Centro Brasileiro de Pesquisas Físicas CNPq
Coordenação de Atividade Técnicas

MONITORAÇÃO DA REDE LOCAL: O MODELO DO CBPF

Márcio P. de **Albuquerque**,
e-mail: mpa@cat.cbpf.br

Marcelo P. **Albuquerque**
e-mail: marcelo@cat.cbpf.br

Celso dos Santos **Almeida**
e-mail: celsomy@cat.cbpf.br

Luiz Gustavo Cardoso **D'avila**
e-mail: davila@novell.cat.cbpf.br

ÍNDICE

PREFÁCIO

Este trabalho consiste na análise e monitoramento da rede local do CBPF e baseia-se em um modelo simplificado de comunicação entre computadores, visando medir a situação atual de tráfego da rede da nossa instituição.

Em redes locais, a disponibilidade da rede (para usuários ou servidores) passa por um pico de "ótima" utilização e cai rapidamente em função do número de estações nela presente. Baseados nesta característica fomos incentivados a desenvolver uma metodologia de medida do tráfego da rede.

Para obtermos a situação atual do CBPF construímos um programa em linguagem C que, anexado a um driver de comunicação da companhia americana Crynwr, nos permite ter acesso ao conteúdo do pacote de dados enviado numa rede Ethernet.

Um pacote na rede Ethernet é definido pelo padrão ANSI - IEEE 802.3 e podemos dividi-lo simplificadamente em 5 partes principais: o endereço de destino, o endereço de origem, o tipo e o tamanho do pacote e finalmente a parte de dados. O endereço de destino nos fornece dados sobre o envio de informações por diferentes máquinas já o tipo de pacote informam sobre os diferentes protocolos (como os utilizados na rede do CBPF: SPX/IPX, que caracterizam o trafego Novell e TCP/IP que caracteriza as máquinas Unix).

As análises obtidas mostram uma utilização da rede em função das horas de trabalho porém uma análise do trabalho em conjunto com os diferentes tipos de estações está em andamento.

Baseado nesta análise nós pretendemos conhecer o peso da comunicação das diferentes máquinas presentes na rede: estação servidora, clientes e computadores locais.

Isto permitirá prever uma possível utilização da rede por tecnologias crescentes como a vídeo conferência e vindouras.

INTRODUÇÃO

A rede do CBPF, como muitas outras redes locais, começou com um pequeno número de computadores e gradativamente aumentou sua necessidade de expansão. Esse crescimento se deve à grandes vantagens encontradas em rede locais de computadores, como por exemplo o compartilhamento de recursos, a alta confiabilidade, os serviços de acesso, o correio eletrônico etc. No compartilhamento de recursos tem-se como objetivo fazer com que todos os programas, dados e equipamentos estejam disponíveis para qualquer um na rede independente da localização física do recurso e do usuário. Como serviço de acesso temos por exemplo a consulta ao acervo disponível na biblioteca. A previsão de crescimento desta rede é elevada para os próximos anos, sobretudo na utilização do parque computacional de estações de trabalho.

O CBPF dispõe atualmente de uma rede local de computadores, onde estão distribuídas diferentes plataformas do tipo pessoal, estações de trabalho, impressoras e um roteador. Esta rede é composta de quatro segmentos, centrados num servidor Novell localizado na CAT (Coordenação de Atividades Técnicas). Esses quatros segmentos são designados pelas conexões físicas neles presentes : LNCC, CBPF, LAFEX e Estações de Trabalho. Através do braço do LNCC temos acesso à **Rede Rio** e consequentemente à **Internet**. No segmento CBPF localizam-se a maioria dos PC, com uma clientela bastante heterogênea que vai desde computadores que prestam serviços a laboratórios àqueles que estão nos serviços de apoio, como as secretarias e departamentos.

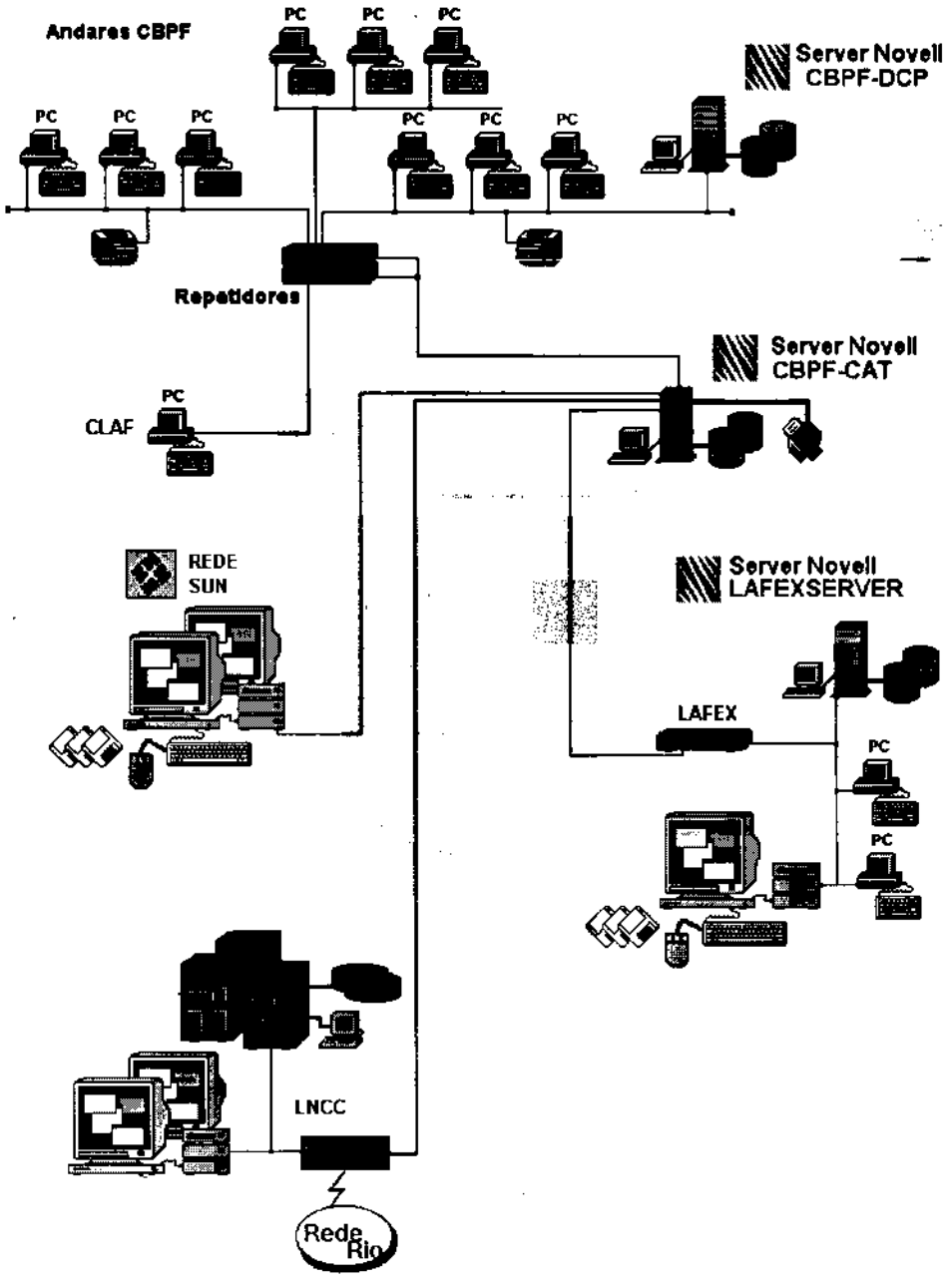

Figura 1 - Disposição física do parque computacional - Rede Local do **CBPF.** A rede esta centrada num servidor Novell/CAT. Deste servidor saem 4 segmentos : LNCC, CBPF, LAFEX e SUN/IBM. Via o LNCC está a saída da rede para a Rede-Rio e Internet.

Constatando-se que com o desenvolvimento hoje já alcançado, não só oriundo da entrada de novas máquinas mas também do maior uso que vem sendo dado à informática

através de incentivo interno e da mídia, inúmeros problemas surgiram. Dentre os quais podemos citar:

1) sobrecarga das estações servidoras de rede.
2) a baixa confiabilidade da rede: o tipo de topologia adotada torna a rede extremamente sensível a uma desconexão "descuidada" de algum ponto de rede, ou de uma pane em uma interface de um nó da rede.
3) não gerenciamento: dificultando a operacionalidade e a expansão.
4) baixa flexibilidade tecnológica.
5) saturação do meio físico: banda de frequência está saturada pelo número de usuários atuais.
6) difícil manutenção: crescimento desordenado.

Este trabalho destina-se adquirir as informações relativas a saturação do meio físico, e de como este estará daqui a alguns anos com o aparecimento de novas tecnologias que exijam um meio físico de alta performace (exemplo video-conferência). A detecção, prevenção e solução dos problemas que atualmente ocorrem impôs a criação e o desenvolvimento de um sistema de monitoramento da rede, objetivando planejar o crescimento, otimizar a produtividade e a qualidade dos serviços oferecidos.

DESCRIÇÕES: A ESTRUTURA DA REDE DO CBPF

Abaixo estão listados alguns tópicos importantes para a compreensão do sistema de monitoração da rede:

Arquitetura - A arquitetura da rede do CBPF é do tipo broadcasting (difusão), onde um único canal de comunicação é compartilhado por todas as máquinas da rede.

Mensagem - As mensagens para serem transportadas através da rede são segmentadas, recebendo o nome de pacotes. Um pacote é dividido em vários campos, cada um correspondendo a uma determinada informação na troca da mensagem, como por exemplo o endereço de origem, o endereço de destino, o tipo de dado, etc. Pacotes enviados por quaisquer máquinas são recebidos por todas as outras presentes no barramento da rede. Ao receber um pacote cada máquina verifica o campo de endereço destino, se o pacote for destinado a uma outra máquina ele é simplesmente ignorado.

Topologia - Na topologia de rede por barra, a qualquer instante uma máquina é o mestre e tem a permissão de transmitir, todas as outras máquinas devem abster-se de transmitir.

Alocação do canal - O método para alocação do canal é denominado *dinâmico descentralizado*; não há entidade central, cada máquina deve decidir por si só se deve ou não transmitir, não há uma divisão do tempo em intervalos discretos estabelecendo um rodízio (como no método estático), que permita cada máquina transmitir apenas durante o seu intervalo de tempo.

Transmissão - A transmissão de dados entre as estações é síncrona. O sincronismo entre a estação transmissora e a receptora é estabelecido no início de cada pacote, sendo mantido durante todo o tempo de transmissão do pacote. Na prática, o pacote é precedido de um preâmbulo de 7 octetos.

Estruturação - Para reduzir a complexidade de projeto, a maioria das redes é organizada em camadas ou níveis, cada uma construída sobre sua predecessora. O número de camadas, o nome, o conteúdo e a função de cada camada diferem de uma rede para outra. No entanto, o propósito de cada camada é oferecer certos serviços às camadas superiores, protegendo essas camadas do detalhe de como os serviços são de fato implementados.

Modelo - É utilizado o modelo OSI/ISO (ISO - "International Standards Organization"; OSI - Interconexão de Sistemas Abertos) por lidar com sistemas abertos, isto é, sistemas de definição aberta, e que consequentemente permitem a interconexão com outros sistemas (diferentes fabricante). Este modelo está organizado hierarquicamente em sete camadas: aplicação, apresentação, sessão, transporte, rede, enlace de dados e física.

As sete camadas:
A definição precisa das camadas de rede estão relacionadas com o processo de comunicação de um usuário ao outro presente na rede. Estas estão descritas resumidamente abaixo. Para maiores detalhes ver Tanenbaum [Tanenbaum - 87].

Aplicações - essa camada dentro do processo de comunicação é representada pelo usuário final.

Apresentações - a camada de apresentação se relaciona com a sintaxe e a semântica da informação transmitida. Desempenha certas funções que são solicitadas com freqüência suficiente para justificar que se encontre uma solução geral para elas, em vez de deixar que cada usuário resolva os problemas. Ao contrário de todas as camadas inferiores, que se interessam em mover bits.

Sessão - o propósito desta camada é prover os mecanismos necessários para organizar e sincronizar o diálogo e o gerenciamento da troca de dados entre entidades de apresentação.

Transporte - a função dessa camada é aceitar dados da camada de sessão, dividi-los se necessário em unidades menores, passá-las à camada da rede e garantir que os pedaços cheguem corretamente ao outro lado[1].

Rede - A camada de rede se preocupa com o controle da operação da rede. Uma questão projeto fundamental é determinar como os pacotes são roteados da origem para o destino. O controle de congestionamento da rede também pertence à camada da rede.

[1] O protocolo TCP está situado nesta camada. O TCP manipula o estabelecimento e a terminação de conexão entre processos, bem como a sequência de dados que podem estar fora de ordem e finalmente a verificação e montagem destes dados ("checksum", "acknowledgments" e "time-out")

Enlace de dados - sua tarefa principal é pegar a facilidade de transmissão de dados brutos e transformá-la em linha que pareça à camada de rede ser livre de erros de transmissão.

Física - a camada lida com a transmissão de bits eletronicamente através do canal de comunicação.

Meio de transmissão - tem o propósito de transportar um fluxo bruto de bits de uma máquina até outra. O meio de transmissão utilizado é o cabo coaxial fino de impedância de 50 ohms (± 2 ohm), com taxa de transmissão de 10 Mbps (10Mhz).

Transmissão dos dados - os dados são transmitidos como um sinal digital que consiste de uma sequência de pulsos de tensão constante. O código é o bifásico Manchester, onde a transição ocorre na metade de cada período de bit. A **Figura-2** apresenta uma referência de pulso trafegante numa rede ethernet clássica. Este transmissão serve como um clock e também como dado.

- transição de nível alto para baixo ⇒ 1 lógico
- transição de nível baixo para alto ⇒ 0 lógico

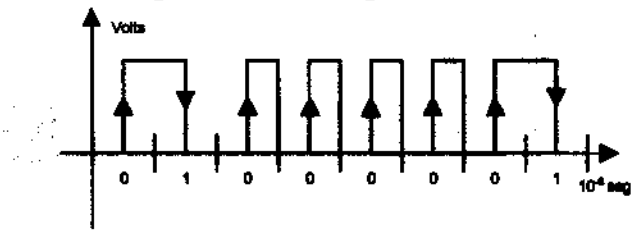

Figura-2 - Caracter "A" em denotação binária "01000001"(41 H). Numa rede Ethernet, uma transição de nível alto para baixo representa o nível lógico '1', e uma transição de nível baixo para alto representa um nível lógico '0'.

Padrão - existem padrões que dividem a camada de enlace de dados em duas subcamadas para assim facilitar o seu entendimento ou compreensão. Essa duas subcamadas compreendem a sub camada de acesso ao meio e a de controle de enlace lógico. O protocolo CSMA/CD ("Carrier Sense Multiple Access Whit Colision Detection") é o tipo de protocolo implementados nos padrões Ethernet 2.0 e IEEE 802.3, que são os padrões utilizados em rede por barramento.

Neste método de acesso ao meio uma estação ao querer transmitir, primeiro "ouve" o meio para saber se existe alguma transmissão em progresso. Caso ninguém esteja transmitindo, a estação pode iniciar a transmissão. Caso contrário, a estação espera por um período de tempo, reiniciando o procedimento novamente. Se duas estações ou mais queiram transmitir ao mesmo instante de tempo, ocorrerá o que chamamos de colisão. Neste método a detecção de colisão é realizada durante a transmissão. Depois de inicializada a transmissão, a estação fica o tempo todo escutando o meio, e notada uma colisão, aborta a transmissão. Detectada a colisão, a estação espera por um tempo e tenta retransmitir novamente.

O padrão IEEE 802 implementa uma série de definições descrevendo a "fiação, a topologia física, a topologia elétrica", e os esquemas de acesso para produtos de rede. O padrão IEEE 802 é uma família de padrões para redes locais de computadores, relacionados ao meio físico e de enlace de dados definidos no modelo OSI (descrevem os protocolos mais usados nas duas mais inferiores do modelo OSI), eles se restringem a essas camadas. O IEEE 802.3 CSMA/CD descreve um padrão ligado ao sistema Ethernet mais antigo.

Estações de trabalho - as estações de trabalho (SUN) usam o sistema operacional UNIX, que utiliza o protocolo TCP/IP para se comunicarem na rede e utilizam o padrão Ethernet 2.0 que corresponde à camada de enlace de dados.

Os PC - os micros computadores da linha PC na sua maioria usam o sistema operacional Windows/MS-DOS localmente. Quando conectados à rede o sistema operacional é o Netware 3.11, sistema operacional do servidor CBPF-CAT que se utiliza do protocolo IPX/SPX para fazer a comunicação na rede e para a camada física e de enlace de dados utiliza o padrão IEEE 802.3

Estrutura de um pacote IPX/SPX - Consiste de duas partes: uma correspondendo ao cabeçalho com 42 bytes e outra correspondendo aos dados que podem variar de 0 até 534 bytes. O IPX consiste de duas partes: um cabeçalho de 30 bytes e outra que são os dados que pode ser de 0 até 546 bytes. O tamanho mínimo do pacote são 30 bytes (correspondente ao tamanho do cabeçalho) e o tamanho máximo 576 bytes. O pacote SPX é semelhante ao pacote IPX exceto que ele contém 12 bytes a mais no cabeçalho. O tamanho mínimo de um pacote SPX é de 42 bytes(cabeçalho) e o tamanho máximo é de 576 bytes.

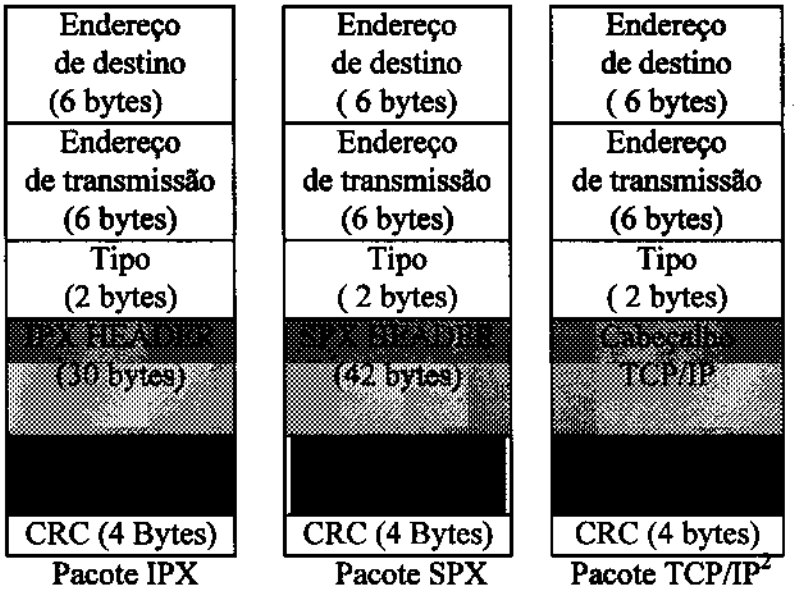

Figura 3 - Representação simplificada de pacotes IPX/SPX e TCP/IP

[2] Maiores informações sobre o pacote TCP/IP vide referências [Douglas - 88] e [Stevens - 90]

O MODELO DA REDE

A seguir descreveremos um modelo simplificado de rede que nos estimulou a realizar as medidas de ocupação do canal de troca de mensagem. Neste modelo levamos em consideração a presença de diferentes máquinas organizada em uma rede em barra, com um probabilidade de enviar mensagem na rede dividida em categorias. A título de exemplificação, podemos fazer uma analogia de uma rede em barra com uma reunião entre indivíduos. Nesta reunião todas as pessoas têm a possibilidade de falar, porém apenas uma deve ocupar o canal de comunicação num determinado momento, enquanto as outras a escutam. Se várias pessoas tentam falar ao mesmo tempo então a conversação fica difícil pois o canal fica incompreensível. As pessoas presentes têm diferentes "vontade de falar" e ocupam o canal de maneira desigual fazendo a conversação fluir . Este mesmo princípio funciona para uma rede em barra.

Da mesma forma, numa rede Ethernet, podemos atribuir dois estados para uma máquina: transmitindo ou recebendo. Algumas máquinas transmitem durante um tempo muito maior que a maioria (como por exemplo as maquinas servidoras de rede que se comunicam com as máquinas clientes). Pretendemos então conhecer qual a relação da transmissão de n máquinas com uma dada taxa de transmissão num canal de comunicação.

Para exemplificar vamos supor um modelo simples de rede onde estão presentes três máquinas. Simbolizaremos que no estado '1', ela esta transmitindo e no estado '0' ela esta calada ou mais precisamente, ouvindo. Numa rede do tipo Ethernet quando 2 ou mais máquinas tentam transmitir ao mesmo tempo ocorre um fenômeno de colisão e ambas não conseguem transmitir a mensagem. Na realidade este fenômeno de transmissão simultânea é mais complexo que a simples consideração de um tempo único para todas as maquinas, pois existe um tempo de propagação da onda eletromagnética pelo canal coaxial[3] [Zakir - 88].

Quando apenas uma máquina transmite ocorre sucesso e diremos que a rede está na situação "*Falando*" (F). Se nenhuma máquina transmite, a rede está na situação "*Calada*" (C). Quando duas máquinas tentam transmitir ao mesmo tempo ocorre a situação de colisão (X). Todas estas configurações são descritas numa rede simplificada na tabela binária abaixo, para três maquinas A, B e C (n = 3). Os estados da rede são listados na ultima coluna à direita.

[3] - Na realidade este tempo é de aproximadamente 2/3 da velociade da luz. As referências para cálculo desta velocidade num cabo coaxial (linha de transmissão), podem ser obtidas em livros de eletromagnetismo clássico.

A	B	C	Estado da Rede
0	0	0	C
0	0	1	F
0	1	0	F
1	0	0	F
1	1	0	X
0	1	1	X
1	0	1	X
1	1	1	X

Tabela 1 - Representação dos estados de uma rede, para três máquinas **A, B e C**. C (*Calado*) para o canal desocupado, F (*Falando*) para o canal ocupado e X para uma colisão.

Podem então ocorrer 2^n situações diferentes na rede, mas só três nos interessam, i.é, quando ocorre sucesso na transmissão (F). Devemos ressaltar que para o estudo de colisões o número de estados '**X**' (**nX**) aumenta consideravelmente com o aumento do número de máquinas, i.e., $nX = (2^n - n - 1)$. Suponhamos que para o modelo as máquinas têm o mesmo peso de comunicação, isto é, a mesma "vontade de falar", independente da sua função na rede. Logo atribuindo probabilidade diferente aos estados '1' (p(1)) e '0' (p(0)) das máquinas A, B e C podemos calcular a **O**cupação do **C**anal (OC) em função do número de máquinas presentes na rede. Para uma rede com ***n*** máquinas teremos :

$$OC = n\, p(0)^{n-1}\, p(1)$$

Sabemos no entanto que na realidade esta "vontade de falar" dada pela probabilidade p( ), varia de máquina para a máquina e é certamente diferente, se ela é uma estação servidora ou uma estação cliente.

Podemos traçar o gráfico de utilização do canal em função do número de máquinas conectadas à rede. Este gráfico, apresentado na **Figura-3**, passará por um pico de "ótima" utilização e cairá rapidamente em função do número de usuários nela presente. Esta queda é função das colisões que se tornam mais expressiva com o aumento do número de máquinas.

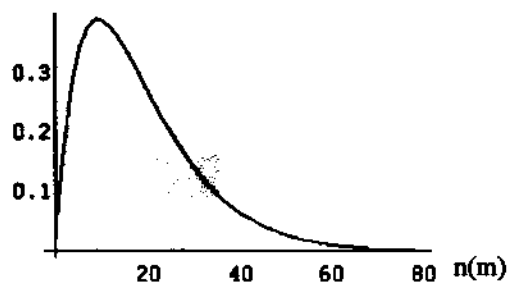

Figura - 3 - Simulação da *Ocupação da rede* **x** *Numero de maquinas*[n(m)] para uma probabilidade de 5% por máquina.

Desta forma pretendemos determinar experimentalmente a ocupação da rede para diferentes categorias de máquinas, i.e., estações servidoras Novell e Unix, micro computadores pessoais e estações que realizem video-conferência.

A título de exemplificação devemos dizer que através deste modelo podemos prever que para 6 máquinas, todas com "vontade de falar" de 0,5 (50%) existe uma saturação completa do canal de comunicação. Logo a expectativa atual é que na rede do CBPF (250 máquinas) esta "vontade de falar" esteja bem abaixo deste valor, pois como usuário, não notamos uma total saturação do canal de comunicação.

REALIZAÇÃO DAS MEDIDAS

As medidas são realizadas através da aquisição e análise de cada pacote enviado na rede. Para cada pacote adquirido contabilizamos, através da análise dos campos de endereços anteriormente descritos, os protocolos que trafegaram na rede (SPX/IPX - TCP/IP).

Para monitoração da rede optamos por desenvolver um software que nos desse controle total dos dados adquiridos, e principalmente não influenciar nas medidas, pois a maioria dos programas de monitoração influenciam em até 10% o trafego de pacotes na rede. Este software foi desenvolvido na linguagem C (Borland C/C++ 3.1) e permite através da presença de um sistema de interfaceamento com a placa de rede acessar os pacotes que nela trafegam. O sistema de interfaceamento ("packet driver") foi desenvolvido pela companhia americana Crynwr [Crynwr - 95], e adquirido por nós via Internet. "Packet Drivers" são rotinas que linkadas com o nosso programa permitem acessar os drivers de rede residentes que são instalados em memória antes de executarmos o software de monitoração. Existe um packet driver para cada placa de rede, que é acessado pelo programa de monitoração via uma interrupção de software padronizada no endereço (0x61)[4].

Na **Figura-4** exemplificamos sucintamente o seu funcionamento. Um microcomputador é conectado à rede em qualquer um dos segmentos que se queira monitorar. O programa de monitoração é "linkado" com o "packet-driver" da Crynwr. Antes da execução do programa de monitoração rodamos um driver de rede, que fica

[4] - Não confundir a interrupção de software 0x61 com a de hardware (IRQ), em geral selecionada na linha 3. No anexo A apresentamos a listagem do packet-driver utilizado. Utilizamos computadores do tipo 386/40Mhz para a monitoração da maior parte da rede. Um 486/100Mhz foi usado na verificação da precisão das medidas. Onde numa rede simples tinhamos conhecimentos do número de bytes nela trafegados.

residente na memória do computador, permitindo a comunicação do software com o buffer da placa. A placa de rede é colocada no modo "promíscuous", i.e., e recebe no seu buffer interno todos os dados que circulam na rede .

Uma vez no buffer interno uma variável "flag"[5] é modificada pela interrupção de hardware no interior do programa de monitoração. Executamos então na interrupção de software 0x61 que transfere todos os dados rapidamente para uma memória interna do computador.

Este dados são em seguida analisados byte a byte, seguindo a organização descrita anteriormente.

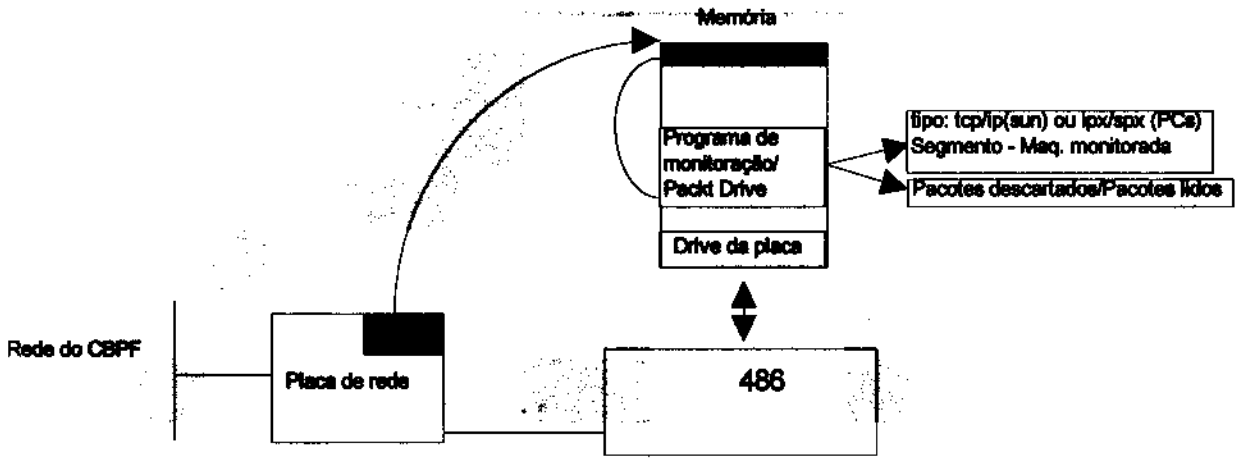

Figura 4 - Representação do funcionamento do programa de Monitoração. O diagrama representa um microcomputador conectado à rede carregado com as rotinas de acesso ao driver da rede. Programa de Monitoração está em execução na memória do microcomputador.

A medida é realizada em intervalos de tempo regulares e o tamanho dos pacotes são integrados ao longo deste intervalo. O programa nos informa o tipo de pacote (TCP/IP ou IPX/SPX), a quantidade de pacotes trafegados, seus tamanhos em bytes, o total de bytes trafegados no segmento e o total para uma máquina escolhida inicialmente. Durante a análise de um pacote pelo computador todos os outros pacotes que chegam são perdidos, até a liberação da memória na placa de rede. No entanto esta informação, relacionada a perda do pacote, é guardada e pode ser utilizada para estimarmos o erro de nossa medida.

A porcentagem de ocupação da rede por máquina é realizada fazendo-se medições, nos diversos segmentos do CBPF, em máquinas previamente selecionadas. Estas medidas são realizadas durante vários dias durante várias horas numa mesma semana.

[5] - Ver variável packet-len no anexo A.

RESULTADOS OBTIDOS

Em seguida apresentamos alguns resultados, que se mostram relevantes, de diferentes categorias de máquinas presentes na rede: micros, estação de trabalho e estação com vídeo-conferência. Devemos ressaltar que alguns micros estão na configuração cliente e outros na configuração servidor.

No eixo horizontal encontram-se os horários do período de monitoração do segmento. Esse período pode ser parte de um dia ou até mais de um dia. No eixo vertical encontra-se a porcentagem de ocupação do canal de comunicação. Este percentual tem como base a capacidade máxima de tráfego de um canal Ethernet fino, 10Mbps.

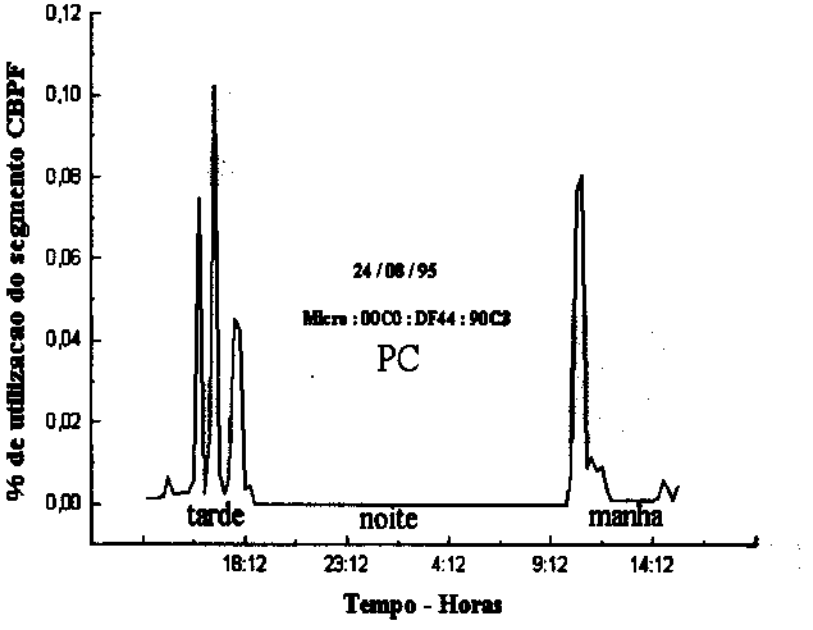

Figura 5 - Gráfico de utilização do segmento CBPF por um Microcomputador. Esse micro [00C0DF4490C3] foi escolhido de forma aleatória entre os 250 micros hoje conectados ao segmento CBPF. Podemos notar a presença de alguns picos entre às 15:00 h e às 10:00 h do outro dia. O máximo de utilização representou 0,1% do total do canal de utilização.

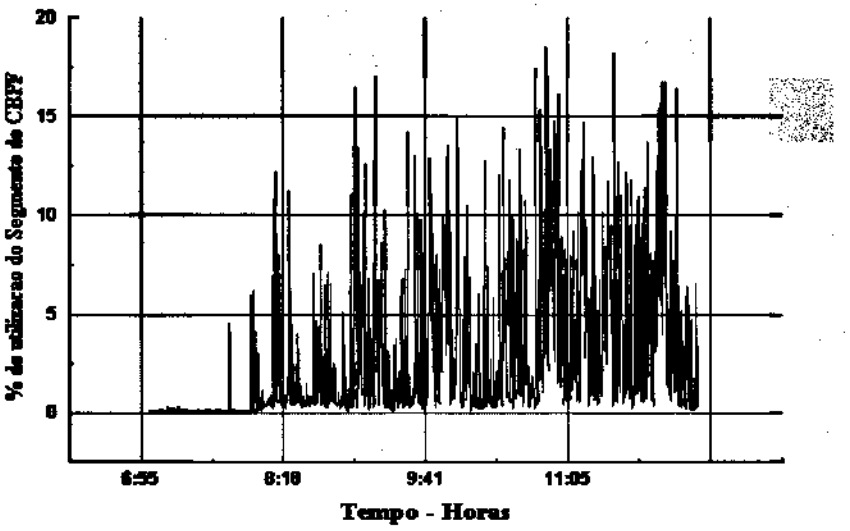

Figura 6 - Gráfico de utilização do segmento CBPF. Podemos notar que o pico de utilização está em torno de 15% do canal de comunicação.

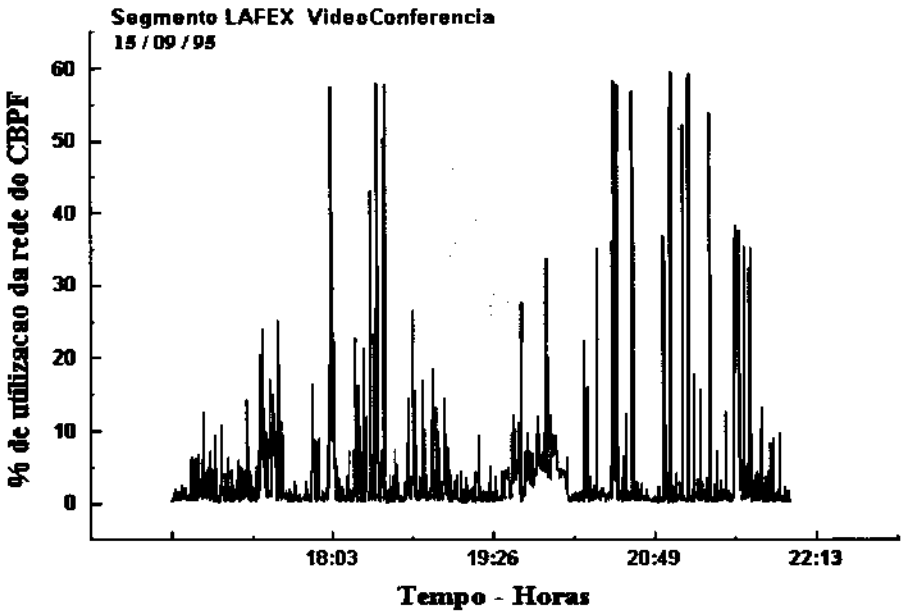

Figura 7 Gráfico do segmento LAFEX. Durante a conferência CHEP95 no Hotel Glória. A Video-conferência foi realizada entre duas estações Suns. O canal foi utilizado a mais de 60%, se considerarmos erros de medidas (que não estão adicionados ao gráfico)

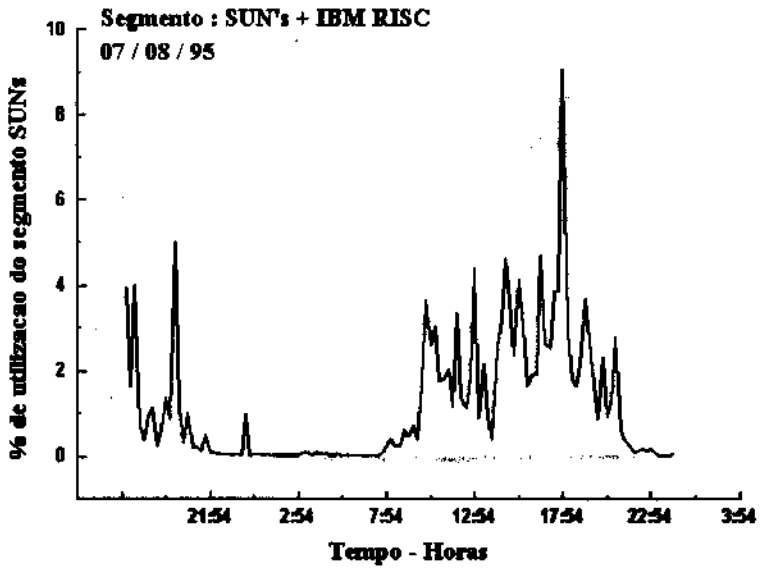

Figura 8 - Gráfico de utilização do segmento SUN's + IBM RISC

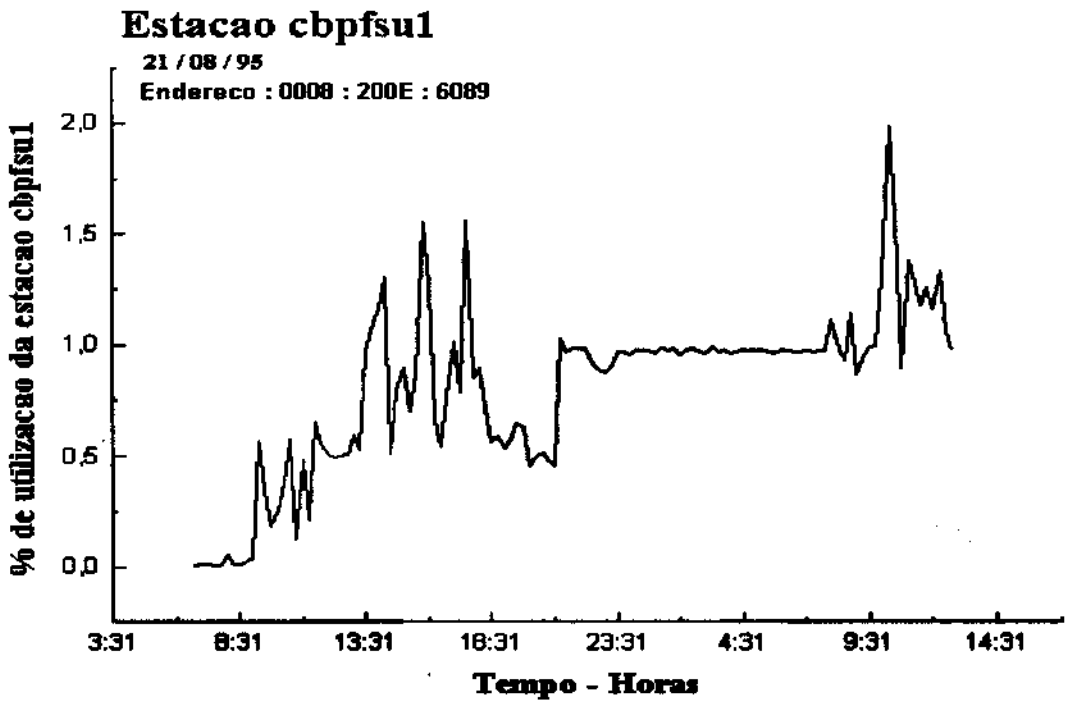

Figura 9 - Gráfico de utilização do segmento SUN's + IBM RISC para a estação Sun servidora cbpfsu1

CONCLUSÃO

Este trabalho foi realizado com o objetivo de nos colocar a par da situação de ocupação da rede local do CBPF, bem como na utilização e no impacto de tecnologias que impõem um alto tráfego de rede sobre a arquitetura atual. O trabalho esta baseado na análise do tráfego de pacotes no meio físico por diferentes estações (servidoras e clientes). Não incluímos neste estudo uma análise da resposta do servidor da rede à fila de pacotes, nem a problemas causados por uma abertura acidental ou proposital de conectores nos diversos pontos. Ressaltamos porém que estes são problemas que na prática podem comprometer o "fator de qualidade operacional" de uma rede. A resposta do servidor de rede diminui de velocidade com o aumento da quantidade de conexões à ele. Uma eventual má conexão também pode comprometer o tempo de resposta do servidor, devido ao tipo de rede em barra por nós utilizado.

A analise exclusiva de ocupação da rede, por categorias, se mostrou satisfatória para respondermos a questão de saturação do meio físico. Poderíamos melhorá-la otimizando as rotinas que diminuíssem as perdas de pacotes trafegados na rede durante a análise através de computadores mais rápidos. Um estudo complementar a este trabalho e que também deve ser realizado para uma análise mais completa da utilização do canal de comunicação é o tempo de retardo (ou tempo de "delay"). Este corresponde ao intervalo de tempo decorrido entre o instante em que um pacote é colocado para ser transmitido até o instante em que o pacote foi recebido com sucesso.

Podemos afirmar então que a situação atual da rede do CBPF, mais especificamente do meio físico, não é de saturação de tráfego. Os problemas atualmente apresentados (lentidão nas conexões, respostas demoradas, etc.) poderão estar relacionadas com o gerenciamento do servidor às fila de pacote à ele submetida, ficando sobrecarregado com o aumento de conexões que ocorrem ao longo do tempo. Desta forma uma possível solução para aumentar a velocidade de resposta à uma grande demanda é a divisão, do principal servidor (CBPF-CAT), em vários outros. Solução esta que já se encontra em andamento no momento desta redação, pois o servidor CBPF-CAT será dividido nos servidores CBPF-CDI[6] e CBPF-CFC[7]. Devemos ressaltar que esta divisão não implica em que a topologia de rede atual e a limitação do canal não ficarão em xeque num breve futuro. Pois ficou claro que a ocupação do canal Ethernet para duas estações em transmissão continua de imagens leva a quase saturação do canal de comunicação atual.

[6] - CDI : Coordenação de Documentação e Informacão Cientifica.
[7] - CFC : Coordenação de Formação Cientifica.

Para finalizarmos esta conclusão lembramos que o atual *Projeto de Cabeamento Estruturado* que prevê principalmente a atualização do meio físico do CBPF e a melhor gerência da rede, se apresenta como uma solução para a demanda futura destas novas tecnologias que estarão disponíveis em breve.

REFERÊNCIAS

[Tanenbaum - 87] *Redes Locais de Computadores*, Andrew S. Tanenbaum, Editora Campus;

[Douglas - 88] *Internetworking With TCP/IP-Principles, Protocols and Arquitecture*, Douglas Comer, Prentice Hall;

[Zakir - 88] *Redes locais - O Estudo do Seu Elementos*, José Zakir Junior , Livros Técnicos e Científicos Ed. LTDA;

[Soares - 90] *Redes locais*, Luis Fernando G. Soares, Campus LTDA;

[Crynwer - 95] - *Crynwr Packet Driver Collection* endereço http://www.crynwr.com;

[Stevens - 90] - *UNIX - Network Programming*, W. Richard Stevens, Prentice Hall;

ANEXO A

Programas Utilizados

Anexo A

Programas Utilizados

Anexo segue o programa fonte por nós desenvolvidos em linguagem C (Borland C/C++ 3.1) para fazermos a monitoração da rede local do CBPF. Para que ele seja executado deve-se antes executar o driver de rede da placa. No nosso caso utilizávamos o driver NE2000 de 1993 da Crynwr Software para placas de rede NE2000. Segue também o Packet-driver, um programa da Crynwr Packet Driver Collection de domínio público que pode ser encontrado na Internet no endereço WWW : http://www.crynwr.com, utilizado para interfacear o programa de monitoração com a placa de rede.

PROGRAMA DE MONITORAÇÃO

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <cty e.h>
#include <time.h>

#include " ktdrvr.h"

/*--------------- variaveis globais ----------------------------------------*/
char buffer[10048];     /* single buffer */
int  acket_len;                 /* the length of data in buffer */
int intno;                              /* our handle */
int oldmode;
char myeaddr[6];                /* our Ethernet address */
int handle iad;

int     x1 f;           /* entrada do endereco via teclado*/
unsigned int s   c d a b;
char    *x *y *k *j *n *l;
char    nome[7];

struct ESTAT{
            unsigned long int TotalSize;
            char strTime[10];
        };

unsigned int  GlobalIndex=0;
unsigned long int SomaSPX=0  SomaIP=0  DiscardPCKT=0  ReceivedPCKT=0  NoneOfAbove=0;
unsigned long int EtherSomaPckt=0  EtherSomaByte=0;
unsigned long int OldSomaSPX=0  OldSomaIP=0  OldDiscardPCKT=0  OldReceivedPCKT=0
OldNoneOfAbove=0;
struct ESTAT EstatSPX[450];
struct ESTAT EstatIP[450];

union XXX{
          unsigned int EtherSearch[3];
          unsigned char Eth[12];
         };

union XXX Add;

unsigned int *PtEt1=buffer+6;
unsigned int *PtEt2=buffer+8;
unsigned int *PtEt3=buffer+10;

unsigned int *ty e= (unsigned int *)(buffer+12);
```

```
time_t t_atual t_old;
long int estat[20];
char nodes_send[200];
char nodes_recev[200];
unsigned char  addr_rec[6];
char  addr_send[6];

/*--------------- PROTOTYPES INTERNOS ------------------------------------*/
void fmtline (int addr  char *buf  int len);
static void ctohex (register char *buf  register int c);
int   estatfunc              (char *buffer);
void  show_estat             (void);
void  GravaEstat     (void);
void  Finaliza               (void);

char user_send (char *buffer);

/********************** ROTINAS ***************************************/
/*----------------- receiver ----------------------------------------------*/
int interru t receiver(b   di  si  ds  es  dx  cx  bx  ax  i   cs  flags)
{
       if ( acket_len || cx > sizeof(buffer))
          {
               es = di = 0;          /* discard this  acket */
           DiscardPCKT++;
          }
   else
          {
              if (ax == 0)
                 {
                  es = FP_SEG(buffer);/* tell them to stick it in our buffer */
                  di = FP_OFF(buffer);
            ReceivedPCKT++;
                 }
            else
                  {
                    acket_len = cx;/* second u call -- remember size. */
                  }
      }
}
/*--------------------- dum _bytes ---------------------------------------*/
dum _bytes(char *bytes  int count)
{
       int n;
       char buf[16];
       int address;
       address = 0;
       while (count)
           {
                if (count > 16) n = 16;
                else n = count;
                fmtline (address  bytes  n);
                address += n;
                count -= n;
                bytes += n;
       }
}
/*------------------- fmtline ------------------------------------------------*/
/* Print a buffer u  to 16 bytes long in formatted hex with ascii
 * translation  e.g.
 * 0000: 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f  0123456789:;<=>? */
void fmtline (int addr  char *buf  int len)
{
       char line[80];
       register char *a tr *c tr;
       unsigned register char c;
   memset (line ' ' sizeof(line));
       ctohex (line addr >> 8);
       ctohex (line+2 addr & 0xff);
       a tr = &line[6];
       c tr = &line[55];
       while (len-- != 0)
            {
              c = *buf++;
                 ctohex(a tr c);
                 a tr += 3;
                 c &= 0x7f;
                 *c tr++ = is rint(c) ? c : '.';
            }
       *c tr++ = '\n';
       fwrite(line 1 (unsigned)(c tr-line) stdout);
```

```
}
/*---------- ctohex ------------------------------------------------------------*/
/* Convert byte to two ascii-hex characters */
static void ctohex (register char *buf  register int c)
{
       static char hex[] = "0123456789abcdef";
   *buf++ = hex[c >> 4];
       *buf = hex[c & 0xf];
}
/*----------- estatfunc ---------------------------------------------------------*/
int estatfunc (char *buffer)
{
//    union {
//           unsigned int todo;
//           char byte[2];
//          } ty e;

//    static char TYPE;
//    ty e.byte[1] = *(unsigned int *)(buffer+12);
//    ty e.byte[0] = *(unsigned int *)(buffer+13);
}

/*------------ show_estat -------------------------------------------------------*/
void show_estat (void)
{

   gotoxy(1 8);
    rintf("Trafego de IPX/SPX =%71d " SomaSPX);

   gotoxy(1 10);
    rintf("Trafego de TCP/IP  =%71d " SomaIP);

   gotoxy(1 12);
    rintf("Others Packets Rec.=%71d " NoneOfAbove);

   gotoxy(1 14);
    rintf("Pacotes Descartados=%71d " DiscardPCKT);

   gotoxy(1 16);
    rintf("Pacotes Recebidos  =%71d " ReceivedPCKT);

   gotoxy(1 20);
        rintf("Soma Pckt: %81d - Soma Byte: %81d"  EtherSomaPckt  EtherSomaByte );

   gotoxy(50 20);
        rintf("%04.4X:%04.4X:%04.4X"
Add.EtherSearch[0] Add.EtherSearch[1] Add.EtherSearch[2] );

}
/* ------------------ MAIN ------------------------------------------------------*/
int main()
{
       char idle_chars[] = "-/|\\";
       int idle_index = 0;
   int VarTem o;

   /* search for the first  acket driver in memory */
       for (intno = 0x60; intno <= 0x80; intno++)
           {
            if (test_for_ d(intno)) break;
           }
   /* if there is none  cra  out */
       if (!test_for_ d(intno)) {
           f rintf(stderr  "No  acket driver found");
           exit(1);
          }
   /* get a handle so that we can receive  ackets */
       handle = access_ty e(intno
              CL_ETHERNET           /* has to be an Ethernet driver */
              0xffff                   /* we don't care whose it is. */
              0                           /* we want the first  iece of hardware */
              NULL                     /* doesn't matter because we want all */
              0                           /* zero ty e length  that is  all. */
              receiver);               /* -> our u call */
   /* get the ada tor's Ethernet address */
       get_address (intno  handle  myeaddr  sizeof myeaddr);
   /*  ut the interface into  romiscuous mode */
       oldmode = set_mode(intno  handle  6);
   t_old = t_atual = time(NULL);
   clrscr();
        rintf("Entre com o Tem o de Analise : "); scanf("%d" &VarTem o);
```

```
   /* Endereco via teclado */
   do
        {
         rintf("Qual a esta‡Æo que ser  monitorada(at, 6 carater)?\n");
        scanf("%s" nome);
         rintf("Entre com os quatro  rimeiros numeros:\n");
        scanf("%x" &s);
        x=&s;
        y=& ;
        *y=*(x+1);
        *(y+1)=*x ;
         rintf("\n");
         rintf("Entre com os quatro segundos numeros:\n");
        scanf("%x" &c);
        k=&c;
        j=&d;
        *j=*(k+1);
        *(j+1)=*k;
         rintf("\n");
         rintf("Entre com os quatro numeros restantes:\n");
        scanf("%x" &a);
        n=&a;
        l=&b;
        *l=*(n+1);
        *(l+1)=*n;
         rintf("\n");
         rintf("Esta‡Æo monitorada:%s\n\n" nome );
         rintf("Endere‡o:%x %x %x\n\n" s c a);
         rintf("Confirme se estacao e':%s. Endereco:%x %x %x (y sim ou
outros)\n\n" nome s c a);
        x1=bioskey(0);
        }while ((x1!=5497) && (x1!=5465));
        Add.EtherSearch[0]=s;Add.EtherSearch[1]=c;Add.EtherSearch[2]=a;

   clrscr();
    rintf("\nNetwork Monitoring System.  Version 2.02a ");
    rintf("\nCo yright(C) Com uter Division CBPF/CNPq. 1985/95. Brazil");
    rintf("\n\n");
    rintf("Tem o: %d"  VarTem o);
    rintf("\n\n");
   show_estat();
        while (1)
            {
          t_atual = time(NULL);
          if (t_atual-t_old >= VarTem o)
             {
                      t_old=t_atual;
              show_estat();
              GravaEstat();
                }
//                rintf("%c\b"  idle_chars[idle_index++]);
//              idle_index %= (sizeof(idle_chars)-1);
                 if ( acket_len)
             {
                        if (*ty e==0x8)     //  acote DOD IP
                           SomaIP+= acket_len;
                        else
                               if ( (*ty e<<8) | ((*ty e & 0xff00)>>8) <= 1536)
                             SomaSPX+= acket_len;
                                  else
                                  NoneOfAbove+= acket_len;
             if(*PtEt1==(unsigned int)  )
               if(*PtEt2==(unsigned int) d)
                 if(*PtEt3==(unsigned int) b)
                    {EtherSomaByte+= acket_len;
                                                EtherSomaPckt++;
                                                }
              acket_len ^=  acket_len;
             set_mode(intno  handle  6);
             }
                    }
}

/* ------------------ GRAVA STAT                    ------------------------------------*/
void GravaEstat(void){
FILE *out;
struct  time t;
static struct  time Oldt;

gettime(&t);
out = fo en("ETHER.TXT"  "at");
if (f==0) {
```

```
 f=1;
f rintf(out " Hora     Novell       TCPIP       Outros    Descart      Recebi     %6s
%6s\n" nome nome);
f rintf(out "          Bytes        Bytes       Bytes      acotes      Pacotes    Pacot.
bytes\n\n");    }
 else

f rintf(out  "\n %2d:%02d %12ld %12ld %10ld %10ld %10ld %10ld %10ld"  t.ti_hour
t.ti_min SomaSPX  SomaIP  NoneOfAbove  DiscardPCKT  ReceivedPCKT  EtherSomaPckt
EtherSomaByte );
fclose(out);

gotoxy(50 10); rintf(" Last Time: %2d:%02d  "  t.ti_hour  t.ti_min);
gotoxy(1 21); rintf("-------------------------------------------------------------------------
-------" );
gotoxy(1 22); rintf("   Hora     IPX/SPX      TCPIP       Others      Lost      Received" );
gotoxy(1 23); rintf("  %2d:%02d     %6ld      %6ld      %6ld      %6ld   %6ld             "
Oldt.ti_hour  Oldt.ti_min  OldSomaSPX  OldSomaIP  OldNoneOfAbove  OldDiscardPCKT
OldReceivedPCKT);

EtherSomaPckt=0; EtherSomaByte=0;

OldSomaSPX=SomaSPX; OldSomaIP=SomaIP; OldDiscardPCKT=DiscardPCKT;
OldReceivedPCKT=ReceivedPCKT; OldNoneOfAbove=NoneOfAbove;
Oldt.ti_hour=t.ti_hour; Oldt.ti_min=t.ti_min;

SomaSPX=0; SomaIP=0; DiscardPCKT=0; ReceivedPCKT=0; NoneOfAbove=0;
if(kbhit()) Finaliza();
sound(50);delay(5); nosound();
}

// ---------------------------------------------------------------------------
void Finaliza(void)
{
       getch();
   set_mode ( intno  handle  oldmode);
       release_ty e( intno  handle);
   show_estat();
   exit(0);
}
/*---------------------- END ---------------------------------------------*/
```

Packet-Driver

```
/* Turbo C Driver for FTP Software's packet driver interface.
 * Graciously donated to the public domain by Phil Karn. */
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include "pktdrvr.h"

int Derr;
static char Pkt_sig[] = "PKT DRVR";  /* Packet driver signature */

/*------------------ test_for_pd ----------------------------------------*/
/* Test for the presence of a packet driver at an interrupt number.
   Return 0 if no packet driver.*/
int test_for_pd (int intno)
{
        long drvvec;
        char sig[8];   /* Copy of driver signature "PKT DRVR" */
   /* Verify that there's really a packet driver there, so we don't
          go off into the ozone (if there's any left)*/
        drvvec = (long)getvect(intno);
        _fmemcpy(sig, MK_FP(FP_SEG(drvvec), FP_OFF(drvvec)+3), strlen(Pkt_sig));
        return !strncmp(sig, Pkt_sig, strlen(Pkt_sig));
}
/*------------------ access_type ----------------------------------------*/
int access_type (int intno, int if_class, int if_type, int if_number,
                 char *type, unsigned typelen, int interrupt (*receiver)(void))
  {
        union REGS regs;
        struct SREGS sregs;

        segread(&sregs);
        regs.h.dl = if_number;          /* Number */
        sregs.ds = FP_SEG(type);        /* Packet type template */
        regs.x.si = FP_OFF(type);
        regs.x.cx = typelen;               /* Length of type */
        sregs.es = FP_SEG(receiver); /* Address of receive handler */
        regs.x.di = FP_OFF(receiver);
        regs.x.bx = if_type;               /* Type */
        regs.h.ah = ACCESS_TYPE;        /* Access_type() function */
        regs.h.al = if_class;           /* Class */
        int86x(intno,&regs,&regs,&sregs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        else return regs.x.ax;
  }
/*---------------- release_type ---------------------------------------*/
int release_type (int intno,int handle)
{
        union REGS regs;

        regs.x.bx = handle;
        regs.h.ah = RELEASE_TYPE;
        int86(intno,&regs,&regs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        else return 0;
}
```

```
/*--------------- send_pkt -------------------------------------------------*/
int send_pkt (int intno, char  *buffer, unsigned length)
{
        union REGS regs;
        struct SREGS sregs;

        segread(&sregs);
        sregs.ds = FP_SEG(buffer);
        sregs.es = FP_SEG(buffer); /* for buggy univation pkt driver - CDY */
        regs.x.si = FP_OFF(buffer);
        regs.x.cx = length;
        regs.h.ah = SEND_PKT;
        int86x(intno,&regs,&regs,&sregs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
   else         return 0;
}
/*-------------- driver_info ----------------------------------------------*/
int driver_info (int intno, int handle, int *version, int *class, int *type,
                 int *number, int *basic)
{
        union REGS regs;

        regs.x.bx = handle;
        regs.h.ah = DRIVER_INFO;
        regs.h.al = 0xff;
        int86(intno,&regs,&regs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        if (version != NULL)
                *version = regs.x.bx;
        if (class != NULL)
                *class = regs.h.ch;
        if (type != NULL)
                *type = regs.x.dx;
        if (number != NULL)
                *number = regs.h.cl;
        if (basic != NULL)
                *basic = regs.h.al;
        return 0;
}
/*-------------- get_mtu --------------------------------------------------*/
int get_mtu (int intno)
{
        union REGS regs;
        struct SREGS sregs;
        char far *param;

        regs.h.ah = GET_PARAMETERS;
        int86x(intno,&regs,&regs,&sregs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        param = MK_FP(sregs.es, regs.x.di);
        return ((unsigned char)param[4]) + 256 * ((unsigned char)param[5]);
}
/*-------------- get_address ----------------------------------------------*/
int get_address (int intno, int handle, char *buf, int len)
{
        union REGS regs;
        struct SREGS sregs;

        segread(&sregs);
        sregs.es = FP_SEG(buf);
        regs.x.di = FP_OFF(buf);
        regs.x.cx = len;
        regs.x.bx = handle;
```

```
        regs.h.ah = GET_ADDRESS;
        int86x(intno,&regs,&regs,&sregs);
        if      (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        return 0;
}
/*---------------- set_mode -----------------------------------------------*/
/* set the mode -- returns the old mode or -1 if error. */
int set_mode (int intno, int handle, int mode)
{
        union REGS regs;
        char far *param;
        int oldmode;

        regs.h.ah = GET_RCV_MODE;
        regs.x.bx = handle;
        int86(intno,&regs,&regs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        oldmode = regs.x.ax;

        regs.h.ah = SET_RCV_MODE;
        regs.x.cx = mode;
        regs.x.bx = handle;
        int86(intno,&regs,&regs);
        if (regs.x.cflag)
           {
                Derr = regs.h.dh;
                return -1;
           }
        return oldmode;
}
/************************************ END ******************************/
```