar o funcionamento do algoritmo usando uma ferramenta de Redes de Petri apropriada, e implementar o algoritmo utilizando a linguagem de programação C.

1.3 ESTRUTURA DO TRABALHO

O presente trabalho está organizado como segue:

Capítulo 2 – Conceituação sobre Redes Ethernet: Um histórico sobre ethernet, suas variações no passar dos tempos, o endereçamento nos quadros e o *hardware* que a compõe.

Capítulo 3 – Estrutura de funcionamento do Emulador, utilizando linguagem informal, explicando as principais fases do funcionamento do algoritmo do emulador.

Capítulo 4 – Redes de Petri: um breve histórico sobre Redes de Petri, Redes de Petri Coloridas, e porque efetuar a modelagem do emulador usando essas redes.

Capítulo 5 – Modelagem e Validação do algoritmo proposto, utilizando o programa DaNAMiCS.

Capítulo 6 – Descrição da implementação do algoritmo, mostrando como foi realizada a implementação em linguagem C.

Capítulo 7 – Finalmente, apresenta as conclusões e sugestões para novos trabalhos.

CAPÍTULO 2. CONCEITUAÇÃO SOBRE REDES ETHERNET

Este capítulo aborda sobre a base para a comunicação entre estações, em uma rede Ethernet. Segundo Charles E. Spurgeon [SPU00], as especificações Ethernet determinam a estrutura de um quadro e quando uma estação pode envia-lo.

O emulador foi projetado para interpretar os quadros que trafegam nas redes, descobrindo, a máquina-remetente, que enviou do mesmo e qual máquina-destinatária o quadro deve ir, qual o protocolo de alto nível que está sendo utilizado para a comunicação dos dados.

Cabe ressaltar que o modelo Ethernet é um dos mais utilizados no mundo. Devido a essa popularização os custos de implantação e aquisição de componentes básicos tornaram-se mais acessíveis, porém, os componentes que auxiliam na administração e correção de falhas, não acompanharam esta tendência. O padrão publicado pela IEEE para redes Ethernet é o 802.3.

O padrão Ethernet é formado por quatro elementos básicos:

O Quadro (*frame*), é um conjunto padronizado de bits usados para transportar dados pelo sistema. O emulador tem como função, interpretar estes bits diferenciando-os e identificando os endereços da máquina-destino, máquina-remetente e o protocolo que está administrando os dados no quadro.

O protocolo Controle de Acesso à Mídia *(Media Acces Control*-MAC), consiste em um conjunto de regras embutidas em cada interface Ethernet para permitir que vários computadores acessem o canal Ethernet compartilhado de um modo ordenado.

Os componentes de sinalização consistem, em dispositivos eletrônicos padronizados, que enviam e recebem sinais por um canal Ethernet. Como exemplo podemos citar as placa de rede.

O meio físico consiste, nos cabos e outros *hardwares* utilizados para transportar os sinais Ethernet digitais entre os computadores ligados à rede.

Antes de comentários sobre os ítens acima, segue abaixo um breve histórico sobre o surgimento das redes Ethernet.

2.1 BREVE HISTÓRICO

Em 22 de maio de 1973, Bob Metcalfe (então na Xerox Palo Alto Research Center, PARC, na Califórnia) escreveu um memorando descrevendo o sistema de rede Ethernet que inventou para a interconexão de estações de trabalho de computador avançadas, possibilitando o envio de dados entre si e para impressoras a laser de alta velocidade. Provavelmente, a invensão mais conhecida na Xerox PARC, foi a primeira estação de trabalho de computador pessoal com *interfaces* gráficas do usuário e dispositivo apontador de *mouse*, chamado Xerox Alto. As invenções do PARC também incluíram as primeiras impressoras para computadores pessoais e, com a criação do Ethernet, a primeira tecnologia de LAN de alta velocidade para ligar tudo.

Esse foi um ambiente de computação marcante para a época, pois o início dos anos 70 foi uma época em que a computação era dominada por computadores *mainframes* grandes e muito caros. Poucos podiam pagar e dar suporte aos *mainframes*, e poucas pessoas sabiam como utilizá-los. As invensões na Xerox PARC ajudaram a realizar uma mudança revolucionária no mundo da computação.

Esse novo modelo de computação fez surgir um mundo totalmente novo de tecnologia de comunicações. Atualmente, o compartilhamento de informações normalmente é feito por meio de uma rede Ethernet; do menor dos escritórios à maior das empresas, de uma única sala de aula, até o maior campus universitário, Ethernet certamente é a tecnologia de rede preferida.

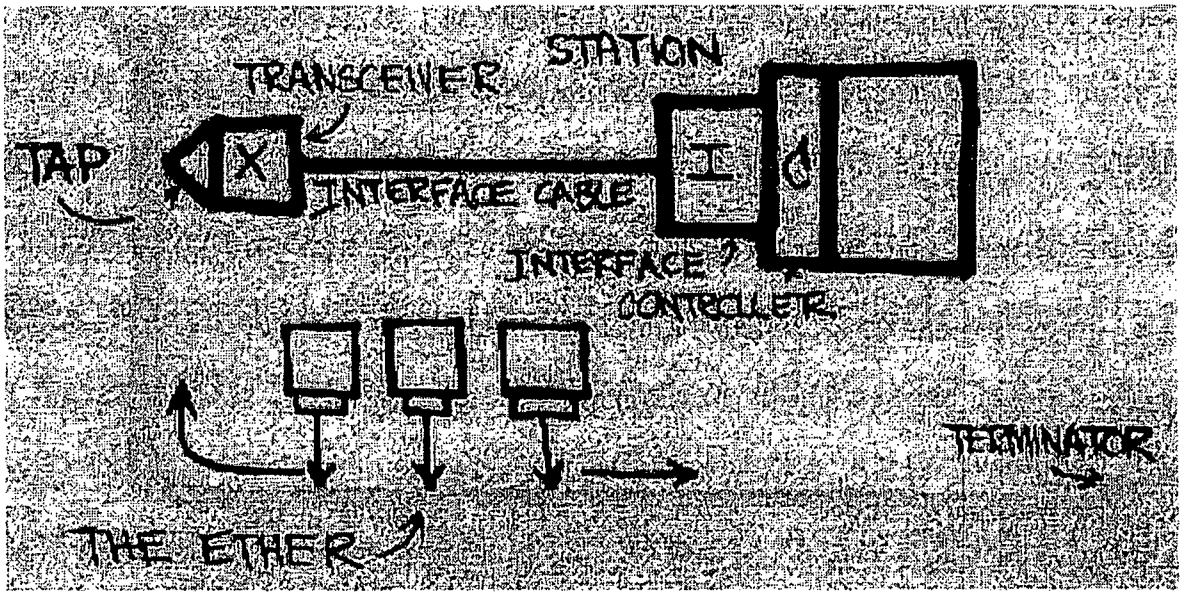
2.2 A INVENSÃO DO ETHERNET

Metcalfe desenvolveu um novo sistema (baseado no Aloha), que incluía um mecanismo para detectar quando ocorria uma colisão (*CD - collision detection*). O

sistema também incluía o conceito em que as estações ouviam as atividades (*carrier sense*) antes de transmitir e aceitavam o acesso a um canal compartilhado por várias estações (*multiple acces*). Devido a esta nova técnica surgiu o protocolo CSMA/CD – *carrier sense multiple acces with collision detect*, ao qual Metcalfe desenvolveu um algoritmo mais sofisticado, permitiu que o sistema Ethernet funcionasse com até 100 por cento de carga.

A primeira rede experimental foi denominada Alto Aloha Network, pois, era baseada no conceito da rede Aloha e foi implementada para interligar o Xerox Alto. Em 1973, Metcalfe mudou o nome para "Ethernet", para deixar claro que o sistema poderia funcionar em qualquer computador e baseado em um antigo conceito de "espaço luminífero" (*ether*) que propagava ondas eletromagnéticas pelo espaço. Assim nasceu o Ethernet.

Figura 1 - Desenho do sistema Ethernet original, feito por Metcalfe em 1976.



2.3 EVOLUÇÃO DO PADRÃO ETHERNET

O padrão Ethernet original de 10 Mbps foi publicado pela primeira vez em 1980, pelo consórcio de empresas DEC-Intel-Xerox, desta união surgiu o padrão DIX Ethernet. Continha às especificações para a operação do Ethernet e também para um sistema de única mídia, baseada em cabo coaxial grosso.

O projeto seguinte desenvolvido em mídia Ethernet, foi à variedade Ethernet coaxial fino, inspirada pela tecnologia comercializada inicialmente pela 3Com Corporation.

Após vieram diversas variedades de mídia, incluindo o par de fios trançados e a fibra óptica. Em seguida foi desenvolvido o sistema Fast Ethernet de 100 Mbps e o Gigabit Ethernet de 1 Gbps.

2.4 O QUADRO (*FRAME*) ETHERNET

No sistema Ethernet, os dispositivos se comunicam um com o outro utilizando quadros Ethernet. O controle de acesso à mídia Ethernet é baseado no *carrier sense with multiple acces e collision detect*, que fez surgir o acrônimo CSMA/CD. Um quadro Ethernet consiste em um conjunto de *bits* que é organizado em campos. Esses campos incluem vários campos de endereço, um campo de dados e um campo de controle de erros que controla a integridade dos dados encapsulados no quadro. O campo de dados pode variar de 46 a 1500 *bytes*.

Tabela 01 - Tamanho em *bytes* conforme cada campo.

64 bits	48 bits	48 bits	16 bits	46 a 1500 bytes	32 bits
Preâmbulo	Endereço de destino	Endereço de Origem	Tipo	Dados	Seqüência de verificação de quadro

Os vários campos do quadro Ethernet IEEE802.3 são:

Preâmbulo – independente do tipo de quadro utilizado, os sinais em todas as redes Ethernet são codificados de uma mesma maneira, por meio de um método conhecido como "Codificação de Manchester" – cada período de *bit* é dividido em dois intervalos iguais. Um *bit* 1 binário é enviado quando a voltagem é definida como alta durante o primeiro intervalo e como baixa no segundo. No 0 binário, acontece exatamente o contrário; primeiro baixa e depois alta. Esse esquema garante que cada período de *bits* tenha uma transição na parte intermediária, tornando fácil para o receptor sincronizar-se com o transmissor. A desvantagem da codificação Manchester é que são requeridos duas vezes mais largura de banda, que a codificação binária direta,

pois os pulsos são a metade da largura. A eficácia desse método está baseada em dois requisitos: os relógios internos de cada computador em uma rede Ethernet estejam sincronizados e o comprimento de cada tempo de *bit* seja acordado durante a própria transmissão. Esses dois requisitos são facilitados pelo preâmbulo, uma seqüência de vários 1s e 0s que precedem o quadro Ethernet real.

O preâmbulo consiste em 8 *bytes* de 1s e 0s alternados, terminando em "11". Quando um computador na rede Ethernet transmite, outros computadores utilizam o preâmbulo para bloquear (ou sincronizar) o relógio interno do remetente. Para estabelecer a sincronização, são necessários alguns segundos. Uma vez que a sincronização entre o remetente e o receptor seja estabelecida, o receptor espera pela seqüência "11" que sinaliza o quadro Ethernet seguinte. Uma vez que o preâmbulo é apenas utilizado para estabelecer a sincronização entre as estações, nenhum *bit* de preâmbulo entra no *buffer* de memória das placas de rede.

Endereço de destino – Cada *interface* de rede Ethernet, recebe um endereço exclusivo de 48 *bits*, chamado endereço físico ou de *hardware* da *interface*. O campo de endereço de destino contém o endereço Ethernet de 48 *bits* que corresponde ao endereço da *interface* na estação que irá receber os dados. O campo de endereço de destino, ao invés disso, pode conter um endereço *multicast* de 48 *bits*, os quais, uma ou mais *interfaces* na rede foram preparadas para receber, ou o endereço de *broadcast* padrão, que é composto por uma seqüência de números 1.

Endereço de origem – Esse é o endereço físico da *interface* de rede, da máquina-remetente que enviou o quadro. O endereço da máquina-destinatária é fornecido exclusivamente para uso dos protocolos de alto nível.

Tipo ou Tamanho – Esse campo de 16 *bits*, contém um identificador que refere-se ao tipo dos dados do protocolo de alto nível, sendo transportados no campo de dados do quadro Ethernet.

Existem identificadores para cada tipo de protocolo de alto nível, abaixo seguem esses identificadores e o protocolo a que esse código representa.

- 0x0060 *Frame de Loopback* Ethernet
- 0x0200 *Frame Echo* Ethernet

- 0x0400 *Frame* Xerox
- 0x0800 *Frame* IP
- 0x0805 CCITT X.25
- 0x0806 *Address Resolution packet* ARP
- 0x08FF G8BPQ AX.25 Ethernet *Packet*
- 0x6000 DEC *Assigned* protocolo
- 0x6001 DEC DNA *Dump/Load*
- 0x6002 DEC DNA *Remote Console*
- 0x6003 DEC DNA *Routing*
- 0x6004 DEC LAT
- 0x6005 DEC *Diagnostics*
- 0x6006 DEC *Customer use*
- 0x6007 DEC *Systems Comms Arch*
- 0x8035 *Reverse Addr Res packet* RARP
- 0x809B Appletalk DDP
- 0x80F3 Appletalk AARP
- 0x8137 IPX over DIX
- 0x86DD IPv6 *over bluebook*
- 0x0001 *Dummy type for* 802.3 *frames*
- 0x0002 *Dummy protocol id for* AX.25
- 0x0003 *Every packet (be careful!!!)*
- 0x0004 802.2 *frames*
- 0x0005 *Internal only*
- 0x0006 DEC DDCMP: *Internal only*

- 0x0007 *Dummy type for WAN PPP frames*
- 0x0008 *Dummy type for PPP MP frames*
- 0x0009 *Localtalk pseudo type*
- 0x0010 *Dummy type for Atalk over* PPP
- 0x0011 802.2 *frames*
- 0x0015 Mobitex
- 0x0016 *Card specific control frames*
- 0x0017 Linux/IR

Campo de dados – Esse campo deverá conter um mínimo de 46 *bytes* de dados, e pode chegar até a um máximo de 1500 *bytes* de dados. O *software* de protocolo de rede deverá fornecer pelo menos 46 *bytes* de dados.

Campo Seqüência de verificação de quadro ou FSC(*Frame Check Sequence*) – Que também pode ser chamado de CRC (*cyclic redundancy check*). Esse campo de 32 *bits* contém um valor que é usado para verificar a integridade dos diversos *bits* nos campos do quadro (sem contar o preâmbulo). Esse valor é calculado por meio do CRC, que é um polinômio calculado com o conteúdo dos campos de destino, origem, tipo (ou tamanho) e dados. Quando o quadro é gerado pela máquina-remetente, o valor do CRC é calculado simultaneamente. Os 32 *bits* do valor de CRC, que são o resultado desse cálculo, são colocados no campo FCS quando o quadro é enviado.

O CRC é calculado novamente pela interface na máquina-destinatária, à medida que o quadro é lido. O resultado desse segundo cálculo é comparado com o valor enviado no campo FCS pela máquina-destinatária. Se os dois valores forem idênticos, então haverá um alto nível de confiança de que nenhum erro ocorreu durante a transmissão pelo canal Ethernet

2.5 O PROTOCOLO CONTROLE DE ACESSO À MÍDIA (*MEDIA ACCESS CONTROL-MAC*)

É um conjunto de regras usadas, para atribuir acesso ao canal compartilhado entre um conjunto de estações conectadas a esse canal. O modo de operação do protocolo de acesso é muito simples: cada computador equipado, com um dispositivo de rede Ethernet, opera de modo independente de todas as outras estações da rede; não existe um controlador central. Todas as estações ligadas a uma rede Ethernet, operando em modo *half-duplex*, são conectadas a um canal de sinalização compartilhado, também conhecido como barramento de sinal.

O Ethernet utiliza um mecanismo de entrega por *broadcast*, em que cada quadro transmitido é ouvido por cada estação. Embora isso possa parecer ineficaz, a vantagem é que a inserção da inteligência de combinação de endereço na *interface* de cada estação, permite que o meio físico seja mantido o mais simples possível. Em uma LAN Ethernet, tudo o que a sinalização física e o sistema de mídia têm a fazer é verificar se os *bits* estão sendo transmitidos com precisão para cada estação; a *interface* Ethernet na estação faz o restante do trabalho.

Os sinais Ethernet são transmitidos da *interface* e enviados pelo canal de sinal compartilhado, para cada estação ligada. Para enviar dados, uma estação primeiro verifica o barramento, se o mesmo estiver ocioso, a estação transmite seus dados na forma de um quadro Ethernet.

À medida que cada quadro Ethernet é enviado pelo barramento ou meio compartilhado, todas as *interfaces* Ethernet conectadas ao canal, lêem os *bits* do sinal e verificam o segundo campo do quadro, que contém o endereço da máquina-destinatária. As *interfaces* comparam o endereço da máquina-destinatária do quadro, com seu próprio endereço *unicast* de 48 *bits*, ou com um endereço *multicast* ao qual foram preparadas para reconhecer. A *interface* Ethernet cujo endereço corresponde ao endereço de destino, continuará lendo o quadro inteiro, passando-o para o *software* de rede em execução nesse computador. Todas as outras *interfaces* de rede deixarão de ler o quadro, quando descobrirem que o endereço de destino, não corresponde ao seu próprio endereço *unicast*, ou a um endereço *multicast* ativado.

2.6 *HARDWARE* UTILIZADO EM UMA REDE ETHERNET

2.6.1 Placa de Rede

Também chamada de n*etwork interface card* (NIC) ou placa Ethernet, é uma placa interna que pode ser adicionada a um computador, para prover uma *interface* de *hardware* entre a mídia de transmissão (cabo, *wireless*) e o método de transporte (Ethernet, Fast Ethernet, Token Ring, ATM, entre outras), usado pelos microcomputadores para aquela mídia de transmissão. Algumas impressoras contêm sua própria placa de rede, permitindo que sejam conectadas diretamente ao cabeamento da rede. Alguns computadores, especialmente servidores, vêm com as placas de rede embutidas (*on-board*). Nesse caso são denominadas *interfaces* de rede, pois não são placas.

A IEEE oferece aos fabricantes um código OUI (*Organization Unique Identifier*), de 24 *bits*. O OUI é um identificador exclusivo, atribuído a cada organização que frabrica interfaces de rede. Isso permite que um fabricante de equipamento Ethernet, ofereça um endereço exclusivo para cada *interface* que eles produzirem. A promoção de endereços exclusivos durante a manufatura, evita o problema de que duas ou mais *interfaces* Ethernet em uma rede, possuam o mesmo endereço. Isso também elimina a necessidade de administrar e gerenciar os endereços Ethernet .

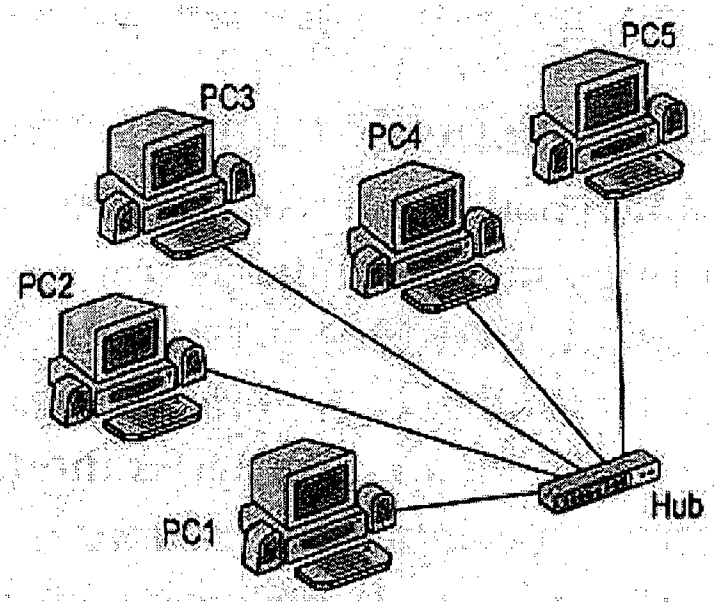
Um fabricante de *interfaces* Ethernet, cria um endereço Ethernet exclusivo de 48 *bits*, para cada *interface* atribuíndo para os 24 *bits* iniciais de endereço, seu OUI. Os 24 *bits* seguintes são atribuídos pelo fabricante, tendo o cuidado de garantir que cada endereço seja exclusivo. O endereço de 48 *bits* resultante, normalmente é chamado de endereço de *hardware* (ou endereço físico). Ele também é chamado de endereço MAC.

2.6.2 *HUB*

É um dispositivo que reside no núcleo de uma rede, ou sistema de cabeamento com topologia estrela. O *hub* está conectado aos computadores e a outros *hubs*. Dentro de uma LAN, um *hub* provê um local centralizado, para conexões e gerenciamento de rede, premitindo aos gerentes de rede configurar e controlar LANs de

grande porte e geograficamente dispersas de um único ponto na rede. Pode-se conectar *hubs* em uma topologia hierárquica estrela, para formar uma rede maior. Porém, um *hub* ao receber um quadro envia-o para todas as máquinas, criando assim, um fluxo enorme de dados, trafegando na rede sem destino e diminuindo a velocidade da rede. Esse problema está presente em todos os *hubs*, não importando seu frabricante ou modelo. Outra desvantagem do *hub* é a divisão do *link* total da rede, pelo número de portas, portanto, uma rede com velocidade de 10 Mbps e um *hub* de 8 portas, a velocidade real de transmissão por porta dos quadros será de 1,25 Mbps.

Figura 2 – Modelos de Estrutura de Rede Utilizando um *Hub*



Hub fixo (stand-alone) , possui uma configuração de portas permanente, não sendo possível adicionar, remover ou trocar as portas, pois, são pré-configuradas de fábrica. É o modelo mais barato de *hub*, porém, o mais restrito.

Hub empilhável (stackable), uma pilha de *hubs* empilháveis é formada por dois ou mais *hubs* conectados por um cabo proprietário, fornecido pelo fabricante do *hub*. Todos os *hubs* na pilha agem como um único repetidor Ethernet. Essa tecnologia é muito utilizada para aumentar a quantidade de portas Ethernet da LAN, porém, é importante salientar, que todos os *hubs* empilháveis são proprietários. Isso significa que um *hub* de um deteminado fabricante não pode compor uma pilha com um *hub* de outro fabricante.

Hub modular, é um dispositivo que suporta várias configurações, obtidas pela inserção de módulos. Isso permite que ele seja utilizado como um "canivete suíço", adequando-se o tipo e a quantidade de portas conforme à necessidade de espanção da rede. Entre os módulos que um *hub* modular suporta estão: par trançado, fibra óptica, coaxial, módulos de comutação(*switching)*, de roteamento, de ponte (*bridging)* e de acesso remoto (*remote acces server* – RAS). O *hub* chassi sempre é um dispositivo gerenciável, robusto, confiável (e caro), normalmente utilizado em LANs com quantidade muito grande de estações ou com nível de complexidade elevado.

Os componentes de um *hub* modular, incluem um chassi e os módulos. O chassi é a estrutura principal do *hub*, e os módulos são inseridos no chassi. Cada módulo executa uma certa tarefa ou função, podendo-se adicionar um módulo ao chassi para agregar funcionalidade à rede.

Os hubs modulares provêem funções Ethernet, Token-Ring e FDDI; aceitam módulos de *internetworking* e de gerenciamento de rede.

2.6.3 Comutador (switch)

É um componente que atua diretamente na camada 2 do modelo OSI. Este equipamente é utilizado para conectar segmentos de redes locais.

Um comutador controla o fluxo de tráfego entre os segmentos Ethernet com o mecanismo de encaminhamento automático de tráfego, baseado no aprendizado de endereços MAC. Tomam decisões de encaminhamento de tráfego com base nos endereços dos quadros Ethernet. Para fazer isso, é efetuada a leitura dos endereços de origem em todos os quadros recebidos.

Ao contrário de uma estação normal, que somente verifica os quadros endereçados diretamente para si. A *interface* Ethernet em cada porta de um comutador trabalha em modo "promíscuo". A *interface* verifica todos quadros que trafegam através da LAN Ethernet e não somente os que estão sendo enviados para o próprio endereço físico ou endereço Mac do comutador. À medida que os quadros são lidos em cada porta

do comutador, o *software* verifica o endereço de origem do quadro e inclui o endereço em uma tabela de endereços chamada SAT (*Source Address Table*). É assim que o comutador descobre quais estações podem ser atingidas simultaneamente, já que comuta pacotes utilizando caminhos dedicados. Colisões não ocorrerão, porém, poderá ser experimentada a contenção de dois ou mais quadros que necessitem do mesmo caminho ao mesmo tempo, que serão transmitidos posteriormente, graças aos *buffers* de entrada e saída das portas.

Alguns comutadores, os de grupos de trabalho (*workgroup)*, suportam apenas uma estação ligada por porta, enquanto em outros, os de *backbone* congestionado, suportam segmentos com múltiplas estações ligados em cada porta.

São utilizados não só para a interconexão, mas também para proporcionar um aumento da largura de banda disponível. Esses equipamentos possuem um reservatório de banda, que são distribuídos por suas portas visando adequar-se às necessidades de desempenho da rede.

O Comutador deve ser usado, quando existem situações em que é necessário uma melhora no desempenho da rede.

2.6.6 Pontes (*Bridge)*

É capaz de segmentar uma rede local em sub-redes com o objetivo de reduzir o tráfego ou converter diferentes padrões de LANs (como exemplo: de Ethernet para Token-Ring).

As pontes diferenciam-se dos repetidores, pois manipulam quadros e não sinais elétricos. Possuem vantagens sobre os repetidores, pois não retransmitem ruídos, erros ou quadros malformados. Um quadros deve estar completamente válido para ser retransmitido por uma ponte.

São atribuições de uma ponte:

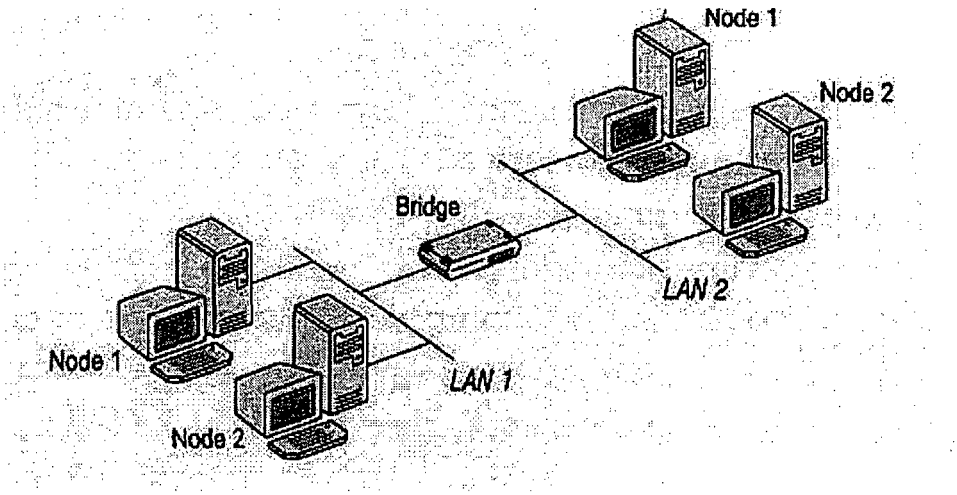
Filtrar as mensagens de tal forma que somente as mensagens endereçadas para ela sejam tratadas;

Armazenar mensagens, quando o tráfego for intenso.

Atuam também, como elementos passivos gerenciadores de rede, podendo coletar dados estatísticos de tráfego de quadros para elaboração de relatórios.

São dispositivos que operam independentemente do protocolo utilizado e os roteadores são dependentes do protocolo, portanto são mais rápidas que os roteadores, pois, não lêem o protocolo para recolher informações de roteamento.

Figura 3 – Exemplo de Utilização de uma Ponte

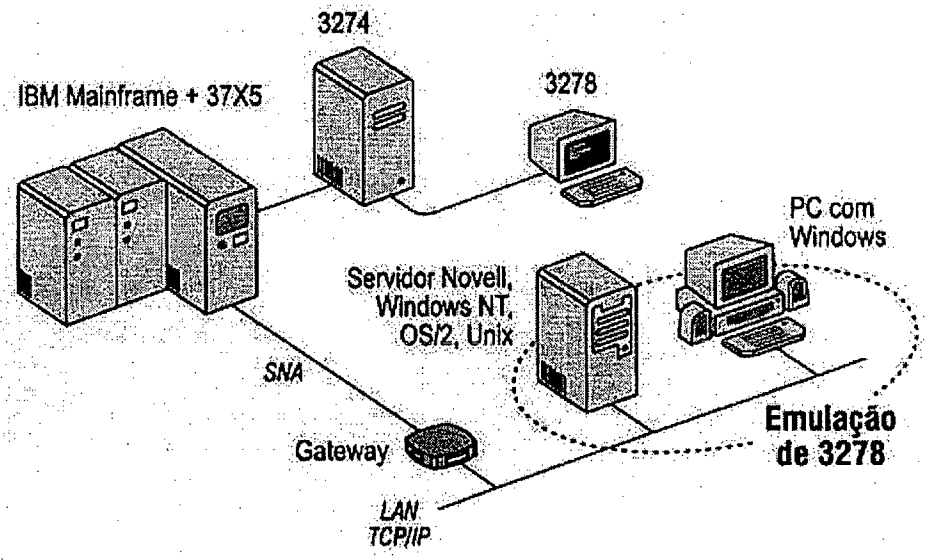


2.6.7 *Gateway*

Permite a comunicação entre duas redes com arquiteturas distintas.

Esses equipamentos resolvem problemas de diferença entre tamanho máximo de quadros, forma de endereçamento, técnicas de roteamento, controle de acesso, *time-outs*, entre outros.

Figura 4 – Exemplo da Utilização de um *Gateway*.



CAPÍTULO 3. ESTRUTURA DE FUNCIONAMENTO DO EMULADOR

O funcionamento do emulador depende basicamente de alguns processos, sendo eles:

3.1 CAPTURA DOS QUADROS ETHERNET

Neste processo, dependemos da captura de todos os quadros que trafegam pelos dispositivos de rede. Através do comando "*ifconfig $i promisc*" (sendo, $i a variável que identifica todos os dispositivos de rede, contidos na máquina que emulará o comutador), no arquivo */etc/rc.d/init.d/network*, acrescentamos na linha de *boot* (onde serão carregados os drives dos dispositivos de rede), automaticamente todas as placas de rede, estarão em modo promíscuo (capturando todos os quadros e os mantendo na memória(*buffer)* do dispositivo), porém, nem todos os *chipsets* das placas de rede, são compatíveis com o modo promíscuo, os produzidos pela Intel não possuem suporte a este modo.

O Linux utiliza as seguintes denominações para os dispositivos de rede.

Eth0 – primeira placa de rede instalada no barramento PCI 1.

Eth1 – segunda placa de rede instalada no barramento PCI 2.

Eth2 – terceira placa de rede instalada no barramento PCI 3

Eth3 – quarta placa de rede instalada no barramento PCI 4.

O emulador, utiliza dois dispositivos de rede, sendo eles eth0 e eth1, sendo, placas da marca 3COM de mesmo modelo e configuração. O uso de dispositivos similares, melhora a compatibilidade para configuração e administração dos mesmos, porém, nada impede que todos os dispositivos de rede sejam de fabricantes diferentes.

3.2 LEITURA DOS QUADROS

```
PROCESSO leitura dos quadros
 Contadoreth0 -> 0
 Contadoreth1 -> 0
 Contadoreth2 -> 0
 recebe(frame0, frame1, frame2)
 SE frame0 then
 contadoreth0 -> 1
 SE frame1 then
 contadoreth1 -> 1
 SE frame2 then
 contadoreth2 -> 1
 retira frame do buffer
 abrir frame
 ler datagrama_Ether -> endrem, enddest, idproto
 limpar tela
  IF idproto = 0X0060 then
   Contador loopback -> 1
   Write (` Protocolo DE LOOPBACK= ',loopback )
  IF idproto = 0X0800 then
   Contador IP -> 1
   Write (` Protocolo IP= ',IP )
  IF idproto = 0x0806 then
   Contador ARP -> 1
   Write (` Protocolo ARP= ',ARP )
  IF idproto = 0x8035 then
   Contador RARP -> 1
   Write (` Protocolo RARP= ',RARP )
  IF idproto = 0x809B then
   Contador Appletalk -> 1
   Write (` Protocolo Appletalk= ',Appletalk )
  IF idproto = 0x8137 then
   Contador IPX -> 1
   Write (` Protocolo IPX= ',loopback )
  IF idproto = 0x0001 then
   Contador Ethernet -> 1
   Write (` Protocolo Ethernet= ',Ethernet )
  IF idproto = 0x0004 then
   Contador SAP -> 1
   Write (` Protocolo SAP 802.2= ',SAP )
  ELSE
   Write (` Protocolo Desconhecido `)
  ENDIF
retorna(endrem, enddest)
FIM leitura dos quadros
```

As placas de rede, em modo promíscuo, capturam todos os quadros. O quadro possui um *header* ou datagrama onde estão contidos os endereços da máquina-destinatária e da máquina-remetente, devido à tecnologia de rede utilizada ser Ethernet,

os endereços estão contidos da seguinte forma: os primeiros 64 *bits* referem-se ao preâmbulo(já comentado no capítulo 1), nos próximos 48 *bits* encontra-se o endereço da máquina-destinatária do quadro e os 48 *bits* seguintes, o endereço da máquina-remetente, como sabe-se quem são as máquina-destinatária e máquina-remetente, qual dispositivo de rede capturou o quadro, obviamente pode-se detectar a localização do mesmo. Segue-se então para o Processo de Consulta na Tabela SAT ou Cadastro na Tabela SAT.

As estatísticas dos protocolos de alto nível, também são efetuadas durante o processo de consulta do datagrama. O programa efetua verificações para identificar qual o protocolo que está atuando na troca de informações entre máquinas, efetua a contabilização e posteriormente a visualização na tela, porém, estas informações não são processadas e visualizadas em tempo real, existe um tempo de atualização dos dados na tela de 4 a 6 segundos, para que o desempenho não seja afetado, pois demanda a utilização do processador e memória, que poderá decrementar o funcionamento do emulador.

Este processo é todo realizado residente em memória.

3.3 CADASTRO/CONSULTA NA TABELA SAT (*SOURCE ADRESS TABLE*)

```
PROCESSO cadastro/consulta na tabela SAT
 recebe( enddest, endrem, dispositivo )
 indicador_broadcast = falso
 porta = -1
 LER_TABELA_SAT
 ir ao inicio da tabela
 DO WHILE não é o final da tabela
   IF TABELA_SAT->enddest = cadastrado THEN
     porta = TABELA_SAT->porta
     ir para o final da tabela
   ELSE
     ir para o próximo registro da tabela
   ENDIF
 ENDDO
 IF porta = -1 THEN
   indicador_broadcast = verdadeiro
   broadcast( pacote, porta )
   IF porta >= 0 THEN
     Enviar frame para destinatario
```

```
      criar novo registro na tabela
      Porta = dispositivo
      gravar TABELA_SAT->endrem = não cadastrado ,
          TABELA_SAT->porta = não cadastrada
    ENDIF
  ENDIF
  retornar( indicador_broadcast, porta, frame )
FIM cadastro/consulta na tabela SAT
```

Sabe-se, a máquina-destinatára do quadro e o dispositivo de rede (eth0 ou eth1) que capturou-o, pode-se então cadastrá-lo na tabela SAT. A tabela contém duas colunas, na primeira coluna estão contidos, os endereços MAC das máquinas que estão conectadas aos dispositivos de rede, através dos *hubs*. Na outra coluna encontramos os dispositivos de rede (eth0 ou eth1) em que o endereço MAC está conectado.

Havendo uma alteração na tabela SAT, automaticamente serão acrescentados os novos registros, quando o emulador cadastrar em sua tabela todos os endereços MAC da rede, as transações tornar-se-ão mais rápidas e o tráfego em *broadcast* raramente acontecerá (salvo algumas exceções como: protocolos ARP, RARP, DHCP, DNS e bootp).

3.4 PROCESSO ENCAMINHAMENTO PARA A PORTA DE DESTINO.

```
PROCESSO encaminhamento para a porta de destino
  Recebe ( porta, indicador_broadcast, frame )
  DO WHILE verdadeiro
    IF fila não vazia THEN
      ler porta
      IF indicador_broadcast = falso THEN
      descobrir_PORTA -> porta
        enviar pacote para porta
      ENDIF
    ENDIF
  ENDDO

FIM encaminhamento para a porta de destino
```

Após a consulta efetuada na tabela SAT para descobrir a máquina-destinatária, o quadro é encaminhado ao dispositivo de rede, ao qual, a máquina-destinatária está conectada, caso, o endereço da máquina-destinatária não esteja

cadastrado na tabela SAT, o quadro será encaminhado a todos os dispositivos de rede através de *broadcast*. Conforme a tabela contiver mais endereços cadastrados, o número de processos de *broadcast* diminuirá, conseqüentemente o tráfego na rede também diminuirá.

CAPÍTULO 4. REDES DE PETRI

4.1 BREVE HISTÓRICO

O conceito de redes de Petri, teve sua origem em 1962 na Alemanha, na tese de doutorado de Carl Adam Petri, *Kommunikation mit Automaten* (Comunicação com Autômatos). O trabalho de Petri foi introduzido nos Estados Unidos por A. W. Holt, que liderava o projeto de teoria de sistemas de informação do *Applied Data Research, Inc.* Entre 1970 e 1975, o grupo mais ativo na condução de pesquisas sobre Redes de Petri foi o de Estruturas de Computação do MIT. Em 1975, no MIT, foi organizada uma conferência sobre redes de Petri. Em 1979 e 1986, pesquisadores, na maioria europeus, organizaram na Alemanha cursos avançados sobre teoria de redes. Anualmente, desde 1980, vêm sendo realizados *workshops* e ultimamente conferências sobre teoria e aplicações de redes de Petri. As últimas conferências têm sido acompanhadas de cursos introdutórios e avançados sobre redes de Petri bem como da demonstração de ferramentas.

O estudo de Redes de Petri desenvolveu-se em duas áreas: aplicações e teoria pura de Redes de Petri. Na primeira, as pesquisas se concentram na aplicação de Redes de Petri na modelagem e análise de sistemas. A segunda área está relacionada com o desenvolvimento de ferramentas básicas, técnicas e conceitos necessários para as aplicações de Redes de Petri.

4.2 O QUE SÃO REDES DE PETRI

É uma ferramenta matemática com uma interpretação gráfica, que permite a modelagem de diversos tipos de sistemas.

Uma Rede de Petri, pode ser interpretada como um grafo direcionado bipartido, mais um estado inicial, denominado, marcação inicial. O grafo direcionado consiste de dois tipos de nós, denominados lugares e transições. Os lugares são elementos essencialmente passivos nas Redes de Petri, ou seja, lugares podem armazenar, mas não podem transformar informações. Em geral são utilizados para representar condições, recursos, informações, banco de dados, entre outros. As transições são os únicos elementos ativos em Redes de Petri. Isso significa que transições podem transformar mas nunca armazenar informações. São utilizadas para representar ações, atividades, transformações, entre outros. Os nós em uma Rede de Petri são relacionados (conectados) por arcos rotulados com pesos (inteiros positivos). Um arco não pode relacionar componentes do mesmo tipo.

Graficamente, lugares são representados por círculos e transições por retângulos. Um lugar *p* é entrada para uma transição *t* se existe um arco direcionado conectando o lugar à transição, nesse caso o lugar é um *lugar de entrada*.

Um lugar *p* é saída para uma transição, se existe um arco direcionado conectando a transição ao lugar, nesse caso o lugar é um *lugar de saída*.

O grafo direcionado define a estrutura de um sistema representado por uma Rede de Petri.

A estrutura de um sistema é também denominada *grafo de suporte* ou *estrutura da rede*.

Uma função de marcação atribui a cada lugar p, um número k, inteiro positivo, de elementos denominados *fichas*. Fichas são representadas graficamente por pontos pretos. As fichas são utilizadas para marcar os lugares e determinar o estado em que se encontra o sistema.

4.3 REDES DE PETRI COLORIDAS - DEFINIÇÃO FORMAL

Com o objetivo de diferenciar as fichas, são associadas cores a estas. Como conseqüência, a cada lugar se associa o conjunto de cores das fichas que podem pertencer a este lugar. A cada transição se associa um conjunto de cores que corresponde as diferentes maneiras de disparar uma transição. Nos casos mais simples,

quando todos os processos possuem rigorosamente a mesma estrutura e são independentes uns dos outros, as cores das transições são diretamente associadas aos processos, e o conjunto de cores dos lugares e das transições são idênticos.

O conjunto de cores de uma transição indica as diferentes maneiras de como ela pode disparar.

- A existência de uma definição formal é muito importante:
  - Base para a simulação.
  - Base para os métodos de verificação formal.
- Na prática, não é necessário conhecer a definição formal:
  - A correta sintaxe é verificada pelas ferramentas de construção de modelos coloridos.
  - O uso correto da semântica de Redes de Petri Coloridas é também garantida pelas ferramentas de simulação e verificação formal.
- **Rede de Petri Colorida** é uma tupla, onde:
  $CPN = (\Sigma, P, T, A, N, C, G, E, I)$

1. Σ é um conjunto finito, não vazio, de tipos, chamado **conjunto de cores**.
2. P é um conjunto finito de **lugares**.
3. T é um conjunto finito de **transições**.
4. A é um conjunto finito de **arcos** onde:

$$P \cap T = P \cap A = T \cap A = \emptyset$$

5. $N : A \rightarrow P \times T \cup T \times P$ é uma função **nó**.
6. $C : P \rightarrow \Sigma$ é uma função **cor**.
7. $G : T \rightarrow Exp$ é uma função **guarda**, onde *Exp* é da forma:

$$\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \Sigma]$$

8. $E : A \rightarrow Exp$ é uma função de **expressão de arco**, onde *Exp* é da forma:

$\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$, onde $p(a)$ é um lugar de $N(a)$.

9. $I : P \rightarrow Exp$ é uma função de **inicialização**, onde *Exp* é da forma:

$\forall p \in P : [Type(I(p)) = C(p)_{MS}]$

- Uma **ligação** de uma transição *t* é uma função *b* definida sobre o conjunto de variáveis de *t* - (*Var*(*t*)), onde:

  1. $\forall v \in Var(t) : b(v) \in Type(v)$
  2. $G(t)\langle b \rangle$

- $B(t)$ é o conjunto de todas a as ligações de *t*.
- Algumas possíveis ligações para **Enviar Pacote**:
  - $\langle n = 1, p = "Redes\_d" \rangle$
  - $\langle n = 3, p = "to\_Nivel :" \rangle$
  - $\langle n = 2, p = "kjghytujg" \rangle$
- Um **elemento de ficha** é um par (*p*, *c*), onde $p \in P$ e $c \in C(p)$.
  - $(Pacotes, (1, "Redes\_d"))$
- Um **elemento de ligação** é um par (*t*, *b*), onde $t \in T$ e $b \in B(t)$.
  - $(TransmitirPacote, (n = 2, p = "e\_Al", s = 8, r = 5))$
- *TE* é o conjunto de todos os elementos de ficha.
- *BE* é o conjunto de todos os elementos de ligação.
- Uma **marcação** é um multi-set de *TE*.

- A marcação inicial M_0 é a marcação que é obtida pela avaliação das expressões de inicialização:
  - $\forall(p,c) \in TE : M_0(p,c) = (I(p))(c)$
- Um **passo** é um multi-conjunto finito e não vazio de *BE*.
- Um passo Y está habilitado em uma marcação M se e somente se:
  - $\forall p \in P : \Sigma_{(t,b)\in Y} E(p,t)\langle b\rangle \leq M(p)$
- Quando um passo Y está habilitado em uma marcação M_1 ela pode ocorrer, mudando a marcação para M_2:
  - $\forall p \in P : M_2(p) = (M_1(p) - \Sigma_{(t,b)\in Y} E(p,t)\langle b\rangle) + \Sigma_{(t,b)\in Y} E(t,p)\langle b\rangle$
- Notação: $M_1[Y\rangle M_2$
- Uma seqüência de ocorrência finita é uma seqüência de marcações e passos:
  - $M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots M_n[Y_n\rangle M_{n+1}$

4.2 PORQUE UTILIZAR REDES DE PETRI COLORIDAS

As Redes de Petri que modelam sistemas reais tendem a ser complexas e extremamente grandes.

As Redes de Petri de alto nível propiciam uma maneira mais compacta de modelar sistemas.

As fichas não são mais anônimas.

A regra de disparo é modificada e depende de certas condições relacionadas com os valores carregados pelas fichas.

Em uma rede de Petri lugar/transição apenas um tipo de ficha é considerado.

Com Redes de Petri Coloridas é possível usar *tipos de dados* e complexa *manipulação de dados*:

Cada ficha tem associada um valor de dado chamado de, *cor da ficha.*

As cores das fichas podem ser investigadas e modificadas, pela ocorrência das transições.

Com Redes de Petri Coloridas é possível obter descrições hierárquicas:

Um modelo grande pode ser obtido pela combinação de um conjunto de submodelos.

Submodelos podem ser reutilizados.

4.3 PRINCIPAIS CARACTERÍSTICAS DE UMA REDE DE PETRI COLORIDA

Combinação de texto e gráfico.

Declarações e inscrições da rede são especificadas através de uma linguagem formal (linguagem de programação):

Tipos, funções, operações, variáveis e expressões.

A estrutura de uma Rede de Petri Colorida é um grafo bipartido:

Para fazer uma Rede de Petri Colorida mais entendível, é importante caprichar no layout gráfico.

O layout gráfico não tem significado formal.

Redes de Petri Coloridas tem os mesmos tipos de propriedades (de concorrência) das Redes Lugar/Transição.

4.4 HABILITAÇÃO E OCORRÊNCIA

Uma ligação (*binding*) atribui uma cor (valor) para cada variável de uma transição.

Um elemento de ligação é um par (t, b).

Um elemento de ligação está habilitado se e somente se:

Existir um número suficiente de fichas corretas (cores corretas) nos lugares de entrada.

A guarda for avaliada verdadeira.

Um elemento de ligação habilitado pode ocorrer: remoção e adição de multi-set de fichas.

Dois ou mais elementos de ligação podem ocorrer concorrentemente.

4.5 EXEMPLO DE UMA REDE DE PETRI COLORIDA

4.5.1 Sistema de Alocação de Recursos

Dois tipos de processos:

- Três processos cíclicos do tipo *q* (estados A, B, C, D e E).
- Dois processos cíclicos do tipo *p* (estados B, C, D, e E).

Três tipos de recursos:

- Representados pelos lugares R, S e T.
- Durante um ciclo, um processo reserva alguns recursos enquanto libera outros:
- Fichas são removidas e adicionadas aos lugares de recursos R, S, e T.
- Um contador de ciclo é incrementado cada vez que um processo fecha o ciclo.

Declarações: Tipos, funções, operações e variáveis.

Cada lugar tem as seguintes inscrições:

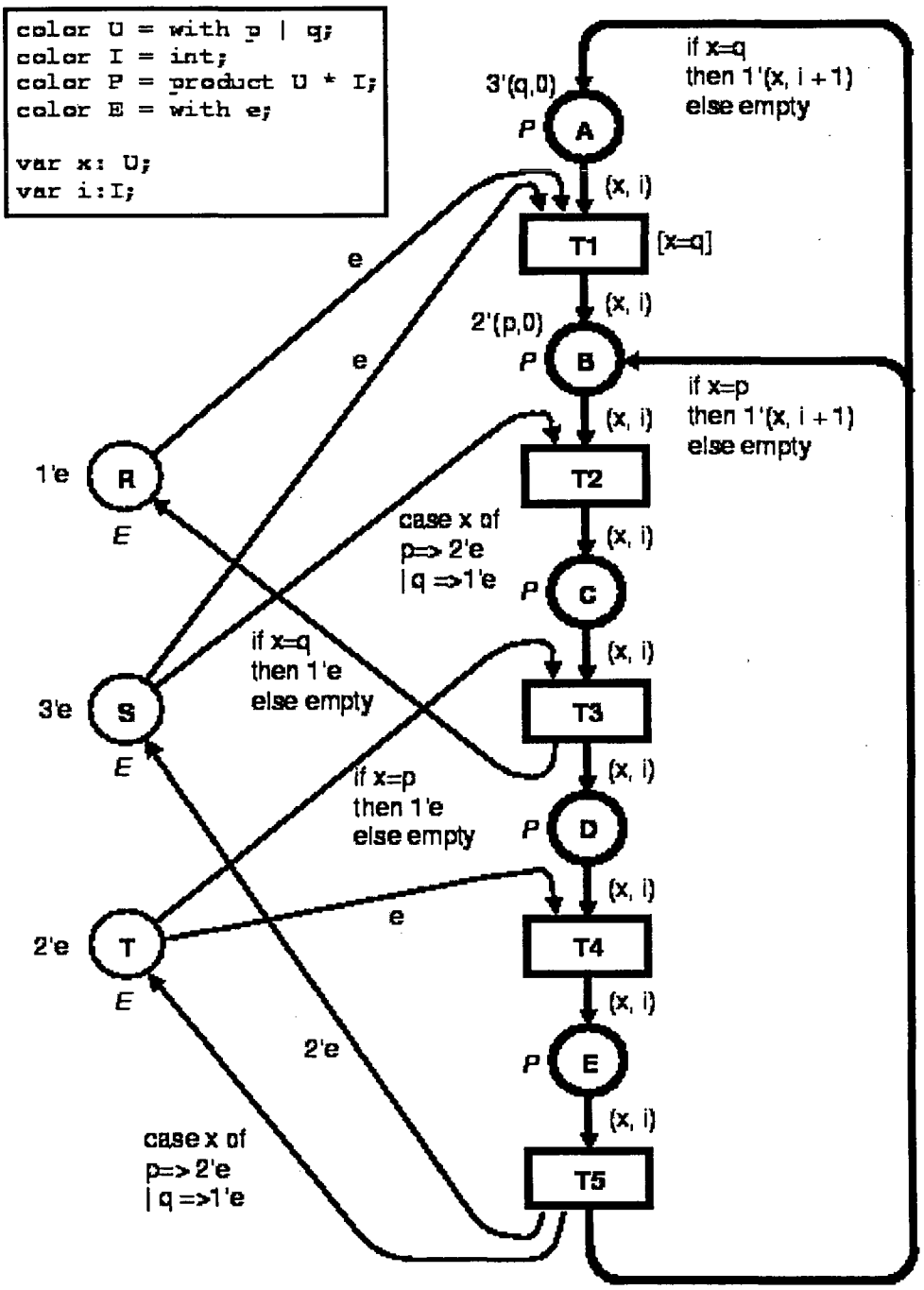
- Nome (para identificação).
- Conjunto de Cores.
- Marcação inicial (multi-set de cores de fichas).

Cada transição tem as seguintes inscrições:

- Nome (para identificação).
- Guarda

Cada arco tem uma expressão de arco que contém algumas variáveis.

Figura 05 - A Modelagem do exemplo.



CAPÍTULO 5. A MODELAGEM E VALIDAÇÃO DA REDE DE PETRI

5.1 DECLARAÇÃO DE VARIÁVEIS

Segue abaixo, a tabela com as declarações globais do modelo. Declaram-se nesta, todas as cores, marcações iniciais e variáveis utilizadas no modelo.

Tabela 2 – tabela de declarações de variáveis

Cores	Variáveis
Cor frameent0 = with Invite0	Var frame0 = frameent0
Cor frameent1 = with Invite1	Var frame1 = frameent1
Cor framesai0 = frame saída eth0	Var frames0 = framesai0
Cor framesai1 = frame saída eth1	Var frames1 = framesai1
Cor buffereth0 = buffer placa eth0	Var buf0 = buffereth0
Cor buffereth1 = buffer placa eth1	Var buf1 = buffereth1
Cor endremetente = endereço remetente	Var endrem = endremetente
Cor endestinatario = endereço destinatario	Var enddest = endestinatario
Cor contador0 = contador entrada eth0	Var cont0 = contador0
Cor contador1 = contador entrada eth1	Var cont1 = contador1

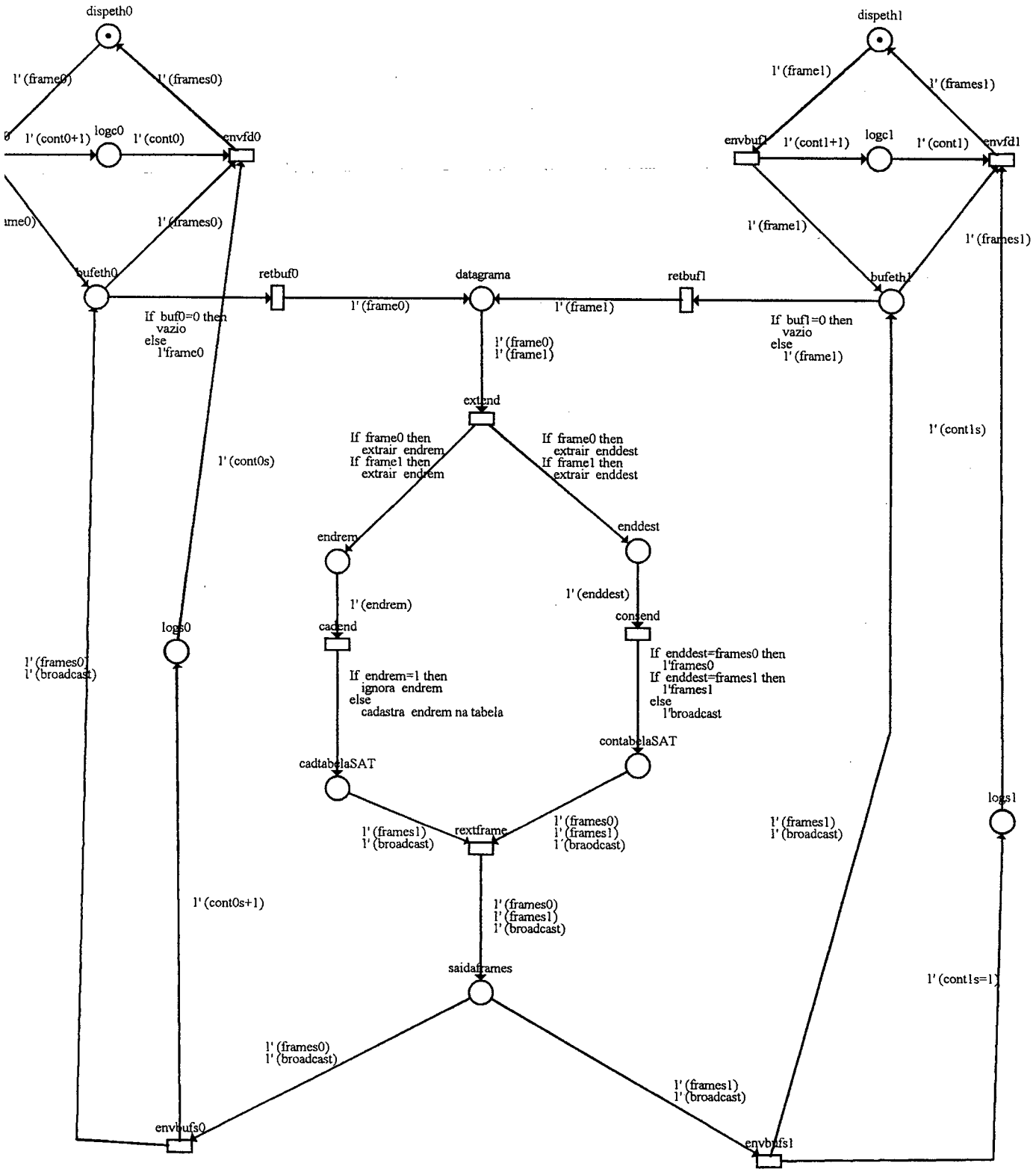
A cor frameent0 refere-se ao quadro capturado pelo dispositivo de rede eth0, cuja marcação inicial é representada pela existência de fichas da cor Invite0. A cor frameent1 refere-se ao quadro capturado pelo dispositivo de rede eth1, cuja marcação inicial é representada pela existência de fichas da cor Invite1. A cor framesai0 refere-se ao quadro que já foi identificado o destinatário e está sendo encaminhado ao dispositivo de rede eth0. A cor framesai1 refere-se ao quadro que já foi identificado o destinatário e está sendo encaminhado ao dispositivo de rede eth1. A cor buffereth0 refere-se a memória (*buffer*) do dispositivo de rede eth0. A cor buffereth1 refere-se a memória do dispositivo de rede eth1. A cor endrem refere-se ao endereço do remetente, extraído dos

quadros de entrada dos dispositivos. A cor enddest refere-se ao endereço do destinatário, extraído dos quadros de entrada dos dispositivos. A cor contador0 refere-se ao número de quadros capturados pelo dispositivo eth0. A cor contador1 refere-se ao número de quadros capturados pelo dispositivo eth1.

5.2 A MODELAGEM

O modelo foi construído e validado, utilizando o programa DaNAMiCS, da University of Cape Town no Sul da África, http://people.cs.uct.ac.za/~idavies/DaNAMiCS.html.

Figura 6 – A modelagem efetuada no programa DaNAMiCS.



5.3 FASES DO MODELO

Os estados implementados no modelo da Rede de Petri Colorida, seguem abaixo:

O modelo é composto por 14 estados, sendo eles: dispeth0, dispeth1, logc0, logc1, bufeth0, bufeth1, datagrama, endrem, enddest, cadtabelaSAT, contabelaSAT, saidaframes, logs0, logs1 e 12 transições, sendo elas: envbuf0, envbuf1, envfd0, envfd1, retbuf0, retbuf1, extend, cadend, consend, rextframe, envbufs0, envbufs1.

Os estados estão caracterizados da seguinte forma:

dispeth0 = Dispositivo de rede eth0;

dispeth1 = Dispositivo de rede eth1;

bufeth0 = Memória (*buffer*) do dispositivo eth0;

bufeth1 = Memória (*buffer*) do dispositivo eth1;

logc0 = contador de quadros capturados pela eth0;

logc1 = contador de quadros capturados pela eth1;

datagrama = cabeçalho do quadro;

endrem = endereço do remetente do quadro;

enddest = endereço do destinatário do quadro;

cadtabelaSAT = cadastrar endereços na tabela SAT;

contabelaSAT = consulta na tabela SAT;

saidaframes = envio dos quadros para os seus dispositivos de saída;

As transições, estão caracterizadas da seguinte forma:

envbuf0 = quadro capturado na eth0 e alocado na memória do dispositivo de rede eth0;

envbuf1 = quadro capturado na eth1 e alocado na memória do dispositivo de rede eth1;

envfd0 = Envio do quadro para o destinatário, conectado ao dispositivo de rede eth0;

envfd1 = Envio do quadro para o destinatário, conectado ao dispositivo de rede eth1;

retbuf0 = retira quadro armazenado na memória do dispositivo eth0;

retbuf1 = retira quadro armazenado na memória do dispositivo eth1;

extend = leitura do datagrama Ethernet e extração dos endereços de destino, remetente e do protocolo de alto nível que esta manipulando os dados;

enddest = consulta na tabela SAT do endereço de destino do quadro, extraído do datagrama Ethernet, o endereço de destino é verificado e encaminhado ao seu destinatário (envbufs0 = eth0 e envbufs1 = eth1), caso o mesmo seja conhecido, caso contrário é enviado através de *broadcast*. O endereço do destinatário é verificado na tabela, se está incluso ou não;

endrem = o endereço do remetente é consultado na tabela para verificar se está incluso ou não, caso não esteja, o mesmo é cadastrado.

rextframe = reestrutura quadro e o envia para o processo de encaminhamento para a máquina-destinatária do quadro;

envbufs0 = Envio do quadro para o dispositivo de rede eth0, para ser encaminhado a máquina-destinatária do mesmo;

envbufs1 = Envio do quadro para o dispositivo de rede eth1, para ser encaminhado a máquina-destinatária do mesmo;

No modelo proposto podemos contar com 4 fases:

1ª) Captura de quadros pelos dispositivos de rede - todos os quadros (aqui representados por fichas) que estão trafegando na camada física, serão capturados pelos processos dispeth0, dispeth1, (variáveis que representam os dispositivos de rede), devido ao fato da promiscuidade imposta no dispositivo de rede (a promiscuidade garante que todos os quadros (fichas) serão capturados pelos dispositivos, não necessitando que estejam endereçados para o emulador), fato este que é gerenciado pelo sistema operacional. As fichas através das transições envbuf0, envbuf1 serão

armazenadas nos processos bufeth0 e bufeth1 (que representam a memória do dispositivo de rede).

2ª) Retirada dos quadros da memória dos dispositivos de rede e extração dos endereços de destino e remetente do datagrama Ethernet - através das transições retbuf0 e retbuf1 as fichas são retiradas e enviadas ao processo datagrama. A transição extend dispara e divide as fichas em dois processos, que representam os endereços de destino através do processo enddest (endereço de destino) e os endereços de remetente através do processo endrem (endereço do remetente).

3ª) Consulta à tabela SAT /Cadastro na tabela SAT – A transição cadend dispara e retira a ficha do processo endrem e deposita na processo cadtabelaSAT (representando o processo de cadastro na tabela SAT). Após a transição consend dispara e retira a ficha do processo enddest e deposita no processo cantabelaSAT (representando o processo de consulta na tabela SAT). O disparo rextframe reestrutura as fichas(quadro) e encaminha para o processo saidaframes

4ª) Envio para o dispositivo de saída – O disparo rextframe reestrutura as fichas(quadro) e encaminha para o processo saidaframes, conforme for o resultado obtido, habilitará os seguintes disparos, envbufs0 e envbufs1 (os quais representam o envio do quadro, para a memória do dispositivo de rede que o enviará ao destinatário correto), e serão entregues aos processos bufeth0 e bufeth1.

As transições envfd0 e envfd1 são responsáveis em retirar as fichas da memória do respectivo dispositivo de rede, e encaminhá-las aos processos dispeth0 e dispeth1, processos esses que representam o envio para a mídia de transmissão (cabeamento).

5.5 SIMULAÇÃO E VALIDAÇÃO

A simulação é um instrumento importante para a descoberta de falhas, eliminação de erros e validação. Durante a construção do modelo, a simulação foi intensivamente usada.

A validação realizada, foi composta por uma matriz de 10.000 fichas. Abaixo segue o resultado obtido que foi gerado pelo programa validador DaNAMiCS em um arquivo chamado emulador.ant.res.

\begin{Simulation}
\begin{transitionThroughput}
envbuf0a : Transition Throughput: 0.13 +/- 0.012
envbufs2a : Transition Throughput: 0.135 +/- 0.013
envbufs0a : Transition Throughput: 0.13 +/- 0.012
retbuf2a : Transition Throughput: 0.135 +/- 0.013
retbuf0a : Transition Throughput: 0.13 +/- 0.012
envfd2a : Transition Throughput: 0.135 +/- 0.013
envbuf2a : Transition Throughput: 0.135 +/- 0.013
envfd0a : Transition Throughput: 0.13 +/- 0.012
envbuf1a : Transition Throughput: 0.136 +/- 0.013
envfd1a : Transition Throughput: 0.135 +/- 0.013
retbuf1a : Transition Throughput: 0.136 +/- 0.013
envbufs1a : Transition Throughput: 0.135 +/- 0.013
extenda : Transition Throughput: 0.135 +/- 0.013
extendb : Transition Throughput: 0.13 +/- 0.012
extendc : Transition Throughput: 0.136 +/- 0.013
rextframea : Transition Throughput: 0.135 +/- 0.013
rextframeb : Transition Throughput: 0.13 +/- 0.012
rextframec : Transition Throughput: 0.135 +/- 0.013
cadenda : Transition Throughput: 0.135 +/- 0.013
cadendb : Transition Throughput: 0.13 +/- 0.012
cadendc : Transition Throughput: 0.135 +/- 0.013
consenda : Transition Throughput: 0.135 +/- 0.013
consendb : Transition Throughput: 0.13 +/- 0.012
consendc : Transition Throughput: 0.136 +/- 0.013

\end{transitionThroughput}
\begin{meanTokensPerPlace}
saidaframesc : Mean Tokens Per Place: 0.138 +/- 0.002
saidaframesb : Mean Tokens Per Place: 0.132 +/- 0.002
saidaframesa : Mean Tokens Per Place: 0.129 +/- 0.002
bufeth2c : Mean Tokens Per Place: 0 +/- 0
bufeth2b : Mean Tokens Per Place: 0.136 +/- 0.002
bufeth2a : Mean Tokens Per Place: 0.136 +/- 0.002
bufeth0c : Mean Tokens Per Place: 0 +/- 0
bufeth0b : Mean Tokens Per Place: 0.135 +/- 0.002
bufeth0a : Mean Tokens Per Place: 0.137 +/- 0.002
dispeth2c : Mean Tokens Per Place: 0 +/- 0
dispeth2b : Mean Tokens Per Place: 0 +/- 0
dispeth2a : Mean Tokens Per Place: 0.129 +/- 0.002
dispeth0c : Mean Tokens Per Place: 0 +/- 0
dispeth0b : Mean Tokens Per Place: 0.135 +/- 0.002
dispeth0a : Mean Tokens Per Place: 0 +/- 0
dispeth1c : Mean Tokens Per Place: 0.127 +/- 0.002
dispeth1b : Mean Tokens Per Place: 0 +/- 0
dispeth1a : Mean Tokens Per Place: 0 +/- 0
bufeth1c : Mean Tokens Per Place: 0.133 +/- 0.002
bufeth1b : Mean Tokens Per Place: 0 +/- 0
bufeth1a : Mean Tokens Per Place: 0.131 +/- 0.002
datagramac : Mean Tokens Per Place: 0.128 +/- 0.002
datagramab : Mean Tokens Per Place: 0.135 +/- 0.002

datagramaa : Mean Tokens Per Place: 0.133 +/- 0.002
endremc : Mean Tokens Per Place: 0.128 +/- 0.002
endremb : Mean Tokens Per Place: 0.127 +/- 0.002
endrema : Mean Tokens Per Place: 0.134 +/- 0.002
enddestc : Mean Tokens Per Place: 0.144 +/- 0.002
enddestb : Mean Tokens Per Place: 0.133 +/- 0.002
enddesta : Mean Tokens Per Place: 0.131 +/- 0.002
logc0c : Mean Tokens Per Place: 0 +/- 0
logc0b : Mean Tokens Per Place: 0 +/- 0
logc0a : Mean Tokens Per Place: 0.865 +/- 0.002
logc1c : Mean Tokens Per Place: 0 +/- 0
logc1b : Mean Tokens Per Place: 0 +/- 0
logc1a : Mean Tokens Per Place: 0.873 +/- 0.002
logc2c : Mean Tokens Per Place: 0 +/- 0
logc2b : Mean Tokens Per Place: 0 +/- 0
logc2a : Mean Tokens Per Place: 0.871 +/- 0.002
cadtabelaSATc : Mean Tokens Per Place: 0.214 +/- 0.003
cadtabelaSATb : Mean Tokens Per Place: 0.198 +/- 0.002
cadtabelaSATa : Mean Tokens Per Place: 0.203 +/- 0.003
contabelaSATc : Mean Tokens Per Place: 0.198 +/- 0.002
contabelaSATb : Mean Tokens Per Place: 0.192 +/- 0.002
contabelaSATa : Mean Tokens Per Place: 0.206 +/- 0.003

\end{meanTokensPerPlace}
\begin{TimeTaken}
timeSpent: 3123.483 Iterations: 10000
\end{TimeTaken}
\end{Simulation}

Foram realizadas 10.000 iterações e obtivemos 100% de aproveitamento.

A alcançabilidade foi atingida em todas as invariantes P e T. O resultado obtido segue abaixo e foi gerado pelo programa. Todas as invariantes estão cobertas tanto em P (Conjunto de Lugares) quanto em T (Conjunto de Transições), significa que todas estão disparando. Segue abaixo apenas os resultados obtidos, a matriz gerada está no anexo 01.

\begin{Invariant}
\begin{T-Statistics}
The net is covered by T-Invariants.
\end{T-Statistics}
\begin{P-Statistics}
The net is covered by P-Invariants.
\end{P-Statistics}
\begin{P-Invariants}

Abaixo segue, o gráfico de coverabilidade obtido, observando que a rede implementada é viva devido à existência de seqüências que tendem ao infinito como é o caso das marcações M=dispeth0,bufeth0 e M=dispeth1,bufeth1. Persistente, pois uma transição uma vez habilitada, permanecerá assim até disparar.

\begin{Coverability}
\begin{Liveness}
The net is Live.
\end{Liveness}
\begin{Boundedness}
The net is Bounded.
\end{Boundedness}
\begin{HomeStates}
The net has Home States.
The net is Safe.
The net is Persistent.
\end{HomeStates}

CAPÍTULO 6. IMPLEMENTAÇÃO DO ALGORITMO

O sistema operacional escolhido foi o Linux. Criado inicialmente em meados de 1991 como *hobby* (passatempo) de um estudante de Ciência da Computação, da Universidade de Helsinki na Finlândia, chamado Linus Torvalds (o nome Linux originou-se da união do nome Linus e do sistema operacional Minix).

Seu código fonte é aberto e gratuito, pois, sua base é o sistema operacional Minix, criado por Andrew Tanenbaum, para utilização no meio acadêmico. A motivação de Tanenbaum ao criar o Minix, originou-se, devido ao sistema operacional Unix, ser proprietário e possuir um valor de mercado oneroso para as instituições de ensino.

O Minix é um clone do Unix, porém, todo o seu código fonte foi reescrito por Tanenbaum, não contém nenhum código da American Telephone and Telegraph (AT&T) e por isso pode ser distribuído gratuitamente.

A origem do Unix tem ligação com o Sistema Operacional Multics, projetado em meados de 60. Tal projeto era realizado pelo Massachusets Institute of Technology (MIT), General Eletric (GE), Laboratórios Bell (Bell Labs) e AT&T. A idéia era que o Multics tivesse características de tempo compartilhado, sendo o sistema mais arrojado da época. Em 1969, foi lançada uma versão primitiva de tal sistema rodando em um computador GE645.

Tempo compartilhado significa, vários usuários compartilhando os recursos de uma mesma máquina. Antigamente, o que existia eram vários terminais (chamados de burros) que faziam acesso à máquinas poderosas que prestavam serviços a este, havia uma grande centralização de operações.

Em 1973, Dennis Ritchie, outro pesquisador da Bell Labs, reescreveu todo sistema para uma linguagem de alto nível, chamada C (desenvolvida por ele), para um PDP-11 (microcomputador mais popular na época).

No período de 1977 a 1981, a AT&T mexeu no sistema, fazendo algumas modificações particulares e lançou o *System III.* Em 1983, após diversas modificações, novidades e otimizações do sistema, foi lançado o UNIX *System V,* comercial.

Hoje, este sistema é padrão internacional no mercado UNIX, sendo comercializado por diversas empresas de grande porte, que necessitam de tudo que um sistema operacional robusto pode oferecer, podemos citar como exemplo as empresas, HP com o sistema operacional HP-UX, a IBM com o AIX, a SUN com o SOLARIS, todos projetados sobre a plataforma UNIX *System V.*

Como linguagem de programação teremos o C, linguagem esta que foi inventada e implementada primeiramente por Dennis Ritchie em um DEC PDP-11. C é o resultado de um processo de desenvolvimento que começou com uma linguagem mais antiga, chamada BCPL, que foi desenvolvida por Martin Richards e influenciou uma linguagem chamada B, inventada por Ken Thompson. Na década de 1970, B levou ao desenvolvimento de C.

Como uma linguagem de médio nível, C permite a manipulação de *bits*, *bytes* e endereços (os elementos básicos com os quais o computador funciona). Um código escrito em C é muito portável.

6.1 PRINCIPAIS FUNÇÕES IMPLEMENTADAS

A seguir, poderemos observar algumas das principais funções implementadas, do algoritmo proposto no capitulo 3.

6.1.1 Criação/Abertura da Tabela SAT

Este processo cria a tabela SAT (quando o programa é inicializado pela primeira vez, ou quando a máquina for desligada, a tabela deverá ser criada novamente), ou efetua a abertura da tabela para consulta dos registros armazenados (esta operação somente será possível, com a tabela já criada e em funcionamento normal).

Figura 07 - Processo de criação/abertura da tabela SAT

```
{
  FILE *fi;
  char registro[12] , branco[6] = "      " ;
  char source[25] = "tab.txt", line[MAXLEN], *p;
  int  continua = 1 , consulta = 1 , i , linha = 0 , linha_g = 0;

  if ((fi = fopen(source, "a+")) == NULL) {
     puts("Nao e' possivel acessar a tabela.\n");
     continua = 0;
}
```

6.1.2 Consulta da Tabela SAT

Apos a criação/abertura da tabela SAT, o processo de consulta será requisitado, quando do recebimento dos quadros. Efetuada a leitura do endereço da máquina-destinatária, faz-se à consulta na Tabela SAT para verificar se o endereço está cadastrado, após a verificação, o quadro será enviado diretamente ao processo de encaminhamento do quadro para o dispositivo de rede em que a máquina-destinatária está conectada.

Figura 08 - Processo de Consulta

```
     }
     fclose(fi);
     if (consulta != 1 )
      for (i=0;i<6;i++)
          registro[i] = endrem[i];
      for (i=0;i<6;i++)
          registro[i+6] = porta[i];
      for (i=0;i<12;i++){
          if (registro[i] == NULL)
             registro[i] = ' ';
      }
      if (linha_g == 0){
         if ((fi = fopen(source, "a+")) == NULL) {
            puts("Nao e' possivel abrir tabela.\n");
         }
         else
         {
           if (fwrite( registro, strlen(registro) , 1 , fi) !=
1 )
              puts("Nao e' possivel gravar na tabela.\n");
         }
      }
      else
      {
         if ((fi = fopen(source, "r+")) == NULL) {
            puts("Nao e' possivel abrir tabela.\n");
         }
         else
         {
           if (fseek( fi, *endrem ,SEEK_CUR) != 0)
             puts("Nao e' possivel movimentacao na tabela.\n");
           if (fwrite( registro, strlen(registro) , 1 , fi) !=
1 )
              puts("Nao e' possivel gravar na tabela.\n");
         }
      }
     }
     fclose(fi);
     return(porta);
  }
  else
  {
     return(branco);
  }
}
```

Em caso negativo o frame será encaminhado para o processo de *broadcast*, que segue abaixo.

Figura 09 - Processo de *Broadcast*

```
char* retorno = ( tabela( destino , porta ));
   puts( "Retorno:");
   puts( retorno ) ;

char* main( void )
{
   char pacote[1500] = " ";
   int  i;
   char porta[6] = "      ";
   char destino[6];
   for (i=0;i<6;i++)
       destino[i] = pacote[i];
   if (porta == ""){
   for (i=6;i<12;i++)
      pacote[i] = '9' ;
      return(pacote);
   }
   else
   {
      or (i=6;i<12;i++)
        pacote[i] = retorno[i];
      return(pacote);
   }
}
```

6.1.3 Cadastro/Atualização da Tabela SAT

Ao obtermos o endereço da máquina-remetente do quadro, pode-se verificar se está cadastrado na tabela SAT, em caso negativo, o endereço será incluído e vinculado ao dispositivo de rede, pelo qual o quadro foi capturado, em caso afirmativo o endereço será descartado.

Figura 10 - Processo para atualização da tabela

```
if (continua == 1) {
      for (i=0;i<6;i++) {
       if (porta[i] != ' ')
          if (porta[i] != NULL )
             consulta = 0;
     }
     fseek( fi, 0, SEEK_SET);
     p = line;
     while(p != NULL) {
        p = fgets(line,MAXLEN,fi);
        linha++ ;
        if (line[0] == enddest[0]){
           if (line[1] == enddest[1]){
            if (line[2] == enddest[2]){
               if (line[3] == enddest[3]){
                  if (line[4] == enddest[4]){
                   if (line[5] == enddest[5]){
                      if (consulta == 1 ) {
                         porta[0] = line[6];
                         porta[1] = line[7];
                         porta[2] = line[8];
                         porta[3] = line[9];
                         porta[4] = line[10];
                         porta[5] = line[11];
                      }
                      else
                      {
                         linha_g = linha ;
                      }
                   }
                  }
               }
            }
           }
        }
     }
     fclose(fi);
```

6.1.4 Encaminhar Quadro para o Destinatário

Após o processo de consulta e descoberta da porta correspondente, o quadro será encaminhado a máquina-destinatária.

Figura 11 - Encaminhar quadro para o destinatário.

```
char* retorno = ( tabela( destino , porta ));
   puts( "Retorno:");
   puts( retorno ) ;

char* main( void )
{
   char pacote[1500] = " ";
   int  i;
   char porta[6] = "      ";
   char destino[6];
   for (i=0;i<6;i++)
       destino[i] = pacote[i];
   if (porta == ""){
   for (i=6;i<12;i++)
      pacote[i] = '9' ;
      return(pacote);
   }
   else
   {
      or (i=6;i<12;i++)
        pacote[i] = retorno[i];
      return(pacote);
   }
}
```

6.1.5 Análise dos Quadros para a Descoberta dos Protocolos de Alto Nível.

Este processo efetua a leitura do datagrama, para identificar o protocolo que está trafegando dados na rede e após encaminha para o processo específico de contabilização, conforme o protocolo identificado.

Figura 11 – Processo para identificar protocolo

```
switch (frame_protocol)
      {
      case ETH_P_IP:
      case SN_PROT_PPP:
      case SN_PROT_SLIP:
      case SN_PROT_LOOP:
        if (options.tcp_option || options.udp_option ||
options.ip_option)
          handle_ip (ip_ptr, length);
        break;
      case ETH_P_ATALK:
        if (options.at_option)
          exatalk ((struct at_frame_type *) ((void *) buf +
ETH_HLEN), length);
        break;
      default:
        if (options.ip_option || options.at_option)
          handle_other (buf, length);
        break;
      }
```

6.1.5.1 Análise do Protocolo Appletalk.

Encontramos no campo de 16 *bits* conhecido por Tipo ou Tamanho, um identificador, refere-se ao tipo dos dados do protocolo de alto nível, sendo transportados no campo de dados do quadro Ethernet.

Ao realizarmos a leitura deste campo, podemos identificar através de códigos especiais, o protocolo de alto nível que está trafegando os dados contidos no quadro, como por exemplo: o código 0x008 refere-se ao protocolo IP.

Figura 12 – Processo para gerar estatísticas no protocolo Appletalk

```
}
void
exatalk (struct at_frame_type *buf, int length)
{
    switch (buf->appletalk_type2)
    {
    case 0x01:
      SM_regis.rtmprd++;
      break;
    case 0x02:
      SM_regis.nbp++;
      break;
    case 0x03:
      SM_regis.atp++;
      break;
    case 0x04:
      SM_regis.aep++;
      break;
    case 0x05:
      SM_regis.rtmpreq++;
      break;
    case 0x06:
      SM_regis.zip++;
      break;
    case 0x07:
      SM_regis.adsp++;
      break;
    default:
      break;
    }
}
```

6.1.5.2 Análise do Protocolo IP (Internet Protocol)

O protocolo IP permite a conexão de computadores tanto em pequenas redes locais, quanto em redes corporativas interligando diversos países. O maior exemplo disso é a *internet*, que adota como padrão o protocolo TCP/IP.

O protocolo IP é dividido em dois protocolos distintos, sendo eles:

TCP (*transfer control protocol*), é um protocolo orientado a conexão, confiável, possui controle de fluxo. Amplamente utilizado em aplicações que necessitam de garantias de entrega de informações.

Através da leitura do cabeçalho do TCP, descobre-se o endereço da porta do TCP, do aplicativo remetente, que originou a solicitação. O comprimento do campo é de dois *bits*.

Figura 13 – Processo para gerar estatísticas no protocolo TCP

```
case IPPROTO_TCP:
      {
      if (
#if (__BYTE_ORDER == __BIG_ENDIAN)
          (buf->ip_off != 0) && ((buf->ip_off & 0x4000) == 0)
#elif (__BYTE_ORDER == __LITTLE_ENDIAN)
          (buf->ip_off != 0) && ((buf->ip_off & 0x0040) == 0)
#else
#error      "Please fix <endian.h>"
#endif
        )

{
          tally( 0, &StatMem->tcp_port );
        }
      else
        {
                if (buf->ip_hl == 5)
            {
            if ((src_port = ntohs (((struct tcphdr *) ((void *) buf
+ 20))->th_sport)) <= SN_MAX_TCP_PORTS)
                tally( src_port, &StatMem->tcp_port );
            if ((dest_port = ntohs (((struct tcphdr *) ((void *) buf
+ 20))->th_dport)) <= SN_MAX_TCP_PORTS)
              if (dest_port != src_port)
                {
                  tally( dest_port, &StatMem->tcp_port );
                }
            }
          else
            {
            if ((src_port = ntohs (((struct tcphdr *) ((void *) buf
+ ((buf->ip_hl) * 4)))->th_sport)) <= SN_MAX_TCP_PORTS)
                tally( src_port, &StatMem->tcp_port );
            if ((dest_port = ntohs (((struct tcphdr *) ((void *) buf
+ ((buf->ip_hl) * 4)))->th_dport)) <= SN_MAX_TCP_PORTS)
              if (dest_port != src_port)
                {
                  tally( dest_port, &StatMem->tcp_port );
                }
            }
        }
      break;
```

O segundo protocolo é o UDP (*User Datagram Protocol*), é um protocolo sem conexão, não confiável, ideal para aplicações que não necessitam nem de controle de fluxo, nem da manutenção da seqüência das mensagens enviadas. Ele é amplamente usado em aplicações em que a entrega imediata é mais importante do que a entrega precisa.

Através da leitura do cabeçalho UDP, descobre-se o endereço da porta do UDP, do aplicativo remetente que originou a solicitação. O comprimento do campo é de dois *bytes*.

Figura 14 – Processo para gerar estatísticas no protocolo UDP.

```
case IPPROTO_UDP:
      {
      if (
#if (__BYTE_ORDER == __BIG_ENDIAN)
          (buf->ip_off != 0) && ((buf->ip_off & 0x4000) == 0)
#elif (__BYTE_ORDER == __LITTLE_ENDIAN)
          (buf->ip_off != 0) && ((buf->ip_off & 0x0040) == 0)
#else
#error      "Please fix <endian.h>"
#endif
        )
        {
          tally( 0, &StatMem->udp_port );
        }
      else
        {
          if (buf->ip_hl == 5)
            {
            if ((src_port = ntohs (((struct udphdr *) ((void *) buf
+ 20))->ui_sport)) <= SN_MAX_UDP_PORTS)
              tally( src_port, &StatMem->udp_port );
            if ((dest_port = ntohs (((struct udphdr *) ((void *) buf
+ 20))->ui_dport)) <= SN_MAX_UDP_PORTS)
              if (dest_port != src_port)
                {
                  tally( dest_port, &StatMem->udp_port );
                }
            }
          else
            {
            if ((src_port = ntohs (((struct tcphdr *) ((void *) buf
+ ((buf->ip_hl) * 4)))->ui_sport)) <= SN_MAX_UDP_PORTS)
              tally( src_port, &StatMem->udp_port );
            if ((dest_port = ntohs (((struct tcphdr *) ((void *) buf
+ ((buf->ip_hl) * 4)))->ui_dport)) <= SN_MAX_UDP_PORTS)
              if (dest_port != src_port)
```

CAPÍTULO 7.CONSIDERAÇÕES FINAIS E SUGESTÕES PARA NOVOS TRABALHOS

Dentre as conclusões obtidas com o presente trabalho, pode-se citar:

As empresas que detém o domínio da tecnologia para confecção de comutadores (*switches)*, não disponibilizam e tão pouco possuem interesse em auxiliar novas experiências.

Os conhecimentos de baixo nível, quando falamos em protocolos estão bastante difundidos, apenas na teoria. Quando são necessárias soluções mais práticas e ilustrativas, a abordagem nos livros sobre o tema é bastante abstrata. O mesmo acontece com o Linux, apesar de ser um sistema aberto e de domínio público, poucas são as referências quando falamos em acesso a dispositivos.

Este projeto pode ser uma solução de baixíssimo custo para pequenas empresas que necessitam melhorar a performance de sua rede de dados. O fato de podermos utilizar equipamentos obsoletos, minimiza o custo na aquisição do *hardware*. O Linux sendo um sistema operacional de domínio público, não necessitando adquirir licença de *software* e principalmente, esta máquina que além de fazer a emulação do comutador, também pode comportar os serviços de: Servidor DNS, Servidor DHCP, Roteador de Pacotes IP, *Firewall*, Servidor NAT, etc.

Uma implementação que pode ser realizada, para melhorar as funcionalidades do emulador é criar uma memória(*buffer)*, para armazenar os quadros de um mesmo arquivo com a finalidade de verificar a existência de vírus, através da recomposição do arquivo. Após a verificação o mesmo seria novamente dividido em quadros e enviado ao destinatário. Isso implicaria na perda de desempenho, porém, justifica-se devido a sua funcionalidade de segurança.

Outra funcionalidade é a aplicabilidade do conceito de agentes móveis. Um agente poderia trafegar junto com os quadros de um determinado processo que não está

completo, no emulador os quadros seriam abertos e recompostos, os agentes continuariam o processamento, após o término, novamente o arquivo seria dividido em e enviado ao destinatário. Assim seria utilizado o poder de processamento da máquina que está emulando o comutador.

Podemos citar também que pode-se adequar a implementação para a leitura dos quadros de outras tecnologias de rede, como exemplo: Token Ring. Assim implementa-se a funcionalidade de Ponte(*Bridge)* no emulador, conectando e possibilitando a troca de informações entre duas redes distintas.

8.REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA,M.G.; ROSA,P. C. **Internet, Intranet e Redes Corporativas.** Editora Brasport. Rio de Janeiro. 2001.

ALVES, J. **Comunicação de Dados**. Editora Makron Books. São Paulo. 1994.

ANUNCIAÇÃO, H. **Linux Guia Prático**. Editora Érica. São Paulo. 1999.

BACH,M.J. **The Design of the Unix Operating System**. Prentice Hall, New Jersey. 1990.

BALL, B. **Usando Linux**. Editora Campus. Rio de Janeiro. 1999.

BRITO, T.C.M. **Arquiteturas de Redes de Computadores OSI e TCP/IP**. Makron Books. São Paulo. 1997.

CARDOSO, J.; VALETTE, R. **Redes de Petri**. Editora da UFSC. Florianópolis. 1997.

Cisco – Site sobre a empresa e seus produtos, disponível em http://www.cisco.com.br/ data do último acesso 05/03/2002.

Coloured Petri Net of Aarhus - These Web pages present the activities of the CPN group at the Department of Computer Science, University of Aarhus, Denmark. Disponível em http://www.daimi.au.dk/CPnets/, último acesso em 21/04/2002.

COMER, D. E. **Interligação em Rede com TCP/IP.** Editora Campus. Rio de Janeiro. 1998.

CYCLADES BRASIL. **Guia Internet de Conectividade**. Editora SENAC. São Paulo. 2000.

Dinamics – a Petri Net Editor - The DaNAMiCS project of the Data Network Laboratory, Computer Science Department at University of Cape Town is to build a Petri Net editor. http://people.cs.uct.ac.za/~idavies/DaNAMiCS.html. 2000, último acesso em 20/07/2002.

DIÓGENES, Y. **Certificação CISCO**. Editora Axcel Books.Rio de Janeiro. 2001.

Enterasys – Site sobre a empresa e seus produtos, disponível em http://www.enterasys.com/ data do último acesso 10/03/2002.

GROSMANN, C.,A , K. **Red Hat Conectiva Linux 4 – Guia do dministrador e do Usuário**. Book Express.1999. Rio de Janeiro.

GROSMANN, C.A.K. **Linux – Guia Prático do Administrador e do Usuário**. Editora Book Express. 1999.

JUNIOR, J.H.T.; SUAVE, J.P.; MOURA, J.A.B.; TEIXEIRA, S.Q.R. **Redes de Computadores**. Makron Books. São Paulo. 1999.

KURT, J. **A Brief Introduction to Coloured Petri Nets**. Computer Science Department, University of Aarhus. Denmark. http://www.daimi.au.ds/~Kjensen/paper-books/. 1997

LEINECKER, R.C.; NYE, J. **Visual C++ Ferramentas Poderosas**. Berkeley Brasil. São Paulo. 1995.

LEINWAND, A .; CONRAY, K.F. **Network Management**. Editora Addison Wesley. Massachusetts. 2000.

MAÇAN, E. **Unix Comandos de Usuário.** Editora Novatec. São Paulo. 2001.

MARCELO, A. **Intranet em Ambiente Linux.** Editora Brasport. Rio de Janeiro, 2002.

MARQUES, W.S. **TCP/IP: Projetando Redes**. Editora Brasport. Rio de Janeiro. 2002.

MORERA, P.H., GONZÁLEZ, T.M.P. **A CPN Model of the MAC layer**. Departamento de Ingenieria Telematica, University of Las Palmas de gran Canaria. http://www.cma.ulpgc.es. 1999.

MORIMOTO, C. **Hardware PC-Manual Completo**. Book Express. Rio de Janeiro. 2000

NUNES, J.R.S. **Comunicação de Dados**. LTC-Livros Técnicos e Científicos. Rio de Janeiro. 1989.

PALLARES, A.C. **Redes e Sistemas de Telecomunicações**. Brasport. Rio de Janeiro. 2001.

PALMA, L.; PRATES, R. **TCP/IP-Guia de Consulta Rápida**. Editora Novatec. São Paulo. 2001.

SAMPAIO, C. **TCP/IP e Intranets**. Editora Brasport. Rio de Janeiro. 2001.

SCHILDT, H. **C Completo e Total.** Makron Books. São Paulo. 1996.

SCRIMGER, R. LASALLE, P. PARIHAR, M. GUPTA, M. **TCP/IP a Bíblia**. Editora Campus. Rio de Janeiro . 2002.

SOARES, L.F.; SOUZA,G.L.; COLCHER, S. **Redes de Computadores das LANs, MANs e WANs às Redes ATM**. Editora Campus. Rio de Janeiro. 1999.

SPURGEON, C.E. **Ethernet o Guia Definitivo**. Editora Campus. Rio de Janeiro. 2000.

STREBE, M.; PERKINS, C. **Firewalls.** Makron Books. São Paulo. 2002

TANENBAUM, A. S. **Redes de Computadores**. Editora Campus. Rio de Janeiro. 1997.

The Petri Nets Bibliografy - The Petri Nets Bibliography This database is the most comprehensive online Petri Nets bibliography. Disponível em http://www.daimi.au.dk/PetriNets/bibl/aboutpnbibl.html, último acesso em 20/04/2002.

THOMAS, R.M. **Introdução às Redes Locais**. Makron Books. São Paulo. 1997

TORRES, G. **Redes de Computadores.** Editora Axcel Books. Rio de Janeiro. 2001.

VAHALIA, V. **Unix Internals – Ther New Frontiers**. Prentice Hall. New Jersey. 1996.

XYLAN COMPANY. **The Switching Book**. Editora Xylan. Rio de Janeiro. 2000.

ANEXOS

Anexo 01 – Matriz de Invariantes gerada pelo programa DaNAMiCS

\begin{P-Invariants}

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0

Places:

contabelaSATc; contabelaSATb; contabelaSATa; cadtabelaSATc; cadtabelaSATb; cadtabelaSATa; logc2c; logc2b; logc2a; logc1c; logc1b; logc1a; logc0c; logc0b; logc0a; enddestc; enddestb; enddesta; endremc; endremb; endrema; datagramac; datagramab; datagramaa; bufeth1c; bufeth1b; bufeth1a; dispeth1c; dispeth1b; dispeth1a; dispeth0c; dispeth0b; dispeth0a; dispeth2c; dispeth2b; dispeth2a; bufeth0c; bufeth0b; bufeth0a; bufeth2c; bufeth2b; bufeth2a; saidaframesc; saidaframesb; saidaframesa

\end{P-Invariants}

\begin{T-Invariants}

1 0 0
0 1 0
0 0 1
1 0 0
0 1 0
0 0 1
1 0 0
0 1 0
0 0 1
1 0 0
0 1 0
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 1 0
1 0 0
1 0 0
0 1 0
1 0 0
0 1 0
1 0 0
0 1 0

Transitions:

consenda; consendb; consendc; cadenda; cadendb; cadendc; rextframea; rextframeb; rextframec; extenda; extendb; extendc; envbufs1a; retbuf1a; envfd1a; envbuf1a; envfd0a; envbuf2a; envfd2a; retbuf0a; retbuf2a; envbufs0a; envbufs2a; envbuf0a

\end{T-Invariants}

\begin{Equations}

logc2c = 0
logc2b = 0
logc2a + dispeth2a = 1
logc1c = 0
logc1b = 0
logc1a + dispeth1c = 1
logc0c = 0
logc0b = 0

logc0a + dispeth0b = 1
contabelaSATc + enddestc + datagramac + bufeth1c + bufeth1a + dispeth1c + saidaframesc = 1
contabelaSATb + enddestb + datagramab + dispeth0b + bufeth0b + bufeth0a + saidaframesb = 1
contabelaSATa + enddesta + datagramaa + dispeth2a + bufeth2b + bufeth2a + saidaframesa = 1
cadtabelaSATc + endremc + datagramac + bufeth1c + bufeth1a + dispeth1c + saidaframesc = 1
cadtabelaSATb + endremb + datagramab + dispeth0b + bufeth0b + bufeth0a + saidaframesb = 1
cadtabelaSATa + endrema + datagramaa + dispeth2a + bufeth2b + bufeth2a + saidaframesa = 1
bufeth1b = 0
dispeth1b = 0
dispeth1a = 0
dispeth0c = 0
dispeth0a = 0
dispeth2c = 0
dispeth2b = 0
bufeth0c = 0
bufeth2c = 0

\end{Equations}
\begin{IncidenceMatrix}

0	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	1	
0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	-1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	-1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	

```
0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 -1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 -1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 -1 0 0 0
0 0 0 0 0 0 0 1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0
```

Places:

contabelaSATc; contabelaSATb; contabelaSATa; cadtabelaSATc; cadtabelaSATb; cadtabelaSATa; logc2c; logc2b; logc2a; logc1c; logc1b; logc1a; logc0c; logc0b; logc0a; enddestc; enddestb; enddesta; endremc; endremb; endrema; datagramac; datagramab; datagramaa; bufeth1c; bufeth1b; bufeth1a; dispeth1c; dispeth1b; dispeth1a; dispeth0c; dispeth0b; dispeth0a; dispeth2c; dispeth2b; dispeth2a; bufeth0c; bufeth0b; bufeth0a; bufeth2c; bufeth2b; bufeth2a; saidaframesc; saidaframesb; saidaframesa

Transitions:

consenda; consendb; consendc; cadenda; cadendb; cadendc; rextframea; rextframeb; rextframec; extenda; extendb; extendc; envbufs1a; retbuf1a; envfd1a; envbuf1a; envfd0a; envbuf2a; envfd2a; retbuf0a; retbuf2a; envbufs0a; envbufs2a; envbuf0a

\end{IncidenceMatrix}

\end{Invariant}