CRIS AMON CAMINHA DA ROCHA

Simple Network Programming Interface

SNPI

ORIENTADOR: PROF. DR. JOSÉ NEUMAN DE SOUZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

FORTALEZA, FEVEREIRO DE 2002

DEDICO ESTA DISSERTAÇÃO A DEUS QUE SEMPRE ILUMINOU MEUS PENSAMENTOS E FOI FONTE DE INSPIRAÇÃO NA ELABORAÇÃO DESTE TRABALHO.

E TAMBÉM À MINHA FAMÍLIA QUE ME INCENTIVOU E MOTIVOU NOS MOMENTOS QUE MAIS PRECISEI.

Agradecimentos

- Ao **Prof. José Neuman de Souza** que foi peça fundamental na elaboração desta dissertação, sempre me guiando com sua experiência e conhecimento.
- Ao amigo **Aécio Paiva Braga** que juntamente com o **Prof. José Neuman de Souza** iniciaram o projeto que resultou nesta dissertação, e por sua ajuda no processo de correção dos diversos trabalhos aceitos em renomados congressos da área.
- À minha namorada **Ana Clarisse Martins Matos**, pelos preciosos momentos cedidos em prol desta dissertação e pelo apoio incondicional ao meu esforço.
- Ao amigo **Rogério Cysne Araújo** e **Marcelo de Alcântara Bezerra** pelos momentos de alegria e pelos conselhos valiosos antes e durante o processo de elaboração deste trabalho.
- Ao amigo **Antônio de Barros Serra** que na sua empreitada rumo ao Doutorado sempre me incentivou para a finalização deste trabalho.
- À equipe do Ponto de Presença da RNP no Ceará (POP-CE) que, de diversas formas, ajudou substancialmente nas minhas tarefas. Onde não posso esquecer as figuras de **Wellington Albano**, **Adriana Myrley**, **Marcos Frota**, **Tiago Macambira**, **Michelly Lima** e, em especial, **Daniel Fernandes** do qual fui co-orientador em um dos seus projetos de iniciação à pesquisa.
- Aos companheiros da **Turma de 2000 de Mestrado em Ciência da Computação da UFC**, pelas horas gastas em conjunto em árduos estudos e pelos momentos de alegria e prazer durante este tempo de convivência

➠ Aos amigos da **Turma de 1996 de Graduação em Ciência da Computação da UFC**, pelos momentos inesquecíveis de confraternização e divertimento e que continuemos com essa união, ainda hoje persistente, mesmo com os diferentes destinos reservados a cada um de nós.

➠ À **FUNCAP** pelo fomento disponibilizado em meu benefício e que nunca deixou de ser pago nem mesmo atrasou durante o tempo que fui bolsista deste órgão. Mais uma vez meus mais sinceros agradecimentos.

➠ Ao **CNPq** pelo apoio financeiro cedido ao Prof. José Neuman de Souza para que o mesmo pudesse apresentar um artigo relacionado a esta dissertação no IFIP/IEEE ICT 2001 realizado em Bucareste, Romênia.

➠ À **ACM** por ter pago meus gastos com transporte áereo e hospedagem na minha ida a San Jose, Costa Rica onde aconteceu o I ACM SIGCOMM LA, *workshop* onde apresentei um artigo também relacionado a esta dissertação.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

- **ACM**: *Association for Computer Machinery*
- **AMD**: *Advanced Micro Devices*
- **ANEP**: *Active Network Encapsulation Protocol*
- **ANTS**: *Active Node Transfer System*
- **ATM**: *Assynchronous Transfer Mode*
- **ARPA**: *Advanced Research Project Agency*
- **ARPANet**: *ARPA Network*
- **BBN**: *Bolt, Beranek and Newman*[1]
- **BPF**: *BSD Packet Filter*
- **BSD**: *Berkeley Software Distribution*
- **CNPq**: *Conselho Nacional de Desenvolvimento Científico e Tecnológico*
- **CMU**: *Carnegie Mellon University*
- **CSPF**: *CMU/Stanford Packet Filter*
- **DARPA**: *Defense Advanced Research Project Agency*
- **DCA**: *Defense Communications Agency*
- **DDoS**: *Distributed Denial of Service*

[1] Os nomes do três professores fundadores da *BBN Technologies*.

- **DiffServ**: *Differentiated Services*
- **DoD**: *Department of Defense*
- **EBONE**: *European Backbone*
- **EuropaNET**: *European Network*
- **FUNCAP**: *Fundação Cearense de Amparo à Pesquisa*
- **GCC**: *GNU Compiler Collection*
- **IBM**: *International Business Machines*
- **IEEE**: *Institute of Electrical and Electronics Engineers*
- **IntServ**: *Integrated Services*
- **IP**: *Internet Protocol*
- **IPv4**: *Internet Protocol version 4*
- **IPv6**: *Internet Protocol version 6*
- **IPX**: *Internet Packet eXchange*
- **ISO**: *International Standards Organization*
- **JVM**: *Java Virtual Machine*
- **KB**: *KiloBytes*
- **MB**: *MegaBytes*
- **MIT**: *Massachusetts Institute of Technology*

- **MPLS**: *MultiProtocol Label Switching*
- **NIT**: *Network Interface Tap*
- **NSFNet**: *National Science Foundation Network*
- **OSI**: *Open System Interconnection*
- **PC**: *Personal Computer*
- **PDF**: *Portable Document Format*
- **PLAN**: *Packet Language for Active Networks*
- **PS**: *PostScript*
- **QoS**: *Quality of Service*
- **RFC**: *Request for Comments*
- **RSA**: *Rivest, Shamir and Adelman*[2]
- **RSVP**: *Resource reSerVation Protocol*
- **RTP**: *Real-time Transport Protocol*
- **RTSP**: *Real-Time Stream Protocol*
- **SNPI**: *Simple Network Programming Interface*
- **SPX**: *Sequenced Packet eXchange*
- **SSL**: *Socket Security Layer*
- **TCP**: *Transmission Control Protocol*

[2]Os nomes dos autores deste método de criptografia assimétrica.

- **TTCP**: *Test TCP*
- **UFC**: *Universidade Federal do Ceará*
- **VoD**: *Video on Demand*
- **VoIP**: *Voice over IP*
- **XRM**: *eXtended Reference Model*
- **Windows XP**: *Windows eXPerience*

Resumo

Atualmente, a comunidade de redes de computadores atravessa novos desafios relacionados a redes ativas e programáveis. Desde o surgimento do conceito de redes programáveis que grandes esforços têm sido realizados para tornar as redes de computadores atuais em redes mais dinâmicas. Infelizmente, tais esforços estão acompanhados de complexidade e, portanto, o processo de dinamização está sendo comprometido. É com o objetivo de tornar o processo mais simples, que este trabalho propõe uma abordagem para a criação de nós programáveis que torna as redes mais dinâmicas sem acrescentar grau de complexidade a este processo. Esta abordagem apresenta o conceito de SNPI que consiste num conjunto de estruturas e artifícios que possibilitam aos programadores e desenvolvedores de novos serviços o total acesso às unidades de dados, alterando a máquina de estados do fluxo destas unidades para atender as necessidades do desenvolvedor.

Abstract

Nowadays, we are facing in the computer network research community new challenges related to the concepts of programmable and active networks. Great efforts have been spent to make current data communication networks more flexible and dynamic. Unfortunately, such efforts are followed by complexity and, therefore, the process of deploying the mechanisms has been committed. In order to make this process simpler, this work presents an open approach for having programmable nodes that make computer networks more dynamic in terms of adding new services without increasing complexity. This approach presents SNPI, which is a group of skeletons and data structures resources that make possible to the programmers and developers a fast deployment of new services into the network. The mechanism presented allows the full access to protocol data units in the network nodes.

1 Introdução

O cenário atual das redes de comunicação de dados[3] pode ser analisado satisfatoriamente quando observado o desempenho de serviços que não consomem muitos recursos. Desta forma, aplicações de transferência de arquivos não volumosos, o envio e recebimento de mensagens eletrônicas, a visualização de páginas pessoais cujo conteúdo é formado na sua maior parte por texto/imagens e quaisquer outros serviços caracterizados por um fluxo diminuto e de comportamento não contínuo, os chamados fluxos em rajadas, são bem suportados pelos elementos que caracterizam estas redes e também são bem tolerados pelos próprios usuários, os quais não se sentem incomodados com uma pequena demora na realização destes serviços.

Infelizmente, os serviços citados acima não conseguem satisfazer o ego do ser humano que cada vez mais procura atividades que satisfaçam suas necessidades. Pensando nisso, empresas e grupos de pesquisa do mundo inteiro buscam dia-a-dia novas tecnologias que melhorem a transmissão de dados, fazendo com que um maior volume de dados seja transferido entre dois pontos no mesmo espaço de tempo. Algumas destas tecnologias, como a famosa fibra ótica, já estão incorporadas ao cotidiano das redes de computadores garantindo taxas maiores de transferência e um melhor provimento dos diferentes serviços.

Acontece que estas tecnologias, assim como tudo que conhecemos, possuem um limite que não está muito longe de ser alcançado. Sendo assim, torna-se cada vez mais difícil criar novas tecnologias que possibilitem taxas maiores de transferên-

[3]Durante este trabalho, os termos “redes de computadores” e “redes de comunicação de dados” serão usados para representar estruturas de comunicação semelhantes.

cia de dados. Para agravar a situação as redes de alcance global, mais exatamente a Internet, ganham maiores proporções a cada dia que passa interconectando um número cada vez maior de pessoas que buscam se beneficiar com as mesmas. Sem mencionar nos dispositivos emergentes no mercado como telefones celulares e computadores portáteis (*palmtops* e *handhelds*) que permitem o acesso à Internet, tornando ainda mais delicada a sobrevivência dessa grande rede.

Devido aos diversos fatores mencionados acima e muitos outros, pesquisadores chegaram à conclusão que é momento de investir menos na criação de novas tecnologias e passar a otimizar os recursos já existentes. Analisar os serviços a serem prestados, observando quais os recursos que mais podem denegrir tais serviços e realizar alocações prévias destes recursos, mesmo que para isso passasse a haver cobrança pelo serviço e não mais pelo simples uso da rede, parece ser a solução mais adequada. Tais estudos passaram a figurar uma nova área em redes de comunicação de dados, chamada de Qualidade de Serviço, ou simplesmente QoS.

Além das pesquisas em QoS, outras medidas estão sendo levadas em consideração na tentativa de alcançar uma otimização dos recursos. Medidas como o surgimento de novos protocolos buscando substituir protocolos considerados ineficientes e ultrapassados, a incorporação de novos mecanismos de segurança à rede que suportem a transferência de dados confidenciais e a implantação de soluções específicas e personalizadas para os diferentes problemas que podem surgir são alguns poucos exemplos dos muitos que surgem a todo momento.

À medida que as soluções referenciadas acima foram sendo desenvolvidas um problema comum a todas elas começou a surgir. Como as redes de computadores atuais possuem comportamento puramente estático, ou seja, comportam-se co-

mo um conjunto de máquinas autônomas que são inteconectadas por um meio de comunicação e que possuem comportamento definido[4], não era possível implantar tais soluções sem substituir boa parte dos equipamentos que compunham tais redes.

Em frente a tal problema, a comunidade científica se organizou a passou a trabalhar em cima de equipamentos e propostas que pudessem tornar as redes atuais em redes com características dinâmicas, onde seria possível implantar novos serviços a estas redes sem haver necessidade de troca dos equipamentos. Como decorrência de tais esforços, já começam a surgir algumas propostas que alcançam bons níveis de dinamismo e, por isso, já começam a passar por experimentos acadêmicos na tentativa de comprovar suas vantagens.

É no cenário ilustrado acima onde esta dissertação apresenta SNPI, que consiste de uma abordagem para a criação, em elementos considerados estáticos, de uma interface simples de programação por onde os desenvolvedores podem observar e modificar as unidades de dados que fluem pelos respectivos equipamentos, possibilitando assim a implantação de novos serviços. Embora tenha uma fundamentação bastante simples, SNPI possui alto nível de funcionalidade permitindo a realização de diversas operações nestas unidades de dados, fato este que até bem pouco tempo não era possível nas redes de comunicação de dados tradicionais.

Esta dissertação é composta de oito capítulos, incluindo este capítulo introdutório, e mais dois anexos que documentam por completo este trabalho. Abaixo se encontra uma breve descrição dos diferentes capítulos e anexos.

[4]O termo “definido” diz respeito à impossibilidade de haver interferência, por parte dos administradores de rede, nos mecanimos internos de seus equipamentos.

- No capítulo 2 será discutido um histórico das redes de comunicação de dados tradicionais, intencionando uma figuração do cenário que motivou a criação e realização desta dissertação. Ênfase maior será dada à rede Internet e à arquitetura TCP/IP devido à disseminação e popularidade de ambas.

- Vistos os aspectos históricos, incluindo uma breve discussão sobre os problemas que as redes atravessam na atualidade, o capítulo 3 apresenta a motivação desta dissertação, acrescido de uma introdução ao estado da arte em redes programáveis e redes ativas.

- O capítulo 4 descreve, de maneira superficial e lacônica, os trabalhos relacionados à dissertação mostrando as contribuições que cada trabalho apresenta para a área de redes programáveis e redes ativas.

- SNPI é apresentada e detalhada no capítulo 5. É apresentado o modelo de uma forma genérica sem se ater a nenhuma arquitetura de rede em especial. Com isso, a proposta pode ser aplicada a qualquer arquitetura de rede que satisfaça os pré-requisitos mencionados neste mesmo capítulo.

- Já no capítulo 6, é mostrada uma implementação da abordagem SNPI para a pilha TCP/IP do sistema operacional Linux. Tal implementação mostra que a proposta, além de simples e funcional, pode ser colocada em prática nos sistemas operacionais utilizados na grande maioria das redes de comunicação de dados.

- O capítulo 7 apresenta um aplicativo que utiliza a interface SNPI para o controle de datagramas IP. Será verificado neste capítulo, quão é funcional a abordagem apresentada nos dois capítulos anteriores.

- O Capítulo 8 encerra a parte prinicipal da dissertação mostrando algumas considerações finais e apresentando uma série de trabalhos futuros que podem ser desenvolvidos e que também podem servir de motivação para outros acadêmicos da área.

- O primeiro anexo consta de uma pequena comparação entre a abordagem aqui apresentada, SNPI, e o BSD *Packet Filter*. Tal comparação é importante para tirar algumas dúvidas que o leitor pode ter sobre a semelhança entre estas duas abordagens[5].

- O segundo anexo finaliza a dissertação listando o código fonte da implementação da interface SNPI para Linux, como também o código fonte do aplicativo NetActeon. Por se tratar de um trabalho científico, o autor entende que a contribuição precisa ser documentada por completo.

[5]Lembrando que SNPI é uma proposta desenvolvida dentro da filosofia de redes programáveis, enquanto BSD *Packet Filter* não possui nenhuma intenção em adicionar programabilidade a uma rede de computadores. Tal comparação a ser feita é extremamente necessária pelo motivo das duas abordagens possuírem semelhanças na concepção que podem levar ao surgimento de questionamentos.

2 Visão Geral

A década de 50 pode ser considerada um marco na história da humanidade. Máquinas de dimensões gigantescas começavam a tomar espaço em grandes corporações e instituições, foi quando o ser humano descobriu que boa parte de seu trabalho poderia ser facilitado e agilizado com o uso destas máquinas. Cálculos que, outrora, demoravam dias para serem realizados por um equipe de profissionais, agora eram calculados num tempo nunca imaginado. Tais máquinas, chamadas de computadores, revolucionaram o mercado financeiro e comercial deixando muitos críticos arrependidos por terem desacreditado nestas máquinas ainda na década de 40, época do surgimento dos primeiros computadores ainda em caráter experimental e acadêmico.

O tempo foi passando e a cada década que se extinguia, computadores mais velozes, de tamanho mais reduzido e com preços bem mais acessíveis iam se multiplicando nas médias e grandes empresas. Nunca na história, o ser humano presenciou uma evolução tão acelerada e benéfica no campo da tecnologia como nestes últimos 50 anos. Tecnologias como circuitos integrados, novos materias supercondutores e novos meios de armazenamento de dados revolucionaram o trabalho humano gerando muitas discussões sobre este assunto que era, e ainda é, visto por muitos como uma verdadeira enfermidade para a sociedade. Evitando maiores discussões de caráter filosófico, chegamos finalmente à disseminação dos computadores em pequenas empresas, os quais passaram a ser objetos corriqueiros e essenciais dentro das mesmas.

Embora facilitasse e otimizasse o cotidiano de empresas e pessoas, o computador em si não era capaz de resolver sérios problemas que o homem enfrenta há tem-

pos, como a organização de informações. Um computador de forma isolada podia organizar seus dados localmente mas não conseguia interagir com outros computadores da mesma empresa, ou até do mesmo departamento, por falta de um meio e de um padrão de comunicação que os levariam a um cenário de compartilhamento de dados e dispositivos. Foi quando consolidou-se a tecnologia de redes de computadores que já tinha dado seus passos ainda na época dos *mainframes*[6]. Com esta tecnologia presente nas empresas, era possível o compartilhamento dos dados de forma organizada e até mesmo uma interação entre os diversos computadores para alcançar determinados objetivos. É chegado o momento de definirmos o que realmente vem a ser uma rede de computadores.

> "Uma rede de computadores consiste de uma coleção de computadores autônomos[7] interconectados. Dois computadores estão interconectados se os mesmos são capazes de trocar informações. "[8] ([34]; pg. 2)

Imediatamente, o ser humano percebeu que as redes de computadores poderiam ser utilizadas para outros fins que não somente o compartilhamento de dados e recursos, pois permitiam a comunicação entre duas pessoas de forma quase imediata através de aplicações simples de transferência de mensagens. Este fator,

[6] O cenário de terminais, os chamados terminais burros, conectados a um *mainframe* não pode ser considerado uma rede de computadores, pois não cabe na definição dada neste trabalho. Mas a tecnologia empregada neste cenário de comunicação foi fundamental para o aparecimento das primeiras redes.

[7] O termo autônomo é usado para caracterizar dispositivos que consigam realizar processamento de maneira independente de qualquer outro dispositivo, portanto, excluindo da definição os cenários que configuram claramente uma relação mestre/escravo como o cenário de *mainframe* com o uso de terminais.

[8] *Computer Network means an interconnected collection of autonomous computers. Two computers are said to be interconnected if they are able to exchange information.*

aliado ao compartilhamento de dados e outros recursos, foi a força motriz que impulsionou o crescimento e desenvolvimento das redes. As pessoas passaram a se comunicar dentro das empresas mesmo que estivessem separadas fisicamente, o que ocasionou uma satisfação do ser humano.

Na década de 80, aconteceu um dos grandes passos no desenvolvimento dos computadores e das redes, a grande corporação IBM decidiu abrir seu projeto de computador pessoal, o famoso PC, para qualquer empresa que estivesse disposta a fabricar computadores de baixo custo mas com um poder de processamento compatível para com as necessidades de mercados pequenos ou interesses pessoais. Com esta estratégia de mercado audaciosa, os computadores pessoais começaram a tomar espaço nos lares de todo o mundo de forma similar à televisão mas com consequências bem mais profundas. Foi a popularização dos computadores, processo esse que acontece cada vez mais de forma acelerada e surpreendente e que promete revolucionar a sociedade.

Com esta popularização dos computadores, as redes começaram a crescer exponencialmente dentro de pequenas empresas e eram itens indispensáveis dentro de médias e grandes empresas. Naquela época, vários padrões de comunicação e diferentes tecnologias eram utilizados na montagem de tais redes, onde os principais foram Ethernet e Token Ring como tecnologias de comunicação e IPX/SPX como padrão de comunicação.

Mas a disseminação destas redes locais por si só não era suficiente para alcançar o cenário atual. Algo que interconectasse todos estes computadores de uma forma global, permitindo a criação de um mundo virtual em que as pessoas pudessem se comunicar não só mais dentro das próprias empresas mas também entre diferentes

sociedades e países, estava por acontecer. Foi, então, que surgiu a rede global que hoje chamamos de Internet. Seu surgimento, inesperado por muitos mas um passo inevitável, foi uma verdadeira revolução que desestruturou não somente os mecanismos de automação mas, também, o comportamento do ser humano.

Seriam necessárias dezenas de páginas deste trabalho para relatar o surgimento da Internet e todas as particularidades que existiram durante sua concepção. Mas, ao mesmo tempo, é importante para o leitor e para o entendimento completo deste trabalho um breve relato desse fenômeno. Isso porque os esforços atuais se concentram na tentativa de resolver os problemas da Internet e de sua arquitetura, o TCP/IP.

A Internet foi fruto de uma rede de caráter militar, criada pela ARPA[9] para estudos de novas tecnologias que pudessem ser utilizadas em possíveis guerras que os Estados Unidos viessem a travar contra seus inimigos. Naquele momento ficou decidido que a tecnologia empregada na construção desta rede, batizada de ARPANet, seria a comutação de pacotes, tecnologia esta que tinha sido criada ainda na década de 40 e que tinha pouca aplicação prática. A ARPANet, inicialmente, interligou alguns órgãos militares dos Estados Unidos e alguns institutos de pesquisa. O padrão de comunicação utilizado no início foi um conjunto de protocolos mal projetados e que pouco se preocupavam com a otimização da comunicação e com a heterogeneidade dos sistemas que compunham a ARPANet, sendo utilizado somente para comprovar o funcionamento da mesma e suas vantagens frente a uma rede de comunicação que utilizasse a tecnologia de comutação de circuitos.

Com o decorrer do tempo, a ARPANet foi tomando maiores proporções e suas ramificações passaram a abraçar outras instituições de pesquisa. Neste momento,

[9] A ARPA é um órgão de pesquisa oriundo e subordinado ao DoD.

sentiu-se a necessidade de criação de um padrão de comunicação mais robusto, ou seja, uma arquitetura de comunicação mais trabalhada e que pudesse interconectar os diferentes sistemas, não importando a tecnologia de comunicação que estivessem utilizando localmente, e que tratasse de forma mais condizente alguns problemas como a corretude dos dados e o processo de roteamento. Tal esforço originou uma arquitetura chamada de TCP/IP[10], a qual passou a ser incorporada ao BSD UNIX com o objetivo de disseminar este novo conjunto de protocolos por toda a ARPANet.

Com o passar do tempo, a ARPANet foi tomando uma esfera intercontinental e passou a servir de modelo para outras redes como a NSFNet, EuropaNet e EBONE. Com esta banalização da ARPANet e do TCP/IP que foi oficialmente liberado em 1983, o DCA que agora administrava a ARPANet decidiu isolar uma parte da rede numa porção estritamente militar e que originou a MILNet. A porção restante da rede, ainda chamada de ARPANet, continuou seu desenvolvimento acelerado juntamente com outras redes que utilizavam o mesmo conjunto de protocolos, ou seja, o TCP/IP. Todo este processo culminou com a interligação destas diferentes redes que utilizavam a arquitetura TCP/IP e a partir do momento que não foi mais possível distinguir os limites destas redes regionais ou não era mais necessário tal distinção, foi quando passou-se a usar o termo Internet para referenciar esse conjunto e, desde então, a Internet só conheceu desenvolvimento e crescimento acelerados.

É importante salientar que a Internet trouxe avanços e mudanças em todas as áreas da sociedade. Com tanto poderio, não demorou muito para que esta grande e

[10]Referência aos dois principais protocolos da arquitetura, o TCP e o IP. Tal arquitetura teve sua concepção ainda no ano de 1974.

promissora rede de computadores caísse na graça de elementos chaves do mundo capitalista. Não foi difícil perceber que a Internet é a ferramenta da próxima revolução social e econômica sendo, portanto, alvo de investimentos nunca vistos antes.

Tal desenvolvimento possibilitou um cenário onde a maioria das atividades, antes cumpridas em largos espaços de tempo, hoje são realizadas de forma quase imediata. Tarefas estas que não dizem respeito tão somente aos mecanismos econômicos e industriais mas, também, aos mecanismos sociais. Se antes as pessoas passavam horas em filas para realizar um pagamento ou uma transferência entre contas, hoje é possível fazer a mesma tarefa através da Internet, aplicação chamada de *Home Banking* por alguns bancos e *Net Banking* por outros, e reservar mais tempo para a família ou outras atividades de lazer.

E mais, meios de comunicação e relacionamento entre pessoas estão sendo reformulados e revolucionados neste processo de integração da sociedade com a Internet. Se antes uma simples carta levava dias para chegar ao seu destino, hoje uma mensagem eletrônica (*e-mail*) leva alguns minutos ou até mesmo segundos na maioria dos casos. Hoje, a comunicação é acelerada e o ser humano não se contenta em esperar para trocar informações, seja via correio eletrônico ou outro meio qualquer.

Além da comunicação, a informação está cada vez mais próxima de qualquer um que possua um computador. Uma quantidade incomensurável de informações está disponível e se torna cada vez mais acessível e mais diversificada. Assuntos que até bem pouco tempo atrás só eram tratados em livros de difícil acesso, hoje estão espalhados na Internet criando a maior fonte de informações jamais cons-

truída pela sociedade humana. Infelizmente, este reduto tão valioso traz algumas informações que preferíamos não ver disponíveis de uma forma tão acessível para o ser humano, atiçando de forma tão direta e forte alguns instintos doentios que existem em todos nós.

Para impulsionar mais ainda este processo veio a disseminação do uso da fibra ótica como meio de comunicação para médias e longas distâncias. Isso possibilitou um aumento na velocidade de transmissão e um fluxo muito maior de informações fazendo com que as pessoas passassem cada vez mais a se utilizar da Internet para suas tarefas diárias e para se comunicar. Esta revolução nos meios de comunicação acabou gerando uma certa vontade pelo lado dos usuários de experimentar formas de comunicação que se aproximam mais do dia-a-dia. Só se beneficiar do processo de escrita e leitura, infelizmente, não é suficiente para satisfazer o desejo do ser humano que pede por mais interação, seja por áudio, por imagens, por vídeo ou por qualquer outro meio que possa atiçar seus diferentes sentidos.

Tais exigências causaram uma certa corrida por parte de empresas de *software* e *hardware* na busca de aplicações que as atendessem. Foram criadas aplicações de voz na Internet (VoIP), videoconferência, teleconferência, transmissão de vídeo em tempo real ou sob demanda (VoD), entre outras dezenas de aplicações que, de certa forma, alegram os usuários que agora podem, por exemplo, escutar a voz de um parente que está no outro lado do planeta ou, então, acompanhar a vida de sua família enquanto está num dia rotineiro de trabalho. Enfim, todas estas aplicações e muitas outras, juntamente com o fenômeno do comércio eletrônico (*e-commerce*), são os alvos de investimentos de empresas e instituições do mundo inteiro.

Infelizmente, a evolução destes novos meios de transação e comunicação pode ser dificultada e até mesmo desestruturada devido às bases em que foram fundamentadas a Internet. Inicialmente, esta referida rede, outrora chamada de ARPANet, não tinha objetivos globais como esses que alcançou hoje. A ARPANet, como já apresentado neste trabalho, era uma rede com fins puramente militares e que, por acaso do destino ou não, acabou sendo utilizada por diversas instituições de pesquisa e se tornou popular a partir do momento que abriu suas portas para o lado comercial. Com isso, não existe uma estrutura definida nem padrões que suportem um crescimento tão acentuado como esse que acontece hoje. A razão de tudo isto é que a Internet não possui mecanismos intrínsecos para garantir muitas das propriedades exigidas por estas novas aplicações como segurança, garantia dos serviços a serem prestados, contabilização, otimização no envio dos fluxos resultantes destas aplicações, entre outros. A Internet está tão subdimensionada em todos os sentidos que não é possível garantir, nem mesmo, que existam endereços IP para todos aqueles dispositivos que queiram fazer parte desta rede, visto que existe um problema sério de escassez de endereços que já preocupa há algum tempo mesmo contrariando a famosa frase de um dos criadores da arquitetura TCP/IP, Vint Cerf, que em 1977 imortalizou a seguinte frase:

> "Um espaço de endereçamento de 32 *bits* deve ser suficiente para a Internet."[11] - Vint Cerf.

Não existe um consenso no que diz respeito à longevidade da Internet, nem se a mesma vai conseguir suportar esta nova gama de aplicações e serviços sem degradá-los de forma substancial. Muitos investimentos e pesquisas ocorrem hoje

[11]"*32 bits should be enough address space*"

na intenção de retificar alguns problemas existentes para que a Internet não precise sofrer uma reestruturação substancial, o que seria dispendioso em termos de tempo e de dinheiro, sem falar em problemas de interoperabilidade e incompatibilidade que iriam perdurar por anos a fio podendo criar um certo desconforto para usuários, desenvolvedores e investidores.

Algumas áreas merecem destaque, pois estão sofrendo mudanças com o advento de novos mecanismos de tratamento dos diferentes fluxos para garantir uma maior robustez na Internet. É importante mencionar e discutir de forma superficial os avanços nestas áreas para que o leitor tome conhecimento das necessidades a serem discutidas posteriormente e para mostrar ao leitor a importância da contribuição a ser apresentada neste trabalho.

- Qualidade de Serviço - QoS

  Esta, com certeza, é a área que concentra a maior parte das pesquisas e investimentos em redes de computadores na atualidade, isto porque garantir a qualidade dos serviços oferecidos através da Internet resulta numa satisfação dos usuários. Para alcançar este objetivo, que parece ser tão simples, existe um conjunto de abordagens que procuram monitorar e controlar as unidades de dados que atravessam uma rede de computadores e as diferentes variáveis que influenciam neste fluxo. Infelizmente, não é fácil alcançar um cenário que consiga garantir certas propriedades necessárias para certos tipos de comunicação.

  Para citar um exemplo podemos analisar o fluxo de dados de uma operação de transferência de arquivos e de uma transmissão de vídeo em tempo real. No primeiro cenário, sendo razoavelmente flexível no que diz respeito

aos requisitos mínimos, não há exigências quanto ao atraso mínimo suportado nem mesmo referente à taxa de transferência, a única propriedade que parece ser irrevogável é a corretude no tratamento final dos dados. Já no segundo cenário, tudo muda de figura e podemos citar uma série de exigências como atraso mínimo, variação mínima do atraso, taxa de transferência mínima, entre outras e que sem as mesmas não é possível garantir a satisfação do usuário que pode experimentar alguns problemas indesejáveis como a falta de sicronismo entre o áudio e a imagem, distorção da imagem, interrupção na transmissão para o processo de *buffering*, entre outros. Mas, por incrível que pareça ser, não há exigências severas quanto a corretude dos dados transmitidos, sendo tal fato corrigido por uma queda na qualidade de reprodução do vídeo e que pode muito bem ser suportada pelo usuário. Tal analogia leva à conclusão que os diferentes fluxos de dados devem se comportar de acordo com os requerimentos dos usuários, pois são eles que estão sendo servidos por estes fluxos e que podem manifestar satisfação ou descontentamento.

Mesmo existindo tanta diversidade de requerimentos por parte dos serviços oferecidos através da Internet, como visto no exemplo acima, tal fato seria facilmente contornado se não existissem problemas de ordem estrutural na Internet e nos equipamentos que a compõe. A arquitetura TCP/IP, na sua especificação original, não reza nenhum mecanismo para garantir exigências como aquelas citadas a pouco e, tal ausência, acaba por refletir nas pilhas TCP/IP implementadas para os diversos sistemas operacionais da atualidade. Além disso, e talvez como uma conseqüência da pobreza da arquitetura TCP/IP, os dispositivos que constituem a rede global de computadores

(*switches*, roteadores, estações finais e outros tipos de equipamentos) não possuem quaisquer mecanismos ou interfaces para que seja possível uma manipulação dos recursos presentes nos mesmos, manipulação esta que é fundamental para se construir abordagens capazes de garantir algumas propriedades importantes para os diferentes tipos de transmissão de dados, o que seria o começo de um cenário de comunicação com garantia da qualidade dos serviços.

Mesmo com tantos obstáculos, muitas pesquisas e esforços oriundos de diversos grupos de pesquisas já apresentam resultados e, a passos curtos mas cautelosos, estão sendo implantados dentro das possibilidades das redes de computadores atuais e do desenvolvimento da Internet, sem decisões precipitadas. ATM, por exemplo, é uma tecnologia que é muito bem fundamentada, com alta escalabilidade e que resolveria, em parte, o problema de controle dos diferentes elementos que participam do processo de transferência de dados, uma vez que equipamentos ATM possuem meios que permitem o controle quase total destes elementos através de controle das filas de espera (*Buffers*), configuração das taxas de transferência e outros. Uma rede de computadores baseada em tecnologia ATM se figura mais controlável e robusta do que qualquer outro cenário da atualidade, abrindo o caminho para a criação e implantação de abordagens que garantam a qualidade dos serviços.

Além das mudanças necessárias a nível de *hardware*, não pode-se esquecer que também são necessárias mudanças substanciais a nível lógico, ou seja, mudanças a nível dos protocolos que governam as regras de comunicação na Internet. Mesmo criando equipamentos que possuam mecanismos que

possibilitem a manipulação completa de todos os seus elementos, enquanto a Internet tiver uma curva de crescimento acentuada sempre existirá mais demanda do que recursos disponíveis e, por tal motivo, os protocolos precisam saber tratar estes casos negando o estabelecimento de novas conexões em prol da qualidade dos serviços que estão sendo oferecidos no momento ou iniciar um processo de renegociação dos recursos que estão sendo utilizados para que possa acatar o novo pedido de estabelecimento de conexão. No universo das muitas abordagens que procuram tratar este problema podemos destacar algumas como RSVP, DiffServ, IntServ e MPLS como sendo as mais promissoras. Aliás, é importante salientar que não é uma destas abordagens que deve garantir um cenário de qualidade de serviço mas a união de duas ou mais para alcançar um controle total da qualidade.

- Segurança

  Este campo é o mais enigmático e o que causa maior desconforto para desenvolvedores e usuários da Internet, já que existem um grande número de relatos e fatos que comprovam insegurança da rede onde as conseqüências foram desastrosas para os usuários que tiveram sua privacidade violada. Tal insegurança é facilmente percebida ao analisar a arquitetura TCP/IP que se mostra carente de mecanismos para garantir a confidencialidade dos dados. Além da confidencialidade, outros aspectos como autenticidade e o controle sobre os fluxos são problemas graves e que não podem passar desapercebidos.

  Estes relatos e fatos mencionados acima são verdadeiros obstáculos para o crescimento e popularização do comércio eletrônico (*e-commerce*) e outras atividades que necessitem de garantia no envio dos dados e, também, de

autenticação dos parceiros envolvidos nestas transações. Problemas com o envio de números de cartões de crédito, senhas e informações confidenciais, além de falhas em processos de autenticação são vistos diariamente através dos veículos de comunicação deixando os usuários temerosos, e com razão, de utilizaram a Internet para realizar suas atividades financeiras.

Na tentativa de remediar alguns deste problemas surgiram diversas propostas para os diferentes problemas. Estas propostas consideradas em conjunto têm conseguido alcançar bons níveis de segurança para a realização de transações. Como exemplo, pode-se citar SSL que objetiva alcançar, principalmente, a confidencialidade dos dados através da criação de uma camada, na verdade um nível de abstração, a mais na arquitetura TCP/IP. SSL é um protocolo desenvolvido pela *Netscape Communications Corporation* que engloba um conjunto de mecanismos de segurança como criptografia assimétrica RSA de 128 *bits*, uso de certificados digitais, e outros mecanismos que, de forma conjunta, estão alcançando segurança na trasmissão das informações.

- Novos Protocolos

  Com objetivos de otimizar e/ou resolver alguns dos muitos problemas existentes na Internet, muito tem se gasto na procura por melhores algoritmos e protocolos mais eficientes. Um exemplo que é muito citado nos dias de hoje é o caso do IPv6. Este protocolo nasceu da necessidade de substituir a atual versão do protocolo IP (IPv4) por uma que atendesse melhor às exigências do cenário atual e, principalmente, para resolver o sério problema de endereçamento, tanto com relação a escassez de endereços IP como também a ausência de uma hierarquia neste endereçamento. O IPv6, felizmente, é

um protocolo criado para ser flexível o bastante para suportar a convivência com sua versão anterior, o IPv4, no mesmo ambiente de comunicação sem maiores problemas. Em outro exemplo, podemos citar o RTP e o RTSP que são protocolos que buscam atender melhor às exigência de transmissão de vídeo em tempo real.

A criação destes, entre outros, protocolos e melhorias nos algoritmos que realizam certas funções como roteamento e tratamento das unidades de dados são muito bem vindos mas sempre trazem a preocupação de como incorporá-los à já pobre e fraca arquitetura TCP/IP. Muitas vezes a solução consiste em anunciar uma nova pilha TCP/IP com os protocolos e algoritmos em questão já aplicados, já em outros casos é utilizado o conceito de pilhas duplas para manter a compatibilidade, este é o caso do IPv4/IPv6, e desta forma tem se conseguido melhorar a Internet, mesmo que de forma ineficiente e inadequada.

- Soluções Específicas

  A Internet se figura como uma das maiores revoluções de todos os tempos e, por isso, tem chamado a atenção de pessoas de todas as faixas etárias, etnias e credos. Assim como acontece com qualquer artifício que prenda a atenção das pessoas, a Internet está sendo a arma de muitas empresas e instituições na busca por novos mercados e na tentativa de aumentar seu raio de ação.

  Portanto, empresas procuram cada vez mais alcançar novas etapas deste processo evolutivo com o objetivo de superar seus concorrentes e chegar ao monopólio tão desejado. Uma das principais etapas deste processo evolutivo diz respeito à diferenciação dos serviços oferecidos. Uma empresa que busca dominar o mercado não se contenta apenas em aplicar as tecnologias

vigentes mas, principalmente, criar novas tecnologias ou novos mecanismos para complementar ou suplementar a satisfação de seus clientes e para funcionar como um diferencial buscando a conquista de novos clientes.

Para que seja possível tal diferenciação dos serviços, é necessário que a arquitetura TCP/IP possua meios para a implantação de diferentes e novas abordagens. Mas, infelizmente, estes meios não existem o que caracteriza a arquitetura TCP/IP como um objeto estático, onde uma alteração em quaisquer dos seus mecanismos de tratamento dos fluxos pode ocasionar numa incompatibilidade com o restante do cenário. Sendo assim, as empresas esperam ansiosas por soluções que tragam flexibilidade à arquitetura TCP/IP para que as mesmas possam personalizar os serviços que oferecem e, desta forma, consigam uma maior fatia de mercado com soluções específicas e únicas.

Mencionados os aspectos que mais causam interesse para a comunidade científica e para o mercado, fica claro que a Internet está procurando alcançar um estado que proporcione satisfação total para seus usuários. Estado este que suportaria segurança nas comunicações e nas transações através da Internet, garantias na qualidade dos serviços oferecidos, otimização dos diversos processos, criação de soluções específicas, entre outros aspectos.

É fácil concluir que existem sérios problemas nas atuais arquiteturas de rede, principalmente na arquitetura TCP/IP que mesmo sendo uma unanimidade neste conjunto, parece ser a mais problemática dada sua origem.

3 Motivação

Como visto no capítulo anterior que trata sobre o estado atual das redes de comunicação, principalmente a Internet, torna-se evidente os esforços da comunidade científica na procura por soluções que melhorem as transmissões de dados e o provimento dos serviços. Tais soluções estão surgindo e se mostram como verdadeiras panacéias uma vez que não existem mecanismos para a implantação das mesmas.

Tal ausência destes mecanismos é uma conseqüência da estrutura atual das redes de comunicação, que chamamos de redes tradicionais de comunicação de dados. Estas redes são compostas por elementos que possuem funções específicas e determinadas, ou seja, não conseguem desempenhar novas tarefas pois não existem meios de aplicar estas novas tarefas nestes elementos.

Como exemplo, podemos analisar o caso de um roteador. Este tipo de equipamento possui a função de rotear os datagramas entre duas redes de acordo com protocolos e padrões previamente definidos e incorporados ao equipamento. Caso seja necessário um tratamento diferenciado no fluxo dos dados, o administrador da rede vai se sentir incapaz. Uma solução seria alterar o sistema que opera o equipamento (sistema operacional) mas como tal sistema é de propriedade do fabricante, tal solução torna-se impraticável.

Portanto, para que as redes de comunicação de dados pudessem evoluir seria necessária uma mudança de boa parte dos equipamentos, principalmente roteadores, *switches* e outros tipos de *gateways.* Acontece que esta medida é, no mínimo, a menos aconselhável já que estaria prorrogando o problema, visto que num futuro

bem próximo outras tendências podem surgir e passarão pelos mesmo dilemas.

Para evitar este cenário estático das redes de comunicação, cenário este que causa mais do que um simples desconforto, a comunidade científica pesquisa formas de tornar as redes de comunicação atuais em redes dinâmicas de comunicação. Tais redes dinâmicas seriam compostas por elementos que possibilitariam, de alguma forma, a incorporação de novos serviços na rede sem haver a necessidade de troca dos mesmos. No decorrer destas pesquisas surgiram duas vertentes que defendem pontos de vista diferentes.

Uma parte dos pesquisadores defende a criação de redes ativas para se alcançar dinamismo nas redes de comunicação. Redes ativas são compostas por elementos "inteligentes" que mudam o comportamento da rede baseado nas unidades de dados que trafegam pela própria rede. Desta forma, surge uma distinção dentro do conjunto de todas as unidades de dados. Tal conjunto passa a ser formado por unidades de dados que dizem respeito, única e exclusivamente, as duas entidades participantes da comunicação e unidades de dados que dizem respeito aos elementos "inteligentes" da rede, pois nestas unidades de dados possuem informações intencionadas a mudar o comportamento da rede de alguma forma para se alcançar um objetivo ou para implantar um novo serviço. Tais unidades de dados que dizem respeito aos elementos "inteligentes" são chamadas de unidades ativas, uma vez que elas possuem capacidade de mudar o cenário de comunicação de dados.

Tal abordagem defendida pela comunidade de redes ativas é bastante complexa e possui uma série de problemas a serem resolvidos, principalmente no que diz respeito à segurança da rede. No entanto, tal abordagem, se implantada comple-

tamente, cria um cenário nunca visto antes onde os elementos da rede podem ser alterados de diversas formas e modos, alcançando assim um dinamismo que só depende das limitações dos recursos presentes na rede. Abaixo estão listados alguns problemas desta vertente:

- **Segurança** pode ser considerado um dos maiores problemas. É preciso de um plano de autenticação e autorização[12] para evitar que unidades ativas não autorizadas consigam realizar mudanças que não deveriam acontecer. Acontece que para garantir um correto processo de autenticação e autorização deve-se implantar uma série de outras medidas para evitar que ocorra alteração dos conteúdos das unidades de dados por entidades não-autorizadas. Enfim, este problema herda todas as dificuldades que são encontradas hoje no mundo das redes de comunicação e que, por muitas vezes, parecem persistir.

- Um cenário como este proposto pela comunidade de redes ativas possui níveis de **complexidade** difíceis de serem alcançados, pois é um cenário completamente novo que exigiria uma mudança na forma de trabalhar com redes de comunicação. Além disso, exige um esforço incomensurável por parte dos fabricantes de equipamentos na busca de um padrão para se alcançar um cenário único onde todos os elementos da rede uma mesma linguagem e que as unidades ativas pudessem ser entendidas por qualquer um destes elementos.

- Talvez como uma conseqüência do problema de segurança, torna-se difícil mas essencial **garantir os limites de cada rede**. Não se pode pensar num

[12] Autenticação é o processo de certificar a identidade de uma entidade. Já autorização é a verificação dos direitos de uma entidade, que de preferência já tenha sido autenticada.

cenário onde unidades ativas pertencentes a uma rede qualquer possam interferir no comportamento de outras redes sem que haja uma autorização para tal interferência. Mesmo assim, é preciso limitar a intensidade desta interferência.

Uma outra linha de pesquisadores argumenta que não é preciso tal cenário para se garantir dinamismo na rede, basta criar uma rede composta por elementos que possuam interfaces de programação para a incorporação de novos serviços ou alteração do comportamento. Tais redes, chamadas de redes programáveis, seriam bem mais palpáveis que as redes ativas mas não tão dinâmicas. Com redes programáveis os novos serviços ou os novos tratamentos no fluxo de dados seriam implantados através de uma interface de programação disponibilizada pelos dispositivos para os desenvolvedores. Com tais interfaces os desenvolvedores teriam acesso aos diferentes elementos do dispositivo e teriam controle sobre os mesmos caso tivesse direitos para tal.

A simplicidade é o ponto forte desta abordagem de redes programáveis. O processo de criar interfaces de programação que dão acesso aos diferentes elementos do dispositivo é de fácil implementação pelos fabricantes e poderia estar disponível para os desenvolvedores tão logo existisse demanda. No entanto, esta abordagem não é tão poderosa quanto a abordagem de redes ativas, pois as interfaces, de início, alteram o comportamento local de um dispositivo e não o comportamento da rede como um todo, como acontece com as unidades ativas.

Embora pareçam bem diferentes e mesmo possuindo níveis diferentes de programabilidade, é possível, em termos práticos, emular uma rede ativa através de

alguns tipos de redes programáveis e vice-versa[33][13].

A possibilidade de criar uma abordagem que permitisse o surgimento de redes de comunicação com características dinâmicas foi a motivação para o surgimento da idéia a ser apresentada no capítulo 5, bem como a elaboração desta dissertação. Outro fator incentivador para a realização deste trabalho é a contribuição que este pode adicionar a uma área tão recente e ainda em formação.

A área de redes programáveis e redes ativas, por ainda estar em plena formação, não possui definições e padronizações claras que possam ser utilizadas como fundamentação para pesquisas relacionadas. No momento, existem grupos de pesquisas trabalhando em projetos isolados onde cada um procura definir da melhor forma possível as fundamentações desta área. Portanto, durante esta dissertação procurou-se utilizar termos e definições que fossem um consenso para estes diversos grupos de pesquisa, e por esse motivo o leitor vai encontrar um pequeno número de referências dado, também, o estado da arte em redes programáveis e ativas.

Em nenhum momento desta dissertação fica explícito a que vertente, redes programáveis ou redes ativas, a abordagem apresentada aqui pertence, mesmo porque já foi mencionado que, em termos práticos, é possível criar um ambiente ativo a partir de um ambiente programável e vice-versa. No entanto, o leitor não deve demorar para concluir que a referida abordagem possui alguns aspectos de redes programáveis, uma vez que não foram apresentadas definições de entidades como unidades de dados ativas, ambientes inteligentes e outros aspectos próprios da vertente de redes ativas.

[13]Neste artigo mencionado como referência, o autor se utiliza do termo *Discrete Approach* como uma estrutura similar à redes programáveis e o termo *Integrated Approach* para referenciar redes tipicamente ativas.

É com todas estas considerações que este trabalho apresenta uma abordagem para a implantação de nós programáveis em redes de comunicação de dados tradicionais, com o objetivo de adicionar níveis de programabilidade às redes de comunicação de dados sem a necessidade de compra de novos e caros dispositivos.

4 Trabalhos Relacionados

É importante a discussão de alguns projetos na área de redes programáveis e redes ativas com o intuito de apresentar as mais diferentes metodologias aplicadas e para, de certa forma, encaixar a abordagem proposta neste trabalho no universo de tantas outras. Com tanta diversidade entre os projetos fica complexo tentar dividi-los pelas linhas de pensamento - Redes ativas e/ou Redes programáveis -, mesmo porque existem projetos que são uma mescla de características das duas vertentes. Desta forma, são apresentados projetos que têm grande significação para a comunidade científica sem, no entanto, aplicar esforços desnecessários na tentativa de classificá-los segundo sua linha de pensamento.

4.1 Smart Packets

Desenvolvido nos laboratórios da BBN(Smart Packets), o projeto **Smart Packets**[29] foi concebido para mostrar que a área de gerenciamento de redes pode ser bastante favorecida com o uso de redes ativas. Para isso, foi construído um ambiente de rede ativa robusto o bastante para que pudesse ser utilizado pelo nó de gerenciamento na tentativa de melhor realizar sua tarefa de monitoramento e controle dos nós gerenciados.

As unidades ativas, chamadas neste projeto de *smart packets*, carregam programas que são executados nos nós gerenciados para atender alguma necessidade. O primeiro problema encontrado na criação destes *smart packets* foi a fragmentação dos mesmos, o que poderia levar a necessidade de criação de um mecanismo de reagrupamento dos diferentes fragmentos para que um *smart packet* pudesse

ser executado num nó intermediário da rede, tendo em vista que na arquitetura TCP/IP o processo de reagrupamento só ocorre no nó de destino. Já outro problema surge na manutenção do estado do ambiente de execução do dispositivo caso um programa seja enviado em diferentes *smart packets*, cada um contendo uma parte do programa. Para resolver estes dois problemas ficou decidido que os programas enviados através de *smart packets* devem ser atômicos, ou seja, não podem estar divididos em diferentes unidades.

Com a decisão de construir cada programa de maneira atômica, só podendo ser enviado dentro de um único *smart packet*, foram escolhidas duas linguagens de programação a serem utilizadas na criação destas unidades ativas e no processo de execução das mesmas. Estas duas linguagens de programação, Sprocket[28] e Spanner[27], são leves e possibilitam a criação de programas para o gerenciamento de redes com o tamanho aproximado de 1KB, tamanho apropriado para os *smart packets* em redes Ethernet[9][14] para que não haja fragmentação. A figura 1[15] apresenta a arquitetura do projeto Smart Packets.

O programa de gerenciamento ou monitoramento da rede fica responsável por gerar *smart packets*, os quais são encapsulados em quadros ANEP[1], e passá-los para o processo ANEP. Esse processo, por sua vez, injeta os *smart packets* na rede de forma *end-to-end*, onde o programa vai ser executado somente no destino, ou na forma *hop-by-hop*, onde o programa vai ser executado na origem, no destino e em todos os elementos intermediários por onde passar.

Como visto na figura1, o processo ANEP possui duas responsabilidades:

[14]Tal limitação dos *smart packets* decorre do fato de que os quadros, em redes Ethernet, possuem o tamanho aproximado de 1KB.

[15]Ilustração retirada de [29] com fins puramente explicativos.

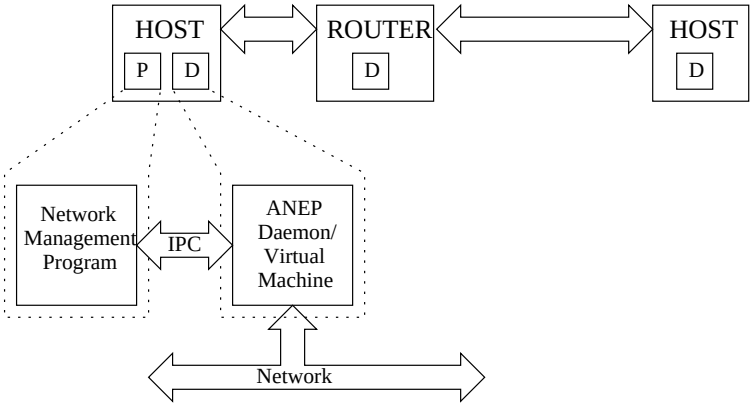

Figura 1: Arquitetura do projeto Smart Packets

- Injetar e receber os *smart packets* na/da rede.
- E fornecer um ambiente de execução para que os programas possam ser executados. Este ambiente de execução, máquina virtual, oferece uma interface segura para a execução dos programas contidos nos *smart packets*.

4.2 ANTS

Como fruto de uma tese de doutorado no MIT, o modelo **ANTS**[40] combina um modelo de programação de redes flexível baseado em cápsulas[16] e nós ativos. As entidades que fazem parte do modelo ANTS estão relacionados na figura 2[17].

As aplicações se utilizam da rede para enviar e receber pacotes especiais, chamados de cápsulas, através de seus próprios nós ativos, os quais atuam como roteadores programáveis. Cada nó ativo é conectado aos seus vizinhos por canais da camada de enlace para formar a rede como um todo. A chave para obter a

[16] Cápsulas são definidas como a especialização de agentes móveis na camada de rede.
[17] Ilustração feita com base em [40] com fins puramente explicativos.

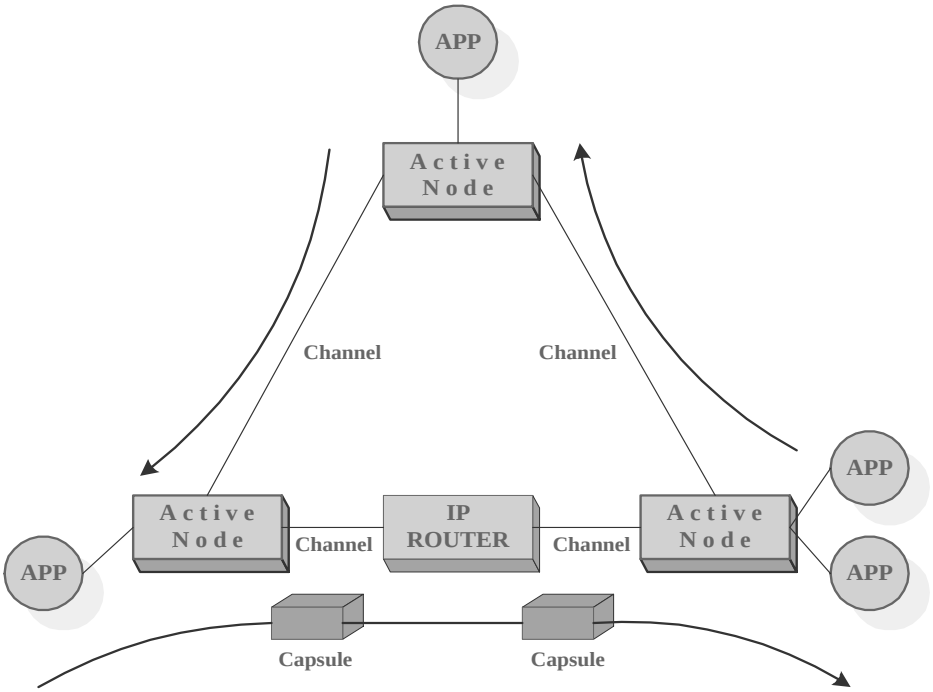

Figura 2: Entidades do modelo ANTS

implantação de serviços personalizados é a seleção de diferentes tipos de cápsulas pela aplicação final do usuário e a subseqüente interação destas cápsulas com os nós ativos dentro da infraestrutura da rede.

Assim como um agente móvel, o código carregado pelas cápsulas é executado em cada nó ativo visitado durante o percurso. Desta forma, para implantar um novo serviço é necessário somente construir um novo conjunto de cápsulas associado a um código de encaminhamento destas cápsulas que captura o processamento desejado e a interação entre as mesmas.

O modelo descrito acima foi concretizado através de um conjunto de ferramentas, ANTS *Toolkit[38],* implementadas em Java e que foram utilizadas pelos autores deste projeto para tratamento de protocolos de *Web Caching* e *Multicast*, entre

outras aplicações. Todos os elementos presentes na figura 2 foram modelados em classes Java que são executadas em JVMs presentes em cada nó ativo. Sendo assim, aplicações e serviços, para se comportarem segundo o modelo ANTS, precisam ser construídos através de classes abstratas já existentes. Este conjunto de ferramentas não foi construído como uma extensão de alguma implementação de protocolo IP já existente, mas se comporta como uma camada de rede complementar que disponibiliza todo um ambiente de construção de aplicações e serviços utilizando a filosofia do modelo ANTS.

4.3 Switchware

Switchware[3] é um projeto desenvolvido na Universidade da *Pennsylvania* que tenta balancear dois aspectos: flexibilidade de uma rede programável com requisitos de segurança da mesma. No nível do sistema operacional, um roteador IP ativo é responsável por prover uma base segura que garante a integridade do sistema. Extensões ativas podem ser dinamicamente carregadas através de um conjunto de mecanismos seguros como autenticação, verificação e encriptação.

A figura 3[18] mostra a arquitetura do projeto Switchware. São empregadas três camadas :

- **Pacotes ativos** substituem os pacotes tradicionais de rede por um programa móvel que é composto de um código ativo e de dados. A porção que representa o código ativo provê a função de controle de um cabeçalho de um pacote tradicional, acrescido de flexibilidade já que o mesmo pode interagir

[18]Ilustração retirada de [3] com fins puramente explicativos.

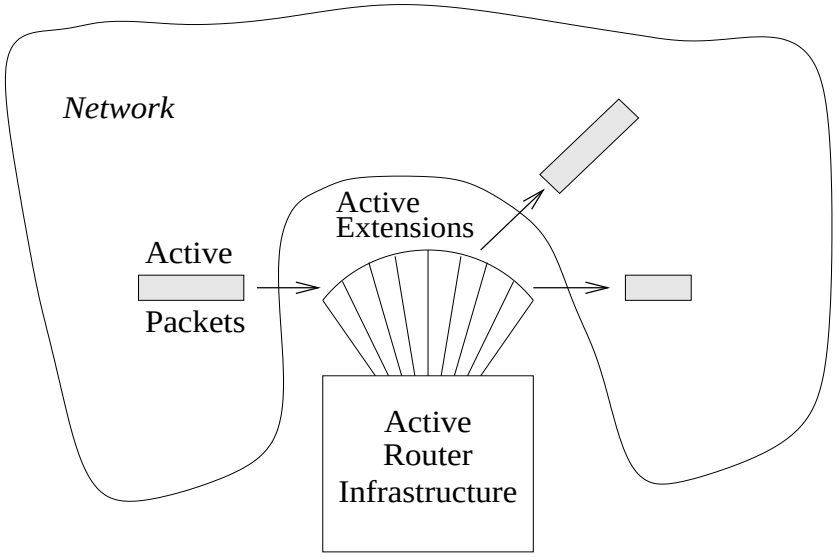

Figura 3: Arquitetura Switchware

com o ambiente do roteador de uma forma mais completa e especializada do que uma simples tabela de consulta provida pelos cabeçalhos de pacotes tradicionais. A porção de dados substitui o *payload* de pacotes tradicionais, mas provê uma estrutura flexível que pode ser utilizada pelo programa móvel. PLAN[7] é a linguagem de programação utilizada nos pacotes ativos[19]. Programa feitos em PLAN são fortemente tipados para prover segurança, sendo estaticamente verificados antes de serem injetados na rede para eliminar a possibilidade de erros de tipagem quando estiverem sendo executados nos roteadores remotos. PLAN se encaixa perfeitamente no uso em redes ativas, pois possui diferentes mecanismos para limitar os recursos utilizados por um pacote ativo.

- A camada intermediária é composta de **extensões ativas** que podem ser carregadas dinamicamente ou podem fazer parte da base de funcionalidade do

[19]Caml[5] também pode ser utilizada como linguagem de programação de pacotes ativos mas tal linguagem não possui mecanismos de limitação de recursos.

roteador. Essas extensões não são móveis e precisam se utilizar dos pacotes ativos para se comunicar com outros roteadores. Elas podem ser implementadas através de linguagens de programação de propósito geral e podem usar uma variedade de mecanismos de segurança, incluindo autenticação baseada em criptografia e verificação de programas. Tais extensões podem ser utilizadas em conjunto com pacotes ativos para a implantação de sistemas e protocolos mais complexos.

- Um **roteador ativo** forma a camada de mais baixo nível do modelo mostrado na figura 3. Enquanto as duas camadas de mais alto nível tratam a flexibilidade do sistema, o roteador ativo é inteiramente estático. O objetivo desta camada é prover uma base segura sobre a qual as outras duas camadas possam implantar seus serviços. Esta camada fornece um ambiente seguro que garante a integridade para que o sistema não seja modificado para um estado desconhecido.

Através destas três camadas citadas e comentadas acima, *Switchware* disponibiliza um ambiente inteligente e seguro por onde podem ser implantados novos serviços de forma flexível sem deixar de lado os aspectos relativos à segurança dos elementos presentes no ambiente.

4.4 XBind

O desenvolvimento de interfaces para prover acesso aberto a recursos do nó e a funções de redes ATM, se utilizando de abstrações para alcançar este objetivo, é

no que se constitui o projeto **XBind**[13], projeto que foi desenvolvido na universidade de *Columbia*. As interfaces são construídas para suportar a programabilidade dos planos de controle e gerenciamento de redes ATM. XBind é uma plataforma flexível de programação multimídia para construir, implantar e gerenciar serviços multimídia sofisticados.

XBind é baseado conceitualmente no G-Model do XRM que modela a arquitetura de comunicação da rede e as plataformas de computação multimídia. O XRM consiste de 3 componentes chamados *Broaband Network*, *Multimedia Network* e *Services and Application Network*. Tal modelo é mostrado na figura 4[20].

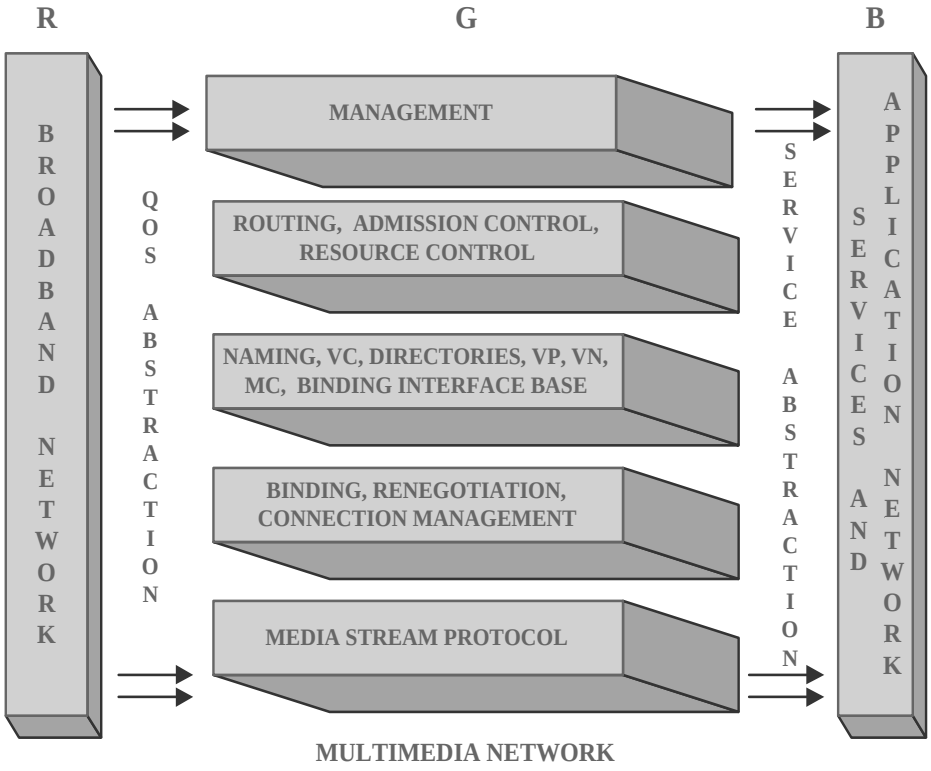

Figura 4: Modelo utilizado pelo XBind

[20]Ilustração feita com base em [13] e com fins puramente explicativos.

O *Broadband Network* é definido como a rede física que consiste de equipamentos de comutação e comunicação e dispositivos multimídia finais. Sobre esta infraestrutura reside o *Multimedia Network* cuja sua função principal é prover um suporte de *middleware* necessário para realizar serviços com garantia de QoS. Tal característica é alcançada através de um conjunto de abstrações derivado do *Broadband Network*. Estas características são disponibilizadas para o usuário final através de um conjunto de interfaces presentes no componente *Services and Application Network*.

4.5 Netscript

Netscript[41], desenvolvido na Universidade de *Columbia*, é uma linguagem de programação e um ambiente para a implementação de protocolos ou serviços. Programas *Netscripts* podem ser implantados em nós para dinamicamente estender a rede com novos protocolos e serviços. Seus programas são organizados como agentes móveis que são enviados para sistemas remotos e executados sobre controle local ou remoto. Uma rede *Netscript* consiste de uma coleção de nós onde cada um deles executa um ou mais módulos *Netscript*. Um módulo *Netscript* (*Netscript Engine*) é uma abstração de software de um dispositivo programável de processamento de pacotes de rede. Cada módulo *Netscript* consiste de componentes de tratamento do fluxo de dados, chamados de *boxes*, que processam *streams* de pacotes que fluem através deles.

O sistema *Netscript* é formado por dois componentes:

- *Netscript*, uma linguagem textual para tratamento do fluxo de dados e para

composição de protocolos de processamento de pacotes. Esta linguagem consiste de três componentes integrados chamados de:

  - *Composition language*: para composição e especificação do fluxo de dados.
  - *Presentation language*: para definição do formato dos pacotes de rede.
  - *Classification language*: para construção dos classificadores de pacotes.

- *Netscript Toolkit*, um conjunto de classes Java que abstraem os recursos de um nó da rede e utilizado para a compilação da linguagem *Netscript*.

Netscript possui um aspecto interessante que é o de procurar prover uma linguagem universal para redes programáveis de forma análoga ao *Postscript*. Da mesma forma que o *Postscript* abstrai a programabilidade dos mecanismos das impressoras, *Netscript* procura capturar a programabilidade das funções dos nós da rede.

5 Simple Network Programming Interface - SNPI

Os projetos citados no capítulo anterior possuem diferentes características com diferentes níveis de flexibilidade, mas todos eles procuram através de algum mecanismo criar poderosos ambientes que proporcionam níveis de programabilidade à rede. Infelizmente, tais projetos citados até o momento possuem dois grandes problemas que dificultam sua implantação e disseminação pelo cenário das redes de comunicação de dados.

O primeiro problema, e provavelmente o mais grave deles, é a complexidade que estes projetos imprimem na tentativa de se buscar programabilidade na rede. Iniciativas que se baseiam no uso de agentes móveis, pacotes ativos ou cápsulas, exigem uma mudança profunda na estrutura das redes de comunicação de dados como também uma mudança na forma como os serviços são requisitados. Esta abordagem de utilização de unidades ativas traz juntamente a ela outros problemas como novas políticas de segurança para evitar que estas unidades ativas interfiram em domínios não autorizados ou configurem um estado não desejável na rede; novas políticas de manipulação dos pacotes, já que agora temos dados e trechos de código trafegando no mesmo meio; uma carga maior de dados(trechos de código) trafegando pela rede; criação de ambientes de compilação e execução *run-time*, para que tais trechos de código possam tomar a forma de um novo serviço de rede; entre muitos outros problemas. Mesmo aquelas iniciativas que buscam a padronização de interfaces que abstraem recursos de *hardware* trazem problemas de complexidade pela dificuldade de abstrair e generalizar dispositivos tão complexos e distintos como roteadores, *switches*, *gateways* e outros.

Não restam dúvidas que estes níveis de complexidade apresentados por tais pro-

postas são inevitáveis e são necessários para a criação de uma rede dinâmica robusta e completa, mas esta mesma complexidade deve ser evitada inicialmente em prol da simplicidade de implantação. Com propostas mais simples de serem implantadas e mesmo sem tanto poder de programabilidade, os administradores e desenvolvedores das redes de comunicação de dados passariam a se utilizar deste novo paradigma de implantação de serviços e os mesmos passariam, em pouco tempo, a clamar por mecanismos mais completos que resultassem em níveis melhores de programabilidade, momento este em que as propostas atuais seriam exaltadas como possíveis soluções.

O segundo problema se refere a ausência de uma fundamentação nesta área de redes programáveis e ativas. Não existe, no momento, definições que possam servir de base para pesquisas e análises, o que existe são iniciativas bem estruturadas mas que não seguem padrão algum, mesmo porque não existe um. Estas iniciativas são vistas como esforços isolados e não despertam o interesse de fabricantes e empresas que poderiam ajudar no processo de implantação das mesmas. Isso cria um pessimismo quanto ao futuro das redes programáveis, pois, mesmo grandes empresas que dominam o mercado de roteadores e *switches* não investem tão alto na implantação de projetos, como aqueles citados no capítulo 4, se não houver uma padronização consistente.

É pensando nesses problemas citados acima, principalmente o primeiro deles pois o segundo necessita de um esforço conjunto da comunidade científica em busca de uma fundamentação da área, que este trabalho propõe uma abordagem para a criação de nós programáveis em redes de computadores, que possui a simplicidade de implantação como seu ponto forte e que não cria sérios limites quanto ao seu poder de transformar uma rede comum e estática, numa rede dinâmica com

a possibilidade de implantação de novos serviços. Esta abordagem é chamada de SNPI pelo fato de se comportar como uma interface entre o usuário[21] dos dispositivos da rede e o fluxo de unidades de dados que atravessam estes dispositivos, com possibilidades de manipulação dos recursos utilizados no tratamento destas unidades.

A abordagem proposta pressupõe a existência de uma arquitetura para aplicar suas estruturas e se utiliza, principalmente, da modularização e das interfaces bem definidas da mesma.

É proposta a criação de um mecanismo de desvio do fluxo de unidades de dados e o uso de janelas programáveis para se atingir níveis de programabilidade na rede. Esta mecanismo de desvio altera o fluxo normal de unidades de dados da arquitetura à qual SNPI é implantada, fazendo que este fluxo atravesse as janelas programáveis para só depois continuar com seu caminho natural. Para melhor entendimento, é preciso especificar de que se constitui este mecanismo de desvio, o qual passaremos a nos referenciar com o nome de *hook*, e definir o que vem a ser uma janela programável.

5.1 Estruturas da Abordagem

5.1.1 Hook

Um *hook* é um artifício, de qualquer natureza, que conseguem alterar o fluxo de unidades de dados de uma arquitetura qualquer para o meio exterior. Entenda

[21] A partir deste ponto da dissertação, a palavra **usuário** quando utilizada no contexto de redes de comunicação de dados significará a figura do administrador e/ou desenvolvedor, ou seja, o indivíduo responsável e autorizado a realizar alterações nos dispositivos para que se mude o comportamento da rede.

meio exterior como um ambiente qualquer onde o usuário possa ter acesso a estas unidades de dados. O ambiente interno de uma arquitetura qualquer, por exemplo, não pode ser visto como um meio exterior, já que um usuário não pode ter acesso a este ambiente durante o processo de fluxo de unidades de dados.

Não há como especificar rigidamente como deve ser a implementação de um *hook*, pois, como será visto no próxima capítulo, tal implementação depende do ambiente em que a abordagem proposta é desenvolvida e dos recursos que o sistema operacional do dispositivo disponibiliza. Abaixo, na figura 5 , se encontra uma abstração do que seria um *hook* na esfera de um dispositivo que recebe/envia um fluxo.

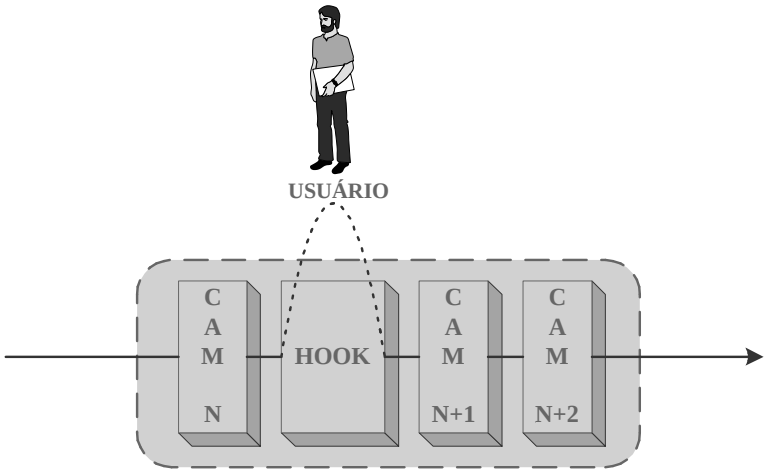

Figura 5: O comportamento de um *hook*

5.1.2 Janela Programável e Esqueletos de Procedimentos

Uma janela programável é definida como um meio exterior, conceito apresentado na seção anterior, que permite o acesso, através de outras estruturas chamadas de

esqueletos de procedimentos, às unidades de dados provenientes do fluxo desviado pelo *hook*, criando assim um ambiente de programação na rede. Desta forma, uma janela programável é uma estrutura, de qualquer natureza, capaz de receber o fluxo que foi desviado pelo *hook.*

Já os esqueletos de procedimentos são os locais dentro das janelas programáveis onde o usuário vai ter acesso às unidades de dados e às estruturas da arquitetura para que possa implantar novos serviços ou alterar as próprias unidades de dados. A disponibilidade das estruturas depende de como é implementado o *hook*. Pode ser desviado para dentro da janela programável somente as unidade de dados da arquitetura, ou até mesmo somente os cabeçalhos das unidades de dados, como pode ser estendido o nível de programabilidade disponibilizando estruturas internas mais significantes.

A programabilidade dentro das janelas programáveis, mais exatamente nos esqueletos de procedimentos, pode acontecer de duas formas:

- Definindo e disponibilizando uma linguagem de alto nível que crie facilidades para o preenchimento dos esqueletos de procedimentos e que seja limitada dentro do escopo desejado. Uma linguagem como PLAN[7] seria bem aproveitada pela abordagem já que permite, além do tratamento das unidades de dados, um gerenciamento dos recursos utilizados pelo dispositivo.

- Outra forma é permitir que o usuário se utilize de linguagens de baixo/médio nível(C/C++, *Assembly*, etc.) que, por ventura, construíram a própria arquitetura principal . Esta última possibilidade dispensa a especificação da

linguagem, só necessitando de especificação das estruturas internas disponibilizadas pelo *hook*, e fornece ao usuário um ambiente que se assemelha ao ambiente interno da própria implementação da arquitetura principal, só que localizado num meio externo. Neste último caso, os esqueletos depois de preenchidos de acordo com o usuário tornam-se procedimentos semelhantes àqueles que são internos à implementação da arquitetura, com a diferença de estarem localizados num ambiente externo.

Na figura 6 encontramos uma janela programável com a presença dos elementos usuário e esqueleto de procedimento, onde fica evidente que uma janela programável é uma estrutura responsável por receber o fluxo de unidades de dados oriundo de um ambiente interno, no caso a arquitetura à qual o *hook* está implantado. Já os esqueletos de procedimentos são estruturas próprias para a implantação de novos serviços ou alteração das unidades de dados provenientes deste fluxo.

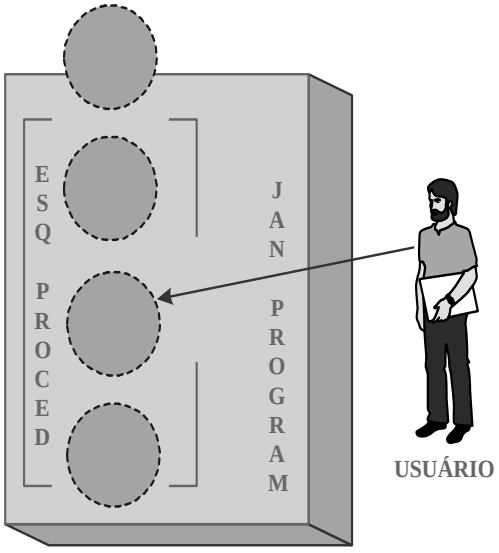

Figura 6: Janela programável

5.2 Especificação da Abordagem

Como mencionado anteriormente, a abordagem apresentada é diretamente dependente de uma arquitetura existente. É possível de ser implantada em qualquer arquitetura que seja modular e que tenha as interfaces entre suas camadas muito bem definidas. Felizmente, as principais arquiteturas (TCP/IP, OSI) possuem essas duas propriedades.

Estas duas características citadas acima são consideradas fundamentais, pois sem uma arquitetura modular e com serviços bem definidos em cada camada fica comprometido um melhor esclarecimento para o usuário de onde o fluxo, o qual está sendo recebido através da janela programável, está sendo oriundo e por quais tratamentos o mesmo já passou ou ainda vai passar.

Como exemplo, considere a existência de uma arquitetura de redes de comunicação de dados que não seja modular. Desta forma, o usuário ao receber o fluxo desviado pelo *hook* vai desconhecer os tratamentos já feitos neste fluxo e não vai ter conhecimentos suficientes para melhor modificar as unidades de dados, nem mesmo implantar serviços. Uma solução para tal problema seria uma especificação, partindo do responsável pela criação do *hook*, mostrando por quais tratamento o fluxo já atravessou e por quais ainda deve passar caso o mesmo retorne para a arquitetura, somente desta forma o usuário poderá alterar as unidades de dados com mais segurança e evitar inconsistências. O problema citado é ilustrado na figura 7.

A abordagem proposta consiste na criação de *hooks* que permitam o acesso a determinadas estruturas internas da arquitetura e que altere o fluxo de unidades de

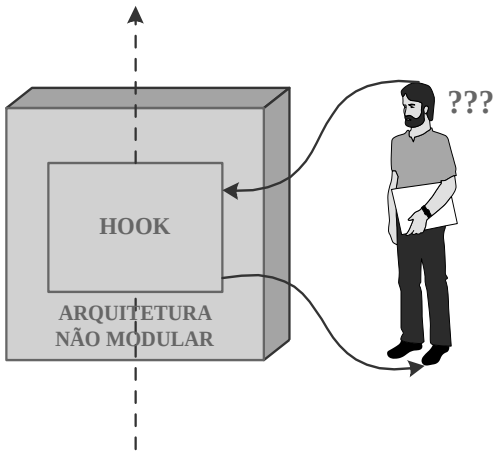

Figura 7: Problema da abordagem SNPI com arquiteturas não modulares

dados para janelas programáveis. Na figura 8 é mostrada uma arquitetura qualquer, modular e com interfaces entre suas camadas bem definidas, onde foi aplicada a abordagem em questão, que por sua vez é composta pelas janelas programáveis, esqueletos de procedimentos e pelos *hooks*. A arquitetura em questão é composta de N camadas e, neste exemplo, existem *hooks* da camada 2 até a camada N(Na figura 8, para melhor visualização, somente *hooks* na camada 2 são mostrados), significando que o fluxo normal de unidades de dados será desviado para janelas programáveis N - 1 vezes em cada direção. Lembrando que o fluxo de unidade de dados numa arquitetura é bidirecional e por esta razão é que existem dois *hooks* por camada.

Um aspecto muito importante desta abordagem diz respeito à colocação dos *hooks* dentro da arquitetura . Foi decidido por colocar os *hooks* que tratam do fluxo ascendente sempre no início de uma nova camada da arquitetura. Desta forma, se uma unidade de dados é recebida por uma camada M e existe um *hook* no interior desta camada, então, esta unidade será, primeiramente, desviada para um

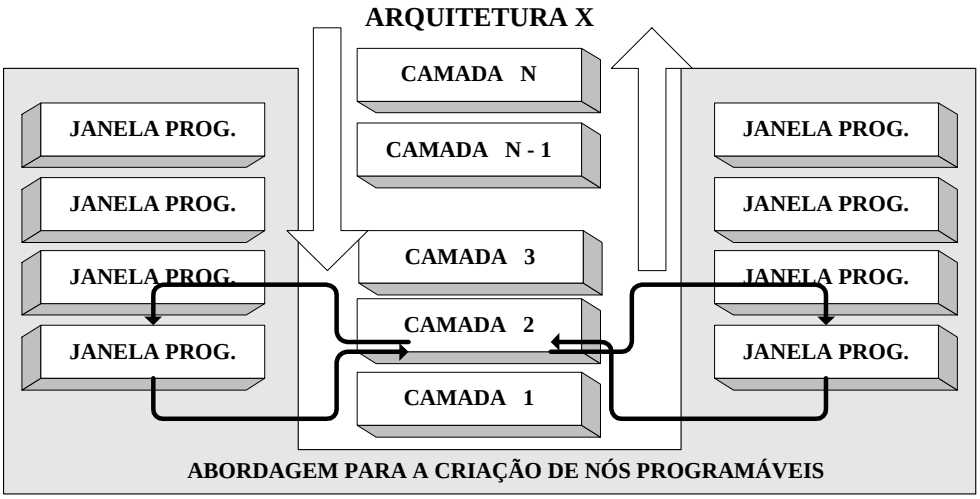

Figura 8: Abordagem SNPI aplicada a uma arquitetura genérica.

esqueleto de procedimento da janela programável correspondente. O usuário fará o tratamento de acordo com as estruturas que lhe são disponibilizadas e somente depois é que a unidade retorna ao seu curso normal sendo tratada pela camada M. Já no caso do fluxo descendente ocorre diferente, as unidades de dados são desviadas depois que tenham sido tratadas pela camada correspondente.

A escolha dos locais de implantação dos *hooks* dentro das camadas foi baseada na análise dos serviços possíveis de serem implantados através da abordagem proposta. A maioria dos serviços que se utilizam do fluxo ascendente são serviços que buscam filtrar, analisar, modificar ou descartar as unidades de dados para que elas não continuem seu caminho na arquitetura. Tendo em vista que a unidade de dados daquela camada já está pronta para passar por tais tratamentos, uma vez que seu cabeçalho e *payload* surgiram com a retirada do cabeçalho da camada inferior, é de simples conclusão que tal unidade de dados já pode ter seu comportamento alterado. De forma análoga, serviços que se utilizam do fluxo descendente precisam de informações do cabeçalho da camada em questão, no

entanto, tal cabeçalho é formado pela própria camada em toda sua extensão, daí tal unidade de dados só poder ser desviada ao final de sua formação, ou seja, no final do tratamento realizado por sua respectiva camada.

Outro aspecto muito importante diz respeito à necessidade de *hooks* numa camada. Embora a figura 8 pressuponha a existência de *hooks* da camada 2 até a camada N da arquitetura, é evidente que esse aspecto é flexível e inteiramente dependente de quem estiver implantando a abordagem. Se tomarmos a arquitetura TCP/IP como exemplo, a existência de *hooks* na camada de transporte seria importante para estações finais mas certamente desnecessária para estações intermediárias que só implementam até a camada de inter-rede. Com isso, os responsáveis por implantar as estruturas de *hook* podem se sentir livres para implantar nas camadas que atendam melhor suas necessidades.

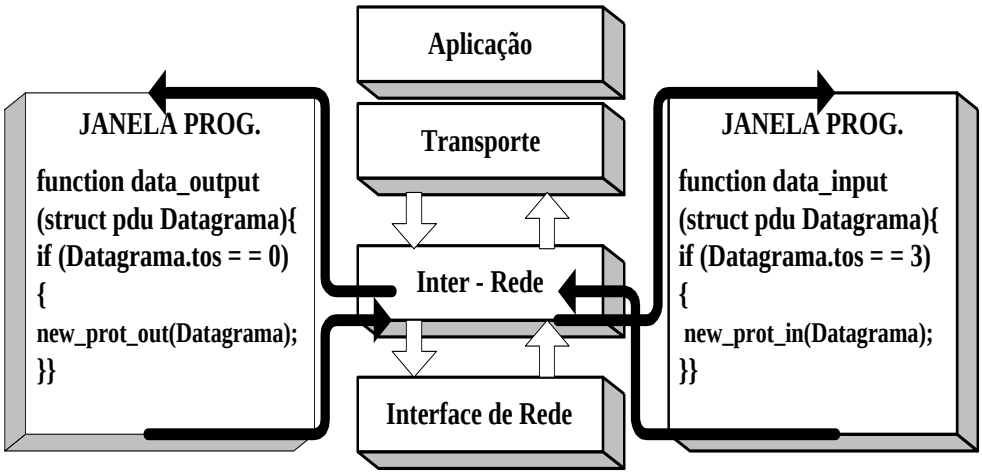

Figura 9: Abordagem SNPI aplicada à arquitetura TCP/IP.

Para melhor visualizar a abordagem proposta neste trabalho, faz-se necessário o uso de um exemplo apresentado na figura 9. Este exemplo facilitará a compreensão da implementação que é descrita no próximo capítulo. Na figura 9 é mostrada

a arquitetura TCP/IP. Foram colocados *hooks* somente na camada de inter-rede. Portanto, são dois *hooks*: um para o fluxo ascendente e outro para o fluxo descendente. Quando um *frame* é passado da camada de interface de rede para a camada de inter-rede, o cabeçalho do frame é retirado e temos um datagrama. Como na camada de inter-rede existe um *hook*, este datagrama juntamente com outras possíveis estruturas internas são passados para uma janela programável chegando ao esqueleto do procedimento correspondente que, por sua vez, recebe tais estruturas como argumentos. De dentro desta janela programável o usuário pode implementar o corpo do esqueleto de procedimento para alcançar seus objetivos. Depois, o datagrama é retornado à camada de inter-rede que realiza seu tratamento normal. Algo muito parecido ocorre no fluxo descendente, a única diferença é que o datagrama só será passado para a janela programável quando a camada de inter-rede tiver feito todo o tratamento e construção do datagrama.

Finalizando esta seção, é importante mencionar a ocorrência de uma alteração na máquina de estados da arquitetura à qual a abordagem foi aplicada. Tal fato ocorre no momento em que os *hooks* alteram o fluxo de dados para as janelas programáveis. A comparação das máquinas de estados é mostrada na figura 10 que demonstra um crescimento, por fluxo, em duas unidades no número de estados. Tal fato é importante para a implementação que vai precisar se preocupar com aspectos como desempenho e, por isso, é interessante que busque a melhor maneira de se implementar esta abordagem.

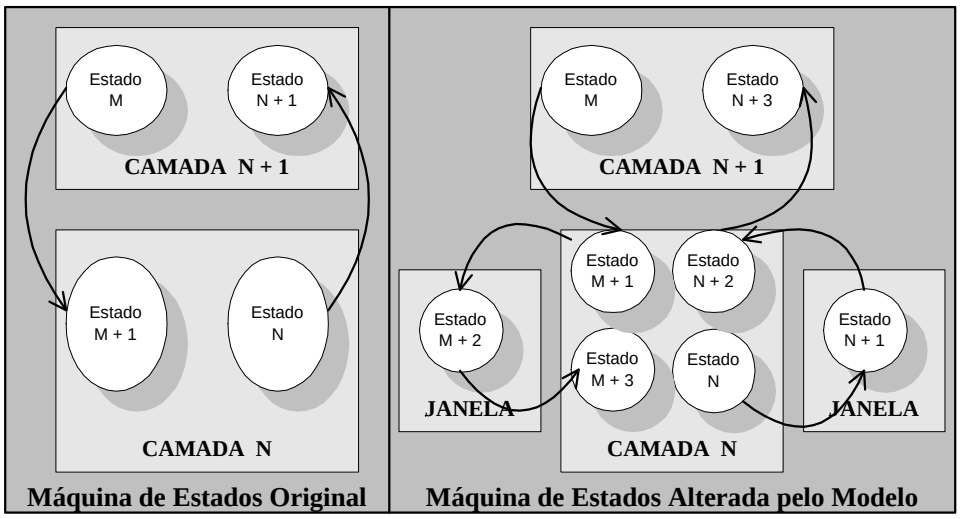

Figura 10: Comparação das máquinas de estados.

5.3 Conclusões sobre a Abordagem

Apresentada a abordagem a que este trabalho se propunha, faz-se necessário analisar certos aspectos que são importantes no detalhamento da proposta. Ficou evidente durante a apresentação da proposta a simplicidade na concepção da abordagem. Tal aspecto é importante, já que minimiza o problema mencionado anteriormente que era a complexidade na implantação das abordagens, sendo um ponto positivo desta proposta.

Interrogações podem aparecer no que diz respeito à implantação da abordagem. A criação dos *hooks*, dos esqueletos de procedimentos e das janelas programáveis vai depender da flexibilidade e de particularidades do sistema operacional do dispositivo em questão e de como a arquitetura principal foi implementada. Todos estes aspectos serão melhor endereçados no próximo capítulo que trata de uma implementação desta abordagem para plataforma Linux.

O trabalho proposto cria um ambiente programável poderoso e muito flexível, mas

um aspecto negativo do mesmo pode ser notado, que é a dependência dos novos serviços adicionados através de janelas programáveis com a arquitetura à qual a abordagem foi aplicada. Mesmo sendo possível a criação de novos serviços somente com o uso das janelas programáveis, *hooks* e esqueletos de procedimentos, estes novos serviços ficam dependentes da arquitetura em questão. No entanto, este problema de dependência não compromete a abordagem, visto que a arquitetura TCP/IP é utilizada como um padrão em quase todas as redes de comunicação de dados do mundo e possui os requerimentos de ser modular e possuir seus serviços bem definidos por camada. Ao mesmo tempo, outras arquiteturas de rede comprometidas com o padrão OSI/ISO, por estarem bem organizadas e implementadas em camadas, são grandes candidatas a lançarem mão desta abordagem.

Com todas estas considerações e analisando o estado das atuais arquiteturas de rede, este problema de dependência se torna diminuto e pode até mesmo ser desconsiderado enquanto o ambiente for favorável.

6 Aspectos Práticos - Uma Implementação

A implantação da abordagem proposta neste trabalho fica condicionada aos vários mecanismos e facilidades que o sistema operacional pode disponibilizar e da forma que foi implementada a arquitetura escolhida.

Antes de mostrar uma implementação da abordagem, faz-se necessária a menção de alguns aspectos que são considerados essenciais para uma adequada implantação. O desrespeito ou não cumprimento de algum destes aspectos pode denegrir a abordagem ou modificá-la de tal forma que sua flexibilidade, seu dinamismo e sua robustez tornarão-se diminutos.

6.1 Aspectos Essenciais

- A implantação da abordagem deve se utilizar da implementação de uma arquitetura modular e com interfaces bem definidas entre suas camadas, com objetivos de evitar desconfortos para o usuário.
- As estruturas da abordagem, mencionadas na seção 5.1, devem estar presentes e serem respeitadas durante a implantação das mesmas.
- Não existe especificação de quais estruturas internas da arquitetura devem ser desviadas pelos *hooks*. Mas tais estruturas devem estar acessíveis nas janelas programáveis por meio dos esqueletos de procedimentos e estes, por sua vez, devem ser preenchidos pelos usuários.
- Não há necessidade do usuário tomar conhecimentos de detalhes internos do sistema operacional do dispositivo nem de detalhes de implementação

da arquitetura. É necessário somente, o detalhamento das estruturas que são acessíveis, em que ponto da arquitetura tais estruturas estão sendo desviadas e caso seja utilizada uma linguagem própria para a programação dos esqueletos, que esta seja apresentada e especificada.

- Não é desejado que ocorram recompilações no sistema operacional do dispositivo e nem na implementação da arquitetura para que os serviços, construídos nas janelas programáveis, possam ser adicionados. Caso este fato venha a ser imprescindível, então a implementação não será fiel à abordagem e não contribuirá em nada para a implantação dinâmica de serviços, uma vez que existem sistemas abertos que possibilitam este tipo de prática.

6.2 Uma Implementação

Respeitando os aspectos essenciais mencionados a pouco e a abordagem proposta neste trabalho, apresentamos uma implementação feita na camada de inter-rede da pilha TCP/IP do sistema operacional *Red Hat*[22] *Linux*[24]. O sistema operacional Linux foi escolhido por diversas razões, onde as principais são:

- A necessidade de acesso à implementação da pilha TCP/IP para que fossem implementados os *hooks* no seu interior.

  Como Linux é um sistema operacional que possui um *kernel* monolítico[23], isso significa que boa parte das tarefas a serem realizadas pelo sistema é

[22] A distribuição *Red Hat* foi escolhida por ser a mais disseminada na atualidade. Mesmo assim, a implementação apresentada neste trabalho deve ser operacional em outras distribuições, uma vez que as modificações no *kernel* foram diminutas e em certas partes que não devem ter sido alteradas nas principais distribuições.

[23] Existe uma certa discussão sobre o comportamento do *kernel* do Linux (monolítico ou *micro-kernel*?). A verdade é que ele é monolítico mas com características modulares, aspecto este que

de responsabilidade do *kernel*, inclusive aspectos de interligação em rede (*drivers* e pilhas de protocolo).

A figura 11 apresenta as principais tarefas de um *kernel* com características monolíticas.

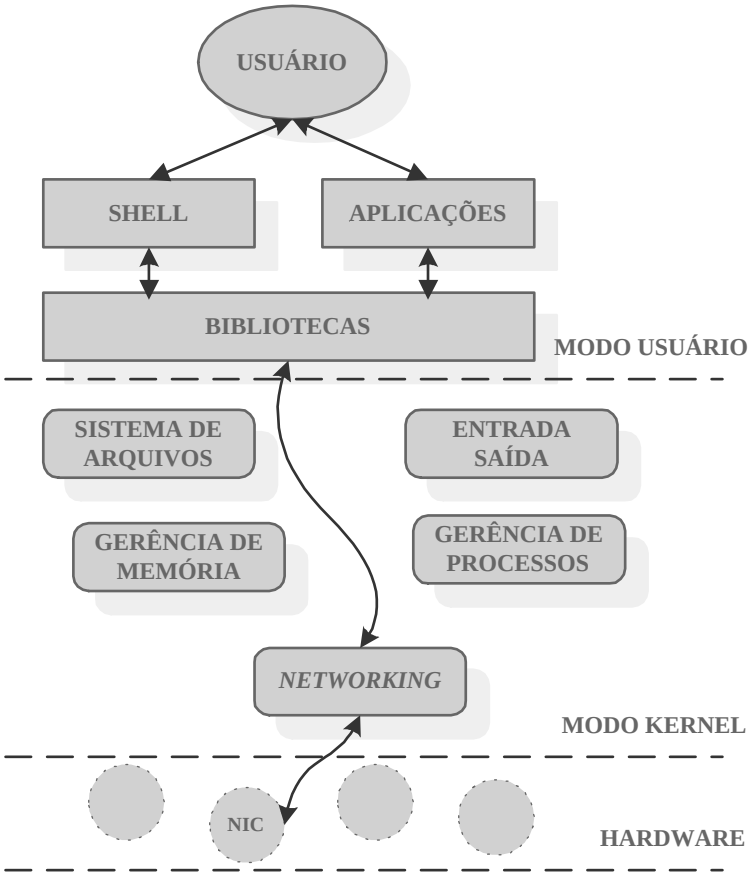

Figura 11: Funcionamento de um *kernel* monolítico

Além do mais, o código fonte do *kernel* do Linux está disponível para averiguações e modificações o que possibilitou o acesso à pilha TCP/IP.

➠ O poder de programação que o Linux possui frente a outros sistemas ope-

é apresentado nesta mesma seção. Para se tornar um *microkernel*, seria preciso enxugar o *kernel* atual deixando somente as funções vitais e que não podem ser colocadas no espaço do usuário, fato este que não acontece.

racionais. E mais, certas particularidades que foram de grande valia como, por exemplo, o acesso ao *kernel* através de módulos.

- A grande penetração que o Linux possui no ambiente acadêmico, possibilitando que pesquisadores tomem conhecimento dos detalhes da implementação e criando expectativas de trabalhos futuros e melhoramentos.

Antes que seja detalhada a implementação, é preciso mencionar o que vem a ser um módulo no sistema operacional Linux, já que foi através da característica modular do *kernel* que a implementação se tornou flexível e tão poderosa quanto a abordagem, evitando o uso de mecanismos de intercomunicação entre processos o que poderia prejudicar de forma significativa o desempenho.

Um módulo é, em suma, uma parte do *kernel* que não é carregada diretamente, ou seja, em tempo de inicialização do sistema operacional, podendo ser incorporada ao *kernel* a qualquer momento e sem necessitar de interrupções. Um usuário que possua permissões suficientes, por exemplo, pode escrever um módulo, depois compilá-lo e, então, carregá-lo no *kernel* em tempo de operação.

A figura 12 mostra o processo de incorporação de um módulo ao *kernel*. O Usuário constrói um módulo e compila-o normalmente utilizando o compilador **gcc**, por exemplo. Depois, caso possua permissões suficientes, incorpora-o ao *kernel* através do comando **insmod**. Feito isto, o módulo fará parte do *kernel* podendo utilizar seus mecanismos internos.

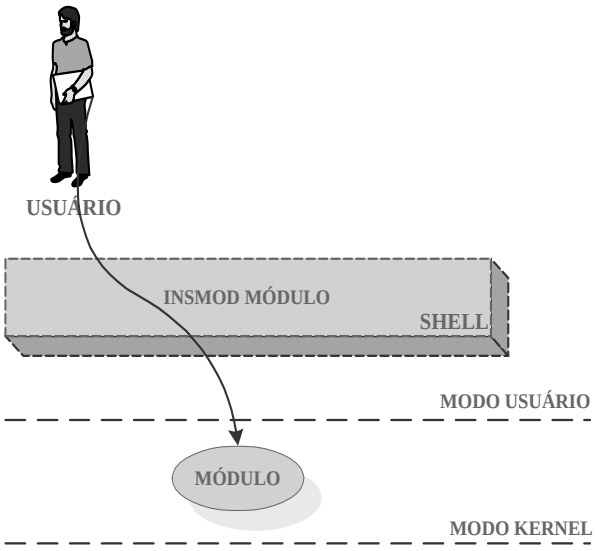

Figura 12: Incorporação de um módulo ao *kernel*

6.2.1 Mapeamento das Estruturas da Abordagem

Apresentado todo o conhecimento necessário para o entendimento da implementação, é chegado o momento de mapear as estruturas da abordagem proposta para os mecanismos que foram utilizados na implementação.

1. Os *hooks* foram implementados no código fonte da camada de inter-rede da pilha TCP/IP como ponteiros que referenciam funções. A estes ponteiros é associado, inicialmente, o valor *NULL*, ou seja, não referencia nenhuma função.

2. Cada janela programável é mapeada como sendo, simplesmente, um módulo e todas as particularidades que ele oferece. No caso da implementação apresentada nesta seção, foi utilizada somente uma janela programável, ou seja, um módulo para guardar os dois esqueletos de procedimentos, o que trata o fluxo ascendente e o que trata o fluxo descendente.

3. Os esqueletos de procedimentos foram implementados como funções dentro do módulo e que possuem seus corpos vazios à espera da construção de novos serviços por parte dos usuários.

A seção seguinte apresenta, de forma mais técnica, os mecanismos utilizados e o relacionamento entre eles.

6.2.2 Detalhamento da Implementação

Já que os principais aspectos da implementação já foram mostrados, será descrito brevemente como ocorreu a implementação das estruturas que compõem a interface SNPI no *Red Hat Linux* .

Primeiramente, foi criado um arquivo **snpi.c** onde foram declarados dois ponteiros que referenciam funções que precisam retornar um inteiro, como pode ser visto no trecho de código abaixo. Estas funções são, na verdade, os esqueletos de procedimentos que ainda não foram implementados pelo usuário e, por isso, não podem ser referenciadas. Sendo assim, os ponteiros recebem o valor *NULL* nas suas declarações, evitando referências inconsistentes.

```
/* This function's pointer is used by SNPI */

int (*snpi_input_call)(struct sk_buff *)=NULL;

/* This function's pointer is used by SNPI */
```

```
int (*snpi_output_call)(struct sk_buff *)=NULL;
```

Depois, mais precisamente nos arquivos **ip_input.c**, **ip_output.c** e **ip_forward.c**, foram inseridas chamadas para os ponteiros declarados anteriormente. Estas chamadas alteram o fluxo das estruturas **sk_buff** que representam o datagrama IP. Abaixo está um exemplo de como o ponteiro que trata o fluxo ascendente, **snpi_input_call**, foi utilizado no interior do arquivo **ip_input.c**.

```
/* SNPI Experimental(ip_input.c) */
#ifdef  CONFIG_SNPI
   if(snpi_input_call!=NULL)
   {
       if(snpi_input_call(skb)==PACKET_DROP) goto inhdr_error;
   }
#endif
/* End of SNPI */
```

Pelos trechos de código apresentados até o momento, pode-se concluir que cada datagrama IP (**sk_buff**) que chegar ao dispositivo, ou sair dele, será passado como argumento para as funções referenciadas pelos respectivos ponteiros. Talvez, o leitor pode se perguntar que funções são estas já que em nenhum momento os valores dos ponteiros deixaram de ser *NULL*. Tal dúvida deve ficar esclarecida logo mais.

Um fato interessante que pode ser observado nos trechos de código mostrados acima é que os ponteiros são direcionados para funções que retornam inteiros e não para procedimentos. Tal aspecto não é especificado na abordagem mas foi uma facilidade adicionada permitindo que os desenvolvedores de novos serviços não se preocupem com particularidades como a de liberar a área de memória ocupada pela estrutura **sk_buff**, caso ele considere que esta não deve continuar seu caminho na pilha. Basta, então, ao terminar a implementação das funções, que o desenvolvedor especifique se ele deseja que o datagrama continue seu caminho, `PACKET_OK`, ou se ele deve ser imediatamente descartado pelo própria camada, `PACKET_DROP`.

Outro aspecto que deve ser de conhecimento do usuário é que o Linux não possui uma estrutura para cada unidade de dados. Só existe uma única estrutura, **sk_buff**, que representa a mensagem completa que chegou ao dispositivo[24]. Mas como esta implementação está limitada à camada de inter-redes, não existem problemas de referenciar esta estrutura como um datagrama IP. A verdade é que o usuário terá um gama maior de informações que poderá usar no momento de preencher o corpo do esqueleto de procedimento.

Com os *hooks* já implementados, falta somente detalhar a janela programável (módulo) e os esqueletos de procedimentos (funções).

A janela programável é um módulo, implementado fora do *kernel*, onde está declarado que existem dois ponteiros internos ao *kernel* e que referenciam funções. Depois de implementado, o módulo pode ser compilado e carregado no *kernel*

[24]Mesmo tendo acesso a esta estrutura global, não significa que quaisquer modificações realizadas surtirão efeitos. Se você tiver tratando um fluxo ascendente na camada de transporte, por exemplo, modificações no cabeçalho referente à camada de inter-rede poderão ser realizadas mas não produzirão qualquer efeito, uma vez que o respectivo tratamento já foi realizado.

durante o tempo de operação do sistema.

A função que é executada durante a carga do módulo, `int init_module()`, atribui os endereços dos respectivos esqueletos de procedimentos (**your_input_call** e **your_output_call**) aos ponteiros, que até o momento eram *NULL*. Na descarga do módulo, a função `void cleanup_module()` atribui de volta os valores *NULL* aos ponteiros.

Para finalizar, os esqueletos são duas funções que tem a mesma declaração de argumentos que os ponteiros declarados no interior do *kernel*. O código abaixo mostra os esqueletos de procedimentos, ficando evidente que seus corpos estão vazios esperando por implementações dos usuários. O único comando existente é aquele que trata o valor de retorno mas que pode ser alterado pelo usuário de acordo com suas necessidades.

```
int your_input_call(struct sk_buff *skb)
{
    return PACKET_OK;
}

int your_output_call(struct sk_buff *skb)
{
    return PACKET_OK;
}
```

Terminado o detalhamento da implementação, é hora de mostrar como ocorre todo o processo que pode ser acompanhado pela figura 13. Quando o Linux é iniciado

a pilha TCP/IP juntamente com os hooks já se tornam ativos. Sendo assim, todo e qualquer datagrama IP que passar pela camada de inter-rede não sofrerá qualquer mudança no seu fluxo, pois os ponteiros estão com o valor *NULL*. Depois de ter programado o módulo de acordo com sua vontade, o usuário compila-o através do **gcc**, por exemplo, e carrega o binário resultante no interior do *kernel* através do comando `insmod nome_do_modulo`. Feito isso, o módulo modificará os valores dos ponteiros de *NULL* para os endereços dos respectivos esqueletos de procedimentos que tiveram seus corpos preenchidos pelo usuário. A partir deste momento, os datagramas IP serão desviados e tratados de acordo com aquilo que foi escrito pelo usuário.

Eventualmente, pode-se descarregar o módulo através do comando `rmmod`. Se isso acontece, o módulo irá atribuir o valor *NULL* aos ponteiros para que os mesmos não continuem apontando para regiões de memória que não estão mais alocadas.

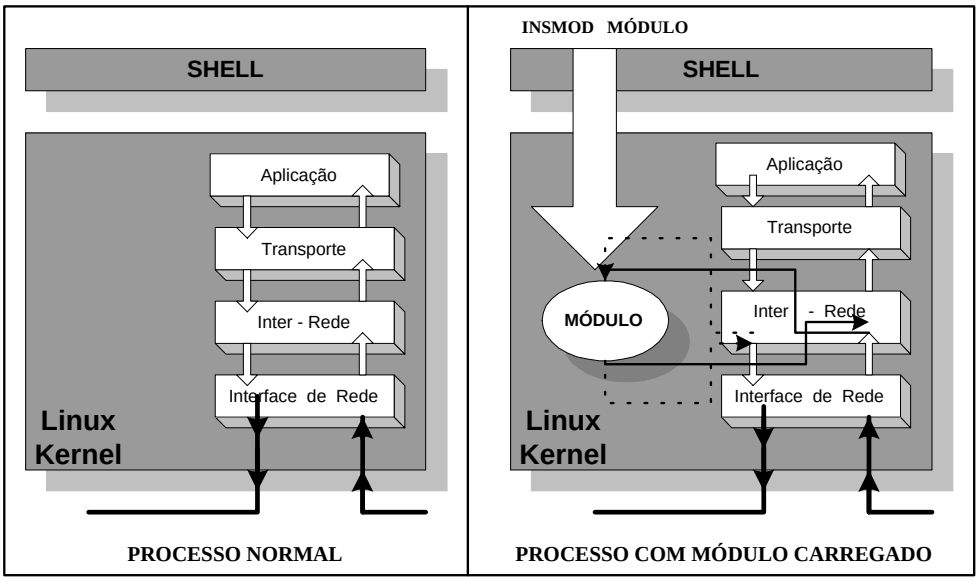

Figura 13: Fluxo de dados antes e depois da carga do módulo

Um detalhe importante, mas que pode ter passado desapercebido, diz respeito ao

acesso aos ponteiros, *hooks*, uma vez que eles foram implementados no interior do *kernel* do Linux. Como um módulo, que foi desenvolvido externamente ao *kernel*, pode ter acesso às estruturas internas podendo alterá-las? Deve-se lembrar que o módulo é definido como sendo uma parte do *kernel* que não é carregado diretamente. Portanto, existem mecanismos para que mesmo de dentro dos módulos se tenha acesso a estruturas pertencentes ao *kernel* propriamente dito. Se você deseja que uma estrutura fique acessível por módulos basta que você especifique. Veja o exemplo abaixo, onde foram exportados os dois ponteiros que dizem respeito aos *hooks*. Este trecho de código foi inserido no arquivo **netsyms.c** que possui todas as estruturas de *networking* que podem ser acessíveis por módulos.

```
#ifdef CONFIG_SNPI

EXPORT_SYMBOL(snpi_input_call);

EXPORT_SYMBOL(snpi_output_call);

#endif /* CONFIG_SNPI */
```

Vale ressaltar que a programação no interior dos módulos ocorre da mesma forma que no *kernel*[22].

6.3 Análise da Implementação

Após especificada a implementação, é hora de apresentar um aspecto importante para o seu detalhamento, a questão desempenho. Como já era esperado, não existe nenhuma alteração no desempenho quando SNPI é analisado individualmente, ou seja, sem nenhum serviço implantado através dele. Para montar a tabela 1 foi realizada uma transferência de dados aproximadamente 32 MB através da própria interface de *loopback*[25] em três ambientes diferentes. O primeiro ambiente consiste de um sistema sem suporte SNPI. Já o segundo consiste do mesmo ambiente anterior mas agora com suporte SNPI. E, por último, carregou-se no sistema com suporte SNPI um módulo que não realizava nenhum processamento nos pacotes, funcionando somente como uma estrutura de *bypass*. Em cada ambiente foram retiradas três amostras para resultar numa média, além disso, resultados com desvios padrões acentuados foram desconsiderados.

Ambiente	**A1**	**A2**	**A3**	**Média**	**Diferença**
Sem Suporte SNPI	1,48 s	1,50 s	1,50 s	1,49 s	***
Com Suporte SNPI	1,52 s	1,51 s	1,47 s	1,50 s	+ 0,67%
SNPI + Módulo	1,48 s	1,49 s	1,46 s	1,47 s	- 1,34%

Tabela 1: Análise de desempenho da interface SNPI

A máquina utilizada consistiu de um PC IBM com processador AMD K6-II de 333 Mhz e barramento de 100 Mhz. O mínimo de processos possíveis foram deixados ativos durante os testes e a interface gráfica foi evitada. A ferramenta **ttcp**[19] foi utilizada para a transferência dos dados e para a medição dos tempos.

[25]Já que SNPI, quando analisado somente como um desvio e sem serviços, não modifica o conteúdo do datagrama IP, então, não é preciso medir variáveis de tempo que não são influenciadas por SNPI e que só dificultariam o processo de medição. Usando a interface de *loopback* consegue-se calcular, de forma coerente, o tempo que é levado no processamento dos pacotes, ponto onde SNPI poderia interferir.

Esta ferramenta realiza a transferência de dados presentes na memória principal, evitando o acesso a dispositivos de entrada e saída como discos rígidos.

6.4 Conclusões sobre a Implementação

Mostrados os detalhes da implementação, pode-se concluir que a abordagem para a criação de nós programáveis em redes tradicionais, SNPI, proposta neste trabalho foi concretizada com fidelidade. Todo o poder de programabilidade de rede que a abordagem alcança está presente na implementação.

Embora não tenha sido utilizada toda a pilha TCP/IP pela implementação, já que somente a camada de inter-rede teve sua máquina de estados alterada, tal fato não foge à abordagem. Apenas ficou decidido que não era necessário implantar *hooks* em toda a arquitetura TCP/IP para demonstrar a funcionalidade e robustez da implementação, podendo tal tarefa ser realizada por outros pesquisadores seguindo os mesmo passos descritos acima e adaptando-os quando necessário.

Torna-se evidente que a implementação num determinado sistema operacional, ou numa pilha de protocolos, vai ficar dependente das facilidades que este/esta venha a oferecer. O uso de módulos, facilidade encontrada no Linux, permitiu recriar a alta flexibilidade e dinamismo da abordagem. Outros sistemas operacionais que possuam mecanismos semelhantes ou estruturas flexíveis como aqueles baseados num *microkernel* podem experimentar SNPI sem maiores dificuldades. No entanto, sistemas operacionais não tão robustos dificultam, e até mesmo impossibilitam, a implantação da abordagem. Assim como, provavelmente, dificultariam o desenvolvimento de trabalhos como aqueles apresentados no capítulo 4 deste trabalho.

Para finalizar os aspectos da implementação, como foram necessárias algumas mudanças no *kernel*, foi preciso criar um *patch* que deve ser aplicado ao código fonte do *kernel*, induzindo em uma única recompilação. O *patch* pode ser aplicado para algumas versões do *kernel* 2.2.X e 2.4.X, principalmente aquelas versões que são utilizadas pelo Red Hat Linux.

A implementação está disponível para averiguações, testes e colaborações no *site* do projeto[25]. A mesma está disponível na forma de um arquivo *gzipped* que contém: o *patch* para aplicar os *hooks* ao *kernel*; o módulo que representa a janela programável e no interior do mesmo estão presentes os esqueletos de procedimentos prontos para serem preenchidos. No mesmo endereço eletrônico encontram-se mais informações para auxiliar aqueles que desejam experimentar esta nova abordagem e, também, uma cópia deste trabalho no formato **pdf**.

7 Aplicação - NetActeon

7.1 Introdução

Apresentada a proposta desta dissertação e sua implementação, é chegado o momento de exemplificar como a mesma pode ser aplicada na construção de aplicações que objetivam, dentre diversos fins, o monitoramento e controle da rede.

Uma das grande dificuldades que os administradores de rede enfrentam na sua árdua tarefa de gestão é controlar as máquinas que compõem sua rede, garantindo que as mesmas não vão ser manipuladas por usuários mal intencionados que agem de forma compulsiva na busca de mecanismos ou ferramentas que danifiquem ou barrem serviços oferecidos por servidores de outras redes de computadores. Muitas vezes, esse usuário mal intencionado que foi bem sucedido na sua tentativa de ataque não consegue ser descoberto e toda a culpa recai sobre o administrador da rede.

Mesmo sendo tarefa de um administrador manter o bom funcionamento de uma rede e sua segurança é na prática impossível garantir que uma rede de computadores esteja 100% segura contra invasões ou que a mesma não possa ser usada como parte de um ataque. Os próprios sistemas operacionais de rede utilizados atualmente possuem diversos problemas de segurança que são descobertos dia-a-dia, tornando a vida de qualquer administrador de redes uma verdadeira corrida contra o tempo na busca por correções ou por atualizações que possam sanar esse ou aquele problema de segurança surgido no dia anterior.

Um dos grandes problemas de segurança nos dias atuais e que deixa os administradores de rede temerosos é o ataque do tipo DDoS. O ataque do tipo DDoS

consiste num cenário onde várias estações de diferentes redes são infectadas por um programa malicioso que se aproveita de alguma falha de segurança destes sistemas para invadi-los. Tais estações infectadas, as chamadas estações zumbis, passam a obedecer a uma estação também infectada e chamada de estação mestre. Quando esta estação mestre, a qual é controlada pelo atacante, emite uma ordem de ataque, as estações zumbis começam a disparar requisições de conexão para um único destino. Este ataque em massa objetiva inviabilizar os *links* de comunicação da máquina atacada como também saturá-la de requisições de forma a impedir que esta máquina venha a prover seus serviços.

Na tentativa de não serem descobertas pelos respectivos administradores de rede, as estações zumbis não transmitem quantidades de dados volumosas. Os danos surgem não quando se possui duas ou três estações zumbis mas cem ou duzentas estações zumbis enviando requisições de forma sincronizada para um mesmo destino. Este destino, em pouco tempo, estará com milhares de requisições pendentes para atender e não vai ser capaz de prover seus serviços para nenhum cliente, pois estará ocupado com as requisições provenientes das estações zumbis.

Infelizmente o problema do DDoS não possui uma solução definitiva e, hoje, é o tipo de ataque mais temido pelos administradores de grandes redes. Depois que um ataque de DDoS é descoberto e já tem causado um prejuízo valioso pelo número de transações que deixaram de ser atendidas enquanto o servidor estava saturado, resta ao administrador detectar as estações zumbis através de um simples *software* de monitoramento que pode dizer quais os IPs de origem das requisições. Depois, basta entrar em contato com os administradores das respectivas redes para alertá-los do estado de suas máquinas ou, no pior dos casos, colocar filtros no *link* de entrada para impedir que pacotes oriundos destas estações zumbis possam

prejudicar mais uma vez o funcionamento do servidor.

Embora seja um ataque que possa causar sérios prejuízos para uma empresa de *e-commerce*, depois de descoberto este tipo de ataque pode ser remediado como mencionado acima se não fosse um agravante. É possível que as estações zumbis enviem requisições com endereços IP de origem falsos na tentativa de impedir que o servidor atacado possa se defender, tal cenário gera um caos de proporções gigantescas, pois não é mais possível filtrar quais pacotes podem passar pelo *link* de entrada uma vez que não é mais possível destinguir quais requisições são oriundas de clientes e quais são oriundas de estações zumbis.

Tal cenário caótico mencionado acima não está tão longe quanto se pensa, através de uma interface *socket*, chamada de *raw sockets*, é possível criar pacotes especificando todos os campos que desejar, até mesmo o IP de origem. Até bem pouco tempo a interface de *raw sockets* só era acessível por usuários e/ou aplicações que possuíssem direitos para tal. No entanto, com a chegada do Windows XP essa limitação não existe mais. Se antes os sistemas operacionais de rede impediam o uso da interface *raw sockets* por usuários que não tivessem direitos de administrador, agora, o Windows XP já traz a interface de *raw sockets* habilitada, na sua instalação padrão, para qualquer usuário e/ou aplicação.

Neste caso citado acima, é preciso que os administradores de rede implementem filtros *anti-spoofing* nos roteadores, impedindo que saiam de suas redes pacotes que possuam IPs de origem que não façam parte de seu domínio. Depois desta exemplificação, fica claro que em determinadas situações o administrador não possui bons mecanismos para impedir que pessoas mal intencionadas possam usar sua rede para fins maliciosos. Muitos problemas seriam resolvidos se os mesmos

tivessem controle sobre os pacotes que são enviados ou recebidos por suas estações, ou seja, que os administradores além de filtrar pacotes, o que é muito comum hoje em dia, pudessem modificá-los no intuito de impedir e descobrir situações como esta. No problema acima, uma solução razoável seria impor que todos os pacotes que saíssem das estações que compõem sua rede não fossem descartados mas saíssem com determinados campos imutáveis, como por exemplo o endereço IP de origem assinalado corretamente para o endereço da máquina de origem.

É para ajudar o administrador na sua tarefa diária que este capítulo apresenta um *software* baseado em SNPI que permite ao administrador o controle e filtragem dos pacotes que fluem pelas estação de sua rede, podendo alterar certos campos dos datagramas IP, se assim achar necessário.

7.2 NetActeon - Uma Ferramenta de Controle

NetActeon[26] é uma ferramenta que auxilia o administrador na tarefa de controle de suas estações. Beseado em SNPI, NetActeon é uma ferramenta que foi desenvolvida em Java 1.2 e por isso pode ser executada em qualquer plataforma desde que exista suporte SNPI para aquela plataforma.

Com NetActeon o administrador pode especificar, campo a campo do cabeçalho, regras de filtragem e regras de alteração. Com as regras de filtragem é possível barrar datagramas indesejado tanto no recebimento como no envio. Já com as regras de alteração é possível modificar alguns campos do cabeçalho IP para valores

[26]O termo Acteon (ou Acteão) vem da mitologia grega e corresponde a um exímio caçador de animais. Este termo foi escolhido pelo seu significado e por sua semelhança com o substantivo inglês *action*.

definidos pelo administrador. A tabela 2 lista de todos os possíveis campos de um cabeçalho IP e o que é possível fazer com eles no NetActeon.

Campo do Cabeçalho IP	**Permissão**
Version	Somente Filtragem
Header Length	Somente Filtragem
Type Of Service	Filtragem e Alteração
Datagram Length	Somente Filtragem
Datagram Identification	Somente Filtragem
Flags e Fragment Offset	Não Implementado
Time To Live	Filtragem e Alteração
Protocol	Filtragem e Alteração
Checksum	Somente Filtragem
Source IP Address	Filtragem e Alteração
Destination IP Address	Filtragem e Alteração
IP Options	Não Implementado

Tabela 2: Funcionalidade implementadas pelo NetActeon

Ao executar NetActeon pela primeira vez, o usuário recebe uma mensagem de criação de um arquivo de registro (registry.dat). Neste arquivo ficarão armazenados todos os valores especificados pelo usuário na hora de estabalecer as regras. Sendo assim, numa próxima execução do programa não vai ser preciso recriar as regras pois as mesmas estarão salvas e serão recuperadas sempre que o usuário quiser modificá-las. Ainda no processo de inicialização, NetActeon verifica a existência de suporte SNPI no sistema, caso seja comprovada a existência o usuário terá acesso à tela inicial, caso contrário, será mostrado um aviso e a aplicação será finalizada. Depois de tais verificações o usuário pode criar suas regras e ativá-las.

A figura 14 explicita o funcionamento do NetActeon. Numa primeira etapa (1) as regras de filtragem e de alteração definidas pelo usuário são base para a criação de uma janela programável e para o preenchimento do esqueleto de procedimento, estruturas apresentadas no seção 5.1. Numa segunda etapa (2), NetActeon chama

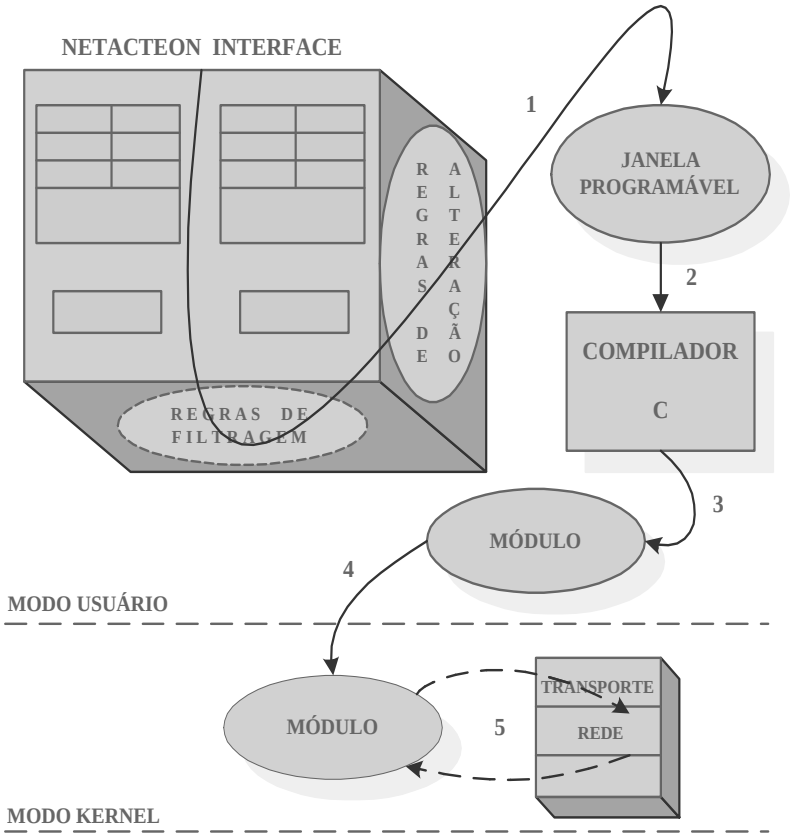

Figura 14: Funcionamento do aplicativo NetActeon no Linux

um compilador nativo da plataforma passando a janela programável como parâmetro, esse processo deixa NetActeon em espera e qualquer erro nesta etapa vai ser reportado para o usuário. Se a segunda fase é realizada com sucesso, então alcança-se a terceira etapa tendo como resultado um objeto pronto para ser carregado no sistema. Logo após, NetActeon inicia a quarta etapa (4) que consiste em carregar este objeto no sistema e fazer sua ligação com os *hooks* já presentes. A última etapa (5) consiste no processo de receber os datagramas que estão atravessando a pilha TCP/IP, fazer alterações de acordo com as regras de alteração e depois injetá-los de volta se assim as regras de filtragem permiterem. Em qual-

quer momento, o usuário pode interromper o processo de filtragem/alteração dos datagramas para especificar novas regras de filtragem e alteração.

Cada campo do cabeçalho IP presente no NetActeon originou um classe Java, além de outras classes necessárias para a tarefa de verificação dos valores entrados pelo usuário e classes para facilitar a tarefa do usuário na hora de criar suas regras. A figura 15 explicita todas as classes Java que compõem NetActeon e suas dependências. E na tabela 3 encontra-se uma pequena descrição de todas as classes com suas repectivas funções dentro do NetActeon.

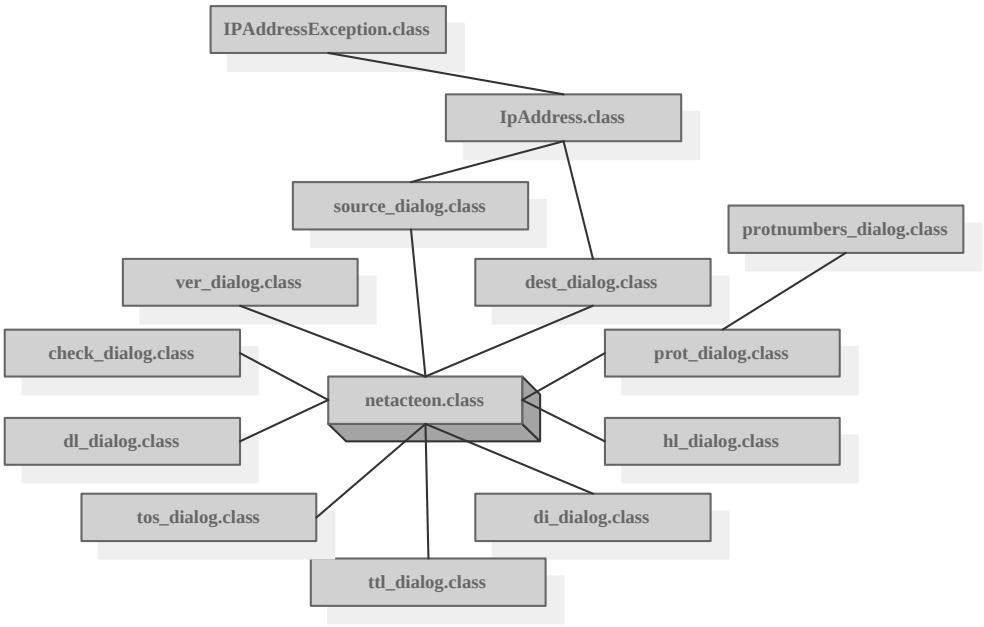

Figura 15: Classes Java que compõem NetActeon

7.3 Análise da Ferramenta NetActeon

Embora seja uma ferramenta de administraçao e controle de rede interessante e muito útil, que pode solucionar alguns problemas simples e até mesmo auxiliar na

Classe	Função
ver_dialog.class	Responsável pelo campo *Version* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 0 - 15.
hl_dialog.class	Responsável pelo campo *Header Length* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 5 -15 .
tos_dialog.class	Responsável pelo campo *Type Of Service* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 0 - 255.
dl_dialog.class	Responsável pelo campo *Datagram Length* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 20 - 65535.
di_dialog.class	Responsável pelo campo *Datagram Identification* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 0 - 65535.
ttl_dialog.class	Responsável pelo campo *Time To Live* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 0 - 255.
proto_dialog.class	Responsável pelo campo *Protocol* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 0 - 255.
proto_numbers.class	Usada pela classe proto_dialog.class e que auxilia apresentando ao usuário valores de protocolos conhecidos, para que o mesmo não precise procurar por tais valores.
check_dialog.class	Responsável pelo campo *Checksum* do datagrama IP. A própria classe realiza as verificações necessárias, aceitando valores no intervalo 0 - 65535.
source_dialog.class	Responsável pelo campo *Source IP Address* do datagrama IP.
dest_dialog.class	Responsável pelo campo *Destination IP Address* do datagrama IP.
IpAddress.class	Usada pelas classes source_dialog.class e dest_dialog.class para realizar o processo de verificação de endereços IP, evitando que o usuário entre com endereços inválidos.
IPAddressException.class	Utilizada pela classe IpAddress.class quando ocorre alguma exceção.

Tabela 3: Classes Java do NetActeon e suas funções

solução de problemas mais complexos como aquele mencionado no início deste capítulo, o que impulsionou a concepção de tal ferramenta foi a possibilidade de estudar o desempenho de aplicações que usam SNPI.

Como já discutido na seção 6.3, a implementação da abordagem SNPI para Linux não acarreta em alterações no desempenho da pilha TCP/IP desde que utilizada única e exclusivamente como uma estrutura de desvio, ou seja, sem nenhum processamento a mais. Isso significa dizer que qualquer alteração no desempenho vai ser dependente da aplicação que está utilizando SNPI mas, até o momento, nenhum teste com aplicações tinha sido apresentado.

As tabelas que se seguem mostram comparativos de desempenho entre um sistema Linux com suporte SNPI mas sem NetActeon (sem regras) e o mesmo sistema mas agora com NetActeon em ação. A primeira tabela foi construída com resultados extraídos de uma transferência de dados de aproximadamente 32 MB. Num primeiro momento foi realizada a transferência sem a existência de regras, depois foram adicionadas dez regras de filtragem e uma regra de alteração e assim por diante como mostra a tabela 4. Para cada transferência foram realizadas três amostras para o cálculo de uma média, além disso, amostras com desvios padrões acentuados foram desconsideradas.

Número de Regras	A1	A2	A3	Média	Diferença
Sem Regras	1,52 s	1,50 s	1,52 s	1,51 s	***
10 R.F e 1 R.A	1,59 s	1,56 s	1,56 s	1,57 s	3,9%
20 R.F e 2 R.A	1,64 s	1,63 s	1,61 s	1,62 s	7,2%
30 R.F e 3 R.A	1,69 s	1,69 s	1,70 s	1,69 s	11,9%

R.F: Regras de Filtragem | **R.A: Regras de Alteração**

Tabela 4: Análise de desempenho com transferências de 32 MB.

Na tabela 5, foi realizado o mesmo experimento mas agora a quantidade de dados

a ser transferida tinha um tamanho de aproximandamente 128 MB.

Número de Regras	A1	A2	A3	Média	Diferença
Sem Regras	6,09 s	6,10 s	6,07 s	6,09 s	***
10 R.F e 1 R.A	6,30 s	6,27 s	6,27 s	6,28 s	3,1%
20 R.F e 2 R.A	6,50 s	6,48 s	6,50 s	6,49 s	6,5%
30 R.F e 3 R.A	6,76 s	6,76 s	6,72 s	6,74 s	10,6%

R.F: Regras de Filtragem | **R.A: Regras de Alteração**

Tabela 5: Análise de desempenho com transferências de 128 MB.

Durante os testes realizados foram emitidos e recebidos pacotes que não poderiam ser filtrados por nenhuma das regras de filtragem especificadas, pois pacotes descartados dificultariam o processo de análise de desempenho. Os testes foram realizados com a própria interface de *loopback* assim como os testes realizados na seção 6.3. Além disso, a máquina e as ferramentas também foram as mesmas da seção 6.3

Constata-se através das tabelas 4 e 5 que NetActeon pode adicionar um tempo de processamento considerável se o número de regras aumentar de forma exagerada. Mas quando analisado o conjunto, ou seja, o processamento do pacote; seu envio para a interface; seu enfileiramento nos buffers da interface de transmissão e recepção; sua transmissão e outros fatores é possível supor que este tempo a mais no processamento não prejudica de forma decisiva o tráfego.

7.4 Considerações Finais sobre NetActeon

NetActeon apresenta-se como uma ferramenta bastante útil na rotina de um admnistrador de rede, já que o mesmo pode utilizá-la para resolver diversos problemas

de natureza simples. É possível utilizá-la facilmente como uma ferramenta de filtragem de datagramas IP, possuindo todo um mecanismo de verificação de tipos evitando assim que valores inválidos sejam entrados e possam causar situações inesperadas. Além disso, NetActeon traz a funcionalidade de modificar certos campos do cabeçalho IP destes datagramas, dando mais poder para o administrador da rede. Na questão desempenho, NetActeon se mostrou muito eficiente uma vez que utiliza SNPI e por isso adiciona um carga pequena ao processamento dos datagramas.

NetActeon está disponível para *download* no próprio *site* do projeto SNPI[25]. Além desta, muitas outras ferramentas devem estar disponíveis em breve se beneficiando assim do poderio alcançado com SNPI.

8 Comentários Finais

Foi apresentada neste trabalho uma abordagem para a criação de nós programáveis em dispositivos que compõem uma rede de comunicação. Tal abordagem, chamada de SNPI, é aplicável a uma arquitetura qualquer, desde que a mesma seja modular e tenha as interfaces entre suas camadas bem definidas para que a abordagem possa ser implantada corretamente e tenha funcionalidade.

A proposta se mostrou simples na sua concepção e de fácil implantação, desde que o sistema operacional do dispositivo e a arquitetura escolhida possuam algumas características e facilidades que possam deixar a implementação tão robusta quanto a abordagem. Tal proposta apresenta algumas vantagens, tais como:

- Extrema facilidade de implementação.
- Barateamento considerável do investimento em recursos computacionais.
- Alta flexibilidade para disponibilização de novos serviços.
- Total independência de grandes fabricantes de *softwares* para rede.
- Abertura franca do mercado desenvolvedor de *softwares* para redes.

Foi apresentada uma implementação onde todas as características da abordagem, tais como flexibilidade, dinamismo e robustez ficaram evidentes. A implementação foi feita para o sistema operacional Red Hat Linux 6.1 e 7.1, estando disponível para testes e colaborações. Em futuro bem próximo, será possível encontrar implementações desta abordagem para outros sistemas operacionais com o objetivo de demonstrar a viabilidade, robustez e flexibilidade da mesma.

Quanto ao aspecto desempenho, SNPI não adiciona nenhuma carga ao processamento dos datagramas. Na verdade, o desempenho vai estar dependente das aplicações e serviços que venham a se utilizar de SNPI. Mesmo assim, testes realizados com a aplicação NetActeon mostraram que muito pode ser feito através de SNPI sem que para isso seja preciso comprometer o desempenho do processo de transmissão e recepção de dados.

Um aspecto que não foi discutido, não por esquecimento, foi a questão da segurança. Infelizmente, é difícil ou até mesmo inviável analisar aspectos de segurança da abordagem, pois tal fator é extremamente dependente da implementação, do sistema operacional e dos mecanismos utilizados. Na implementação da interface SNPI para Linux, por exemplo, a segurança é garantida através das permissões de certos comandos utilizados para manipular uma janela programável, ou módulo, como **insmod** e **rmmod**. Com isso, só terão direitos de carregar as janelas programáveis aqueles que possuírem permissões suficientes para tal. Depois de carregadas as janelas programáveis, a segurança fica por conta do sistema operacional, da arquitetura de rede e do processo de comunicação entre a arquitetura e as janelas programáveis.

Alguns trabalhos futuros são interessantes para a abordagem e seriam de grande valia para complementar e suplementar este estudo. A criação de aplicações e serviços que se utilizassem de SNPI só viriam a confirmar a grande utilidade e poderio desta abordagem sendo de grande significado para a disseminação desta. No entanto, outras pesquisas podem ser realizadas contribuindo ainda mais para a melhoria da interface SNPI, como por exemplos, pesquisas que busquem criar ou até mesmo adaptar uma linguagem para tratamento de datagramas seria muito interessante e de grande utilidade para os administradores que não precisariam

escrever as janelas programáveis em linguagens de programação de baixo e médio nível.

Espera-se que SNPI tenha a repercussão que sua grande funcionalidade merece. Ainda existe muito trabalho a ser desenvolvido sobre esta abordagem. Implementações para os sistemas operacionais mais utilizados e poderosos logo devem estar disponíveis, no entanto, é interessante que outros pesquisadores trabalhem de forma parela no desenvolvimento de aplicações e serviços para esta abordagem ou até mesmo em mecanismos que maximizem o poder da interface SNPI disponibilizando para os usuários todas as ferramentas e facilidades esperadas de um grande projeto.

A SNPI x BSD Packet Filter

No ano de 1993 é apresentada à comunidade acadêmica, na conferência **Winter USENIX 1993,** uma proposta de uma nova arquitetura para captura de pacotes no nível do usuário intencionada a substituir antigas arquiteturas presentes em sistemas Unix e que datam ainda da década de 80 como NIT[31] e CSPF[16]. Tal proposta ficou conhecida como BPF[15]. As principais diferenças entre BPF e as outras arquiteturas é que a primeira se utiliza de mecanismos que garantem um melhor desempenho evitando que o processo de monitoramento se torne inconsistente com a situação real do dispositivo, no entanto, a tarefa continua a mesma, ou seja, transferir uma cópia dos pacotes para o modo usuário, possibilitando que aplicações de monitoramento possam se utilizar destas cópias para cálculos estatísticos ou análise dos conteúdos destes pacotes.

Alguns leitores que desconhecem o funcionamento interno da arquitetura BPF mas que a utilizam com freqüência podem estar questionando as diferenças entre SNPI e BPF. É importante mencionar, inicialmente, que SNPI é uma abordagem no campo de redes programáveis e ativas que preza pela simplicidade e busca adicionar dinamismo e programabilidade a nós outrora tidos como estáticos. Já BPF é uma arquitetura que realiza cópias de pacotes no nível do *kernel* e os envia para o nível do usuário para que possam ser utilizados por aplicações com características de monitoramento.

Para um melhor entendimento, será preciso compreender como funcionam ambas arquiteturas e para isso serão utilizadas duas figuras que mostram o funcionamento interno de cada uma.

Na figura 16 está especificado o funcionamento da arquitetura BPF. Quando um pacote chega a um dispositivo por meio da rede de comunicação este é, primei-

ramente, tratado pelo *driver* da interface que o recebeu. Depois, este pacote é passado para a pilha de protocolos correspondente sendo, momentos antes, realizada uma cópia do mesmo para que possa ser enviada para a arquitetura BPF. Desta forma o pacote e sua cópia seguirão caminhos diferentes mas processados de forma paralela[27]. O pacote seguirá seu percurso normal e não sofrerá nenhuma interferência a não ser um pequeno atraso resultante da duplicação. Já a cópia do pacote será analisado pelo BPF e, de acordo com filtros pré-determinados, será entregue à ferramentas de monitoramento para análises.

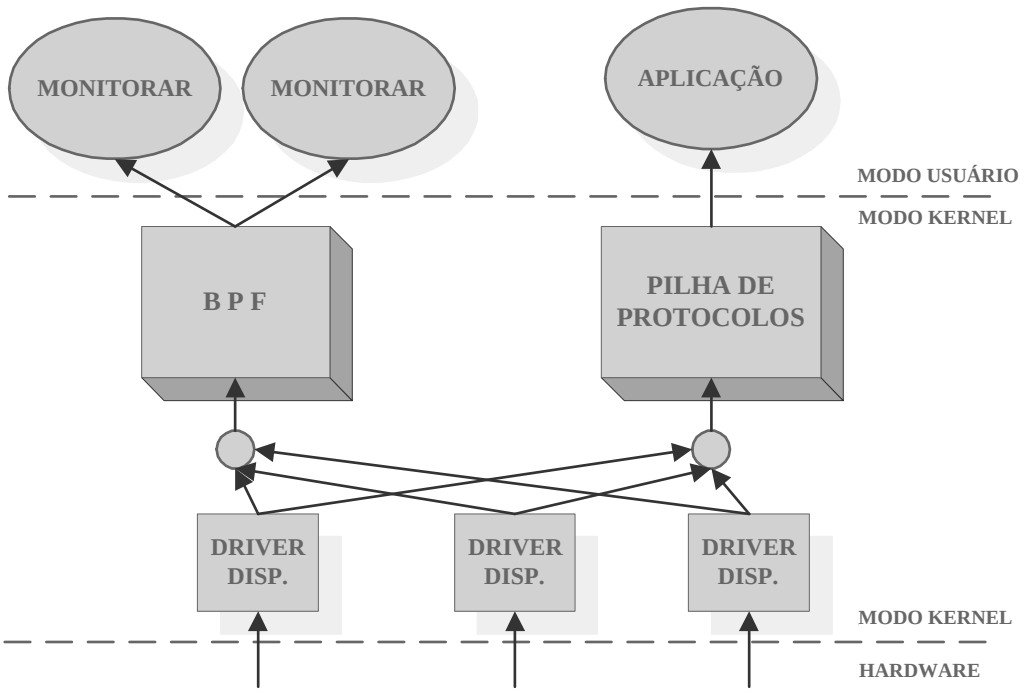

Figura 16: Arquitetura BPF

Com isso, podemos concluir que BPF não cria nenhum ambiente de programabilidade. Se um usuário realizar alguma modificação na cópia do pacote que lhe foi

[27]Lembrando que não estamos tratando com arquiteturas de processamento paralelo. O termo paralelo aqui empregado caracteriza a concorrência eficiente realizada por dispositivos atuais o que cria uma sensação de paralelismo.

entregue, tal ato não surtirá efeito visto que o pacote original já foi tratado pela pilha de protocolos e, possivelmente, nem existe mais.

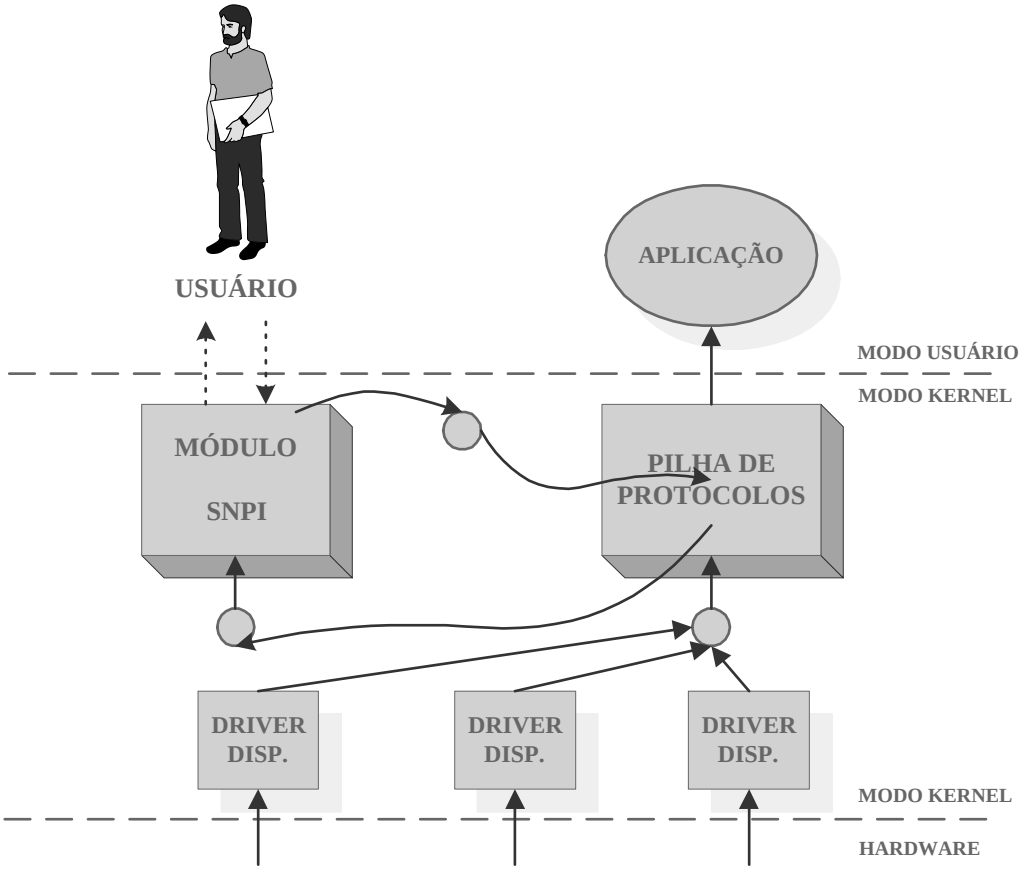

Figura 17: Arquitetura SNPI

Já na figura 17 pode-se ver o funcionamento da abordagem SNPI. Quando um pacote chega a um dispositivo o mesmo é tratado, primeiramente, pelo *driver* da interface que o recebe. Posteriormente, este pacote é passado para a pilha de protocolos correspondente e inicia-se seu tratamento. Durante este tratamento, se o pacote encontra algum *hook* no caminho, então este é desviado para um módulo desenvolvido pelo usuário que trata o pacote de acordo com suas necessidades podendo até mesmo alterá-lo. No momento em que o módulo terminar o tratamento,

o pacote retorna para a pilha e continua com seu percurso inicial.

Como pode ser visto, não existem cópias do pacote original o que leva à conclusão que alterações ocasionarão mudanças no comportamento do dispositivo e da rede de comunicação de dados como um todo.

Portanto, mesmo possuindo semelhanças e sendo completamente compreensíveis confusões entre SNPI e BPF, fica claro que ambas propostas atacam problemas diferentes e não podem, em nenhum momento, serem colocadas lado a lado durante quaisquer comparações.

Referências Bibliográficas

[1] ALEXANDER, D. S.; BRADEN, B.; GUNTER, C. A.; JACKSON A. W.; KEROMYTIS, A. D.; MIDEN, G. J.; WETHERALL, D. Active Network Encapsulation Protocol (ANEP). RFC Draft, 1997. Disponível em: <http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt>.

[2] ALEXANDER, D. S.; ARBAUGH, W. A.; KEROMYTIS, A. D.; SMITH, J. M. Performance Implications of Securing Active Networks. Technical Report MS-CIS-98-02, 1998.

[3] ALEXANDER, D. S.; ARBAUGH, W. A.; HICKS, M.; KAKKAR, P.; KEROMYTIS, A.; MOORE, J. T.; GUNTER, C. A.; NETTLES, S. M.; SMITH, J. M. The switchware active network architecture. IEEE Network Magazine, Special issue on Active and Controllable Networks, 2000.

[4] BHATTACHARJEE, S. et al. Commentaries on Active Networking and End-to-End Arguments. IEEE Network Magazine, Special issue on Active and Programmable Network, v. 12, n. 3, 1998.

[5] CAML. Caml Home Page. Disponível em <http://pauillac.inria.fr/caml/index-eng.html>.

[6] CAMPBELL, A. T; MEER, H. G.; KOUNAVIS, M. E.; MIKI, K.; VICENTE, J. B.; VILLELA, D. A Survey of Programmable Networks. ACM SIGCOMM Computer Communications Review, v. 29, n. 2, p. 7-23, 1999.

[7] HICKS, M. W.; KAKKAR, P.; MOORE, J. T.; GUNTER, C. A.; NETTLES, S. PLAN: A Packet Language for Active Networks. In: International Conference on Functional Programming, 1998. p. 86-93.

[8] HICKS, M. W.; MOORE, J. T.; ALEXANDER, D. S.; GUNTER, C. A.; NETTLES S. PLANet: An Active InterNetwork. In: IEEE INFOCOMM, p. 1124-1133, 1999.

[9] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. IEEE 802.3: IEEE Standards for Local Area Networks - Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. New York, 1985.

[10] JEFFREY, A.; WAKEMAN, I. A Survey on Semantic Techniques for Active Networks, 1997. Disponível em : <http://www.cogs.susx.ac.uk/users/ianw/papers/an-survey.ps.gz>.

[11] KEROMYTIS, A. D.; BLAZE, M.; IOANNIDIS, J.; SMITH, J. M. Firewalls in active networks. Technical Report, University of Pennsylvania, 1998.

[12] LAVIAN, T.; JAEGEER, R.; HOLLINGSWORTH, J. Open programmable architecture for java-enabled network devices. In: Proceedings of the Seventh IEEE Workshop on Hot Interconnects, 1999.

[13] LAZAR, A. A; MARCONCINI, F. Towards an Open API for ATM Switch Control, CTR Technical Report #441-96-07, 1996. Disponível em: <http://www.ctr.columbia.edu/comet/xbind/xbind.html>.

[14] MARSHALL, I.; COVACI, S.; VELTE, T.; JUHOLA, A.; PARKILLA S.; DONOHOE M. The Impact of Active Networks on Established Network Operators. In: IWAN, Lecture Notes in Computer Science, v.1653, p. 188-199, 1999.

[15] MCCANNE, S.; JACOBSON, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In: Winter USENIX 1993, 1993. p. 259-270.

[16] MOGUL, J.; RASHID, R.; ACCETTA, M. The Packet Filter: An Efficient Mechanism for User-Level Network Code.ACM Operating Systems Review, SIGOPS, v. 21, n. 5, p. 39-51, 1987.

[17] MURPHY, D. Building an Active Node on the Internet. Technical Report MIT/LCS/TR-723, 1997. 66p.

[18] MURPHY, S.; LEWIS, E.; PUGA, R.; WATSON, R.; YEE, R. Strong Security in Active Networks. In: IEEE OPENARCH 2001, 2001. p. 63-70.

[19] MUUSS, M.; SLATTERY T. ttcp: Test TCP. US Army Ballistics Research Lab (BRL), 1994. Disponível em: <http://www.pcausa.com/Utilities/pcattcp.htm>.

[20] NAUGHTON, P; SCHILDT, H. Java 1.1: The Complete Reference. 2. ed. Berkeley: Osborne McGraw-Hill, 1998. 1028p.

[21] O'MALLEY, S. W.; PETERSON, A. A. A Dynamic Network Architecture. In: ACM Transactions on Computer Systems, 1992. p. 110-143

[22] POMERANTZ, O. The Linux Kernel Module Programming Guide, v. 1.1.0, 1999. Disponível em: <http://www.linuxdoc.org/LDP/lkmpg/mpg.html>.

[23] PSOUNIS, K. Active Networks: Applications, Security, Safety, and Architectures. IEEE Communications Survey, 1999. p. 1-16.

[24] Red Hat Linux. Red Hat Software, Inc. Disponível em: <http://www.redhat.com>.

[25] ROCHA, C. A. C. SNPI: Simple Network Programming Interface, 2002. Disponível em: <http://www.lia.ufc.br/~crisamon/snpi/index.html>.

[26] SCHILDT, H. C++: The Complete Reference. 3. ed. Berkeley: Osborne McGraw-Hill, 1998. 1008p.

[27] SCHWARTZ, B. I. Introduction to Spanner: Assembly Language for the Smart Packets Project. BBN Technical Memorandum 1220, 1999.

[28] SCHWARTZ, B. I. Sprocket Language Description for the Smart Packets Project. BBN Technical Memorandum 1221, 1999.

[29] SCHWARTZ, B.; JACKSON, A. W.; STRAYER, T.; ZHOU, W.; ROCKWELL, R. D.; PARTRIDGE, C. Smart packets: applying active networks to network management. ACM Transactions on Computer Systems, v. 18, n. 1, p. 67-68, 2000.

[30] SCOTT, D. et al. A Secure Active Network Environment Architecture. IEEE Network Magazine, Special issue on Active and Programmable Network, v. 12, n. 3, 1998.

[31] SUN MICROSYSTEMS INC. NIT - Network Interface Tap. SunOS 4.1.1 Reference Manual 800-5480-10, 1990.

[32] TENNENHOUSE, D. L.; CLARK, D. D. Architectural Considerations for a New Generation of Protocols. In: ACM Sigcomm Symposium, 1990.

[33] TENNENHOUSE, D. L.; WETHERALL, D. J. Towards an Active Network Architecture. Computer Communication Review, v. 26, n. 2, 1996.

[34] TANENBAUM, A. S. Computer Networks. 3. ed. New Jersey: Prentice Hall, 1996. 814p.

[35] TOUCH, J.; HOTZ, S. The X-Bone. In: Third Global Internet Mini-Conference, 1998.

[36] VAN, V. C. A defense against address spoofing using active networks. Bachelor's Thesis, MIT, 1997.

[37] WANG, P. Y.; LAVIAN, T. Active Networking On A Programmable Networking Plataform. In: IEEE OPENARCH 2001, 2001. p. 95-103.

[38] WETHERALL, D. J.; GUTTAG, J.; TENNENHOUSE, D. L. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In: IEEE OPENARCH 1998, 1998. Disponível em: <ftp://ftp.tns.lcs.mit.edu/pub/papers/openarch98.ps.gz>.

[39] WETHERALL, D. J.; LEGEDZA, U.; GUTTAG, J. Introducing new internet services: Why and how. IEEE Network Magazine, Special issue on Active and Programmable Networks, 1998.

[40] WETHERALL, D. J. Service introduction in an active network. 1999. 157 f. PhD thesis, Dept. Electrical Engineering and Computer Science, MIT. Disponível em: <http://www.cs.washington.edu/homes/djw/papers/antsthesis.ps.gz.>.

[41] YEMENI, Y; SILVA, S. Towards Programmable Networks. In: IFIP/IEEE International Workshop on Distributed Sys-

tems Operations and Management, 1996. Disponível em: <http://www.cs.columbia.edu/~dasilva/pubs/dsom96.ps>.