第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·网络与通信· 文章编号：1000—3428(2010)19—0148—03 文献标识码：A 中图分类号：TP393

无线传感器网络数据融合路由算法的改进

周 琴[1]，戴佳筑[1]，蒋 红[2]
(1. 上海大学计算机工程与科学学院，上海 200072；2. 贵州广播电视大学计算中心，贵阳 550004)

摘 要：无线传感器网络能量有限，数据融合能通过合并冗余数据减少传输数据量，但其本身的代价不可忽略。针对该问题，研究数据融合代价和数据传输代价对数据融合路由的影响，在基于决策数据融合技术AFST中，对直传数据采用动态最短路径(DSPT)算法，动态识别网络环境和数据特征变化，以最小的代价调整路由。实验与分析结果表明，当网络结构发生变化时，DSPT算法比SPT算法效率更高、更节能。

关键词：无线传感器网络；数据融合；动态最短路径树；路由

Improvement of Routing Algorithm for Data Aggregation in Wireless Sensor Network

ZHOU Qin[1], DAI Jia-zhu[1], JIANG Hong[2]
(1. School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China;
2. Computing Center, Guizhou Radio and TV University, Guiyang 550004, China)

【Abstract】 Energy is limited in Wireless Sensor Network(WSN), and data aggregation is used in WSN to save energy by aggregating redundant data to reduce the data size being transmitted. But the aggregation cost itself is non-neglectable. Focusing on minimizing the total energy cost of the WSN, this paper improves the Adaptive Fusion Steiner Tree(AFST) by substituting the Dynamic Shortest Path Tree(DSPT) routing algorithm for the Shortest Path Tree(SPT) routing algorithm to adapt to the changes of network environment and data size. Experimental and analysis results show that using the DSPT routing algorithm for directly relayed data has a better performance in energy saving and efficiency than SPT routing algorithm when data size changes.

【Key words】 WSN; data aggregation; Dynamic Shortest Path Tree(DSPT); routing

1 概述

由于无线传感器网络中节点的能量十分有限，因此在设计各种网络协议时必须考虑节能。采用网内数据处理技术是降低能耗的重要手段，而数据融合与数据路由相结合是实现网内数据处理的重要方法[1-3]。数据融合能减少数据传输量，降低网络负载，但传统的数据融合方法只考虑了传输代价，忽略了融合过程本身的能量代价。在实际网络中，数据融合的代价有时会大于因融合而节省的传输代价，此时从节能角度而言数据融合失去意义。

文献[3]提出一种基于决策数据融合的路由算法AFST (Adaptive Fusion Steiner Tree)，它通过权衡融合代价与传输代价，动态决定数据相遇点是否进行数据融合，可以进一步减少网络的能耗代价，延长系统的生命周期。但该方法不适合网络结构变化快的传感器网络，因为它没有考虑网络结构发生变化的情况，包括每条链路上的融合代价和传输代价是可变的、节点的分布是随机的、数据间相关性的变化导致融合后数据包的大小变化等，此时，当数据不融合而沿原最短路径直传给sink节点的路径就未必是真正的最短路径了，会导致能量浪费。

针对上述问题，本文提出新的动态最短路径算法(Dynamic Shortest Path Tree, DSPT)，只对发生变化而受影响的节点重新计算最短路径，不仅可以适应网络结构的动态变化，而且能将算法最优化。当网络拓扑很大时，可以极大减少计算量，比原来算法中SPT路由算法更能适应网络结构变化，且更节能。

2 预备知识

2.1 网络模型和能量模型

无线传感器网络用图表示，其中，V 表示所有节点的集合；E 表示所有节点间可以通信链路的集合。信息源点集合 $S \subset V$，包含 k 个传感器节点，它们将以反向组播树的形式周期性地向sink $t \in V$ 报告采集到的数据。

从每个节点 v 流出的数据量大小用 $w(v)$ 表示。每条边 $e=(u,v)$，u 为起点，v 为终点，边的权值 $w(e)=w(u)$。传输代价与每个节点流出的数据量有关，融合代价与每个节点流入的数据量有关。每条边上的单位传输代价用 $c(e)$ 表示，单位融合代价用 $q(e)$ 表示，则边 e 上的传输代价为 $t(e)=w(e)c(e)$。如果在节点 v 对 u、v 的数据进行融合，v 点的数据量在融合前后会改变，以 $\tilde{w}(v)$ 和 $w(v)$ 分别表示数据融合前后节点 v 的数据量。$x_{uv} \in 0,1\}$ 表示边 $e=(u,v)$ 上是否进行数据融合(1融合，0不融合)。σ_{uv} 表示节点 u 和 v 之间数据的压缩率，则数据融合函数为 $w(v)=(\tilde{w}(v)+w(u))(1-x_{uv}\sigma_{uv})$，而数

基金项目：上海市重点学科建设基金资助项目(J50103)
作者简介：周 琴(1983－)，女，硕士研究生，主研方向：无线传感器网络；戴佳筑，副教授、博士；蒋 红，副教授
收稿日期：2010-05-26 **E-mail：**zhouqinppq@163.com

据融合的代价为 $f(e)=q(e)\,(\,w(u)+\tilde{w}(v)\,)$。

2.2 基于决策的数据融合路由问题描述

基于决策的数据融合路由问题就是找一个包含所有信息源节点 S 和 sink 的子图 $G^*=(V^*,E^*)$ 使得 G^* ：$\sum_{e\in E^*_f}(f(e)+t(e))+\sum_{e\in E^*_n}t(e)$ 值最小，其中，边集合 E^* 由 2 个不相交子集 E^*_f 和 E^*_n 构成，分别表示所有进行数据融合的边的集合和所有不进行数据融合的边的集合。

本文针对 G^* 中的 $\sum_{e\in E^*_n}t(e)$ ，为“直传数据”找到动态的最短路径，以最小的代价适应数据量的实时变化。

3 算法描述

3.1 AFST 算法

AFST 算法[3]的基本思想是在每轮循环中对剩余的融合点进行“配对-决策-融合”过程。先寻找信息源节点的最佳融合配对，对每对待融合节点对判断其融合是否有得益，有益则融合，否则不融合也不配对，然后将数据直接以最短路径算法 SPT 传给 sink 节点，直到所有节点都处理完。具体算法见文献[3]。

3.2 DSPT 算法

AFST 算法的第 5 步[3]针对非融合节点对，将数据经 SPT 最短路径“直传”给 sink 节点，但由于每次融合后数据量会发生频繁变化，即边的权值发生变化，原来的最短路径此时很可能不是真正的最短路径。针对该问题，将原第 5 步用的 SPT 路由算法用 DSPT 算法代替，对发生变化而受影响的节点重新计算最短路径，以最小的代价适应数据流的实时变化。采用 DSPT 后的 AFST 算法记为 DynAFST 算法。算法基本思想如下：由于此时数据融合没有得益，因此数据不用融合或配对，数据要以最短路径“直传”到 sink 节点才是最节能的。此时所有的非融合节点组成了一个新图 $\overline{\overline{G}}$。根据节点流出数据量的实时变化情况(如因融合程度变化，融合后数据量发生改变)在最小范围内重新计算直传数据的最短路径，使得到的动态最短路径树 T^*_s 与原树 T_s 差别最小，且通过整枝处理的方法能减少迭代次数，起到为传感器节点节能的效果。算法描述如下：

输入 $\overline{\overline{G}}$，sink 节点 t，数据量变化前 $\overline{\overline{G}}$ 对应的以 t 为源点的最短路径树 T_s，增加或减少(不是两者同时)了数据量的边集合 $\varepsilon^{+-}=\{\langle e_i,\tau_i\rangle \mid e_i\in E \text{ and } w(e)^*=w(e)\pm\tau_i\}$。

输出 边更新后图 $\overline{\overline{G}}*$ 的以 t 为源点的最短路径树 T^*_s

约定 对任意节点 v，d_v 表示它的最短路径长度；$status(v)$ 用来存放节点 v 的状态，它有 2 个值：状态 open 表示 v 尚未处理，状态 closed 表示 v 已经处理过，或称归并过；所有顶上加“ ^ ”标识的表示循环过程中该量的当前值，加“ * ”表示该量的最终值。loccally affected 节点是指所有因权值增加而最短路径(路线)发生变化的节点；边界点 boundary vertex 指与原 T_s 中未受影响的、或与受影响但已被归并处理的节点能直接连接的 locally affected 节点；内部节点 inner vertex 表示是 locally affected 节点，但不是边界点的节点，即与原 T_s 中不受影响节点不能直接连接的点。针对边值增加和减少 2 种情况，每个边界点加入待处理节点队列 Q 分别采用如下数据结构：〈边界点，其候选父节点，〈δ，候选最短距离〉〉和〈边界点，其候选父节点，候选最短距离〉。

具体算法分为 2 个部分：MBallStingInc 算法和 DynDijkDec 算法，分别针对数据量增加和减少的情况，2 个算法的伪代码如下：

MBallStringInc 算法
//第 1 步
$\varepsilon \leftarrow \varnothing$
for each $e_i \in \varepsilon^+$ do
update e_i 's edge weight in $\hat{G}$
if e_i is an edge in $\hat{T}_s$ then
remove it from $\hat{T}_s$ and add it to ε
end if
end for
$\overline{N} \leftarrow$ findLocallyAffectedVertices($\hat{T}_s,\varepsilon$)
//第 2 步
for each vertex $a \in \overline{N}$ do
status(a) $\leftarrow$ open
newdist $\leftarrow \min(\{d_b + w(b,a)^* \mid (b,a)\in In_a$
and $b \notin \overline{N}\} \cup \{\infty\})$
if newdist $\neq \infty$ then
$\delta \leftarrow$ newdist-d_a
ENQUEUE(Q,$\langle a,b,\langle\delta,$newdist$\rangle\rangle$),
其中 b 是 a 的候选父节点
end if
end for
//第 3 步
while $Q \neq \varnothing$ do
$\langle y,x,\langle\delta,d\rangle\rangle \leftarrow$ EXTRACTMIN(Q),
重新置 y 的最短路径父节点为 x
$N \leftarrow$ des($\hat{T}_s$,y)
for each $v \in N$ do
$\hat{d}_v \leftarrow d_v+\delta$
$\widehat{status}(v) \leftarrow$ closed
if $v \in Q$ then
REMOVE(v,Q)
end if
end for
for each $e \in Out_N$ do
if $\widehat{status(head(e))}$=open then
newdist $\leftarrow \widehat{d_{tail(e)}}+w(e)^*$
$\delta \leftarrow$ newdist-$\widehat{d_{head(e)}}$
ENQUEUE(Q,$\langle$head(e),tail(e), $\langle\delta$,newdist$\rangle\rangle$)
end if
end for
end while
return $\hat{T}_s$

DynDijkDec 算法
//第 1 步
for each $e_i \in \varepsilon^-$ do
$w(e_i)^* \leftarrow w(e_i)-\tau_i$
tail $\leftarrow$ tail(e_i),head $\leftarrow$ head(e_i)
if $\widehat{d_{tail}}+w(e_i)^* < \widehat{d_{head}}$ then
$\widehat{d_{head}} \leftarrow \widehat{d_{tail}}+w(e_i)^*$
ENQUEUE(Q,$\langle$head,tail,$\widehat{d_{head}}\rangle$)
end if
end for
//第 2 步

while $Q \neq \varnothing$ do
$\langle y,x,d\rangle \leftarrow$ EXTRACTMIN(Q)
重新置 y 的最短路径父节点为 x。
for each $e \in Out_y$ do
$q \leftarrow$ head(e)
if $\widehat{d_y}+w(e)^* < \widehat{d_q}$ then
$\widehat{d_q} \leftarrow \widehat{d_y}+w(e)^*$
ENQUEUE(Q,$\langle q,y,\widehat{d_q}\rangle$)
end if
end for
end while
return $\widehat{T_s}$

算法说明如下：

(1)若图 $\overline{\overline{G}}$ 中有边的权值增加(即数据量增多)，则采用 MBallStringInc 最短路径算法[4]。

算法的基本思想是找出所有因边的权值增加而最短路径受影响的点，对它引入最短距离增加量 δ，每次取 δ 最小的点来排队，并对以该点为根的子树进行整枝处理。

第 1 步 更新 $\overline{\overline{G}}$ 中数据量增加了的边，若其中有边恰好是原最短路径树 T_s 的边，则从 T_s 中去掉这些边。用函数 findLocallyAffectedVertices()找出所有 loccally affected 节点组成集合 $\overline{N}$。

第 2 步 为每个边界节点找“候选最短距离”和“候选父节点”。计算 $\overline{N}$ 中的每个节点 a 的候选距离 $newdist$，若不为 ∞，说明 a 是边界点。计算该点最短距离的增加量 δ，用于对以 a 为根节点的子树进行统一处理，并将边界点按规定的数据结构加入队列。否则说明该节点是内部节点。

第 3 步 取队列 Q 中 δ 值最小的节点逐一归并处理(若 δ 相等，则取候选距离最小的)。每次取出的边界点 y 的结构 $\langle y,x,\langle\delta,d\rangle\rangle$ 说明 y 的父节点就是 x。T_s 中对以 y 为根的子树(whole branch，包括 y 自身)，都从原树中切掉，接到 y 的新父节点 x 上，且给每个子孙节点的最短距离都增加 δ。于是 y 这整支就都被处理了。这时如果 y 之前有子孙节点已经在队列 Q 里面，就立即删掉队列中的这个点，因为此点的最短距离已经被找到，无需再处理一遍了。$\overline{\overline{G}}$ 中对于所有从这些子孙节点射出的边，如果它的头节点尚未被处理，则重新计算该头节点的候选距离后按规定的数据结构加入队列 Q。重复此步，直到队列空。

每次选择要处理的节点都是以 δ 即权值变化最小为优先选择，将 δ 应用于每个删除变化权值边后原 T_s 形成的子树上，可以很快减少算法迭代次数，而且能使最后生成的新树的结构相对原树改变最小，不会出现一个节点最短距离变了，但仍可以保留它原最短路线却没有保留的情况。

(2)若图 $\overline{\overline{G}}$ 中有边的权值减少(即数据量减少)，则采用 DynDijkDec 最短路径算法[4]。

算法的基本思想是从受影响的头节点 affected heads 开始，对每个头节点，贪婪地遍历 $\overline{\overline{G}}$ 中所有它能到达的点直到对这些点找不出更短的距离。

第 1 步 更新图 $\overline{\overline{G}}$ 中权值减少的边，记为 e_i。对每条 e_i，若其头节点最短距离减少，则更新该头节点最短距离为 $\widehat{d_{tail}}+w(e_i)^*$，并将该头节点以格式 $\langle head_e,tail_e,d_{head(e)}\rangle$ 放入队列 Q。

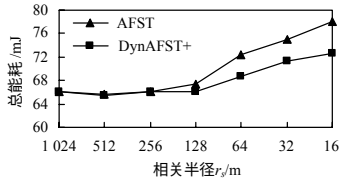
第 2 步 取 Q 中最短距离最小的点，在 $\overline{\overline{G}}$ 中通过遍历它的下一级子节点的方式找出并处理(包括更新它的最短距离和父节点)locally affected 节点加入队列，找不出更短距离的子节点就不是 locally affected，即不加入队列。重复处理直到 Q 为空。

权值减少的情况不同于权值增加的情况，不计算节点的新最短距离就不知道 locally affected 节点集，因为对每条权值变化的边 e，$\overline{\overline{G}}$ 中所有能从 $head(e)$ 连接到的点都可能是 locally affected。因此，在这种情况下，算法从受影响的头节点 affected heads 开始，贪婪地遍历所有它能到达的点直到对这些点找不出更短的距离。

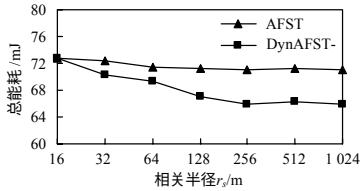
4 仿真实验

本文采用的仿真环境是 UC Berkeley 大学的基于 TinyOS 的无线传感器网络模拟器 TOSSIM，模拟 40 个节点在 40 m×40 m 范围内的随机分布。节点在每个采集周期收集数据，送到位于网络边缘的 sink 节点。r_c 表示该节点的通信范围，在此范围内，单位传输代价 $c(e)=\beta d^2+\varepsilon$，其中，$d<r_c$ 为传输距离；β 为发送机放大器上的消耗，取 100 (pJ·bit^{-1})/m^2；ε 为接受发送电路消耗，取 100 nJ/bit。数据相关性使用空间相关模型，用相关半径 r_s 表示，即在 r_s 内距离越近的节点其数据相关性越强，相关度 ρ 越大，为 $1-d/r_s$；距离超过 r_s 的 2 个节之间的 ρ =0。通过调节 r_s 的大小，可以控制传感器网络平均的数据关联程度，进而影响数据融合后的平均数据压缩率。

实验采用固定通信范围 r_c =30 m 和单位传输代价 ω= 60 nJ/bit 来模拟中等复杂度的单位融合代价，数据量的变化与物理环境、数据相关度、相关半径等因素有关，本文只通过改变相关半径 r_s 来控制数据量的增大和减小。r_s 增大时，数据相关性增大，冗余度高，压缩率就高，数据融合后直传数据量就会减小。反之，当 r_s 减小时，直传数据量就会增大。设置 r_s 的改变每 2 个采集周期触发一次，并且每组实验中 r_s 的改变都是单向离散的。DynAFST 与 AFST 的能耗测试结果如图 1 所示。



(a)数据量增加时的能耗



(b)数据量减少时的能耗

图 1　2 种算法的总能耗(r_c=30 m、ω=60 nJ/bit)

从图 1 可以看到，程序每次开始时 2 种算法能耗完全一样，随着 r_s 骤减(骤增)，数据量增大(减小)，2 种算法能耗开始显现差异。AFST 虽然数据量变化，但仍采用原来首次的

（下转第 153 页）