软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(Suppl.(1)):17−28

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于规则引擎的无线传感器网络动态路由系统*

张足生, 赵铁柱, 袁华强

(东莞理工学院 计算机学院,广东 东莞 523808)

通讯作者: 袁华强, E-mail: M030331@163.com, http://www.dgut.edu.cn

摘 要: 无线传感器网络路由协议往往是针对特定的任务类型和网络状态设计的,动态路由系统可以在运行时,自适应地选择性能最优的路由协议.利用规则引擎设计了一种无线传感器网络动态路由系统.采用了模块化的设计方法,使得多路由协议共存时可以共享资源.利用规则引擎的灵活性和智能性实现路由协议的自适应切换机制.实验结果显示,在多任务网络环境下,动态路由在满足服务质量的同时可以有效地降低网络能耗.

关键词: 无线传感器网络;动态路由;路由协议切换;规则引擎;模块化设计

中文引用格式: 张足生,赵铁柱,袁华强.基于规则引擎的无线传感器网络动态路由系统.软件学报,2015,26(Suppl.(1)):17−28. http://www.jos.org.cn/1000-9825/15003.htm

英文引用格式: Zhang ZS, Zhao TZ, Yuan HQ. Dynamic routing system for WSNs based on policy engine. Ruan Jian Xue Bao/ Journal of Software, 2015,26(Suppl.(1)):17−28 (in Chinese). http://www.jos.org.cn/1000-9825/15003.htm

Dynamic Routing System for WSNs Based on Policy Engine

ZHANG Zu-Sheng, ZHAO Tie-Zhu, YUAN Hua-Qiang

(School of Computer, Dongguan University of Technology, Dongguan 523808, China)

Abstract: Routing protocols in wireless sensor networks tend to be designed for a particular task type and network status. The dynamic routing system can adaptively choose the optimal performance of routing protocols at run time. Based on policy engine, a dynamic routing system is designed for wireless sensor networks. This work uses a modular design approach which makes each sensor node to carry multiple routing protocols. Resources, such as neighbor table and output buffer, are shared between routing protocols. A routing protocol switching mechanism is implemented based on the policy engine. Experimental results show that, in multitasking network environments, dynamic routing can meet the quality of service while effectively reducing energy consumption.

Key words: wireless sensor networks; dynamic routing; routing protocol switching; policy engine; modular design

传感器网络[1]需要在监测区域大规模、密集地部署.大多数节点无法与基站直接通信,路由协议负责在基站和其余节点间建立及维护一个多跳的拓扑结构,为数据提供路由服务.传感器网络的链路质量不稳定、比特出错率较高、节点失效和新节点加入等问题都要求路由协议能够根据网络状态的变化而动态地调整,以达到既满足可靠性要求,同时又节省能量的目的.

传感器网络中的路由协议[2,3]与具体应用紧密相关,没有一个能适用于所有应用的路由协议,同一路由协议在不同的任务及网络状态下的性能不同.而传感器网络需要在相同监测区域内完成不同的任务,在高效完成各种任务的同时,需要尽量减少能耗,延长网络生命周期.通常对于一个传感器网络而言,需要以下几种不同类型的任务:控制指令的下发[4,5]、数据收集[6,7]、数据融合[8,9]和事件触发[10,11].针对这些不同的任务有不同的路由协议.为了能够能量高效地适用于多种任务,传感器网络需要根据具体任务和网络状态自主选择节能、可靠、低延时的路由协议.

* 基金项目: 国家自然科学基金(61402105, 61170216); 广东省自然科学基金(2014A030313631); 广东省科技计划项目(2014B090901064)

收稿时间: 2015-04-15; 定稿时间: 2015-07-20

为了使传感器网络能够能量高效地适应于多任务环境,本文提出了基于规则的动态路由,主要贡献包括:1) 采用了模块化的设计方法,使得多路由协议共存时可以共享资源;2) 实现了一种规则引擎,系统根据任务类型和网络状态自主切换到性能最优的路由协议.

1 相关工作

动态路由是指节点携带多种路由协议,网络根据应用场景[12]的变化自主地选择最优的路由协议进行工作.Heidemann 等人在文献[13]中分析了几类数据扩散协议与应用之间的匹配关系.他们在文献[14]中提出一种基于过滤的体系结构,网络中包含一系列过滤处理函数,函数可以是 GEAR,Gradient 等数据扩散路由协议.让数据包依次经过处理函数,当与某个函数的属性相吻合时,则让数据包作为输入执行该函数.He 等人提出了一个可编程的传感器网络框架[15],实现了路由协议之间的自主切换,当需要使用一个新的路由协议时,该框架利用空中编程的方式对路由协议代码进行更新.

文献[16]针对无线传感器网络数据查询提出了集中控制的路由切换方法.每个节点携带多个路由协议,对路由协议进行模块化设计,减少代码量,增加模块的重用性.基站根据数据查询的类型选择路由协议并将该信息下发到每个节点,当传感器节点收到信息后统一执行路由切换.

文献[17]介绍了一种在 Ad-hoc 网络中根据环境和应用需求的路由协议自动选择工具.在实际应用中,需要根据环境(速度、密度等)与用户需求(比如延时限制)来选择最优的路由协议.该工具的工作过程,需要输入环境信息、用户需求和路由协议相关的先验知识,该工具的输出为推荐的路由协议.该工具的缺点是,路由选择是建立在设计者对环境和被选路由协议的特征非常清楚的基础上,需要的输入条件过于苛刻,它并不是真正意义上的自主路由选择.

文献[18]介绍了一种 Ad Hoc 网络路由协议模块化设计框架,与 He 等人的工作[15]类似,设计者可以通过空中编程的方式对路由协议进行更新,该框架主要关注于如何通过模块化设计,使得一种新的路由协议的更新所需传递的代码量最少.

综上所述,动态路由的研究成果大部分都集中在两个方面:一是如何实现应用场景与路由协议间的匹配;二是设计一种可编程的路由协议框架,实现路由协议的空中编程.目前研究不足在于:无线传感器网络的路由协议,不管是单个还是多个,都是在部署之前已经设计好,哪类数据走哪类路由协议也是事先设定好,无法根据上下文感知进行路由协议的自动优化选择.因此,本文在现有研究的基础上,针对上述问题对动态路由进行研究,实现动态路由系统.

2 规则引擎

规则引擎框架如图 1 所示,由规则说明模块、网络状态模块、决策模块与执行模块这 4 部分组成.传感器网络的设计者或者管理员根据自己的经验和需求来制定规则.在路由协议切换中,网络设计者可以根据任务选择路由协议,这样,只要确定任务类型,就可以通过上位机系统把定义好的规则发送到整个网络.

1) 规则说明模块由 3 个组件组成:规则定义、规则解析、规则发布.规则定义组件允许用户通过图形界面定义管理规则.规则的解析组件负责验证用户定义规则的正确性,并将规则翻译为能被传感器网络节点所理解的数据结构.规则的发布组件负责将解析好的规则发布给传感器网络,传感器节点接收后将会把规则存储在本地的规则表中.使用规则框架定义语言(policy framework definition language,简称 PFDL)来描述规则,规则表示为 IF ⟨condition⟩ THEN ⟨action⟩ SCOPE ⟨scope⟩,其中,⟨condition⟩是一个条件表达式的析取范式;⟨action⟩是一系列动作的声明;<scope⟩是用来表示该规则在传感器网络中的作用域.每个条件由一个或多个逻辑操作符连接,例如,与操作符、或操作符等.条件表达式中的每个条件由一个预变量,一个二元操作符和一个参照值组成,其中,二元操作符限定在{<,<=,=,>=,>},参照值必须为一个真值或者是一个预先定义的一个常数.规则的作用域分为 3 种类型:LOCAL 表示作用域为单个节点;NETWORK 表示作用域为整个网络;BS 表示作用域为基站节点.如果条件表达式都得到满足,那么相应的动作将会被执行.表 1 为一个路由切换规则的示例.在这个规则中,

ISFUSION 条件的参考值 1 是一个事先定义好的一个值,为 1 表示该数据类型为融合数据,为 0 表示不是融合数据.该规则表示如果数据为上行数据,并且为融合数据,那么切换到 HEED 路由协议.

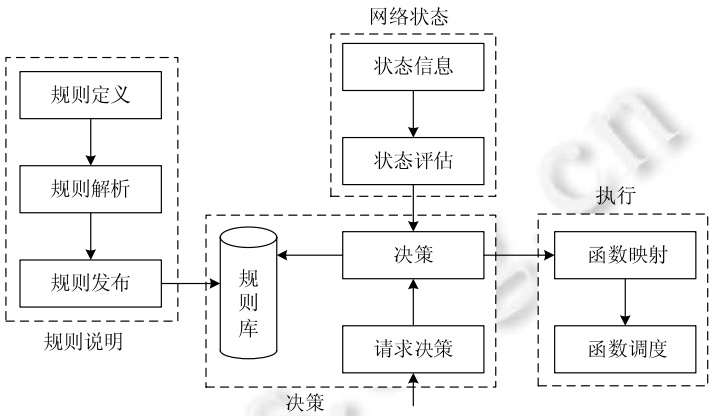

Fig.1 Framework of the policy engine

图 1 规则引擎框架

Table 1 Example of a policy

表 1 规则示例

ID	Condition	Action	Scope
1	IF UPSTREAM=1 AND ISFUSION=1 THEN	SWITCH_TO_HEED	LOCAL

2) 网络状态模块包含两个组件:状态收集与状态评估.状态收集组件负责以适当的频率收集用于衡量网络状态的信息.状态收集调度有两种模式:被动模式,系统以给定的频率周期性地收集信息;主动模式,仅当需要收集该信息时,系统才启动收集信息.状态评估组件将当前的网络状态与事先定义好的网络状态量化指标进行对比,然后评估网络的状态是否发生改变.

3) 决策模块包含两个部分:规则库与决策模块.规则库用来存储网络设计者或用户设定的规则,与规则库相关的操作包括规则的注册、更新与删除.当决策模块收到来自规则系统或者外部的决策请求时,决策模块根据收到的信息与规则库中的规则进行对照分析,如果找到对应的规则,则通知执行模块.图 2 为决策示意图,决策过程就是利用一个函数 $f(x_1,x_2,\ldots,x_i)$将输入映射到一个匹配的规则,然后输出执行动作.

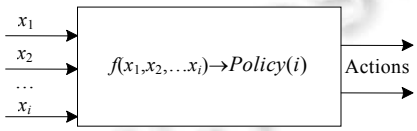

Fig.2 Sketch map of decision

图 2 决策示意图

4) 执行模块有两个组件:函数映射与函数调度.函数映射组件管理执行所要用到的一系列函数,负责在规则与被执行函数之间建立一种映射.如果需要执行的函数不存在,那么函数调度负责通过空中编程的方式将需要执行的函数发送到节点.执行模块是负责执行由规则评估之后所做出的决策,比如,如果规则评估之后需要切换路由协议,该规则的执行就是控制对应路由协议的组网模块的开启与关闭.

3 网络层模块化设计

为了使路由协议适合于路由切换,将协议可共用的模块与私有的模块分开,这样可以增加代码共享,减少存储空间[19].根据功能结构,可以将网络层分解为四大模块,如图 3 所示,数据转发模块(forwarding engine,简称

FE)、路由组网模块(routing engine,简称 RE)、邻居表模块(neighbor table,简称 NT)以及输出队列模块(output queue,简称 OQ).其中,邻居表模块与输出队列模块为路由协议所共享,而数据转发模块与路由组网模块为路由协议特有的模块.

如图 4 所示为路由切换的框架图,其中数据转发模块(FE)和路由组网模块(RE)是多份,因为不同的路由协议有不同的 FE 和 RE.与图 3 不同的是,在路由协议与各层之间增加了 HighDispatcher,LowDispatcher,TopoControl 这 3 个模块.路由切换的设计原则是尽量不改变原有的应用层、网络层、MAC 层的接口,对各层的内部代码尽量少作改动.采用 HighDispatcher 和 LowDispatcher 屏蔽因为路由协议的不同而带来的数据收发的差异,用 TopoControl 屏蔽路由拓扑结构切换的差异.

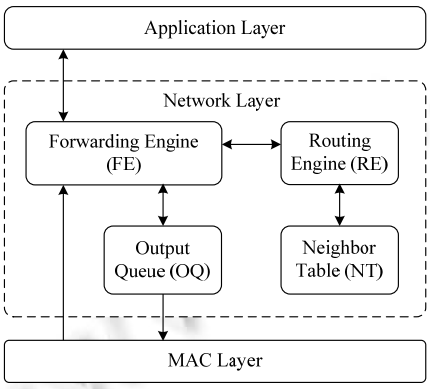

Fig.3 Modular design of network layer

图 3 网络层分解

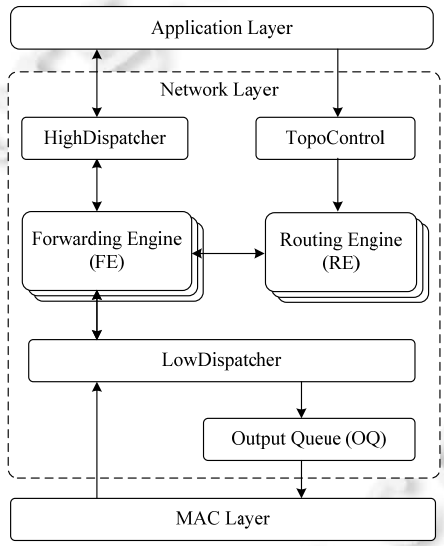

Fig.4 Framework of dynamic routing

图 4 路由切换框架图

HighDispatcher 和 LowDispatcher 模块作为连接器为数据包的收发服务.网络层的包头结构如图 5 所示,在原有的包头结构上增加了 High header 与 Low header 两个域.High header 只能被 HighDispatcher 模块管理与访问,包含任务标识(Application ID,简称 AppID).HighDispatcher 通过 AppID 来实现在应用层模块与 FEs 之间的连接.Low header 只能被 LowDispatcher 模块管理与访问,包含路由协议标识(Routing protocol ID,简称 RpID).LowDispatcher 通过 RpID 来实现 MAC 层与 FEs 之间的切换.

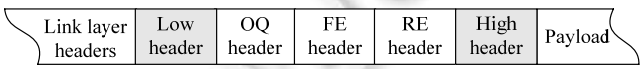

Fig.5 The network layer header of a packet

图 5 网络层包头结构

传感器节点通常有两种角色,一是源节点:若数据包从应用层获得并从应用层发出,该节点就是源节点.其数据包的来源有两种途径:自身的传感器感知获得,或者通过截取网络发送来的数据包进行融合后操作产生一个新的数据包.二是路由器:若节点收到来自网络的包,而本地应用层不需处理,则将该包转发出去.

HighDispatcher 的伪代码如图 6 所示,其目的是向应用层和网络层屏蔽由于多个路由协议而带来的差异性,应用层和网络层的接口相对于单一路由协议来说不需做任何改变.HighDispatcher 作为一个开关器，将数据包在应用模块和各协议的转发模块之间进行切换.它主要完成以下 3 种任务:

1) 发送(send):从规则系统中可以获得 RpID,调用 HighDispatcher 的 send 函数传递数据包并附带 RpID 与 AppID 两个参数.该函数将 AppID 填充在数据包的 High header 域,然后根据 RpID 把消息转发给对应路由协议

的 FE 模块进行处理.

2) 接收(receive):当路由协议的 FE 模块接收到来自网络的数据包时,通过对比该节点的地址与数据包的目的地址,如果一致,就通知 HighDispatcher 进行接收,HighDispatcher 根据数据包的 AppID 通知相应的应用程序进行处理.因为每个应用程序产生的包只能被该应用程序所解析.

3) 截取(intercept):当收到来自网络的数据包时,有些应用层程序需要对该包进行数据融合或者数据压缩,所以需要从网络层截取该包.intercept 函数根据 AppID 检查应用程序是否注册了截取事件,如果没有,则通知相应的 FE 对该数据包进行转发;如果有,就将该数据包通知上层进行处理.

```
HighDispatcher {
    //send a local packet
    send (int RpID, int AppID, message* msg) {
        Fill the AppID field in the msg header;
        switch RpID:
        case Rpi: call Rpi-FE.send(msg);
    }

    //receive a packet from network
    receive (message* msg) {
        Get the value of AppID field in the msg header;
        switch AppID:
        case Appi: signal Appi.receive(msg);
    }

    //checks for packet interception requests from higher layers
    intercept (int RplD, message* msg) {
        Get the value of AppID field in the msg header;
        switch AppID:
        case Appi:
            //application layer registers to intercept the packet
            if (Appi.registerIntercept)
                signal Appi.Intercept(msg);
            //application layer does not register to intercept the packet
            else
                switch RpID:
                case RpIDi: call RpIDi-FE.forward(msg);
    }
}
```

Fig.6 Pseudo-Code of the HighDispatcher model

图 6 HighDispatcher 模块的伪代码

LowDispatcher 的伪代码如图 7 所示.LowDispatcher 为网络层与 MAC 层之间提供无差异访问,其作为开关器实现在 FEs 与 MAC 协议之间的切换.它主要有以下 3 个方面的功能:

1) 发送(send):路由协议将源节点产生的数据包发送到 LowDispatcher 模块并附带 RpID 参数,LowDispatcher 模块往数据包的 Low header 域填充 RpID 字段,然后将该数据包发送到输出队列.

2) 接收(receive):接收来自 MAC 层的数据包,通过该包的 RpID 字段将该包交给对应的路由协议的 FE 模块.

3) 转发(forward):路由协议需要转发一个数据包,该函数对转发的数据包不做任何处理,直接将其发送到输出队列.

```
LowDispatcher {
    //send a local packet
    send(int RpID, message* msg) {
        Fill the RpID field in the msg header;
        send the msg to Output Queue;
    }

    //receive a packet from network
    receive(message* msg) {
        Get the value of RpID field in the msg header;
        switch RpID:
        case Rp1: signal Rp1-FE.receive(msg);
        case Rp2: signal Rp2-FE.receive(msg) ;
        …
        case Rpi: signal Rpi-FE.receive(msg);
    }

    //forward a packet to output queue
    forward(message* msg) {
        send the msg to Output Queue;
    }
}
```

Fig.7 Pseudo-Code of the LowDispatcher model

图 7 LowDispatcher 模块的伪代码

4 路由协议切换

基于规则的路由切换由于决策权在于节点本身,节点可以根据任务类型和网络状态自适应地做出路由切换.任务类型可以用任务的特征来描述,比如,数据的流向是上行还是下发、数据收集、数据融合等;而网络状态用来反映网络的全局或者局部的状态信息,例如,节点密度、网络拥塞情况、数据可靠性等.

1) 路由切换规则:在 TinyOS-2.x 操作系统上实现了 Dissemination[4],CTP[6],HEED[9]这 3 个路由协议之间基于规则的切换.CTP 是 TinyOS-2.x 自带的一种树型结构的数据收集协议.定义的规则见表 2,其中,规则的条件部分:isUpstream 表示数据是上行还是下行,isFusion 表示该数据是否为融合数据,isEvent 表示该数据是否为报警数据.规则的动作表示切换到相应的路由协议.规则 1 表示如果是数据下行,则切换到 Dissemination;规则 2 表示如果是数据上行,且该数据不融合(即为数据收集(data acquisition)),则切换到 CTP;规则 3 表示如果是数据上行,且为融合数据(data aggregation),则切换到 HEED.这些规则都放在传感器节点上执行.在用户端用 XML 语言表示规则,然后基站将规则翻译成传感器节点能理解的数据结构.

Table 2 Policy table

表 2 规则表

Policy ID	Condition			Action
	isEvent	isFusion	isUpstream	Switch to protocol
1	0	0	0	Dissemination
2	0	0	1	CTP
3	0	1	1	HEED

2) 拓扑的切换:如图 8 所示为基于规则的拓扑切换过程,其中,Data Acquisition 数据不融合,Data Aggregation 数据须融合,实线为数据收集任务的拓扑切换过程,虚线为数据融合任务的拓扑切换过程.TopoControl 为应用层与路由协议之间提供一种开关器,两个路由协议共用邻居列表.下面以数据收集任务为例来解析基于规则的拓扑切换过程,节点收到数据收集查询任务,数据收集模块请求 TopoControl 模块启动建立网络拓扑结构.TopoControl 模块向规则系统提出路由切换决策请求,并附带任务类别与数据流向等参数.规则库见表 2,规则决策模块根据当前网络状态及输入的参数决定使用 CTP 路由协议来处理该查询的数据流,并触发

TopoControl 模块进行路由切换.TopoControl 调用 CTP 路由协议的组网程序进行组网.

3) 数据的收发:在网络中有 3 种类型的数据需要收发,一类是建立路由的过程中需要用到的 Beacon 包;另一类是节点的采样数据,称为数据包;第 3 类为控制指令,比如规则信息,网络配置参数等.Beacon 包只在一跳范围内交换,为每个路由协议所特有,所以规则系统对 Beacon 包的收发不做任何改变.而数据包与控制指令需要根据任务类型和网络状态采用不同的路由协议进行收发.图 9 描述了从本地节点应用层发送数据包的过程.

S1. 节点采样,产生数据包,将数据包发送给 HighDispatcher 模块.

S2. HighDispatcher 填充 AppID 包头字段,并将该数据包发送给规则系统请求决策.

S3. 规则决策模块根据网络状态和数据包携带的信息进行决策.

S4. 将决策结果反馈给 HighDispatcher 模块,例如决策结果是采用 CTP 的转发模块发送此数据包.

S5. HighDispatcher 选择 CTP 路由协议的数据转发模块进行发送.

S6. CTP-FE 填充数据包的路由头部,将数据发送给 LowDispatcher,携带参数 RpID.

S7. LowDispatcher 填充 RpID,并将数据发送给输出队列.

S8. 输出队列将数据发送给 MAC 层.

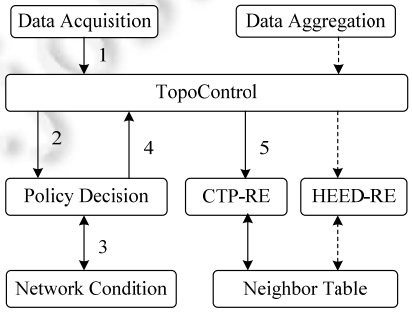

Fig.8 Switching of network topology

图 8 基于规则的拓扑切换

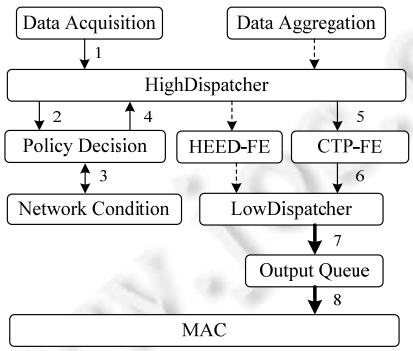

Fig.9 Processing of a packet is sent

图 9 从本地节点发送数据包的处理过程

图 10 描述了从网络中接收一个包的处理过程,该包有两种可能,一是采样数据包,二是控制指令包.

S1. MAC 将接收的数据包发送给 LowDispatcher 模块.

S2. LowDispatcher 模块根据 RpID 通知对应的路由协议进行接收,假定 RpID 对应的是 CTP 路由协议.

S3. CTP 转发模块将数据包发送给 HighDispatcher 模块.

S4. HighDispatcher 查看该数据是否被上层应用所截取,如果没有,则将数据发送给规则模块进行决策.

S5. 规则模块根据数据包的内容与网络状态进行决策.

S6. 将决策结果反馈给 HighDispatcher,假定决策结果仍然采用 CTP 进行转发.

S7. HighDispatcher 检查决策后的路由协议的选择是否发生改变.如果路由协议发生改变,如图 6 所示,则取出该数据包的应用层部分,进行重新封装,当作一个新的数据包发送给改变后的路由协议.如果路由协议没有改变,则通知 CTP 的数据转发模块进行转发.

S8. CTP 的转发模块对该数据包进行转发.

S9. LowDispatcher 模块转发该数据包给输出队列.

S10. 输出队列将数据发送给 MAC 层.

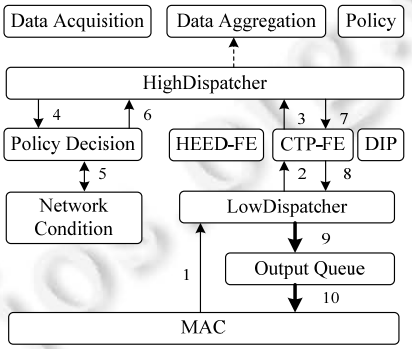

Fig.10 Processing of a packet is received

图 10 从网络中接收一个数据包的处理过程

5 仿 真

采用简单能量模型[20]进行能量分析,如式(1)与式(2)所示,E_{Tx} 是发送 l-比特数据的能耗,E_{Rx} 是接收数据的能耗,d 是通信距离,d_0 是临界距离.

$$E_{Rx}(l)=lE_{elec} \tag{1}$$

$$E_{Tx}(l,d)=\begin{cases} lE_{elec}+l\xi_{fs}d^2, & d<d_0 \\ lE_{elec}+l\xi_{mp}d^4, & d>d_0 \end{cases} \tag{2}$$

将 100 个传感器节点随机分布在 100m×100m 的区域内,基站节点位于(0,0)处,节点的最大通信半径为 30m,仿真场景参数见表 3.采用有效能耗作为性能评价标准.有效能耗是网络消耗的能量和基站成功接收包的数量的比值 *Total*(dissipated energy)/*Num*(received data at BS).有效能耗越少,路由协议的性能就越好.

Table 3 Parameters of simulation

表 3 仿真参数

参数名称	值
网络大小	100m×100m
基站位置	(0m,0m)
节点个数	100
节点最大通信半径	30m
簇内最大通信半径	15m
数据包的大小	500bytes
控制包的大小	25 bytes
包头大小	25 bytes
初始能量	2J
E_{elec}	50nJ/bit
ξ_{fs}	10pJ/bit
d_{cross}	75m
E_{fusion}	5nJ/bit
ξ_{tr}	0.001 3pJ/bit
路由维护周期	60s