应用层组播

背景技术

（IP组播与应用层组播）

1. 选举簇首

文献[2]选举簇首及辅助簇首的方法，仅考虑了簇内节点间的距离，或者是仅考虑了节点的出度及传输时延，而没有考虑节点的移动性，应用层组播节点的移动性大，如果没有考虑组播节点的移动性，推举出的簇首可能是非常不稳定的节点。文献[3]推举簇首的方法则仅考虑了节点的移动性，没有考虑节点的出度和传输时延，这种情况下选举出的簇首的转发能力和传输效率可能会满足不了需求。本文选择簇首的方法，充分考虑了节点的移动性、数据转发能力和传输时延等因素，在节点移动的情况下，保证数据传输的可靠性和高效性。

* 1. 节点稳定度

首先说明两个参数：链路过期时间（LET）和节点稳定度。

参考文献[1]，根据链路中两个节点的位置、移动方向和移动速度，可以预测该链路的过期时间LET，其定义为：



其中：



（,）、（,）分别为节点的二维空间坐标; 、分别为节点的移动速度值; 、分别为节点的移动方向。

根据LET，定义节点的稳定度为该节点与其所有相邻结点组成的链路的LET之和，即：



其中为节点的相邻节点集合。

* 1. 节点优先值

节点的优先值，考虑的因素有节点的稳定度、节点的出度、节点的最大出度和节点的传输时延。其定义如下：



其中V为拓扑图的节点集合，是[0,1]之间的平衡系数，用于协调稳定度、节点度数和节点传输时延的权重。、分别为节点及其邻节点的出度之和、拓扑中所有节点的出度之和，为以节点为源结点，其余结点为目的结点的传输路径的传输时延之和，。即：



若两个节点间有多条传输路径，取传输路径时延的平均值。

节点的稳定度越高，节点退出的概率就越小；节点及其相邻节点的度越大，则通过该节点接收到数据的节点越多，数据传播得越快，节点的转发能力就越好。节点的传输时延越小，则组播树的传播效率也越高，工作性能越好。因此，节点稳定性越高、节点及其邻结点的度越大、传输时延越低，节点的稳定性和数据转发效率就越高。可以看出，节点的优先值越低，节点的性能就越好。

* 1. 选举簇首的过程

1. 节点维护一个信息表info-list，存储节点的ip地址、稳定值、出度和传输时延。
2. 各节点向相邻节点发送探测报文，探测报文中包含信息表info-list的信息。随机选择某一节点作为暂定簇首，暂定簇首创建成员信息表member-list。当节点收到探测报文时，如果该节点不是暂定簇首，则转发该探测报文给其相邻节点；如果该节点是暂定簇首，并且暂定簇首的member-list中没有包含该探测报文的信息，将该信息添加至member-list中。
3. 经过一段时间，暂定簇首的成员信息表member-list中包含了所有节点的信息，并计算出所有节点的优先值。将优先值最小的结点命为簇首，次小值命为备用簇首。并向簇内所有成员发送选举出的簇首和备用簇首的信息。成员接收信息后，将簇首和备用簇首的信息记录在info-list中。同时，还需要处理member-list：
   1. 若暂定簇首不是选举出的簇首和备用簇首，则将member-list发送给簇首和备用簇首。成员信息表被接收并保存后，暂定簇首将member-list删除。
   2. 若暂定簇首是选举出的簇首，则暂定簇首只需记录备用簇首信息，并将member-list发送给备用簇首。
   3. 若暂定簇首是选举出的备用簇首，则暂定簇首只需记录簇首的信息，并将member-list发送给簇首。
4. 簇内传输路径构建

簇内信息的传播采用广播的方式。以节点间的距离作为权值，并基于Dijkstra算法，构建簇内信息传输路径。通过簇内信息传输路径：（1）簇首可以将信息广播至该簇的所有成员；（2）簇内成员可以将信息发送给簇首，然后簇首再将信息广播至簇内的其它成员，或者是通过组播树，将信息发送至其它簇的簇首，其它簇的簇首再将信息广播至其成员，从而实现节点之间信息的互通。

若簇中的某个节点i与相邻簇的某个节点j之间存在链路，此时有两种情况：（1）节点j与其簇首之间可达。则簇首不会通过簇内的信息传输路径将信息发送至节点j。节点j接收其相邻簇节点的信息的方式为：首先，所有簇的簇首需要构建组播树，通过组播树，各个簇可以互通信息。节点j的簇首接收到了相邻簇的簇首发送的信息后，再通过其簇内信息传输路径，将信息发送至节点j。

（2）节点j与其簇首之间不可达。因为节点j与其簇首之间不可达，因此节点j不能通过其簇首获取其它簇的节点的信息。为了使节点j能获取其它簇节点的信息，节点i在接收到簇首发送的信息时，也应将此信息转发给节点j。若节点j与其所属簇内的部分节点之间是可达的，显示这部分节点与簇首也是不可达的，则节点j在收到由节点i转发的信息时，应该将信息也转发给这些节点。

如图2.1，被圈出的部分即是簇3中与簇首不可达的节点。簇3的节点4与簇1的节点2之间存在链路，则节点2（簇1）在收到其簇首发送的信息时，应转发该信息给节点4（簇3），然后节点4再将信息转发给同一簇的节点2和节点3。

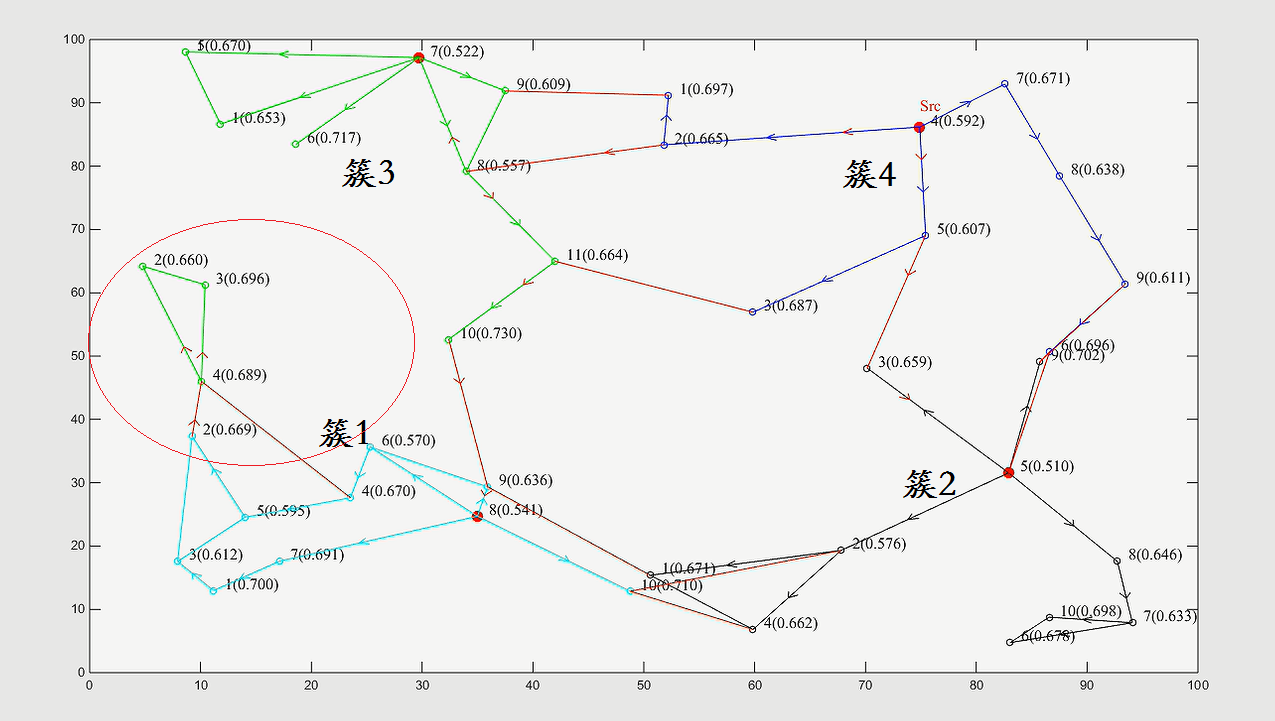


图2.1 与簇首脱离的节点

1. 组播树构建

所有推举出的簇首构建组播树，簇间成员的通信依靠簇首来完成。随机选择某一簇首为组播源，将信息组播至各个簇首。

src

图3.1 组播树—虚链路

如图3.1所示，簇首之间通过构建虚链路，形成组播树。所谓虚链路的含义是，假定簇首之间存在某一条路径，使得簇首可以互通。

考虑实际的情况，为减少组播树中的冗余路径，组播树中可能会存在一些分叉点。

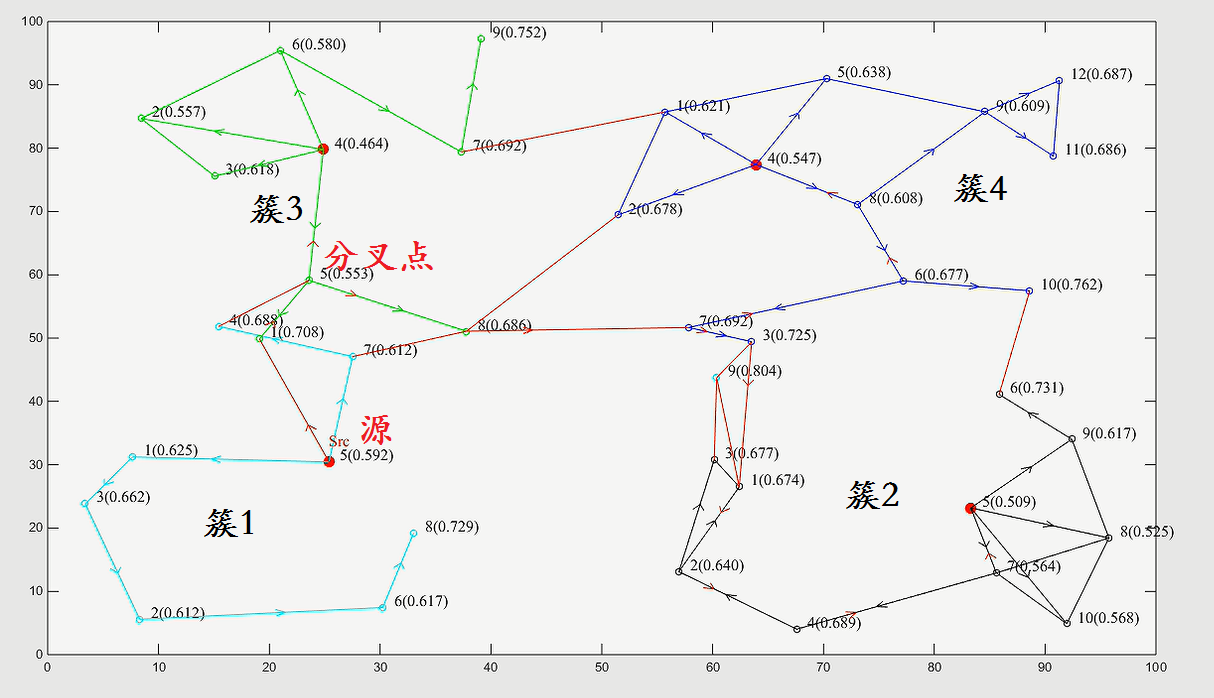


图 3.2 组播树—分叉点

如图3.2所示，若节点根据其所在的位置，分成图中所示的四个簇，并选定簇1的节点5作为簇首，则簇1至簇3的组播路径为5(簇1)-->1(簇3)-->5(簇3) -->4(簇3)，而簇3至簇4的组播路径，若还是以簇3的簇首（即节点4）为起点，则其组播路径为4(簇3)-->5(簇3)-->8(簇3)-->2(簇4)-->4(簇4)。可以看出，这条路径与簇1至簇3的组播路径有重合，即产生了冗余路径4(簇3)-->5（簇3）。为了减少组播树的冗余路径，簇1至簇4的合理组播路径应为图中所示的组播路径，即：

5(簇1)-->1(簇3)-->5(簇3) -->4(簇3)

8(簇3)-->2(簇4)-->4(簇4)

其中簇3的节点5就是一个分叉点。

3.1 建立组播树过程

若某条链路的两个节点属于不同的簇，则这两个节点称为中间结点(mediator)，这条链路称为中间链路(meLink)。随机选择某个簇首作为组播源，记为src。则建立组播树的过程可以分解为：

1. src-->mediator\_A-->mediator\_B。
2. mediator\_B-->head
3. head-->mediator\_C or mediator\_B->mediator\_C
4. 重复步骤2和步骤3，直至组播树中包含所有簇首

3.1.1 组播源与中间结点

建立组播树的第一步是在组播源src和其所在簇的mediator之间建立路径。在前文中，已经建立了组播源至簇内其它节点的信息传输路径。如果将某个mediator选为组播树的一部分，则组播源至该mediator的信息传输路径也就作为组播树的一部分。若有多个mediator与不同的簇的相通，则组播源与所有的这些mediator之间的簇内传输路径都应属于组播树的一部分，也就是说组播源应尽可能地将信息扩散至所有与其可达的相邻簇。

一个簇中可能同时存在多个mediator，若组播源所在的簇中的多个mediator的链路的另一端的节点同属于一个簇（记为簇p），此时的准则为：若组播源与某个mediator之间的路径跳数最少，则选择该mediator。若跳数相同，则优先选择与簇p的簇首直接相连的mediator，若不存在这样的mediator，则选择传输时延最小的mediator。由于应用层组播节点的移动性，簇首可能不能总是与其它的结点都保持可连通，此时若所有的mediator都与组播源不可达，则组播树就无法成功地构建。

组播源src与其所属簇的mediator之间的组播树分支构建完成后，若mediator的中间链路仅有一条，则将这条中间链路添加进组播树中。若mediator的中间链路不仅一条，且所有中间链路的另一端节点均属于同一簇（记为簇p）时，此时，应优先选择与簇p簇首直接相连的mediator，若不存在这样的mediator，则选择链路传输时延（即节点间距离）更小的mediator。

3.1.2 中间结点与簇首

组播源src与中间结点mediator的组播树分支构建完成后，需要构建相邻簇（记为簇p）的mediator与其簇首的组播树分支。下一步需要构建的则是簇p与其相邻簇(记为簇q)之间的组播树分支。上文已经描述过，为减少组播树的冗余路径，组播树中可能存在分叉点。因此簇p与簇q之间的组播树分支有两种可能：（1）若簇p的簇首同时也是一个与簇q相连的mediator，则不存在分叉点，即此时的组播树分支为mediator\_B-->head-->mediator\_C。（2）若簇p的簇首不是一个mediator，则此时会存在分叉点，即此时的组播树分支会分为两个部分: mediator\_B-->分叉点-->head及分叉点-->mediator\_C。

首先，所有簇首构建无线mesh网络，根据簇首的数量N判断第二层是否需要分簇及推举簇首：

1. N <= 3k – 1, 则将第一层簇首组成一个簇，不需推举簇首。这种情况下，认为簇内的每个成员都是平等的，因此随机选择某一结点为组播源结点。构建簇内传输路径的方法，与第一层中的过程相同。
2. 若N > 3k – 1，则将第一层簇首分成多个簇，各个簇需推举簇首，以组成传输结构的第三层。各个簇的数量n的大小维持在范围k~3k-1。簇内推举簇首及构建簇内传输路径的方法与第一层的相同。

其中k可以根据应用场景动态调整。如果需要构建第三层及以上的传输结构，则采用以上相同的方式构建，直至顶层节点的数量不大于3K-1。

1. 组播树的动态维护

分簇完成后，节点就固定属于某个簇。

* 1. 移除无效链路
     1. 若簇内节点的链路超过一定的长度，链路失效
     2. 若簇间节点的链路超过一定的长度，链路失效
  2. 簇内节点尝试重连（Reconnect）

1. 若节点脱离，尝试让节点重新连接

策略：从am(i,j) == 0的结点中寻找最近的结点，并与其连接

(1)若当前结点n度超出，break

(2)若最近的结点未超出MaxLinkDistance,但度超出，则尝试与下一个最近的结点相连

(3)若最近结点已超出MaxLinkDistance，该结点无法重连，break

(4)若与最近的结点相连，拓扑通，重连成功，break

(5)若与最近结点相连，拓扑仍不通，则使最近的结点重复以上过程

(6)若所有结点均已重复以上过程，重连过程结束

1. 链路断开，使得节点与簇内大部分节点脱离，应使结点往质心方向移动
2. 簇间传输路径过少，难以使簇互联互通（因某些簇内的节点太过密集）

策略：若簇间结点过于密集（容易导致簇间不能互通），使簇中节点往质心的反方向移动

特殊情况1：簇内的某节点n，依赖节点m来与簇内其它节点保持可通信状态，但节点m的度已经是最大值。

n

p

m

图 1

处理策略：

* + - 1. 尝试断开节点m与簇内其它结点的某条链路，若该链路的断开，不会影响节点m及簇内其余结点的正常通信，则确认断开该链路。
      2. 断开某条簇间链路，若存在多条簇间链路，则选择一条对簇间正常通信影响最小的链路断开。

如图1所示，节点n因与其它节点的距离过大，而只能与节点m尝试建立链路，则节点n依赖与节点m的链路来保持其与簇内其余结点的通信，若节点m的最大度为3,则因为度约束，节点m无法与节点n建立链路，此时节点n应断开与节点p的链路，再与节点n建立链路。与结点p的链路断开，不会影响节点n及簇内其它结点的正常通信。

特殊情况2：节点n需要与其相邻簇的节点m相联，才能保证簇间正常通信，而节点m的度已达最大值

n

p

m

图 2

处理策略：节点m应尝试断开其某一条链路（簇内或簇间链路），若断开该链路后，不会影响节点m、簇内其它节点及簇间的正常通信，则确认断开该链路，并与节点n建立链路。

如图2所示，若结点n须与结点m建立链路，才能保证簇间的正常通信，而节点m已达最大度3，则节点应断开与节点p的链路，与节点n建立链路。与节点p建立链路，并不会影响节点m及其簇内的其余节点的正常通信。

* 1. 节点移动至其它簇

在战术通信网的场景下，认为节点一般不会频繁地离开簇的区域（若节点移动至区域外，应使节点往所属簇的质心方向移动）。但是，若节点移动至其它簇，且节点与所属簇内的大部分节点连接不上，应尝试与节点所在的簇内节点相连。

若连接成功，节点需要增加的信息有：节点连接到的簇的编号，节点连接到的节点的编号。

若节点与其所在簇的节点相连，若相连的节点与其簇内的大部分节点已脱离，则仍尝试连接簇内的其它结点。若相连的节点与其簇内的大部分节点都能连通，则不尝试连接簇内的其它结点。

* 1. 与附近簇相连

为保证簇间的信息传输，节点应与附近簇的结点尝试相连。尝试连接的节点需要能够与其所属簇内的大多数节点（一半以上）连通。若尝试连接成功，则放弃与该相邻簇的其它结点及其它相邻簇的节点相连。

若相连成功，节点需要增加的信息有：节点连接到的簇的编号，节点连接到的节点的编号。

若节点与其附近簇的节点相连，若相连的节点与其簇内的大部分节点已脱离，则仍尝试连接簇内的其它结点。若相连的节点与其簇内的大部分节点都能连通，则不尝试连接簇内的其它结点。

* 1. 节点的离开与加入

节点的离开事件与新节点的加入事件，应以什么样的频率发生？（赋予节点生存概率的概念）

* 1. 簇的分裂与合并

参考文献

[1] W. Su, S.-J. Lee, M. Gerla, “Mobility prediction in wireless networks,” MILCOM 2000, Volume: 1, Pages:491-495, 22-25 Oct. 2000

[2] 杨玲.基于优先度的层次化应用层组播算法研究[D].华中科技大学,2007.

[3] Yao Yu, Qi Zhang, and Sidan Du,“Mobility-aware Multicast over Mobile Ad Hoc Networks,” in IEEE International Colloquium on Computing, Communication, Control, and Management, CCCM 2009.