Uma Abordagem para a Implantação de Nós Programáveis em Redes de Comunicação

Cris Amon C. da Rocha
crisamon@lia.ufc.br
Departamento de Computação
Universidade Federal do Ceará

Aécio Paiva Braga
aecio@ufc.br
Departamento de Computação
Universidade Federal do Ceará

José Neuman de Souza
neuman@ufc.br
Departamento de Computação
Universidade Federal do Ceará

Resumo

Desde o surgimento do conceito de redes programáveis que grandes esforços têm sido realizados para tornar as redes de computadores atuais 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 apresenta 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 e se utiliza do conceito de SNPI(Acrônimo de *Simple Network Programming Interface*), que consiste num conjunto de esqueletos de procedimentos e artifícios que possibilitam aos programadores e desenvolvedores de novos serviços o acesso aos mecanismos internos de tratamento do fluxo de dados, alterando a máquina de estados deste fluxo 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 (Simple Network Programming Interface), 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 access to internal mechanisms of data flow processing in the network nodes.

Palavras-chave: Redes Ativas, Redes Programáveis, Serviços de Rede.

1 Introdução

Tradicionalmente, uma rede de computadores é vista como um conjunto de módulos processadores interligados por um meio de comunicação. Estes módulos processadores possuem funções determinadas como roteamento, controle de tráfego, monitoramento, entre outras. Tal comportamento estático das redes tradicionais impossibilita a implantação de novos serviços à rede sem que haja uma substituição dos módulos processadores, ou parte deles.

A necessidade de implantação imediata de novos serviços como resposta às novas demandas do usuário e a reutilização da estrutura da rede existente, são os principais

fatores que impulsionam o surgimento de redes mais flexíveis e dinâmicas, as chamadas redes programáveis.

As redes programáveis têm o dever, portanto, de simplificar a implantação de novos serviços à rede. Para tal fim, se utilizam de estruturas bem definidas e de métodos de composição que facilitam a criação de novos serviços. Com o objetivo de tornar as redes mais programáveis surgiram duas escolas de pensamento. A primeira delas, denominada comunidade *Opensig*[1], defende a utilização de um conjunto de interfaces abertas de programação de rede que modelam o hardware de comunicação. Isso possibilita o acesso completo à roteadores e *switches*. De acordo com a comunidade *Opensig*, a abertura dos roteadores e *switches* através de interfaces possibilita o desenvolvimento de novos e distintos serviços e arquiteturas. Atualmente, o projeto 1520 [2] realizado pelo IEEE na área de interfaces de programação para redes está seguindo a abordagem da comunidade *Opensig* e procura padronizar tais interfaces para roteadores IP, *switches* ATM e redes móveis de telecomunicação. A próxima seção apresenta alguns projetos que se baseiam nesta filosofia de interfaces abertas de programação objetivando a abstração de recursos físicos.

A segunda escola de pensamento, denominada comunidade *Active Networks(AN)*[3], defende a implantação dinâmica de novos serviços. Este dinamismo vai muito além daquele proposto pela comunidade *Opensig*. Aqui, se faz uso de pacotes de rede que levam trechos de código que podem compor um novo serviço. A compilação de tais códigos durante o processo de operação da rede, a mudança no comportamento da rede baseado no conteúdo dos pacotes, a distinção entre pacotes de dados e pacotes ativos(Pacotes contendo trechos de código). Enfim, todas estas características tornam esta abordagem um tanto quanto mais dinâmica e poderosa do que a abordagem defendida pela comunidade *Opensig*, mas sua complexidade é muito superior. Tal linha de raciocínio só foi possível graças a novas tecnologias, como mobilidade de código, e ao avanço tecnológico nos equipamentos de rede(Roteadores e *Switches*)que podem suportar esta carga de processamento a mais sem degradar a rede como um todo.

As duas abordagens citadas acima possuem grandes méritos e poderosos artifícios para transformar as redes de computadores atuais em redes mais dinâmicas e programáveis. No entanto, ambas abordagens possuem semelhanças que podem comprometer a continuidade destas. Estamos nos referindo a complexidade de implantação e a dificuldade de padronização. Principalmente, a filosofia adotada pela comunidade *Active Networks(AN)* enfrenta tal problema devido à sua complexidade. É por tudo isso que neste artigo é apresentada uma abordagem poderosa e simples no que diz respeito à sua concepção. É apresentada também a *Simple Network Programming Interface*, ou simplesmente SNPI, que é um conjunto de esqueletos de procedimentos e artifícios que se utilizam de estruturas simples para ter acesso aos mecanismos internos do fluxo das unidades de dados dos nós da rede. Tal proposta tem a vantagem de ser facilmente implantada e permitir o aproveitamento de equipamentos já utilizados nas redes atuais, plataformas Linux servindo de roteadores, por exemplo.

Este artigo está dividido da seguintes forma: A seção 2 apresenta alguns projetos relacionados a redes programáveis, tanto na abordagem da comunidade *Opensig* como também na abordagem da comunidade *Active Networks(AN)*. A seção 3 apresenta uma abordagem simples e poderosa para se alcançar programabilidade na rede, aonde várias subseções são necessárias para melhor esquematizá-la. Uma vez que a abordagem apresentada poderá criar algumas interrogações quanto à sua possibilidade de implantação,

a seção 4 mostra aspectos práticos da abordagem e uma implementação da mesma na camada de inter-rede da pilha TCP/IP do sistema operacional Linux. Nesta mesma seção serão analisados pontos que apresentam-se como essenciais para sua implantação. E por fim, a seção 5 conclui o artigo mostrando algumas considerações finais e citando possíveis trabalhos futuros.

2 Trabalhos Relacionados

É importante a discussão de alguns projetos na área de redes programáveis com o intuito de apresentar as mais diferentes metodologias aplicadas e para, de certa forma, encaixar a abordagem proposta neste artigo no universo de tantas outras. Com tanta diversidade entre os projetos fica complexo tentar dividi-los pelas escolas *Opensing* e *Active Networks*, mesmo porque existem projetos que são uma mescla de características das duas escolas. Desta forma, são apresentados projetos que têm grande significação para o campo sem, no entanto, aplicar esforços desnecessários na tentativa de classificá-los segundo a escola de pensamento.

2.1 Smart Packets

Desenvolvido pela *University of Kansas*[4], este projeto se baseia no conceito de *smart packets*, que são pacotes especializados que carregam código, mais especificamente classes Java, e que trafegam por um ambiente IP programável.*Smart packets* propagam informações de estado na forma de objetos serializados e levam identificadores para sua autenticação. Um nó ativo da rede oferece um conjunto de abstrações e primitivas que podem ser acessadas pelos *smart packets*. Estes, por sua vez, são alocados em *threads* nos nós ativos e consomem recursos do nó para que seja feito o processamento desejado. Um ambiente experimental baseado no conceito de *smart packets* tem sido utilizado para demonstrar ganho de desempenho de versões modificadas dos protocolos HTTP e SMTP sobre os protocolos originais.

2.2 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*[5]. As interfaces são construídas para suportar a programabilidade dos planos de controle e gerenciamento em redes ATM. Outro aspecto importante neste projeto é o *xbind broadband kernel* que incorpora três abstrações de rede:

- Rede de banda larga
- Rede multimídia
- Rede de serviços

Tais abstrações, fornecidas pelo *xbind broadband kernel*, facilitam a criação de serviços variados. Existe um experimento baseado na arquitetura *xbind* que incorpora *switches ATM* dos mais diversos fabricantes.

2.3 ANTS

Projeto do MIT, *ANTS*[6] permite o emprego dinâmico de novos protocolos, tanto nos roteadores como em sistemas finais, sem a necessidade por coordenação e sem interações indesejadas entre protocolos co-existentes.

ANTS provê um ambiente de programação de rede para a construção de arquiteturas de redes programáveis baseadas em cápsulas. Estas, por sua vez, funcionam como unidades atômicas de programabilidade de rede suportando interfaces de processamento e de *forwarding*. Nós ativos executam as cápsulas e rotinas, mantém estado local e suportam serviços de distribuição de código para automatizar o emprego de novos serviços.

2.4 Switchware

Switchware[7] é um projeto desenvolvido na *University of Pennsylvania* que tenta balancear dois aspectos: flexibilidade de uma rede programável com requisitos de segurança da mesma. No nível do sistema operacional, um roteador ativo IP 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.

Pacotes ativos podem percorrer a rede modificando o seu comportamento. Estes pacotes são escritos em linguagem funcional (Caml[8] e PLAN[9]) e levam pequenos e leves programas que invocam rotinas oferecidas pelas extensões ativas. A abordagem apresentada pelo projeto *Switchware* permite aos desenvolvedores balancear aspectos de segurança e desempenho, já que este se utiliza de pesadas verificações nas extensões ativas e leves averiguações nos pacotes ativos.

2.5 Netscript

Netscript[10] é uma linguagem de programação e um ambiente para a construção de sistemas de rede desenvolvido na *University of Columbia*. Seus programas são organizados como agentes móveis que são enviados para sistemas remotos e executados sobre controle local ou remoto. A linguagem *Netscript* inclui construções e abstrações que facilitam o desenvolvimento de *softwares* de manipulação de tráfego. Mensagens dos protocolos são definidas e codificadas como objetos *Netscript* de alto nível ou num formato compatível com padrões existentes.

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* captura a programabilidade das funções dos nós da rede.

3 Apresentação da Abordagem

Os projetos citados na seção anterior possuem diferentes características, mas todos eles procuram de alguma forma criar poderosos ambientes que proporcionam níveis de programabilidade à rede. Infelizmente, todos os trabalhos citados até o momento possuem dois grandes problemas que dificultam sua implantação e perpetuação no cenário das redes.

O primeiro problema, e provavelmente o maior deles, é a complexidade que estes projetos imprimem. Iniciativas como *Smart Packets* e outras que se baseiam no conceito de pacotes ativos ou cápsulas, exigem uma mudança na estrutura das redes de computadores como também na forma de como os serviços são requisitados. Tal abordagem, que é a mesma defendida pela comunidade *Active Networks*, traz junto à ela outros problemas como novas políticas de segurança; novas políticas de manipulação das unidades de dados, 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. Mesmo aquelas iniciativas que buscam a padronização de interfaces que abstraem recursos de hardware, abordagem defendida pela comunidade *Opensig*, trazem problemas de complexidade pela dificuldade de abstrair e generalizar dispositivos tão complexos e distintos como roteadores, *switches*, *gateways* e outros. É certo que, estes níveis de complexidade são necessários para a criação de uma rede programável robusta e completa, mas esta mesma complexidade deve ser evitada numa primeira etapa.

O segundo problema, que pode ser colocado como uma conseqüência do problema citado acima mas que será tratado de forma separada devido à sua gravidade, é a dificuldade encontrada na padronização. Tal dificuldade de padronização é resultado direto da complexidade de abstrair os recursos de rede e das grandes diferenças que existem entre recursos da mesma natureza. Tudo isso cria um pessimismo quanto ao futuro das redes programáveis. Mesmo grandes empresas que dominam o mercado de roteadores e *switches* não investem tão alto na implantação de projetos, como aqueles citados na seção 2, se não houver uma padronização consistente.

É pensando nestes dois problemas citados acima que este artigo 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.

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 uma interface, SNPI, e o uso de janelas programáveis para se atingir níveis de programabilidade na rede. Esta interface altera o fluxo normal das unidades de dados da arquitetura à qual ela é implantada, fazendo que estas unidades atravessem as janelas programáveis para só depois continuarem com o seu fluxo normal. Para melhor entendimento, é preciso especificar do que se constitui esta interface(SNPI) e definir o que vem a ser uma janela programável.

3.1 Estruturas da Abordagem

3.1.1 SNPI - Simple Network Programming Interface

Apresentamos SNPI como sendo um conjunto de esqueletos de procedimentos e artifícios, que conseguem alterar o fluxo das unidades de dados de uma arquitetura qualquer para o meio exterior. Entenda meio exterior como um ambiente qualquer aonde o desenvolvedor/programador possa ter acesso a este fluxo. O ambiente interno dos mecanismos de uma arquitetura qualquer, por exemplo, não pode ser visto como um meio exterior, já que

o desenvolvedor de novos serviços não pode ter acesso a este ambiente durante o processo de fluxo das unidades de dados .

Não há como especificar rigidamente como deve ser a implementação de uma SNPI, pois, como será visto na próxima seção, tal implementação depende do ambiente em que a abordagem proposta é desenvolvida e dos recursos que o sistema operacional do dispositivo disponibiliza.

3.1.2 Janela Programável

Uma janela programável é definida como um meio exterior, conceito apresentado na subseção anterior, que permite o acesso, através dos esqueletos de procedimentos da SNPI, ao mínimo de estrutura interna da arquitetura principal, criando assim um ambiente de programação na rede.

Portanto, uma janela programável é o ambiente aonde o desenvolvedor vai ter acesso às estruturas da arquitetura para que possa implantar os novos serviços desejados. A disponibilidade das estruturas depende de como é implementada a interface SNPI, apresentada na subseção anterior. Pode estar acessível dentro da janela somente o cabeçalho de uma unidade de dados da arquitetura, como pode ser estendido o poder do desenvolvedor disponibilizando estruturas internas mais significantes.

A programabilidade dentro das janelas programáveis pode acontecer de duas formas. Definindo e disponibilizando uma linguagem de alto nível que crie facilidades para o preenchimento dos esqueletos disponibilizados pela SNPI e que seja limitada dentro do escopo desejado, ou permite-se que o desenvolvedor se utilize das linguagens de baixo nível(C/C++, Assembly, etc.) que, por ventura, construíram a própria arquitetura principal . Esta última possibilidade dispensa a especificação da linguagem, só necessitando da especificação das estruturas internas disponibilizadas, e fornece ao desenvolvedor 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 desenvolvedor tornam-se procedimentos semelhantes àqueles que são internos da implementação da arquitetura, com a diferença de estarem localizados num ambiente externo.

3.2 Especificação da Abordagem

Como mencionado anteriormente, a abordagem em questão é diretamente dependente de uma arquitetura existente. É possível de ser implantada sobre 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.

A abordagem proposta consiste na criação de uma interface SNPI que permita o acesso à determinadas estruturas internas da arquitetura escolhida e que altere o fluxo das unidades de dados para as janelas programáveis. Na figura 1 é mostrada uma arquitetura qualquer X, modular e com interfaces entre suas camadas bem definidas, aonde foi aplicada a abordagem em questão, que por sua vez é composta pelas janelas programáveis e pela interface SNPI. A arquitetura X é composta de N camadas e, neste exemplo, existem "ganchos"da camada 2 até a camada N(Na figura 1, para melhor visualização, somente "ganchos"na camada 2 são mostrados), significando que o fluxo normal de unidades de dados será desviado para as janelas programáveis N - 1 vezes em cada direção. Lembrando

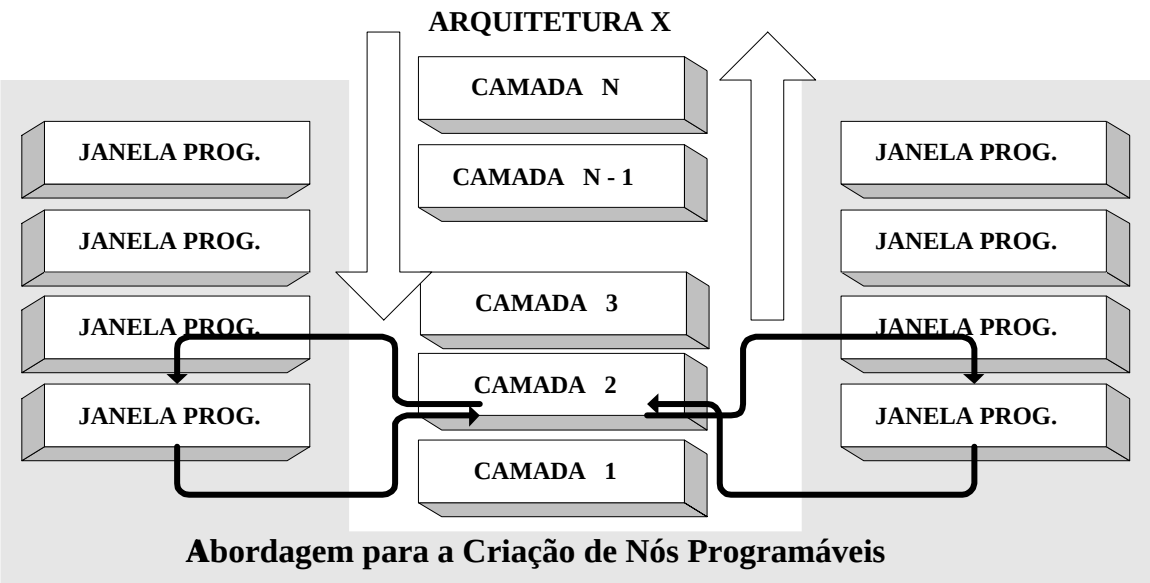

Figura 1: Abordagem proposta aplicada a uma arquitetura X.

que o fluxo de dados numa arquitetura é bidirecional e por esta razão é que existem dois "ganchos"por camada.

Tais "ganchos"são os artifícios que juntamente com os esqueletos de procedimentos compõem a interface SNPI. Estes artifícios serão tratados pelo nome de *hooks* no restante deste artigo.

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 de subida das unidades de dados 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, este fluxo será, primeiramente, desviado para o correspondente esqueleto da interface SNPI e poderá ser utilizado numa janela programável. Esta janela programável fará o tratamento em cima das estruturas que lhe são disponibilizadas e somente depois é que o fluxo retorna ao seu curso normal tendo a unidade de dados tratada pela camada M. Já no caso do fluxo de descida das unidades de dados ocorre diferente. O fluxo só será desviado depois que a unidade de dados tiver sido tratada pela camada correspondente.

A escolha dos locais de implantação dos *hooks* dentro das camadas foi baseada na análise dos serviços possíveis de serem implementados através da abordagem proposta. A maioria dos serviços que observam o fluxo de subida dos dados são serviços que buscam filtrar, analisar, modificar ou descartar as unidades de dados para que elas não continuem seu caminho na arquitetura. Além do mais, todas as informações que um serviço pode requerer já estão presentes, que são: o cabeçalho e os dados da unidade de dados daquela camada. Por outro lado, serviços que observam o fluxo de descida precisam, na sua grande maioria, esperar que seja construída a unidade de dados, cabeçalho e dados, daquela camada para tomar suas decisões.

Outro aspecto muito importante diz respeito à necessidade de *hooks* numa camada. Embora a figura 1 pressuponha a existência de *hooks* da camada 2 até a camada N, é evidente que esse aspecto é flexível e inteiramente dependente de quem estiver implan-

tando a abordagem. Se tomarmos a arquitetura TCP/IP como exemplo, a existência de *hooks* na camada de transporte seria importante para as estações finais mas certamente desnecessária para as estações intermediárias que só implementam até a camada de inter-rede.

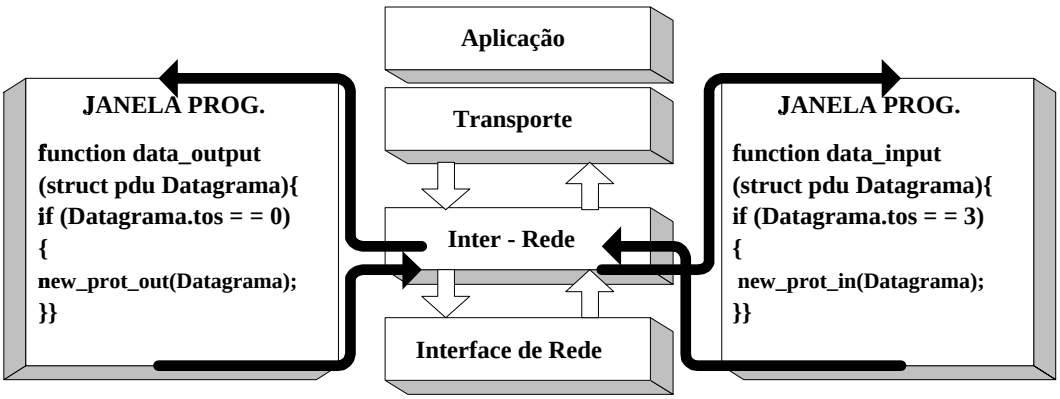

Figura 2: Abordagem aplicada à arquitetura TCP/IP.

Para melhor visualizar a abordagem proposta neste artigo, faz-se necessário o uso de um exemplo apresentado na figura 2. Este exemplo facilitará a compreensão da implementação feita por nosso grupo de pesquisa e que é descrita na próxima seção. Na figura 2 é mostrada a arquitetura TCP/IP. Foram colocados *hooks* somente na camada de inter-rede. Portanto, são dois *hooks*: um para o fluxo de subida e outro para o fluxo de descida. 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 através do esqueleto do procedimento correspondente ao *hook* que, por sua vez, recebe tais estruturas como argumentos. No interior da janela programável o desenvolvedor pode implementar o corpo deste esqueleto para alcançar seus objetivos. Depois, o fluxo é retornado à camada de inter-rede que realiza seu tratamento normal. Algo muito parecido ocorre no fluxo de descida, 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.

Finalizando esta subseção, 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 por intermédio dos esqueletos de procedimentos. A comparação das máquinas de estados é mostrada na figura 3.

3.3 Análise da Abordagem Proposta

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 resolve o problema mencionado anteriormente que era a complexidade na implantação das abordagens.

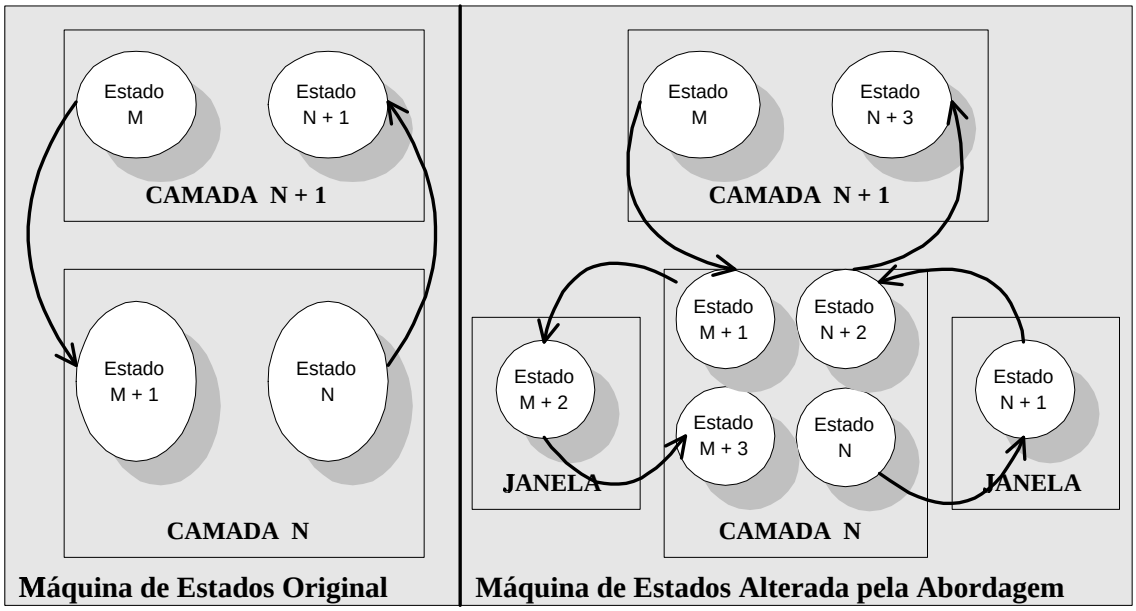

Figura 3: Comparação das máquinas de estados.

Interrogações podem aparecer no que diz respeito à implantação da abordagem. A criação dos *hooks*, dos esqueletos e das janelas programáveis vai depender da flexibilidade e de particularidades do sistema operacional do dispositivo e de como a arquitetura principal foi implementada, como será visto na próxima seção.

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 no interior de janelas programáveis com a arquitetura à qual a abordagem foi aplicada. Desta forma, é possível a criação de novos serviços somente com o uso das janelas programáveis, *hooks* e esqueletos de procedimentos. Porém, estes novos serviços ficam dependentes da arquitetura em questão. Mesmo assim, este problema de dependência não compromete a abordagem, já que a arquitetura TCP/IP é amplamente utilizada. 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 de nossa abordagem.

Desta forma, será que essa citada dependência é de fato tão negativa, diante da simplicidade da implementação?

4 Aspectos Práticos - Uma Implementação

A implantação da abordagem proposta vai ser dependente dos vários mecanismos e facilidades que o sistema operacional que governa o dispositivo em questão pode disponibilizar e da forma que foi implementada a arquitetura escolhida.

Antes de mostrar uma implementação baseada na abordagem proposta neste artigo, faz-se necessária a menção de alguns aspectos que são considerados essenciais para uma adequada implantação. O desrespeito a algum destes aspectos pode denegrir a abordagem ou mascará-la de tal forma que sua flexibilidade e seu dinamismo tornarão-se diminutos.

4.1 Aspectos Essenciais

- A implantação da abordagem deve se utilizar da implementação de uma arquitetura modular e com interfaces bem definidas entre suas camadas.
- As estruturas da abordagem, mencionadas na subseção 2.1, devem estar presentes e serem respeitadas durante a implantação da mesma.
- Não existe especificação de quais estruturas internas da arquitetura devem ser passadas pelos *hooks*. Mas tais estruturas devem estar acessíveis nos esqueletos de procedimentos e estes, por sua vez, devem ser preenchidos nas próprias janelas programáveis.
- Não há necessidade do desenvolvedor de novos serviços 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 e caso seja usada uma linguagem própria para a programação dos esqueletos, que esta seja apresentada e especificada.
- Não é desejado que ocorra recompilações no sistema operacional do dispositivo e nem na implementação da arquitetura para que os serviços, construídos nas janelas programáveis, possam estar disponíveis.

4.2 Uma Implementação

Respeitando os aspectos mencionados a pouco e a abordagem proposta neste artigo, apresentamos uma implementação feita na camada de inter-rede da pilha TCP/IP do sistema operacional *Red Hat Linux - versão 6.1*[11]. O sistema operacional Linux foi escolhido por diversas razões, aonde as principais são:

- A necessidade de se ter acesso à pilha TCP/IP para que fossem implementados os *hooks* no seu interior. E tal exigência foi alcançada, já que a pilha TCP/IP no Linux é implementada a nível de kernel e este, por sua vez, possui seu código fonte acessível por qualquer desenvolvedor.
- O poder de programação que o Linux possui frente a outros sistemas operacionais. 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 do uso de módulos que a implementação se tornou flexível e tão poderosa quanto a abordagem.

Um módulo é, em suma, uma parte do kernel que não é carregada diretamente. Um desenvolvedor qualquer pode escrever um módulo, depois compila-lo e, então, carregá-lo no kernel em tempo de operação.

4.2.1 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 as estruturas que são utilizadas na implementação.

1. Os *hooks* foram implementados no código fonte da camada de inter-rede da pilha TCP/IP como ponteiros para funções. A estes ponteiros é associado, de início, o valor *NULL*.

2. Os esqueletos de procedimentos foram implementados como funções dentro de um módulo e que possuem seu corpo vazio à espera da implementação de novos serviços.

3. 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.

4.2.2 Detalhamento da Implementação

Já que os principais aspectos da implementação já foram mostrados, será descrito brevemente como ocorreu a implementação da SNPI e a criação das janelas programáveis no *Red Hat Linux - versão 6.1* .

Primeiramente, foram declarados no arquivo **snpi.c** dois ponteiros para funções inteiras.

```
/* This function's pointer will be called when a packet is received */
int (*snpi_input_call)(struct sk_buff *)=NULL;

/* This function's pointer will be called when a packet is sent */
int (*snpi_output_call)(struct sk_buff *)=NULL;
```

Depois, mais precisamente nos arquivos **ip_input.c** e **ip_output.c**, foram inseridas chamadas para os ponteiros declarados anteriormente. Estas chamadas alteram o fluxo das estruturas **sk_buff** que representam o Datagrama IP.

```
   /* 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 */
```

Todo o resto da implementação da SNPI no interior do kernel do Linux é dispensável de ser apresentado, já que não diz respeito à abordagem. 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 encontrada para permitir que os desenvolvedores de

novos serviços não se preocupem com particularidades como a de liberar a área de memória ocupada pelo Datagrama IP, caso ele considere que este 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`.

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

A janela programável é um módulo aonde está declarado que existem dois ponteiros para funções externos ao módulo. A função que é executada durante a carga do módulo, `int init_module()`, atribui os endereços dos esqueletos para os 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 correspondentes.

```
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 4. Quando o Linux é iniciado a pilha TCP/IP já se torna ativa. 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 desenvolvedor de novos serviços compila o mesmo e carrega-o no interior do kernel através do comando `insmod nome_do_modulo`. Feito isto, o módulo modificará os valores dos ponteiros de *NULL* para os endereços dos respectivos esqueletos que tiveram seus corpos preenchidos pelo desenvolvedor. A partir deste momento, o fluxo de Datagramas IP será desviado e tratado de acordo com aquilo que foi escrito pelo desenvolvedor.

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.

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*. 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. Vale ressaltar que a programação no interior dos módulos[12] ocorre da mesma forma que no kernel.

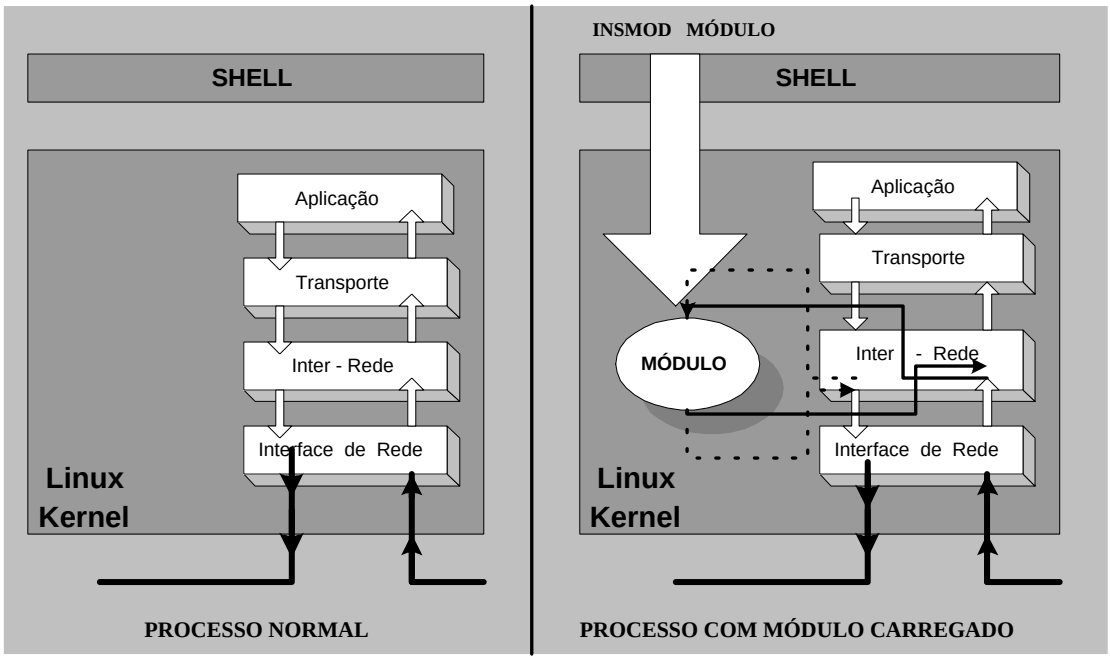

Figura 4: Situação do fluxo de dados antes e depois da carga do módulo

4.3 Análise da Implementação

Mostrados os detalhes da implementação, pode-se concluir que a abordagem para a criação de nós programáveis proposta neste artigo foi reproduzida com fidelidade. Todo o poder de programabilidade de rede que a abordagem alcança foi demonstrado pela 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 o grupo de pesquisa não sentiu necessidade de implementar nas camadas restantes.

Torna-se evidente que a implementação num determinado sistema operacional, ou numa pilha de protocolos, vai ficar dependente das facilidades que este/esta oferece. O uso de módulos, facilidade encontrada no Linux, permitiu recriar a alta flexibilidade e dinamismo da abordagem. Sendo assim, sistemas operacionais não tão robustos dificultam, e até mesmo impossibilitam, a implantação da mesma. Assim como, provavelmente, dificultariam o desenvolvimento de trabalhos como os apresentados na sessão 2.

A implementação está disponível para averiguações, testes e colaborações no *site* do projeto[13]. A mesma está disponível na forma de um arquivo *gzipped* que contém: um patch para aplicar os *hooks* ao kernel; o módulo que representa a janela programável e no interior do mesmo estão presentes os esqueletos prontos para serem preenchidos. No mesmo local se encontra uma cópia deste artigo no formato **ps** e **pdf**.

5 Comentários Finais

Foi apresentada neste artigo uma abordagem para a criação de nós programáveis nos dispositivos que compõem uma rede de comunicação. Tal abordagem 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.

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 aonde 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 está 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 e robustez da mesma.

Referências

[1] Open Signaling Working Group. http://comet.columbia.edu/opensig.

[2] J. Biswas, et al., "The IEEE P1520 Standards Iniative for Programmable Network Interfaces". IEEE Communications Magazine, 1998.

[3] DARPA Active Network Program. http://www.darpa.mil/ito/research/anets .

[4] A.B. Kulkarni, A. Gopinath, A. Nagarajan, F. Wahhab, G.J. Minden, H. Pindi, R. Hill, Y. Wijata. "Implementation of a Prototype Active Networks". First IEEE OPENARCH, 1998.

[5] XBind code. http://comet.columbia.edu/xbind .

[6] D. Tennenhouse, D. Wetherall, J. Guttag. "ANTS: A Toolkit for building and Dynamically Deploying Network Protocols". First IEEE OPENARCH, 1998.

[7] A.Keromytis, D.S Alexander, J.M. Smith, J.T. Moore, M.A. Hicks, P. Kakkar, S.M. Nettles, W.A Arbaugh. "The Switchware Active Network Architecture". Edição Especial do IEEE Magazine, 1998.

[8] Caml Language. http://caml.inria.fr .

[9] M. Hicks, et al. "PLAN: A Packet Language for Active Network". ICFP'98, 1998.

[10] S. da Silva, Y. Yemini. "Towards Programmable Network". International Workshop on Distributed Systems, 1996.

[11] Red Hat Linux. http://www.redhat.com .

[12] O. Pomerantz. Linux Kernel Module Programming Guide, 1999.

[13] SNPI Project Web Site. http://www.lia.ufc.br/~crisamon/snpi .