Estudo sobre armazenagem eficiente de tabela de rotas em Network Processors

Marco Caetano Lazarine Bontorin

5 de março de 2009

1 Introdução

A *Internet* é uma rede de redes de tecnologias distintas. O protocolo IP fornece um conjunto de regras para que as máquinas possam se comunicar usando endereços definidos conforme o *Internet Protocol*. Quando alguém deseja conectar suas máquinas à *Internet*, deve contratar serviço de um provedor de acesso à *Internet* (*Internet Service Provider (ISP)*). Esse provedor de acesso fornece um endereço ou um conjunto de endereços IP ao contratante, para serem usados no endereçamento de suas máquinas e dessa forma elas podem ser endereçadas na *Internet*.

Um Sistema Autônomo (*Autonomous System (AS)*) é uma rede ou um conjunto de redes sob o controle de uma única entidade administrativa. Quando um Sistema Autônomo necessita enviar informações para as máquinas que estão sob a administração de outro AS, estes devem estabelecer contratos que permitam a sua comunicação e criar a infra-estrutura necessária. Geralmente os ISPs são Sistemas Autônomos que permitem a utilização de sua infra-estrutura para transmissão de pacotes de usuários que querem acesso à *Internet*. Isso é feito mediante um contrato do usuário com um ISP, no qual o ISP estabelece os preços que devem ser pagos pelos seus serviços. A partir do momento que o usuário se conecta em um ISP e recebe um endereço IP, ele faz parte da rede daquele ISP. Portanto, para que uma máquina possa enviar mensagens para uma outra máquina qualquer, esta deve estar conectada ao mesmo ISP (fazer parte da rede desse ISP) que a nossa, ou em outro que seja alcançável a partir dele. A Figura 1 ilustra um conjunto de Sistemas Autônomos conectados entre si. Os dispositivos que interconectam as redes internas de um AS, bem como os Sistemas Autônomos entre si são comutadores com funções especializadas, chamados de roteadores.

Um roteador é um computador especializado que possui interfaces de entrada e saída, podendo através delas interconetar duas ou mais redes. A tarefa básica de um roteador é garantir que os pacotes que passam através dele não vão onde não deveriam ir e assegurar que eles atingem o destino desejado. Para alcançar esse objetivo, os roteadores devem implementar internamente pelo menos essas funções. Para que os roteadores possam realizar tais funções, eles armazenam internamente uma tabela de roteamento, em que cada elemento associa um endereço IP de destino (ou faixa de endereços) a uma porta de saída.

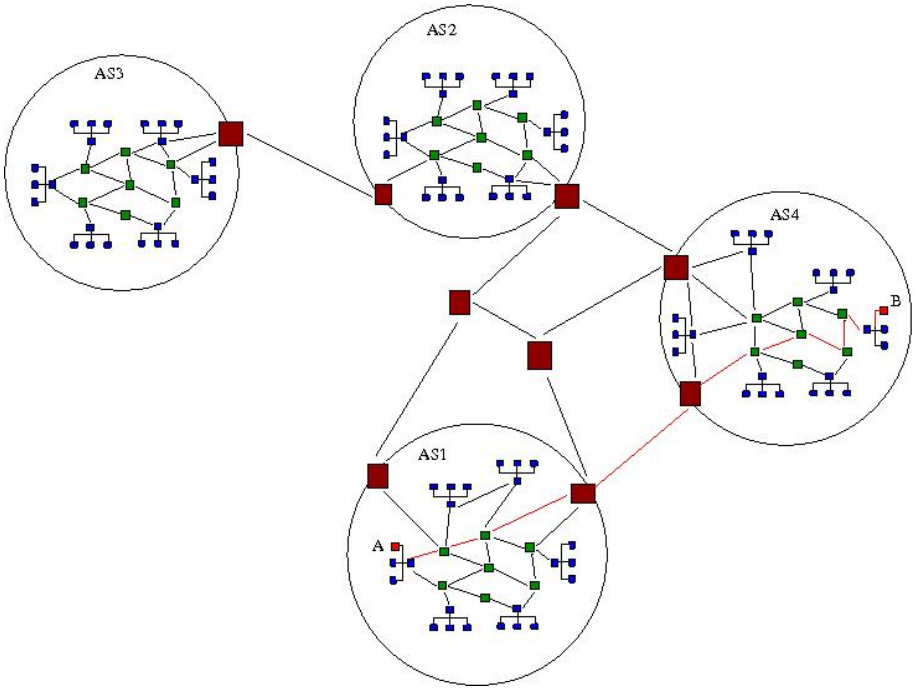

Figura 1: Rede que interliga Sistemas Autônomos.

Dessa forma, o roteador sabe para onde encaminhar cada pacote que recebe, comparando seu endereço de destino com os elementos de sua tabela de roteamento. Para construir e manter atualizada essa tabela são utilizados os protocolos de roteamento.

Com o aumento na largura de banda nas redes, o tempo alocado para a tarefa de procurar o destino de um pacote na tabela de roteamento tem se tornado cada vez menor. Portanto, foram desenvolvidos vários algoritmos e estruturas de *hardware* para acelerar esse processo. Uma classe interessante de estruturas são as *Content Addressable Memories (CAMs)*, nas quais o endereço de destino no pacote é comparado com todos os elementos da tabela em paralelo. Uma das várias arquiteturas de CAMs propostas é a *Ternary-CAM (TCAM)*. A principal característica da TCAM é que cada célula de memória pode armazenar não apenas '0's e '1's, mas também 'X' (*"don't care"*). A Figura 2 ilustra uma forma de organização de uma TCAM na implementação de uma tabela de roteamento, segundo [4].

Com o surgimento de novas tecnologias de rede, a transferência de informações tem se tornado cada vez mais rápida tanto em redes locais como nas redes de longa distância. Os diferentes tipos de dados que são enviados pela *Internet*, tais como voz e vídeo, determinam requisitos de largura de banda e outros serviços diferenciados. Para atender a essa diversidade, existem protocolos que permitem uma definição de diferentes prioridades de transmissão de pacotes na rede. O contínuo aumento na velocidade de transmissão, bem como a crescente demanda por serviços diferenciados fez com que a indústria passasse a fornecer roteadores que sejam capazes de suportar essa nova realidade, com processamento diferenciado para serviços distintos, ao mesmo tempo que garante que a largura de banda não será comprometida. Esse processamento adicional deve ser feito da forma mais

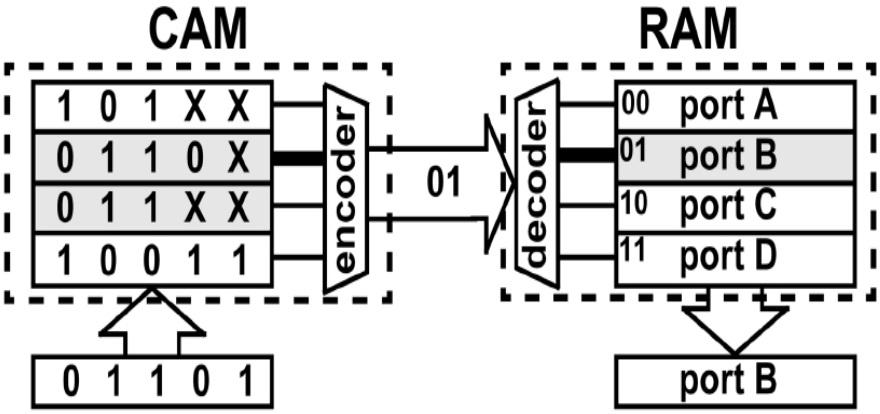

Figura 2: Uso de uma TCAM na implementação de uma tabela de roteamento.

rápida possível de modo que o tempo acrescentado pelos roteadores da origem ao destino de um pacote não seja signficativo. Os processadores comuns utilizados nos computadores de mesa podiam ser utilizados inicialmente para o processamento simples de pacotes em redes de baixa velocidade, mas agora com essa nova realidade, eles não conseguem garantir a banda necessária. Para fazer frente a essa situação, foram desenvolvidos os *Network Processors (NPs)*, que são processadores especializados no processamento de pacotes, que além de dar suporte às funções básicas de um roteador pode desempenhar também funções de garantia de qualidade de serviço.

Neste trabalho foram pesquisados diferentes NPs, com atenção especial ao IBM PowerNP NP4GS3 [8]. Dentre seus diversos blocos funcionais, foi escrito um simulador do Hardware Classifier (HC), que faz o *parsing* dos pacotes e disponibiliza os dados para processamento; e do *Tree Search Engine (TSE)*, que é um coprocessador que fornece operações de *hardware* para busca em árvores. Para analisar o comportamento de uma árvore de prefixos (faixa de endereços) e simular o TSE foi implementada uma versão simplificada de uma TCAM que foi projetada especificamente para aplicação em roteamento na *Internet* [6]. Para tráfego de entrada do programa foi utilizado um *log* de pacotes que atravessaram um roteador real.

2 Contexto

A tarefa desempenhada pelos roteadores na *Internet* é de grande importância porque o seu correto funcionamento garante que cada pacote chega ao seu destino, no tempo em que deveria chegar. Além disso, muitas outras funções têm sido agregadas aos roteadores, tais como gerenciamento de conexões à servidores e modelagem de tráfego (*traffic shaping*).

2.1 Definição de Conceitos

2.1.1 Endereços na *Internet*

Os métodos de endereçamento do *Internet Prococol (IP)* têm variado nos últimos anos e isso muito se deve ao reduzido número de endereços distintos definidos pelo *Internet Protocol*. Primeiramente utilizavam-se as classes pré definidas de endereços em que uma parte endereça a rede (*netid*) e segunda endereça as máquinas da rede (*hostid*). Essas classes são: 1) A, com netid de 8 bits e hostid de 24 bits, 2) B, com netid de 16 bits e hostid de 16 bits, 3) C, com netid de 24 bits e hostid de 8 bits, 4) D, usada para *multicast* e 5) E, reservada para usos futuros. As classes são identificadas examinando os 5 bits mais significativos do endereço IP.

Com o crescimento da *Internet*, os endereços disponíveis foram todos sendo ocupados. Isso motivou a criação de métodos que permitissem uma melhor utilização dos endereços existentes. Um dos meios utilizados para "economizar" endereços, foi o desenvolvimento do protocolo *Network Address Translation (NAT)*.

O NAT é implementado em muitos roteadores (*Gateways*) que conectam redes locais à *Internet*. As máquinas utilizam endereços IP reservados na comunicação na rede local, e o NAT evita que estas tenham endereços reais na *Internet*. Somente o *Gateway* da rede precisa de um endereço e as demais máquinas se conectam através dele. Sempre que um pacote passa pelo *Gateway*, este realiza a tradução de endereços modificando o cabeçalho do pacote e só então encaminhando o pacote. Isso é implementado através de uma tabela, em que cada elemento contém um endereço de origem local, uma porta de origem local, e uma porta de origem virtual. Sempre que um pacote passar através do *Gateway*, se o endereço IP de origem não está na tabela, é criado um elemento com o endereço IP de origem, a porta de origem e um número de porta virtual que é atribuído pelo *Gateway*. Caso contrário, o endereço IP de origem no pacote é trocado pelo IP do *Gateway* e a porta de origem é trocada pela porta virtual. Quando uma resposta chega para um desses pacotes, ela chega endereçada ao roteador e este identifica para qual máquina mandar o pacote através do número de porta virtual que consta nesse pacote, que é então usado para encontrar o endereço local correspondente na tabela. Através do NAT qualquer rede local pode utilizar os mesmos endereços IP locais para suas máquinas, pois eles são traduzidos automaticamente ao atravessar o *Gateway*, dessa forma contribuindo para que um número maior de endereços IP continue disponível.

Com o passar do tempo, foi constatado que a utilização das classes pré-definidas de endereços desperdiçava grande quantidade de endereços. Isso acontecia porque um subconjunto dos *hostids* não era utilizado e esses endereços não poderiam ser utilizados por outra instituição, pois uma vez que uma classe era alocada para uma instituição, todos os endereços de máquinas daquela classe estavam reservados. Para resolver esse problema e utilizar de forma mais eficiente os endereços IP, foi criado o *Classless Inter-Domain Routing (CIDR)*. No CIDR não há classes pré-definidas e qualquer tamanho de prefixo (*netid*) pode ser alocado. A representação de um endereço na notação CIDR é um endereço IP

normal seguido de um número inteiro representando o número de bits do endereço que são usados como *netid*. Com o CIDR, se uma organização precisa endereçar 12 máquinas ela não tem que solicitar uma faixa de endereços classe C, dos quais 242 [1] endereços ficariam sem uso, ela pode solicitar uma faixa de endereços com prefixo /28, no qual somente 2 endereços ($2^4 - (12 + 2) = 2$) não seriam utilizados.

Quando um pacote é aceito por um roteador, o endereço de destino é utilizado como chave de pesquisa na tabela. O endereço de destino pode casar com vários elementos da tabela, como por exemplo, o endereço 128.96.34.55 casaria com os prefixos 128.96.0.0/16 e com 129.96.34.0/24, mas a informação associada ao segundo deveria ser retornada pois ele tem um prefixo maior, mais específico e portanto a rota correta. Uma das tarefas que devem ser realizadas em uma busca na tabela de rotamento é a determinação do elemento que tem o maior prefixo dentre os elementos que casaram com a chave (*Longest Prefix Match (LPM)*).

2.1.2 Principais Tarefas na Implementação dos Protocolos

Niraj Shah definiu algumas tarefas importantes, nas quais qualquer aplicação de rede pode ser decomposta [7]:

- Casamento de padrão: é o processo de casar bits de um pacote com uma expressão regular ou um padrão de bits pré-estabelecido. Isso é utilizado sempre que existe uma tabela no roteador e se deseja procurar por um elemento que case com um determinado campo de um pacote. Isto pode ser observado na procura de VCIs/VPIs na comutação ATM ou na procura do próximo nó através do endereço IP do destino.

- Busca (*Lookup*): é a tarefa de procurar informações associadas a um determinado elemento em uma tabela que casem com uma chave de busca. Na maioria dos casos é utilizada juntamente com o casamento de padrão e uma vez que um elemento casa com a chave, a informação associada é retornada. Por exemplo, a busca pelo próximo nó (*next hop*) com a chave sendo o endereço IP de destino.

- Computação: quando alguma computação é feita sobre o pacote, causando modificações no conteúdo. Exemplos são o cálculo do CRC, encriptação e desencriptação dos dados.

- Manipulação: qualquer função que modifique o cabeçalho do pacote, como por exemplo decrementar o campo *Time To Live (TTL)* ou adicionar novos campos ao cabeçalho.

- Gerenciamento de Filas: são todas as ações que envolvem entrada e saída de pacotes, tais como armazenamento e tarefas de qualidade de serviço.

- Processamento de Controle: são funções que são executadas eventualmente. Podemos citar, por exemplo, atualizações de tabela que são iniciadas somente quando são

[1] $2^8 - (12 + 2) = 256 - 14 = 242$, onde o "2" contabilza os endereços de rede e *broadcast*

feitas trocas de pacotes de controle que implicam em uma alteração, ou quando algum temporizador associado a um elemento da tabela de roteamento expira e aquele elemento deve ser removido.

Essas tarefas são mapeadas para os NPs da forma em que o fabricante achar melhor para atender o seu mercado alvo, com circuitos desenhados especificamente para cada função, ou um misto de circuitos específicos e processadores de uso geral, para tornar o projeto mais flexível tendo em vista a constante evolução nas tecnologias.

2.1.3 Conceitos Importantes nos *Network Processors*

Segundo Allen at al. [8] as arquiteturas de sistemas com *Network Processors* podem ser divididas em dois modelos gerais: *Run to completion (RTC)* e em *pipeline*. No modelo RTC, que é baseado na arquitetura *Symmetric Multiprocessor (SMP)*, existem várias CPUs que compartilham a mesma memória. Uma *thread* em execução pode acessar todos os recursos compartilhados, tais como tabelas armazenadas em uma memória interna ou externa ao processador. Para que isso ocorra, há dispositivos (árbitros) que controlam o acesso a esses recursos fazendo a sincronização entre as várias *threads*. Nesse modelo, as instruções do código dessas *threads* executam todas as tarefas do processamento de um pacote. Uma vez que sua execução termina, o resultado do processamento é armazenado em uma fila de saída. No modelo em *pipeline*, o processamento dos pacotes é particionado em tarefas menores e cada uma delas é realizada por um processador. Esses processadores estão dispostos na ordem em que essas tarefas devem ser executadas e cada um deles está em um estágio diferente do *pipeline*. Quando um processador termina a execução de sua tarefa, passa o resultado do processamento para o próximo estágio. O modelo em *pipeline* pode ser encontrado no processador Cisco PXF [9], por exemplo.

Os coprocessadores são processadores utilizados para tarefas mais específicas que complementam as funções do processador principal. Esses processadores geralmente não são capazes de buscar instruções na memória, executar um fluxo de instruções, ou realizar operações de entrada e saída. Essas tarefas devem ser executadas pelo processador principal e somente a operação específica é encaminhada ao coprocessador. Os coprocessadores são usados, por exemplo, para acelerar certas funções como operações de ponto flutuante. Além da lógica otimizada para a tarefa que devem executar, eles executam em paralelo com o processador principal, o que pode ser muito vantajoso.

A Figura 4 ilustra alguns blocos funcionais do processador PowerNP da IBM [8]. Uma vez que um pacote é recebido pela interface do processador, é armazenado em uma fila de entrada. A unidade de emissão de pacotes (*Dispatch Unit*) retira informações dos pacotes da fila e as encaminha para processamento assim que uma *thread* fica disponível. A Figura mostra um Processador de Protocolo (Dyadic Protocol Processor Unit (DPPU)), que contém todos os processadores e coprocessadores necessários para o processamento do pacote. Cada DPPU contém um *Core Language Processor (CLP)*, que é um processador de propósito geral bastante simples que executa duas *threads*, sendo que cada uma processa um pacote. O DPPU na Figura 4 contém dois coprocessadores. Desses coprocessadores, um

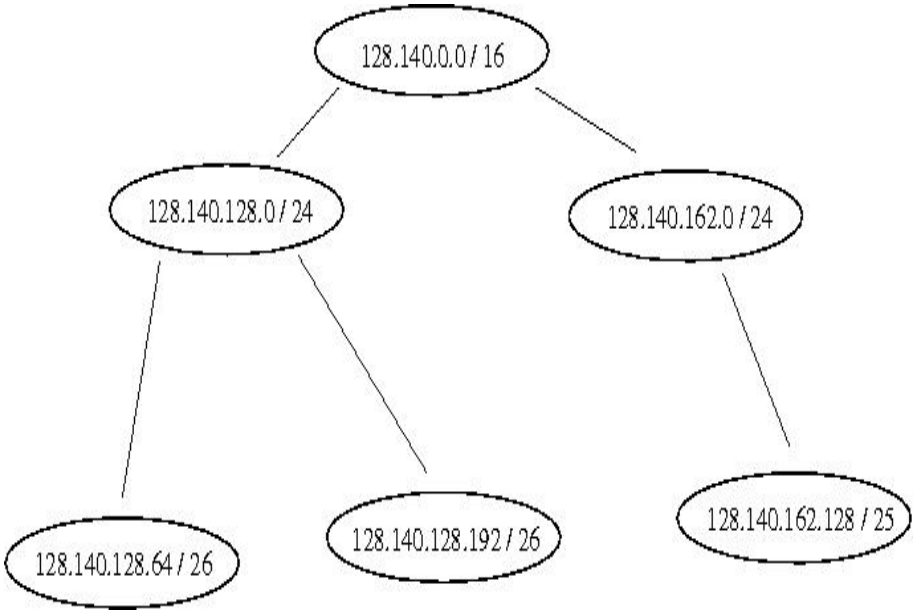

Figura 3: Exemplo de árvore de prefixos.

é o *Tree Search Engine (TSE)*, que executa operações para busca em árvores em *hardware*. Essas árvores são abstrações que representam estruturas hierárquicas de endereços, tais como os prefixos IP mostrados na Figura 3. Nessas estruturas os níveis mais baixos são subconjuntos dos níveis de cima: os prefixos de nível mais alto são prefixos dos dos prefixos de nível mais baixo e dois prefixos em um mesmo nível não são prefixos unm do outro. As árvores são armazenadas em estruturas de memória que podem ser internas ou externas ao processador. O outro coprocessador mostrado é um coprocessador responsável pelo cálculo do *checksum*. Esses coprocessadores uma vez acionados pelo CLP, executam em paralelo, entregando o resultado final à *thread*. As duas *threads* compartilham os coprocessadores do DPPU e uma *thread* só poderá utilizar o coprocessador quando este terminar a tarefa solicitada pela outra. A Unidade de Finalização (*Completion Unit*) assegura que os pacotes processados por diferentes *threads* são inseridos em ordem na fila de saída se eles pertencem a um mesmo fluxo de pacotes. Esse diagrama é bastante simplificado, o PowerNP real contém muitas outras unidades funcionais.

Multithreading consiste na execução de instruções de várias threads executando em um processador. Através da Figura 5, podemos comparar o modelo superescalar com dois modelos de *multithreading*. No modelo tradicional, também conhecido como *Superthreading*, a cada ciclo o escalonador pode escolher instruções de uma *thread* diferente para executar e se o processador fosse capaz de buscar várias instrucões por ciclo, um lote de instruções de uma mesma *thread* pode ser buscado. A granularidade é um conceito associado a *multithreading*:

- *Multithreading de grão grosso (Coarse-Grain Multithreading)*: o processador troca de contexto somente quando ocorre um evento de grande latência, como por exemplo, uma falta na cache.

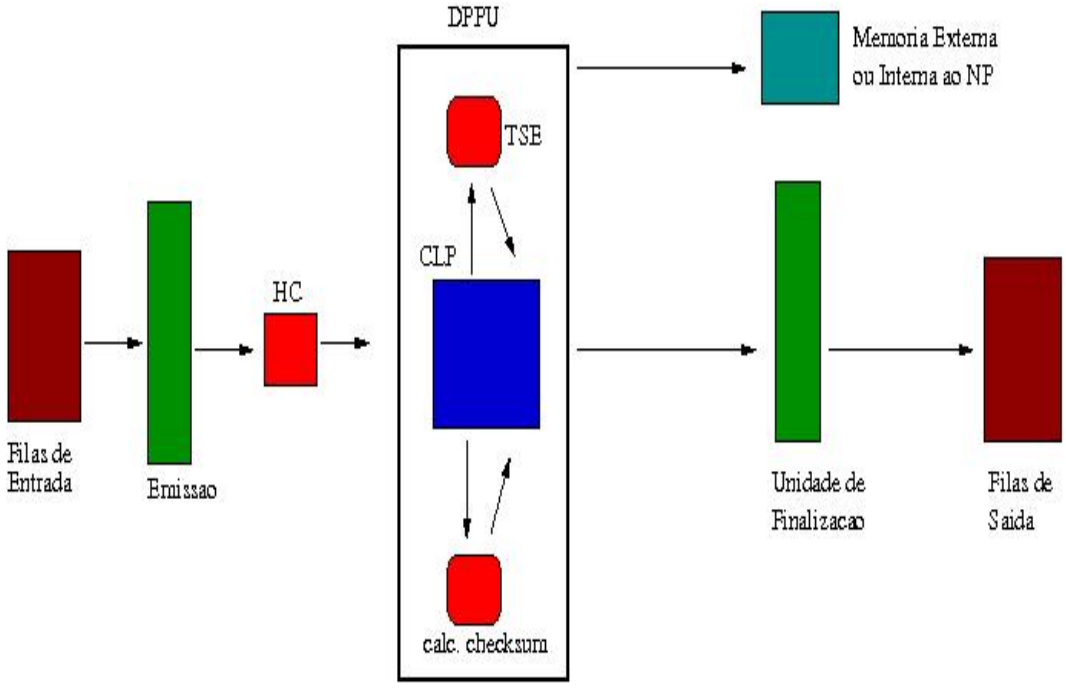

Figura 4: Diagrama simplificado de alguns blocos funcionais presentes no processador PowerNP da IBM.

- *Multithreading de grão fino (Fine-Grain Multithreading)*: o processador troca de contexto todo ciclo, por exemplo, no primeiro ciclo o escalonador escolhe instruções da *thread* 1, no segundo da *thread* 2 e assim por diante, resultando em um aproveitamento melhor dos ciclos do processador do que no grão grosso. Neste caso, a implementação deve garantir que o custo da troca de contexto é muito baixo.

O conceito de *Simultaneous Multithreading (SMT)* é uma variação do modelo tradicional descrito anteriormente. Com SMT, a cada ciclo instruções de diferentes *threads* podem ser buscadas e dessa forma pode-se escolher para executar instruções de *threads* distintas em um mesmo ciclo. Esse modelo, portanto, transforma paralelismo em nível de *thread* em paralelismo em nível de instrução, possibilitando uma utilização maior dos ciclos do processador [5].

2.2 Trabalhos Relacionados

Os *Network Processors* são processadores especializados no processamento de pacotes e os seus fabricantes tem focado no desempenho e na flexibilidade, de forma a permitir o tratamento de novas aplicações e tarefas de gerenciamento ao mesmo tempo em que garantem a largura de banda da rede e a qualidade de serviço para os diferentes fluxos de pacotes. O projeto desses processadores busca um compromisso entre o desempenho de um circuito integrado projetado especificamente para uma tarefa (*Application-specific integrated circuit (ASIC)*) e um processador de propósito geral (*General purpose processor (GPP)*) [7].

Os ASICs são circuitos integrados que implementam uma função ou conjunto de funções diretamente em *hardware*. Esses circuitos integrados geralmente são projetados baseados

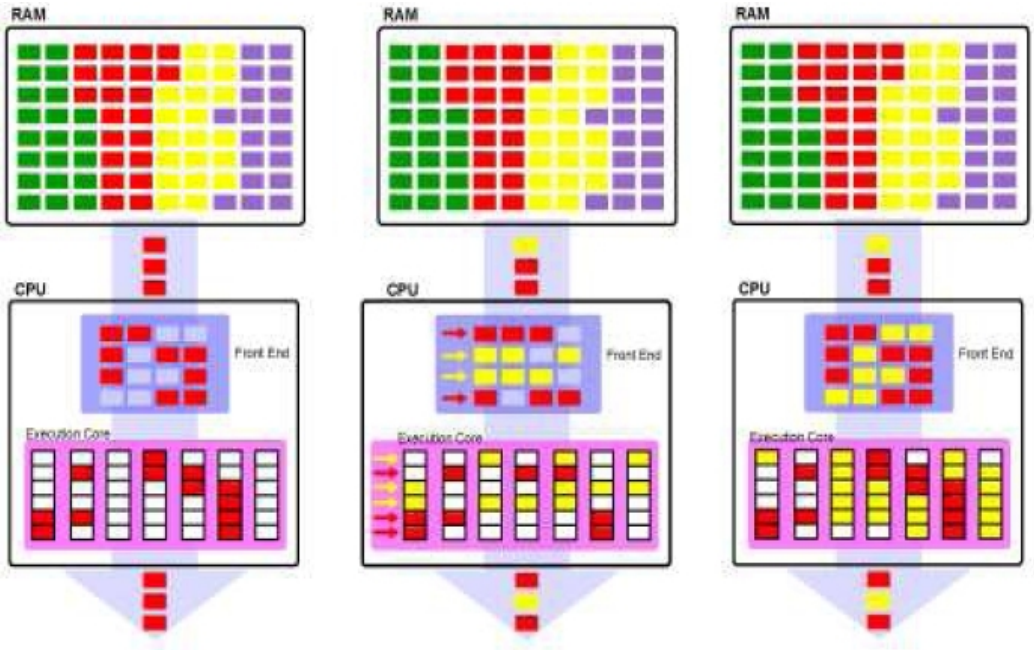

Figura 5: Da esquerda para direita estão respectivamente as arquiteturas: superescalar, *multithreding* de grão fino e *Simultaneous Multithreading (SMT)*.

numa especificação de funções bem definida. Portanto, mudanças nos protocolos e o tratamento de novas aplicações provavelmente necessitariam de um novo projeto. Já os GPPs, são processadores comuns com um conjunto de instruções básicas para implementar qualquer função em *software*. Os processadores de propósito geral têm alta flexibilidade e além disso, implementar um programa para tratar alterações em um protocolo ou mesmo novos protocolos, demanda muito menos tempo e dinheiro do que o projeto de um novo circuito integrado. O problema dos GPPs é o seu baixo desempenho em comparação com os ASICs. Portanto os NPs geralmente utilizam circuitos específicos para tratamento de certas funções básicas, tais como cálculo de *checksum* e processadores de propósito geral para realizar tarefas de controle.

Os *Network Processors* geralmente são *multicore* e apresentam vários blocos funcionais que são especializados em determinadas funções como cálculo de *checksum* e busca em árvores. Algumas unidades funcionais são mostradas na Figura 4, que mostra um diagrama simplificado do PowerNP. O PowerNP contém oito unidades de processamento de protocolo (DPPU), e cada uma contém dois *Core Language Processors (CLPs)*, que são processadores RISC de propósito geral, totalizando 16 *cores* e cada um dos CLPs executa duas *threads*. Essas *threads* contém as instruções necessárias para o processamento dos pacotes, sendo cada uma responsável pelo processamento de um pacote. Além disso, em cada DPPU existem 8 coprocessadores, que são compartilhados pelas 4 *threads* em execução nos 2 CLPs desse DPPU.

Apesar do PowerNP poder executar 32 *threads*, somente 16 estão em execução em um dado instante, pois os CLPs não são *multithreaded*. Os CLPs trocam de contexto sempre

que acontece um evento que faz com que a *thread* em execução espere (*stall*) e essas trocas de contexto são otimizadas para serem realizadas em "tempo zero". Crowley at al. [1] avaliaram as arquiteturas superescalar com execução fora de ordem, *multithreading* de grão fino, multiprocessador simples e *simultaneous multithreading (SMT)* e mostraram que esta última é a mais indicada para os *cores* dos NPs. Portanto, mesmo com as otimizações na troca de contexto nos CLPs, o desempenho poderia ser ainda melhor com *cores* SMT.

Wolf e Franklin criaram um modelo analítico de desempenho (baseado no diagrama mostrado na Figura 6) dos *Network Processors*, parametrizando os diferentes componentes e analisando o custo de diferentes configurações, tais como uso de caches de diferentes tamanhos e associatividades, número de processadores e nível de *multithreading* de cada processador. Como carga de trabalho foi utilizada a suite CommBench, desenvolvida pelos autores especificamente para a análise de NPs, para obtenção de parâmetros de entrada. Os autores discutem os custos e benefícios entre diferentes configurações dos componentes e o impacto que alterações na configuração dos mesmos teria no desempenho do sistema. Através desse modelo concluíram que o desempenho do processador é bastante sensível à mudanças na configuração da cache, bem como mostraram que em um nível de *multithreading* com 2 ou 4 *threads* por processador tem um desempenho satisfatório [10].

Como discutido anteriormente, o desempenho dos NPs é bastante sensível às configurações de cache. Por mais rápido que o acesso à memória seja, acessos a memórias externas ainda tem alta latência e o uso de caches é necessário. Liu [2] mostrou que apesar de não existir localidade significativa em endereços individuais em roteadores de *backbone*, existe boa localidade em prefixos. Porém, fazer cache de prefixos pode levar a problemas que não ocorriam antes, pois se um endereço IP de destino casar com um prefixo na cache e existir um prefixo mais específico na tabela de roteamento o pacote seguirá uma rota errada indicada pelo prefixo na cache. Os autores mostraram diferentes formas de evitar esse problema através de expansões dos prefixos antes desses serem armazenados na tabela de roteamento. Tomando como exemplo os prefixos 0* e 0101*, temos que o segundo prefixo casa com o primeiro, porque 0* é subprefixo de 0101* e portanto não pode ser inserido na cache. As técnicas investigadas pelos autores são: *Complete Prefix Tree Expansion (CPTE), Prefix Expansion (NPE) e Partial Prefix Tree Expansion (PPTE).* Na CPTE há expansão completa dos prefixos antes deles serem armazenados na tabela de roteamento. No caso dos prefixos 0* e 0101*, seriam criados as expansões 00*, 011* e 0100* que seriam armazenados na tabela em vez de 0* com o mesmo *next hop* deste. Na NPE os prefixos que são subprefixos de algum outro são marcados para não serem inseridos na cache. Na PPTE a expansão é feita no primeiro nível apenas. No exemplo, teríamos então 00* em vez de 0* na tabela. Os autores mostraram que a CPTE apresenta um melhor desempenho, mas que a PPTE apresenta um compromisso melhor entre o desempenho e o tamanho da tabela.

Como foi dito anteriormente, a operação de busca na memória deve ser muito rápida. McAuley e Francis desenvolveram um estudo de diferentes arquiteturas e algoritmos que poderiam ser utilizados na implementação da tabela de roteamento. Foram estudadas

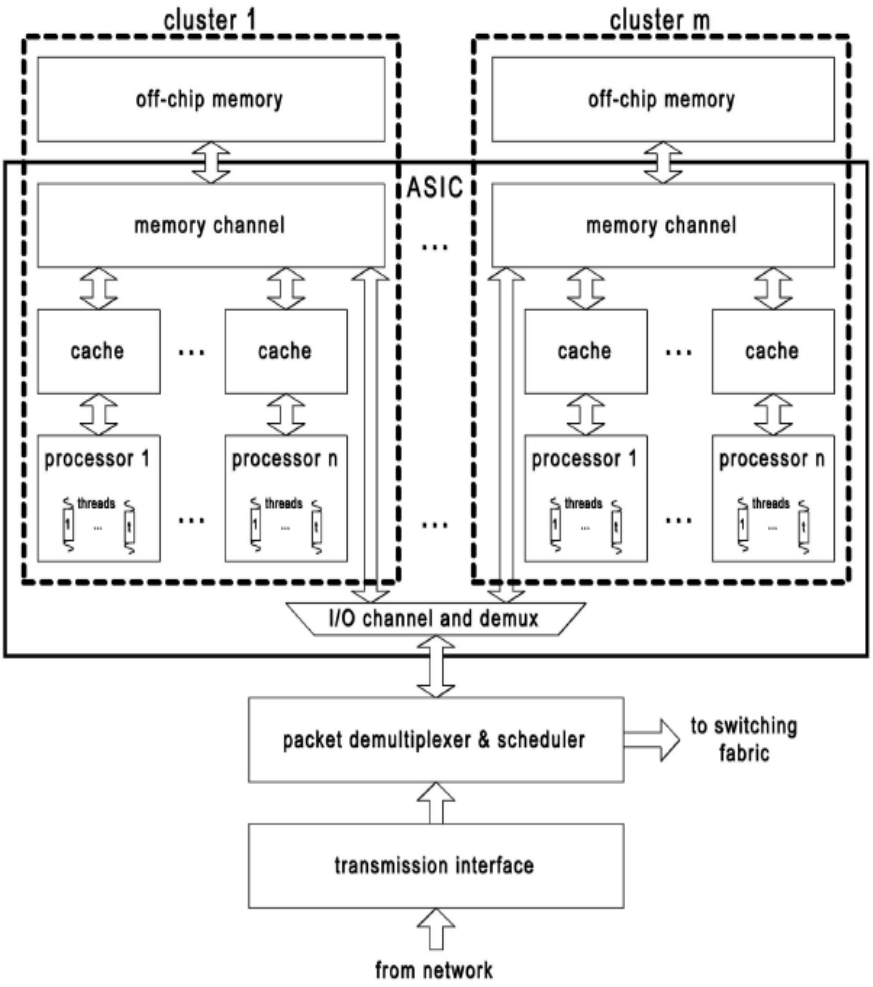

Figura 6: Arquitetura de um *Network Processor* utilizada no modelo analítico.

técnicas de busca baseada em *Random Access Memories (RAMs)* utilizando indexação direta ou estruturas de árvores tais como árvores Binária e Patrícia. Analisaram também as *Content Adressable Memories (CAMs)*, discutindo diferentes técnicas para implementar buscas e atualizações em *Binary-CAMs (BCAMs)* e *Ternary-CAMs (TCAMs)*. As BCAMs são CAMs em que cada bit pode assumir o valor de '0' ou '1'. Os autores analisaram diferentes técnicas, dentre elas a B3 (*Single-Cycle Multiple-Logical CAM*) que utiliza *clusters* diferentes para cada tamanho de máscara e precisa de *hardware* extra para determinar a prioridade dentre os diferentes tamanhos. Foi analisada também, a TCAM, na qual os bits podem assumir um de três valores, '0', '1' ou 'X' (*don't care*). A técnica utilizando T3 (*Single-Cycle Multiple-Logical CAM*) permite atualização rápida e tem baixa complexidade, mas assim como a técnica B3 precisa de *hardware* adicional para determinar a prioridade [3].

Em outro estudo relacionado, Gamache at al. projetaram uma TCAM rápida e flexível para aplicações em roteamento na *Internet*, permitindo buscas eficientes em redes a velocidades de até 76.8 Gb/s, ao mesmo tempo em que oferece flexibilidade tendo elementos de 512 bits, dos quais apenas 32 são utilizados para endereços IPv4 e 128 em IPv6 , sendo

que os demais bits que não serão utilizados devem ser desabilitados [6]. Neste trabalho, foi implementada uma versão simples desta TCAM para simular a tabela de roteamento. Esse projeto é descrito mais detalhadamente nas próximas seções.

3 Trabalho

O trabalho consiste na implementação de um simulador de um roteador, buscando investigar novas técnicas de roteamento na *Internet* e analisar as estruturas envolvidas no processamento e encaminhamento dos pacotes que são recebidos. O simulador foi desenvolvido com base em alguns blocos funcionais do processador da IBM PowerNP [8]. Esse processador contém uma unidade que faz a classificação dos pacotes (*Hardware Classifier (HC)*) e envia as informações para a *thread* que processará os mesmos. Em cada processador de protocolo (DPPU) existe um *Tree Search Engine (TSE)*, que é um coprocessador que efetua operações de busca em três tipos de árvores diferentes, das quais a busca em uma *LPM tree* foi considerada neste trabalho para implementar a tabela de roteamento. Destas tarefas, foram implementados uma versão simples do HC, que realiza um *parsing* dos pacotes e coloca as informações relevantes em uma fila, e uma versão simplificada do TSE, que retira aquelas informações da fila e realiza uma busca na tabela de roteamento.

A tabela de roteamento foi implementada com base na TCAM desenvolvida por [6] especificamente para tarefas de busca em redes. No projeto dessa TCAM, os autores buscavam flexibilidade para dar suporte a tamanhos variáveis de endereços de destino e tamanho variável das informações que serão retornadas da busca (geralmente o próximo nó para o qual o pacote deve ser enviado), ao mesmo tempo que sua estrutura de blocos e cálculo do LPM foram desenvolvidos para ser, no nível de circuito integrado, bastante rápidos. Neste trabalho foi implementada uma versão simplificada dessa TCAM, dando ênfase na análise da sua estrutura, visto que a maioria dos seus aspectos de *hardware* que a tornam extremamente rápida são perdidos em *software*, como por exemplo, o paralelismo na comparação de todas as entradas. A Figura 7 é uma representação gráfica do que foi implementado.

O simulador foi implementado na linguagem C, na distribuição de Linux Ubuntu, compilador GCC. Foram necessárias 2900 linhas de código, incluindo os arquivos de cabeçalho. Para capturar os pacotes foi utilizada a biblioteca *pcap*, a mesma que é utilizado no conhecido utilitário de redes *tcpdump*. A biblioteca *pcap* contém funções de captura que podem ser utilizadas para obter pacotes que estão na rede ou para obter pacotes de um *log* de pacotes que foi capturado e arquivado em formato legível pela biblioteca. Esse é o caso do *log* que é utilizado como entrada para o programa, capturado em um roteador real.

3.1 Classificação dos pacotes

A classificação dos pacotes é feita com um *parser* que implementa um atômato finito. No processador usado como exemplo, o resultado da classificação do pacote é encaminhado para a *thread* encarregada do seu processamento. Qualquer informação sobre os cabeçalhos

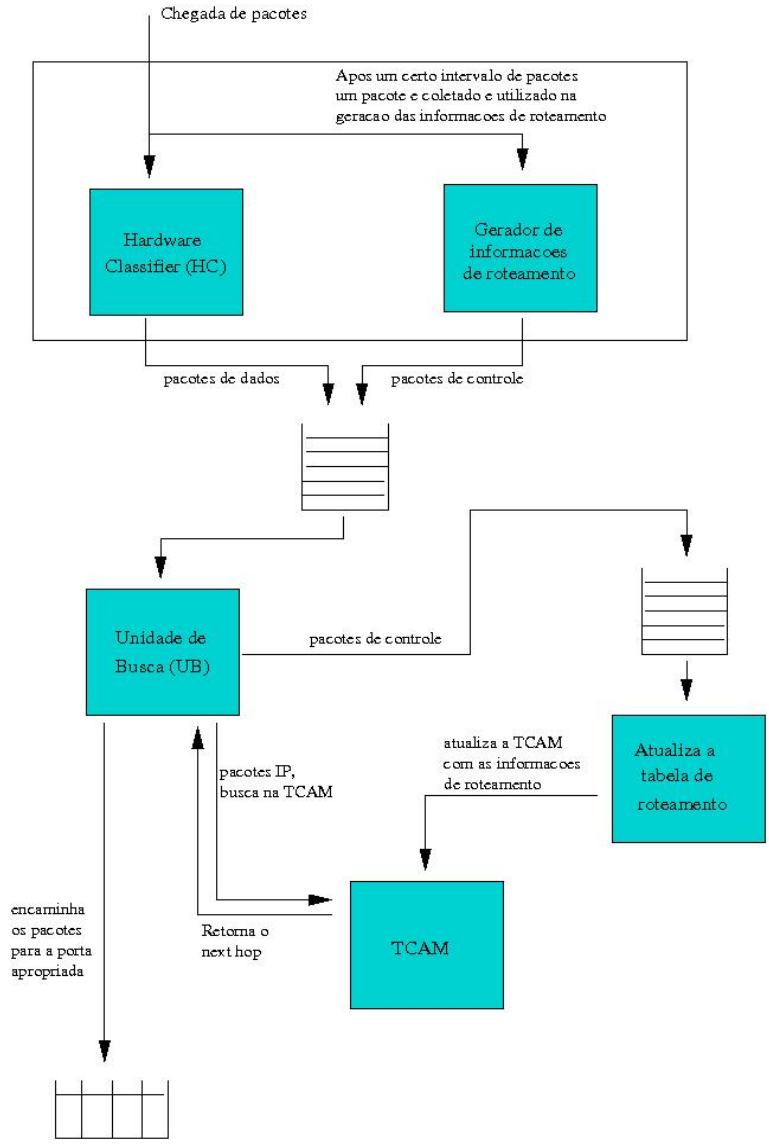

Figura 7: Representação gráfica do que foi desenvolvido no trabalho.

dos diferentes protocolos que estão presentes no pacote pode ser extraída pelo *parser*. Neste trabalho, como a próxima etapa é a busca na tabela de roteamento, são recolhidas informações pertinentes à busca e estas são inseridas em uma fila para posteriormente serem utilizadas.

Como foi dito anteriormente, a implementação foi baseada no processador PowerNP da IBM. Este processador possui várias configurações de suas interfaces de entrada. O processador contém dois *Physical MAC multiplexer (PMM)*, um de entrada e outro de saída que são responsáveis pela movimentação de dados entre dispositivos que implementam a camada física e o PowerNP. Cada um deles contém quatro *Data Mover Units (DMUs)*, que podem ser configurados independentemente para ser uma interface POS (*packet over SONET*) ou Ethernet. Além disso, cada DMU pode ser configurado para suportar diferentes números de porta dependendo da velocidade desejada para cada porta, por exemplo 1 porta Gigabit Ethernet de 1 Gb/s ou 10 portas Fast Ethernet de 100 Mb/s. Neste tra-

balho, como não houve meios de obter pacotes de uma interface POS, foram utilizados somente pacotes vindos de uma interface Ethernet. Por simplicidade, são simuladas quatro interfaces de saída, como se cada DMU estivesse configurado para suportar uma única porta.

No autômato implementado (a Figura 8 mostra uma parte do desse autômato), cada nó é um estado do sistema que representa um protocolo que está sendo analisado. Em cada estado, o cabeçalho do protocolo correspondente é extraído do pacote, são recolhidas informações relevantes e com base no campo apropriado, determina-se o próximo estado. Inicialmente, como o sistema recebe dados de uma interface Ethernet, o autômato é iniciado no estado "Ethernet". A partir daí, com base no campo *ethertype* do cabeçalho Ethernet extraído do pacote, determina-se o próximo estado do autômato e assim sucessivamente, até chegarmos em um protocolo limite, que é um dos protocolos da camada de transporte TCP ou UDP, protocolos de controle como ICMP, ou outros que não encapsulam informações ou encapsulam informações que não foram investigadas aqui, neste trabalho.

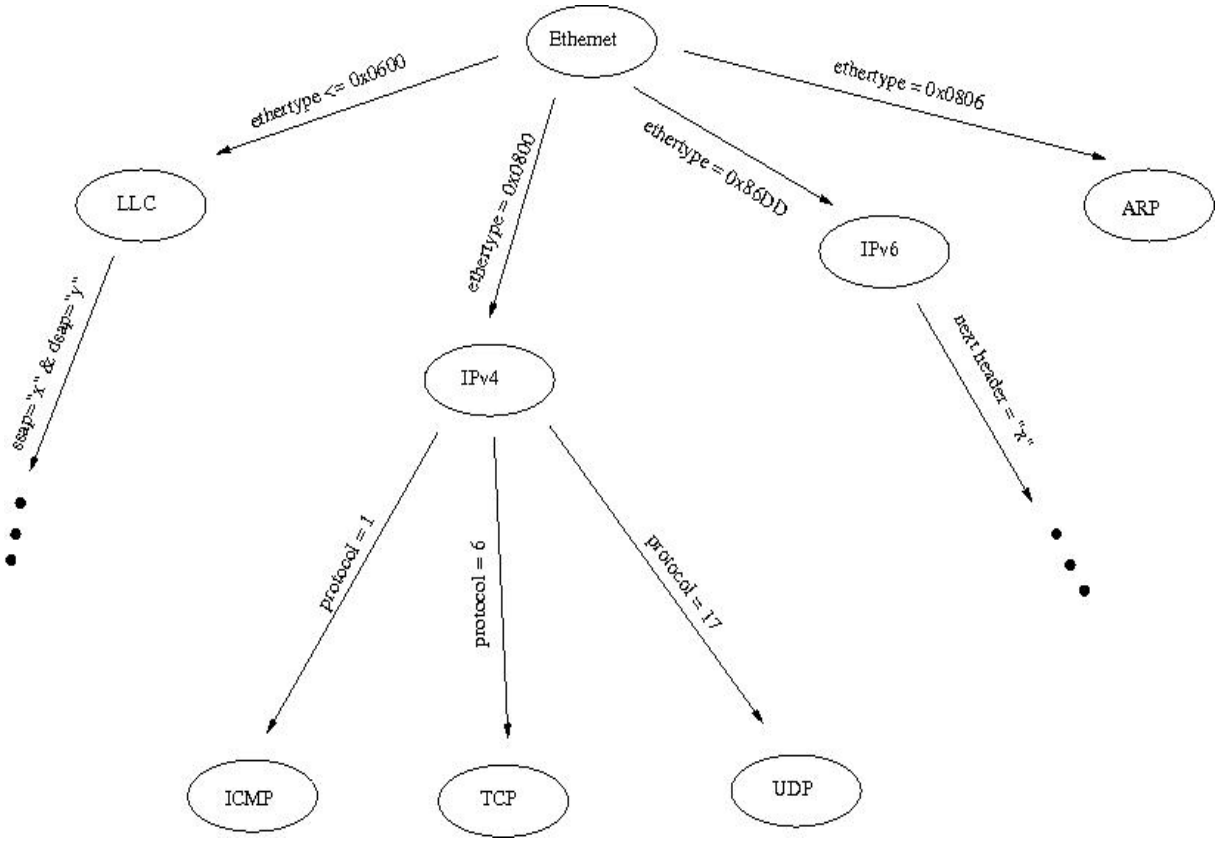

Figura 8: Alguns estados do autômato implementado pelo *parser*.

Durante os diferentes estados pelos quais passa o *parser*, informações relevantes vão sendo recolhidas e inseridas numa estrutura à parte. Quando o estado representa um protocolo limite, a função que implementa essa etapa coloca a estrutura em uma fila para ser utilizada posteriormente. Essa etapa seria a passagem das informações da classificação de pacotes para a *thread* no PowerNP. Na implementação, essas informações serão apenas

utilizadas para busca em uma tabela de roteamento e, portanto, dentre as informações recolhidas, a única informação que é realmente utilizada na tarefa de busca é o endereço de destino do pacote.

A Figura 10 mostra a estrutura que foi usada para armazenar os cabeçalhos UDP assim que este era identificado durante o *parsing* do pacote. As outras estruturas de cabeçalho são semelhantes. No caso do cabeçalho UDP, que é um cabeçalho simples, todos os campos têm 16 bits, portanto foi utilizado o tipo de dado *unsigned short* para armazená-los. Quando algum campo não pode ser armazenado em um tipo de dado simples, são utilizados *arrays*, para campos que são múltiplos de 8 bits ou macros que encapsulam a operação de deslocamento em caso de campos com tamanhos não múltiplos de 8 bits. A Figura **??** mostra o cabeçalho PPPOE, no qual podemos ver o acesso aos campos *tipo* e *versão*, ambos com 4 bits, através de macros.

```
/*****************************************
 *                                  UDP
 *****************************************/
struct udp {
    u_short sport;
    u_short dport;
    u_short len;
    u_short checksum;
};
/*****************************************/
```

Figura 9: Estrutura que armazena o cabeçalho UDP.

```
struct pppoe {
    u_char vtype;       /* version | type       */
    u_char code;        /* pppoe code           */
    u_short session_id; /* pppoe session ID     */
    u_short len;        /* pppoe payload length */

};
#define PPPOE_V(_pppoe) ((_pppoe)->vtype >> 4)       /* pppoe version */
#define PPPOE_TYPE(_pppoe) ((_pppoe)->vtype & 0x0F)  /* pppoe type    */
    /* pppoe tag codes */
#define PPP_CODE        0x00
#define PADI_CODE       0x09
#define PADO_CODE       0x07
#define PADR_CODE       0x19
#define PADS_CODE       0x65
#define PADT_CODE       0xa7
```

Figura 10: Estrutura que armazena o cabeçalho PPPOE e macros para acesso aos campos de 4 bits.

A Figura 11 mostra a estrutura onde são armazenadas as informações relevantes obtidadas através do *parsing* de um pacote. Assim que as informações são obtidas através do

parsing, são armazenadas nessa estrutura e esta, por sua vez, é inserida em uma fila. A Unidade de Busca irá retirá-la da fila e verificar o campo *type* para ver se é um pacote de controle ou um pacote IP comum.

```
/*****************************************************************************
 *                        ROUTER INFO
 *- Information which are passed to the router in order to perform tree searches
 *
 *****************************************************************************/

    struct router_info {
        u_char type;        // Control packet or Data packet
        int packet_count;
        u_char *packet;
        int len;
        u_short src_port;
        u_short dst_port;
        struct  in_addr ip_src, ip_dst;
        u_short pid;
        u_char src_mac[ETHER_ADDR_LEN];
        u_char dst_mac[ETHER_ADDR_LEN];
    };
    #define DATA_PACKET 0x88
    #define CONTROL_PACKET 0x44
/*****************************************************************************/
```

Figura 11: Estrutura que armazena as informações recolhidas pelo *parser*.

3.2 Geração das Informações de Roteamento

A geração das informações de roteamento em roteadores reais é baseada em protocolos de roteamento que definem um conjunto de regras para que, periodicamente, os roteadores troquem informações sobre rotas com os seus vizinhos. Neste trabalho, como é simulado apenas um roteador simples, essas informações são geradas de forma pseudo-aleatória com base nos pacotes e no número de pacotes. As informações de roteamento neste caso se resumem a uma tripla <prefixo IP, máscara, *next hop*> e para que os prefixos armazenados na tabela de roteamento sejam compatíveis com o tráfego da rede simulado, o prefixo IP gerado é obtido a partir de um pacote qualquer que é processado pelo roteador.

A geração dessas informações é ilustrada na Figura 12. É estabelecida inicialmente uma quantidade de pacotes que deve ser recebida até uma informação de roteamento ser gerada. Assim que esse intervalo é atingido, ele é alterado somando-se a ele um número aleatório. Por exemplo, se o intervalo inicial é de 1000 pacotes, assim que chegarem os 1000 pacotes, o intervalo é alterado para $(1975 + \theta \in 0..49)$ permitindo uma variação no próximo valor entre 1975 e 2024. Essa variação aproxima o modelo do trabalho com o funcionamento real, pois na realidade não se sabe exatamente quando um roteador receberá uma informação de roteamento. Quando a quantidade de pacotes determinada nesse intervalo é recebida,

quando o contador de pacotes é um múltiplo do intervalo estabelecido, o endereço IP de destino do pacote que está sendo processado naquele momento é encaminhado a uma função, na qual será utilizado para gerar um prefixo IP. No exemplo ilustrado na figura, o endereço é 128.96.34.192. Essa função gera um número aleatório entre 15 e 24 que é usado como tamanho do prefixo e da máscara (número de bits '1' da esquerda para direita na máscara). Na figura, o número gerado é o 16 e portanto, 16 bits do endereço são usados como prefixo, o que resulta no prefixo 128.96.0.0 e na máscara 255.255.0.0. Em seguida o *next hop* é gerado, também de forma aleatória, assumindo valores entre 0 e 3 que representam as portas de saída do roteador. Na ilustração foi gerado o número 3, representando a porta de saída 3. Por fim, a tripla <prefixo IP, máscara, *next hop*> que foi gerada, é gravada em uma estrutura comum que é inserida na mesma fila onde entram as informações geradas pelo *parser* para cada pacote, para posterior retirada pela Unidade de Busca.

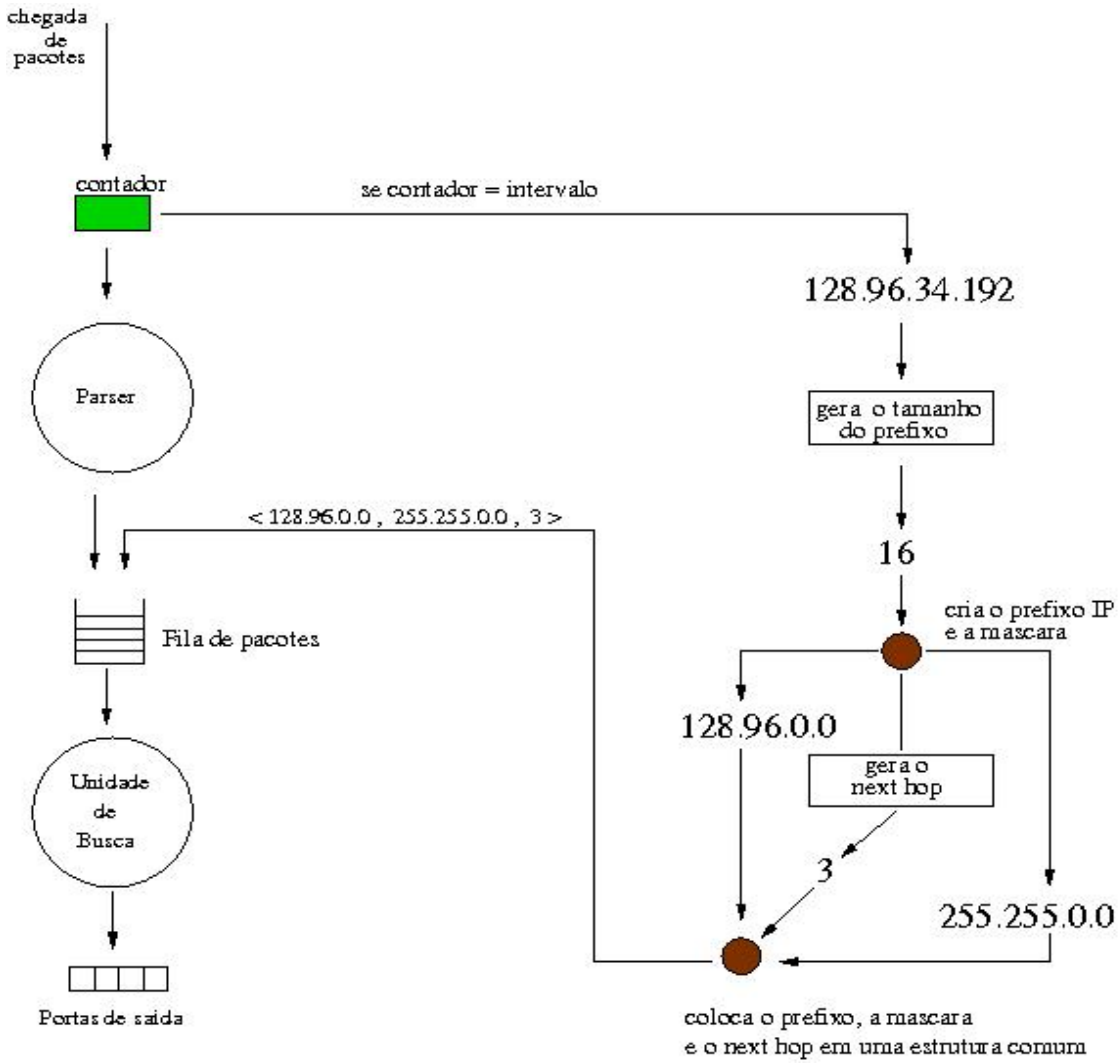

Figura 12: Geração das informações de roteamento.

3.3 Tabela de Roteamento

A tabela de roteamento é uma tabela que relaciona um prefixo IP com uma porta de saída. Sempre que um pacote chegar no roteador, este examina a tabela usando como chave o endereço IP de destino do pacote. Se o endereço casar com algum prefixo na sua tabela, o roteador encaminha o pacote para a porta associada àquele prefixo.

Como foi discutido em seções anteriores, a tabela de roteamento pode ser implementada de diferentes formas. Neste trabalho foi implementada uma versão simplificada de uma TCAM projetada por [6]. A seguir será descrito o funcionamento dessa TCAM e em seguida a sua implementação neste trabalho.

A TCAM [6] mostrada na Figura 13 tem o *layout* de um quadrado, organizada como uma matriz de 13x13 de quadradinhos menores, totalizando 169, dos quais 168 são blocos que contém os elementos da tabela e o outro que está localizado no centro, é o *winner's circle LPM (WC-LPM)* descrito mais adiante. A chave de busca é enviada simultaneamente para todos blocos, que realizam uma busca independentemente dos demais. Cada bloco contém 128 linhas, sendo que uma linha armazena um elemento da tabela de roteamento. Cada linha contém um campo TCAM, que é composto de 512 células TCAM; um campo LPM, composto 9 células LPM; e um campo SRAM, composto de 512 células SRAM. Nessa linha, o campo TCAM armazena um prefixo IP. Junto com o campo SRAM, que armazena a informação que será retornada, o campo TCAM forma um elemento da tabela de roteamento. O campo LPM armazena o tamanho do prefixo e serve para determinar o elemento de prefixo mais longo dentre os que casaram com a chave dentro de um bloco.

Cada célula TCAM armazena um bit do prefixo e um bit da máscara. Cada bit da chave de busca é comparado com o bit correspondente da máscara; se esse bit for '0' o valor armazenado na célula é considerado um 'X' (*don't care*) e o bit da chave automaticamente "casa" com o bit de prefixo correspondente não importando o valor deste. Se o bit da máscara for '1', o valor do bit de prefixo da célula é '0' ou '1'. Portanto, para casar com o valor dessa célula, o valor do bit da chave deve ser igual ao valor do bit correspondente do prefixo.

O campo SRAM armazena a informação que será retornada na busca (geralmente o próximo nó para o qual o pacote deve ser encaminhado, o *next hop*). O *layout* das células SRAM é diferente dos demais. Como uma célula SRAM não tem nenhuma lógica adicional e somente serve de armazenamento de um bit, é menor do que as outras. As células TCAM e as células LPM são organizadas uma ao lado da outra, mas as células SRAM primeiramente são agrupadas em 4 células (uma sobre a outra), para que estas fiquem com a mesma altura das outras, e então cada agrupamento desses é posicionado um ao lado do outro.

Cada célula LPM armazena um bit de um número que representa o tamanho do prefixo daquele elemento. Se um elemento tem um prefixo de tamanho 22, por exemplo, os bits dessas células serão 000010110. Como são 9 células em cada linha por conta das 512 células TCAMs, são $2^9 = 512$ os prefixos possíveis. Isso demonstra a flexibilidade do projeto, pois para armazenar prefixos simples da versão 6 do IP (com endereços de 128 bits), só seriam

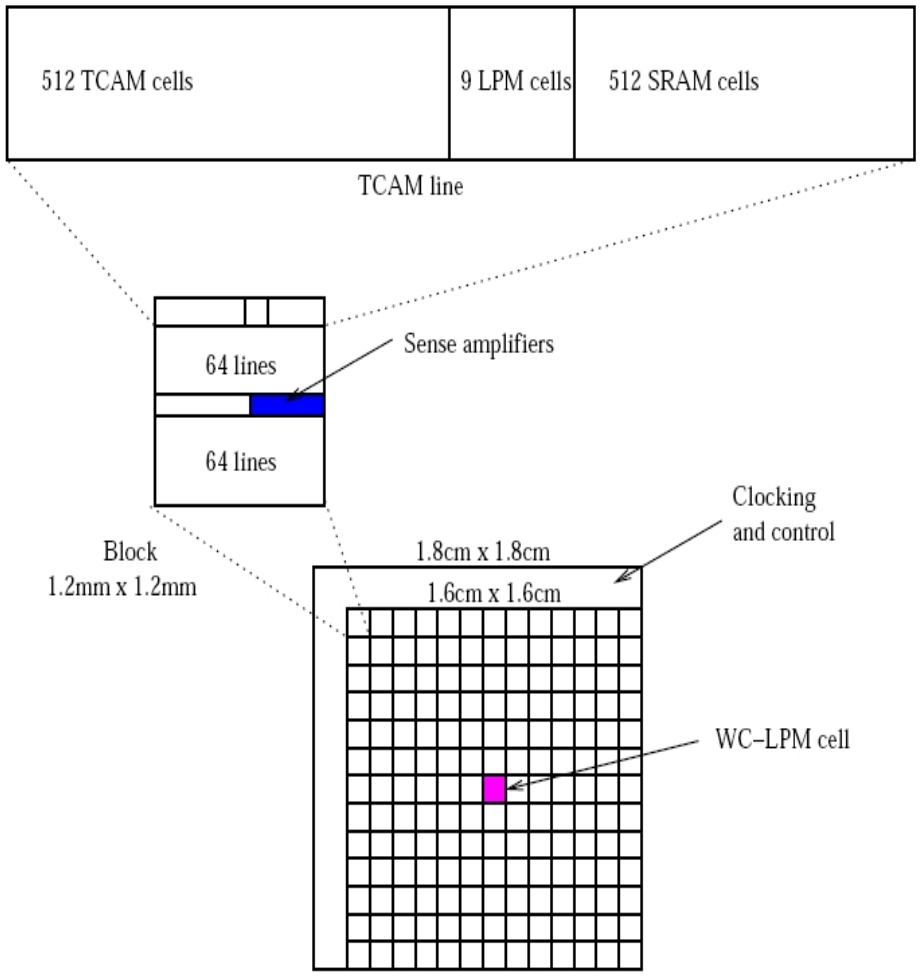

Figura 13: Arranjo das células no *layout* de uma implementação alternativa da TCAM.

utilizados 1/4 do número de células TCAM disponíveis, as demais devem ser desabilitadas. A função das células LPM é determinar qual dentre os elementos de um bloco que casaram com a chave tem o maior prefixo.

A linha com o maior prefixo dentre as linhas do bloco é determinada através da lógica de controle da TCAM e da lógica de controle em cada célula LPM. Como são 9 células LPM, a operação é feita em 9 fases. Em cada fase é analisada uma célula de cada campo LPM de cada linha, iniciando no bit_8 e analisando os bits sucessivamente até o bit_0. Essa operação é baseada no fato de que, dentre os elementos que casam com a chave, aquele que tem o maior prefixo em alguma fase f tem um bit '1' armazenado na $célula_{9-f}$, enquanto outra tem um bit '0'. Por exemplo, supondo que a chave seja o endereço IP de destino 128.96.34.192 e que um dos blocos da TCAM contenha dois prefixos, 128.96.34.0/24 e 128.96.0.0/16, e seus campos LPM armazenam respectivamente 000011000 (24) e 000010000 (16). Como na sexta fase são analisados os bits 3 $(9-6=3)$ de cada campo LPM e a segunda linha tem o $bit_3 = 0$, ela é removida das futuras comparações. Se alguma linha tem '1' armazenado na célula LPM que está sendo analisada, somente as outras que também tem '1' armazenado, serão consideradas na fase seguinte. Ao final de todas as 9 fases, só haverá uma linha (a

linha "vencedora" do bloco) que ainda está em consideração, aquela que tem o maior prefixo armazenado no campo TCAM dentre as que casaram com a chave.

A linha "vencedora", que é a linha na qual o campo TCAM casa com a chave e o campo LPM tem o maior valor dentre todas as linhas de cada bloco, é então enviada ao WC-LPM. O WC-LPM contém 168 linhas e cada uma delas armazena a linha "vencedora" de um bloco. Como foi dito anteriormente, a chave é enviada a todos os blocos simultaneamente. Quando os blocos determinam a linha com maior valor armazenado no campo LPM dentre as que casaram com a chave, essas linhas são armazenadas no WC-LPM. Aí então as 9 fases LPM recomeçam, agora no WC-LPM. São analisadas as células LPM das 168 linhas e aquela que tiver o maior valor será a linha "vencedora" de todos os blocos. O elemento correspondente a essa linha é então o elemento da tabela de roteamento que casou com a chave. Suas informações associadas, que estão armazenadas no campo SRAM, são retornadas e a busca termina. A operação de busca tem uma latência de 3 ciclos, mas ela está oculta pela implementação em *pipeline*.

Neste trabalho foram implementadas algumas das estruturas da TCAM descritas anteriormente, células TCAM, células LPM e células SRAM. Por simplicidade, foi utilizado somente um bloco para representar a TCAM em vez de 168 blocos e um WC-LPM. O número de linhas do bloco pode ser configurado, sabendo que um bloco de it N linhas pode armazenar N elementos. Cada elemento é composto de um campo de prefixo, um campo tamanho do prefixo (campo LPM) e um campo *next hop*. O campo prefixo é armazenado nas células TCAM, o campo LPM é armazenado nas células LPM e o campo *next hop* é armazenado nas células SRAM. A Figura 14 mostra as estruturas que representam a TCAM e suas configurações de tamanho, sendo que nesta figura o bloco tem 32 linhas.

As células TCAM são implementadas em uma estrutura com dois campos, um representando um bit do prefixo e outro o bit da máscara. As células LPM e SRAM foram implementadas com um tipo de dados simples para cada bit: *unsigned char*. Uma linha da TCAM, que armazena um elemento da tabela, é uma estrutura que contém 3 campos. O primeiro campo é o campo prefixo, que consiste em um *array* de células TCAM que armazenam um prefixo IP; o segundo é o campo LPM, um *array* de células LPM que armazenam o tamanho do prefixo; e o terceiro é o campo *next hop*, que é um *array* de células SRAM que armazenam o *next hop*.

A operação de busca foi implementada de modo a ilustrar as etapas executadas pela TCAM descrita anteriormente. Dado um endereço IP de destino, é executado um laço que percorre as linhas da TCAM, que são os elementos da tabela. Se esse endereço casa com algum prefixo, a linha associada é armazenada em uma lista *matchlist*, que ao final dessa operação contém todas as linhas cujos prefixos casaram com o aquele endereço IP. Em seguida, as fases LPM descritas anteriormente são executadas sobre essa lista através de um laço em que cada volta representa uma fase LPM. Em cada fase é analisada uma célula LPM. Se alguma célula contém um '1' armazenado, todas as outras linhas da *matchlist* que não contém o valor '1' armazenado na célula LPM correspondente, são eliminadas da *matchlist* e não são mais consideradas nas fases seguintes. Dessa forma, depois de todas

```c
#define NUM_TCAM_CELLS 512
#define NUM_SRAM_CELLS NUM_TCAM_CELLS
#define NUM_LPM_CELLS 9
#define NUM_BLOCK_ROWS 32
#define ADDR_SIZE 32 // address size, for ipv4 is 32 bits

typedef struct tcam_cell {
    unsigned char p_bit; //prefix bit
    unsigned char m_bit; //mask bit
} tcam_cell;

typedef unsigned char lpm_cell;

typedef unsigned char sram_cell;

typedef struct tcam_row {
    int index;
    int last_match;
    tcam_cell tcam_cl[NUM_TCAM_CELLS];
    lpm_cell lpm_cl[NUM_LPM_CELLS];
    sram_cell sram_cl[NUM_SRAM_CELLS];
} tcam_row;

typedef struct tcam_block {
    tcam_row tcam_r[NUM_BLOCK_ROWS];
} tcam_block;
```

Figura 14: Estruturas utilizadas na implementação da TCAM.

as fases, a unica linha que restará na *matchlist* será a linha que armazena o maior prefixo dentre as que casaram com o endereço IP usado como chave de busca. Portanto, se mais de um elemento da tabela casar com a chave, as fases LPM garantem que somente um deles será retornado como resultado da busca.

A operação de atualização da tabela é bastante simples, pois o esquema de fases para determinar o LPM possibilita que os elementos sejam inseridos em qualquer lugar da tabela. Na implementação, inicialmente um novo elemento é adicionado na próxima posição livre da tabela. Quando a TCAM já está preenchida, novos elementos são adicionados no lugar daqueles que foram menos recentemente acessados. A atualização será discutida mais detalhadamente nas próximas sessões.

3.4 Unidade de Busca

A Unidade de Busca (UB), é a parte do NP responsável pelas buscas que são realizadas. A busca é a tarefa de procurar um determinado elemento em uma tabela que case com uma chave, geralmente retornando alguma informação associada com aquele elemento. Neste trabalho, a chave utilizada é o endereço IP de destino, e a busca na tabela visa encontrar um prefixo IP que case com esse endereço e obter o *next hop*, que é a informação associada ao prefixo IP no elemento da tabela. Na implementação, é levada em conta a busca na tabela de roteamento apenas, mas nos NPs em geral essas unidades podem ser utilizadas

para busca em outras estruturas, tais como tabelas de endereços MAC.

A Figura 7 (na página 13) ilustra a relação da UB com os outros componentes do modelo. Como foi discutido anteriormente, as informações extraídas dos pacotes e as informações de roteamento geradas são inseridas em uma fila comum, que é a fila de entrada da UB. A UB retira as informações dessa fila, e se ela for de um pacote IP comum, a UB realiza a busca na tabela de roteamento. Dessa operação de busca é retornado o *next hop* para aquele pacote e um contador associado à porta de saída correspondente é incrementado. Assim, ao final da simulação, temos o número de pacotes que são encaminhados para cada porta de saída. Se a UB retirar de sua fila de entrada uma informação de roteamento, ela a coloca em outra fila, a fila de atualização.

A atualização da tabela de roteamento em roteadores reais é executada sempre que um pacote de controle com informações de roteamento é recebido e contém informações novas sobre as rotas ou atualizações nas informações já existentes. As operações que são executadas após receber essas informações dependem do protocolo de roteamento implementado. Neste trabalho, a atualização é feita sempre que uma informação de roteamento é retirada da fila de entrada da Unidade de Busca e não está presente na tabela. Em roteadores reais, o processamento de informações de roteamento e a atualização das tabelas é efetuada independentemente da busca. Para refletir esse comportamento, as informações de roteamento são inseridas em uma fila separada, a fila de atualização, e seu processamento é realizado por uma outra *thread* que retira as informações dessa fila.

Na TCAM original, as informações são inseridas em qualquer posição livre. Os elementos da tabela geralmente têm um tempo de vida associado, gerenciado por um temporizador (*timer*). Quando esse cronômetro chega a zero, esse elemento é removido e a posição correspondente na TCAM fica livre. Neste trabalho, as informações são inseridas seqüencialmente até que a TCAM seja preenchida. Cada elemento tem em sua estrutura um campo *lastmatch* que indica a ordem de chegada do último pacote que casou com aquele elemento que é quando ele foi acessado pela última vez. Isso é feito através de um contador de pacotes. Sempre que a UB faz uma busca na tabela e a chave casa com algum elemento, o valor do *lastmatch* associado com este elemento é atualizado com o valor do contador de pacotes quando o pacote em questão chegou ao roteador. Se a tabela já estiver cheia e uma nova informação de roteamento é recebida, esta é inserida na posição da tabela menos recentemente usada (*least-recently- used (LRU)*), na posição do elemento com o menor *lastmatch*.

3.5 Resultados

Na simulação realizada neste trabalho foi utilizado um *log* de pacotes que atravessaram um roteador real. O programa recebe como entrada 240.000 pacotes desse *log*, que são considerados como se fossem recebidos pelo roteador simulado. Como descrito em seções anteriores, a tabela de roteamento é gerada durante o recebimento de pacotes e cada pacote IP recebido resulta em uma busca na tabela. O *log* foi dividido em quatro trechos de 60.000. Para capturar o segundo trecho, são reiniciadas as estruturas para que os trechos distintos

funcionem como execuções separadas, como se fossem quatro logs de 60.000 pacotes como entrada para quatro simulações distintas. A tabela 3 mostra o número de pacotes IP dentre os 60.000 pacotes, bem como algumas informações referentes à classificação de pacotes como o número de pacotes TCP e UDP. Na tabela, LOG_i representa o trecho i correspondente do *log* utilizado.

	LOG_1	LOG_2	LOG_3	LOG_4
Número de pacotes IP	59532	59797	59753	59781
Número de pacotes UDP	42137	50438	34869	28278
Número de pacotes TCP	17366	9085	24859	31483
Número de pacotes ARP	196	92	128	104

Tabela 1: *Informações sobre a simulação utilizando diferentes logs como entrada.*

Foram feitas várias simulações com tamanhos diferentes para a tabela de roteamento. Os valores das tabelas **??** e **??** são uma média aritmética simples de 4 execuções utilizando as mesmas configurações, tais como o tamanho da tabela. Foram feitas simulações utilizando 60.000, 120.000, 240.000 e 300.000 pacotes como entrada, mas a tabela mostra os resultados obtidos através da simulação com 300.000 pacotes dos quais 298.627 são pacotes IP. A tabela **??** mostra os resultados obtidos fixando o intervalo entre as atualizações dos pacotes como 140, assim a cada 140 pacotes recebidos o simulador gera uma atualização na tabela. Já a tabela **??** mostra os resultados obtidos com o simulador configurado para gerar atualizações na tabela com um intervalo de pacotes aleatório entre uma e outra, esse intervalo varia entre 51 e 649 pacotes e busca aproximar o modelo do funcionamento real, no qual não dá pra saber ao certo quando um roteador receberá um pacote de controle com informações de roteamento para atualizar sua tabela. Percebemos na tabela **??** que a taxa de acertos na tabela de roteamento além de ser influenciada pelo aumento no número de elementos, também é influenciada pelo número de atualizações.

Tamanho da tabela	**32**	**64**	**128**	**256**	**512**
Atualizações na tabela	2142	2142	2142	2142	2142
Acertos na tabela	264854	269067	271480	272312	274512
Taxa de acertos	88,69%	90,10%	90,90%	91,19%	91,92%

Tabela 2: *Dados referentes a simulações variando o tamanho da tabela entre 32 e 512 elementos, utilizando como entrada um log de 300.000 pacotes e intervalo fixo de pacotes entre uma atualização e outra da tabela de roteamento.*

A tabela 4 mostra os 16 primeiros elementos da tabela de roteamento ao final de uma simulação. Podemos ver na tabela alguns dos prefixos que foram gerados na execução, o campo LPM e o *next hop* associados a cada prefixo.

Ao final da execução é contabilizado o número de pacotes que é encaminhado para cada porta de saída. A tabela 5 mostra a distribuição dos 59532 pacotes IP entre as 4 portas de saída para a simulação do LOG_1. Os pacotes cujos endereços IP de destino não casam

Tamanho da tabela	**32**	**64**	**128**	**256**	**512**
Atualizações na tabela	1162	1443	1451	1412	1370
Acertos na tabela	262230	267138	265146	266336	271171
Taxa de acertos	87,81%	89,45%	88,78%	89,18%	90,80%

Tabela 3: *Dados referentes a simulações variando o tamanho da tabela entre 32 e 512 elementos, utilizando como entrada um log de 300.000 pacotes e intervalo variável de pacotes entre uma atualização e outra da tabela de roteamento.*

Prefixo	**LPM**	*Next Hop*
10101100000100000XXXXXXXXXXXXXXXX	000010000	010
10101100000100000111 0XXXXXXXXXXX	000010101	000
10101100000100000XXXXXXXXXXXXXXXX	000001111	011
10101100000100000111 00XXXXXXXXXX	000010110	001
10101100000100000111 0XXXXXXXXXXX	000010101	000
10101100000100000111 000XXXXXXXXX	000010111	011
10101100000100000XXXXXXXXXXXXXXXX	000001111	001
10101100000100000111 00XXXXXXXXXX	000010110	001
10000111000010000011 110XXXXXXXXX	000010111	001
10101100000100000 1XXXXXXXXXXXXXX	000010010	001
10101100000100000XXXXXXXXXXXXXXXX	000001111	000
10101100000100000 11XXXXXXXXXXXXX	000010011	001
10101100000100000XXXXXXXXXXXXXXXX	000010001	001
10101100000100000111 00XXXXXXXXXX	000010110	001
10101100000100000 11XXXXXXXXXXXXX	000010011	001
11000101110110101 0XXXXXXXXXXXXXX	000010010	011

Tabela 4: *Alguns elementos da tabela de roteamento.*

com nenhum prefixo na tabela são encaminhados para a porta 0, que é a porta *default*. Os dados dessa tabela foram obtidos na mesma execução que gerou a tabela de roteamento da tabela 4.

	Porta 0	**Porta 1**	**Porta 2**	**Porta 3**
Número de pacotes	13334	3178	39	42980

Tabela 5: *Número de pacotes que foram encaminhados para cada porta em uma execução do simulador.*

Neste trabalho são estudadas diferentes arquiteturas de *Network Processors (NPs)*, seus componentes internos e o contexto que motivou seu desenvolvimento. É dada atenção especial ao *parsing* e ao roteamento dos pacotes. O simulador foi implementado com o objetivo de analisar as estruturas internas, tais como a TCAM e as tarefas atribuídas à unidades funcionais presentes em vários NPs, tais como o *parser* e a Unidade de Busca, bem como a relação destes com os demais componentes. O *parser* é implementado para

funcionar como um classificador de pacotes do PowerNP, recolhendo informações relevantes e encaminhando para a próxima etapa do processamento, que neste caso é realizada pela Unidade de Busca. A tabela 3 mostra somente o número de pacotes IP, TCP, UDP e ARP, que foram recolhidas através da classificação dos pacotes, mas qualquer informação referente aos pacotes pode ser obtida através do *parsing* dos pacotes. Neste trabalho, embora muitas informações sejam recolhidas, somente os pacotes IP são considerados, e desses pacotes o endereço IP de destino é utilizado como chave de busca pela Unidade de Busca. A TCAM foi implementada respeitando a sua estrutura interna estudada e a forma com que é realizada a sua operação de busca foi simulada com o objetivo de aproximar, na simulação, as diferentes etapas executadas para efetuar a operação de busca em *hardware*.

4 Conclusão e Trabalhos Futuros

4.1 Conclusão

Os roteadores desempenham tarefas de grande importância na *Internet*. O seu correto funcionamento garante que cada pacote chega ao seu destino, no tempo em que deveria chegar. Com o crescimento da *Internet*, novas aplicações com novos requisitos de qualidade de serviço foram incorporadas às redes e isso exigiu maior poder de processamento e flexibilidade nos roteadores, para que estes agregassem novas funcionalidades sem comprometer o seu desempenho. Os *Network Processors* são processadores especializados no processamento de pacotes e buscam atender às necessidades das novas aplicações.

Neste trabalho foram estudados os *Network Processors (NPs)*, suas diferentes unidades funcionais e formas de implementação e organização. Foram discutidas tarefas que devem ser executadas por esses processadores e como algumas delas são mapeadas para a arquitetura de um NP. Foram discutidos trabalhos que apresentam formas diferentes de implementar e organizar as estruturas presentes em um NP, nos quais são analisados os custos e benefícios de cada solução. Uma dentre as estruturas discutidas foi memória que armazena a tabela de roteamento, sendo a TCAM uma das mais eficientes.

No simulador foram implementados um *parser* de pacotes, a tabela de roteamento, bem como outras estruturas que possibilitam a busca e atualização da tabela. Através da simulação foi possível entender melhor essas tarefas. A implementação da tabela de roteamento foi baseada na TCAM projetada por [6], buscando aproximar a simulação das tarefas realmente executadas e de sua estrutura interna em *hardware*. Por fim, são apresentados alguns resultados que foram obtidos ao final de uma simulação.

4.2 Trabalhos Futuros

Simular os roteadores em condições mais próximas da realidade, criar uma rede na qual os algoritmos de busca nas tabelas e as arquiteturas de memória possam ser configuradas nos diferentes nós com o objetivo de comparar o desempenho entre elas. O modelo de arquitetura do roteador e da tabela de roteamento devem ter uma relação mais próxima

do funcionamento real para que o resultado da simulação possa ser utilizado para avaliar o desempenho de roteador real de forma mais confiável. Essa rede deve funcionar como uma rede normal, implementando protocolos de roteamento e trocando pacotes em determinadas velocidades e instantes.

Referências

[1] P. Crowley; M. E. Fiuczynski; J.-L. Baer; and B. N. Bershad. Characterizing Processor Architectures for Programmable Network Interfaces. *Proc. 2000 Int'l Conf. Supercomputing*, pages 54–65, May 2000.

[2] H. Liu. Routing Prefix Caching in Network Processor Design. *Proc. International Conference on Computer Communications and networks (ICCCN)*, 2001.

[3] A. McAuley and P. Francis. Fast Routing Table Lookup Using CAMs. *Proc. Infocom 93*, Vol. 3:pp. 1382–91, Mar. 1993.

[4] K. Pagiamtzis and A. Sheikholeslami. Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. *IEEE Journal of Solid-State Circuits*, vol. 41(no. 3):pp. 712–727, March 2006.

[5] Miquel Pericàs. Simultaneous Multithreading: Present Developments and Future Directions. `http://personals.ac.upc.edu/mpericas/trabajo_smt.pdf`, Jun 2003.

[6] B. Gamache; Z. Pfeffer; and S. P. Khatri. A Fast Ternary CAM Design for IP Networking Applications. *12th International Conference on Computer Communications and Networks (IC3N-03)*, October 2003.

[7] Niraj Shah. Understanding Network Processors. Technical report, Department of Electrical Engineering and Computer Science, University of Berkeley, September 2001.

[8] J. Allen; B. Bass; C. Basso; R. Boivie; J. Calvignac; G. Davis; L. Frelechoux; M. Heddes; A. Herkersdorf; A. Kind; J. Logan; M. Peyravian; M. Rinaldi; R. Sabhikhi; M. Siegel; and M. Waldvogel. IBM PowerNP Network Processor: Hardware, Software, and Applications. *IBM J. Research and Development*, vol. 47(nos. 2/3):pp. 177–194, 2003.

[9] Cisco Systems. Parallel eXpress Forwarding in the Cisco 10000 Edge Service Router. *White Paper*, October 2000. `http://www.cisco.com/en/US/prod/collateral/routers/ps133/prod_white_paper09186a008008902a.pdf`.

[10] T. Wolf and M. A. Franklin. Performance Models for Network Processor Design. *IEEE Transactions on Parallel and Distributed Systems*, vol. 17(no. 6):pp. 548–561, Jun. 2006.