Asynchronous Resource Discovery

Ittai Abraham Danny Dolev †

January 22, 2004

# Abstract

考虑一个动态的，大规模的通信基础设施（如Internet）的节点（例如，在点对点系统）可以沟通与节点的ID（如IP地址），他们被称为唯一的基本建筑之一块这样一个分布式系统是资源的发现 - ..有效地发现系统中目前存在的节点的ID，我们提出的资源发现问题的上限和下限范围对于由Harchol巴尔特顿提出的原始问题，卢因​​我们提出了一个Ω（n logn）消息复杂度较低，其大小是未知的异步网络的约束。对于这个模型，我们给出渐近消息的优化算法，提高了Kutten和皮莱格位复杂[2][3]。当每个节点知道其连接元件的大小，我们提供接近线性O(nα(n, n))消息复杂（逆阿克曼的功能在哪里）1新型和高效的算法。此外，我们定义和研究广告特别资源发现问题，这是一个原始的问题的实际放松。我们特设的资源发现算法具有接近线性为O(nα(n, n))消息复杂，该算法有效地处理系统与动态节点增加，从而解决[2]开放的问题，我们从经典的联盟，查找问题，减少证明特设的资源发现问题的约束Ω(nα(n, n))较低，因此，我们特设资源发现算法是渐近消息最优。

# 1 Introduction

当一个动态的分布式系统中的多个节点要进行交互和合作，所需的基本构建块之一是需要一种机制用于节点发现对方节点的存在，这种机制还必须管理迅速节点增加活力和节点清除。本来Harchol，巴尔特，礼顿，和列文[2]中定义的分布式学习系统的节点的ID的问题，并把它命名为资源发现。显然，这个问题是重要的发展像Akamai这样的系统上（<http://www.akamai.com>）。

在许多P2P系统出现问题时，在互联网上的点最初知道只有少数同行。在这种情况下，资源发现算法，可用于为了有效地发现，弱相互连接的所有的节点。一旦所有感兴趣的节点，了解他们可能互相合作，共同任务（例如：同行可划分一个复杂的计算工作，或可能会建立一个覆盖网络，形成一个分布式哈希表[7，8，11，6作为一个关键的构建块]）。资源发现可用于修复受损的点对点系统。考虑一个系统中的许多节点复位或从系统中完全删除。朝着重建这样一个系统的第一步是发现重组目前所有网上的节点。

我们开始正式定义“知识图”，考虑n个节点的集合V，每个节点v拥有独特的标识符id(v)包含O(log n) 位组成。此标识符可以被认为该节点的IP地址的。

\_This research was supported in part by NDS PhD fellowship program.School of Engineering and Computer Science. The

Hebrew University. Jerusalem, Israel. Email :ittaia@cs.huji.ac.il

†This research was supported in part by Intel COMM Grant - Internet Network/Transport Layer & QoS Environment (IXA).

School of Engineering and Computer Science. The Hebrew University. Jerusalem, Israel. Email: [dolev@cs.huji.ac.il](mailto:dolev@cs.huji.ac.il)

网络是仿照作为一个虚拟的有向图G =（V，E），其中边缘（u→v）ϵ E表示节点u知道id(v)。一个节点u可以将消息发送到节点v只有当（U→V）ϵ E。 表示为E0的初始边缘; ......除非另有说明，我们不承担（V，E0）是强连通的。消息发送可以是任意长度，并可能包含其他节点的边集的ID，边缘集合E的增长，随着每次一个节点接收了它不知道节点ID，因此，当一个节点v接收到一个包含id（w），则E:= E∪ [{（v→ w）}。

这个正式的“知识图”系统，可以模拟计算互联网，在互联网上，每个节点都有一个唯一的IP地址，TCP / IP路由协议创建一个完全连接的底层网络，一旦IP地址被一个节点所知晓，节点可能使用TCP / IP协议将消息路由到目的地的IP地址，它知道。因此节点u后得知节点v的IP地址，节点u可以发送消息到节点v。“知识图”模型假设信息费用平等，这可能代表建立一个TCP / IP连接的固定费用，而忽视通过TCP / IP引起跳数的变化。

回想一下，两个节点属于弱连接的相同的组件，如果它们之间有一个路径中引起的无向图中，我们忽略了边缘方向。两个节点U，V属于强连接的相同的组件，是否有向路径从u到V，和直接从v到u的路径。

Resource Discovery，一个分布式资源发现算法，努力实现以下：

1。在每一个弱连接的组件之一节点被指定为领导者。  
2，领导者节点知道其组件中的所有节点的ID。  
3，所有的节点都知道他们的领导人的ID。

资源发现算法测量三种常见的复杂性措施：消息的总数，发送位的总数，并完成数轮（同步模式）。

强连通的网络，为O（n）消息复杂领导人 （戈帕尔，Cidon，并Kutten[1]）选举算法可以使用。一旦选择了一个领导者，另一个为O（n）消息需要履行的资源发现终止。对于弱连接，同步网络的性质，它是可能强制增加每个定向边缘向相反方向的边缘，因此，发送连接的图|E0|的消息。因此，一个 O(n+|E0|) 消息复杂算法实现基于[1]工作。在稀疏网络，|E0| = O(n)，这是渐近消息最优，因此，资源发现算法的挑战是弱连接和非网络的基础上，算法可以实现稀疏（其中的|E0| =Ω (n log n) ）。

## 1.1 Previous Results 以前的成果

对于同步模式，Harchol-Balter，等[2]提出了一种随机算法，概率高，达到O(n log² n) 消息复杂度为O(n² log³ n)的位复杂度，和O O(log²n)次复杂度。Law and Siu [5]提出了一种随机算法，算法[2]实现概率高，为O(n log n)消息复杂度为O(n² log² n)位复杂度的元素相结合，和O的时间复杂度。（O(log n)的时间复杂度的弱连通图。注意两个随机算法[2]和[5]继电器巨资对网络中的节点数目是众所周知的事实。Kutten，Peleg，Vishkin的[4]提出了一个确定性算法，实现了消息复杂度为O(n log n)，O(|E0| log² n) 位复杂度，和O( log n)的时间复杂度，并不需要事先知道，除了网络的大小[4]表明，当存在网络规模的上限，终止检测是可能的。

所有上述算法建立适用于同步网络和，在异步设置不保持其正确性和复杂性的措施。实际的通信系统往往有上发送消息的变量和不确定性延迟时间，因此带来了解决方案更接近了一步现实模型，Kutten和皮莱格[3]研究了在异步网络资源的发现，并提出了一个确定性算法，实现为O(n log n)消息复杂度、 O(|E0| log² n)位复杂度。

## 1.2 Problem Definition

继Kutten和皮莱格[3]，我们研究了异步模式，网络通信是异步的和可靠的。发送的消息将最终抵达后，有没有全局的初始化时间，但节点开始是异步的，唤醒附近邻居。因此唤醒时间复杂度为Ω（n），我们假定从节点u到节点v发送的所有消息，保持一个FIFO（先进先出）顺序到达v。

定义的异步资源发现问题时，有一个微妙的问题：有关终止的检测。一般情况下，我们可以不要求算法检测，它已经达到其目标，并可能终止。这即使在早期的分布式领导人选举的问题上属真确无误“知识图”。一个分布式的领导者选举算法开始时，所有节点都是领导者状态和必须终止领导者状态，只留下一个领导者，其它节点都是非领导者状态。在异步弱连接网络，领导者选举算法检测不到是否满足终止条件。假设对网络G，领导人的选举算法终止执行，然后组合两个G和一个单一的节点u。从u分别增加直边到G的两个副本，立即唤醒节点u以外的所有节点。G的每个副本，将选出的领导者和终止。这将导致两个领导人的终止。因此，我们必须定义----不需要参与检测明确终止的、异步资源发现问题。

异步资源发现：下列安全要求的，应在执行过程中的任何阶段举行：

1，每个节点要么是一个领导者，或属于一个单一的领导。  
2，领导者节点知道属于它的所有节点的ID。  
3，所有非领导节点知道他们的领导人的ID。

此外，我们有以下的活性要求：

4，当所有节点都醒了，在一个状态永远不会发送任何消息，所有消息队列是空的，只有一个领导者仍然存在，在每一个弱连接组成环。

异步资源发现算法的复杂性措施的使用直到达到稳定状态位和消息总数。

我们进一步完善异步资源发现，界定和研究三个变化：

Oblivious Resource Discovery不经意资源发现：节点不知道他们的弱连接组件环的大小。

Bounded Resource Discovery范围内的资源发现：每个节点知道其弱连接的组件中的节点数目。

Ad-hoc资源发现：这种变化是放宽了原问题的一个Ad-hoc资源发现算法必须有属性（1），（2），异步资源发现（4），下列关联的属性（3）：

3a，每个非领导节点有一个确定的指针;  
3b，这些指针从任何非领导节点到其领导人的直连路径。  
请注意，Ad-hoc资源发现，捕捉自然的情况下，在任何给定时间只有在一些节点可能需要知道的所有其他节点的ID。每当一个节点希望得到所有的IDS，它可以联系及其领导人，以获得信息。

## 1.3 Our Results

我们提出异步资源发现的上限和下限及其变化。

Oblivious Resource Discovery不经意资源发现：我们提出Ω(n log n)消息复杂度的下界。这一结果表明，资源发现有向图，是不同的复杂类，同比于无向图，为O(n)的上限（参见[1]）。  
 我们还表明确定性为O(n log n)消息复杂性，为O(|E0| log n + n log² n)位复杂度的算法，提高了位的复杂性[3]。

Bounded Resource Discovery范围内的资源发现：我们提出了一个确定性线性为O(nα(n, n))消息复杂度算法、位复杂度O(|E0| log n+n log² n)。此外，我们的算法知道何时终止，从而解决问题[2]提出的。

Ad-hoc资源发现：我们允许节点来间接了解他们的领导人的ID，同时还要求领导直接了解所有ID。这导致的问题，其消息复杂度是Θ(nα(n, n))。请注意，这个结果规避古典较低选择一个领导者的约束Θ(n log n)的消息。我们表明，特设资源发现经典联盟查找问题紧密相关。我们证明这Ω (nα(n, n))消息复杂度低于使用还原技术的模型。

我们的特别资源发现的算法O(nα(n, n))消息复杂度（渐近最优）和O(|E0| log n + n log² n)位复杂度，理想的情况下，我们希望任何非领导者的领导节点之间的路径的长度以O（1）为界，我们的算法实现摊销时限：对任何m请求到达领导，领导人的选举和所有的请求的答复消息的总成本是O((m + n) α(m, n)) 。

另一个悬而未决的问题[2]提出的要求，如何有效地执行动态网络资源发现。明确，可以充分纳入一个新的节点的复杂性降低到比再次运行整个算法的复杂性？对于特设资源发现，我们表明，该算法有效地处理动态增加节点和链路。

# 2 Message Complexity Lower Bound for Oblivious Resource Discovery消息复杂度下界不经意资源发现

在本节中，我们证明，任何资源发现算法，异步系统有n个节点，其中n是未知的参与节点，必须发送至少Ω (n log n) 消息2.。该证明的想法，显示某个网络上的拓扑结构，一个对手控制消息到达的时间，可以强制任何算法花费造成临时领导人，他们给他们知道所有的节点发送消息，然后透露给他们，（在他们的弱连接的组件环中有）更多的节点信息。

**定理1：**为解决不经意的资源发现任何分布式算法存在在执行至少发送0.5n log n – 2 个消息。

证明：考虑一个完整的根二叉树T(i) 有 n = 2i – 1个节点，所有的边直接连到叶子，任何网络上的T(i)算法，可以强制发送i2i−1 – 2个消息，对于i = 2, T（2）是3个节点的树，任何算法选择任何一个领导者，必须知道另外两个节点的ID，所以需要发送至少2条消息。

1 Inverse Ackerman function: 逆阿克曼函数

\_(m, n) = min{i \_ 1|A(i, bm/nc) > log n},

where A is Ackerman’s function:

for m = 0: A(0, n) = n + 1,

for m > 0, n = 0: A(m, 0) = A(m − 1, 1),

for m > 0, n > 0: A(m, n) = A(m − 1,A(m, n − 1)).

2We will later prove that if n is known in advance then the problem can be solved in message complexity O(n\_(n, n)).

我们将在以后证明，如果n是事先知道的问题可以解决在消息复杂度为O(n\_(n, n))。

假设定理证明了i，我认为，这样的T（i +1）的对手拖延根发送的所有消息，直到两个子树有没有更多的消息发送。注意，根发送消息前没有子树可以了解其余的树，因此每个子树被迫相信，其子树是整个弱连接的组件。归纳假设每一个算法为每一个子树可以强制发送至少i2i−1 – 2 个消息。

现在考虑两种情况：（1）根最终被选定为领导者，在这种情况下所有其他2i+1 – 2个节点必须收到一条消息，以知道领导者的ID。（2）一个非根节点最终被选为领导者。在这种情况下，根必须发送一个消息，至少有一个其他的消息（带有其他节点id信息）发送给领导者， 2i节点（根和对方的节点）必须接收到领导者节点的ID消息。在这两种情况下在T(i + 1)中至少发送有2(i2i−1 − 2) + 1 + 1 + 2i = (i + 1)2i − 2条消息。

请注意，要求所有节点将学习领导者的ID是对数因子的主要来源的证明。因此这种结合并不持有特设的资源发现。

# 3 Message Complexity Lower Bound for Ad-hoc Resource Discovery消息复杂度下界为特设资源发现

在本节中，我们展示了从经典的Union-Find问题对不相交的问题减少设置Ad-hoc资源发现。减少限定Ad-hoc资源发现的消息复杂Union-Find问题的运行时间。

Lemma 3.1：对于集合n，Ad-hoc资源发现任何h(n)消息复杂度的算法，上了宇宙联盟查找算法，可建有h(2n−1+m) 次、n-1次序列合并和m次查找。

证明：考虑一个集合S = {S1，S2...，SN}， F（i）表示发现Si的代表操作，并，U(i, j) 表示，连接两个含有Si 和 Sj的集的联合操作。我们认为此操作前，两集是不相交的。U是n-1个联合操作的任何序列和m次查找操作。

假设一个特设消息的复杂性h(n)为n个节点的网络资源发现的分布式算法，我们建立了以下的网络G：......对于每一个集Si，添加一个节点si， U中的每一个U(i, j)连接操作，添加一个节点ui,j，和两个有向边：(ui,j → si)，(ui,j→ sj )，在U中对于每个查找操作F(i)，添加节点fi和一个定向的边(fi → si)。

由于异步网络的分布式算法，我们可以控制的节点唤醒方式如下：从第一次运行在U开始，如果操作是U(i, j)，然后唤醒节点ui,j，如果操作是F(i)，然后唤醒节点fi，等待，直到该算法没有更多的消息发送，U中移动到下一个操作和循环操作直到所有的节点被唤醒。

Union-Find算法的序列U需要模拟分布式资源发现算法在网络G 中的唤醒序列描述。模拟 h(2n − 1 + m)发送的消息将花费 h(2n − 1 + m)的时间。

由于Ad-hoc资源发现的要求，每次一个节点的ui,j是唤醒了，该算法将继续执行，直到两个节点si，sj具有相同的领导者。因此，唤醒忠实模拟联合操作。每次一个节点fi被唤醒，计算从si达到领导者（因为领导者必须知道fi的id）。因此，唤醒fi忠实模拟查找操作。

Tarjan工作[9]，结果表明，任何算法的一个指针的机器，有分离的属性，一个具有n - 1个序列的合并、混杂n次查找将需要Ω(nα(n, n))的时间。请注意，我们的模拟指针机，具有分离属性（一个不相交的组件节点永远不会有另一个不相交的组件节点的指针），因此，我们已经证明是：

定理2：对于任何解决Ad-hoc资源发现存在一个执行的分布式算法，该算法至少发送Ω(nα(n, n)) 条消息。

# 4 The Generic Algorithm遗传算法

在这一节中，我们提出为不经意模型设计，在网络的大小是未知的，可能不是被强连接的同步网络的通用资源发现算法，后来我们展示了两个变化：（1）有界模型，当网络的规模已知；（2）Ad-hoc资源发现。

每个节点被唤醒并开始执行。最初节点开始作为领导者，并试图征服其他领导人节点，当一个节点被征服了，它变成inactive，只有作为一个消息路由器。active的时候，领导者节点v，旨在扩大其领域通过以下步骤：

1。找到一个未开发的节点u。  
2。达到目前节点u的领导者l， 。  
3。合并l到v  
4。告知他们的新领导人给L的节点。

每个节点开始一个与所有的状态转换图如图1所示，在explore’状态，并在执行过程中可能会改变其状态为‘wait’和‘conqueror’, ‘passive’被动, ‘conquered’ 或 ‘inactive’。

我们将调用节点的领导者，如果其状态不是“‘conquered’或‘inactive’或‘passive’。因此，一个领导节点的状态是‘explore’或‘wait’或‘conqueror’。每个节点维护五套的ID集：local，done，more，unaware，和unexplored，一个FIFO队列previous，两个ID指针：ID，next，和一个整数：phase。ID字段保存节点的唯一ID，最初本地拥有的一套节点最初知道的ID集合。集最初包含元素{id}，next=id，phase =1， done，unaware，unexplored集合，和队列previous是空的。这些数据结构和它们的初始值，如图2所示。

## 4.1 Finding an unexplored node找到一个未开发的节点

一个领导节点v首先需要找到1个并没有属于其群集、未开发的节点。在此操作的伪代码显示在图3。这是节点v的集群节点集已经分成两套：more和done。Done集合的节点已经发送到V的他们知道的所有的邻居，在more集合的节点还没有发送（还）其知道的所有节点给他们的领导人。

不是群集的一部分的节点存储在unexplored集合。为了征服其他节点，领导者v必须在其集群内搜索未开发的节点。如果v.unexplored非空，那么只要从v.unexplored中选择任意id并继续4.2节。

否则，领导人v必须设法询问集群中的节点，他们是否知道任何未开发的节点。如果v.more是不为空，那么v从v.more中任选一个id w，向w一个 '查询'发送消息，通知它从w.local集来删除min{|v.more|+|v.done|+1, |w.local|} id集，回复给他们“查询答复”消息。任何收到由V而来的“查询答复”消息的新的id（不在v.more或v.done）放入到v.unexplored。注意把v本身可能会出现在v.more，在这种情况下v模拟内部发送的消息。非活动节点接到“查询”消息的伪代码出现在图5。

如果|v.more| + |v.done| + 1 >= |w.local|，v现在知道W的所有信息。在这种情况下，w在它的“查询答复”消息中宣布和v然后移动w从v.more到v.done。

既然v本身之包含集群中的|v.more|+|v.done|的节点信息，接着收到min{|v.more|+|v.done|+1, |w.local|}的节点信息，那么v.unexplored非空，或者w被移动到v.done（或者两者都）。该算法的低位的复杂性由于这种平衡，领袖节点接收尽可能多的ID信息需要进步。接收所有w的ID集合会导致更高位的复杂度O(|E0| log² n)。

如果v.unexplored和v.more都是空的，领导者v等待直到v.more变成非空（这可能发生在从一个前所未知的节点发出的一个“搜索”消息到达，在下一小节解释）。

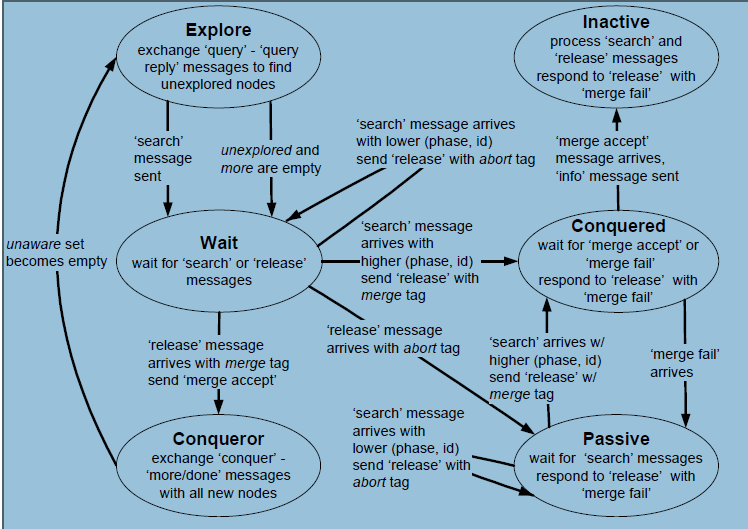


Figure 1: Node’s state transition diagram.

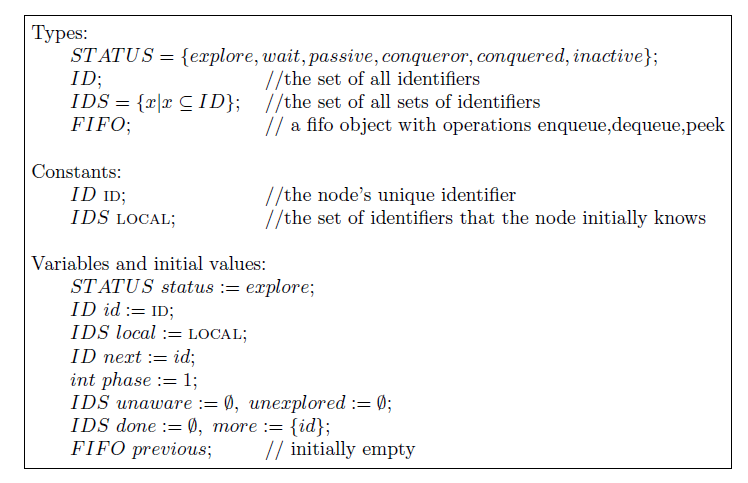


Figure 2: The data structure of each node.

## 4.2 Reaching the current leader of another node到达另一个节点的现任领导人

一旦领导者节点v发现一个未知节点u，它寻找u的领导者，节点u的现任领导人是通过发送“搜索”的消息，每个非活动节点的next指针，直到到达领导人。出现在图5中的非活动节点接收“搜索”的消息的伪代码。“搜索”的消息由<v.id, v.phase, u.id, false>组成。当“搜索”消息M从一个节点y到达一个非活动的节点x，然后键值对（M，y）的入队，形式是x.previous.enqueue(M, y)。仅当M是队列中唯一的消息时，节点x转发消息M 给x.next，即|x.previous| = 1。

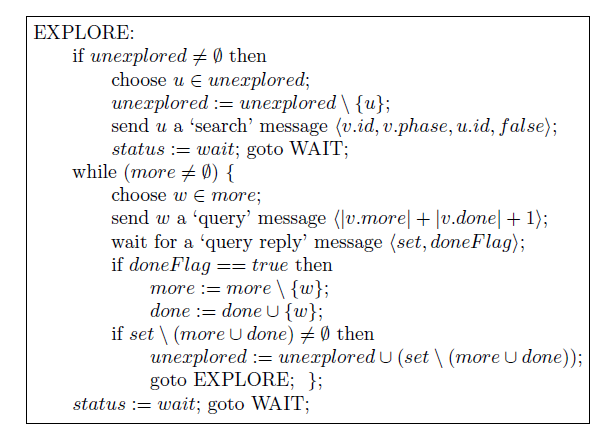


Figure 3: Finding an unexplored node for leader node v.

一旦搜索消息被发送，领导者进入一个等待循环，等待消息一个“搜索”或“释放”消息，在这个阶段出现的伪代码在图4。

一个等待循环中的“搜索”消息到达时u的现任领导人l，最终含l的路径压缩的‘release’的消息会被发送，此消息被按路径上相反的方向、使用存储在previous的指针队列发送。一个非活动的节点收到‘release’消息的伪代码显示在图5。当一个‘release’的消息到达一个非激活节点x，节点x设置(M, y) := x.previous.dequeue(), (M0, y0) := x.previous.peek()。它转发‘release’消息到y，x.next := l，如果M0存在，则转发给x.next，以释放“搜索”消息M0。

注意，它可能是，虽然v知道u，节点u可能不知道v，在这种情况下需要更新其local集合进行相应的设置。此外，它可能是U的领导者l已经把u放置在其l.done集，在这样一个典型案例l需要将u移至l.more。这个过程允许节点l或将征服节点l的未来的领导人，有可能达到和征服节点v，这要求在我们的算法中的“搜索”消息的最后一个字段的增加一个指示标志，值为true表示u不知道v。

## 4.3 Merging of two leaders两个领导人的合并

每当一个领导节点v发出的“搜索”的消息M，达到了等待领导人l，l需要来决定是否发送一个请求合并到v或放弃合并。这一决定是根据领导人是否有一个字典更高的（phase,id）对。

• If v.phase > l.phase or if v.phase == l.phase and v.id > l.id then node l will request to merge.

Specifically, node l will send a ‘release - merge’ message towards v and change its state to ‘conquered’.被征服

被征服的状态的伪代码出现在图6。

当v收到带合并值的release消息，可能：

1. 如果它的状态是 ‘conquered’ (see Figure 6) or ‘passive’ (see Figure 4) or ‘inactive’ (see Figure 5)，则发出‘merge fail’消息。

2. 如果状态为‘wait’ (see Figure 4)，则发出‘merge accept’消息。

如果l收到一个从v发出的‘merge fail’消息，l设置自己的状态为‘passive’。在‘passive’状态，节点l不会发出任何新的‘search’消息，它将被动等待直到一个‘search’消息发现它。

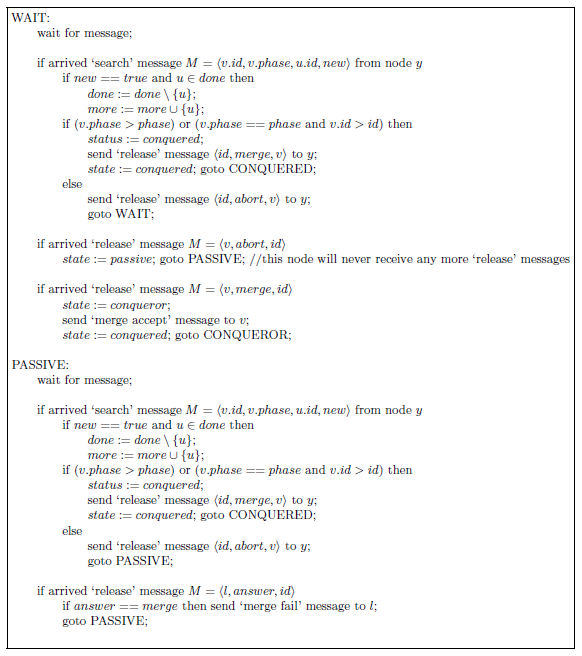


Figure 4: Pseudo code for wait/passive node l .

否则，如果l接收从V 发出的‘merge accept’的消息然后，l.next设置到v，l发送一个包含所有l

l已收集的信息“info”消息M到v：

M = <l.phase, l.more, l.done, l.unaware, l.unexplored> .

最终，l设置状态为‘inactive’，变成非领导者节点。

否则，如果节点v的phase小，或phase相同但id小，则合并被终止。在此情况下，一个包含M = <l, abort> 的‘release - abort’消息被发送到v，接收到带终止值的‘release’消息的领导者节点停止发送搜索消息，其进入‘passive’状态，循环直到其他领导者节点的搜索消息发现并征服它。

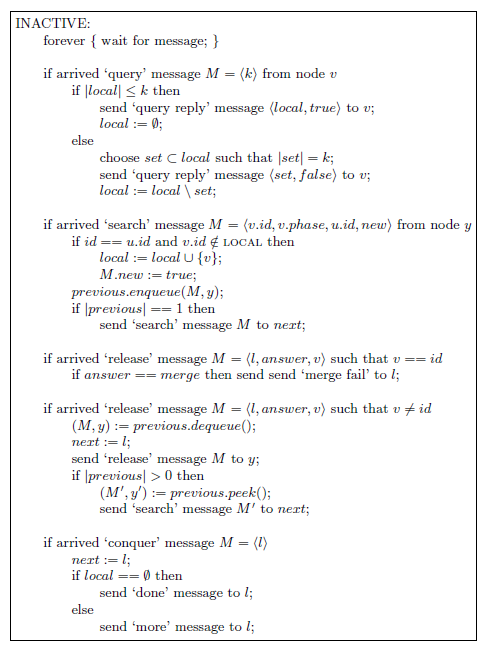


Figure 5: pseudo code for inactive node x.

## 4.4 Conquering unaware nodes征服不知道节点

当一个领导者节点收到一个‘release - merge’消息，它本身处于‘wait’状态，则其进入‘conqueror’征服者状态，回应一个‘merge accept’消息，等待一个‘info’消息。（参见Figure 4 和 Figure 6）

Once an ‘info’ message arrives from a node l, node v sets:

v.unaware := v.unaware ∪ l.more ∪ l.done ∪ l.unaware, and

v.unexplored := v.unexplored ∪ l.unexplored \ {l.more ∪ l.done ∪ l.unaware}.

Leader v checks whether l.phase = v.phase or if |v.more| + |v.done| + |v.unaware| >= 2v.phase+1 in these

cases node v increments its phase (v.phase := v.phase + 1).

集合包含了所有（不知道其领导者已改变了的）新节点。现在v 通知v.unaware中的这些新节点，对每个节点发送一个包含v.id 和 v.phase的‘conquer’消息，告知其v的id。

inactive节点从一个阶段高于其目前的领导人接收‘conquer’的讯息，更新相应next和phase指针，并响应“more/done”消息，用一个字节位响应。

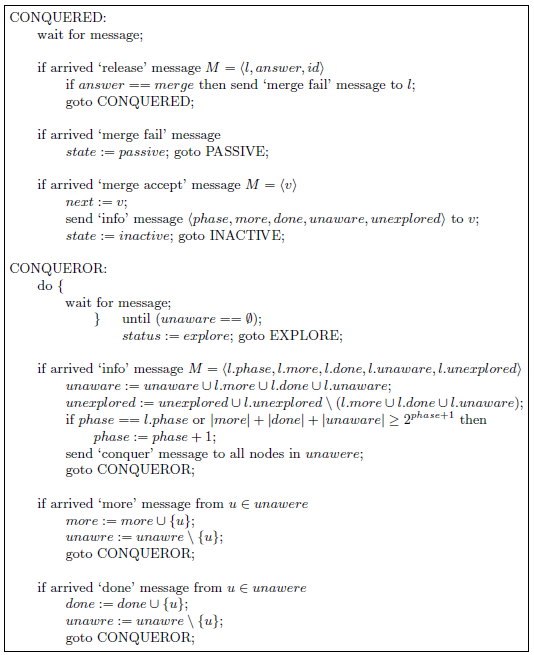


Figure 6: pseudo code for conquered/conqueror node v.

如果节点的local集为空，这是因为我们用一个征服者节点合并了集合，所以失去了这部分信息。（参见图5 接收“征服者”消息的伪代码）。

当v收到一个‘more/done’消息，它根据指示位，移动节点的v.unaware集合到v.done 或者 v.more。当v.unaware变为空，v重置其状态为‘explore’，重新开始循环“发现一个新位置节点”算法(see Figure 3).

## 4.5 Variations on the Generic Algorithm遗传算法的变量

所有的变量值根据该算法的征服阶段的行为而变化。变量不维护不知道数据结构。当从v发出的‘merge accept’消息到达状态为‘conquered’被征服的节点l，节点l 回复一个包含(l.phase, l.more, l.done, l.unexplored)的“info”消息，l.next指向v。v收到l的“info”消息，据此合并各个集合：

v.done := v.done ∪ l.done,

v.more := v.more ∪ l.more, and

v.unexplored := v.unexplored ∪ l.unexplored \ {l.done∪ l.more}).

### 4.5.1 Bounded Resource Discovery范围内的资源发现

有界模型的每个节点知道组件的大小，属于以让n表示组件的大小，当领导节点到达 |done| = n时候，它发送“征服”的消息到所有的节点，完成和终止。

### 4.5.2 Ad-hoc Resource Discovery

Ad-hoc Resource Discovery，领导人从来没有发送“征服”的消息。相反，当一个节点想知道当前组件的ID集合快照，它发送一条消息给领导者（类似“搜索”消息）和在答复时执行路径压缩（类似的“释放”消息）。

Section 6 discuss how extend the Ad-hoc algorithm to handle online link and nodes addition.

Section 6讨论如何扩展Ad-hoc算法来处理网上链接和节点增加。

# 5 Analysis of the algorithms算法分析

## 5.1 Liveness and Termination活性和终止

在本节中，我们表明，所有的节点时，有没有更多的消息发送，当消息队列是空的，只剩一个领导者，将继续保留到每个弱连接组件环中的所有已唤醒节点的链接，异步资源发现的要求得到满足。

Lemma 5.1：在执行的任何阶段，至少有一个节点，在每一个弱连接的组件环中仍然是一个领导者的状态。

证明：一个节点不是一个领导者的状态，它将是 ‘conquered’ “被征服”或“被动” ‘passive’ 或“无效”‘inactive’。这是发生在下列情况下：（1）当它接收到一个‘release - abort’释放-中止的消息，成为‘passive’，（2）当它发出了一个‘release - merge’释放-合并的消息，并收到一个‘merge fail’消息，先变为‘conquered’，后来变成‘passive’，或（3）它发出了一个“‘release - merge’的消息，并收到一个‘merge accept’消息，状态先变为 ‘conquered’，后来变为‘inactive’。在任何时刻，字典最高的（phase, id）对的节点将总停留在领导者状态，因为它既不接受的中止，也不能被征服。

另一个值得关注的是，一些节点自己的队列中存在消息时，该算法将成为陷入僵局。

Lemma 5.2：如果该算法是陷入僵局，那么任何消息队列中都没有消息。

证明：如果该算法僵持不下，那么没有消息会被发送，假设某些节点有非空的消息队列。

让我们看看在next指针引起的图，该图是一个树形，因为更新的next指针总是朝着一个更高的阶段节点完成。

请注意，没有陷入僵局节点的状态可以为 ‘explore’ 或 ‘conqueror’。这是真实的，因为处于这些状态的节点，等待来自非活动节点的立即回复。

此外，没有陷入僵局的节点可以在 ‘conquered’状态，处于这个状态的节点等待 ‘merge accept’或‘merge fail’消息，而收到‘release - merge’消息的节点会发出这些消息。注意处于‘conquered’状态的节点所发出‘release - merge’消息的永不会被中间的‘inactive’节点所延迟。保他们的返回路径是由指针和previous队列保持，消息最终将到达适当的‘search’消息的始作俑者，并立即响应做出回答。

现在，考虑在森林里的树节点之一，其中有一个非空的消息队列。设v是消息队列非空的、最接近根的节点。首先假设v是根。在这种情况下，它的状态不能是‘inactive’，因为这些节点的next指针指向另外的节点，于是其状态将是‘wait’ 或 ‘passive’。但节点在任一状态都不可能其消息队列非空且陷入僵局。

现在，假设v是‘inactive’的中间节点。如果它的队列非空，它在等待其发给根节点的搜索消息的回复。中间节点没有举行任何队列，从而搜索消息应该导致从根节点发回一个release消息的答复的一对矛盾。

Lemma 5.3：如果所有节点都醒了，然后只要有比单一的领导节点在每个弱连接组成，至少有一个领导者的计算是免费的继续执行。

证明：假设系统死锁，我们有更多的引理5.2证明单领导人在一些弱连接组件节点，领导节点可以只在“等待”状态，在这种状态下僵持有没有剩下挂起的搜索消息。

既然有超过一个的单一的领导节点，所有节点都是清醒的，然后有至少一个领导节点的unexplored的集合不为空。这个节点，应该发起一个搜索消息。一对矛盾。

Lemma 5.4:如果所有节点都醒了，有一个单一的领导者，那么其最终知悉其弱连接的组件中的所有节点的ID。

证明：假设单领导人节点，l，死锁，而不知道在其弱连接的组件中的某些节点的ID，这种死锁只有当所有的more和unexplored集是空时才能发生，这意味着，所有的节点可达l，在l的l.done集。

这样，对于l不知道的节点们，l不能到达。从弱连接组件删除l.done、创建一个或多个残留的弱连接组件环。所有边缘在这些组件环之间，l.done直接连着l.done。

考虑这样一个弱连接的剩余部分G0， G0不包含一个领导者的节点。引理5.1证明在G0的字典最高的节点，必定已收到来自在l.done节点之一发出的‘release’消息，这样的消息造成其经过的G0和l.done之间的边是双向的，矛盾。

因此，弱连接在每个组件环有一个单一的领导。

结合引理，我们已经证明了以下内容：

定理3：遗传算法（代表为Ad-hoc算法）符合不经意资源发现（代表为Ad-hoc资源发现）的要求。

定理4：有限的资源发现算法从来没有进入死锁状态和终止时，符合资源发现的要求。

证明：在有限的模型，弱连接的组件环的节点数量是事先已知的，一位领导人发送‘conquer’消息到所有节点的节点，都知道这是唯一的领导者，并终止。

## 5.2 Message complexity消息的复杂性

我们受消息的复杂性，包围每个消息类型，然后总结。

引理5.5：‘query’和‘query reply’消息的数量最多是4n。

证明：我们从‘query’和‘query reply’消息类型开始。

每次这样一对发送，则：（1）查询的节点移动到done，或（2）领导节点收到足够的ID，以确保其unexplored集是非空的，在这种情况下，领导最终将要么（2a）战胜另一位领导人，或从“等待”（2b）状态从‘wait’移动到非领导者状态。

案例（1）可能发生的最2n次。注意，节点可能不止一次从more到done。第一个n统计首次移动的节点的数目，第二个n是由于这样的事实，每一次节点的移动，从done到more，领导发起的‘search’消息引起的变化变为inactive，这种情况可能发生最多n次。

例（2a）和（2b），每一个可能发生最多的n次，因为一个节点可能征服其他至多n个节点和n个节点可能移动从‘wait’到非领导状态一次。

引理5.6：‘search’和‘release’消息数为O(nα(n, n))。

证明：每一个‘search’ - ‘release’的计算，模拟一个发现操作，这个计算，要么导致一个领导节点与另一节点的合并，或导致领导节点收到一个‘release - abort’的消息，并再也没有回到一个领导者状态。因此，至多为2n次发现操作和n次合并业务发生。我们的算法模拟 Tarjan的经典联合/查找相交集算法，对不相交的集合做顺序执行。通过分析Tarjan和范•鲁凡[10]，总共发送了O（Nα（N，N））的‘search’ - ‘release’消息。

引理5.7：‘merge accept’, ‘merge fail’, 和 ‘info’的消息总数最多是2n。

证明：......继‘release - merge’发件人将永远不会返回到领导者的状态，这种情况至多会出现n次，每一个这样的‘release - merge’消息的序列导致‘release - merge’、 ‘merge fail’的消息或序列‘release - merge’、 ‘merge accept’、‘info’的消息。

引理5.8：‘conquer’, ‘more/done’的消息数在遗传算法中至多为2n log n，在有限的模型中最多为2n。

证明：..在遗传算法中，每次inactive节点发送‘conquer’消息，其phase增加，由于Union-Find类比，任何一个领导者的最高phase是log n，因此，分析最多有2n log n个类型为‘conquer’, ‘more/done’的消息。

在界模型，这些消息只在最后阶段被发送，从而发送2n个这样的消息。

上述引理意味着下面的定理：

定理5：遗传算法的消息复杂度为O(n log n)。

定理6：有界的消息的复杂性，特别算法为O(nα(n, n))。

## 5.3 Bit complexity位复杂度

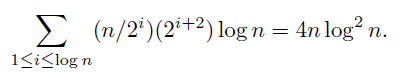
类似上一小节的消息复杂度分析，我们受边界位总数为每个消息类型，然后总结了对所有类型的总数位。

引理5.9：在‘query reply’的消息总位数最多为 2|E0| log n。

证明：在E0的每一个边(u → v)，id v在‘query reply’消息中最多只出现一次。发生这种情况时，它第一次通过查询消息向u发请求。注意在此执行过程中，E0的有向边，可能会导致相反的边被添加到节点的本地组（细节见4.2节）。因此，除了为每个E0的边缘（U→V），id u可能会在‘query reply’消息中出现最多一次。

引理5.10：在‘info’的消息的总位数最多为4n log² n。

Proof: Every leader maintains the inequality |more|+|done|+|unaware| < 2phase+1, and thus |unexplored| <= 2phase+1. So when a node gets conquered, the ‘info’ message it sends contains at most 2i+2 log n bits. Since the number of leader nodes that reach phase i is at most n/2i, summing over all leader nodes by phase results in:



Theorem 7 The bit complexity of the Generic Algorithm is O(|E0| log n + n log² n).

Proof: All other messages are of length O(log n) and the message complexity is O(n log n). Thus the bit

complexity of all other messages is O(n log² n)

# 6 Node and link additions in Ad-hoc Resource Discovery在Ad-hoc资源发现的节点和链路增加

在本节中，我们扩展我Ad-hoc Resource Discovery算法在动态环境中工作

在动态模型中，我们允许被添加到系统中的节点和链路的时候，资源发现算法仍在运行。本质，在某个时间节点加入系统，同时有节点唤醒，这没有区别。因此，没有特殊的操作用于节点到达。

当添加一个链接，它可能需要的节点通知其领导人的这种变化，具体来说，添加一个新的边缘(u → v)在时间t，有两种情况考虑：

1，u尚未在其响应查询消息中报告所有的边缘。

2, u已经报告其所有的边（u.local为空）

在第一种情况中，可以考虑边缘作为尚未报告的边缘，直到时间t，并没有别的需要做。在第二种情况下，u向其领导人启动一个新标志设置为true的搜索消息。该消息将导致u的领导人移动u从done到more。这将导致领导人知道的U有更多的链接，并在未来查询它。

定理8：对于动态增加ˆn新节点和ˆe新边到任意网络|V | = n，初始状态的总消息复杂度是O(mα(m, n + ˆn))，其中m = n + ˆn + ˆe。

证明：增加ˆn个节点相当于同时启动n + ˆn个节点；增加ˆe个链接导致最多ˆe次发现操作。于是总的发现操作数是n + ˆn + ˆe。

因此，有没有必要重新运行算法在每一次新的组件环增加进来。由于网络的动态性，更多的消息和额外的节点和边的数量几乎是线性的。

# 7 Conclusion结论

..为异步资源发现提供了一个新的消息复杂度较低的约束。我们提出一种算法，改进了[3]的位的复杂度，我们探索[2]中公开提出的两个问题：如何处理网络不是静态的，并能终止检测。我们为Ad-hoc资源发现，提供了一个有效地管理动态节点和链路增加的算法，部分回答了第一个问题，我们证明了不经意的资源发现和Ad-hoc资源发现的消息的复杂度下界，从而显示出我们的算法是渐近最优的。对第二个问题。我们表明，在已知网络规模时，终止检测是可能的。此外，我们提出的第一个连接我们已知的经典union/find数据结构[10，9]和分布式算法。

Kutten and Peleg [3]描述了唤醒模式，在一些全局的广播机制需要ΔT的时间唤醒所有节点，在这样的模式在同步设置运行时，其算法的时间复杂度为O(ΔT +log n)。注意，在这样一个模型，我们的算法的时间复杂度为O(ΔT +n)，一个悬而未决的问题是提高位的复杂性，同时保持对数时间复杂。

另一个有趣的其余悬而未决的问题是如何有效地处理动态节点清除。这是关系到提高资源发现的鲁棒性的主题

# 8 Acknowledgements致谢

笔者想感谢Shay Kutten有益的讨论和发言，并指出，为O（n）消息复杂度的领导者选举算法[1]可以应用于强连通的网络。

# References

[1] I. Cidon, I. S. Gopal, and S. Kutten. New models and algorithms for future networks. In Symposium on Principles of Distributed Computing, pages 79–89, 1988.

[2] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource discovery in distributed networks. In Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, pages 229–237. ACM Press, 1999.

[3] S. Kutten and D. Peleg. Asynchronous resource discovery in peer to peer networks. In Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS’02), pages 224–, 2002.

[4] S. Kutten, D. Peleg, and U. Vishkin. Deterministic resource discovery in distributed networks. In

Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures, pages

77–83. ACM Press, 2001.

[5] C. Law and K. Siu. An o(log n) randomized resource discovery algorithm. In Brief Announcements of the 14th International Symposium on Distributed Computing, Technical Report, Technical University of Madrid, pages 5–8.

[6] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC ’02), pages 183–192, 2002.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In Proceedings of the ACM SIGCOMM 2001 Technical Conference, 2001.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the SIGCOMM 2001, 2001.

16

[9] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. Journal of

Computer and System Sciences, 18(2):110–127, 1979.

[10] R. E. Tarjan and J. van Leeuwen. Worst-case analysis of set union algorithms. J. ACM, 31(2):245–281, 1984.

[11] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications, 2003.