**电 子 科 技 大 学**

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

**学士学位论文**

**BACHELOR THESIS**



论文题目 **传感器网络虫洞攻击检测研究**

专 业 **信息安全**

学 号 **2013060202009**

作者姓名 **杨涵麟**

指导教师  **鲁力**

摘 要

无线传感器网络（Wireless Sensor Networks，WSN）是当前国际上备受关注的、涉及多学科高度交叉、知识高度集成的前沿热点研究领域。传感器技术、微机电系统、现代网络和无线通信等技术的进步，推动了现代无线传感器网络的产生和发展。

而在无线传感器网络中，虫洞攻击是一项非常重要的安全问题。攻击者通过在两个节点间建立高速通道，可以吸收大量的流量进而实现窃取、监听、篡改、等多种攻击方式。本文就如何检测虫洞攻击提出了一种新的方法。

在实验中，我们发现，当一个网络中被植入了虫洞节点之后，整个网络的平均路由跳数和延时都会有较大的降低，而这时普通的网络波动所做不到的。因此我们可以藉由观察网络状态概率性的判断网络中是否存在虫洞节点。

同时，虫洞节点为了吸收大量的网络流量，必然在所有节点的路由路径中出现的概率远高于正常节点。而连接虫洞节点的两个端点也由于高速通路的存在，而大幅度提高在所有节点路由路径中出现的概率。由前者我们更可加精确的判断是否存在虫洞节点的存在，由后者，我们可以定位到节点的位置。

本文就是根据观察到的这些现象，提出了一种新的虫洞检测算法。这种算法只需要在原来的包结构上进行小幅度的修改，对中间节点的计算仅造成极微小的负担。因此可以适用于各类无线传感器网络。

**关键词：WSN 虫洞攻击 无线传感器网络**

ABSTRACT

Wireless sensor networks (Wireless Sensor Networks，WSN) is a international, highly multidisciplinary, highly integrated knowledge of advanced hot research field. Sensor technology, micro-electro-mechanical systems, network and wireless communication technology advances, promotes the emergence and development of modern wireless sensor networks.

While in the field of wireless sensor networks, wormhole attack is a very important security issues. Attacker can absorb a lot of traffic by establishing a high-speed channel between two nodes in the network, after that the attacker can theft, eavesdrop, tamper and analysis protocol format. This essay developed a new way to detect the wormhole attack.

In experimenting, we discovered one symptom which that after a wormhole inserted to the network, the network’s average hop count and delay must be cut down deeply. And this degree of decline can not be done by network fluctuation. So we can estimate the status of network to determine where there is a wormhole or not with certain probability.

Meanwhile, because the wormhole will absorb a lot of traffic, the proximity of appearance of wormhole node will increase hugely in the network’s all route paths. And the two nodes connected to the wormhole directly will do so as well. By the former, we can increase our algorithm’s accuracy further. And by the latter, we can locate the wormhole node easily.

My new algorithm is based on these symptoms. This algorithm will ask a little change on the packet’s structure and consume computational power a little more. So this algorithm can adapt to new network environment breezily.

**Keywords:**  **WSN Wormhole Attack Wireless Sensor Network**

目 录

第一章 绪 论 1

1.1 研究工作的背景与意义 1

1.2 国内外研究历史与现状 1

1.3 主要研究内容与创新 2

1.4 本论文的结构安排 2

第二章 无线传感器网络基础 4

2.1 无线传感器网络基础 4

2.1.1 体系结构 4

2.1.2 无线传感器网络的安全需求 5

2.1.3无线传感器网络的重要性 7

2.2 无线传感器网络安全性分析 7

2.2.1 攻击种类 7

2.2.2 无线传感器网络安全防范对策 8

2.3 本章小结 10

第三章 算法测试系统设计 11

3.1 系统需求分析 11

3.2 系统设计 11

3.2.1 系统使用说明 12

3.2.2 类设计 12

3.3 系统主要算法及架构解释 14

3.3.1 网络设计 14

3.3.2 路由算法的设计 16

3.3.3 虫洞节点发现算法-四状态法 17

3.5 本章小结 19

第四章 测试系统功能实现 20

4.1开发平台 20

4.2 Node类的实现 20

4.2.1 Node类成员的说明 20

4.2.2 Node类函数的说明 21

4.2.3 Node类重要方法的实现详解 23

4.3 Wormhole类的实现 24

4.3.1 Wormhole类成员的说明 24

4.3.2 Wormhole类函数的说明 24

4.3.3 Wormhole重要方法的实现详解 24

4.4 RouteTable类的实现 26

4.4.1 RouteTable类成员的说明 26

4.4.2 RouteTable类函数的说明 26

4.4.3 RouteTable重要方法的实现详解 26

4.5 Cache类的实现 27

4.5.1 Cache类成员的说明 27

4.5.2 Cache类函数的说明 27

4.5.3 Cache类重要方法的实现详解 28

4.6 WSNNetwork类的实现 28

4.6.1 WSNNetwork类成员的说明 28

4.6.2 WSNNetwork类函数的说明 28

4.6.3 WSNNetwork类重要方法的实现详解 29

4.7本章小结 29

第五章 系统测试及算法测试 30

5.1 软件测试 30

5.2 测试步骤 31

5.2.1 功能点的划分 31

5.2.2 网络连通性 32

5.2.3 虫洞节点对信息的窃取 32

5.2.4 汇聚节点对虫洞的检测 33

5.3 算法实用性测试 35

5.3.1 选取特定虫洞节点的测试 35

5.3.2 随机节点的虫洞测试 35

5.3.4 算法测试小结 36

5.3 本章小结 37

第六章 结束语 38

致 谢 39

参考文献 40

第一章 绪 论

1.1 研究工作的背景与意义

随着无线技术的快速发展和日趋成熟[1]，无线通信也发展到一定的阶段，其发展的技术越来越成熟，方向也越来越多，越来越重要，大量的应用方案开始采用无线技术进行数据采集和通信。微机电系统和低功耗高集成数字设备的发展，使得低成本、低功耗、小体积的传感器节点得以实现。这样的节点配合各类型的传感器，可组成无线传感器网络（WSN）。无线传感网络是一种开创了新的应用领域的新兴概念和技术。广泛应用于战场监视、大规模环境监测和大区域内的目标追踪等领域。传感技术、传感网络已经被认定为最重要的研究之一。2003年,美国《技术评论》[2]杂志论述未来新兴十大技术时,WSN被列为第一;美国《今日防务》杂志更认为WSN的应用和发展将引起一场划时代的军事技术革命和未来战争的变革。可以预测,WSN是信息感知和采集的一场革命,是21世纪最重要的技术之一。低功耗无线传感模块，便是组成无线传感网络的节点。此方面的研究由来已久，是计算机应用的扩展，采用了大规模集成电路和嵌入式技术，使用智能微处理器对采集到的信息进行处理和加工。现已广泛应用于社会建设的各个层面和人们的日常生活当中。

由于传感器网络一般配置在恶劣环境、无人区域或敌方阵地中，加之无线网络本身固有的脆弱性，因为传感器网络安全引起来人们的极大关注，传感器网络的许多应用（如军事目标的检测和跟踪等）在很大程度上取决于网络的安全运行，一旦传感器网络受到攻击或破坏，将可能导致灾难性的后果，如何在节点计算速度、电源电量、通信能力和存储空间非常有限的情况下，通过设计安全机制，提供机密性保护和身份认证功能，防止各种恶意攻击，为传感器网络创造一个相对安全的工作环境，是一个关系到传感器网络能否真正走向实用的关键性问题。[3]

1.2 国内外研究历史与现状

传感器网络的研究起步于20世纪90年代末期，但安全同题的研究成果近几年才陆出现，且大多数方案都是基于AdHoc及传统网络的安全机制，密钥管理体制也并非完全意义上的分布式自组织方案，而是分级式的管理方案，因此，在传感器网络还未被模型化和量化之前，无线传感器网络安全方案正处于理论研究阶段，距离实际应用和形成普遍接受的标准还相差甚远。[2]

1.3 主要研究内容与创新

基于Python多线程编程技术，编写一套网络架构，实现节点发现、路由寻址、载波侦听、包交换等WSN网络核心功能，模拟一个节点数目在100以上的无线传感器网络系统。

在这个基础上，再增加虫洞模块，可以实现对包的截取，流量的吸收。

最后，在现有的虫洞检测算法上推出一套新的虫洞检测算法，并实现，要求能检测出虫洞的存在并定位虫洞的逻辑位置。

从架构上来看，主要包括如下几大功能模块：

（1）网络架构模块：最基础的核心模块，主要包括节点类的设计与实现、汇聚节点（Sink Node）类的设计与实现、虫洞节点的设计与实现、路由表的设计与实现、网络整体的设计与实现、物理层帧（Frame）的设计与实现、网络层包（Packet）的设计与实现以及应用层载荷（Payload）的设计与实现。

（2）网络通信模块：包括帧的交换的实现、载波侦听的模拟、帧的处理模块的实现、包的处理模块的实现、载荷的处理模块的实现、路由表建立与更新模块的实现、路由算法的设计与实现、邻居节点发现算法的实现。

（3）虫洞攻击模块：主要包括虫洞节点插入的实现，虫洞节点对信息处理模块的实现，虫洞节点的高速连接路径的实现。

（4）虫洞检测模块：包括为了虫洞检测的辅助数据结构的实现（如拓扑图），网络状态变化监测的实现、虫洞攻击前后的状态比较的实现、虫洞判断检测模块的实现。

（5）网络布置模块：包括对网络的设置以及节点的分布、节点数目的选取、虫洞节点插入位置的选取的实现。

本文的主要创新点：

（1） 该系统的架构设计与实现完全由本人完成，有非常大的灵活性，且基于Python进行实现，后期可扩展性非常强，没有利用已经成熟的方案，免除了安装平台的麻烦。

（2） 基于现有的算法，改进以适应现有架构，提出了一套新的虫洞检测方案，与原方案相比，对于中间节点的要求更低，虫洞检测的方法更加简单，但是带来的问题是与原方案相比，实时性不强。

1.4 本论文的结构安排

本文的章节结构安排如下：

第一章介绍了无线传感器网络的的研究工作的背景与意义、国内外研究历史与现状及主要研究内容与创新；第二章主要介绍了无线传感器网络系统的相关技术，主要内容有系统的特点、系统架构、系统的安全性需求、系统的重要性等；第三章主要介绍本次实验的方案设计，包括需求分析、软件设计、系统功能模块设计、系统的基本框架，最后详细阐述了虫洞检测的核心算法；第四章主要介绍测试系统的功能实现，包括介绍了系统开发使用的平台以及各个类的核心代码的具体实现；第五章主要介绍系统的测试，首先介绍何为软件测试、接着撰写测试步骤、最后根据单元划分对该系统进行测试，在完成系统测试之后对论文提出的方法和可行性进行验证；第六章则是总结与展望，对本文的一个总结概括以及提出可以改善的地方。

第二章 无线传感器网络基础

2.1 无线传感器网络基础

2.1.1 体系结构

无线传感器网络结构如图2-1所示。传感器网络系统通常包括传感器节点（Sensor Node）、汇聚节点（Sink Node）和管理节点。大量传感器节点随机部署在检测区域（Sensing Region）内，以自组织方式构成网络，通过多跳中继方式将检测到的数据传送给汇聚节点，最后通过互联网或者卫星传达给管理节点或者数据中心。用户通过管理节点或者数据中心[4]对传感器网络进行配置和管理，发布检测任务以及收集检测数据。

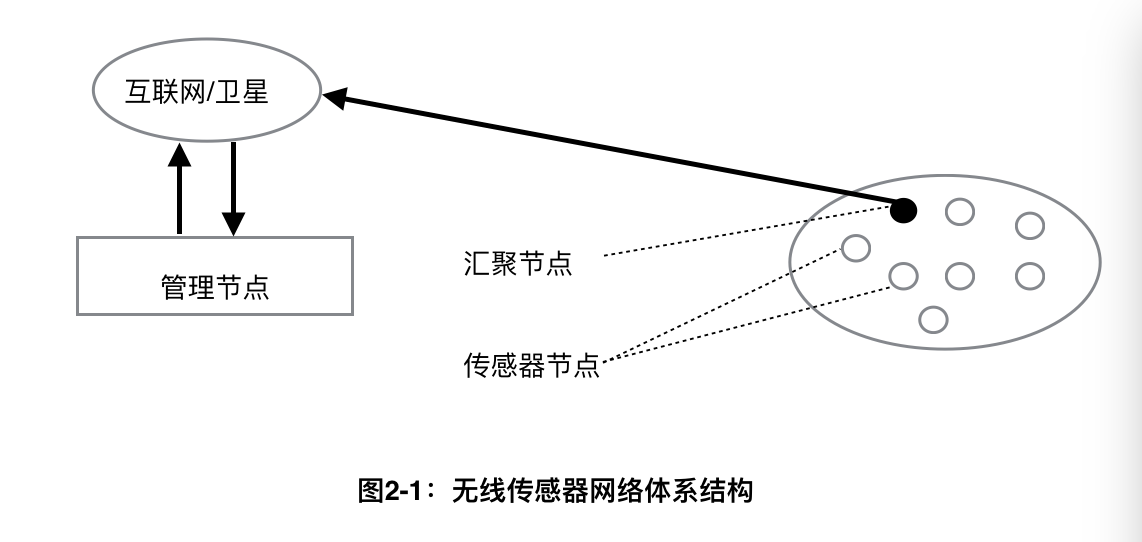
 传感器节点是一个微型的嵌入式系统[5]，计算能力、存储能力和通信能力非常有限，能量也很有限。传感器节点除了进行本身信息的收集之外，还要对其他节点转发来的数据进行融合。相比较而言，汇聚节点各方面的能力要强得多，并且具有足够的能量供给。汇聚节点通常与外部网络直接相连，负责发布管理节点的检测任务，并把收集的数据转发给外部网络。

图2-1 无线传感器网络结构

传感器节点一般由数据采集模块、处理器模块、无线通信模块和能量供应模块这四部分组成。数据采集模块负责数据的采集和转换。处理器模块负责数据处理。无线通信模块负责与其他节点进行数据传输，能量供应模块负责运行所需的能量，通常采用微型电池。

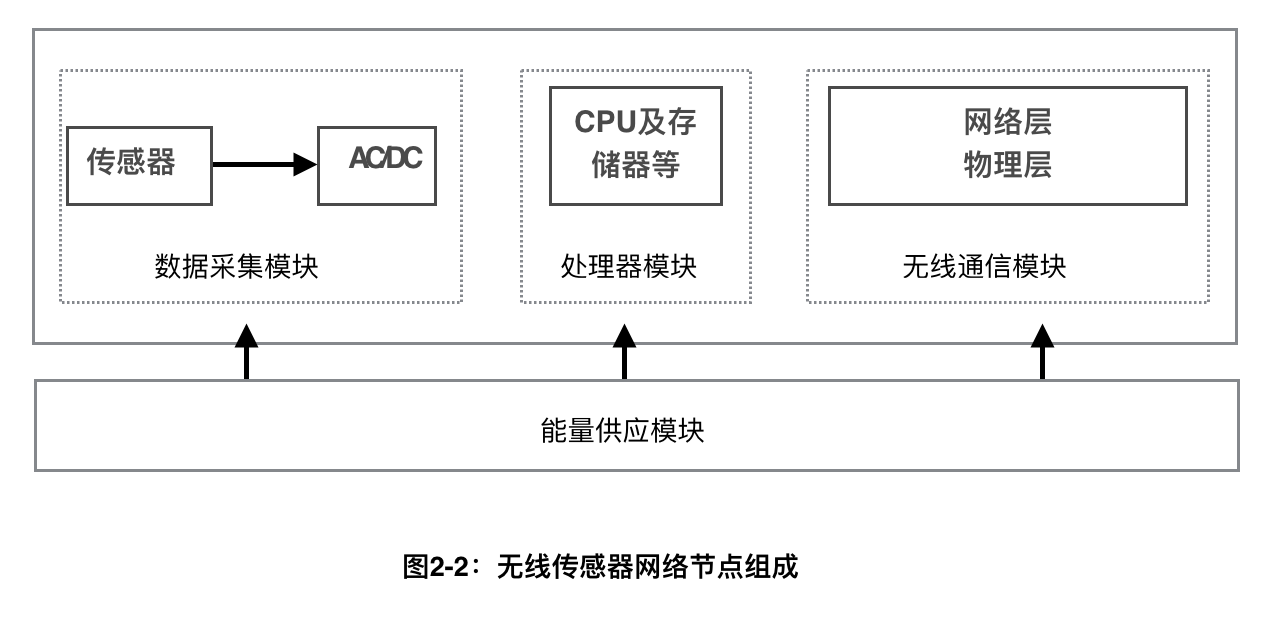
 传感器节点的处理器通常使用嵌入式CPU，如Intel的8086，另外系统还需要一个微型化的操作系统以进行任务调度与管理，如UC Berkeley的TinyOS，嵌入式Linux等。图2-2描述了节点的组成，数据感知单元通过对传感器所在区域进行数据采集和感知，进行模数转换，经由数据处理单元对数据信号进行简单处理后由数据传送单元调制后发射出去。

图2-2 无线传感器网络节点组成

数据采集模块：由一组传感器和数模转换装置构成的数据采集模块，主要负责将周围环境的物理现象转换成数字信号，例如测量所在周围环境中的热、红外、声呐、雷达和地震波信号，从而探测包括温度、湿度、噪声、光强度、压力、土壤成分、移动物体的大小、速度和方向等众多用户感兴趣的物理参数。数据采集模块提供了采集信息的能力，将数字世界与物理世界联系起来。

处理器模块：由处理器和存储器构成的处理器模块，负责协调无线传感器各个模块的工作，如对数据采集模块获取信息进行必要的处理和存储，控制无线通信模块和能量供应模块的工作模式等。处理器模块[6]提供了处理信息的能量，将无线传感器节点智能化。

无线通信模块：由短距离无线收发电路构成的无线通信模块，负责与其他无线传感器邻居节点或基站进行无线通信。无线通信模块提供了传输信息的能力，将单独的无线传感器节点连接成为协作网络。

能量供应模块：由电池构成的能量供应模块，为无线传感器的其他模块提供电源。

2.1.2 无线传感器网络的安全需求

无线传感器网络可能会遇到窃听、消息修改、消息注入、路由欺骗、拒绝服务、恶意代码等安全威胁。另外，在无线传感器网络中，安全的概念也发生了变化，通信安全是其中重要的一部分，隐私保护日渐重要，而授权重要性则降低。

无线传感器网络的开放性分布和无线广播通信特征存在安全隐患，而不同应用背景的无线传感器网络对信息提出了不同的保护需求。无线传感器网络的安全需求主要表现为以下几个方面：

（1）机密性

机密性是确保传感器网络节点间传输的敏感信息安全的基本要求。无线通信的广播特征使得通信很容易被监听。机密性要求窃听方即使截获节点间的物理通信信号仍然不能知道其所携带的消息内容。

（2）完整性

无线传感器网络的通信环境给恶意节点实施数据丢失或损坏攻击提供了方便。完整性要求网络节点收到的数据包在传输过程中未被插入、删除、篡改等，即保证收到的消息和源方发出的消息是完全一致的。

（3）真实性

无线传感器网络的真实性需求主要体现在点对点的消息认证和广播认证，前者指任何一个节点在收到来自另一个节点的消息时，能够核实这个消息来源的真实性，不是被伪造或假冒的。后者解决的是单一节点向一组节点发送统一通告时的真实性确认问题。

（4）可用性

可用性要求无线传感器网络能够随时按预先设定的工作方式向系统合法用户提供信息访问服务，但攻击者可以通过复制、伪造和信号干扰等方式使传感器网络处于部分或全部瘫痪状态，从而破坏系统的可用性，典型的，如拒绝服务（DoS）攻击。

（5）新鲜性

无线传感器网络中节点数目众多，其多路径消息传输机制或重放攻击可能使目标接收方收到延后的相同数据包。新鲜性[7]要求接收方收到数据包都是最新的、非重放的，即体现消息的时效性。

（6）鲁棒性

无线传感器网络应用具有很强的动态性和不确定性，包括网络拓扑的变化、节点的去除或加入、面临多种威胁等，因此，无线传感器网络对各种安全攻击应具有强适应性和存活性，即使某次攻击行为得逞，该特征要求其影响被最小化，单个节点受到威胁并不会导致整个网络的瘫痪。

（7）访问控制

访问控制要求能对访问无线传感器网络的用户身份进行确认，确保其合法性。但传感器网络区别于传统网络的是每个节点都是物理可访问的，不能设置防火墙进行访问过滤。无线传感器网络的资源受限特征也使得基于非对称加密体制的数字签名和公钥证书体制难以应用。

2.1.3无线传感器网络的重要性

无线传感器网络集成了传感器、微机电系统和网络3大技术，是一种全新的信息获取和处理技术。它能够协作的实时监测感知和采集网络分布区域内的各种环境或监测对象的信息，并对这些信息进行处理，获得详尽而准确地信息传送到需要这些信息的用户。传感器网络可以使人们在任何时间、地点和任何环境条件下获取大量详实而可靠的信息。无线传感器网络系统可以被广泛地应用于军事领域、环境、科学、医疗、健康、空间探索、交通管理制造业、反恐抗灾等领域。

（1）军事领域

由于无线传感器网络节点具有小巧、隐藏性好、环境适应好等特点，使其非常适用于恶劣的战场环境中，可以应用在侦察敌情、监控兵力、装备和物资等多方面用途。

（2）环境的检测和预报

随着人们对于环境问题的关注程度越来越高，需要采集的环境数据也越来越多，越来越具体，无线传感器网络的出现为随机性的研究数据获取提供了便利。无线传感器网络还可以跟踪候鸟和昆虫的迁移，研究环境变化对农作物的影响，检测海洋、大气和土壤的成分等。

（3）医疗护理

无线传感器网络在医疗研究、护理领域也可以大展身手。罗彻特大学的科学家使用无线传感器创建了一个智能医疗房间，使用微尘测量居住者的重要征兆、睡觉姿势以及每天24小时的活动状况。

（4）其他用途

无线传感器网络还被应用于其他一些领域。比如一些危险的工业环境如矿井、核电厂等，工作人员可以通过它来实施安全监测，也可以用在交通领域作为车辆监控的有力工具。此外还可以在工业自动化生产线等诸多领域。

2.2 无线传感器网络安全性分析

2.2.1 攻击种类

对于无线传感器网络的攻击[8]主要分为主动式攻击和被动式攻击。被动式攻击并不破坏路由协议的执行过程而仅仅通过侦听路由的通信过程获取对它有用的路由信息。由于无线频谱的开放性，对这类攻击的检测相当困难。主动式攻击指利用恶意节点阻止路由的建立、更改包的传送方向、中断路由的使用以及利用虚假数据欺骗网络的认证和授权等破坏性行为。主动性攻击可进一步分为外部攻击和内部攻击。外部攻击是指位于网络外部的攻击者对网络发起的攻击，而内部攻击是指经过了认证和授权已成为网络成员的恶意节点（即Compromised Node叛变节点）在网络内部发起的攻击。显然，内部攻击的破坏性更强。这里将网络路由攻击方法归结为以下几种：窃听、欺骗、篡改或重放路由信息、选择性转发攻击、虫洞攻击等。

1.窃听、欺骗、篡改或重放路由信息

对路由协议最直接的攻击目标是节点之间交换的路由信息。攻击者通过窃听、欺骗、篡改或重放路由信息，可生成路由环路、引诱或拒绝通信流量、延长或缩短源路由、产生虚假的错误消息、分隔网络、增加端到端的延迟等。

2.选择性转发攻击

多跳网络通常假设参与传输的节点会透明的转发它接受到的消息。在选择性转发攻击中，恶意节点可能会拒绝转发某些消息并丢弃它们。该攻击的一种简单形式就是恶意节点像黑洞一样拒绝转发它接受到的数据包。但是，该攻击的弱点就是邻近节点可识别出恶意节点的无效性从而选择另外一条路由。因此，该攻击的一种更巧妙形式就是攻击者选择性地转发数据包，攻击者通过删除或更改一些感兴趣节点的数据包，并将其它信息进行转发以减少节点对其非法操作的猜疑。当攻击者处于数据流路径中时，选择性转发攻击一般最有效。

3．Wormhole攻击

在Wormhole攻击中，常见的攻击形式是存在2个相距很远的正常节点，恶意节点通过特定的方式虚报它们之间的距离。位于基站附近的攻击者通过适当设置的Wormhole可以完全破坏路由。攻击者通过Wormhole可以使基站正常为多跳的节点相信它们到基站只有一两跳的距离。这样可以产生一个虫洞，因为Wormhole另一方的攻击者可以人为地提供到基站的高质量路由，如果其他路由没有明显的吸引力则周围区域所有可能的通信将被引诱到这条路由上来。本文的目标就是检测出Wormhole攻击，并且定位虫洞节点的位置。

2.2.2 无线传感器网络安全防范对策

从上文可以看出，无线传感器网络的安全形势十分严峻，因此，世界上也涌现出了一大批算法来抵御这些攻击。因本文主要是应对虫洞攻击，因此，最后这里展示一些目前业界对于如何防御虫洞攻击的成果。

首先我们需要明确的是这种攻击方法非常难于检测，因为它用于传递信息的路径通常不是实际网络的一部分，同时它还特别危险，因为它们能够在不知道使用的协议或网络提供的服务的情况下进行破坏，研究者都提出了许多的方案。下面依次介绍。

（1）基于地理位置

GEAR（Geography hic and Energe Aware Routing）是基于地理位置的路由协议[9]。利用GPS（Global Positioning System）或GNSS（Global Navigation Satellite Systems）等全球定位系统对无线节点的位置进行确定，并根据节点的具体坐标计算出节点间的相对距离，如果节点间的距离大于节点的传输距离，则可以断定存在恶意节点。另外在根据地理位置信息的基础上，另外还衍生出端到端的虫洞检测方法[10]。

（2）基于同步时钟

Yih Chun Hu等人[11]提出了一种称为“数据包限制”（packet leashes）的机制，所有网络节点必须要具有严格同步的时钟。并采用一种有效的认证协议TIK来检测并预防冲动攻击目的节点。目的节点可以根据接受时间和发送时间检测数据包传输的距离是否太长，同样也可以在数据包内设置一个失效时间，超过这个时间，接受者应该收不到数据包。如果数据包传输距离过长或超过数据包中定义的实效时间，则认为存在虫洞，但此方案增加了计算与通信开销，也需要较大的存储空间。

（3）基于监听与信赖

这种方法[12]首先设定网络节点天线都是全方向的，并支持混杂模式，这样源节点在发出数据包之后可以对邻居节点进行监听，也就是可以监听邻居节点转发数据包的时间，根据转发时间可以对邻居节点进行信任评估，在源节点发送数据包之前，根据邻居节点的信任值进行选择，如果邻居节点是虫洞攻击的恶意节点，那它转发数据包不可能被源节点监听到，也就是说恶意节点的信任值是最低的，从而不会被选择到，这样自然就孤立了恶意节点，也避免了虫洞攻击的危害。

（4）RTT

Jane Zhen和Sampalli Srin ivas[13]使用了一种称之为循环旅行时间（RTT）的方法来检测虫洞。A节点计算与节点B之间的RTT，通过发送一个消息给B，要求立刻回复。A，B间的RTT就是从A发送请求到接收到B的回复所经历的时间。每个节点都计算与邻居节点间的RTT，因为2个假冒邻居间的RTT必然大于真正邻居间的RTT。因此通过比较A和A的邻居间的RTT，A节点可以确定哪个邻居是假冒邻居。这种方法不需要额外的硬件并容易实现，但是对于显式虫洞攻击的检测是无效的。

（5）使用加密算法

基于加密算法的安全策略，比如：SAODV、SEAD、SAR。这些安全路由协议可以提供较完善的路由安全保障，但是这些策略需要在路由协议中加入复杂的加解密算法和对应的协议，并且需要节点间具有很强的协作关系。这显然不适用于一般商用性和自发性、临时性的自组织网络。该类网络所面临的安全威胁（比如大规模的协同攻击）和需要达到的安全标准比较低，采用上述几种策略的代价太高。

（6）使用RF（射频）水印

节点间通过特有的方法改变RF的波形，来认证无线传输，防止恶意节点的接入。

（7）SECTOR

SECTOR[14]方法使用特殊的能接受单个二进制位的硬件收发器来计算通信双方的精确距离，接收节点可以据此确定所收的包是由合法路径传输而来还是由恶意节点通过隧道传输而来。由于此方案需要安装特殊的硬件设备，所以限制了它的适用范围。

（8）基于统计分析的方法

SAM协议[15]是由分裂路由协议（SMR）改进而来，使用统计分析的方法进行虫洞检测，这种方法在多路路由协议的环境下使用，每次检测前，当前节点发起路由发现过程，将所得到的所有路由经过的链路进行统计，因为在虫洞攻击下，恶意节点所在的链接将在路由表中出现的比例很高，所以此方法统计处出现比例最高的链接，并使用探测包对它进行检测，由此确定恶意节点。

另外还有一种基于统计分析的检测方法，称之为邻居数量检测方法[16]，这种方法是基于恶意节点周围的邻居数量将增加这个简单的假设之上的，基站将得到所有节点的邻居信息，计算邻居数量分布的估计值，并且使用统计测试来决定是否存在虫洞。

除了上述8种方法外，还有基于定向天线的方法[17]，DELPHI方法等等，我们就不一一赘述了。本文在这些方法之外，另外提出了一种新的虫洞检测方法，具体参考第三章。

2.3 本章小结

本章主要介绍无线传感器网络的基础知识以及无线传感器网络安全领域的最新进展，在此之上，我们将在第三章提出新的算法。

第三章 算法测试系统设计

为了测试本文新提出的虫洞检测算法，需要首先模拟出一个能够插入虫洞的无线传感器网络。因此本章首先对软件生命周期中的需求分析及系统设计做了重点阐述，通过对系统框架的构成来认识各技术功能模块，对系统的主要功能模块给出了详细的介绍。对整个测试系统的设计进行阐述之后，再对如何判断算法的有效性、实用性、可靠性进行阐述。

3.1 系统需求分析

需求分析包括需求的获取、分析、规格说明、变更、验证、管理等一系列需求工程。软件人员与用户共同讨论决定，哪些需求是可以满足的，并且加以确切地描述。它是在可行性研究的基础上进行的更细致的分析工作，是软件定义时期的最后一个阶段对软件目标及范围的求精和细化。通过可行性研究和分析，充分了解用户对软件系统的要求，把用户要求表达出来，解决“软件系统必须做什么”的问题。它主要包括三个层次的需求：业务需求、用户需求、功能需求。

本次设计要完成一个能够测试虫洞检测算法有效性、实用性、可靠性的系统。因此我们要求这个系统能模拟无线传感器网络的正常运行及基本功能，例如路由转发、邻居发现等。在此之上，还要求该系统能够支持插入虫洞节点，要求虫洞节点能够模拟正常虫洞节点可以完成的功能，包括对包的窃取、篡改等。最后要使得这套系统能适应我们提出的算法。

从需求出发，本系统的处理场景应该是这样的：每一个传感器节点定期的搜集环境信息，根据路由路径将包传回给汇聚节点，汇聚节点进行相应的处理。当用户输入某个信息之后，开始植入虫洞节点。当虫洞节点植入完毕之后，在虫洞节点周围的流量包理应被大量的吸收，我们可以在虫洞节点上面对数据包进行相应的修改，来反应虫洞节点的植入成功。之后，汇聚节点应该能够在一定的时间内检测到虫洞节点，并且能够打印出虫洞节点两端的节点的ID号。

3.2 系统设计

软件设计是软件工程的技术核心。设计人员应该建立一个与确定的各项需求响应的体系结构，这个结构保证每一部分都有一个明确的一一，针对需求的模块组成，并对每一个模块进行工作量描述。所有设计中的考虑以设计说明书的形式加以描述，以供后续工作使用并评审。

软件软件主要包括概要设计和详细设计两部分。概要设计的主要任务是根据用户需求分析阶段得到的目标系统的物理模型，确定一个合理的软件系统的体系结构。详细设计则是进行具体细化的设计，分为算法设计、数据结构设计、物理设计、其他设计（如代码设计、输入/输出格式设计、人机对话设计等）、编写详细设计说明书、评审等。

3.2.1 系统使用说明

根据概要设计，传感器网络虫洞检测测试软件的的主要功能描述如下：

用户首先根据需求配置节点数目，然后可以选择节点是随机分布，还是定点分布，如果是定点分布，需要提供相应的每个节点所在的位置。然后点击开始运行，程序开始执行，所有的节点在各自的后台线程中执行，首先执行邻居发现算法，之后执行路由算法，查找到汇聚节点（Sink Node）的路由路径，在此之后，屏幕上将会打印出汇聚节点对每个包的处理过程，以及包的内容。当收集到一定数目的环境数据之后，用户通过键入任何的输入，通知主线程植入虫洞节点。主线程在接收到命令后开始执行虫洞节点植入算法，开始植入虫洞。当虫洞完成基本的初始化任务后，用户应该可以在屏幕上看到虫洞节点对包的处理过程，以及吸收到的流量。这个时候用户已经不再需要任何输入，程序会自动对网络状况进行评估，一旦检测到虫洞节点就会退出程序，并打印出虫洞节点的位置及信息，如果没有发现虫洞节点，则这个网络将会继续执行下去。

3.2.2 类设计

类是面向对象程序设计实现信息封装的基础，是一种数据类型，每个类包含数据说明和一组操作数据或传递消息的函数。用类来实现这个系统，体现了面向对象的思想，同时提高了模块之间的内聚性，降低了模块之间的耦合性，使得每个模块的可复用性得到了极大的提高。

根据概要设计，我将这个系统划分为几个大方向，首先是节点类，节点类是本程序的核心，每一个对象都代表着一个虚拟的网络节点。因此节点类的设计非常重要。其次是节点支持类，主要包含为了支持节点完成冲动检测、路由建立等的辅助类。最后是网络类，通过网络类，程序不需要直接调用底层的节点类来构造网络，只需要传递参数给网络类，网络类会自动构造一个网络所需要的节点，这可以极大的方便用户使用。

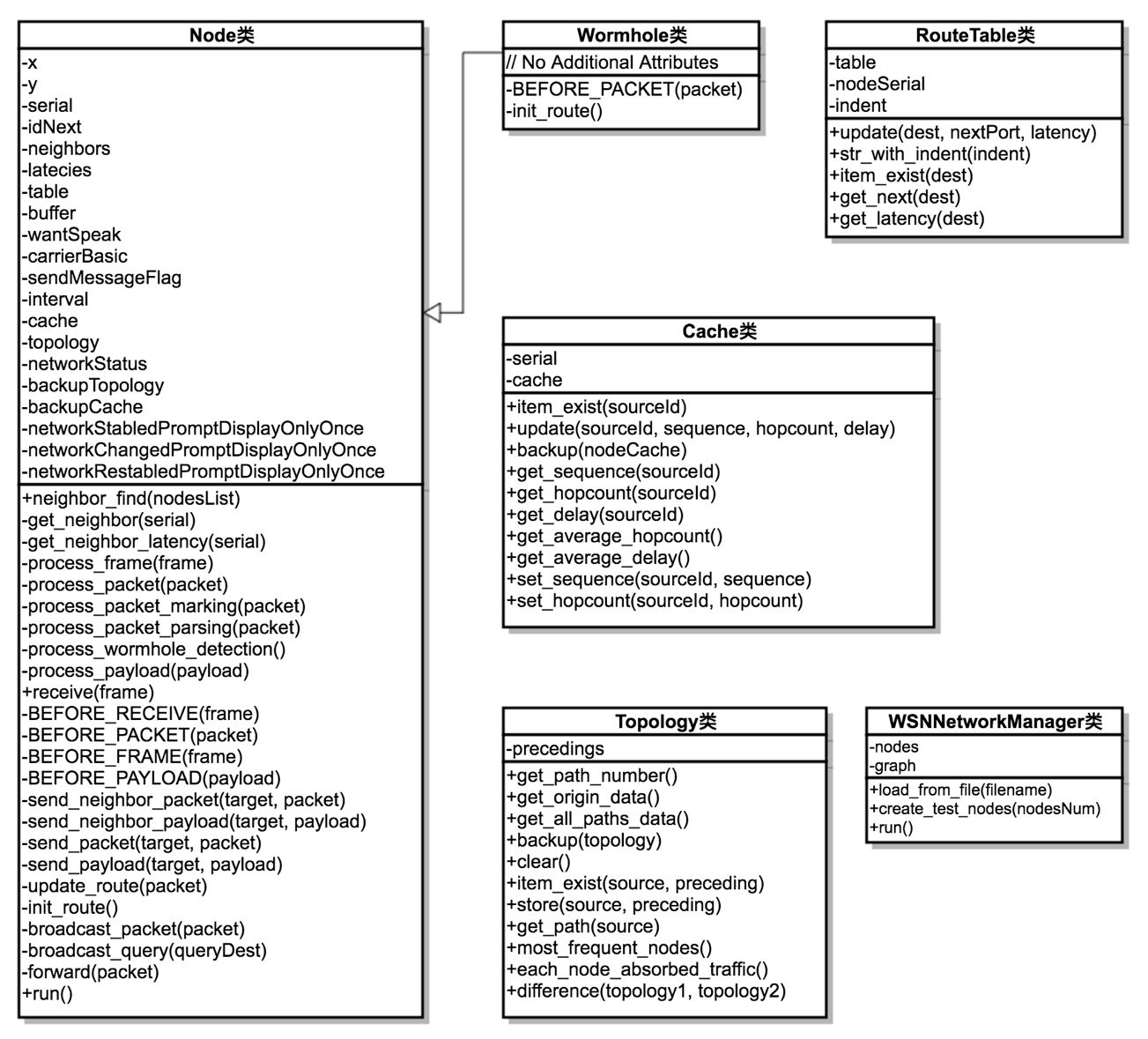
具体来说，节点包括了Node类、Wormhole类。节点辅助类包括NodeCache类、RouteTable类、Topology类。网络类包括Network类、NetworkTest类。类图见图3-2。下面详细讲解各个类的功能及作用。具体每个节点的功能实现以及每个成员及函数的作用请参考第四章。

图3-1 系统UML图

如图3-1，Node类占据了大量的篇幅，因为这个类是本系统的核心，它主要的功能有数据采集、数据发送。

Wormhole继承自Node类，它重写了init\_route方法，主要是为了更改路由设置，通知整个网络此节点存在一条高速通道。BEFORE\_PACKET用于虫洞节点对数据包的篡改、窃听等操作。

RouteTable类实现了一个简单的路由表，为节点访问路由表项、查找路由路径提供了极大的便利，同时实现了一个str\_with\_indent()方法，可以得到具有一定空格的路由表字符串，这样打印出来的路由表会显得更加的美观。

Cache类及Topology类是为了汇聚节点进行虫洞检测而专门设计的类，他们的功能分别是：Cache类提供了存储每个节点的跳数、包的序列号的表；Topology类提供了访问整个网络拓扑图的方法，可以让汇聚节点很方便的访问、查找拓扑路径。

WSNNetworkManager类是为了方便网络的管理而设计的类，可以读取文件中关于节点的信息来建立网络，也可以建立一个小型的节点网络，这在前期主要是为了方便测试路由算法等。用户只需要在建立了一定规模的网络后，运行run()方法，就可以看到网络数据的输出。

3.3 系统主要算法及架构解释

3.3.1 网络设计

本系统参考了TCP/IP的网络实现，这主要表现在通信协议的分层设计上，同时还借鉴了Linux对于网络包处理方面的Netfilter/IPTable的内容来方便虫洞节点的编写。

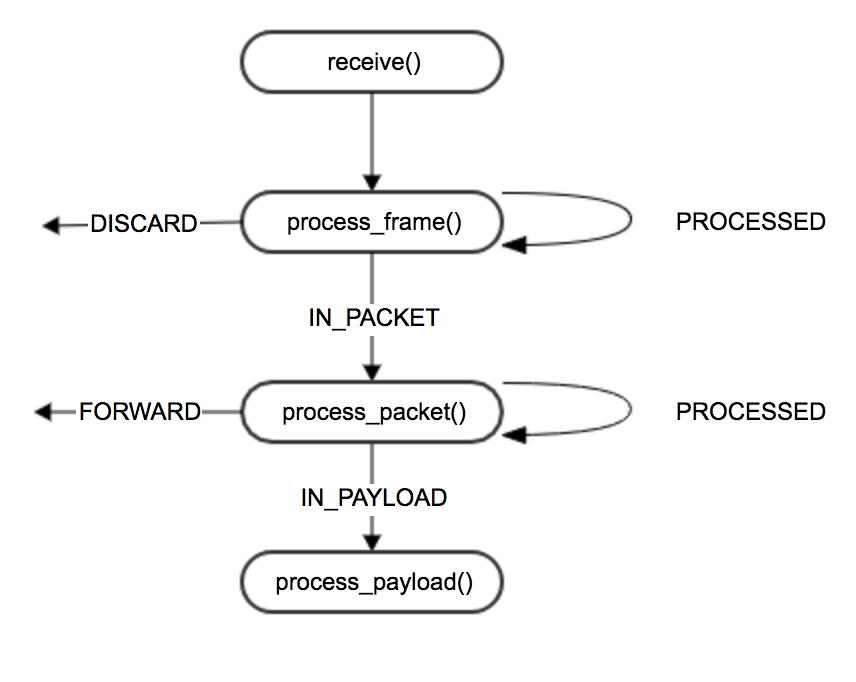
 整体的设计见图3-3。

图3-2 一个节点对一个帧的完整处理过程

如图所示，一个节点收到一个帧之后，首先会交由process\_frame()进行处理，这里这个函数充当网络的最底层，类比TCP/IP协议中的数据链路层。然后根据返回值决定这一帧是丢弃，还是进入下一层。

如果帧是完好无损的，并且不是控制帧，则会进入process\_packet()，类似TCP/IP中的网络层。在这一层会首先判断包的目的ID，如果该包的目的节点不是本节点，则会被转发，如果是本节点则根据包的类型来进行处理，如果包是控制包，那么这个包就将会在本层进行处理，如果这个包只是简单的数据包，则会进入下一层——应用层。

process\_payload()函数代表了整个网络的应用层，在这一层我们可以执行对包实际携带的数据进行操作。但是考虑到WSN的实际应用场景，我们只在汇聚节点对payload进行打印，其余的节点不进行任何处理。

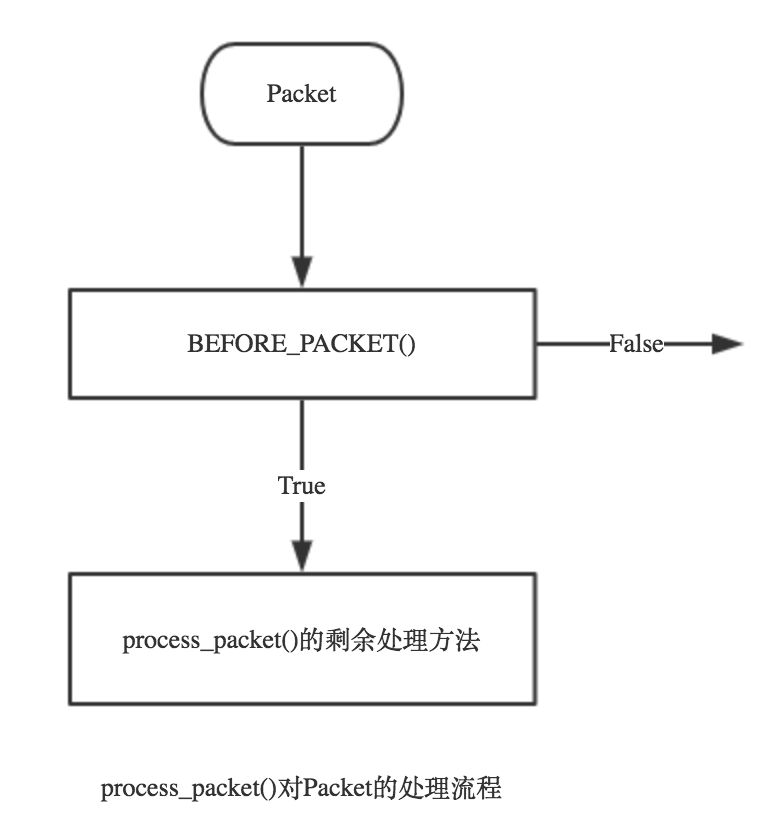
 另外还需注意的是，为了更好、更方便的实现虫洞节点的功能，我们仿照Netfilter/IPTable设计了一套插桩系统，见图3-3。

图3-3 process\_packet方法的具体处理过程

如图，在process\_packet()方法的前部我们插入了一个新的方法BEFORE\_PACKET()，这样wormhole在需要完成虫洞功能的时候，主要重写这个方法就可以了，而不必去更改process\_packet方法，因为这个方法里面还设计到对路由表的修改，以及其他网络控制包的处理。除了process\_packet()方法之外，我还在process\_frame()，process\_payload()，receive()方法前面分别编写了BEFORE\_FRAME()，BEFORE\_PAYLOAD()，BEFORE\_RECEIVE()这三个方法。这样分离之后，就使得整个网络的设计更加的清晰明了。

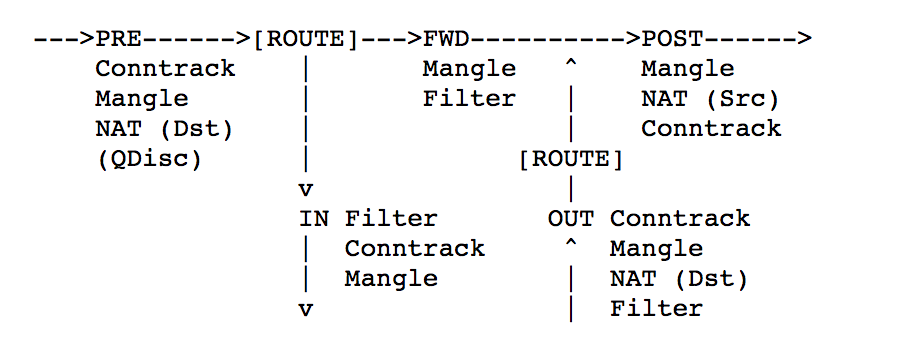
而图3-4就是我设计就这套系统思想的来源。如图Linux在收到一个包之后首先来到PRE的处理处，在这里要依次经过Conntrack、Mangle、NAT三个表项，每一个表项就是一个插桩点，我们可以设置相应的处理函数来处理包。

图3-4 Linux下Netfilter/IPTable结构图

3.3.2 路由算法的设计

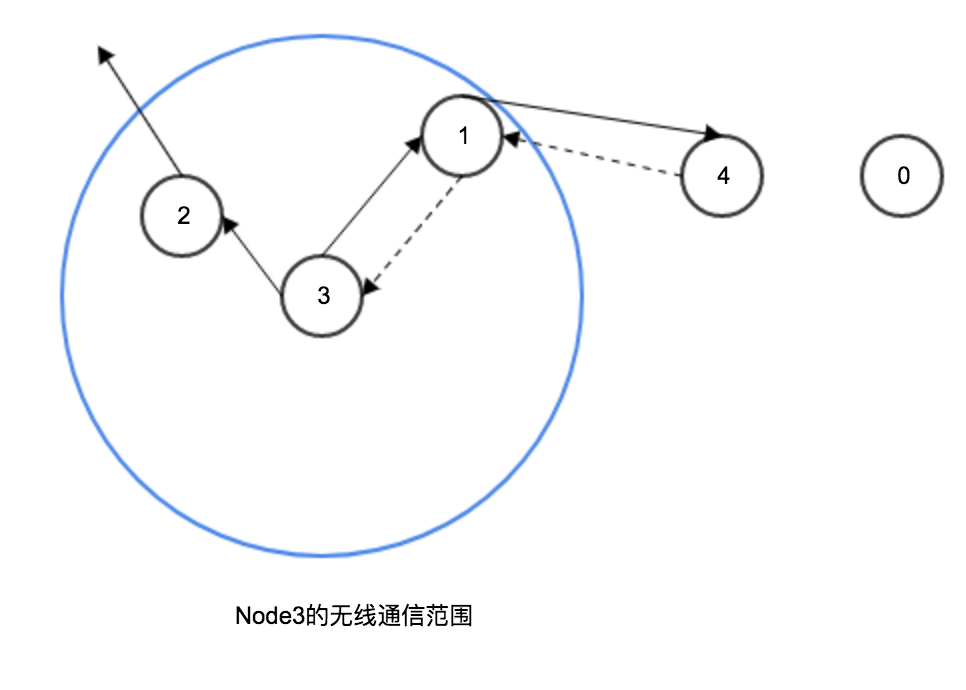
 路由算法，我们采用了广播查询的方法。示意图见示意图3-5.

图3-5 路由算法的示意图

如图，节点0为汇聚节点。节点3希望向汇聚节点传输信息，但是由于路由表中，没有关于汇聚节点的表项，因此，它向能直接通信的所有节点发送路由查询包，节点1和节点2收到包后，由于也不知晓汇聚节点的位置，于是再向周围广播路由查询。节点4收到路由查询包之后，发现在路由表项中有关于汇聚节点的下一跳节点位置（因为节点4在汇聚节点的直接通信范围），于是将传回一个路由响应包。这个包包含了到汇聚节点的下一跳地址（在本例中是4），这个响应包在被1号节点收到后，将会查询自己的表项，是否需要更新（是否存在路径，或者路径更短），如果自己的表项得到更新，则修改路由响应包的下一跳地址为1，并广播这个新的路由包。如果节点的表项没有更新，则对此路由响应包不再做任何处理，也不会再进行相应的修改与转发。

通过这个算法，一定时间之后，网络中的所有节点就会知晓关于汇聚节点的下一跳路径，并且每个节点到汇聚节点的路径将会是最短路径。因此也就建立起了整个网络的路由系统。

3.3.3 虫洞节点发现算法-四状态法

根据前文所述，现有的算法要么需要特殊的硬件支持，要么对整个网络的提出了较高的要求。比如一些利用地理位置信息的算法需要相应的硬件，如GPS，方向天线。这类方法提高了硬件的成本、能量和计算处理开销。另外一些对网络提出较高要求的算法，比如全节点统一时钟，特殊的定位节点，特殊的检测事件或者网络探测算法。这些要求使得他们的适用性不强，难以在各种网络环境中应用，除此之外在这些算法中只有少部分能够检测出虫洞的位置。因此，我提出了一种新的算法，这种算法只需要在现有的包的结构上进行非常微小的修改，并且只会对中间节点造成很少的能量、运算消耗。

这个算法的核心基于观察到的这样一种现象：每当一个网络插入了虫洞节点后必然会造成整个网络的状态的改编，并且这种改变一定是跳数减少或延时减少，或二者兼而有之。这是因为虫洞节点的本质在于创造一个高速连接，因此原先一些需要多跳转发才能将数据包传递给汇聚节点的节点，他们所发送的数据包会受到“吸引”。通过这种插入节点前后的网络状态的比较，我们就可以判断网络中是否出现了虫洞节点。同时根据我们的观察，在正常网络情况下，每一个节点出现在路由路径中的频率不会过多，而虫洞节点之所以能吸收巨额的流量，必然是因为在许多条路由路径中反复出现。而虫洞节点在路由路径中的反复出现，必然也导致与其相连的两个端点在所有的路由路径中多次出现。因此通过比较前后的拓扑图，我们就能判定节点的逻辑位置。

基于此，我提出这样一种算法：假设，在架设无线传感器网络的时候并不会有虫洞节点的植入，中间节点通过标记包的方式来将自己下一跳、再下一跳的路由信息传递给汇聚节点，当汇聚节点收集到一定的路由信息后，就可以形成一个关于网络的拓扑图。在经过一段时间后，这个拓扑图就会稳定下来，于是汇聚节点将其稳定后的拓扑图保存下来。在此之后整个网络就具备了检测、定位虫洞节点的能力了。我们假设有虫洞节点在这个时候植入，那么必然会带来路由路径的变化。汇聚节点应该能侦测到这一变化，并且记录下来。当路由路径再次稳定下来之时，我们比较原先备份的拓扑图与现有的拓扑图，并且计算平均跳数、平均延时的变化，就可以以一定的概率去判断是否存在虫洞节点。

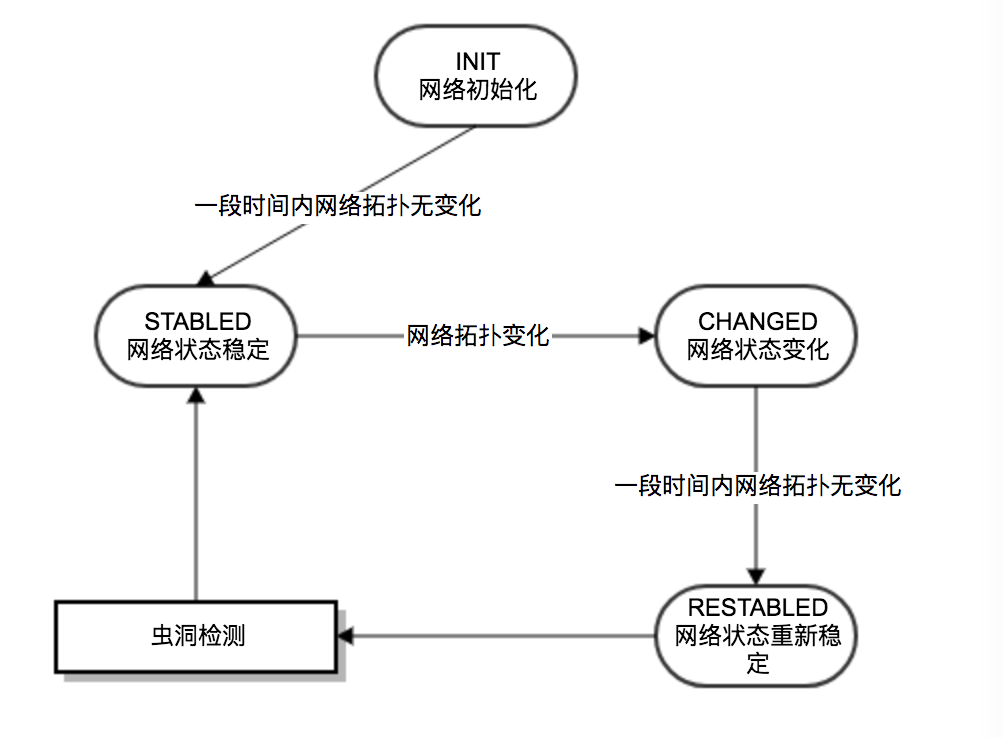
 这里，我需要指出的是，正常网络环境的变化，同样会带来拓扑图的变化，但是这种变化应该是有限的。因此这种算法是一种概率性的算法，存在一定的误报率，但是，通过不断观察虫洞植入和正常网络变化带来的跳数变化、延时的变化，我们可以大致得出一个阈值，这个阈值就是正常网络变化与虫洞节点植入的分界点。然后我们可以通过这个阈值来进行虫洞检测，从而大幅度的提高判断的准确性，降低误报的可能性。

图3-6 汇聚节点的状态转换图及虫洞检测发生的时机

由于汇聚节点需要判断什么时候网络已经稳定下来，什么时候网络开始有波动，因此需要一个变量来表示网络的状态。显而易见，网络应具有初始化、稳定、网络波动、网络再稳定四种状态，因此，这种算法被我称之为四状态法，而具体的状态转换图见图3-6。

从图中，我们可以看到虫洞检测应该发生在网络状态再次稳定之后，因此具有一定的延迟，也就是说这种算法不具有实时性，这是此算法的一个重要缺陷。

3.5 本章小结

本章介绍了测试系统的使用方法以及测试系统的整体架构、类的设计以及关键算法的核心思想，下一章将详细描述每一个类的成员及其函数的作用，以及关键算法的具体实现。

第四章 测试系统功能实现

4.1开发平台

开发环境：macOS Sierra 10.12.5

开发语言：Python 2.7

支持运行环境：Windows、Linux/Unix、Mac OSX等

4.2 Node类的实现

4.2.1 Node类成员的说明

x：节点物理位置的X坐标。

y：节点物理位置的Y坐标。基于节点的物理位置我们可以确定节点的邻居，本次试验中，我们设置的直接通信距离为10。因此如果两个节点的笛卡尔距离小于10，则表示他们是邻居，也表示他们可以直接通信。

serial：节点的唯一标识ID。

idNext：节点所产生的下一个包的包序列号。

neighbors：节点可以直接通信的邻居节点的对象表。

latencies：节点邻居节点的延时表，这个表将在后面用来初始化路由表，当完成路由表的初始化之后这个结构将被废弃。

table：节点的路由表，是一个RouteTable对象。通过对这个对象函数的调用，可以获取诸如下一跳、延时等信息。

buffer：节点的网络缓冲区，其他节点向本节点发送帧，最终会被存储在这里。

wantSpeak：用于模拟载波侦听的数据结构，表示本节点希望能够“发言”。同时如果接受到其他节点发出的希望“发言”的帧，根据这个结构决定如何进行相应。

carrierBasic：相当于物理层数据报，也就是帧的idNext，值得注意的是这个结构仅仅用来实现模拟载波侦听。

sendMessageFlag：一个标志信号，表示节点当前可以向汇聚节点发送信息，主要用来对每个节点的发送时机进行控制，避免太多的数据包涌入网络，造成汇聚节点打印出现延时。

interval：启动延时，让每个节点启动的时机略微有些不同，来模拟网络的布置。

cache：汇聚节点用来收集网络运行状况（每个节点的跳数，每个节点的延时）的数据结构，是一个NodeCache对象。

topology：汇聚节点用来收集、管理网络拓扑结构的数据结构，是一个Topology对象。

networkStatus：汇聚节点存储当前网络状态的结构，可选值有NETWORK\_INIT、NETWORK\_STABLED、NETWORK\_CHANGED、NETWORK\_RESTABLED。

networkStabledPrompotDisplayOnlyOnce：控制汇聚节点在网络稳定后只打印一次提示信息。

networkChangedPromptDisplayOnlyOnce：控制汇聚节点在网络变化后只打印一次提示信息。

networkRestabledPromptDisplayOnlyOnce：控制汇聚节点在网络再次稳定后仅打印一次提示信息。

4.2.2 Node类函数的说明

neighbor\_find(nodesList)：通过遍历整个网络的节点，来确定邻居节点，并存储到相应的数据结构。

get\_neighbor(serial)：根据邻居节点的ID查找邻居节点的对象。

get\_neighbor\_latency(serial)：根据邻居节点的ID查找邻居节点与当前节点的延时。

process\_frame(frame)：节点在这里进行帧的处理，返回一个常量表示当前帧是否已经被处理，如果没有应该如何继续处理。返回值有FRAME\_IN\_PACKET、FRAME\_DISCARD、FRAME\_HAS\_PROCESSED。如果帧已经被破坏返回FRAME\_DISCARD，表示帧已经被丢弃；如果帧已经被处理（控制帧），则返回FRAME\_HAS\_PROCESSED；如果帧需要进入下一层网络结构（网络层）返回FRAME\_IN\_PACKET。

process\_packet(packet)：节点在这里进行网络层数据结构--包的处理。返回一个常量表示当前包是否已经被处理，如果没有应该如何继续处理。常量有PACKET\_IN\_PAYLOAD、PACKET\_FORWARD、PACKET\_HAS\_PROCESSED。如果当前数据包的目的地址不是本节点则返回PACKET\_FORWARD表示当前包需要被转发；如果当前数据包是控制包（如路由查询包，路由响应包）则返回PACKET\_HAS\_PROCESSED表示该包已经在该函数内被处理；如果数据包仅仅是携带载荷的有效数据包，则返回PACKET\_IN\_PAYLOAD，表示包将在应用层进行下一步的处理。

process\_packet\_marking(packet)：对当前包进行拓扑标记，使得汇聚节点能够进行路由信息的收集，然后建立关于网络的拓扑信息。

process\_packet\_parsing(packet)：只有汇聚节点调用这个方法，利用这个方法更新对应的cache、topo结构，更新网络状态，决定是否进行虫洞检测。

process\_wormhole\_detection()：只有汇聚节点调用这个方法，利用这个方法进行虫洞检测。

process\_payload(payload)：节点在这里进行载荷也就是应用层数据的处理，在本次实验中，我们仅仅利用汇聚节点在这一层对载荷进行打印。

receive(frame)：其他节点调用这个公开的接口，来模拟物理层的交付。

BEFORE\_RECEIVE(frame)、BEFORE\_FRAME(frame)、BEFORE\_PACKET(pa-

cket)、BEFORE\_PAYLOAD(payload)：分别是在receive方法、process\_frame方法、process\_packet方法、process\_payload方法执行前的一个插桩点，模拟Linux下的Netfilter/IPTable的功能，在本次实验中主要是为了方便虫洞节点在尽量小的改动下完成相应的功能。

send\_neighbor\_packet(target, packet)：向邻居节点发送包，这里会有模拟载波侦听。

send\_neighbor\_payload(target, payload)：向邻居节点发送载荷，这里只是简单的对send\_neighbor\_packet方法的一个封装。

send\_packet(packet)：向节点发送包，这里可以对汇聚节点进行发包，因为在这里会有路由判定，获取下一跳地址，然后调用上述的向邻居节点发包的函数进行发包。

send\_payload(target, payload)：功能同上send\_packet方法，不过是另一种封装形式。

update\_route(packet)：如果packet是路由响应包，则根据这个包的内容进行路由表的更新。

init\_route(packet)：根据neighbors和latencies首先更新路由表，然后如果当前表中没有关于汇聚节点的路由信息，则发送一个路由查询包。

broadcast\_packet(packet)：向邻居节点广播一个包。

broadcast\_query(queryDest)：向邻居节点广播一个路由查询包。查询的地址是参数queryDest。

forward(packet)：对包进行路由转发。

run()：Node的主方法，封装了所有的功能。

4.2.3 Node类重要方法的实现详解

4.2.3.1 Node节点对路由响应包的处理

Node节点在收到路由响应包之后，首先执行update\_table更新路由表，这个函数返回一个bool型变量，判断当前包是否造成了路由表的更新，如果有的话，那么节点则重新产生这个路由响应包，并将其广播给邻居节点。具体的实现如下所示。

res = Packet.test\_RR\_packet(packet)  
**if** res **is not** None:  
 **if** self.update\_route(packet):  
 latency = res['latency']  
 dest = res['dest']  
 nextPort = self.serial  
  
 # broadcast the update of the route  
 **for** each **in** self.neighbors:  
 each.receive(Frame(self.serial, each.serial,  
 Packet.generate\_RR\_packet(self.serial,  
 each.serial,  
 dest,  
 self.idNext,  
 latency + self.table.get\_latency(each.serial),  
 nextPort,  
 None)), self.get\_neighbor\_latency(each.serial))  
  
 self.idNext += 1

4.2.3.2 中间节点对包的标记过程

中间节点对于虫洞检测的额外的处理是非常小的，每个包首先被转发到的第一个节点标记markId，然后被转发到的第二个人标记preceding。每一次标记都是简单的将本节点的ID填写到对象的包的数据中去。当汇聚节点接收到这个包就有了三个信息，一个是包的源节点source，包的下一路由点markId，再下一级路由点preceding。具体实现如下所示

**def process\_packet\_marking**(self, packet):  
 # simplify version, designed by p1usj4de  
 **if not** packet.test\_mark\_id():  
 packet.mark\_id(self.serial)  
 **elif not** packet.test\_preceding():  
 packet.mark\_preceding(self.serial) **return**

4.3 Wormhole类的实现

Wormhole类是Node类的子类，基本的功能如路由转发等已经实现，因此不需要再写很多的代码，关键在于透过直接修改与其相连两个节点的数据结构来达到改变网络路由路径，最终吸收流量的目的

4.3.1 Wormhole类成员的说明

oneEnd、anotherEnd：虫洞连接点的两端节点。

linkConsume：虫洞节点的延时。在本例中我们设置两端都是0.01。

4.3.2 Wormhole类函数的说明

BEFORE\_PACKET(packet)：对Node的BEFORE\_PACKET方法的重写，这里就体现了本系统设计的优点，比照Linux的Netfilter/IPTable设计了插桩点，因此Wormhole类的编写异常简单。

init\_route()：重写了Node的init\_route方法。

4.3.3 Wormhole重要方法的实现详解

4.3.3.1 重写init\_route()方法的实现

虫洞节点将透过重写init\_route方法来达到修改路由路径的目的，也就是说达到建立高速连接的目的。虫洞节点首先更新自己的路由表，然后手动修改两端节点的路由表，并且将自己加入两端节点的neighbors表中，再让两端节点的latencies表中存入linkConsume，最后，如果虫洞节点本身没有关于汇聚节点的路由信息，则广播查询，如果有的话，则广播一则路由响应包，通告整个网络自己的高速连接。具体的实现如下所示。

def init\_route(self):

# add myself port

self.table.update(self.serial, self.serial, 0.0)

# modify my route table

self.table.update(self.oneEnd.serial, self.oneEnd.serial, self.linkConsume)

self.table.update(self.anotherEnd.serial, self.anotherEnd.serial, self.linkConsume)

# modify the ends's route table

self.oneEnd.table.update(self.serial, self.serial, self.linkConsume)

self.anotherEnd.table.update(self.serial, self.serial, self.linkConsume)

# add myself to two ends's neighbors

self.oneEnd.neighbors.append(self)

self.anotherEnd.neighbors.append(self)

self.oneEnd.latencies.append(self.linkConsume)

self.anotherEnd.latencies.append(self.linkConsume)

# add two ends to my neighbors

self.neighbors.append(self.oneEnd)

self.neighbors.append(self.anotherEnd)

self.latencies.append(self.linkConsume)

self.latencies.append(self.linkConsume)

# send packets to modify the route path

#self.oneEnd.receive(Frame(self.serial,

self.oneEnd.serial,

Packet.generate\_RR\_packet(self.serial,

self.oneEnd.serial,

self.oneEnd.serial,

self.idNext,

self.linkConsume, self.serial, 0)))

#self.anotherEnd.receive(Frame(self.serial,

self.anotherEnd.serial,

Packet.generate\_RR\_packet(self.serial,

self.anotherEnd.serial,

self.anotherEnd.serial,

self.idNext,

self.linkConsume,

self.serial, 0)))

# send a packet to query the sink node's route path

if not self.table.item\_exist(SINK\_NODE\_ID):

self.broadcast\_query(SINK\_NODE\_ID)

else:

notSinkNode = self.oneEnd if self.oneEnd.serial != SINK\_NODE\_ID

else self.anotherEnd

sinkNode = self.oneEnd if self.oneEnd.serial == SINK\_NODE\_ID

else self.anotherEnd

self.unitTestAfterInesertWormhole = True

notSinkNode.receive(Frame(self.serial,

notSinkNode.serial,

Packet.generate\_RR\_packet(self.serial,

notSinkNode.serial,

SINK\_NODE\_ID,

self.idNext,

self.linkConsume,

self.serial, 0)),

self.get\_neighbor\_latency(notSinkNode.serial))

self.idNext += 1

4.4 RouteTable类的实现

RouteTable类是一个非常关键的类，透过这个类，节点可以非常方便的实现路由表的查询、更新等操作。

4.4.1 RouteTable类成员的说明

table：table变量是一个字典型变量，键值是目的地址，返回值是一个元组对象，由两个成员构成，分别是下一跳序号及延时。

nodeSerial：表示存储该表的节点的序列号，为了能在打印路由表中附上节点的序号，使显示结果显得更加清楚而加入这个域。从设计的角度上来讲，这个域应该是不被需要的。

indent：打印的首部空格数，可以产生缩进的效果，实现更美观的路由表结果显示。

4.4.2 RouteTable类函数的说明

update(dest, nextPort, latency)：更新路由表，参数分别是目的地址，下一跳地址，延时。如果表得到了更新返回True，如果没有（表项已经存在或者存在更短的路径）则返回False。

str\_with\_indent(indet)：返回一个可打印的路由表字符串序列，带首部缩进，缩进长度为indet。

item\_exist(dest)：查看表中是否存在关于目标dest的表项。

get\_next(dest)：获取到达目的地址dest的下一跳序列号。

get\_latency(dest)：获取到达目的地址dest的下一跳的延时。

4.4.3 RouteTable重要方法的实现详解

4.4.3.1 update方法的实现

函数首先判断表中是否已经存在关于dest的信息，如果没有则直接添加，如果有的话，比较该条路线与已有路线的延时，如果该条路线延时更短，则更新表项，否则不做任何处理。如果表项得到更新返回True，否则返回False。函数的具体实现如下所示。

**def update**(self, dest, nextPort, latency):  
 latency = float(latency)  
 dest = str(dest)  
 **if** self.table.has\_key(dest) **and** latency < self.table[dest][1]:  
 self.table[dest] = (nextPort, latency)  
 **return** True  
 **elif not** self.table.has\_key(dest):  
 self.table[dest] = (nextPort, latency)  
 **return** True  
 **else**:  
 **return** False

4.5 Cache类的实现

Cache类是实现虫洞检测的基础之一。本系统采用的虫洞检测算法利用的是跳数减少、延时减少来概率性判断是否存在节点，因此需要一个数据结构来保存前后的跳数和延时信息，Cache这个类的设计就是为了完成这个目标。

4.5.1 Cache类成员的说明

serial：同RouteTable的nodeSerial目的一样，都是保存节点的ID号来实现更加精致的表的打印字符串。

cache：一个字典型结构，键值是节点的ID号，存储的值是一个包含三个元素的列表，分别表示包序列号、跳数、延时。

4.5.2 Cache类函数的说明

itemExist(sourceId)：判断源节点为sourceId的表项是否存在，如果存在返回True，否则返回False。

update(sourceId, sequence, hopcount, delay)：更新源节点为sourceId的表项，使得序号为sequence，跳数为hopcount，延时为delay，如果表中不存在这个项，则新建此项。

backup(nodeCache)：将本节点的Cache内容备份到nodeCache中去。

get\_sequence(sourceId)：获得源节点为sourceId的包序号，如果不存在这个项，则返回None。

get\_hopcount(sourceId)：获得源节点为sourceId的跳数，如果不存在这个项，则返回None。

get\_delay(sourceId)：获得源节点为sourceId的延时，如果不存在这个项，则返回None。

get\_average\_hopcount()：获得该表中所有项的平均跳数。

get\_average\_delay()：获得该表中所有项的平均延时。

set\_sequence(sourceId, sequence)：设置源节点为sourceId的表项的序号。

set\_hopcount(sourceId, hopcount)：设置源节点为sourceId的表项的跳数。

set\_delay(sourceId, delay)：设置sourceId的表项的延时。

4.5.3 Cache类重要方法的实现详解

4.5.3.1 update方法的实现

在该方法中，首先判断键值为sourceId的项是否存在，如果不存在，首先新建一个值全为0的新项。这里需要注意sequence、hopcount、delay的默认参数都是None，当某一项为None的时候，对该项就不进行更新，这样使得该方法使用起来相当灵活，而后面的三个set方法实际上是对该方法的再次封装。如下是具体的实现。

**def update**(self, sourceId, sequence=None, hopcount=None, delay=None):  
 # if entry not existed, create one  
 **if not** self.itemExist(sourceId):  
 self.cache[str(sourceId)] = [0, 0, 0.0]  
  
 **if** sequence **is not** None:  
 self.cache[str(sourceId)][0] = sequence  
 **if** hopcount **is not** None:  
 self.cache[str(sourceId)][1] = hopcount  
 **if** delay **is not** None:  
 self.cache[str(sourceId)][2] = float(delay)

4.6 WSNNetwork类的实现

前面的几个类，关注的是单个节点的设计，在完成了单个节点的设计后，我们还需要一个类能对整个网络进行管理、配置，并且能提供简单易用的接口，这就是WSNNetwork类的功能。

4.6.1 WSNNetwork类成员的说明

nodes：保存该网络中所有的节点对象。

4.6.2 WSNNetwork类函数的说明

load\_from\_file(filename)：从文件中读取节点的配置信息。文件应保存在系统目录下的NetworkFiles文件夹下。文件格式为每行一个节点，每行需要定义x坐标和y坐标，中间用一个空格隔开。

create\_test\_nodes()：在这里实现了一些测试节点，不需要再调用load\_from\_file方法，通过这个小型的网络，可以很方便地进行测试。

run()：接口函数，使用该类只需要调用这个方法即可。

4.6.3 WSNNetwork类重要方法的实现详解

该方法首先为每一个节点的run方法定义了一个线程，然后设置后台运行后执行。之后主线程等待用户的输入，一旦有输入，不论输入的是什么，就声明一个虫洞节点，并将其植入到网络中去，在该例中，我们选择植入的地点为0号节点和62号节点。在植入完成后再次输入任何字符，就会终止程序，退出所有线程。具体实现如下所示。

**def run**(self):  
 **for** x **in** self.nodes:  
 x.neighbors\_find(self.nodes)  
  
 threads = [threading.Thread(target=Node.run, args=(each,)) **for** each **in** self.nodes]  
 **for** x **in** threads:  
 x.setDaemon(True)  
 x.start()  
  
 # after any input insert a wormhole  
 raw\_input()  
  
 # insert the wormhole node here  
 wormhole = Wormhole(1000, self.nodes[0], self.nodes[62])  
 thread = threading.Thread(target=Wormhole.run, args=(wormhole,))  
 thread.setDaemon(True)  
 thread.start()  
  
 # after any input, shutdown the game and print the new topology with the most frequently appear nodes  
 raw\_input()  
  
 **print** "Exit!"

4.7本章小结

本章主要对虫洞检测软件中出现的所有类进行一个详细介绍，包括成员、函数的说明，还有核心步骤的具体实现。

第五章 系统测试及算法测试

5.1 软件测试

1、软件测试的介绍：

软件测试（Software Testing），是描述一种用来促进鉴定软件的正确性、完整性、安全性和质量的过程。换句话说，软件测试是一种实际输出与预期输出间的审核或者比较过程。软件测试的经典定义是：在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程。

软件测试是每个软件项目必不可少的部分，在软件的生命周期中所占比重在35%以上，有的甚至会更高。软件测试不仅在开发后需要进行，在开发中也要进行测试，测试的目标是发现程序和软件设计及开发过程中存在的各种缺陷。

2、软件测试的目的：

基于不同的立场，存在着两种完全不同的测试目的。从用户的角度出发，希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可以接受该产品；从软件开发者的角度出发，则希望测试成为表明软件产品中不存在错误的过程。验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。

3、软件测试的方法：

经常用到的测试方法有静态测试方法、动态测试方法、回归测试方法以及模拟用户操作测试方法。其中，静态测试是指无须执行被测代码，而是借助专用的软件测试工具评审软件文档或程序，度量程序静态复杂度，检查软件是否符合标准，介意发现编写的程序的不足之处，减少错误出现的概率。动态测试方法则是在相对真实的环境下运行被测代码，从多角度观察程序运行时能体现的功能、逻辑、行为、结构等行为，以发现其中的错误现象。动态测试方法分又为黑盒测试和白盒测试。黑盒测试是基于功能的测试，只关心软件的功能，而不考虑其内部结构，也叫功能测试；白盒测试只关心软件内部逻辑结构，测试覆盖率，是由逻辑驱动的测试。回归测试是为了验证修改以及增加后的部分是否正确，而且不对其他部分造成影响。模拟用户操作测试方法主要是从用户对要使用软件情况的了解来开发测试的方法。

本文主要对虫洞检测系统进行单元测试。测试的主要功能点包括网络的连通性、网络收发包的正确性、网络体系结构处理的完备性。根据系统的功能需求分析，拟定了本系统的测试步骤。再完成对整个测试系统的测试后，我们将对算法的实用性进行测试，并对测试结果进行说明。

5.2 测试步骤

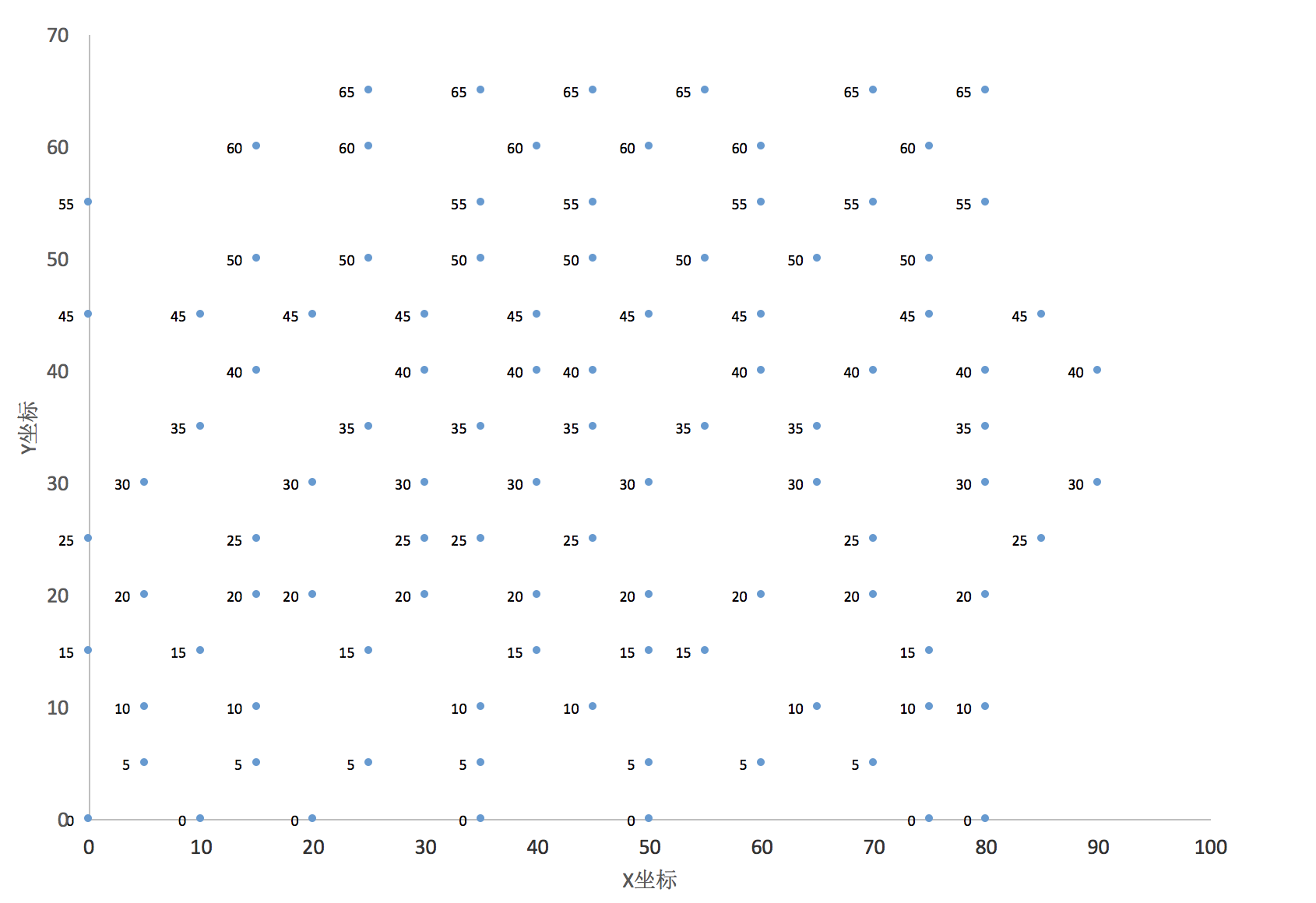
 本系统设计到一个完整的网络系统，在网络由小壮大的过程中难免发生许多BUG，因此从设计之初，就拟定了相应的测试计划，测试主要以单元测试为主。如图5-1是本次测试选用的网络结构图。

图5-1 测试数据图

从图中我们可以看到，本次测试选用了100个中间节点加一个汇聚节点，汇聚节点的坐标在(0,0)，节点的ID号为0。虫洞节点将会被植入到0和62号节点之间。

5.2.1 功能点的划分

根据系统设计，我将功能点划分为：网络连通性、虫洞节点对信息的窃取、汇聚节点对虫洞的检测。下面依次展示功能点的测试结果。

5.2.2 网络连通性

为了测试网络的连通性，首先在process\_packet()最后面加入了一段测试代码，如下面所示。

**if** self.serial == SINK\_NODE\_ID **and** packet.dest == SINK\_NODE\_ID:  
 self.unitTestNetworkConnectivity[str(oacket.dest)] = 1  
 **if** len(self.unitTestNetworkConnectivity.keys()) == self.intermediateNodesNum:  
 **print** "--------------------------------------------------------------------"  
 **print** "| [!] SINK NODE GOT PACKETS FROM EVERY NODES!"  
 **print** "! [!] THE NETWORK'S CONNECTIVITY IS OKAY!"  
 **print** "--------------------------------------------------------------------"

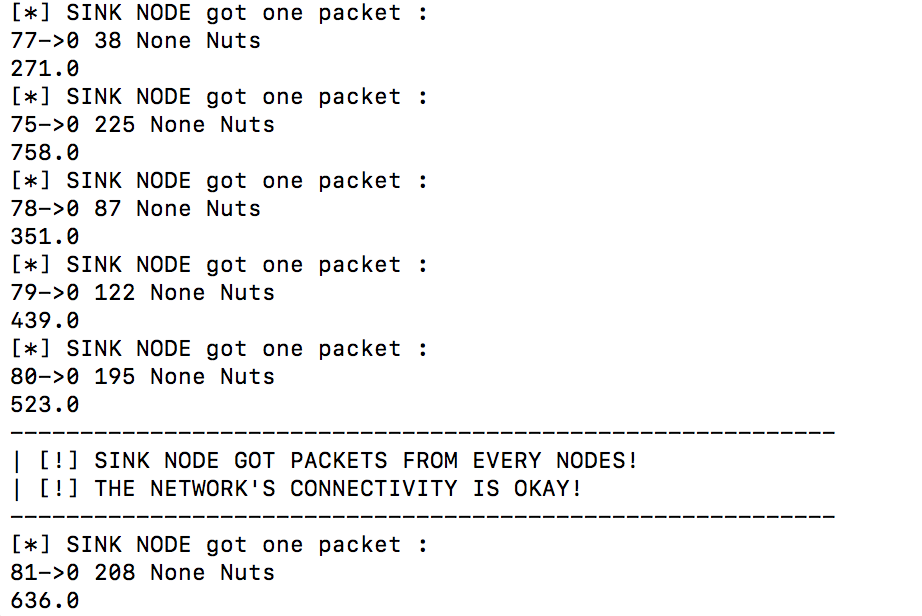
 如上述代码的功能是记录收到包的源节点的ID号，如果收到了所有节点的包，则表示网络是连通的。

图5-2 测试网络连通性运行结果

从图523可以看到汇聚节点打印出了这条消息，说明网络是连通的。

5.2.3 虫洞节点对信息的窃取

为了演示虫洞节点对信息的窃取，在Wormhole类中的BEFORE\_PACKET方法中加入一段代码，如下所示。可以从代码中看出，如果虫洞植入成功，我们应该能收到packet的信息，以及Wormhole got packet这条提示信息。

**def BEFORE\_PACKET**(self, packet):  
 **print** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  
 **print** "[\*\*\*] Wormhole got packet."  
 **print** packet  
 **print** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

运行结果如图5-3所示。

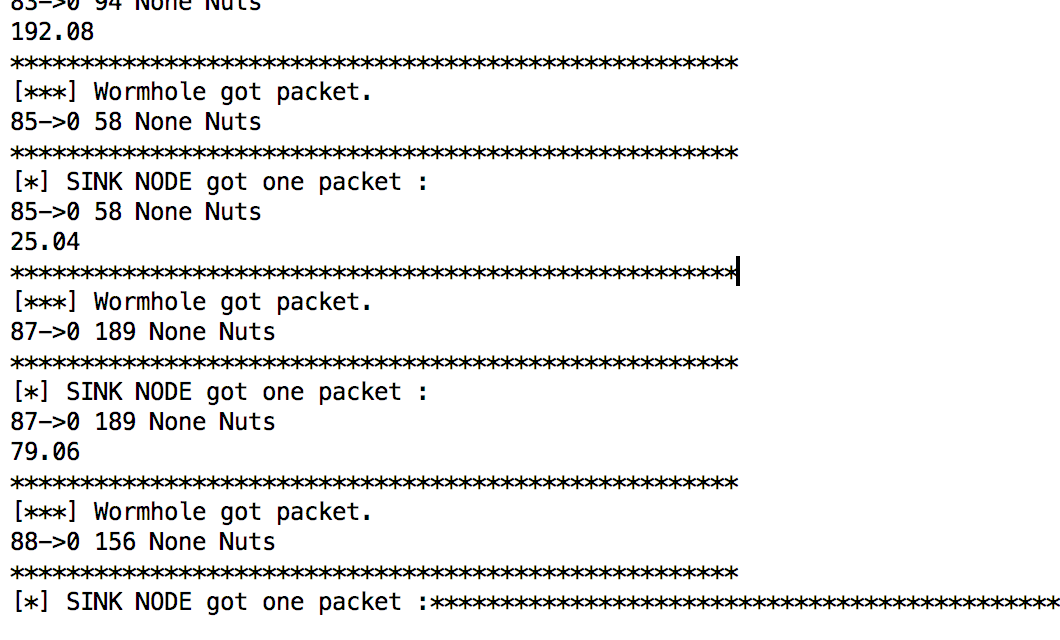
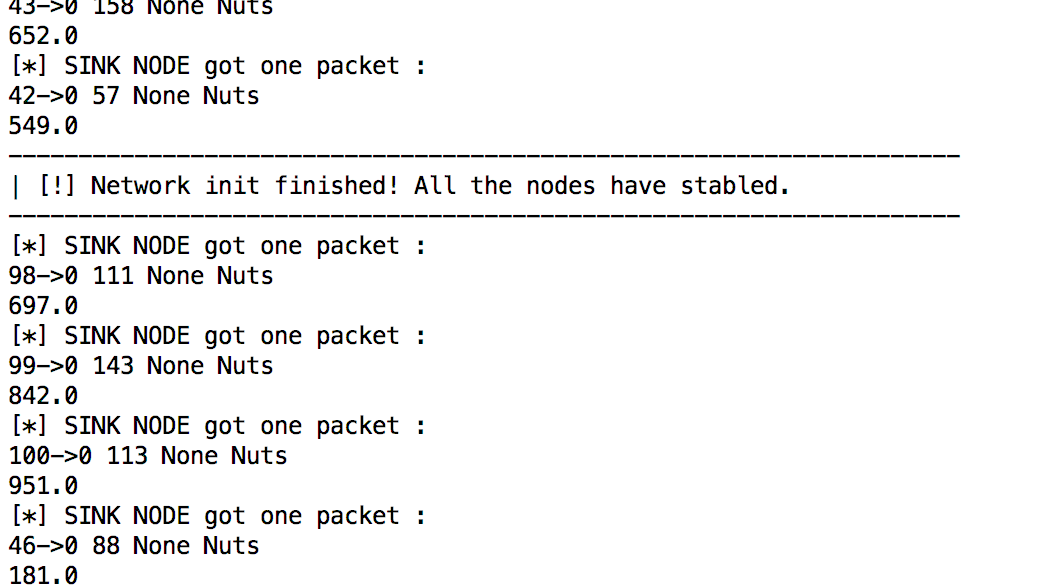
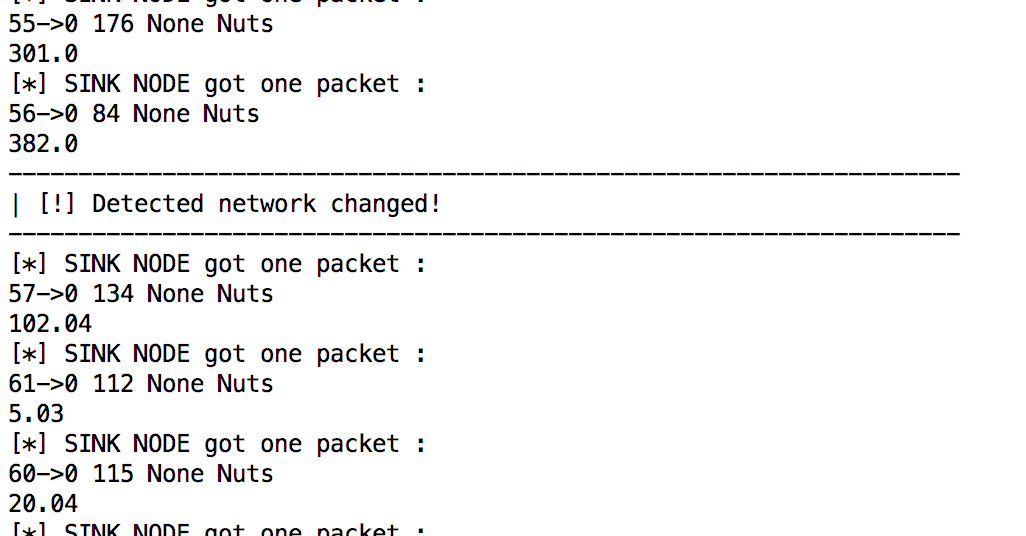
 从图中我们可以看出，虫洞节点确实能够吸收流量，并且也成功的将信息打印了出来。因此证明了虫洞节点植入后确实能起到流量篡改、吸收的作用。测试结果符合测试预期。

图5-3 虫洞节点对信息的窃取测试结果

5.2.4 汇聚节点对虫洞的检测

未进入排名 最后是本文的重点，也是检验本文提出的算法的正确性的关键一步。我们首先注释掉有关单元注释的一切代码。然后运行程序，得到的结果如图5-4、图5-5、图5-6所示。

图5-4 运行一段时间后，网络进入稳定状态

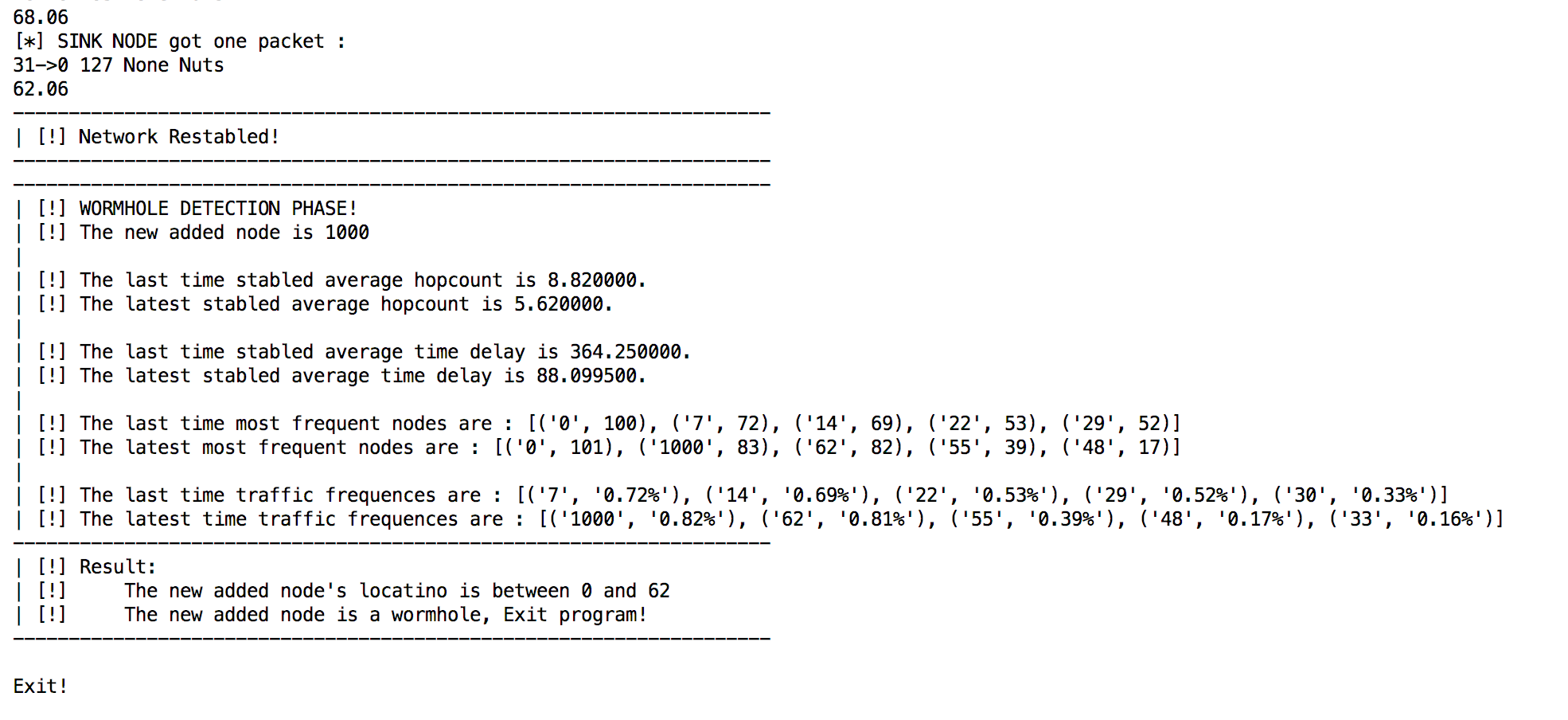
图5-5 汇聚节点检测到网络变化

图 5-6 网络重新进入稳定状态，检测出虫洞节点

从图5-4可以看出，网络在运行一段时间后由汇聚节点检测出稳定状态，并且保存下了稳定后拓扑图及运行状态缓存表。在植入虫洞后，汇聚节点检测出网络变化，并及时作出了响应，当网络再次稳定下来之后，我们可以由5-6看出，平均延时从364.25下降到了88.01，而平均跳数也从8.82降低到了5.62， 因此判定为虫洞节点，退出程序。并且根据前文所述，虫洞节点植入的位置也确实在0和62之间。可见本算法也能够有效的定位到虫洞的位置。

5.3 算法实用性测试

在完成了对整个测试系统的测试后，也就是表明整个系统可以正常运转，但是对于本系统所采用的算法，我们仍需要做测试，因为本系统采用的基于网络状态的概率算法是否有效，仍然是个问题，因此，本节阐述这部分内容。

5.3.1 选取特定虫洞节点的测试

首先，我们特定在几对端点插入节点，这几对端点要求是有一端靠近汇聚节点，另外一端能尽可能靠近数据节点的中心，因为只有这样，虫洞才能依靠自身的高速连接，先吸收大量的数据流量，测试的结果见表5-1。

表5-1 选定端点的虫洞对网络的影响测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 端点号1 | 端点号2 | 跳数变化率 | 延时变化率 | 出现在拓扑结构中次数的排名 |
| 0 | 62 | -34% | -71% | 2 |
| 1 | 60 | -22% | -51% | 3 |
| 8 | 57 | -23% | -52% | 3 |
| 29 | 56 | -8% | -35% | 5 |
| 7 | 47 | -26% | -63% | 3 |
| 21 | 80 | -15% | -44% | 5 |
| 7 | 29 | -3% | -34% | 3 |
| 15 | 48 | -14% | -41% | 5 |
| 14 | 45 | -5% | -35% | 4 |
| 0 | 62 | -37% | -77% | 2 |

如图5-2所示，每一对虫洞节点都对网络产生了非常巨大的影响。具体来说，跳数的变化率最小是3%，最大是37%，而延时的变化率最小是34%最大达到了77%。并且在每一条拓扑路径中出现的频率排名中都占据了前五的位置。说明了，虫洞的植入是会对网络状态造成巨大影响的，而且基于本文的算法是可以检测出虫洞攻击的。这里需要注意的是为什么跳数的变化率远远小于延时的变化率。因为某些节点了为了赶上虫洞节点的“快车道”，不得不绕路而行，这虽然造成了跳数的增加，但是却可以显著减少延时。

5.3.2 随机节点的虫洞测试

在完成了指定节点虫洞测试之后，我们将虫洞的两个端点号用随机算法来生成，进一步查看算法的结果，如表5-2。

表5-2 随机两端点的虫洞节点的网络变化测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 端点号1 | 端点号2 | 跳数变化率 | 延时变化率 | 出现在拓扑结构中次数的排名 |
| 59 | 73 | -6% | -17% | 未进入排名 |
| 73 | 100 | 0% | -1% | 未进入排名 |
| 2 | 29 | 2% | -7% | 4 |
| 21 | 83 | 0% | -1% | 未进入排名 |
| 25 | 98 | -3% | -5% | 未进入排名 |
| 69 | 96 | -2% | -2% | 未进入排名 |
| 6 | 56 | 1% | 0% | 未进入排名 |
| 52 | 72 | -9% | -21% | 未进入排名 |
| 61 | 85 | -1% | -6% | 未进入排名 |
| 12 | 39 | 4% | -1% | 未进入排名 |
| 57 | 100 | 0% | -2% | 未进入排名 |
| 0 | 97 | -24% | -57% | 2 |
| 25 | 96 | -2% | -2% | 未进入排名 |
| 2 | 75 | -20% | -41% | 4 |
| 80 | 98 | 0% | 0% | 未进入排名 |

注意表5-2中第5列未进入排名表示虫洞节点在所有的路径中出现的频率没有排在前十。从这里我们可以看出，虫洞两端点随机选取后，网络状态都没有产生很显著的变化，这是否意味着我们算法的失效呢？经过仔细的分析，我们发现，这是由于随机选取节点后，虫洞节点根本无法完成虫洞节点本应该完成的任务导致。我们已经知道，虫洞节点的原理在于选取两个节点，并在其间形成一个高速的连接，从而来吸引大量的流量，从图5-2中的列5，我们可以看到，虫洞没有出现在大部分的路径中，说明虫洞节点没有吸收到大量的流量，也就无法完成虫洞节点吸收流量的任务，也就称不上虫洞节点。

而我们可以看到，一旦虫洞节点在所有拓扑路径中的排名靠前，就可以对网络产生巨大的影响，这说明我们的算法还是有效的。而其余的无效情况经过分析我们可以分成下列两种情况：1.虫洞节点的两个节点处于路由层次的同一段，也就是说两个节点到汇聚节点的跳数大致相同；2.虫洞节点的两个端节点位置过于靠近。

5.3.4 算法测试小结

在本节中，我们分别利用手动的方式和随机的方式设置了虫洞节点的两个端点来测试算法的实效性。在测试随机设置的方式中，出现了我们意料外的情况，不过在缜密的分析下，可以得出在这种情况下，虫洞节点无法完成其对应的功能，它只能吸收到几个节点的小部分数据流量，我们必须承认本算法对这种情况确实无力解决，因为局部范围的网络变化，无法影响到汇聚节点对整个网络状态变化的判断，也因此，成为本算法的一个短板。在进一步的思考中，我想到，可以分析每一条拓扑路径的变化，然后利用大数据的原理来进行判断，但这又是后话了。

总结起来，本算法可以应对常规的虫洞节点攻击，但对于某些只想对部分节点进行数据窃取的虫洞节点来说本算法显得力不从心。

5.3 本章小结

本章从功能点出发，对整个系统进行了单元测试，并且得到了预期的结果，说明由我设计并实现的系统是正确无误的。随后，对算法的有效性进行了验证，也说明了算法的有效性。

第六章 结束语

在此次开发无线传感器网络虫洞检测测试软件的过程中，我接触了一个完整的系统从设计到实现，到修改设计再次实现反反复复长达三次的过程。给我的影响很大，我深刻的认识到了相比于实现一个系统，设计一个系统难度会更大。同时，也幸亏在项目期间，我另外接触到了一些有关TCP\IP网络，Linux关于Netfilter\IPTable的相关知识，才让我在第三次的设计中体验到了设计之美。我深刻的理解到了，要想成为一个优秀的工程师，不光要有从底层出发的实现能力，更重要的是要学会思考，要学会从顶层设计来看待问题。一个系统只有设计好了，后期的实现才能如鱼得水，才能够免掉许多不必要的麻烦。如果一个计算机学院的学生只拥有底层的实现能力，那么他终将只是一名码农，而成为不了一名优秀的工程师。

同时，也多亏了这次项目，让我跳出了坐井观天、狂妄自大的泥潭，认识到了自己的不足，让我明白了今后的学习道路该怎么走。

从更实际的角度来讲，我更多的了解了有关传感器网络的知识，在此之前，我对此了解几乎为零，当老师告诉我图书馆有关这方面的书籍很多的时候，我甚至相当震惊。因为在此之前，我都认为这次的项目是一个非常偏的、非常小众化的题目。在完成了项目之后，我了解到了这个行业的发展动向、相关的一些理论知识，并且在老师的指导下，设计完成了自己的第一个算法，这对我来说不得不说是一次全新的体验。

这次的项目虽然使我得到了充分的锻炼，并且我自认为对于整个系统的设计也达到了比较完善的地步，但是仍然有几点不足。首先，汇聚节点类应该从设计上从Node类中分离出去，这样，可以极大提高每一个中间节点的内存占有量。并且能使得代码显得更加整洁美观。

其次，这次设计的算法，实际上是老师的一个更完善的算法的简化版本，限于时间限制，我没能完成它，这不能不说是一件很遗憾的事情。

最后，也是由于时间的限制，我没能完成UI界面的设计。

因此，作为我人生中第一次设计的一个完整的系统，我希望在接下来的时间内继续完善这个系统，并且补充适当的文档，最终上传到我个人的Github账号上面去。

致 谢

本次实验是在我导师鲁力老师的带领和帮助下完成的，在早期的试错阶段，他及时的发现了我在设计和理论上面的欠缺，让我及时的补上知识上的漏洞，从而完成了这个系统，在此我表示最衷心的感谢。

其次，在完成论文阶段，我的室友侯晓凯同学在格式排版、图片制作等方面给予了我非常必要的支持，这里表示感谢。

最后，我要感谢我的父母，是他们在我最痛苦、挣扎的时候，给予我爱和鼓励。谢谢！

参考文献

1. 刘颖.一种无线传感器的Amorphous定位算法改进[J].制造业自动化，2015:1-1
2. 龙慧.无线传感器网络发展现状研究[J].单片机与嵌入式系统应用，2011
3. 郎为民，杨宗凯，吴世忠，等.无线传感器网络安全研究[J].计算机科学，2005，32：1-2
4. 穆灵.无线传感器网络安全认证方案及认证协议的研究与设计[J].解放军信息工程大学硕士论文，2015:1-1
5. 胡洪坡.无线传感器网络综述[J].电信快报：网络与通信，2016:1-1
6. 陈娟.无线传感器网络安全研究综述[J].哈尔滨工业大学学报，2011:1-1
7. 马海松.无线传感器网络安全的研究[J].广东工业大学硕士论文，2011:1-1
8. 周龙飞，王潜平.Ad Hoc网络下虫洞攻击的检测方法[J].网络通讯及安全，2008，15：1-3.
9. Y.Yu，R.Govindan，D.Estrin.Geographical and energy aware routing:A recursive data dissemination protocol for wireless sensor networks[C].University of California at Los Angeles Computer Science Department，May，2001.
10. X Wang，J Wong.An End-to-end Detection of Wormhole Attack in Wireless Ad-hoc Networks[C].31st Annual International Software and Application Conference(CPMPSAC 2007).
11. HU Y C，PERRIG A，JOHNSON D B.Packet Leashes:A defense against wormhole attacks in wireless networks[C].Kluwer Academic Publishers，2003.
12. A Pirzada，C McDonald.Circumventing sinkholes and wormholes in wireless sensor networks[R].Internation Conference on Wireless AdHoc Networks(IWWAN)，2005.
13. J Zhen，S Srinivas.Preventing replay attacks for secure routing in ad hoc networks[J].Ad Hoc Networks&Wireless.2003:140-150.
14. S Capkun，L Buttyan，J Hubaux.Sector:Secure Tracking of Node Encouters in Multi-hop Wireless Networks[J].ACM Workshop on Security of Ad Hoc and Sensor Networks.2003:21-32.
15. Lijun Qian，Ning Song，Xiangfang Li.Detecting and Locating Wormhole Attacks in Wireless Ad Hoc Networks[C].Statistical Analysis of Multi-path IEEE Communications Society/WCNC，2005.
16. Levente Buttyon，LazkDora，Istvon Vajda.Statistical Wormhole Detection in Sensor Networks[C].Second European Workshop on Security and Privacy in Ad Hoc and Sensor Networks(ESAS)，2005
17. Lazos L，Poovendran R.SeRLoc.Secure range-independent localization for wireless sensor networks[M].2004 ACM Workshop on Wireless Security.New York.ACM Press，2004，21-30

外文资料原文





外文资料译文

无线传感器网络中的安全问题

他们是非常容易受到各种攻击的侵扰的，包括节点捕获、物理层篡改还有拒绝服务，也因此对基础研究提出了一大批的挑战。

无线传感器网络的应用环境包括海洋和生物监测、制造业效率监测、无风险的地震监测和许多军事方面的应用。未来可能还有更多的行业和领域将会利用到这一新技术，包括高速路监测，环境、污染、森林大火监测，给人类生活提供更多安全保证，水质量监测还有甚至人类心率监测。这项技术的关键优点在于可以从海量的数据中提取出对人类有用的信息。因此如何保护这些信息至关重要。

因为传感器网络提出了与以往不同的挑战，因此传统的对网络的安全措施无法直接应用在这上面。首先，要使传感器网络在经济上可行，必须首先限制传感器节点的能量、计算消耗和通信能力。其次，传感器一般布置在人类不容易接触到的地方，这就增加了物理层攻击的风险。第三，传感器网络直接与物理环境、人类进行接触，这就无疑提出了新的安全问题。因此，现有的安全措施是不适合的，必须要有一套新的方案。幸运的是，新的问题激发了研究者的热情，使得他们可以从一开始就合理的处置传感器网络的安全问题。

在本文中，我们首先提纲挈领的解释在网络中的这些问题，解释在这些问题背后隐藏的美学并且对未来的研究方向提出自己的见解。本文将会提及许多重大的安全问题，包括密钥建立、秘密性、认证、隐秘性、抵御拒绝服务的健壮性、安全路由还有节点捕获。我们同时也讲解了一些更高层次的问题—关于整个传感器网络的问题，并且对未来的挑战作出我们自己的结论。

一个安全的系统

安全问题有时候被认为是一个独立于体系结构的单独的问题，因为总是有一个单独的模块来提供安全性。然而这种方法大部分情况下是一种有缺陷的实现方法。为了实现一个真正安全的系统，安全性必须渗透到系统中的每一个部件，因为任何一个环节的漏洞都会带来安全威胁。因此安全设计必须渗透到系统设计的方方面面。

***密钥和信任的建立。***当需要布置一个传感器网络的时候，一个首先的要求就是为了将来的应用建立加密密钥。在几把年来，为了这个有趣的问题，研究者们已经提出了一大堆的方案。但是为什么这些方案都不能适用于传感器网络呢？答案是传感器网络的内在属性使得所有这些方案都无法使用。大部分市面上的传感器设备都被限制了计算能力，这使得加密算法对它们来说太难了。密钥建立技术需要考虑到一个拥有几百个甚至几千个的网络。更有甚者，里面的一些网络跟传统的网络还大相径庭。传感器节点需要与它们的邻居还有汇聚节点都分享密钥。

最简单的解决方案就是全网共用一个密钥。不幸的，任何一个节点的叛节都会导致密钥的丢失，从而使得攻击者能够解密整个网络的数据流量。另外一种变种方案是这样的，用一把密钥保护一串密钥，或者说每一对通信节点一对密钥。它的工作原理是这样的，一个网络每个节点都用第一把密钥来交换会话密钥，当每一对节点都拥有了各自的会话密钥之后，原来的第一把密钥就不再被需要了。这种方案的问题在于，在网络初始化完成后，无法使新的节点加入进来。

公钥体制是另外一种超出传感器网络计算能力的方法，他的主要优点在于可以使得网络中的任意两个节点建立安全的密钥。

另外一种方法就是为网络中的每对节点预先设置好一个对称密钥。这个方案的问题在于在网络规模较大的时候对存储和计算的消耗超过了传感器网络可以承受的范围。举例来说，一个具有n个节点的网络，每个节点就需要存储n-1个密钥，那么总共就会有n\*(n-1)个密钥被存储了。