

高效IP路由查找实验

中国科学院大学
袁欣怡 2018K8009929021
2021.5.20

实验内容

基于 forwarding-table.txt 数据集(Network, Prefix Length, Port), 实现高效 IP 路由查找算法, 具体包括:

1. 实现最基本的前缀树查找;
2. 调研并实现某种 IP 前缀查找方案;
3. 比较后者在占用内存和运算时间上的提升。

实验流程

1. 搭建实验环境

本实验中涉及到的文件主要有:

- code
 - simple
 - forwarding-table.txt # 数据集
 - lookup.c # 最基本的前缀树查找
 - Makefile # 处理make all和make clean命令
 - run.sh # shell脚本, 运行可执行文件10次
 - optimize
 - forwarding-table.txt # 数据集
 - lookup.c # 实现改进方法Leaf Pushing
 - Makefile # 处理make all和make clean命令
 - run.sh # shell脚本, 运行可执行文件10次

2. 实验代码设计

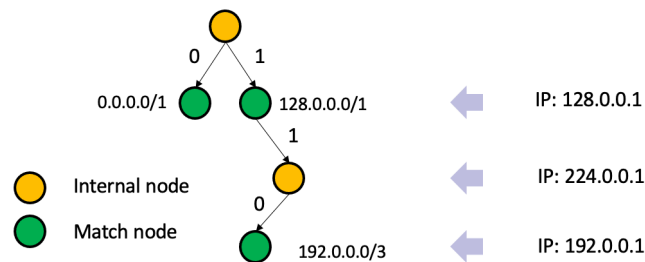
基础设计

路由器在进行路由转发时, 需要查找路由表来确定是否有匹配的条目。在之前的实验中, 我们已经实现了基础的 IP 查找方法。但是随着网络规模的扩大, 线性查找的时间和空间开销都迅速增加, 导致效率降低。因此, 我们需要减少查找次数, 实现更高效的查找算法。

我们这里实现的 1-bit 前缀树有以下特点:

1. 根结点不对应任何 IP，除根结点外每个结点对应一个 IP；
2. 从某结点到它的子结点的路径对应一个字符，从根结点到某一结点的路径上所有字符连接起来，就是该结点对应的 IP；
3. 每个结点对应的 IP 各不相同。

构造出的前缀树如下图所示：



结点数据结构：

```
1 struct TreeNode{
2     int net_id;
3     int prefix_len;
4     int port;
5     struct TreeNode *lchild;
6     struct TreeNode *rchild;
7     struct TreeNode *parent;
8 };
```

相应的，为路由表项也设计了数据结构：

```
1 struct RouterEntry{
2     int net_id;
3     int prefix_len;
4     int port;
5 };
```

在我们的实验中，需要先扫描 `forwarding-table.txt` 中的路由表项，构造前缀树，然后进行查找，并计算所需要的时间和空间。代码如下：

```
1 int main() {
2     FILE * fd = fopen("forwarding-table.txt", "r"); //读文件
3
4     if (fd==NULL){
5         printf("ERROR: File does not exist!");
6         return 0;
7     }
8
9     // 建立前缀树，计算需要的空间
10    char * line = (char *) malloc(MaxLine);
11
12    struct TreeNode * root=NULL;
13    root = init_tree();
14}
```

```

15     while(fgets(line, MaxLine, fd)!=NULL){
16         struct RouterEntry * router=NULL;
17         router = line_parser(line);
18         add_node(router, root);
19     }
20
21     printf("INFORMATION: Memory used: %ld B =%ld KB =%ld
22     MB\n",mem_use,mem_use/1024,mem_use/1024/1024);
23
24     // 查找前缀树，计算需要的时间
25     double time_use;
26     double handle_time;
27     double avg_time;
28     struct timeval start, end;
29     int input_num=0;
30
31     //calculate time needed for processing input
32     fseek(fd, 0, SEEK_SET);
33     gettimeofday(&start, NULL);
34     while(fgets(line,MaxLine,fd) != NULL){
35         char * net_id=NULL;
36         net_id = strtok(line, " ");
37         char_to_int(net_id);
38         input_num++;
39     }
40     gettimeofday(&end, NULL);
41     time_use = (end.tv_sec-start.tv_sec)*1E9+(end.tv_usec-start.tv_usec)*1E3;
42     handle_time = time_use / input_num;
43
44     //calculate time needed in total
45     input_num=0;
46     fseek(fd, 0, SEEK_SET);
47     gettimeofday(&start, NULL);
48     while(fgets(line,MaxLine,fd) != NULL){
49         char * net_id=NULL;
50         net_id = strtok(line, " ");
51         lookup(char_to_int(net_id), root);
52         input_num++;
53     }
54     gettimeofday(&end, NULL);
55     time_use = (end.tv_sec-start.tv_sec)*1E9+(end.tv_usec-start.tv_usec)*1E3;
56     avg_time = time_use / input_num;
57
58     printf("INFORMATION: Average time needed: %.9f ns\n", avg_time-handle_time);
59
60     return 0;
61 }

```

下面来看 1-bit 前缀树的建立和查找。

(1) 1-bit 前缀树的建立

先初始化前缀树的根结点 `root`，然后从数据集中读取表项。从高位开始，如果为 0 则访问左子结点（若不存在则新建一个左子结点），如果为 1 则访问右子结点（若不存在则新建一个右子结点），以此类推，直到访问的长度达到掩码为止。这个过程中同时需要注意设置端口号。

具体代码实现如下：

```
1  int add_node(struct RouterEntry * router, struct TreeNode * root){
2      struct TreeNode * ptr = root;
3      int mask = 0x80000000;
4      unsigned int prefix_bit = 0x80000000;
5      int insert_num = 0;
6      int i=0;
7
8      for(i=1; i<=router->prefix_len; i++){
9          if ( (router->net_id & prefix_bit)==0 ){
10             if (ptr->lchild == NULL){
11                 ptr->lchild = insert_tree(router, i, ptr);
12                 insert_num++;
13             }
14             ptr = ptr->lchild;
15         }
16         else{
17             if (ptr->rchild == NULL){
18                 ptr->rchild = insert_tree(router, i, ptr);
19                 insert_num++;
20             }
21             ptr = ptr->rchild;
22         }
23
24         mask = mask>>1;
25         prefix_bit = prefix_bit>>1;
26     }
27
28     if (insert_num==0)
29         printf("net:%x update -> port: %d\n", router->net_id&mask, ptr->port);
30
31     if ((router->net_id&mask) != (ptr->net_id&mask))
32         return -1;
33
34     free(router);
35     mem_use -= sizeof(struct RouterEntry);
36
37     return ptr->port;
38 }
```

(2) 1-bit 前缀树的查找

从根结点出发，与 `net_id` 的最高位开始匹配。如果最高位为 0 则访问根结点的左子结点，否则访问右子结点，以此类推，最后返回端口号。需要注意的是，如果端口号为 -1，说明查询路径最终落在中间结点上，则要调用 `look_back` 函数进行回溯。

```
1  int lookup(int net_id, struct TreeNode *root){
2      struct TreeNode * ptr=root;
3      unsigned int prefix_bit = 0x80000000;
4      int i;
5      int port;
6
7      for (i=1; i<32; i++){
8          if ((net_id & prefix_bit) == 0){
9              if (ptr->lchild == NULL){
10                 if ((net_id&get_mask(ptr->prefix_len)) != (ptr->net_id&get_mask(ptr->prefix_len))){
11                     port = -1;
12                     break;
13                 }
14                 port = ptr->port;
15                 break;
16             }
17             ptr = ptr->lchild;
18         }
19         else{
20             if (ptr->rchild == NULL) {
21                 if ((net_id&get_mask(ptr->prefix_len)) != (ptr->net_id&get_mask(ptr->prefix_len))){
22                     port = -1;
23                     break;
24                 }
25                 port = ptr->port;
26                 break;
27             }
28             ptr = ptr->rchild;
29         }
30         prefix_bit = prefix_bit >> 1;
31     }
32
33     if ((net_id&get_mask(i)) != (ptr->net_id&get_mask(i)))
34         port = -1;
35
36     if(port == -1)
37         port = look_back(ptr);
38
39     return port;
40 }
```

`look_back` 函数的功能是通过回溯找到最近的最长前缀匹配的网络号，并返回该网络对应的转发端口。具体代码如下：

```

1  int look_back(struct TreeNode * node){
2      struct TreeNode * ptr = node;
3
4      while (ptr->port == -1)
5          ptr = ptr->parent;
6
7      return ptr->port;
8  }

```

基于Leaf Pushing的优化

基础的查找函数中，`TreeNode` 结构中需要存储 `prefix_len` 、 `net_id` 和 `parent` ，但这三项其实都可以删除，这样可以简化 `TreeNode` 结构，减少占用的内存空间。

新设计的`TreeNode`结构：

```

1  struct TreeNode{
2      int port;
3      struct TreeNode *lchild;
4      struct TreeNode *rchild;
5  };

```

针对这个新的结构，设计的算法也需要进行修改。具体如下：

`add_node` 函数和 `insert_tree` 函数：

```

1  int add_node(struct RouterEntry * router, struct TreeNode * root){
2      struct TreeNode * ptr = root;
3      // int mask = 0x80000000;
4      unsigned int prefix_bit = 0x80000000;
5      int insert_num = 0;
6      int i=0;
7
8      for(i=1; i<=router->prefix_len; i++){
9          if ( (router->net_id & prefix_bit)==0 ){
10             if (ptr->lchild == NULL){
11                 ptr->lchild = insert_tree(router, i, ptr);
12                 insert_num++;
13             }
14             ptr = ptr->lchild;
15         }
16         else{
17             if (ptr->rchild == NULL){
18                 ptr->rchild = insert_tree(router, i, ptr);
19                 insert_num++;
20             }
21             ptr = ptr->rchild;
22         }
23
24         // mask = mask>>1;

```

```

25     prefix_bit = prefix_bit>>1;
26 }
27
28 /*
29 if (insert_num==0)
30     printf("net:%x update -> port: %d\n ", router->net_id&mask, ptr->port);
31
32 if ((router->net_id&mask) != (ptr->net_id&mask))
33     return -1;
34 */
35
36 free(router);
37 mem_use -= sizeof(struct RouterEntry);
38
39 return ptr->port;
40 }
41
42 // 新的insert_tree函数只保存端口号。倘若某个结点是中间结点，则只需要继承父结点的端口号即可
43 struct TreeNode * insert_tree(struct RouterEntry * router, int prefix_len, struct TreeNode *parent){
44     struct TreeNode * node = (struct TreeNode *) malloc(sizeof(struct TreeNode));
45     mem_use += sizeof(struct TreeNode);
46
47     // node->net_id = router->net_id & get_mask(prefix_len);
48     // node->prefix_len = prefix_len;
49     if (router->prefix_len != prefix_len)
50         // node->port = -1;
51         node->port = parent->port;
52     else
53         node->port = router->port;
54
55     // node->parent = parent;
56     return node;
57 }

```

lookup 函数：删除了回溯查找的部分。因为中间结点保存了最近的最长前缀匹配结果所对应的端口号，所以只要进行简单的前缀查找即可。

```

1 int lookup(int net_id, struct TreeNode *root){
2     struct TreeNode * ptr=root;
3     unsigned int prefix_bit = 0x80000000;
4     int i;
5     int port;
6
7     for (i=1; i<32; i++){
8         if ((net_id & prefix_bit) == 0){
9             if (ptr->lchild == NULL){
10                 /*
11                 if ((net_id&get_mask(ptr->prefix_len)) != (ptr->net_id&get_mask(ptr->prefix_len))){
12                     port = -1;

```

```

13         break;
14     }
15     */
16     port = ptr->port;
17     break;
18 }
19 ptr = ptr->lchild;
20 }
21 else{
22     if (ptr->rchild == NULL) {
23         /*
24         if ((net_id&get_mask(ptr->prefix_len)) != (ptr->net_id&get_mask(ptr->prefix_len))){
25             port = -1;
26             break;
27         }
28         */
29         port = ptr->port;
30         break;
31     }
32     ptr = ptr->rchild;
33 }
34 prefix_bit = prefix_bit >> 1;
35 }
36
37 /*
38 if ((net_id&get_mask(i)) != (ptr->net_id&get_mask(i)))
39     port = -1;
40
41 if(port == -1)
42     port = look_back(ptr);
43 */
44
45 return port;
46 }

```

3. 实验结果分析

基础前缀树查找：

运行 `run.sh` 脚本，将 `lookup` 执行十遍，实验结果：


```
Terminal
shawn@shawn: /mnt/hgfs/share/10-lookup/simple
File Edit View Search Terminal Help
shawn@shawn:~$ cd /mnt/hgfs/share/10-lookup/
shawn@shawn:/mnt/hgfs/share/10-lookup$ cd simple
shawn@shawn:/mnt/hgfs/share/10-lookup/simple$ make clean
shawn@shawn:/mnt/hgfs/share/10-lookup/simple$ make all
gcc -Wall -g lookup.c -o lookup
shawn@shawn:/mnt/hgfs/share/10-lookup/simple$ sudo ./run.sh
[sudo] password for shawn:
1:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 96.704027328 ns
2:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 140.876251286 ns
3:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 104.365781035 ns
4:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 190.976411485 ns
5:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 85.745154625 ns
6:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 106.658432228 ns
7:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 112.501826956 ns
8:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 107.071109443 ns
9:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 109.368059357 ns
10:
INFORMATION: Memory used: 31143456 B =30413 KB =29 MB
INFORMATION: Average time needed: 106.115360476 ns
shawn@shawn:/mnt/hgfs/share/10-lookup/simple$
```

平均内存开销为：31143456B = 30413KB = 29MB

平均花费时间为：116.04ns

基于Leaf Pushing优化

运行 `run.sh` 脚本，将 `lookup` 执行十遍，实验结果：

```
Terminal
shawn@shawn: /mnt/hgfs/share/10-lookup/optimize
File Edit View Search Terminal Help
shawn@shawn:~$ cd /mnt/hgfs/share/10-lookup/
shawn@shawn:/mnt/hgfs/share/10-lookup$ cd optimize/
shawn@shawn:/mnt/hgfs/share/10-lookup/optimize$ make clean
shawn@shawn:/mnt/hgfs/share/10-lookup/optimize$ make all
gcc -Wall -g lookup.c -o lookup
shawn@shawn:/mnt/hgfs/share/10-lookup/optimize$ sudo ./run.sh
[sudo] password for shawn:
1:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 94.240860203 ns
2:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 93.763702173 ns
3:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 94.391315437 ns
4:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 85.570339972 ns
5:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 81.768837712 ns
6:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 236.615932206 ns
7:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 83.767742971 ns
8:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 53.494716072 ns
9:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 91.257547838 ns
10:
INFORMATION: Memory used: 11384436 B =11117 KB =10 MB
INFORMATION: Average time needed: 89.860463517 ns
shawn@shawn:/mnt/hgfs/share/10-lookup/optimize$
```

平均内存开销为：11384436B = 11117KB = 10MB

平均花费时间为：100.47ns

比较

平均内存开销减少明显，平均花费时间略有减少。

原因：1. **TreeNode** 结点内容减少了一半左右，导致内存开销大幅降低；2. 查找过程中减少回溯的次数，但是可以减少的次数不多，因此时间有减少但不太明显。

参考资料

1. 数据结构与算法：字典树（前缀树）：<https://zhuanlan.zhihu.com/p/28891541>
2. 计算机网络复习——路由器和路由查找算法：https://blog.csdn.net/not_simple_name/article/details/103745833