高效 IP 路由查找实验

孙佳钰 2015K8009929051

2018 年 11 月 11 日

1 实验内容

在路由器的转发过程中，需要根据目的 IP 查找路由表，来确定从哪个端口转发出去。本实验要实现如下内容：

- 单比特前缀树查找：每次只比对 1 个比特进行查找，我实现了最简单的单比特前缀树。
- 多比特前缀树查找：每次比对两个比特进行查找，我实现了增加了叶推的 2 比特前缀树。
- 多比特前缀树的优化：对多比特前缀树进行压缩指针与压缩向量，我直接对叶推后的 2 比特前缀树进行了优化。

2 实验流程

由于代码太多，故报告中没有加入完整代码，完整代码可见附件。

2.1 单比特前缀树查找

1. 单比特前缀树结点定义为：

```
typedef struct TNode {
    struct TNode *LNode, *RNode;
    u8 port;
} TNode, *Tree;
```

2. 建树过程中，对读取到的深度进行循环。在新建结点的过程中，不特别区分中间结点与叶子结点，若该结点不是该 ip 对应的叶子结点，则将它的端口号赋为它父结点的端口号；循环到要求深度后，即到达该 ip 对应的叶子结点时，才将其端口号赋为读到的端口号。部分代码如下：

```
for (int j = 0; j < ii->length; j++) {
    if (ii->ip & (ip1 >> j)) {
```

```
        if (!node->RNode) {
            TNode *node_tmp = (TNode*)malloc(sizeof(TNode));
            node_tmp->LNode = node_tmp->RNode = NULL;
            node_tmp->port = node->port;
            node->RNode = node_tmp;
        }
        node = node->RNode;
    } else {
        if (!node->LNode) {
            TNode *node_tmp = (TNode*)malloc(sizeof(TNode));
            node_tmp->LNode = node_tmp->RNode = NULL;
            node_tmp->port = node->port;
            node->LNode = node_tmp;
        }
        node = node->LNode;
    }
}
node->port = ii->port;
```

3. 查找过程中，由于是最长前缀查找，故一直查找到与所查 ip 所对应的叶子结点为止，此时前缀最长。代码如下：

```
u8 lookup_pref_tree(Tree tr, u32 ip) {
    u32 ip1 = 1 << 31;
    TNode *node = tr;
    TNode *node_tmp = NULL;
    for (int i = 0; node; i++) {
        node_tmp = node;
        node = (ip & (ip1 >> i)) ? node->RNode : node->LNode;
    }
    return node_tmp->port;
}
```

2.2 多比特前缀树查找

1. 多 (2) 比特前缀树结点定义为：

```
typedef struct TNode_pro {
    struct TNode_pro *LLNode, *LRNode, *RLNode, *RRNode;
    u8 port;
```

```
} TNode_pro, *Tree_pro;
```

2. 建树过程中，同样不区分中间结点与叶子结点，中间结点的端口赋值与单比特类似。当读取到的长度为奇数时，最下一层需要建两个相同的叶子结点都赋为读到的端口号，此时需要最后单独讨论。部分代码如下：

```
for (int j = 0; j < ii->length-1; j ++) {
    if (ii->ip & (ip1 >> j)) {
        if (ii->ip & (ip1 >> ++j)) {
            if (!node->RRNode) {
                TNode_pro *node_tmp = (TNode_pro*)malloc \
                                                    (sizeof(TNode_pro));
                init_node_pro(node_tmp, node->port);
                node->RRNode = node_tmp;
            }
            node = node->RRNode;
        } else {
            ...
        }
    } else {
        ...
    }
} // for j
if (ii->length % 2) {
    if (ii->ip & (ip1 >> (ii->length-1))) {
        if (!node->RRNode) {
            TNode_pro *node_tmp = (TNode_pro*)malloc(sizeof(TNode_pro));
            init_node_pro(node_tmp, ii->port);
            node->RRNode = node_tmp;
        }
        ...
    } else {
        ...
    }
} else {
    node->port = ii->port;
}
```

3. 由于建树时中间结点的端口号处理成与父结点相同，故叶推时只需简单递归即可。

4. 查找过程与单比特类似，只需改成一次匹配 2 比特即可。代码如下：

```
u8 lookup_pref_tree_pro(Tree_pro tr, u32 ip) {
    u32 ip1 = 1 << 31;
    TNode_pro *node = tr;
    TNode_pro *node_tmp = NULL;
    for (int i = 0; node; i++) {
        node_tmp = node;
        if (ip & (ip1 >> i)) {
            node = (ip & (ip1 >> ++i)) ? node->RRNode : node->RLNode;
        } else {
            node = (ip & (ip1 >> ++i)) ? node->LRNode : node->LLNode;
        }
    }
    return node_tmp->port;
}
```

2.3 多比特前缀树的优化

1. 优化后的 2 比特前缀树结点定义为：

```
typedef struct TNode_comp {
    u8 type;
    u8 port;
    struct TNode_comp *ptr_0, *ptr_1;
} TNode_comp, *Tree_comp;
```

2. 根据叶推后的 2 比特前缀树来建树的过程中，我将指针与向量压缩同时进行，只需处理好 bit array 的高低位与指针所指数组的索引之间的关系即可。代码如下：

```
void build_pref_tree_comp(Tree_pro tr2, Tree_comp tr3){
    tr3->type = (tr2->LLNode->LLNode)? 1 : 0;
    tr3->type = tr3->type << 1 | ((tr2->LRNode->LLNode)? 1 : 0);
    tr3->type = tr3->type << 1 | ((tr2->RLNode->LLNode)? 1 : 0);
    tr3->type = tr3->type << 1 | ((tr2->RRNode->LLNode)? 1 : 0);
    int num_inter = __builtin_popcount(tr3->type);
    int num_leaf = 4 - num_inter;
    tr3->ptr_0 = (TNode_comp*)malloc(num_leaf * sizeof(TNode_comp));
    tr3->ptr_1 = (TNode_comp*)malloc(num_inter * sizeof(TNode_comp));
    int i_inter = 0;
    int i_leaf = 0;
    for (int i = 0; i < 4; i++) {
```

```
13          if (tr3->type & (1 << i)) {
14              build_pref_tree_comp(i_to_node(tr2, i), &(tr3->ptr_1[i_inter]));
15              i_inter++;
16          } else {
17              tr3->ptr_0[i_leaf].port = i_to_node(tr2, i)->port;
18              i_leaf++;
19          }
20      }
21  }
```

3. 查找过程中的重点也是要处理好 bit array 的高低位与指针所指数组之间的关系。

3 实验结果及分析

```
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ make
gcc -Wall -g main.c -o pref-tree
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree has 1646585 nodes, takes 39518040 B.
For all ips in test-table.txt, pref_tree takes about 0.783002 seconds.
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree has 1646585 nodes, takes 39518040 B.
For all ips in test-table.txt, pref_tree takes about 0.778389 seconds.
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree has 1646585 nodes, takes 39518040 B.
For all ips in test-table.txt, pref_tree takes about 0.787349 seconds.
```

上图是单比特前缀树的结果，总结点数使用递归遍历来计数，所占空间为总结点数乘结构体大小。

```
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ make
gcc -Wall -g main.c -o pref-tree
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree_pro has 4810993 nodes, takes 192439720 B.
For all ips in test-table.txt, pref_tree_pro takes about 0.770852 seconds.
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree_pro has 4810993 nodes, takes 192439720 B.
For all ips in test-table.txt, pref_tree_pro takes about 0.774113 seconds.
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree_pro has 4810993 nodes, takes 192439720 B.
For all ips in test-table.txt, pref_tree_pro takes about 0.782435 seconds.
```

上图是实现叶推后的 2 比特前缀树的结果。

```
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ make
gcc -Wall -g main.c -o pref-tree
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree_comp has 4810993 nodes, takes 115463832 B.
For all ips in test-table.txt, pref_tree_comp takes about 0.856770 seconds.
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree_comp has 4810993 nodes, takes 115463832 B.
For all ips in test-table.txt, pref_tree_comp takes about 0.865831 seconds.
sjy@sjy-PC:/media/sjy/新加卷/国科大/网络实验/09-lookup$ ./pref-tree
For all ips in forwarding-table.txt, pref_tree_comp has 4810993 nodes, takes 115463832 B.
For all ips in test-table.txt, pref_tree_comp takes about 0.869653 seconds.
```

上图是实现了指针和向量压缩后的 2 比特前缀树的结果。