第 34 卷 第 2 期 计 算 机 工 程 2008 年 1 月
Vol.34 No.2 Computer Engineering January 2008
·网络与通信· 文章编号：1000—3428(2008)02—0171—04 文献标识码：A 中图分类号：TP393

SNMP 协议动态 MIB 结构与高效查找算法

周 剑，张晓彤，王 沁
(北京科技大学计算机科学与技术系，北京 100083)

摘 要：SNMP 是一种简单网络管理协议，作为 HFC 网络管理的重要组成部分，其资源利用率和执行效率的高低对被管系统的整体性能有重要影响。而 SNMP 软件的主要性能瓶颈在于保存和查找 MIB 对象的效率。该文通过分析 MIB 的结构特点，对比目前普遍使用的 Hash 表法，提出了使用多路径树来保存完整 MIB 结构的方法，消除公共 ID 的重复存储，提高资源利用率。在此基础上，给出一种综合使用多路径树和 AVL 树的高效查找算法，它能够有效地提高 MIB 对象的查找效率。该方法已经成功地应用于自主开发的“双向有线 HDTV 信道传输系统”中，实际网络验证表明，该方法具有较高的查找性能和可靠性。

关键词：HFC 网络管理；SNMP 协议；AVL 树

Dynamic MIB Structure of SNMP and High Performance Search Algorithm

ZHOU Jian, ZHANG Xiao-tong, WANG Qin
(Dept. of Computer Science and Technology, University of Science and Technology Beijing, Beijing 100083)

【Abstract】SNMP protocol is a simple network management protocol. As an important part of most network management softwares, its resource wastage and efficiency have important influence on the performance of the whole system. According to study, the main bottleneck of SNMP software is MIB objects' storage and search. This paper proposes to use multi-path tree to keep the entire structure of MIB to eliminate the repeated storage of mutual indentifiers by analysing the structure of MIB tree. Compared with widely used Hash table, multi-path tree promotes the efficiency of resource usage. On the basis of that, this paper also proposes a high speed algorithm to search MIB objects by the combination of multi-path tree and AVL tree. The method above has been successfully applied to the bidirectional HDTV signal transmitting system and it is proved to be efficient and reliable in actual network environment.

【Key words】HFC network management; Simple Network Management Protocol (SNMP); AVL tree

随着数字电视技术在全世界的兴起和单向传输的模拟电视向交互式数字电视的转变，基于HFC网络的、具有双向通信功能的网络芯片和交互式机顶盒已成为交互式数字电视和其他数字综合业务的核心技术和关键设备之一。在符合DOCSIS规范的数字电视设备中，为满足网络管理的需要，使用了SNMP[1-2]协议。目前已有的开发工具在效率上无法满足机顶盒这类具有复杂的基于DVB-C和DOCSIS规范的物理层、MAC层的嵌入式环境的要求。因此，提高嵌入式SNMP系统的处理效率和资源利用率就成为开发过程中的关键问题。本文以北京市科技重大项目“双向有线HDTV信道传输核心技术研究”为研究背景，开发支持DOCSIS[3-4]和以太网规范的SNMP协议，以满足管理数字电视网络机顶盒的需要。本文对SNMP协议中MIB(Management Information Base)的特点进行分析和归类，重点研究MIB对象OID的存储和检索问题，提出有效的解决方案，并在此基础上实现了一种以多路径树和AVL树为主要手段的查找方法，提高了查找效率。

1 问题分析

1.1 MIB 树的结构

SNMP 的性能主要与 MIB 的搜索效率有关。按照 SMIv2(SMI version 2)[5]的规则，MIB中所有被管实体在概念上组成了一个如图 1 所示的树状结构，树结构从未命名的根开始。每个被管对象都具有唯一的名字，以对象标识符(Object Identifier, OID)作为标识，采用点分形式的整数序列定义，这些被点分割开的整数分别代表从根结点到对象结点所经过路径上所有结点的ID号，查找MIB对象的过程就是在MIB树中搜索OID的过程。

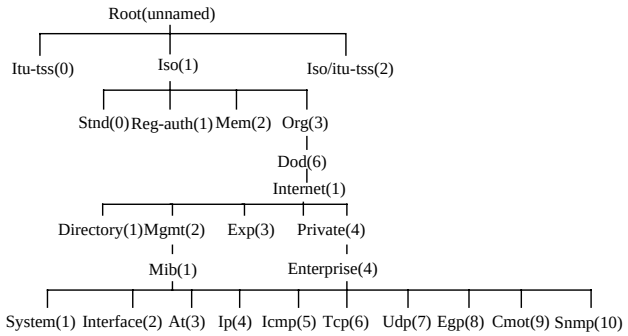

图 1 MIB 树形结构及 MIB-Ⅱ中的对象分类

为简化问题，本文将讨论范围缩小到单个 MIB 对象的

基金项目：北京市科技基金资助重大项目“交互式数字电视信道传输核心技术开发”(京科技发[2002]188 号)

作者简介：周 剑 (1982－)，男，硕士，主研方向：计算机网络与通信，嵌入式系统；张晓彤，副教授、博士；王 沁，教授、博士、博士生导师

收稿日期：2007-01-22 **E-mail：**bluesu37@163.com

GetRequest 和 GetNextRequest 操作。这是因为写操作与读操作的流程基本相似，同时其他操作都与 GetNextRequest 实现原理相似。

许多 SNMP 代理实现方案都通过 Hash 函数将请求中包含的 OID 映射到 Hash 表中一个 MIB 对象的数据结构上。这种方法虽然有一定的优点，即通过单纯的数学计算和一定的冲突检测可以在短时间内找到目标结点，但缺点也较为突出：

(1)由于 Hash 函数冲突很多，因此需要保存 MIB 对象的完整 OID 以便在冲突发生时判断当前对象是否是目标对象。由图 1 可以看出，保存完整 OID 将会造成公共祖先结点 ID 的重复。

(2)Hash 函数很难实现有序遍历。Hash 表法对个数相对固定的简单结点还有可能按照录入顺序来排序，对动态变化的表格数据则无法进行操作。因此，查找表格数据时，使用链表搜索效率低下。

本文使用多路径树维持 MIB 的组织形式，节省 OID 的存储空间，并利用 AVL 树来提高表格数据查找效率。

1.2 OID 的组成

为说明 OID 的概念，引入如下定义：

定义 1 设从MIB树的根结点到任一结点n_p的路径为P，该路径经过的结点依次为 $n_0\cdots n_i\cdots n_p(p\in Z,p>0)$，它们的$id$分别为$id_0\cdots id_i\cdots id_p$，$id_i(0\le i\le p)$为整数且大于等于 0；则结点$n$的OID可以表示为

$$oid_n=(id_0\cdot id_1\cdots id_i\cdots id_{p-1}\cdot id_p) \tag{1}$$

MIB 树中的结点可分为 4 类：简单结点，数据结点，表格 *entry* 结点和组结点。简单结点代表被管实体，不保存实际数据而只能包含一个数据结点。数据结点分两种：简单数据结点和表格数据结点。简单数据结点与简单结点一一对应，它保存实际数据。表格数据结点保存对应表格中的某行数据。表格 *entry* 结点指向表格数据的入口。组结点也不保存数据，但组内可以包含 1 个以上子结点，子结点可以是上述 4 种结点的任意一种。还有一种表格结点，本文将它也看作组结点的一种。除表格数据结点外，其他类型结点 OID 都按定义 1 得出。

定义 2 设任意表格 t 有 α 行 β 列数据，$\alpha,\beta\in Z$ 且 $\alpha\ge 0$，$\beta\ge 1$。设第$c_1\cdots c_i(1\le i\le\beta)$列为索引列，第 m 行 n 列单元格对象表示为 $t(m,n)$，它的值为 $V_{(m,n)}$，则该行索引值定义为

$$index_m=V_{(m,c_1)}\cdot V_{(m,c_2)}\cdots V_{(m,c_{i-1})}\cdot V_{(m,c_i)} \tag{2}$$

设表格 t 的 OID 为 oid_t，$t(m,n)$ 的 OID 值为 $oid_{(m,n)}$，则：

$$oid_{(m,n)}=oid_t\cdot id_{entry}\cdot n\cdot index_m \tag{3}$$

其中，id_{entry} 是表格对应的表格 *entry* 结点 *id* 值。

由定义 1 和定义 2 可知，表格数据结点和其他结点的 OID 值获取方式是不同的。这是因为若表格没有数据，则只需存储表格入口，无需数据节点。若表格有数据，由定义 2 可知，目标 OID 可分成 oid_t，id_{entry}，n 和 $index_m$ 4 部分。考虑有 r 行 l 列数据 $(r,l\in Z,r>0,l>2)$ 的表格 t，共有 $r\times l$ 个不同的 OID。如果每个单元格都完整地存储它的 OID，$index_m$ 将被重复存储 l 次，n 将被重复存储 r 次，而 oid_t 和 id_{entry} 将被重复存储 $r\times l$ 次。当表格行数目很大的时候，会带来很大的内存开销，同时搜索树的结点数和键值的比较时间也将大大增加。

2 改进的 MIB 存储策略

2.1 非表格数据结点的多路径树存储

本文使非表格数据结点都只保存它本身的 ID，并通过 MIB 中的父子关系来查找对象。将组结点的所有子结点都以数组存储，同时设计相应的数据结构记录组结点应有子结点个数及子结点数组指针，于是构成一棵多路径树，其形状与 MIB 结构完全相同。这样做的好处是每个结点只存储一次，且维持了 MIB 的树形结构。然后，就可以通过逐个匹配请求 OID 中各个子 ID 来查找目标。

设请求 OID 为 $(id_0\cdots id_n)$，其中 $n\ge 0$，则目标对象存在的充分必要条件是：对于任意 $0\le i<n$，设 id_i 对应结点 a，则 a 必有一子结点 b 的 ID 为 id_{i+1}。这是由 MIB 的性质决定的，因为如果请求 OID 有对应对象存在，则所有的子 ID 都要有对应的对象。

设 $i=0$，MIB 根结点为未命名结点 *unnamed*，它是个组结点。当前 MIB 结点为 *cur*，初始 *cur* = *unnamed*，每个结点对应的结构体中都包含一个 *id* 字段来表示它们自己的 ID 值，在不考虑表格 *entry* 结点的情况下，查找算法如下：

(1)设 *cur* 的子结点数组为 *array*，大小为 k。若 *array* 中存在下标为 $s1$ 的元素，使 $array[s1].id=id_i$，令 $cur=array[s1]$，i 加 1，转步骤(2)，否则返回空值。

(2)若 $i>n$ 且 *cur* 是数据结点，返回 *cur*，否则返回空值。若 $i\le n$，转步骤(1)。

算法中认为简单结点的子结点是对应的简单数据结点，而数据结点没有子结点。另外由于 SNMP 协议的主要目的是获取 MIB 对象的值，而非数据结点是不存储数据的，因此如果请求 OID 对应的是非数据结点，则返回 NoSuchName 错误。

多路径树的可扩展性在于，如果实际工程需要增加或删除 MIB 结点，只需在编译前将该结点在相应的子结点数组中添加或删除，并修改数组大小即可。对于如何在 *cur* 的子结点数组中查找 id_i，本文使用下标法何二分法来提高效率。

2.1.1 下标法

若 MIB 组中成员的 ID 号是连续的，设组结点 g 的 OID 为 oid_g，共有 k 个子结点，若第 $i(1\le i\le k)$ 个子结点的 ID 号正好为 i，就可以将所有子结点以大小为 k 的数组保存，下标为 $(i-1)$ 的数组元素对应结点 OID 为 $oid_g\cdot i$。假设请求 OID 为 $oid_g\cdot p$；若 $p\le k$，则返回下标为 $(p-1)$ 的数组元素。若 $p>k$，查找失败。

使用下标法可以在一次减法运算后直接定位目标结点，有利提高效率，前提条件是子结点序号连续，而且是从 1 开始递增。

2.1.2 二分法

对于子结点序号不连续的现象，本文也尽可能提高其查找效率。仍然使用数组来存储子结点，并使用查找效率较高的二分法来匹配子结点序号。

设当前结点共有 k 个子结点，且第 $i(1\le i\le k)$ 个子结点的 ID 号为 id_i。查找 ID 为 p 的子结点算法如下：

算法 BinarySearch(*p*, *k*)

```
low ← 0; high ← k-1; mid ← 0;
while(low≤high)    {
        mid ← (low + high)/2;
        if(mid = low) mid ← high;
```

```
        if(idmid = p) return mid;
        if(idmid < p) low ← mid;
        else high ← mid;
        if((low + 1)== high)
            if(idlow == p) return low; else return -1;
    }
return -1;
```

若子结点数组包含 ID 为 p 的结点，BinarySearch(p, k)返回该结点下标，否则返回-1。

综上所述，本文主要利用了 MIB 结点的父子关系来避免 ID 的重复存储，通过数组的映射关系来提高效率，同时保持了易于理解和实现的组织形式。

2.2 表格数据的存储

如果表格数据的每一个单元格都通过完整 OID 来访问，会造成 OID 公共部分的大量重复。通过前面的分析，可以发现若只保存表格索引，既可以减少存储空间的使用，又使每一行只占一个搜索结点，提高查找效率。

因此为每个表格建立一棵单独的AVL搜索树，在表格 *entry* 结点中保存指向该搜索树根结点的指针，每个树结点中保存指向对应表格行的指针，并以行索引为键值，省掉对定义 2 中的oid_t, id_{entry}和n的存储。查找表格数据时首先通过多路径树找到对应的表格 *entry* 结点，然后以请求OID中索引部分作为目标结点的键值，查找对应的AVL树。AVL树的实现方法可以参考相关文献，本文不再赘述。

设请求 OID 为 $(id_0 \cdots id_n)$，其中 $n \ge 0$，沿用非表格数据查找的相关定义，将查找表格数据对象的过程如下描述：

(1)设 *cur* 的子结点数组为 *array*，大小为 k。若存在 $s1 < k$，使 $array[s1].id = id_i$。令 $cur = array[s1]$，i 加 1，转步骤(2)，否则返回空值；

(2)若 *cur* 不是表格 *entry* 结点，转步骤(3)，否则转步骤(4)；

(3)若 $i > n$，且 *cur* 是数据结点，返回 *cur*，否则返回空值。如果 $i \le n$，转步骤(1)；

(4)若 $i \le n$，则列号为 id_i，i 加 1，转步骤(5)。若 $i > n$，列号不存在，返回空值；

(5)设表格索引长度为 k。若 $n - i + 1 = k$，则以 $id_i \cdots id_n$ 为目标索引，搜索对应搜索树。若搜索结果不为空，查找成功，通过树结点指向的行指针，调用相应函数返回该行对应列值，否则返回空值。若 $n - i + 1 \ne k$，查找失败，返回空值。

3 MIB 的遍历

MIB 的遍历是 SNMP 操作的重要组成部分，GetNextRequest 报文就是为此定义的，作用是查找 PDU 中给定 OID 后继对象的值。

定义 3 设请求 OID 的值为 a，MIB 中所有 OID 大于 a 的数据结点集合为 $N = \{\delta \mid oid_\delta > a\}$，若某数据结点的 OID 为 oid_n，且对任意结点 $\delta \in N$ 有 $oid_n \le oid_\delta$，则该结点为请求的后继结点。

定义 3 说明，请求OID的后继对象就是所有OID大于请求OID的对象中OID最小的那个对象。图 2 中虚线描述了简单数据结点的遍历顺序[6-7]。图 3 中的箭头描述了表格数据结点的遍历顺序，即先列后行。参考定义 2 可知，列号也是影响字典序的重要因素。在列号不变的情况下，OID的后继结点就是目标结点；若未能找到后继结点，但列号未达到最大，将列号加 1 取下一列索引最小的表格单元。

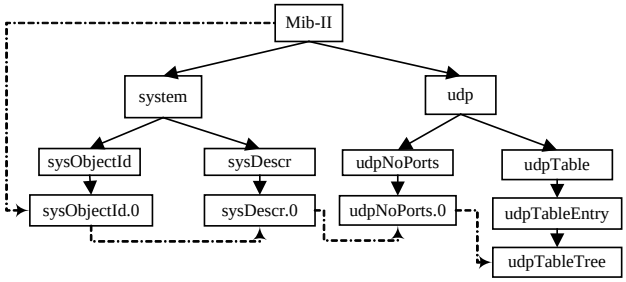

图 2 简单数据结点的遍历

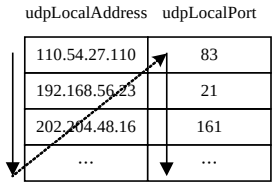

图 3 表格数据结点的遍历

只需中序遍历 AVL 树即可查找表格数据的后继结点。而对于非表格数据结点，可以利用递归查找下一个结点。设请求 OID 为 $(id_0 \cdots id_n)$, $n \ge 0$，沿用前面的相关定义，则查找后继对象步骤如下：

(1)若 *cur* 是简单结点或组结点，设其子结点数组为 *array*，大小为 k。若 $i \le n$，且 *array* 中存在下标为 $s1$ 的元素，使 $array[s1].id = id_i$，令 $cur = array[s1]$，i 加 1，递归调用本算法。若 $i > n$，则令 $cur = array[0]$，递归调用本算法(注意 i 是局部变量，在不同层次的递归函数中是不同的)。若未找到对应元素或递归调用返回结果为空，则在子结点数组中查找下标为 $s2$ 的元素，满足 $array[s2-1] < id_i < array[s2](1 \le s2 \le k-1)$ 或 $s2 = 0$ 且 $array[0] > id_i$，令 $cur = array[s2]$，递归调用本算法；若无满足条件的元素，则返回空值。

(2)若 *cur* 是表格 *entry* 结点，设该表格共有 m 列，当前列号为 λ，则分 3 种情况：若 $i > n$，令列号为 1，返回搜索树中索引最小的结点；若 $i = n$，令 $\lambda = id_i$，同样返回搜索树中索引最小的结点；若 $i + 1 \le n$，令 $\lambda = id_i$，在搜索树中查找索引刚好大于 $id_{i+1} \cdots id_n$ 的结点。若未找到这样的结点，分两种情况；若 $\lambda < m$，令 $\lambda = \lambda + 1$，查找搜索树中索引最小的结点，否则返回空值；若搜索结果不为空，查找成功，通过树结点指向的行指针，调用相应函数返回该行对应 λ 列的值，否则返回空值。

(3)若 *cur* 是数据结点，返回空值。

4 测试

本文的测试环境为 Windows XP 操作系统，CPU 主频为 1.84 GHz 的 AMD 处理器，内存 256 MB，并分别使用本文所述方法和 Hash 表法编程测试。其中 Hash 函数的计算方法是将请求 OID 中所有子 ID 相加除以 Hash 表大小(1001)的取余。

首先看内存消耗问题。在测试环境中，依次向 MIB 中添加 udp 组，ip 组，tcp 组，snmp 组，计算两种方法用于存储 OID 所消耗的内存大小。内存消耗曲线对比如图 4 所示。

对于查找效率问题，本文通过查找 udp 组中所有数据结点所消耗的时间来对比二者优劣。数据结点包括表格数据结点，重复查找 1 000 遍。此外，还通过不断增加表格数据行

数，来验证表格数据增加对查找时间的影响。查找时间与表格行数关系曲线如图 5 所示。

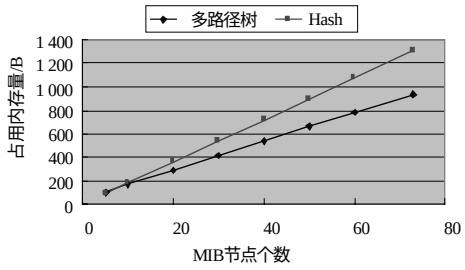

图 4　节点的内存消耗曲线

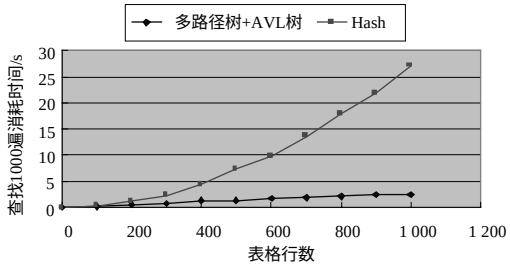

图 5　查找时间与表格行数关系曲线

由图 4 可以看出，随着结点个数的增加，两种方法消耗的内存量都呈线性增长，这是因为对于特定的 MIB 对象来说 OID 的长度是固定的，所以占用内存数与对象个数基本成正比关系。在图 4 中，两条直线的斜率之比约为 $12/18=2/3$，即当节点数大于一定数目后，多路径树法通常能节省大约 33.3%的空间。由图 5 可以看出，在没有表格数据，也就是仅查找普通结点的情况下，本文方法与 Hash 表法消耗时间接近，几乎都在约 1 ms 内完成。随着表格行数的递增，Hash 表法查找时间急剧增加，而 AVL 树的查找时间则增长缓慢且保持较小值。这说明使用 AVL 树存储表格数据能够显著提高查找效率。

5　结束语

测试结果表明，本文提出的实现方案和算法能够有效减少 OID 的存储空间，并大大提高动态 MIB 的查找效率，特别是表格数据的查找效率，解决了困扰 SNMP 项目的效率问题。对比 Hash 表法，利用结点的父子关系构造的多路径树在非表格数据结点的内存使用和查找效率方面都取得较好的效果，易于理解和实现。而在 Hash 表很难解决的表格数据遍历方面，使用 AVL 树不仅能够实现遍历，还能大大提高查找效率，使系统整体性能更好。

参考文献

[1] Case J D, Fedor M, Schoffstall M L, et al. A Simple Network Management Protocol (SNMP)[S]. RFC 1157, 1991.

[2] McCloghrie K, Rose M. Management Information Base for Network Management of TCP/IP-based Internets: MIB-II[S]. RFC 1213, 1991.

[3] Cable Television Laboratories Inc. Data-over-cable Service Interface Specifications-radio Frequency Interface Specification[Z]. SP-RFIv 1.1-I06-001215, 2002.

[4] 王　沁, 戴　鹏, 张晓彤, 等. 一种高效的计算带宽请求微时隙的算法[J]. 计算机学报, 2006, 29(5): 705-710.

[5] Larmouth J. ASN.1 Complete[Z]. (1999-10-20). http://www.oss.com.

[6] Breitgand D, Raz D, Shavitt Y. SNMP GetPrev: An Efficient Way to Browse Large MIB Tables[J]. IEEE Journal on Selected Areas in Communications, 2002, 20(4): 656-667.

[7] Seung-hyun Park, Myong-soon Park. An Efficient Transmission for Large MIB Tables in Polling-based SNMP[J]. Telecommunications, 2003, 23(1): 246-252.

(上接第 170 页)

为(1,1,1)，当再为某节点分配一个 MAC 地址时，按照前面所描述的 MAC 分配规则，为新节点分配 MAC 地址时的状态值应该是(1,1,1,1)，所生成的 MAC 地址值为 $210(=2^1\cdot3^1\cdot5^1\cdot7^1)$。但是，如在遵循状态改变位的前提下，为新节点分配 MAC 地址时的状态值可为(2,1,1)，即状态位移到末尾时，可翻转增加到第 1 位状态位上。当状态值为(2,1,1)时，所生成的 MAC 地址值为 $60(=2^2\cdot3^1\cdot5^1)$，远小于状态值为(1,1,1,1)时的 210。

3　结束语

为实现 MAC 地址的动态分配，其分配责任被转移到一个已配置好 MAC 地址的节点，称作 MAC 地址分配代理节点，它可根据当前状态值及有状态函数 $f(n)$为新到达的节点在整个移动自组网内分配唯一的 MAC 地址，且具有较高的鲁棒性。但当 MAC 地址的取值范围限制在 1 B 所表示的范围时，本文的方法仅能用于规模较小的网络，且未能很好地解决 MAC 地址回收的问题，这将是今后的工作目标。

参考文献

[1] Johnson D B, Maltz D A. Dynamic Source Routing in Ad Hoc Wireless Networks[M]. [S. l.]: Kluwer Academic Publishers, 1996.

[2] Droms R. Dynamic Host Configuration Protocol[EB/OL]. (1997-03-12). http://www.ietf.org/rfc/rfc2131.txt.

[3] Thomson S, Narten T. IPv6 Stateless Address Autoconfiguration Request for Comments[S]. RFC2462, 1998-12.

[4] Gunes M, Reibel J. An IP Address Configuration Algorithm for Zeroconf. Mobile Multi-hop Ad-Hoc Networks[C]//Proceedings of the International Workshop on Broadband Wireless Ad Hoc Networks and Services. Sophia Antipolis, France: [s. n.], 2002-09.

[5] Vaidya N H. Weak Duplicate Address Detection in Mobile Ad Hoc Networks[C]//Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing. Switzerland: ACM Press, 2002-06.

[6] Weniger K. Passive Duplicate Address Detection in Mobile Ad Hoc Networks[C]//Proc. of WCNC'03. Florence, Italy: ACM Press, 2003-02.

[7] Zhou H, Ni L M, Mutka M W. Prophet Address Allocation for Large Scale Manets[C]//Proc. of IEEE INFOCOM'03. San Francisco, CA: IEEE Press, 2003-03.