

计算机网络实验报告-12

阮星程

2015K8009929047

一、 实验题目

网络路由实验二。

二、 实验内容

理解 OSPF 协议构建路由表的原理，完成构建的第二步——利用 Dijkstra 算法计算最短路径，基于实验一构建的完整链路数据库，得到完整的路由表。

三、 实验过程

本次实验的主要代码在程序 `mospf_daemon.c` 中，主要构架了一个新的函数，`generate_rt()`，函数的作用是根据已有的链路数据库，构建出新的路由表，由于路由表条目在设计时没有超时的记录结构，一种做法是额外建立另一种表，与路由表同步维护，负责记录对应路由表项的存在时间；另一种是每次调用函数的时候不在意已经生成了的路由条目，直接重新生成。但第一种存在问题，不论这个新的数据结构效率怎样，在我们试图去添加路由条目的时候，为了决定这个路由条目是否在路由表中已有，这就要带来 $O(n^2)$ 的计算量，有些浪费，不如直接添加来的方便，所以选择了第二种做法。

Dijkstra 算法本身很简洁，在构建时也没有遇到什么问题，比较麻烦的反倒是构建起最初的距离条目以及邻接矩阵。由于本身数据结构的影响，在不考虑偷懒直接 `memcpy` 然后强行把原有的某些字段做其它用处的情况下，搬运与收集数据确实成了一个繁杂的工作，而且在链表数据结构的影响下，复杂度也是节节高升，让人总算是理解到了数据结构的重要性。

至于代码方面，这次我觉得没有太多需要展示的部分，毕竟只是按照 Dijkstra 算法逐步实现罢了，值得一提的是在 `rtable` 中增加了一个锁，避免了在发包时出现程序进入无限循环的情况。按道理来说这里没有死锁可以构成，找了一下之后还是没有注意到什么特殊的地方。在 `rtable` 中增加锁后，各程序就能正常有序地运行了。

接下来还是按照常规把写的核心代码放上来，加以简单的解释。

```

dist = (struct dist_entry * )malloc((rnum + 1)*sizeof(struct dist_entry));
struct dist_entry * tmp = dist;
u32 graph[rnum][rnum];
memset(graph, 0, rnum*rnum*sizeof(u32));
//fprintf(stdout,"phase init router dist list\n");

//init router dist list
list_for_each_entry(db, &mospf_db, list){
    tmp->rid = db->rid;
    tmp->visited = 0;
    tmp->dist = MAX_DIST;
    tmp->gw = BAD_GW;
    if(tmp->rid == instance->router_id){
        //fprintf(stdout,"phase init router dist list if\n");
        tmp->dist = 0;
        tmp->visited = 1;
        tmp->gw = 0;
    }
    else{
        //fprintf(stdout,"phase init router dist list else\n");
        for(i = 0; i < self_db->nadv; i++){
            if(tmp->rid == self_db->array[i].rid){
                tmp->dist = 1;
                tmp->gw = tmp->rid;
                break;
            }
        }
    }

    tmp++;
}

```

为了方便 Dijkstra 算法的实现，我新增加了一个数据结构

```

struct dist_entry{
    u32 rid;
    u32 dist;
    int visited;
    u32 gw;
};

```

在这里对这个距离条目进行初始化，gw 项负责记录到达这个节点所要通过的网关。从上图中可知，对于远处的节点，初始化 rid，visited 置 0，dist 置为最大距离，gw 先置为 BAD_GW，对于源节点，dist 置 0，visited 置 1，gw 置 0。对于源节点的邻居节点，gw 置为他们自身，距离置 1，作为初始化条件，方便我们在之后的循环中扩散各自对应的 gw。

```

//fprintf(stdout,"phase init graph\n");
//init graph(containing edges and distances between neighbours)
k = 0;
list_for_each_entry(db, &mospf_db, list){
    lsa = db->array;
    for(i = 0; i < db->nadv; i++){
        j = 0;
        list_for_each_entry(db1, &mospf_db, list){
            if(lsa[i].rid == db1->rid){
                break;
            }
            else
                j++;
        }
        graph[k][j] = 1;
    }
    k++;
}

```

初始化邻接矩阵。

```

//fprintf(stdout,"phase main loop\n");
//main loop of dijkstra algorithm
for(i = 0; i < rnum - 1; i++){
    min_dist = MAX_DIST;
    min_j = 0;

    //find closest unvisited router
    for(j = 0; j < rnum; j++){

        //if not visited
        if(dist[j].visited == 0){
            if(dist[j].dist < min_dist){
                min_dist = dist[j].dist;
                min_j = j;
            }
        }
    }

    dist[min_j].visited = 1;

    //change the distance of min_j's neighbours
    for(j = 0; j < rnum; j++){
        if(graph[min_j][j] && dist[j].visited == 0 && graph[min_j][j] + dist[min_j].dist
            < dist[j].dist){
            dist[j].dist = graph[min_j][j] + dist[min_j].dist;
            dist[j].gw = (dist[min_j].gw < dist[j].gw) ? dist[min_j].gw : dist[j].gw;
        }
    }
}

```

Dijkstra 算法的核心循环，构建起了到各路由器的最短路径。

```

//fprintf(stdout,"phase transfer\n");
rt_entry_t * rt;
j = 0;
int found = 0;
list_for_each_entry(db, &mospf_db, list){
    for(i = 0; i < db->nadv; i++){
        found = 0;
        list_for_each_entry(rt, &rtable, list){
            if(db->array[i].subnet == rt->dest ){
                found = 1;
                if(rt->dist > dist[j].dist){
                    rt->dist = dist[j].dist;
                    rt->gw = dist[j].gw;
                    rt->iface = (dist[j].gw ? gw_to_iface(dist[j].gw) : subnet_to_iface(db->
                    if(!rt->iface){
                        log(WARNING,"gw to iface miss, within transfer found loop");
                        break;
                    }
                    rt->mask = rt->iface->mask;
                    break;
                }
                //log(DEBUG,"found and not exchange,ip "IP_FMT" gw "IP_FMT" %d",HOST_IP_FMT_S
                break;
            }
        }

        if(!found){
            //log(DEBUG,"not found,ip "IP_FMT" gw "IP_FMT" %d",HOST_IP_FMT_STR(db->array[i].s
            if(!gw_to_iface(dist[j].gw)){
                if(dist[j].gw != 0){
                    log(WARNING,"gw to iface miss, within transfer not fdound loop, gw:"IP_FM
                    continue;
                }
                rt = new_rt_entry(db->array[i].subnet, db->array[i].mask, 0, subnet_to_iface(
                add_rt_entry(rt);
                continue;
            }
            rt = new_rt_entry(db->array[i].subnet, ((dist[j].gw) ? gw_to_iface(dist[j].gw) :
            add_rt_entry(rt);
            rt->dist = dist[j].dist;
        }
    }
}

```

最后再将路径转化为对应的路由表条目。

四、 实验结果

