Instituto Superior Técnico
Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Algoritmia e Desempenho Em Redes de Computadores

Encaminhamento de Pacotes na Internet

David Fialho n.º 73530
Nuno Mendes n.º 73716

Lisboa, 19 de Outubro de 2015

1 Introdução

O problema do encaminhamento de pacotes é cada vez mais relevante nos dias de hoje, dado ao enorme tráfego (em crescimento) que percorre a internet. Por esta razão, os encaminhadores têm que ser muito eficientes a determinar para onde encaminhar os pacotes que recebem. Para tal a tabela de encaminhamento deve ser representada numa estrutura de dados que permite uma pesquisa rápida o suficiente. Além deste factor, também se tem que ter em conta a memória utilizada para armazenar a tabela uma vez que os encaminhadores utilizam memória muito rápida e por isso com capacidade muito limitada.

Neste trabalho são apresentadas e discutidas duas estruturas de dados diferentes - Árvore Binária e Árvore 2-Binária.

2 Árvores Binárias

2.1 Inserção de Prefixo para interface n

A inserção de um prefixo numa árvore binária é feito em tempo polinomial com uma complexidade de $O(n)$ em que n é o número de bits do prefixo a ser inserido.

2.2 Remoção de prefixo para interface n

Neste caso a remoção de um prefixo para uma interface específica também é realizada em tempo polinomial com complexidade $O(n+m)$, onde n é o número de bits do prefixo e m é o número de *saltos* para chegar ao primeiro nó ascendente (vindo do nó com profundidade n) cujo seu valor n exista.

```
node = root
visited = Stack
lookup(prefix,visited)
if node has nexthop then
    delete nexthop foreach (previous,bit) in visited do
        if not exists left and right child and has not nexthop then
            if bit is '1' then
                remove right child
            else
                remove left child
            node = previous
    end
```

Algoritmo 1: Remoção de um prefixo numa árvore binária.

2.3 Pesquisa de interface destino para endereço ip

Para determinar a interface de destino de um pacote com destino em *ip*, é realizada uma procura pelo prefixo mais específico em comum com *ip*, podendo terminar o algoritmo

em qualquer ramo da árvore, se esse for o mais específico encontrado. A complexidade da execução deste algoritmo é $O(n)$, em que n é o número de bits que *ip* tem em comum com o maior prefixo correspondente da árvore, sabendo que $0 \leq n \leq 32$.

```
node = root; hop = default
foreach bit in ip, from MSB do
    if bit is '1' then
        if node has nexthop then
            hop = node.nexthop
        if node has no right child then
            return hop
        move to right child
    else
        if node has nexthop then
            hop = node.nexthop
        if node has no left child then
            return hop
        move to left child
end
```

Algoritmo 2: Procura de interface destino numa árvore binária.

2.4 Vantagens e Desvantagens da implementação

Este tipo de implementação permite guardar prefixos para interfaces que estejam omissas por um prefixo mais longo, possui também tempos de inserção, remoção e pesquisa polinomiais. Para fazer uma impressão da tabela de expedição é apenas necessário fazer uma impressão em pré-ordem dos nós da árvore. Porém, este tipo de implementação não optimiza a memória utilizada ao longo da pesquisa pelo maior prefixo comum a um endereço.

3 2-Árvores Binárias

3.1 Pesquisa de interface destino para endereço ip

A pesquisa numa árvore deste género tem uma complexidade $O(n)$, onde n é o maior prefixo em comum com o endereço ip. A pesquisa nesta estrutura de dados optimiza a memória utilizada durante a execução do algoritmo, pois só nas folhas da árvore é que se acede ao valor da interface destino.

```
node = root; hop = none
foreach bit in ip, from MSB do
    if node not exists then
        break
    if node has hop then
        hop = node.nexthop
    if bit is '1' then
        move to right child
    else
        move to left child
    return hop
end
```

Algoritmo 3: Procura de interface destino numa 2-árvore binária.

4 Compressão Da Árvore Binária

Durante a construção da tabela de encaminhamento, utilizando uma árvore binária, dependendo da ordem com que os prefixos são adicionados a árvore resultante pode não ser a mais compacta possível, o que pode resultar numa utilização menos eficiente da memória e a pesquisas com maiores profundidades. O algoritmo ORTC (Optimal Routing Table Constructor), desenvolvido pela Microsoft em 1999 [1], permite transformar uma tabela de encaminhamento noutra equivalente, sem qualquer perda de informação, que pode ser representada com o número mínimo de nós possível [1]. Este algoritmo é composto pelos 3 passos seguintes:

1. Normalização da árvore
2. Cálculo dos next-hops mais comuns
3. Seleção do next-hop para cada nó da árvore

No primeiro passo a árvore é normalizada propagando os next-hops para os filhos, o que resulta numa 2-trie. Este passo foi implementado percorrendo em pré-ordem a árvore binária. O pseudo-código é apresentado abaixo:

```
foreach node in nodes(pre-order) do
    if node has next-hop then
        inherited = node.next-hop
    else
        node.next-hop := inherited
end
```

Algoritmo 4: Primeiro passo no algoritmo ORTC.

O segundo passo calcula os next-hops mais comuns em cada nível da árvore através da seguinte operação:

$$A\#B = \begin{cases} A \cap B & \text{if } A \cap B \neq \emptyset \\ A \cup B & \text{if } A \cap B = \emptyset \end{cases}$$

Este passo foi implementado percorrendo a árvore por níveis com recurso a uma fila com todos os nós da árvore ordenados por nível. O pseudo-código é apresentado abaixo:

```
foreach node in nodes(from leaves to root) do
    if node is parent node then
        node.next-hops := node.left.next-hops # node.right.next-hops
end
```

Algoritmo 5: Segundo passo no algoritmo ORTC.

No terceiro e último passo é escolhido, dentro do conjunto de next-hops de cada nó (determinado no passo 2), o next-hop adequado para cada nó. Durante este processo os nós com informação redundate são eliminados. Este passo foi implementado percorrendo a árvore em pré-ordem e abaixo encontra-se o seu pseudo-código:

```
inherited := ∅
foreach node in nodes(pre-order) do
    if node is not root and inherited ∈ node.next-hops then
        node.next-hop := ∅
    else
        node.next-hop := choose(node.next-hops) inherited := node.next-hop
end
```

Algoritmo 6: Terceiro passo no algoritmo ORTC.

Pela descrição dos algoritmos pode verifcar-se que este algoritmo percorre todos os nós da árvore 3 vezes, tendo então uma complexidade O(3N) aprox.. O(N), onde N é o número de nós na árvore binária.

5 Conclusão

Neste trabalho foi implementada uma tabela de encaminhamento em árvore binária que permite um uso de memória eficiente e uma pesquisa de endereços com complexidade de O(log(n)), em que n são o número de nós na árvore. Com tudo verificou-se que esta estrutura de dados, mesmo com compressão ou utilizando uma 2-trie, apresenta alguns defeitos que podem ser melhorados.

Um dos problemas mais evidentes vem do facto de se utilizar o caminho na árvore para representar os prefixos, isto resulta na existência de nós de suporte (nós brancos) que não contém qualquer informação mas ocupam espaço desnecessário em memória e aumentam profundidade da pesquisa, podendo esta atingir 32 níveis no caso de IPv4 e 128 no caso de IPv6.

Na tentativa de resolver este problema foi feita alguma investigação e verificou-se que existem outras estruturas de dados que permitem reduzir ainda mais o tempo de pesquisa de um endereço.

A estrutura mais eficiente encontrada foi a LC Trie [2] que permite reduzir o número de nós de suporte ao mínimo o que resulta num uso de memória muito mais eficiente e uma pesquisa de endereços com complexidade O(log(log(n))), onde n é o número de prefixos na tabela. No entanto esta estrutura tem um custo de construção muito mais elevado do o de uma árvore binária e durante a sua construção informação da tabela pode ser perdida. Isto significa que será necessário utilizar memória extra para armazenar esta informação, podendo isto resultar num uso ineficiente da memória. Na Internet dos dias de hoje o tempo disponível para a alteração de uma tabela de encaminhamento é da ordem dos milisegundos (1000 vezes mais do que a pesquisa), sendo assim o desempenho limitado desta estrutura no que toca à modificação da árvore pode não ser um factor limitativo para a sua utilização.

Conclui-se então, que a utilização de árvores binárias como estrutura base de tabelas de encaminhamento não é a melhor solução de todas no que toca à pesquisa de endereços, mas é uma das mais simples de implementar e apresenta um uso de memória e um tempo de pesquisa bastante reduzidos. Além disto, uma das suas propriedades mais importantes é que pode ser criada e alterada sem perda de informação não sendo necessária a utilização de memória adicional para guardar informação perdida.

Referências

[1] R. P. Draves, C. King, S. Venkatachary, B. N. Zill: Constructing Optimal IP Routing Tables Technical Report MSR-TR-98-59

[2] J. Fu, O. Hagsand and G. Karlsson: Improving and Analyzing LC-Trie Performance for IP-Address Lookup. Journal of networks, vol. 2, no. 3, June 2007