

OPNET 网络仿真分析

编 著： 栾 鹏 陈均功 杜 静



前 言

通信网路的规模日趋庞大，移动通信网络的迅猛发展，也使得无线网络的发展研究成为热点。网络体系的架构以及协议的仿真成为研究网络的主要方法。由于更新技术快，结构和功能越来越复杂，通信研究与开发变得越来越苦难。网络仿真技术因此将成为网络规划设计不可缺少的环节。

网络仿真能在建设网络，开展网络业务之前需要对配置的网络设备、所采用的网络技术、承载的网络业务等方面的投资进行综合分析和评估，提出性能价格比最优的解决方案。同时在构建新网络，升级改造现有网络，或者测试新协议，都需要对网络的可靠性和有效性进行客观地评估，从而降低网络建设的投资风险，使设计的网络有很高的性能，或者使测试结果能够真实反映新协议的表现。

OPNET Modeler 是当前业界领先的网络技术开发环境，可以以无与伦比的灵活性用于设计和研究通信网络，设备，协议和应用。Modeler 为开发人员提供了建模，仿真以及分析的集成环境，大大减轻了编程以及数据分析的工作量。Modeler 被世界各大公司和组织用来加速研发过程。

为此，作者编写了《OPNET 网络仿真分析》，本书帮助初学者从 0 开始了解 OPNET Modeler 仿真软件，从软件使用、仿真体系架构、编程实现协议流程、仿真调试、动画演示等方面介绍了 OPNET 的使用。并给出多个应用实例，供使用者参考。其中第 1 章首先对 OPENT 及网络仿真进行了简单的介绍。其次对 OPENT 的第一步使用门槛，安装步骤进行了介绍，尤其是针对 64 位系统下，和目前的 14.5 的 OPNET 安装版本，以及 VS2010 的联合使用。然后介绍了 OPNET 中的基本概念，专业术语，和中英文解释进行介绍。并对 OPNET 存储文件对应的存储内容和 OPNET 的软件操作进行了介绍。第 2 章从网络结构的角度，分三层网络体系结构分别介绍了网络层，节点层，进程层。添加了更多的操作说明，并与应用实例相结合，能更好适用于初学者。把理论，操作，实例同一在一起，能加深对仿真的理解。网络层详细介绍了轨迹轨道、业务模型、拓扑结构、链路模型、管道模型等。节点层详细介绍了处理器与队列模块、收发信机、包流、定向天线等。进程层详细介绍了状态与多进程。第 3 章从编程实现的角度讲述了 OPNET 中各种机制及功能，以及如何通过代码进行实现和实现中的问题。先介绍了关于 OPNET 的变量问题，以及常用结构体、列表、OPNET 常量。其次介绍了 OPNET 中使用对象属性连接网络元素，实现网络架构的功能。然后介绍了 OPNET 中事件机制，包交互机制，并详细介绍了事件机制中的中断，ICI，以及数据包的 TDA 使用。最后讲述了统计量的收集和显示问题。第 4 章讲述了 OPNET 的调制和动画问题，介绍了 OPNET 的 ODB 调试以及与 vs 和 matlab 联合调试，以及 ESA 外部调试功能。另外对 OPNET 的动画功能进行讲解。第 5 章给出了一些常见问题和常见错误，能帮助使用者解决部分问题，但是在书中其他部分给出相关解决方案的，就没有再重复给出问题及解决方法，提醒读者优先查看具体章节的介绍。第 6 章给出了部分应用实例，能帮助喜欢实例学习的读者加速学习效率。

本书时作者学习研究的总结，添加了大量的操作步骤和应用实例，并且将所有关于同一概念的问题集中在一起讲解，既适合从 0 学习，又适合学习中查询问题，非常适合初学者。但是由于作者水平有限，错漏之处在所难免，恳请读者对书中的缺点和错误基于批评指正。

目 录

第一章 OPENT 基础.....	1
1. 1、OPNET 简介.....	1
1. 1. 1、网络仿真简介.....	1
1. 1. 2、OPNET 简介.....	1
1. 1. 3、OPNET Modeler.....	3
1. 1. 3. 1、OPNET Modeler 主要功能	3
1. 1. 3. 2、OPNET Modeler 主要应用	4
1. 1. 4. 3、OPNET Modeler 主要特点	4
1. 1. 4. 4、OPNET Modeler 仿真步骤	6
1. 1. 4. 5、标准模型库.....	6
1. 2、OPNET 安装教程	8
1. 3、基础概念.....	13
1. 4、OPNET 文件存储内容.....	15
1. 5、OPNET 中英文对比.....	16
1. 6、OPNET 软件使用	20
1. 6. 1、菜单栏介绍.....	20
1. 6. 2、工具栏介绍.....	25
第二章 网络搭建.....	27
2. 1、网络框架.....	27
2. 2、子网模型.....	32
2. 2. 1、场景管理.....	33
2. 2. 1、场景配置.....	35
2. 3、节点模型.....	37
2. 3. 1、队列与处理器.....	40
2. 3. 2、包流.....	43
2. 3. 3、收发信机.....	45
2. 3. 3. 1、无线发信机.....	45
2. 3. 3. 2、无线接收机.....	48
2. 3. 3. 3、点对点发信机.....	49
2. 3. 3. 4、点对点收信机.....	49
2. 3. 4、定向天线.....	50
2. 4、进程模型.....	55
2. 4. 1、进程概念.....	55
2. 4. 2、多进程.....	57
2. 4. 3、状态.....	59
2. 5、轨迹轨道.....	63
2. 5. 1、轨迹的定义.....	63
2. 5. 2、轨迹的应用.....	65
2. 5. 3、轨迹修改.....	66
2. 5. 5、卫星轨道的使用	67
2. 5. 6、轨迹模块定义.....	69

2.6、链路模型.....	71
2.6.1、有线链路的链路模型.....	72
2.6.2、管道函数.....	74
2.6.2.1、管道函数的创建修改.....	75
2.6.2.2、有线无线中的管道类别.....	76
2.6.2.3、点对点链路管道阶段函数.....	77
(1) 传输时延阶段.....	77
(2) 传播时延阶段.....	78
(3) 纠错阶段.....	79
2.6.2.4、总线型管道模型.....	79
(1) 封闭性计算阶段.....	79
(2) 传输时延阶段.....	80
(3) 传播时延阶段.....	80
(4) 冲突检测阶段.....	80
(5) 误码数目分配阶段.....	81
(6) 纠错阶段.....	81
2.6.2.5、无线链路管道阶段函数.....	81
第一部分：无线发信机管道阶段流程.....	83
第二部分：无线接收机管道阶段流程.....	85
第三部分：无线链路管道阶段函数.....	86
2.7、拓扑.....	102
第三章 代码实现.....	104
3.1、变量.....	104
3.1.1、变量区别.....	105
3.1.2、变量定义.....	106
3.1.3、随机变量分布.....	107
3.2、结构体与列表.....	110
3.2.1、结构体.....	110
3.2.2、列表.....	111
3.3、常量.....	113
3.3.1、对象常量.....	113
3.3.2、中断类型常量.....	114
3.4、对象属性.....	115
3.4.1、对象拓扑.....	115
3.4.2、属性.....	117
3.4.3、对象位置.....	118
3.4.4、对象操作.....	119
3.5、事件.....	120
3.5.1、事件附件状态.....	122
3.5.2、中断.....	122
3.5.2.1、多进程下的中断设置.....	124
3.5.2.2、仿真开始中断.....	125
3.5.2.3、自中断.....	126
3.5.2.4、进程中断.....	128

3.5.2.5、远程中断.....	129
3.5.2.6、流中断.....	130
3.5.2.7、统计量中断.....	133
3.5.2.8、其他中断.....	138
3.5.3、ICI.....	139
3.5.3.1、ICI 格式定义.....	139
3.5.3.2、ICI 操作.....	140
3.5.3.3、ICI 与事件.....	140
3.5.3.4、ICI 与数据包.....	142
3.6、数据包.....	142
3.6.1、数据包格式的定义.....	143
3.6.2、数据包的创建.....	144
3.6.3、数据包字段域.....	144
3.6.3.1、数据包字段的设置：.....	144
3.6.3.2、数据包字段读取.....	146
3.6.3.3、字段域操作.....	146
3.6.4、数据包属性域.....	148
3.6.5、数据包传输.....	151
3.6.6、数据包其他操作.....	154
3.6.7、TDA.....	154
3.7、统计量.....	158
3.7.1、统计量的定义.....	158
3.7.2、统计量的捕获模型 Capture Mode.....	159
3.7.3、绘制风格 Draw style.....	162
3.7.4、注册统计量.....	163
3.7.5、数据收集.....	163
3.7.6、统计量的函数计算.....	163
3.7.7、选择想要收集的统计量.....	163
3.7.8、数据显示及导出.....	165
第四章 调试演示.....	175
4.1、调试.....	175
4.1.1、仿真设置.....	175
4.1.2、ODB 调试.....	178
4.1.2.1、调试输出窗口.....	179
4.1.2.1.1、控制台输出窗口.....	179
4.1.2.1.2、动画输出窗口.....	189
4.1.2.1.3、仿真进度窗口.....	191
4.1.2.2、记录窗口.....	191
4.1.3、调试技巧.....	193
4.1.3.1、语法调试技巧.....	194
4.1.3.2、逻辑调试技巧.....	194
4.2、动画.....	195
第五章 常见问题及错误.....	198
5.1、常见错误.....	198

5.2、常见问题.....	199
实例 1：创建项目与场景	202
实例 2：分布式子网搭建.....	205
实例 3：为 AP 和用户创建节点模型.....	209
实例 4：创建数据包接收进程模型.....	217
实例 5：定向天线方向图.....	220
实例 6：卫星轨迹.....	225
实例 7：配置星状网实例操作.....	229
实例 8：对象属性设置.....	231
实例 9：发送回应的包交互实现.....	234
实例 10：进程中断.....	239
实例 11：随机分布.....	240
实例 12：结构体与列表.....	241
实例 13：统计量的使用.....	243
实例 14：TDA 使用.....	246

第一章 OPENT 基础

1.1、OPNET 简介

1.1.1、网络仿真简介

1、网络仿真的目的：

- 在建设网络，开展网络业务之前需要对配置的网络设备、所采用的网络技术、承载的网络业务等方面的投资进行综合分析和评估，提出性能价格比最优的解决方案。
- 构建新网络，升级改造现有网络，或者测试新协议，都需要对网络的可靠性和有效性进行客观地评估，从而降低网络建设的投资风险，使设计的网络有很高的性能，或者使测试结果能够真实反映新协议的表现。

2、网络仿真的优点

- 为网络的规划设计提供客观、可靠的定量依据
- 缩短网络建设周期
- 提高网络建设中决策的科学性

3、网络设计流程

对于网络的设计和管理，一般分为 3 个阶段：

- 第 1 阶段为设计阶段：包括网络拓扑结构的设计，协议的设计和配置以及网络中设备的设计和选择；
- 第 2 阶段为发布阶段：设计出的网络能够具有一定性能，如吞吐率、响应时间等等；
- 第 3 阶段为实际运营中的故障诊断、排错和升级优化。

OPNET 公司的整个产品线正好能面向网络研发的不同阶段，既可以作网络的设计，也可以作为发布网络性能的依据，还可以作为已投入运营的网络的优化和故障诊断工具。OPNET 公司也是当前业界智能化网络管理分析解决方案的主要提供商。

1.1.2、OPNET 简介

OPNET 公司起源于 MIT（麻省理工学院），成立于 1986 年。1987 年 OPNET 公司发布了其第一个商业化的网络性能仿真软件，提供了具有重要意义的网络性能优化工具，使得具有预测性的网络性能管理和仿真成为可能。1987 年以来，OPNET 迅速而稳步发展，作为高科技之网络规划、仿真及分析工具，OPNET 在通信、国防及电脑网络领域已经被广泛认可和采用。成千上万的组织使用 OPNET 软件来优化网络性能、最大限度地提高通信网络和应用的可用性。

OPNET 目前在全球有近 4000 客户，其中大约 75% 的客户在美国本土，大约 25% 的客户在其他国家。OPNET 的全球以及北美部分客户群包括：

- 军方客户：OPNET 由于其仿真的精确性，友好的界面以及具备 HLA 体系接口，成为军方客户进行和网络有关开发时的首选产品。OPNET 从 1987 年产生起，一直被美国军方作为网络建模和仿真的标准，并参与了许多军方项目的开发。

- 电信级运营商 (A&T, NTT Docomo, France Telecom 等): 这部分客户相对于中大企业，具有更复杂的网络结构和协议配置，因此管理起来更加的复杂。OPNET 利用其极高的网络智能来辅助运营商的网管人员管理其网络，并且 OPNET 具有很好的开放型和互联性，可以和当前很多流行的网络管理和监控软件一起协同工作，如 HP 的 OPENview, Tivoli 的 NETVIEW, CISCO NETFLOW 以及 Agilent 的 NetMatrix。
- 大型的通信设备制造商 (如 3COM, Cisco, Nortel Networks, Lucent 等): 这部分客户需要 OPNET 作为其网络设备，协议以及应用开发的工具。
- 中大型企业 (如汇丰银行、Boeing 等等): 这部分客户一般具有比较庞大的内部网络，企业的业务依靠网络来进行，一些应用对网络的可靠性以及有效性具有较强的依赖性。OPNET 的产品可以帮助企业的网管人员以及开发人员很好的设计和管理企业内部网，以及帮助他们进行故障诊断。

在国内，1998年以前OPNET属限制产品，对中国和其他的社会主义国家是禁运的，近几年才开始进入中国市场。虽然进入中国的时间不长，但也已经有三十多家用户。这三十多家用户中，有一定影响力的客户较多，如总参通信部，电子部54所，海军自动化所、电子部7所、10所、29所、30所等。在民用方面OPNET在我国的应用也越来越广泛，如信息产业部传输所、大唐电信、中兴通信、华为、MOTOROLA等设备制造商用OPNET进行设备、协议等的开发；中国电信广州研究院、中国电信规划设计院等单位用OPNET进行网络规划、优化。

OPNET产品分为如图1-1所示的几个系列：



图1-1. OPNET产品系列

不同的产品针对不同的市场和客户。Modeler主要用于研发，面向研发单位，设备制造商，以及一些大学。ITGuru主要针对大型企业对其内部网络进行管理和分析。SPGuru和WDMGuru面向运营商。ODK是一个软件开发工具包，由许多丰富成熟的软件组件库构成。ODK主要用于开发定制的应用程序，用于网络建模、仿真、分析与优化。

当前，在计算机网络仿真领域方面的经验是十分有限的。越来越多的商业公司需要这种类型的咨询，但是由于缺乏经验从而导致了需要大量的海外投入，从海外寻求这种类型的咨询，特别是从一些西方国家。如果这些经验能够从本地获得，则会节约很多资金。OPNET是一个被业界广泛承认而用于该领域的网络仿真工具。

今天的网络设计者尽管经验十分欠缺，但必须为他们的环境选择合适的技术。OPNET的解决方案使得网络设计者可以实现如下功能：

- ◆ 向网络增加新的业务和用户的性能影响分析。
- ◆ 使用定量的方法来进行精确的升级和规划分析。
- ◆ 在购买设备之前，对各种不同的候选方案进行快速的“如果一怎样”的分析。

- ◆ 针对网络设计优化性价比。
- ◆ 针对主要的设备进行基线预算验证。
- ◆ 使用高级动画功能实现仿真事件可视化。
- ◆ 灵活的开发定制用户模块。

OPNET 公司是全球领先的决策支持工具提供商，总部设在美国华盛顿特区，主要为网络专业人士提供基于软件的预测解决方案。简单说，是一种通信网络仿真工具，包括有线网络和无线网络。OPNET仿真模型库为客户提供了一系列的仿真模型。在这些仿真模型的基础上，实现对通信网络的仿真。包括网络拓扑结构、路由设计、业务配置等等。OPNET仿真模型库与其网络仿真引擎是分离的，这种设计方便修改、升级。同时，客户还可以根据自己的要求定制模型。

针对有线网现有的OSI、TCP/IP等协议簇模型，OPNET拥有完备的网络协议数据库。可以进行通信网络的建设仿真及故障查询。

1. 1. 3、OPNET Modeler

1. 1. 3. 1、OPNET Modeler 主要功能

OPNET Modeler 是当前业界领先的网络技术开发环境，可以以无与伦比的灵活性用于设计和研究通信网络，设备，协议和应用。Modeler 为开发人员提供了建模，仿真以及分析的集成环境，大大减轻了编程以及数据分析的工作量。Modeler被世界各大公司和组织用 来加速研发过程。

Modeler 的面向对象的建模方法和图形化的编辑器反映了实际网络和网络组件的结构，因此实际的系统可以直观的映射到模型中。Modeler支持所有网络类型和技术，能够使您自信的回答任何困难的问题。

使用Modeler，将可以给用户带来如下利益：

- ◆ 提升网络研发的成果：Modeler提供的各种专门的编辑器，以及分析工具和一些最新的模型，使得研发人员可以专注于项目中特定部分的开发，而不用浪费精力在一些没有必要的地方。
 - ◆ 改善产品质量：Modeler提供测试实际产品的一个虚拟网络环境，可以有效的避免一些设计中的错误。
 - ◆ 减小从研发到市场的时间：在完成实际产品之前作充分的验证，采用Modeler来向客户以及合作伙伴展示您的解决方案的价值。
- 如前所述，OPNET Modeler主要面向于研发，主要功能体现在以下四个方面：
- ◆ 设备的研发：一些设备制造商，如3com,Cisco等，在新的设备投入市场之前，需要将其模型放到OPNET的虚拟网络环境中进行验证。
 - ◆ 协议的研发：用于开发用户需要的，或者下一代的通信协议，如IPV6，并且仿真其性能。
 - ◆ 网络的研发：用于分析有线/无线设备组网以后的整体性能与特定参数。
 - ◆ 业务的研发：开发新型的业务模式。

1.1.3.2、OPNET Modeler 主要应用

虽然OPNET Modeler 的功能集中在以上四项，但是并不局限于以上。它可以很灵活的应用于各种网络（有线，无线，卫星以及混合网络）的仿真。以下列出它的一些典型的应用示例：

- ◆ 建模并仿真不同业务类型和负荷下的不同网络技术，例如模拟不同战略网络的网络负荷。
- ◆ 新的路由算法，业务管理方案，通信网络协议和结构的开发和评价，以使其更稳固和有效。
- ◆ 网络基础结构的规划和设计。
- ◆ 在转到实验室的测试床之前，在一个灵活快速的原型环境中实现和调整新设计的网络协议。
- ◆ 通信网络的培训。
- ◆ 从网络中抓取网络信息，并且研究变更将如何影响网络性能。
- ◆ 模拟复杂的环境效应所带来的对性能的影响，如无线通信网络中的路径损耗，背景地形和干扰噪声。
- ◆ 评估在移动性，覆盖范围，吞吐量，发射频率，带宽，收发机的为止和天线增益方面性能和设计的平衡。
- ◆ 用于很多通信网络应用：
 - ◆ LAN/WAN 技术
 - ◆ 当前和未来的蜂窝技术
 - ◆ 无线包消息业务
 - ◆ 无线寻呼
 - ◆ 战略战场通信网络
 - ◆ 无线局域网技术
 - ◆ 卫星通信
 - ◆ MPLS 多协议标签交换
 - ◆ 电路交换
 - ◆ PNNI 专用网络节点接口
 - ◆ IPv6

1.1.4.3、OPNET Modeler 主要特点

Modeler 主要面向研发，其主要特征为：

- ◆ 层次化的网络模型。使用无限嵌套的子网来建立复杂的网络拓扑结构。
- ◆ 简单明了的建模方法。Modeler建模过程分为3个层次：进程（process）层次、节点（Node）层次以及网络（Network）层次。在进程层次模拟单个对象的行为，在节点层次中将其互连成设备，在网络层次中将这些设备互连组成网络。几个不同的网络场景组成“项目”，用以比较不同的设计方案。这也是Modeler建模的重要机制，这种机制有利于项目的管理和分工。
- ◆ 有限状态机。在进程层次使用有限状态机来对协议和其他进程进行建模，因此造成一个进程中的代码只能顺序执行。在有限状态机的状态和转移条件中使用C/C++语言对任何进程进行模拟。用户可以随心所欲地控制仿真的详细程度。有限状态机加上标准的

C/C++以及OPNET本身提供的400多个库函数构成了Modeler编程的核心。OPNET称这个集合为Proto C语言。

- ◆ 对协议编程的全面支持。支持400多个库函数以及书写风格简洁的协议模型。OPNET已经提供了众多协议，因此对于很多协议，无需进行额外的编程。
- ◆ 系统的完全开放性。Modeler中源码全部开放，用户可以根据自己的需要对源码进行添加和修改用于自定义仿真，用户可以对模型进行加密以保护自己的知识产权。
- ◆ 高效的仿真引擎。使用Modeler进行开发的仿真平台，使仿真的效率相当高。
- ◆ 集成的分析工具。Modeler仿真结果的显示界面十分友好，可以轻松刻画和分析各种类型的曲线，也可将曲线数据导出到电子表格中。
- ◆ 集成调试器。快速地验证仿真或发现仿真中存在的问题，OPNET本身有自己的调试工具——OPNET Debugger（ODB）。
- ◆ 源代码调试。方便地调试由OPNET生成的C/C++源代码。
- ◆ 高可扩展性和高效率的仿真引擎。快速仿真引擎能对有线和无线模型进行快速运行仿真，比如，以比标准网络速度快得多的速度仿真一个地形环境下的上千个无线节点的动态应用和路由行为。
- ◆ 对无线链路，点对点链路和点对多点链路分别建模。链路行为是开放的，可编程的。精确定义了链路的时延，可用性，误比特率，和吞吐量等特性。使用增强TIREM模型，Longley-Rice模型和Free Space等传播模型库整合描述物理层特性和环境的共同影响。
- ◆ 最先进的建模平台，具有高度优化的串行和完全并行离散事件仿真，混合仿真和数值仿真，以及HLA和协同仿真技术。
- ◆ 动画：Modeler可以在仿真中或仿真后显示模型行为的动画，使得仿真平台具有很好的演示效果。同时具有3D显示的接口。
- ◆ 从文本文件，XML和流行的软件导入数据，比如Cisco，HP，NetScout，BMC，Concord，Sniffer，Infovista，MRTG，cflowd，tcpdump等。
- ◆ 为每一个设备提供了成本选项。可以将网络成本导出至报表。
- ◆ 丰富的模型库，提供了详细的协议模型和应用模型。其中包括多层应用，语音，TCP，IP，OSPF，BGP，EIGRP，RIP，RSVP，帧中继，FDDI，以太网，ATM，802.11无线局域网，MPLS，PNNI，DOCSIS，UMTS，IP Multicast，Circuit Switch，MANET，Mobile IP，IS-IS，等等。以上模型都以开放源码的有限状态机形式给出。
- ◆ 丰富的网络模型库，标准模型库包含了数百个制造商的专有模型和通用模型，包括路由器，交换机，工作站和包生成器。使用“设备制造器”，您可以快速地创建属于自己的设备模型，并且从局域网模型或者云图模型中汇聚流量。
- ◆ 地理和移动建模。对于无线小区，移动 Ad hoc 网络，无线局域网和卫星网络或者任何带有移动节点的网络都进行了建模。可以动态控制或者预定义每一个节点的移动轨迹。可以通过添加地图或者背景图片来增强可视效果。如果选择了地形建模模块（TMM），就可以导入DTED或者USGS格式的数据，在仿真中考虑地形对无线传播的影响。
- ◆ 模型可以在不同操作系统和硬件体系结构的Windows NT, 2000, XP 和UNIX之间透明共享（无需修改）。
- ◆ 方便的License管理，增强的浮动License系统能够通过Internet自动下载密钥，图形化的界面更利于license管理。
- ◆ Modeler提供的十几个编辑器大大加快了建模的进程。Modeler提供的编辑器为：项目编辑器，节点编辑器，过程编辑器，链路编辑器，路径编辑器，包编辑器，天线模式编辑器，接口控制信息编辑器，调制曲线编辑器，概率分布函数编辑器，探针编辑

器，图标编辑器，源程序编辑器等等。每个编辑器完成一定的功能，使得原先需要编写很长代码的程序，只需要通过图形化的界面进行一些设置就可以完成。

1.1.4.4、OPNET Modeler 仿真步骤

- 1) 创建项目工程
- 2) 设置子网模型 (Network)
配置网络拓扑->配置业务模型->设置节点属性->设置轨迹轨道->配置链路模型
- 3) 设置节点模型 (Node)
搭建节点内部模块连接->设置处理器队列模块->明确包交互流程->设置收发信机->设置统计中断
- 4) 设置进程模型 (Process)
配置进程属性->搭建状态转移图->编写方案代码
- 5) 设置收集结果统计量 (Statistics)
- 6) 运行仿真 (Simulation)
- 7) 调试模块再次仿真 (Re-simulation)
- 8) 最后发布结果和拓扑报告 (Report)

1.1.4.5、标准模型库

(1) 链路层技术如表 1-1 所示。

表1-1 链路层技术

模型	缩写
Asynchronous Transfer Mode 异步传输模式	ATM
Ethernet、Fast Ethernet、Gigabit Ethernet 以太网、快速以太网、千兆以太网	(coaxial, 10BaseT, 100BaseT, 1000BaseX)
EtherChannel 以太通道	
Fiber Distributed Data Interface 光纤分布式数据接口	FDDI
Frame Relay 帧中继	FR
LAN Emulation 局域网仿真域	LANE
Link Access Procedure-Balanced 链路仿真过程平衡	LAPB
Spanning Tree Bridge 生成树网桥	STB
Spatial Reuse Protocol 空间复用协议	SRP
System Network Architecture 系统网络架构	SNA
Token Ring 令牌环	TR
Virtual LANs 虚拟局域网	VLAN
X.25	X.25

(2) 网络层协议如表 1-2 所示

表1-2 网络层技术

模型	缩写
Internet Protocol (IP协议)	IP
Internetworking Packet Exchange 互联网数据包交换协议	IPX
Resource reSerVation Protocol 资源预留协议	RSVP

(3) 路由协议如表 1-3 所示

表1-3 路由层技术

模型	缩写
Open Shortest Path First 开放式最短路径优先	OSPF
Border Gateway Protocol 边界网关协议	BGP
Interior Gateway Routing Protocol 内部网关路由协议	IGRP
Routing Information Protocol 路由信息协议	RIP
Enhanced Interior Gateway Protocol 增强内部网关路由协议	EIGRP

(4) 物理层技术如表 1-4 所示

表1-4 物理层技术

模型	缩写
Synchronous Optical Networks 同步光纤网络	SONET
Asynchronous Digital Subscriber Loop	xDSL
Integrated Services Digital Network 综合业务数字网	ISDN

(5) 传输层协议如表 1-5 所示

表1-5 传输层技术

模型	缩写
Transmission Control Protocol 传输控制协议	TCP
Network Control Protocol 网络控制协议	NCP
User Datagram Protocol 用户数据包协议	UDP

(6) 应用层模型如表 1-6 所示

表1-6 应用层技术

模型	缩写
Custom Application 定制应用	
Database 数据库	
E-mail 电子邮件	
File Transfer Protocol 文件传输	FTP
Hypertext Transfer Protocol 超文本传输	HTTP HTTP 1.0
Remote Login 远程注册	
Print 打印	

Voice Application语音应用	
Video Conferencing视频会议	

(7) 无线网络模块包含的协议如表 1-7 所示

表1-7 无线网络模块技术

模型	缩写
Ad hoc On Demand Distance Vector按需路由协议	AODV
Advanced Mobile Phone Service高级移动电话服务	AMPS
Dynamic Source Routing动态源路由	DSR
Temporally-Ordered Routing Algorithm临时按序路由算法	TORA
Wireless LANs无线局域网	WLAN

1. 2、OPNET 安装教程

本书使用的 OPNET 安装配置环境如表 1-8 所示。

表1-8 OPNET安装软件和系统环境

使用版本	OPNET14.5
	VS2010
	NET4.0
需要软件	VS2010
	modeler_145A_PL1_7116_win.exe
	models_145A_PL1_27Feb08_win
	modeler_docs_28-Jan-2008_win
	OPNET.Modeler.14.5.License.Maker-FFS
安装系统环境	Win7 32位 64位
	Win8 32位 64位
	Win10 32位 64位

第一部分：安装 vs2010，设置 vs 环境变量

本书使用以 win7 64 位安装 OPNET14.5 仿真软件。

【1】安装 vs2010，只需要安装 c++语言（安装过程略）。我的安装目录为 G:\vs2010

【2】开始设置 VC 编译器的环境变量，右键单击“这台电脑”，选择“属性”。单击“高级系统设置”，在弹出的系统属性窗口中选择“高级”选项卡



图1-2. 系统环境变量位置

单击“环境变量”，进入“环境变量”设置窗口。

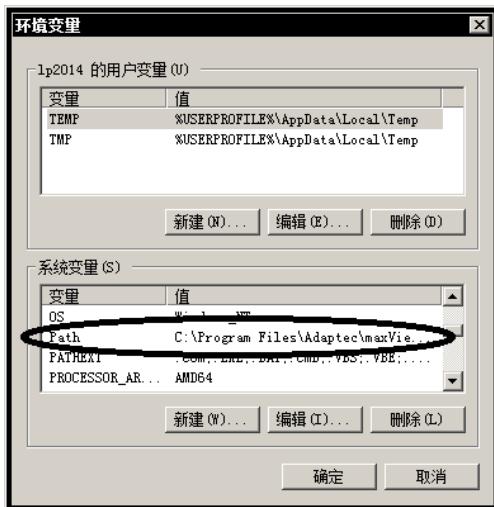


图1-3. 系统环境变量面板

【3】 环境变量，每个路径之间用英文分号间隔，最后一个路径后面的分号不要。需要新建 lib 和 include 环境变量。具体参数如表 1-9 所示。

表1-9 VS环境变量设置

Path	F:\vs\VC\bin; (填写的是 vs 的安装路径) F:\vs\Common7\Tools; F:\vs\Common7\IDE;
lib	F:\vs\VC\lib; 64 位机器下 C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Lib 32 位机器下 C:\Program Files\Microsoft SDKs\Windows\v7.0A\Lib
include	F:\vs\VC\include; 64 位机器下 C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Include 32 位机器下 C:\Program Files\Microsoft SDKs\Windows\v7.0A\Include

配置完可运行“cmd”->cl (opnet 用 ml) 检测是否配置成功。

第二部分：安装 OPNET 软件

(2.1) 管理员身份运行 modeler_145A_PL1_7116_win.exe

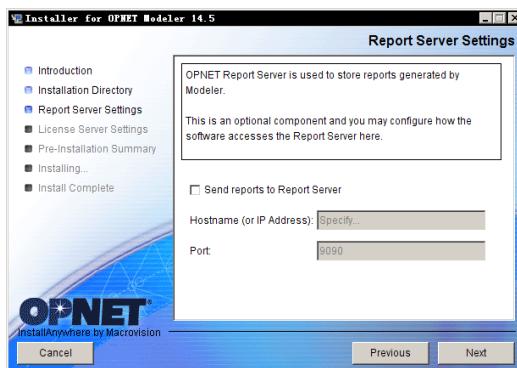
【1】点击 Next



【2】不要修改安装路径，点击 Next



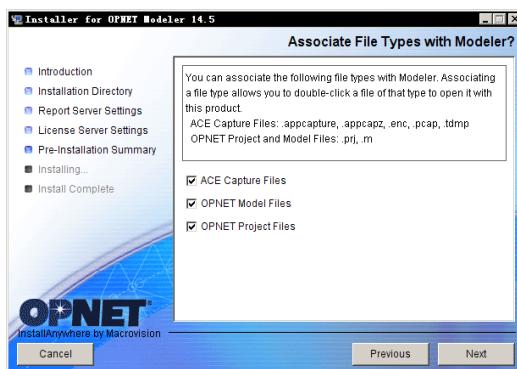
【3】直接点击 Next



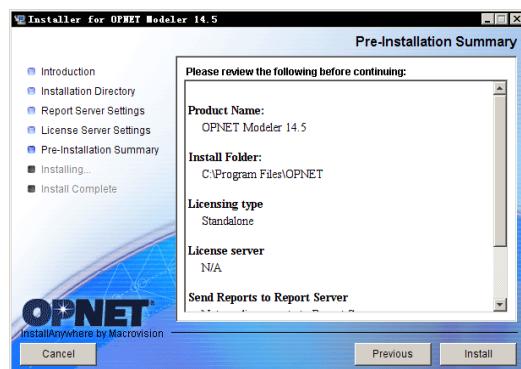
【4】直接点击 Next



【5】选中 OPNET Model Files 点击 Next



【6】点击 Install



【7】点击 Done 关闭窗口

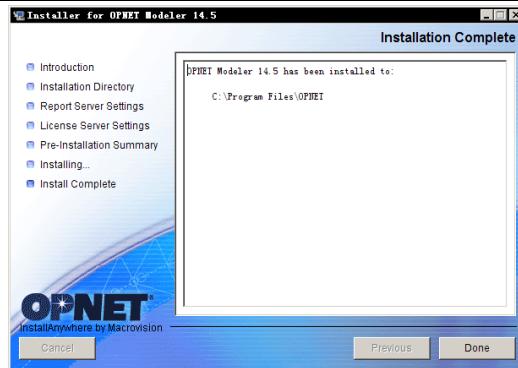
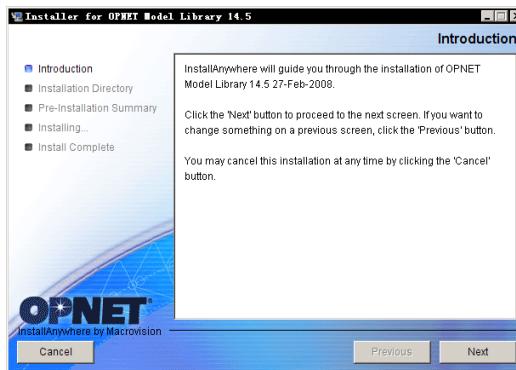


图1-4. modeler_145A_PL1_7116_win安装方法

(2.2) 以管理员身份运行 models_145A_PL1_27Feb08_win.exe

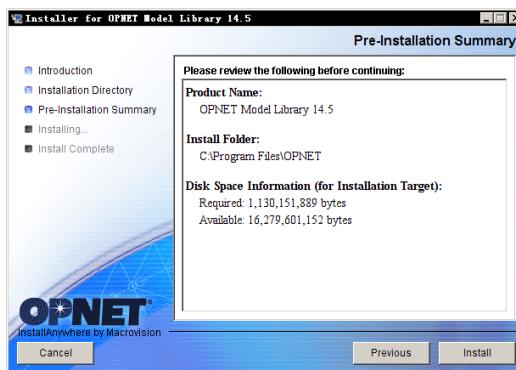
【1】直接点击 Next



【2】不要修改安装路径,点击 Next



【3】点击 Install



【4】安装完成,点击 Done 关闭窗口.

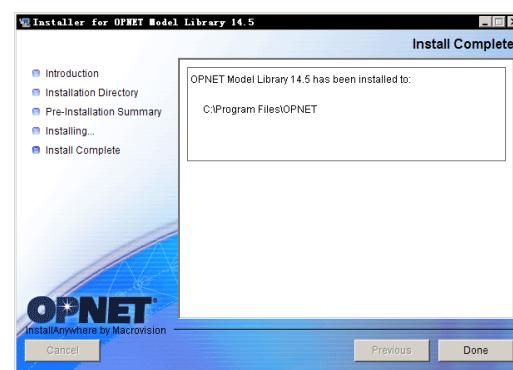
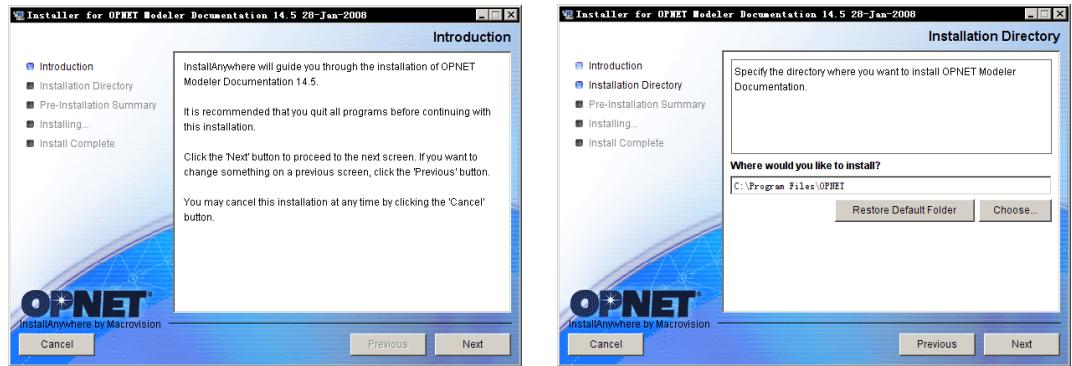


图1-5. models_145A_PL1_27Feb08_win安装方法

(2.3) 以管理员身份运行 modeler_docs_28-Jan-2008_win.exe

【1】直接点击 Next

【2】不要修改安装目录,点击 Next



【3】直接点击 Install

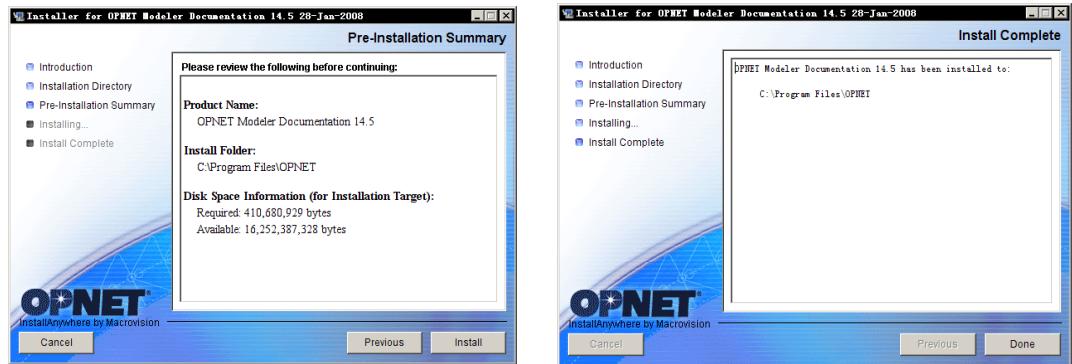


图1-6. modeler_docs_28-Jan-2008_win安装方法

(2.4) 以管理员身份运行 OPNET. Modeler. 14.5. License. Maker-FFS. exe，点击“确定”，注册完毕。



图1-7. OPNET. Modeler. 14.5. License. Maker-FFS安装方法

至此文件安装完毕。安装完之后将

“C:\Program Files\OPNET\14.5.A\sys\pc_intel_win32\bin\manifest_NET2008”的全部文件复制到“C:\Program Files\OPNET\14.5.A\sys\pc_intel_win32\bin”目录下包含文件 op_runsim.exe.manifest、op_runsim_dev.exe.manifest、op_runsim_mtdev.exe.manifest、op_runsim_mtopt.exe.manifest、op_runsim_opt.exe.manifest 这 5 个文件。

第三部分：设置 OPNET 环境变量

下面是 opnet 的环境变量设置，无论机器时 64 位或者 32 位，因为 opnet 安装在了 Program Files 文件夹下，所以就填 Program Files 而不是 Program Files (x86)。环境变量添加如表 1-10 所示。

表1-10 OPNET环境变量设置

Path	C:\Program Files\OPNET\14.5.A\sys\pc_intel_win32\bin
lib	C:\Program Files\OPNET\14.5.A\sys\lib; C:\Program Files\OPNET\14.5.A\sys\pc_intel_win32\lib
include	C:\Program Files\OPNET\14.5.A\sys\include; C:\Program Files\OPNET\14.5.A\models\std\include

第四部分：软件运行

找到 OPNET 软件的快捷方式。不要选择带有 64 位标志的快捷方式，因为我们使用是在 64 位机器上安装运行的 32 位的软件。

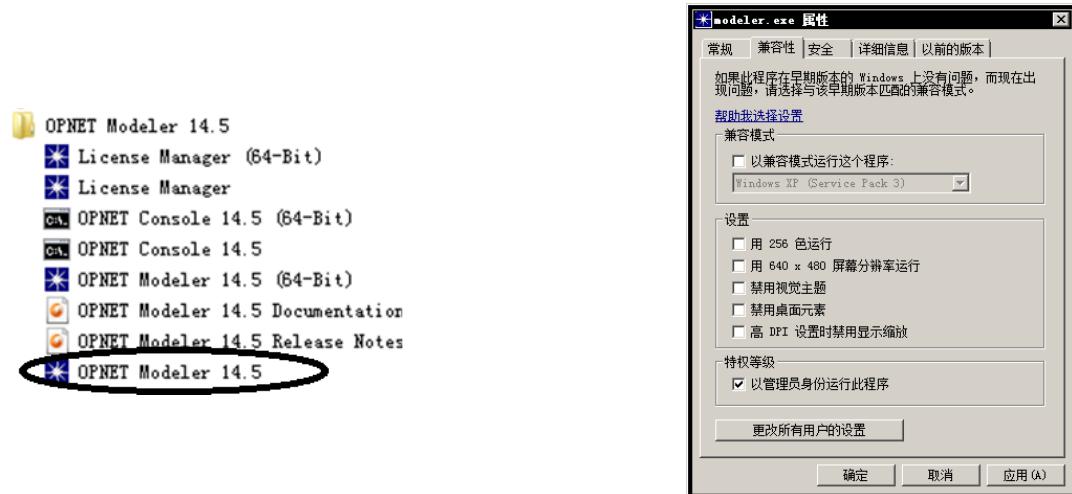


图1-8. OPNET管理员身份运行

通过快捷方式找到软件的目录如图 1-8 所示，右键点击 exe 软件或快捷方式，选择属性，弹出属性对话框如图 1-8 所示。选择兼容性页面下，选中“以管理员身份运行此程序”，点击“应用”，点击“确定”。至此 OPNET 可以使用。

1.3、基础概念

1. 项目 (Project) 与场景 (Scenario)

OPNET Modeler 采用“项目-场景”的方法对网络建模，一个项目包含多个仿真场景。场景是网络的一个实例，一种配置，具体来说就是拓扑结构、协议、应用、流量以及仿真设置的总和。在 Modeler 仿真时，每个项目中至少包含一个仿真场景，代表网络模型，它是具体的网络仿真环境配置。项目提供场景复制功能，可以对场景进行备份，通过改变新场景的参数运行仿真来测试系统各方面的功能及是否存在瓶颈。

2. 子网 (Subnet)

OPNET 中顶层网络始终为一个叫“top”的网络，“top”网络内包含的网络模块称作子网，子网内可以嵌套另一个子网。子网可以是固定子网、移动子网或者卫星子网。子网不具备任何行为，只是为了表示大型网络而提出的一个逻辑实体。

3. 节点 (Node)

节点通常被看作设备或资源，由支持相应处理能力的硬件和软件共同组成。数据在其中生成、传输、接收并被处理。Modeler 包含三种类型的节点：第一种为固定节点，例如路由器、交换机、工作站、服务器等都属于固定节点；第二种为移动节点，例如移动台，车载通信系统等都是移动节点；第三种为卫星节点，顾名思义是代表卫星。每种节点所支持的属性

也不尽相同，如移动节点支持三维或者二维的移动轨迹，卫星节点支持卫星轨道。

4. 进程 (Process)

进程是由有限状态机构成，可以理解成一个顺序执行的一系列函数。是构成协议运行的最基本结构。通过 c 语言编写实现进程模块，来控制 OPNET 中协议的运行，是网络仿真的主要部分。

5. 链路 (Link)

相对固定节点、移动节点以及卫星节点，链路也有不同的类型，有点对点的链路、总线链路以及无线链路。点对点的链路在两个固定节点之间传输数据；总线链路是一个共享媒体，在多个节点之间传输数据；无线链路是在仿真中动态建立的，可以在任何无线的收发信机之间建立。卫星和移动节点必须通过无线链路来进行通信，而固定节点也可以通过无线链路建立通信连接。

6. 仿真随机种子 (seed)

seed 是产生随机数的种子值，反映随机数的状态。只要选定一个种子值，整个随机事件系统就固定了，复杂仿真的随机过程就成了一次实现。目的是测试仿真系统的稳健性，具体来说，针对不同的 seed 值进行一系列仿真，每次不同 seed 值对应的仿真结果相近，则表明建立的模型有较高的稳健性 (scalability)。一般在发布仿真结果之前都要改变仿真种子进行多次测试，如果结果完全改变，则说明模块有疏漏，所得的结果只是一个特例，而不能反映系统的性能。只要随机种子不变，同一个仿真程序，在不同次实现的仿真结果也不变。包括在程序中使用产生随机数函数而产生的随机变量也不会因为实验次数的更改而更改。

7. 属性的隐藏 (hidden)

属性的隐藏使得属性的可读具有阶层性，如有些厂商设备的一些性能参数用户并不需要调节的，而为了避免用户混淆就把这些属性隐藏起来，变成预设值 (default value)，当需要时再去底层查找。

8. 属性的提升 (promoted)

与属性的隐藏相反，OPNET 规定等级低的参数可以不断提升 (promoted)，最后可变成级别最高的仿真属性。这种用法主要用在测试某个参数对网络仿真结果有何影响的场合，用户需要把在底层的参数提升出来就可以在仿真之前在仿真属性设置对话框中调整这些参数。

9. 模块 (module) 与仿真 (simulation)

对于某个协议的仿真，可能因为其涉及的事件及其相互的联系非常庞大，造成建模的困难，这时我们把该协议分解成一系列的协议行为，对这些行为单独建模后通过状态机把它们联系起来后便形成一个系统，这个系统可以称之为模块，它将抽象的协议直观化。而仿真是一系列模块的一组实验，它反映模块和模块之间的互相作用关系。

10. 对象 ID (Objid) 与用户 ID (user id)

Objid 是对象识别号系统分配的，全局唯一，整数。user id 是节点模型（对象的一种）的一个属性，由用户设置，可以不唯一。

11. 传输时延 (transmission delay) 与传播时延 (propagation delay)

传输时延又称为数据包传输时延，为数据包经过收发信机的时间长度，等于数据包第一个比特离开发信机到最后一个比特离开发信机之间的时间长度，与发信机比特率有关。传播时延又称为链路传播时延，为数据包在链路中传输需要的时间，等于第一个比特离开发信机到第一个比特到达接收机之间的时间长度，与收发信机之间的距离有关。

12. 模块 (module) 与模型 (model)

我们习惯称一个对象对应的点叫模块，而这个对象包含属性、内部构造、与其他对象的关联等参数。内部构造被称为模型，模型是由各个内部单元模块和内部模块的属性构成。

模块为对象，模型为模块内部结构的模板。模型作为模块的一个属性。一个模块可以选

择不同的模型，但是只能选择一个模型。模型可以被不同的模块选择。节点模块包含节点模型和其他节点属性。节点模型中又有其他内部模块和属性设置组成。

13. 仿真核心 (Simulation kernel)

在不少的 OPNET 教程中我们都可以看到仿真核心的词语，仿真核心，即系统函数，非自定义事件。比如中断的创建就分为用户创建和仿真核心创建。用户创建，即用户使用代码编程设定。仿真核心创建，即 OPNET 的软件系统中将代码写好了，会自动运行。

14. 仿真序列 (Simulation Series)

OPNET 网络仿真中为了测试参数配置对网络的性能影响有两种方法，1 可以复制一个场景，重新设置参数，同时运行两个仿真场景，2 设置仿真序列，每一个仿真序列都使用相同的场景配置不同的参数。包括配置仿真的随机种子，全局参数，子网属性等，这样就省去了复制场景的问题，同时节省了磁盘占用。但是需要注意的是，一个场景在仿真时会每个仿真序列都运行一遍，在复制场景时也会同时把仿真序列复制过去。

15. 仿真时间 (Simulation Time)

OPNET 仿真时间是针对实际时间而说的。OPNET 是基于事件机制进行的建模，时间仅作为一个参数，并不记录实际占用时间。OPNET 中的仿真时间，是自仿真开始到事件当前的时间。而时间长度与实际时间没关系。在 OPNET 的进程状态机中，在执行出入指令时不损耗时间。进程在挂起状态时，接收到中断，立即执行后续代码，指导再次被挂起，不轮实际仿真时进程是否损耗了时间，OPNET 认为期间不损耗仿真时间。

16. 探针 (Probes)

OPNET 中的探针，我们可以理解为一个指针，它指向一个对象，这个对象可以是统计量，属性，动画。这个对象有自己的属性设置和名称，用探针的方式记录这个对象的相关数据。在 OPNET 中包含全局统计量探针，节点统计量探针，链路统计量探针，路径统计量探针，业务统计量探针，属性探针，自动动画探针，统计量通话探针，自定义动画探针等

1. 4、OPNET 文件存储内容

OPNET 主要文件与对应记录内容如表 1-11 所示

表1-11 OPNET存储文件记录内容

文件格式	记录名称	文件格式	记录名称
xx.ac	分析配置文件	xx.nt.lib	网络模型生成的库文件
xx.ah	动画文件	xx.nt.m	子网模型，Network Model
xx.atc.m	ACE 文件	xx.os	输出矢量
xx.aed.m	ACE 白板文件	xx.ot	仿真文件
xx.bkg.i	背景图片	xx.ov	输出标量
xx.ef	环境文件	xx.pa.m	定向天线，Antenna Pattern
xx.em.c	EMA C 代码	xx.path.d	派生路径模型
xx.em.o	EMA 目标文件	xx.path.m	路径模型
xx.em.x	EMA 执行程序	xx.pb.m	探针模型，Probe Model
xx.esd.m	外部系统模型	xx.pd.m	概率密度函数
xx.ets	外部工具支持文件	xx.pdf.s	概率密度函数
xx.ets.c	外部工具支持 C 代码	xx.pk.m	数据包格式
xx.ets.cpp	外部工具支持 C++ 代码	xx.pr.c	进程 c 代码

xx. ex. c	外部 C 代码
xx. ex. cpp	外部 C++ 代码
xx. ex. h	外部头文件
xx. ex. o	外部目标文件
xx. fl. m	过滤器模型文件
xx. gdf	通用数据文件
xx. ic. m	ICI 格式文件
xx. icons	图标数据库
xx. lk. d	派生的链路模型
xx. lk. m	链路模型文件, Link Model
xx. map. i	地图
xx. mce. m	主机配置文件
xx. nd. d	派生的节点模型
xx. nd. m	节点模型文件, Node Model
xx. nt. dll	网络模型生成的动态链接库
xx. pr. cpp	进程 c++ 代码
xx. pr. m	进程模型, Process Model
xx. pr. obj	进程编译结果
xx. prj	项目, Project
xx. ppl. m	外形库文件
xx. ps. c	管道模型
xx. ps. cpp	管道模型
xx. ps. o	管道目标文件
xx. sa	STK 轨迹文件
xx. sce. m	服务器配置文件
xx. sd	仿真描述
xx. seq	仿真序列
xx. sim	可执行仿真
xx. trj	OPNET 轨迹文件
xx. wdomain. m	无线域模型

1. 5、OPNET 中英文对比

OPNET 中处理代码中打印输出可以识别中文外，其他地方无法识别中文，因此软件也均采用英文，表 1-12 详细介绍了 OPNET 中的常见硬件，并翻译成中文，便于读者理解。

表1-12 OPNET中英文翻译

英文	中文
altitude	海拔
Animation	动画
annotate	注释
Antenna	天线
Antenna Pattern	天线模型
AP, Access Point	接入点
Application	应用
association	关联
attribute	属性
bandwidth	带宽
begsim intrpt	仿真开始中断
BER, Bit Error Rate	误比特率
Bit-range	比特范围
bkpt, breakpoint	断点
blocking	停滞, 一般指进程在非强制状态中运行中断
Bulk size	包大小的校验值
boresight point	天线的基准点
Breakpoint	断点
BSS, Basic Service Set	基本服务子集

bucket	桶状收集，结果收集模式的一种
Capture Mode	收集模式
channel match	信道匹配
child	子对象
child process	子进程
Closure	物理可达性，链路闭锁
Compile	编译
Connectivity	连接关系
Connector	连接器，用来连接两个对象（如包流或链路）
Console	控制台
Coordinate	坐标系
Data rate	数据传输率
dbu, default bus	总线管道阶段文件的缺省前缀
delivery	传递
demand	背景流
Destroy	销毁
Dev, development	OPNET 仿真核心的一种，能够产生调试信息
Discover	一般指协议发现，通常在协议注册之后完成
Discrete event driven	离散时间驱动
Dpt, default point ts point	点对点管道阶段文件的缺省前缀
Dra, default radio	无线管道阶段文件的缺省前缀
Elapsed time	逝去时间
EMA, external model access	外部模型仿真
Endsim intrpt	仿真结束中断
Enter execs, enter executives	状态入口执行代码
ESS, external service set	扩展服务子集
Event	事件
Event list	时间列表
Event scheduler	事件调度器
Evhandle, event handle	事件句柄
Exit execs, exit executives	状态出口执行代码
Export	导出
External file	外部文件
Fan-in	群收
Fan-out	群发
FIFO, first In first Out	先入先出
Flow	流量
Flush	刷新，针对队列的一种操作
Forced state	强制状态
Formatted	有格式的，是包的另一种类型
Free space	自由空间模型，计算空间传播损耗模型的一种
Full range	缓存队列的总比特量
Gain	增益，一般指天线增益或处理机增益

Glitch removal	过滤毛刺，结果收集模式的一种
Global statistics	全局统计量
Header block	头块
HLA, High Level Architecture	高层体系架构
ICI, interface control information	接口控制信息
IMA, internal model access	内部模型访问
Import	导入
Individual statistics	单独显示，是结果显示模式的一种
Input stream	输入流
Install, installation	绑定，安装
Interface	接口
Interrupt	中断
Intrpt code	中断码
Intrpt mode	中断模式
Jammer	干扰机
KP, kernel procedure	核心函数
Label	标签
Latitude	纬度
Link	链路
Load	负载
Local statistics	本地统计量
Longitude	经度
Longley-rice	Longley 和 rice 两个学者提出，计算空间传输损耗模型的一种
MAC	信道接入控制层
Model	模型
Module	模块
Multi-tier	多端，值业务发送经过多台服务器
Normalize	归一化
Object palette	物件拼盘，对象模板
Objid, object ID	对象识别号
ODB, OPNET debugger	OPNET 调试器
Opt, optimize	优化的仿真核心
Orbit	轨道
Orindate	纵坐标
Output stream	输出流
Overlaid	重叠。结果显示模式中的一种
Packet	封包，包，分组
Packet field	包域
Packet header	包头
Parallel simulation	并行仿真
Parent	父对象
Parent process	父进程

Path	路径
Path loss	路径损耗
Payload	净荷
PDF	概率分布函数
Performance	网络性能
Pipeline stage	管道阶段
Plane	层
Platte	面板
Pmo, Pooled memory	池内存, 用来标识核心函数类别的前缀
Power lock	功率锁
Preference	属性, 一般指 OPNET 环境属性
Prg, programming	编程, 用来标示核心函数类别的前缀
Probe	探针
Process	进程
Process tag	进程标记
Processor, Process module	进程模块
Profile	业务主询, 业务规格
Prohandle	进程句柄
Project	工程, 项目
Promote	提升
Propagation delay	传播时延
Queue module	队列模块, 也可以称为进程模型
Reassembly	组装
Receiver	收信机
Reference point	天线的参考点
Register	注册
Root process	根进程
Round robin	轮循
Rxgroup	接收主询, 收信机组
Sample	采样, 结果收集的一种模式
Satellite	卫星
Sbhandle	分段缓存句柄
Scalar	标量
Scenario	场景
Schedule	调度
Seed	仿真种子
Segment	包段
Segmentation	分段
Signal lock	信号锁
Simulation kernel	仿真核心
Simulation time	仿真时间
SLA, service level agreements	服务等级
Slice	片

Smooth	平滑
SNR, signal-to-noise ratio	信噪比
Spreadsheet	数据表
Stacked statistics	统计量合并显示，是结果显示模式的一种
State variables block	状态变量块
Statistic wire	状态线
Stream	包流
Stream index	流索引，或流端口号
Subnet	子网
Subq, subqueue	子队列
SV, state variables	状态变量
Swap	交换，一般指队列中两个包的位置互换
TD, transmission data	传输数据
TDA, transmission data attributes	传输数据属性，用于管道阶段参数计算的传递信息
Temporary variables block	临时变量块
Throughout	吞吐量
Topology	拓扑
Trace	跟踪信息
Traffic	业务
Traffic profile	业务规格
Trajectory	轨迹
Transceiver	收发信机
Transmission delay	传输时延
Transmitter	发信机
Transition	状态转移线
TV, Temp Vriables	临时变量
Unforced state	非强制状态
Unformatted	无格式的，相对于 formatted 是包的一种类型
Unresolved externals	无法定位的外部函数
User id	用户识别号，节点模型的一个属性
Utility	物件拼盘中的特殊物件组合
Value vector	值向量，包类型的一种
Wireless domain	无线区域，用来划分接收主询
WLAN, wireless local area network	无线局域网

1. 6、OPNET 软件使用

1. 6. 1、菜单栏介绍

在子网模块界面，OPNET 菜单栏工具栏如图所示。

File Edit View Scenarios Topology Traffic Services Protocols NetDoctor Flow Analysis DES 3DNV Design Windows Help

第一部分：文件操作

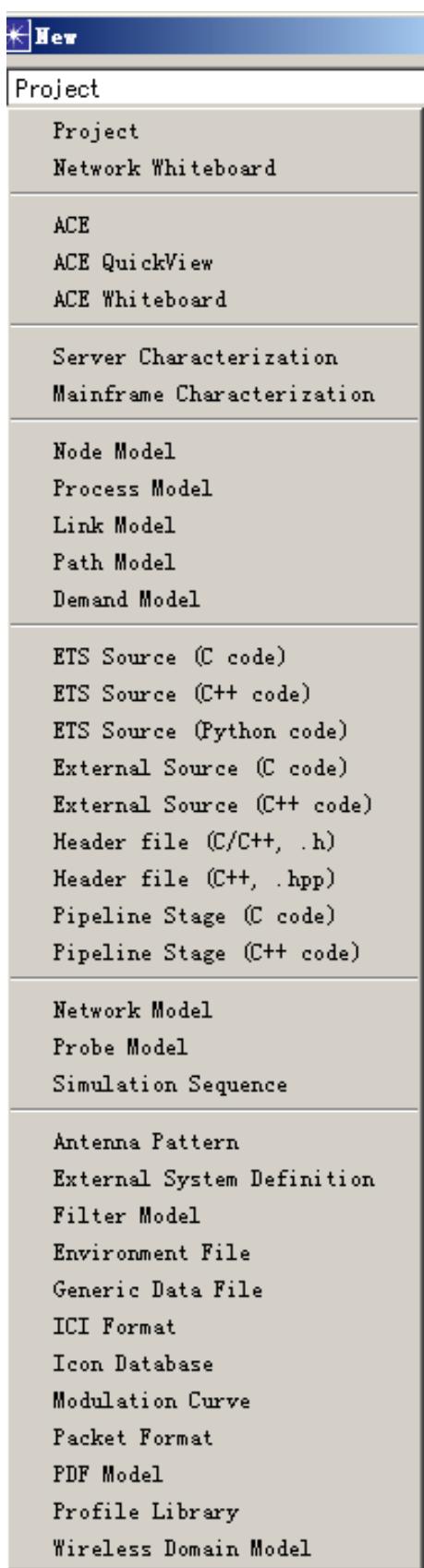
点击 File，弹出如图 1-9 所示。



New	创建内容
Open	打开内容
Close	关闭当前内容
Save	保存当前内容
Save AS	当前内容另存为
Page Setup	打印当前内容
Print	打印输出
Print All Subnets	打印输出子网
Declare External Files	声明外部文件
Automation	设置自动化任务
VNE Server Web Console	VNE 服务器 Web 控制台
Delete Projects	删除项目
Unlock Projects	开启项目
Manage Model Files	管理模型文件
Exit	退出

图1-9. File文件操作

(1) 点击 New，弹出新建内容窗口，会有多种创建内容选择。点击 Open 和 Save 以及 Save As 同样会有不同的文件格式类型。OPNET 文件、名称与功能对应如图 1-10 所示。



内容名称	新建内容
Project	项目
Network Whiteboard	网络白板（输出图片排版）
Server	服务器配置
Characterization	
Mainframe	主机配置
Characterization	
Node Model	节点模型
Process Model	进程模型
Link Model	链路模型
Path Model	路径模型
Demand Model	业务模型
ETS Source (C code)	ETS 的 C 代码
ETS Source (C++ code)	ETS 的 C++ 代码
ETS Source (Python code)	ETS 的 python 代码
External Source (C code)	外部文件 C 代码
External Source (C++ code)	外部文件 C++ 代码
Header file (C/C++)	C 语言头文件
Header file (C++)	C++ 语言头文件
Pipeline Stage (C)	C 语言管道文件
Pipeline Stage (C++)	C++ 语言管道文件
Network Model	网络模型
Probe Model	探针模型
Simulation Sequence	仿真序列
Antenna Pattern	天线方向图
External System Definition	外部系统定义
Definition	
Filter Model	滤波器模型
Environment File	环境文件
Generic Data File	通用数据文件
ICI Format	ICI 数据格式
Icon Database	图标数据库
Modulation Curve	调制曲线
Packet Format	数据包格式
PDF Model	概率密度模型
Profile Library	外形库
Wireless Domain Model	无线域模型

图1-10. OPNET创建对象

(2) 点击 Declare External Files, 可以进行外部文件声明。

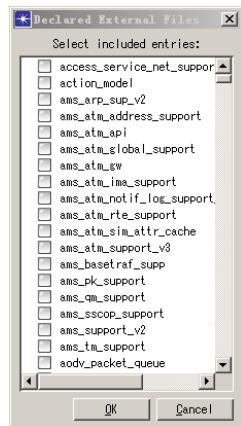


图1-11. OPNET外部模型库

声明外部文件主要针对，在c语言函数中调用了外部函数时，需要将调用的文件声明到当前进程中，如本书末尾实例2中，引入光纤链路时，需要声明部分外部文件，才能保证光纤链路的管道模型正常运行。声明只需在相应文件名选中即可。

(3) 点击 Manage Model Files 可实现模型文件管理。



图1-12. OPNET模型管理快捷操作

在模型文件管理中存在如图1-12所示的快捷操作。由于OPNET在打开时会按照所包含的模型目录扫面模型文件，在模型目录外的模型是不会被OPNET识别的。所以当我们创建了新的项目时，需要将存在模型的目录使用Add Model Directories添加到OPNET的模型目录中，这样，下次启动OPNET就会自动扫描其中的文件。当然还可以通过删除模型文件，更新模型目录等操作。

对模型目录的操作，还有一种方法，在Edit菜单栏中，选择preference，在输入框中输入mod_点击Find，出现如图1-13所示结果。

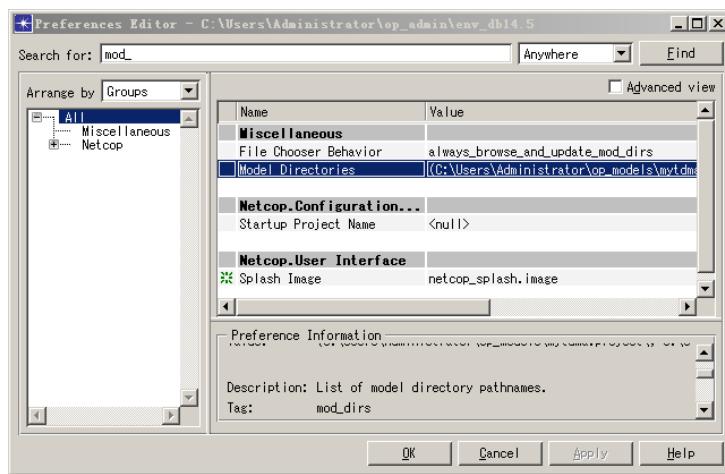


图1-13. 模型路径查看方法

点击Model Directories属性值的位置出现如图1-14所示的模型目录，在这里添加删除模型目录，调整模型目录顺序，第一个模型目录为默认打开文件对话框。

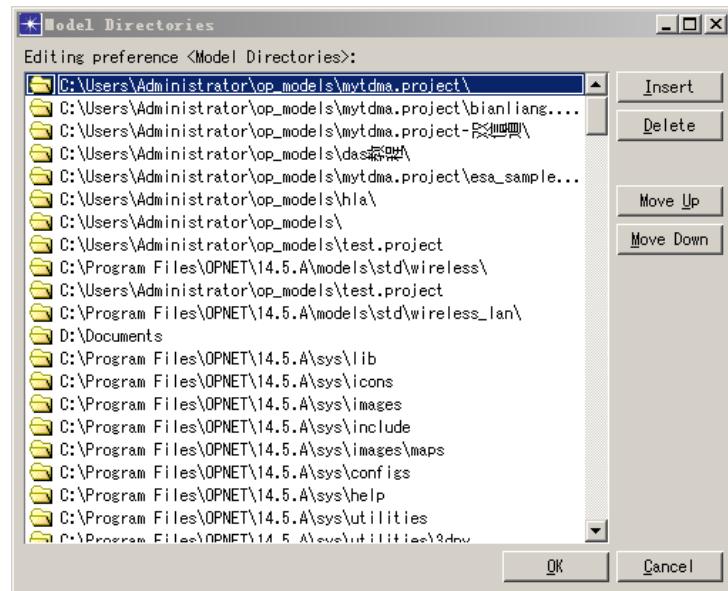


图1-14. 模型路径编辑界面

第二部分：场景操作

场景操作请参考 2.2 章节

第三部分：拓扑操作

拓扑操作请参照 2.8 章节

第四部分：DES 介绍

点击 DES，弹出如图 1-15 所示的菜单选项。

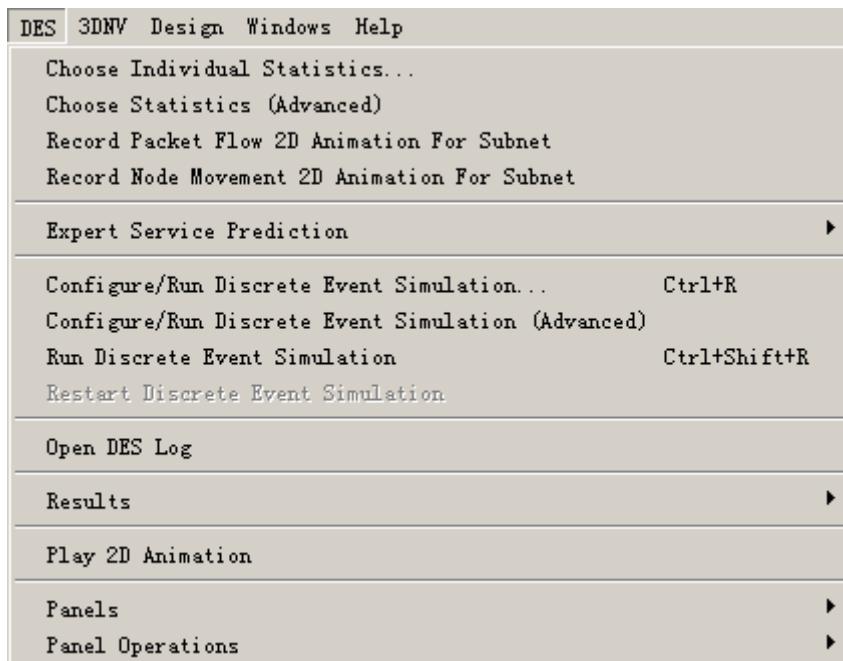


图1-15. DES菜单工具

1) Choose Individual Statistics 为选择自定义统计量，点击后进入统计量选择，可以选择全局统计量，节点统计量以及链路统计量。在子网模型空白处单击右键选择 Choose Individual DES Statistics 同样可以进入这个窗口。

2) Choose Statistics (Advanced) 为高级统计量设计，点击后进入各类型的探针设计界面。如图 1-16 所示，其中包含了 Global Statistic Probes 全局统计量探针，Node Statistics Probes 节点统计量探针，Link Statistic Probes 链路统计量探针，Path Statistic Probe 路径统计量探针，Demand Statistic Probes 业务统计量探针，Attribute Probes 属性探针，Automatic Animation Probes 自动动画探针，Statistic Animation Probes 统计量动画探针，Custom Animation Probes 自定义动画探针。

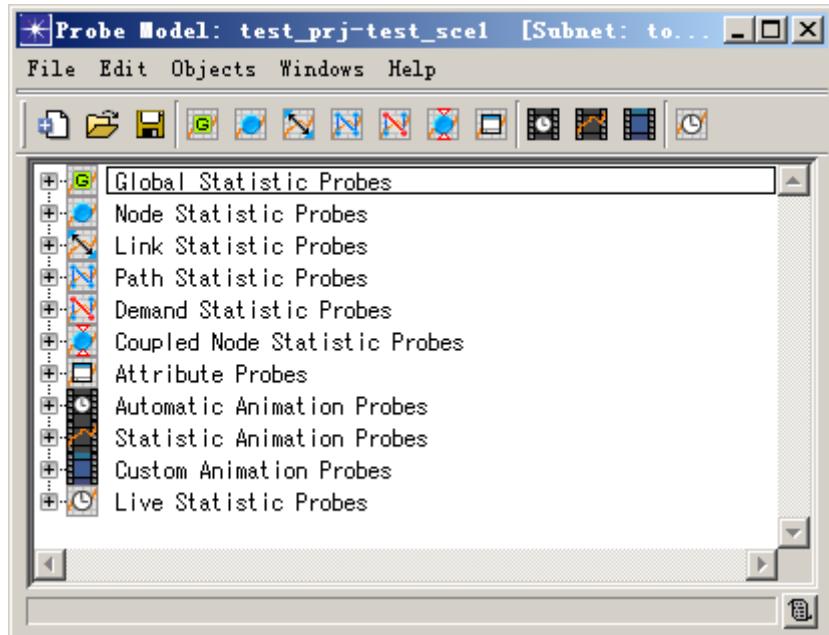


图1-16. 各类探针设置

3) Configure/Run Discrete Event Simulation 为仿真环境配置，点击后仿真环境配置界面。参考 4.1 章节调试部分。

4) Configure/Run Discrete Event Simulation (Advanced) 为高级仿真环境配置，参考 4.1 章节调试部分。

5) Run Discrete Event Simulation 运行离散时间仿真，等同于在仿真配置界面点击 Run 按钮，运行仿真。

6) Open DES Log 打开 DES 调试日志

7) Results 查看各种结果和调试报告

8) Play 2D Animation 为播放 2D 动画，在设置了动画探针，并在仿真输出中设置了存储动画文件时有效。

1.6.2、工具栏介绍

OPNET 场景工具栏如图所示。下面我们逐个来解释它们的作用分别是什么。



是新建文件，功能等同于菜单栏 File-New。是打开 op_models 下的项目文件，功

能等同于 File-Open,  用于保存文件, 功能等同于 File-Save。 是打印输出, 功能等同于 File-Print。 可以打开对象选择面板, 即可选择需要的节点模型、链路模型、路径模型等, 其中包含了 OPNET 中所有的独立对象, 其中很多根据业界 Cisco 等公司的实际设备来设计的, 能够精确直观地体现出使用这些设备所能达到的效果。  可成对使用, 选择网络场景中的一个对象, 然后点击 , 可以使该对象失效而不工作, 选中失效对象, 然后点击 , 可以使该对象恢复运行。若当前层为网络模型, 点击  将跳转到上层网络, 直至顶层“top”网络。  成对使用, 点击 , 即放大图标, 然后在网络模型中拖动鼠标指针即可放大场景, 再点击  可恢复到放大前的状态。   分别为 ACE 配置, CFG 硬件配置, VNE 服务器配置。 为流量配置,  为配置运行网络诊断,  为生成报告,  为配置运行流量分析,  为配置运行生存能力分析,  为运行仿真,  配置运行自动化任务,  为结果查看。

第二章 网络搭建

2.1、网络框架

OPNET 使用三层建模机制，一个场景最顶层网络均为一个叫做“top”的全球网络，如图 2-1 中 A 所示，TOP 网络中由各子网模块组成，如图 2-1 中 A 中圈中对象。每个子网模块内部对应各自的子网模型，如图 B 所示，不同的子网模块可以选择相同的子网模型。子网模型由节点模块，链路模块，轨迹模块，业务模块组成，节点模块如图 B 中圈中对象。每个节点模块内部对应各自的节点模型，如图 C 所示，不同的节点模块可以选择相同的节点模型。节点模型由处理器模块，包流，队列，收发信机，统计线，逻辑线，定向天线组成。处理器模块如图 C 中圈中对象所示。不同的处理器模块或队列模块可以选择相同的进程模型。进程模型如图 D 所示，进程模型由状态和状态转移线组成。状态包含阻塞态和非阻塞态。阻塞态为红色，非阻塞态为绿色。状态如图 D 中圈中对象所示。每个状态都包含入指令和出指令，用来编写代码，实现进程运行。点击状态的上半部分，进入入指令编辑窗口，点击状态的下半部分，进入出指令的编辑窗口。编辑窗口如图 E 所示。

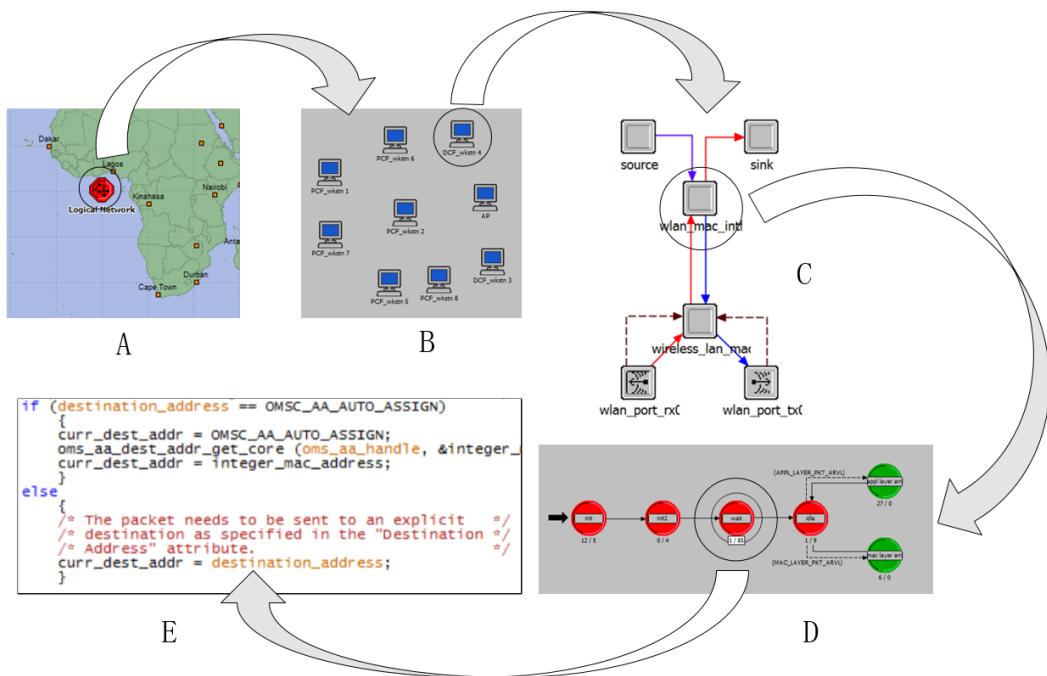


图2-1. OPNET网络框架

网络框架结构如图 2-2 所示。

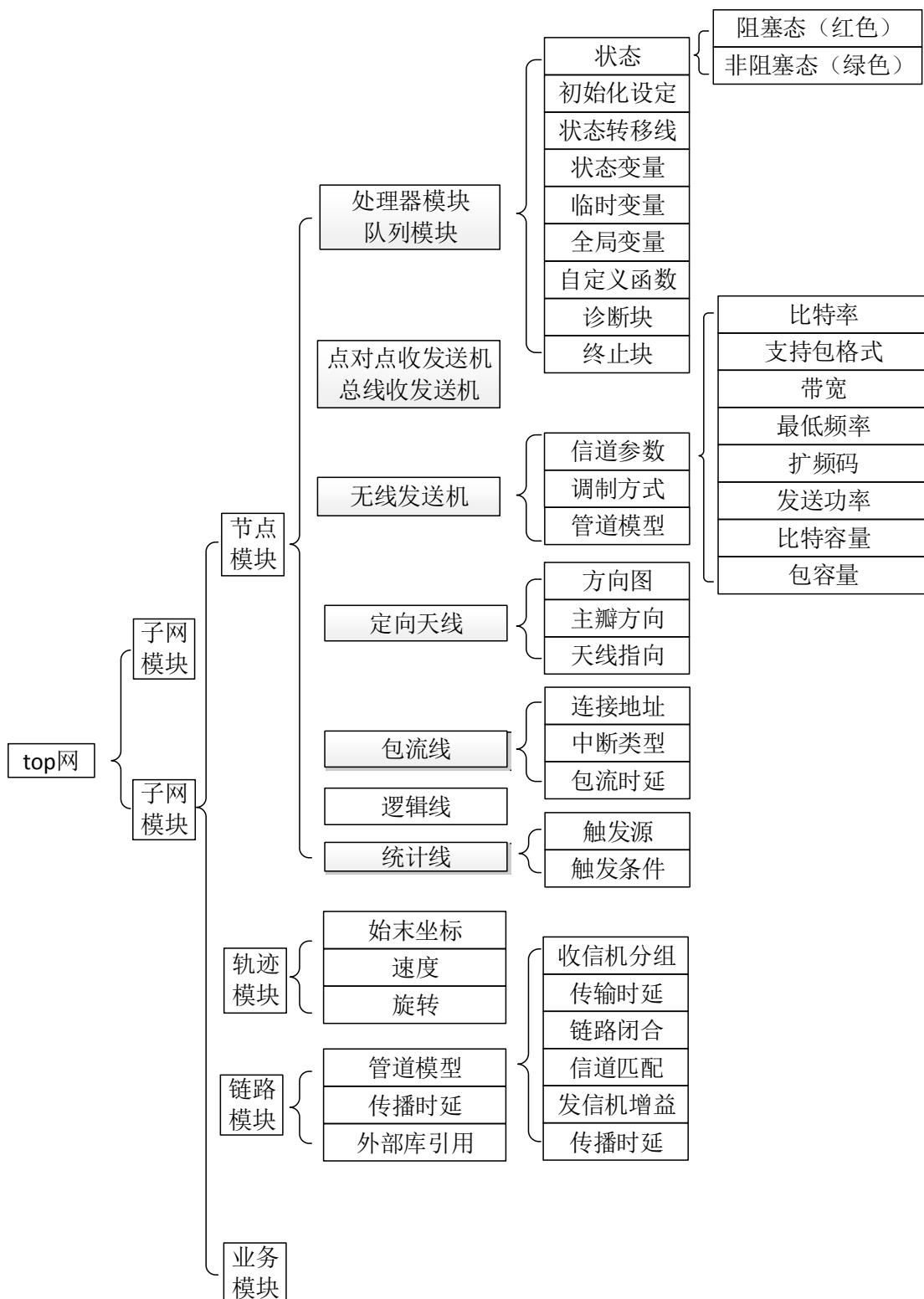


图2-2. OPNET网络框架设计结构

(1) 顶层网络

任何子网的网络顶层均为一个叫做“top”的全球网络。我们创建项目时设置的场景就

是 top 网络下的一个子网。可以通过点击工具栏的按钮^④来回到顶层网络。图 2-3 为两种不同样式的 top 网络。在顶层网络中我们可以设置子网模块的属性。其中包含子网模块的位置大小等。

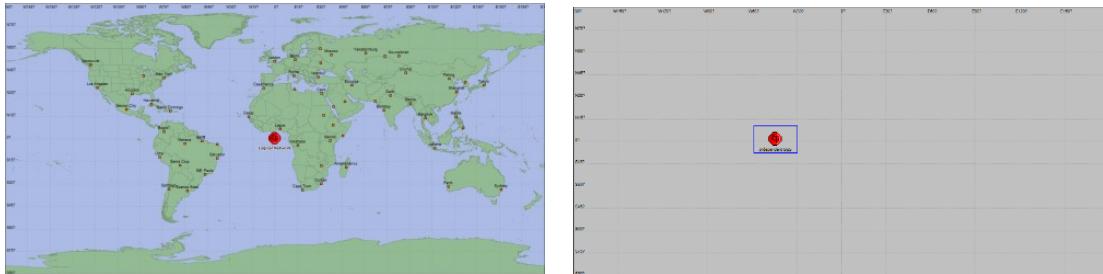


图2-3. OPNET顶层“top”网络示意图

(2) 子网模型

双击顶层“top”网络中的子网模块，进入子网模型。

子网模型用来模拟一个独立的网络，子网模型由节点模块、链路模块、轨迹模块、业务模块等组成。在子网模型中，我们可以设置子网的拓扑结构，业务需求，节点分布、链路类型、轨迹运行等属性参数。文件记录在项目的.nt.m 文件中。图 2-4 为两种不同样式的子网模型。

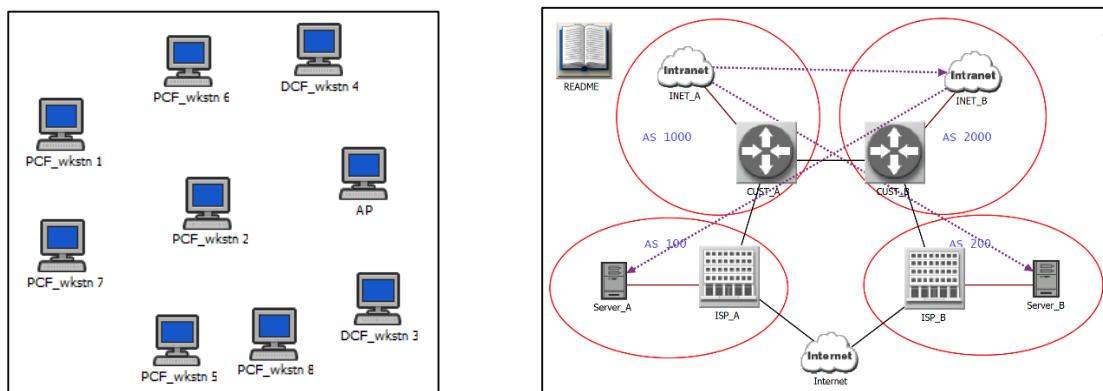


图2-4. OPNET子网模型示意图

(2.1) 链路模块

链路模块是用来仿真信道环境的，用来模拟数据在节点与节点间传输时的信号变化情况，包括时延、衰减、误码等物理层的信息。链路模块的设置包括链路的属性设置和一系列管道阶段函数。链路的属性设置为链路的参数设置，包括传播速度，比特率等，管道阶段函数为根据参数计算获得的一些物理层参数，如根据传播速度获取传播时延的管道阶段。

链路模块分为点对点链路、总线链路、无线链路。链路通过一系列管道模型实现物理层函数。物理层管道阶段函数为一系列的 ps.c 文件，可以编写 ps.c 管道阶段函数文件，作为不同的管道模型，管道模型可供链路模型在属性设置中选择使用。

有线链路模块是可视的，如图 2-4 中的右图所示，管道模型在链路模块属性中设置。无线链路模块是不显示的，如图 2-4 中的左图所示，管道模型在收发节点中的无线收发信机属性中设置。

(2.2) 轨迹模型

轨迹模型用来规定移动节点的移动轨迹，在动态网络中能很好的仿真节点的移动性，以及用户的加入和离开，在仿真路由协议和动态网络拓扑维护时需要使用。轨迹模型可以独立定义，存储为.trj文件，在节点的属性中设置使用。

对于移动节点的轨迹可以在OPNET中绘制或设置使用如图2-5，对于卫星节点的移动轨迹，由于其运行具有固定的规律性，OPNET只能从STK的轨迹文件中导入使用，如图2-6所示。

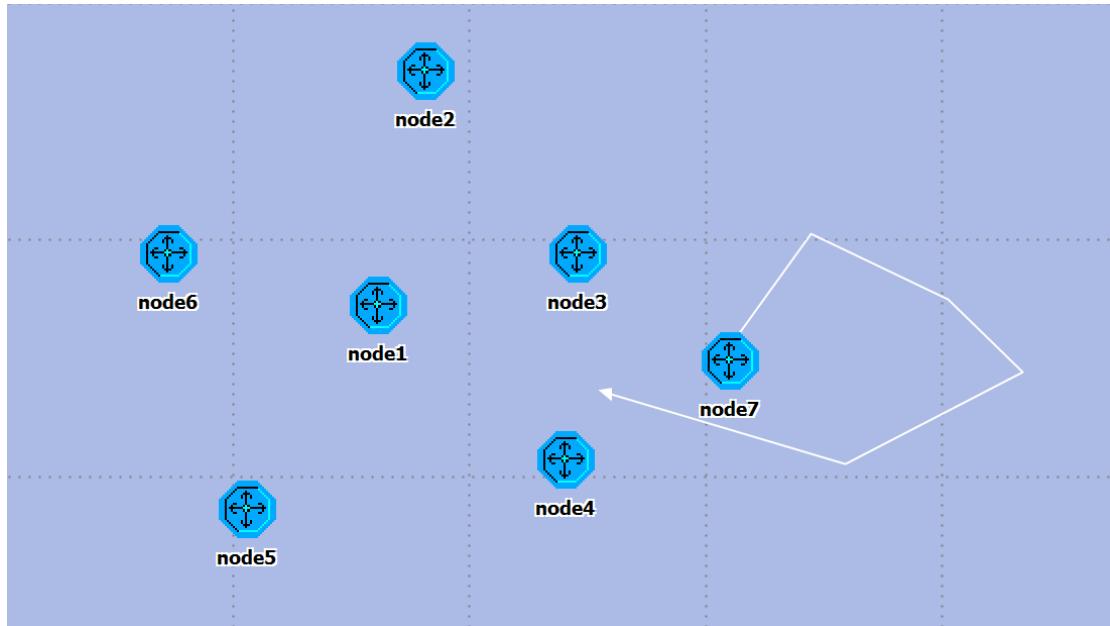


图2-5. OPNET轨迹模型示意图



图2-6. OPNET卫星轨道示意图

(2.3) 业务模块

干什么用？包含什么？怎么用？

(2.4) 节点模块

节点模块用来模拟网络中的用户，是组成网络的主要成员。节点模块可以设置自定义属

性，用于在代码中区分不同的用户或配置参数。节点模型可独立创建，存储成`.nd.m`文件，节点模块以修改属性的方式，选择不同的节点模型。双击子网模型中的节点模块，进入节点模型。

节点模型由处理器模块、队列模块、包流线、统计线、点对点收信机、点对点发信机、总线接收机、总线发信机、无线接收机、无线发信机、定向天线等组成。节点模型如图 2-7 所示。

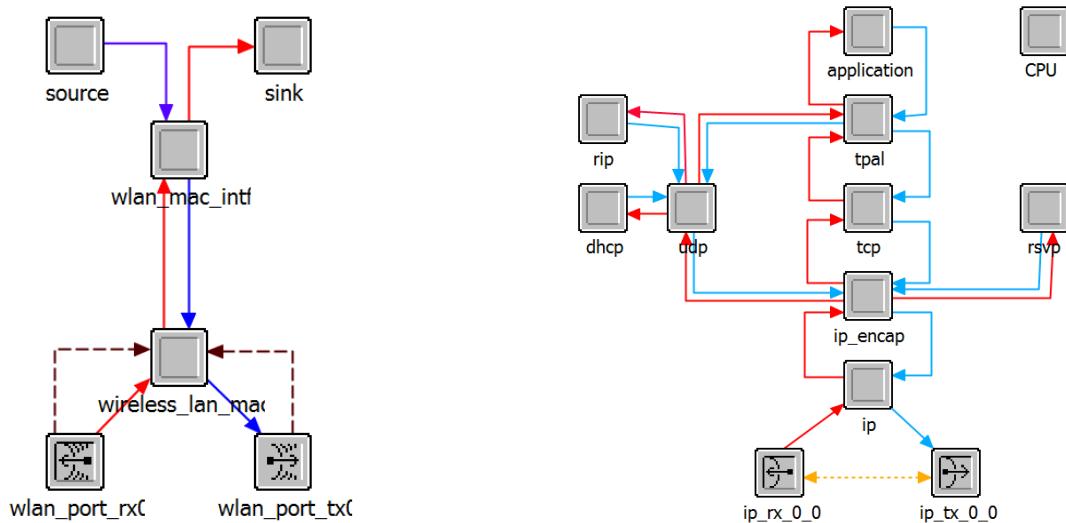


图2-7. OPNET节点模型示意图

- (1) 处理器模块、队列模块用来表示不同的网络协议层，他们可以使用相同的进程模型。队列模块相对于进程模块具有子队列缓存数据包的功能，不过在进程模型中通过 List 列表同样可以使用代码缓存数据包，更能灵活的处理数据包。
- (2) 包流线用来规定数据包的流向，用来表示数据包在硬件模块中的流动。通过包流线的索引号，可以在代码中确定数据包发送的走向。同时保留线也可以模拟在数据在硬件中流动时的时延与中断机制。
- (3) 统计线用来反馈数据模块数据给其他模块，主要应用来收发信机中反馈信息给 MAC 模块。通过统计线中断，实现处理器模块对收发信机状态的掌握，进而实现数据收发的有序性。
- (4) 收发信机用于节点间与节点间通信，通过不同个收发信机类型，表示实际中的线路类型。分为点对点收发信机，总线型收发信机，无线收发信机。并且收发信机与链路模型以及管道函数实现物理层仿真，由于上层协议往往需要物理层协议的基础，所以在不同类型的协议仿真中，往往都需要与收发信机打交道。
- (5) 天线模型仅用在无线通信网络中，需要与无线收发信机联合使用。无线收发信机默认情况下使用全向天线，而在长传输距离的大规模网络下，由于传输距离远往往需要定向天线实现功率节省。同时定向天线对网络结构的主要影响是使网状无线网转变为星状或点对点无线网。天线模型可独立创建，在存储后可供定向天线模块在属性设置中选择。

(2.4.1) 进程模型

双击处理器模块或队列模块，进入进程模型。一个进程模块等价于一个独立的进程。在一个进程内代码只能顺序执行。在 windows 程序中，一个进程中可以创建多个线程并发运行，在 OPNET 中并没有线程的概念。OPNET 一个处理器可以启动多个进程，但是同一时刻只能有一个进程处于执行状态。

进程模型由状态、状态转移线、初始化状态、状态变量、临时变量、头文件、自定义函数、统计量组成。在一个进程模型中，从初始化状态开始按照状态转移线的顺序依次执行每个状态内的入、出指令函数，进程中的状态用来表示电路中的有限状态机。状态转移图大大方便了使用者对协议的流程掌握。进程模型如图 2-8 所示。

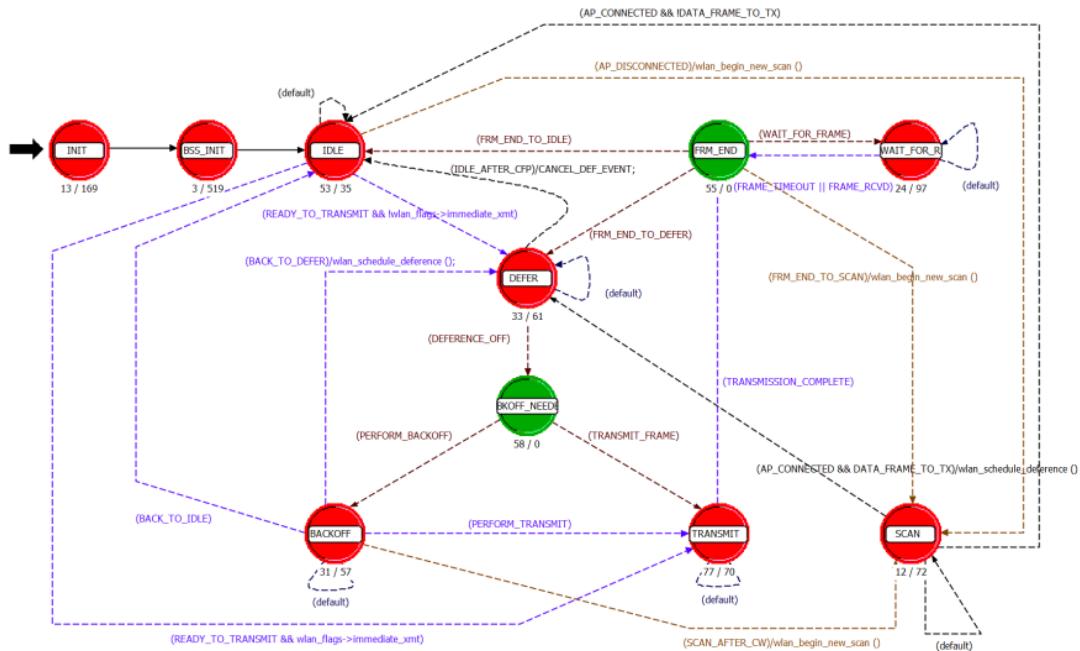


图2-8. OPNET进程模型示意图

示例：参见实例 1

2. 2、子网模型

项目中每个场景(scenario)都是一个子网模型，由与场景名称同名的.nt.m 文件记录。子网模型文件中记录了子网属性、节点参数、链路配置、轨迹属性、业务模型、拓扑结构等。一个项目至少包含一个子网场景。

子网模块在创建项目时会连带要求创建场景，即一个顶层网络为“top”网络的子网模块。OPNET 允许子网模块相互嵌套，在子网模块中可以包含另一个子网模块。如图 2-9 所示，默认子网模型可以设置内部子网模块和节点并存。同时可以对内部子网再进行设计

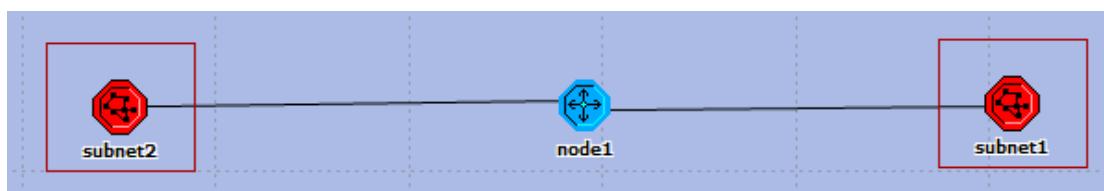


图2-9. OPNET多层子网示意图

点击菜单栏 topology, 选择 subnets 可以创建固定子网, 逻辑子网, 移动子网, 卫星子网, 或移动节点到父子网或选定子网中。

2.2.1、场景管理

由于协议的仿真系统均是以子网的形式存在, 所以我们常常涉及到对场景的操作, OPNET 提供专门提供了场景操作的菜单栏。点击菜单栏 Scenarios, 弹出场景操作快捷菜单, 如图 2-10 所示。

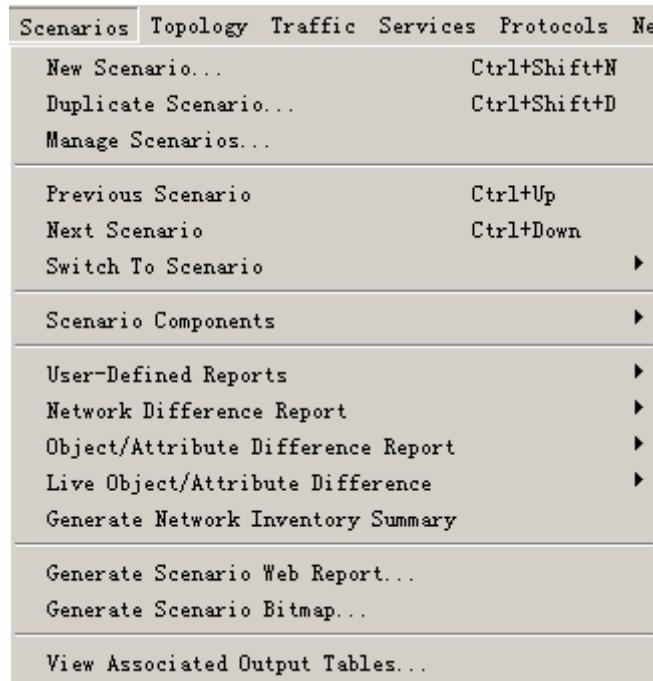


图2-10. OPNET场景菜单工具

- (1) New Scenario 创建新场景: 在创建项目时, 会同时要求创建一个场景, 在项目搭建以后我们希望创建其他的场景以实现系统性能的对比, 就需要创建新的场景, 点击 New Scenarios 按钮进入创建步奏, 创建过程同实例 1 中的步奏。
- (2) Duplicate Scenario 复制场景: 在获取仿真结果时, 我们总是希望获取协议中参数对性能的影响, 或不同协议的仿真性能。因此需要同时展示不同协议参数时的仿真结果图, Duplicate Scenarios 实现场景的完全复制, 仅重新对场景进行重命名。
- (3) Manage Scenarios 场景管理: 当仿真场景数目变多时, 场景的管理开始变的复杂。OPNET 提供了一个场景集中管理的配置界面。点击 Manage Scenarios 进入场景配置界面, 如图 2-11 所示。最左边栏为场景的排放顺序, 在场景数量较多时, 我们习惯按照一定的顺序排放场景, 以便快读定位所需场景的位置, 点击该栏的字段可以调整选中场景到其他位置。Scenarios Name 栏记录了场景的名字, 也是我们快读定位所需场景的主要参考参数。根据名称区别不同场景的主要差异是一个好的编程习惯。

Results 显示了场景的结果收集的命令。可选参数有 uncollected 不进行仿真结果收集, out of date 仿真配置更改, 未收集新的仿真结果, collect 进行仿真结果收集, discard 删除仿真结果, recollect 重新收集仿真结果, updata 已经是最新仿真结果。Sim Duration 和 Time Units 时间单位组成了仿真时间的设定。在点击 OK 之后, 系统会自动按窗口中的设置更改项目参数; 删除 Saved 栏设置为 delete 的场景; 仿真收集 Results 栏命令为 collect 和 recollect 的场景; 删除标志为 discard 的场景仿真结果。

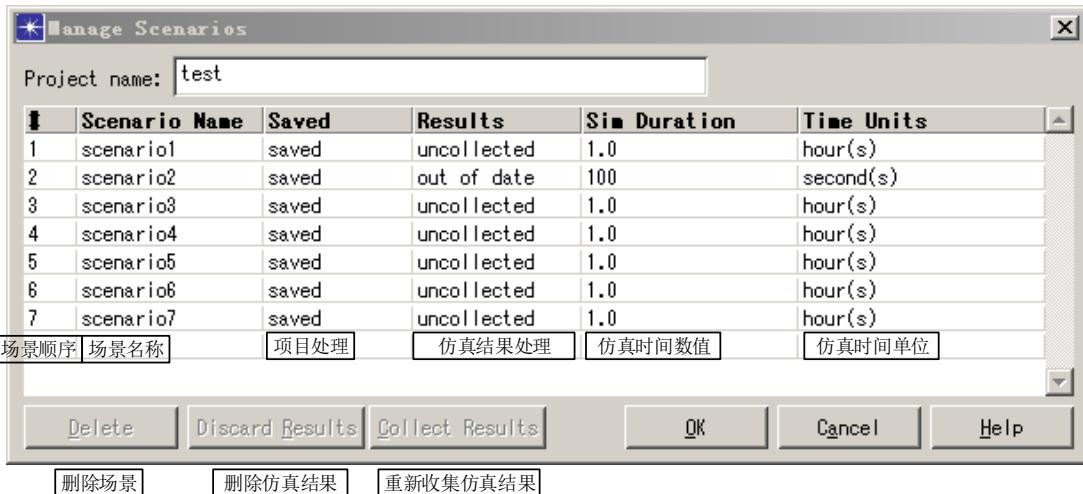


图2-11. OPNET场景管理

- (4) Previous scenario 切换到上一个场景, 同时可以使用快捷键 **ctrl+up** 实现。
- (5) Next scenario 切换到下一个场景, 同时可以使用快捷键 **ctrl+down** 实现。
- (6) Switch To Scenario 可以看到项目中的所有场景, 直接点击进行切换
- (7) Scenario Component 可以实现场景的导入导出。Import 导入功能可以直接利用 OPNET 中自带的协议库, 直接打开 OPNET 已经配置好的各类型协议仿真场景, 能大大减少编程时间。Export 能将自己配置的或修改后的场景导出到文件。
- (8) Generate Scenario Bitmap 生成场景图片。如图 2-12 所示。

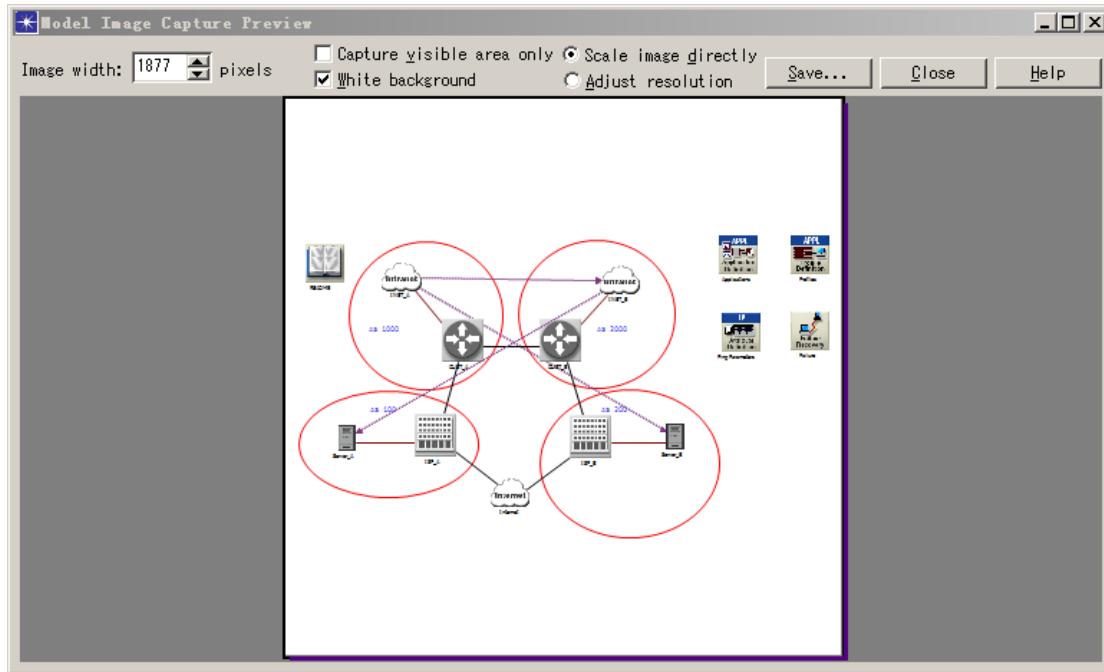


图2-12. OPNET子网模型输出

2.2.1、场景配置

(1) 配置子网属性

点击 \square 按钮，我们可以返回到顶层“top”网络。在子网模块单击右键，选择Edit Attributes(Advanced)可以进入子网模块的属性设置界面。

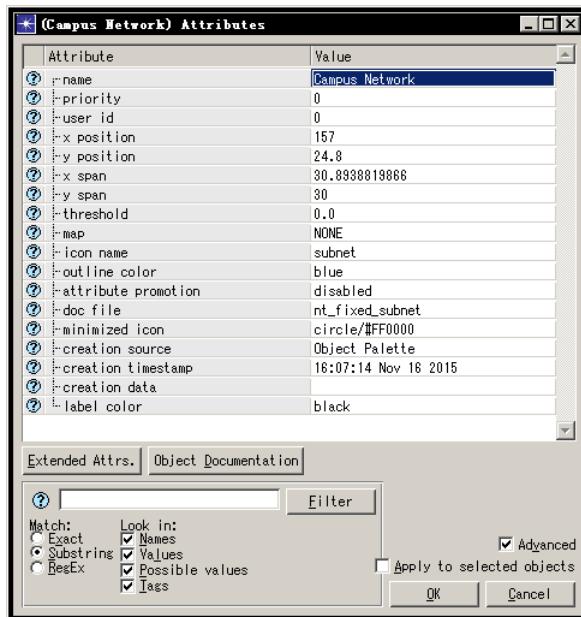


图2-13. OPNET子网模块属性设置

属性设置

表2-1. 子网模块属性内容

属性名	属性取值	备注
x position	网络中心点经度	东半球为正，西半球为负
y position	网络中心点纬度	北半球为正，南半球为负
x span	网络长度	单位 km
y span	网络宽度	单位 km

(2) 添加节点模块

不同类型的节点应用于不同的网络，OPNET 为不同的类型节点设置了不同的自定义属性、显示图标和对应的节点模型。使用者可以完全手动创建设计自己的节点，也可以导入 OPENT 自带的节点模型进行修改使用。

双击子网模块，进入子网模型域，点击工具栏上模型选择按钮 ，进入模型选择窗口。在模块选择窗口我们可以选择节点模型（包含固定节点、移动节点、卫星节点），可以选择链路模型（包含双工链路，单工链路，总线链路），可以选择路径模型，可以选择业务模型，以及共享的对象调色板。

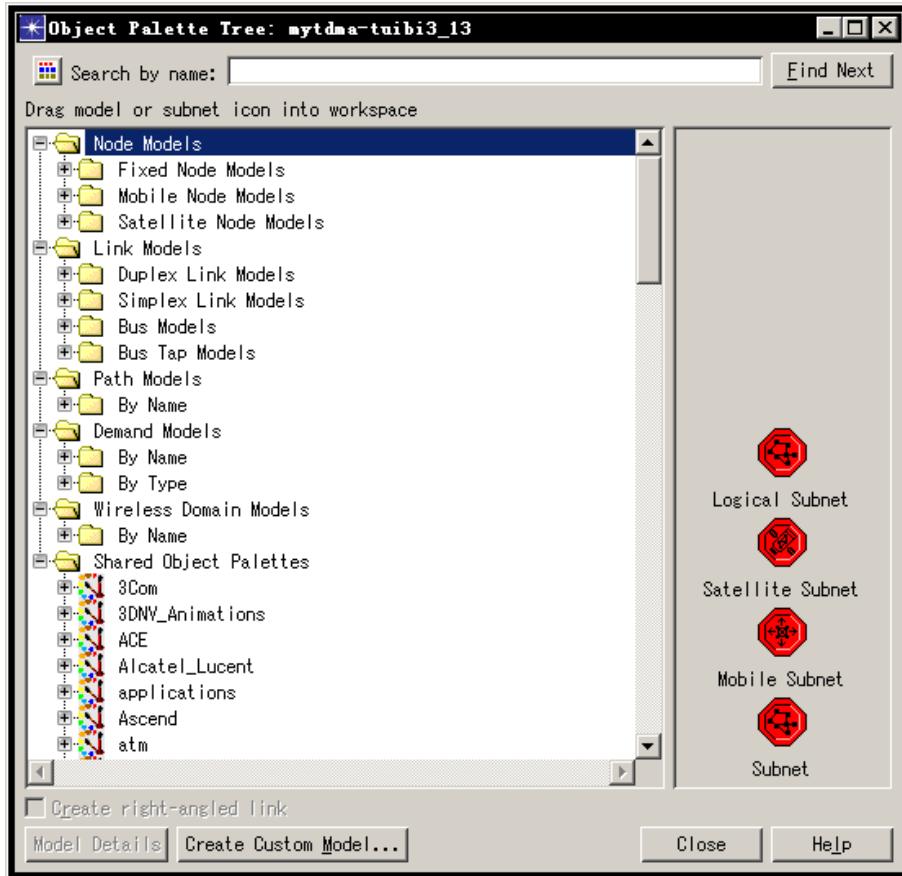


图2-14. OPNET对象插入窗口

添加通用固定节点的方法：通过在模型选择窗口的目录树中，选择 Node Models—Fixed Node Models—By Machine Type—utility—pstn_flan_config，显示图标为 。拖动图标

到子网场景中，即可在子网中创建一个通用的固定节点模型。

添加通用移动节点的方法：通过在模型选择窗口的目录树中，选择 Node Models—Mobile Node Models—By Machine Type—utility—pstn_flan_config，显示图标为。拖动图标到子网场景中，即可在子网中创建一个通用的移动节点模型。

添加通用卫星节点的方法：通过在模型选择窗口的目录树中，选择 Node Models—Mobile Node Models—By Machine Type—utility—pstn_flan_config，显示图标为。拖动图标到子网场景中，即可在子网中创建一个通用的卫星节点模型。

(3) 添加链路模块

不同类型的链路模型应用于不同物理层传输模式的网络，在添加完节点模块后，需要添加链路模块来实现节点与节点间的传输。点击工具栏上模型选择按钮，进入模型选择窗口。我们可以添加双工链路 Duplex Link Models，单工链路 Simplex Link Models，总线 Bus Models，总线抽头 Bus Tap Models。

(4) 设置节点轨迹

只有卫星节点和移动节点能设置轨迹。移动节点可以使用分段式移动模型、向量式移动模型和移动模块配置的方式，其中分段式移动模型参照 2.5.1-2.5.3 章节，向量式移动模型参考 2.5.6 章节，卫星节点的轨道由于必须符合一定的物理规律，所以只能导入 STK 软件生成的轨道文件，参考 2.5.5 章节。

示例：参考实例 2

2.3、节点模型

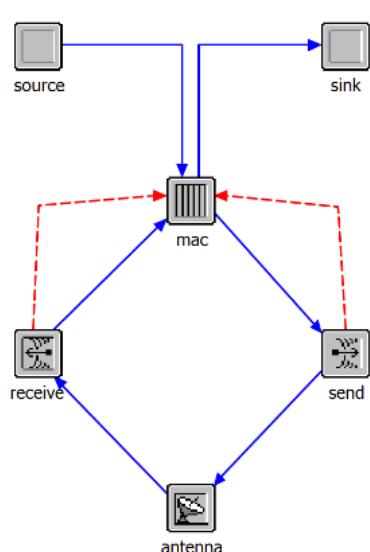


图2-15. 节点模型示意图

节点模型中的对象包含：模型属性、节点接口、节点统计量、包流线、统计线、逻辑线、处理器模块、队列模块、点对点发信机、点对点收信机、无线发信机、无线收信机、总线发信机、总线收信机、定向天线。另外还可以对节点模型接口，中断属性，统计量等进行设置。节点模型中的各个模块通过三种链接互连，分别是：包流，统计线，逻辑关联。包流传送格式化的消息，我们称为数据包。统计线传送简单的数字信号和控制信息，一般用于一个模块用来监测其他模块的性能和状态。统计线和包流都有参数用来配置它们的行为。逻辑关联用来确认模块之间的绑定。目前，它们只被用在发送器和接收器之间，用来表明它们在接入到网络域上的链路时，应当被当作一对模块来使用。节点模型如图 2-15 所示。

节点模块通过 model 属性值修改选择不同的节点模型。所以节点模块的属性不存储在节点模型.nd.m 文件中，而是存储在子网模型.nt.m 文件中。这里的讲解隶属于子网模型部分，同时隶属于节点模块部分，所以和节点模型一同在这里讲解。

右键单击节点模块，选择 Edit Attributes (Advanced) 进入属性编辑窗口。不同类型的节点模块所具有的属性字段不同。节点属性主要包括 name 名称、model 节点模型、x position, y position 坐标、color 颜色、设置为显示的节点接口，节点模型属性、节点模块自定义属性。

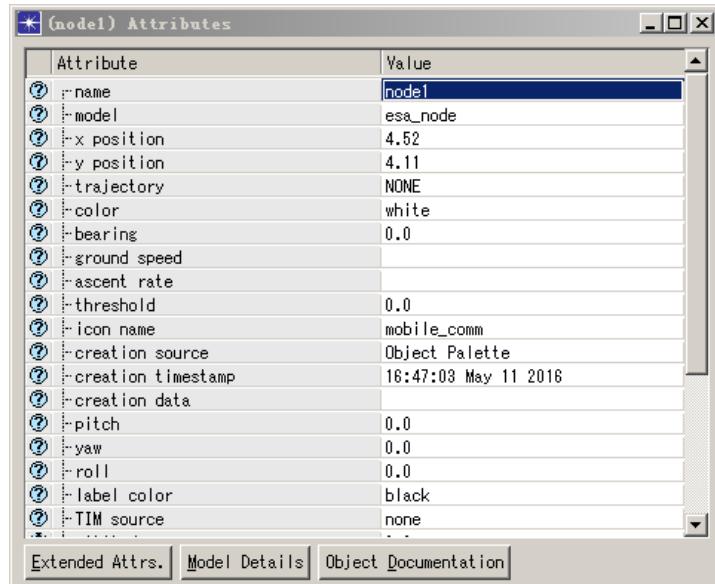


图2-16. 节点模块属性窗口

(1) 节点模块自定义属性

在节点模块属性编辑对话框，单击 Extended Attrs. 按钮进入扩展属性编辑界面。这里可以定义属性的名称、分组、变量类型、单位、默认值等。“名称”用于在属性中显示和代码中通过名称获取属性值。“分组”用于多个属性的方便管理，在代码中可以视为分组不存在。“变量类型”表征数据的类型，代码中需要使用对应的类型读写，否则会报错，可以使用的变量类型包括 integer 整型、double 双浮点型、string 字符串、toggle 开关、typed filed、compound 混合属性、profile 外形、color 颜色。“单位”只是用于在属性中显示，没有其他用处，OPNET 是不能自动进行单位转化的。“默认值”的设定是为了避免在使用时没有初始化值而报错。点击 OK 按钮，节点模块的属性列表中按照不同分组添加了自定义属性。

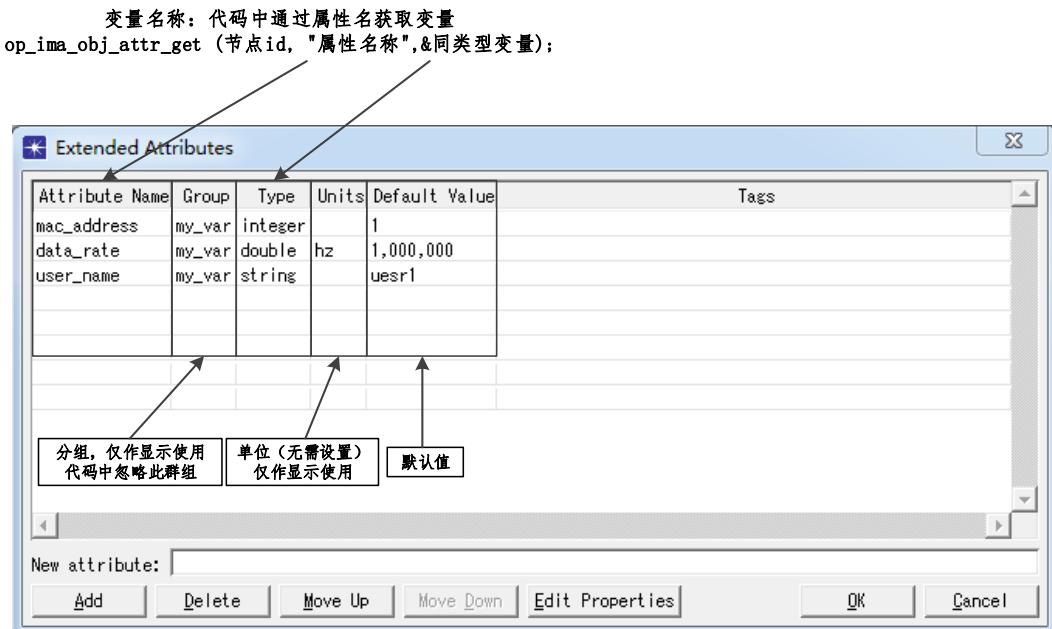


图2-17. 节点模块自定义属性

(2) 设置模型属性

在进行仿真时，有些参数往往需要经常修改设置，我们习惯将这些参数处理成模型属性，在代码中获取，这样我们就可以很方便的修改参数的值。进入节点模型界面，选择菜单栏 Interfaces，选择 Model Attributes 选项，进入属性编辑界面。

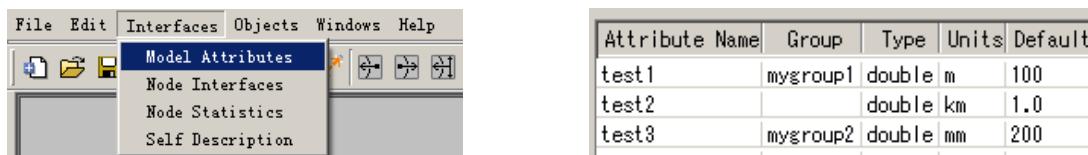


图2-18. 节点模型属性设置

模型属性的编辑同模块自定义属性。包含属性名，分组，变量类型，单位，默认值。属性设置之后可以在上层模块属性中查看到设置的自定义属性，但是这种方法定义的模型属性无法在点击 Extended Attrs. 按钮进入的扩展属性编辑器中修改。并且通过这种方式设置的分组，在代码中获取时，需要先获取分组对象的 ID，再获取属性对象的 ID。

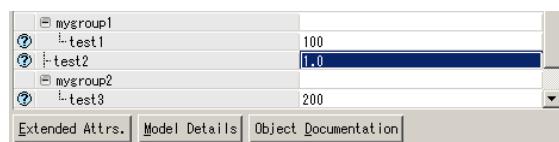


图2-19. 节点模型属性设置效果

(3) 设置节点接口

进入节点模型界面，选择菜单栏 Interfaces，选择 Node Interfaces 选项，进入节点接口编辑界面。可以在 Comments 输入 ESA 联合仿真命令，在 Keywords 管理节点关键词，在 Node types 设置节点对固定模式、移动模式、卫星模型是否支持，在 Attributes 设置属性

是否显示和属性初始值。将属性设置为 hidden 时，则该属性不会在节点模块右键属性中查到，设置为 promoted，则该属性会被提升到高层网络。使用 set，该属性会被显示在节点模块右键属性中。使用 Rename/Merge 可以修改自带属性的名称和设置。

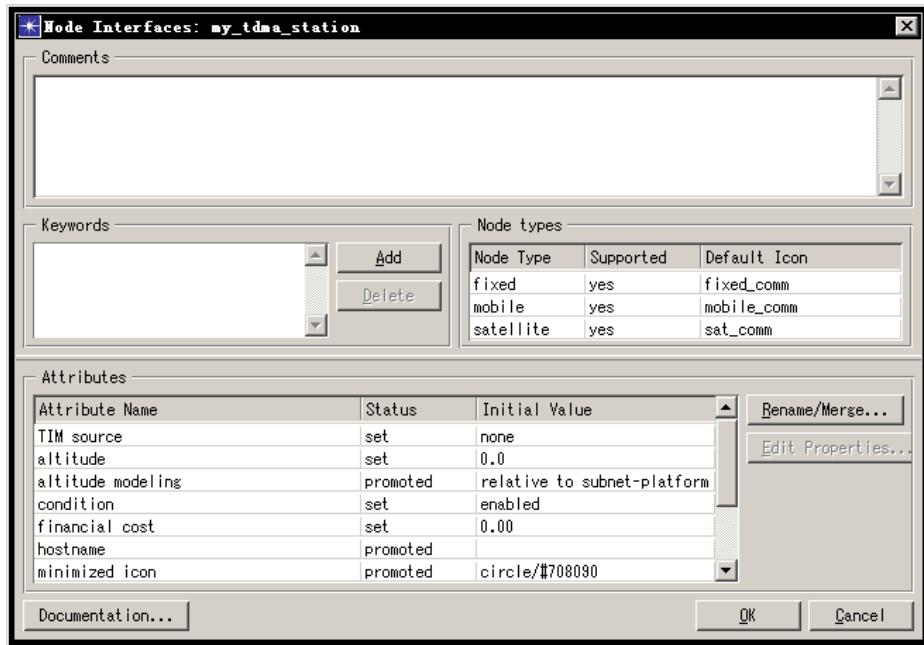


图2-20. 节点接口设置

(4) 其他模块

关于其他模块我们在后面的章节重点讲解。节点统计量的使用操作 3.7 章节讲解，处理器模块的使用在 2.3.1 章节讲解，队列模块的使用在 2.3.1 章节讲解，包流线的使用在 2.3.2 章节讲解，统计线的使用在 3.7 章节讲解，收发信机的使用在 2.3.3 章节讲解，定向天线的使用在 2.3.4 章节讲解。

示例：参考实例 3

2.3.1、队列与处理器

处理器模块和队列模块时进程协议算法实现的主要模块，因为只有在处理器模块和队列模块中才可以包含进程模型。

(1) 处理器模块

右键点击处理器模块弹出的快捷菜单包含三个设置：Edit Attributes 属性编辑、Set Name 名称修改、Show Connectivity 查看连接。单击属性编辑快捷栏，进入属性编辑界面。处理器模块属性主要包括名称，进程模型，图标名称，以及进程接口。



图2-21. OPNET处理器模块属性设置窗口

(2) 队列模块

右键点击队列模块弹出的快捷菜单包含 Edit Attributes、Set Name、Show Connectivity 三个设置。队列模块的模块属性包含如表 2-2

表2-2. 队列模块属性设置

属性名	属性值	备注
name 名称设置	符合变量名称规范	
process.model 进程模型	状态转移文件. pr.m	尽量不使用自带文件，需要时复制文件更改名称后再用。
icon.name 图标样式	图标显示图案	并不更改模块类型，仅改变显示效果
bit capacity (bits) 比特容量	整型, infinity 为不限制	缓存比特数目超过时溢出
pk capacity (pk) 包容量	整型, infinity 为不限制	缓存包数目超过时溢出

队列模块相对于处理器模块，具有数据包自动排队功能。队列模块内子队列结构图如图 2-22 所示。

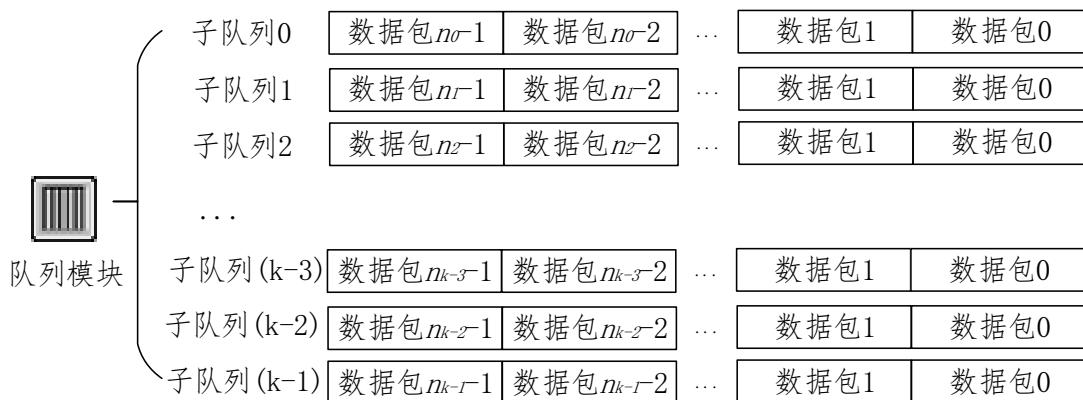


图2-22. OPNET队列模块包缓存结构

队列模块包含多个子队列，也就具有一系列的队列操作函数的使用，如表 2-3 所示。

表2-3. 队列操作常用函数

`op_subq_pk_insert:`

函数	<code>int op_subq_pk_insert (int subq_index, Packet* pkptr, int pos_index)</code>
函数功能	将 pkptr 所指向的数据包插入到 subq_index 队列的 pos_index 处。队列从 0 开始，依次增加，pos_index 可以是 OPC_QPOS_HEAD, OPC_QPOS_TAIL, 以及 OPC_QPOS_PRIO。返回插入情况，可以是 OPC_QINS_OK, OPC_QINS_FAIL, OPC_QINS_PK_ERROR, OPC_QINS_SEL_ERROR。
举例	<code>op_subq_pk_insert (strm_index, pkptr, OPC_QPOS_TAIL);</code>
解释	//将索引号为 strm_index 的子队列的尾部插入 pkptr 指向的数据包

`op_subq_empty:`

函数	<code>Boolean op_subq_empty (int subq_index)</code>
函数功能	判断队列是否为空。是空返回 OPC_TRUE，否则返回 OPC_FALSE。
举例	<code>if(op_subq_empty (int subq_index))</code>
解释	//如果索引号为 subq_index 的子队列为空

`op_subq_pk_remove:`

函数	<code>Packet* op_subq_pk_remove (int subq_index, int pos_index)</code>
函数功能	删除指定队列的指定数据包，并返回指向该删除数据包的指针。pos_index 可以是：OPC_QPOS_HEAD, OPC_QPOS_TAIL, and OPC_QPOS_PRIO。
举例	<code>pkptr = op_subq_pk_remove (subq_index, OPC_QPOS_HEAD);</code>
解释	//从索引号为 subq_index 的子队列的头部移除数据包到指针 pkptr。

`op_subq_pk_access:`

函数	<code>Packet* op_subq_pk_access (int subq_index, int pos_index)</code>
函数功能	返回子队列索引 subq_index 所向的队列的指定位置 pos_index 处的数据包的指针
举例	<code>pkptr = op_subq_pk_access (subq_index, 3);</code>
解释	//读取索引号为 subq_index 的子队列的第 4 个数据包到指针 pkptr。

`op_subq_sort:`

函数	<code>void op_subq_sort (int subq_index)</code>
函数功能	将子队列按包相关的优先级排序。优先级越大，越靠近队首
举例	<code>op_subq_sort (subq_index);</code>
解释	//将索引号为 subq_index 的子队列中的数据包按相关优先级排序。

`op_subq_stat:`

函数	<code>double op_subq_stat (int subq_index, int stat_index)</code>
函数功能	获取指定子队列的指定统计量值。发生错误返回 OPC_DBL_INVALID
举例	<code>num_pkts = op_subq_stat (i, OPC_QSTAT_PKSIZE);</code>
解释	//返回索引号为 i 的子队列的数据包个数。

`op_subq_flush:`

函数	<code>void op_subq_flush (int subq_index)</code>
----	--

函数功能	清空指定子队列中全部数据包
举例	op_subq_flush (subq_index);
解释	//清空当前进程模型索引号为 i 的子队列中的数据包。

op_subq_pk_swap:

函数	void op_subq_pk_swap (int subq_index, int pos_index1, int pos_index2)
函数功能	交换指定子队列中两个位置上的包的顺序
举例	op_subq_pk_swap (subq_idx, OPC_QPOS_HEAD, OPC_QPOS_TAIL);
解释	//交换子队列 subq_idx 中头部和尾部的数据包。

op_subq_print:

函数	void op_subq_print (subq_index)
函数功能	打印指定子队列中的包信息，包括平均包长，包 ID，地址，优先级，插入时间等
举例	op_subq_print (0);
解释	//打印 0 号子队列中的包信息。

2.3.2、包流

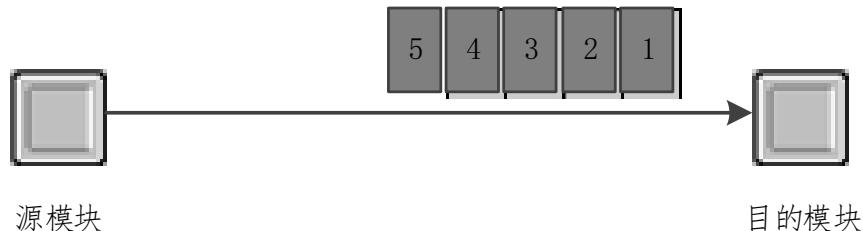


图2-23. OPNET包流缓存结构

在图 2-23 表示了模块间通过包流传输数据包结构，数据包发送和分离时独立的。源模块发送数据包后，包流根据属性设置，决定将包发送到目的模块的时间和引起的中断类型。目的模块主动捕获数据包，才能将数据包从缓存中取走。源模块和目的模块间存在缓存，这种缓存是设置在目的模块端的。如图 2-23 所示的 5 个数据包是存储在目的模块内的，包流本身没有缓存，只表示时延和发起中断。OPNET 节点模型使用包流实现模块间的数据包交互，右键单击模块选择 Show Connectivity 可以查看模块间的包流索引。数据包的传输通过包流索引决定数据包走向。如图 2-24 所示。表示 user_mac 模块的输出 0 号索引可以向 send 模块的输入 0 号索引发送数据包，user_mac 模块的输出 1 号索引可以向 user_sink 模块的输入 0 号索引发送数据包，user_source 模块的输出 0 号索引可以向 user_mac 模块的输入 0 号索引发送数据包，receive 模块的输出 0 号索引可以向 user_mac 模块的输入 1 号索引发送数据包。

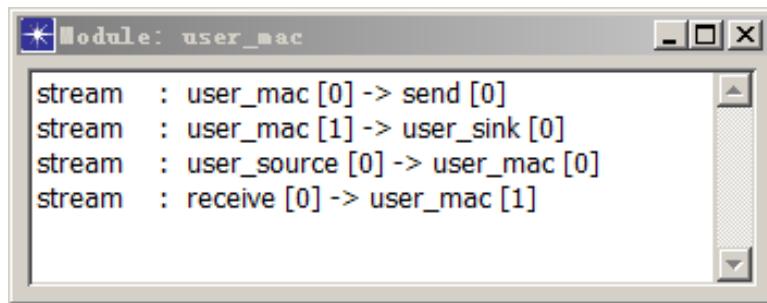


图2-24. OPNET关联连接图

包流也往往和数据包的传输一块使用，节点内数据包沿包流方向传输。包流具有数据包缓存和流中断控制的功能，源模块发送数据包总是先发送到包流缓存中。目标模块会按照不同的发送模式接收到流中断，目标模块接收到流中断后再从包流缓存中读取移除数据包。详细介绍请参照 xx 章节，数据包传输。数据包发送方式包含静默式，强制式，调度式。

点击包流，右键选择属性编辑，可以进入包流属性设置窗口如图 2-25。其中中断方式的设置包含 quiet、scheduled、forced。中断方式属性的设置并不起作用。delay 时延的设定在使用 op_pk_send_forced() 函数时无效。当包流属性中的 delay 设置为 0 时，数据包在模块间传输不损耗时间。包流属性中的 src stream[n] 和 dest stream[n] 分别表示源模块的此条包流线所占的源模块的输出包流索引号与目的模块的输入包流索引号。

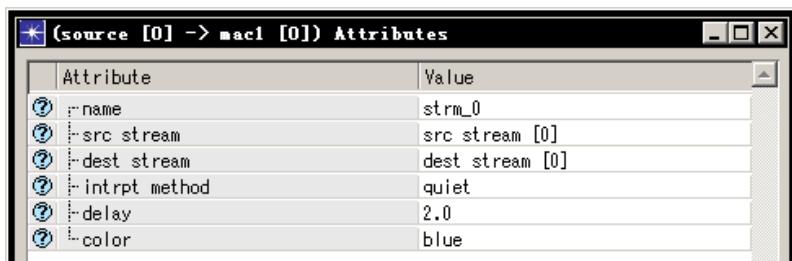


图2-25. OPNET包流属性设置

这里介绍一下关于包流的相关函数及使用说明，如表 xx 所示。

表2-4. 包流操作常用函数

op_strm_flush():

函数	void* op_strm_flush(int instrm_index)
功能	清除输入流中数据包
举例	op_strm_flush (0); //清除索引号为 0 的包流中缓存的数据包

op_strm_max_index_in():

函数	int op_strm_max_index_in()
功能	获得任何与周围进程或队列相连接的输入流的索引中的最大值
举例	strm_num = op_strm_max_index_in (); //获取输入流索号的最大值

op_strm_max_index_out():

函数	int op_strm_max_index_out()
功能	获得任何与周围进程或队列相连接的输出流的索引中的最大值

举例	strm_num = op_strm_max_index_out (); //获取输出流索号的最大值
----	---

op_strm_connected():

函数	Boolean op_strm_connected(int strm_type, int strm_index)
功能	判定周围是否存在处理器以指定的输入或输出流与本模块相关联, strm_type 取值为 OPC_STRM_IN 或 OPC_STRM_OUT
举例	if (op_strm_connected (OPC_STRM_IN, i)) //如果本模块输入 i 与处理器向关联。

op_strm_empty():

函数	Boolean op_strm_empty(int instrm_index)
功能	判断当前指定的输入流是否有包
举例	if (op_strm_empty (0)) //如果索引号为 0 的包流中不存在数据包。

op_strm_access():

函数	Comcode op_strm_access(int instrm_index)
功能	向与指定的输入流相关的模块发送一个访问中断
举例	op_strm_access (i); //向以索引号为 i 的输入流相关的模块发送访问中断

op_strm_pksize():

函数	int op_strm_pksize(int instrm_index)
功能	指定输入流中的数据包的个数
举例	pack_num = op_strm_pksize (i); //获取索引号为 i 的输入流中数据包的个数

2.3.3、收发信机

收发信机是连接节点内部模块与物理层链路的中间模块，在需要定向天线的模型时，又需要与定向天线连接，同时又常常需要添加统计量中断作为反馈信息，以便处理器模块控制数据包的接收与发送。收发信机位于节点模型内，包含点对点接收机、点对点发送机、总线发送机、总线接收机、无线发送机、无线接收机。

收发信机的操作很多，其中收发信机的统计量中断请参照 xx 节统计量中断，无线收发信机与定向天线的连接请参照 xx 节定向天线，收发信机中的管道模型请参照 xx 节管道模型，收发信机中的数据包的发送接收参照 xx 节数据包。

2.3.3.1、无线发信机

右键单击无线发信机编辑发信机属性设置。

Attribute	Value
name	send
channel	(...)
Number of Rows	1
Row 0	
data rate (bps)	1,024
packet formats	all formatted, unformatted
bandwidth (kHz)	10
min frequency (MHz)	30
spreading code	disabled
power (W)	100
bit capacity (bits)	infinity
pk capacity (pkts)	1,000
modulation	bpsk
rxgroup model	dra_rxgroup
txdel model	dra_txdel
closure model	dra_closure
chanmatch model	dra_chanmatch
tagain model	dra_tagain
propdel model	dra_propdel
icon name	ra_tx

图2-26. OPNET无线发信机属性设置

在无线发信机的属性中包含信道参数设置、调制方式、物理层管道参数设置等。其中信道参数主要包括比特率、数据包格式、带宽、频率下边界、扩频码、发送功率、最大比特缓存、最大包缓存。

(1) 信道参数

(1.1) 比特率(data rate):

比特率参数按仿真需求来设置，数值类型为 double。比特率决定了传输时长，即数据包第一个比特离开发信机到最后一个比特离开发信机之间的时长，这也是影响网络协议的主要因素。在传输速率较低时，数据包发送的时间久更长，冲突更大，吞吐量更低。

(1.2) 数据包格式(packet formats):

点击可选择软件识别到的数据包，也可以直接选择 support all packet formats、或者 support unformatted packets，数值类型为 string。数据包格式一般选择全部支持，新定义的数据包格式要重启 OPNET 才能被识别到。

(1.3) 带宽(bandwidth):

带宽参数按仿真需求来设置，数值类型为 double。带宽大小必须要大于 2 倍比特率。带宽大小决定了传输速率的大小，影响信道利用率和节点间干扰程度。

(1.4) 频率下边界(min frequency):

频率下边界参数按仿真需求来设置，数值类型为 double。根据频率下边界和带宽可以计算节点占用的频率范围，在默认的干扰模型中，当两个节点有重叠频率范围就会存在干扰，不管其他参数设置是否相同。同一节点的收发信机存在重叠频带时也会相互干扰。如接收机 A 正在正常接收数据包 b 中途，受到一个参数不匹配但是有重叠频带的发信机发送的数据包的干扰，则 A 不能正常接收数据包 b。

(1.5) 扩频码(spreading code):

扩频码的可选参数中，disabled 代表不支持扩频，或选择输入扩频码，数值类型为 toggle double。

(1.6) 发送功率(power):

发送功率的参数设置按仿真需求来设置，数值类型为 double。发送功率、灵敏度、链路衰减模型、定向天线、收发信机增益共同决定是否能成功接收。功率大小决定了可接收范围，这是因为功率不足会产生较大的误码率，所以会存在产生误码率大不能正确接收数据包的情况。自由空间的功率损耗模型为，接收功率=带内发射功率*发射天线增益*路径损耗*接收天

线增益。如下图所示，在距离发送节点较近的地方，接收功率大，没有误码率，能完全接收数据包。如图中 A 手机所在区域能完全接收数据包。在逐渐远离完全接收区域时，接收节点的接收功率逐渐下降，开始出现误码率，而这种误码率是随机的。如图中 B 手机所在的区域为不稳定接收范围，手机 B 既有可能接收到数据包也有可能接收不到数据包，同一个用户在同一个地点，不同时刻，有可能接收到也有可能接收不到数据包。当接收节点继续远离发送节点时，接收功率低于灵敏度，数据包完全无法接收。如图中 C 手机所在区域为无法接收范围。

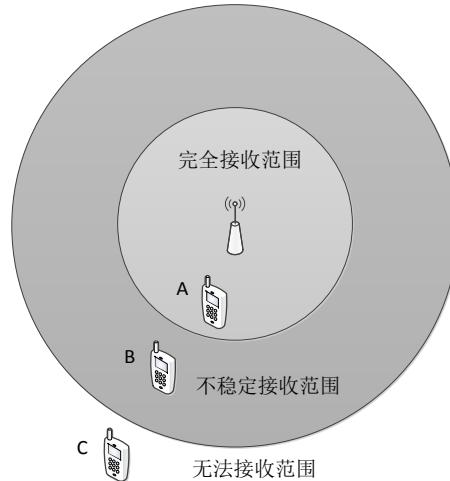


图2-27. 无线通信不稳定通信范围

(1.7) 比特最大缓存容量(bit capacity):

一般设置为 infinity 无限制，否则设置具体的值，数值类型为 double。当数据包到达发信机后，发信机会根据缓存容量设置计算当前缓存的比特数与缓存容量对比，当缓存比特数大于缓存容量，则会直接删除最后一个到达的数据包。

(1.8) 包最大缓存容量(pk capacity):

一般设置为 infinity 无限制，否则设置具体的值，数值类型为 double。同样，数据包到来时，发信机模块判断，如果缓存数据包数目与缓存容量大小，当缓存包数目超过包容量时（两者相等时不溢出），会直接删除最后到来的数据包。

(2) 调制方式

调制方式的选择按照实际需求设定，调制方式的选择与 MAC 层及以上层关系不大，只要收发信机调制方式相同即可。当需要对物理层管道模型进行建模或与 MATLAB 进行联合仿真时要按照真实场景设置。

(3) 管道模型

无线收发信机由于没有链路模型，所以通过在无线收发信机内设置管道模型，实现物理层的调控。无线发信机管道模型包含的函数阶段包括表 2-5 所示的内容，具体管道模型参考 2.6.2 章节。

表2-5. 无线发信机管道阶段函数

管道阶段名称	对应属性名	参考设置值
收信机分组	rxgroup model	dra_rxgroup
传输时延	txdel model	dra_txdel

链路闭合	closure model	dra_closure
信道匹配	chanmatch model	dra_chanmatch
发信机天线增益	tagain model	dra_tagain
传播时延	propdel model	dra_propdel

2.3.3.2、无线接收机

右键单击无线接收机编辑接收机属性设置。

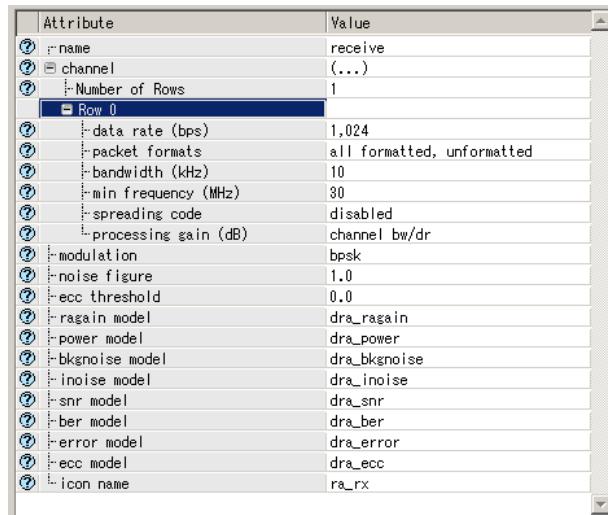


图2-28. OPNET无线接收机属性设置

在无线接收机的属性中包含信道参数设置、调制方式、物理层管道参数设置等。

(1) 信道参数

信道参数与无线发送机中参数设置相同，无线发行及信道匹配管道阶段函数对收发信机参数进行识别，将数据包划分为有效包、干扰包、忽略包。默认情况下，属性的值需要设置相同才能实现与无线发信机之间的数据包交换。其中处理增益的作用也是为了提高接收机的接收功率。

(2) 调制方式

调制方式需要与无线发信机相同才能匹配成功。可以通过调试方式的不同实现收发信机分组，但是要注意即使不能调试方式不同，使得收发信机间不能通行，如果占用频宽存在重叠，就一定会存在干扰。

(3) 管道模型

无线收发信机由于没有链路模型，所以通过在无线收发信机内设置管道模型，实现物理层的调控。无线接收机管道阶段包含表 2-6 所示的内容，具体参考 2.6 章节。

表2-6. 无线接收机管道阶段函数

管道阶段名称	对应属性名	参考设置值
收信机天线增益	ragain model	dra_ragain

接收信号功率	power model	dra_power
干扰噪声	inoise model	dra_inoise
背景噪声	bkgnoise model	dra_bkgnoise
信噪比	snr model	dra_snr
误比特率	ber model	dra_ber
误码分配	error model	dra_error
纠错	ecc model	dra_ecc

2.3.3.3、点对点发信机

由于有线收发信机必须连接有线链路使用，而只有固定节点可以连接有线链路，所以必须要使用固定节点才能正确应用具有线收发信机的节点模型。有线收发信机由于具有链路模型，所以属性设置比无线收发信机简单很多。仅包含 data rate 比特率、packet formats 包格式、bit capacity 比特容量、pk capacity 包容量。参数设置同无线收发信机。

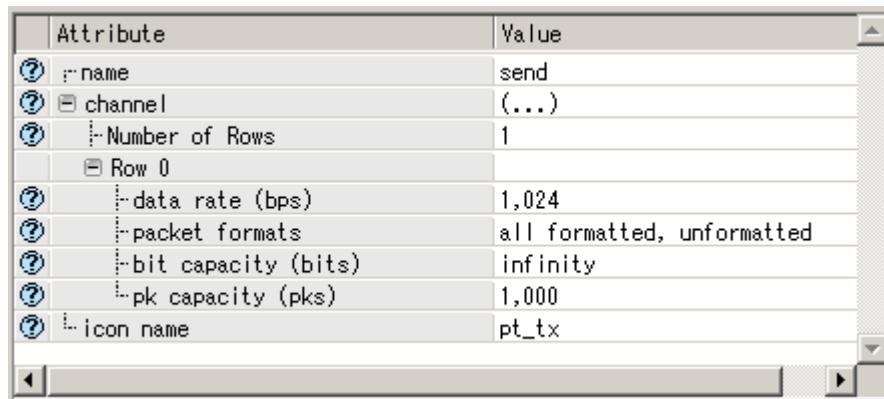


图2-29. OPNET点对点发信机属性设置

2.3.3.4、点对点收信机

点对点收信机的属性包含信道参数与误码率门限。信道参数设置中的比特率 data rate 和包格式与点对点发信机中的设置相同。

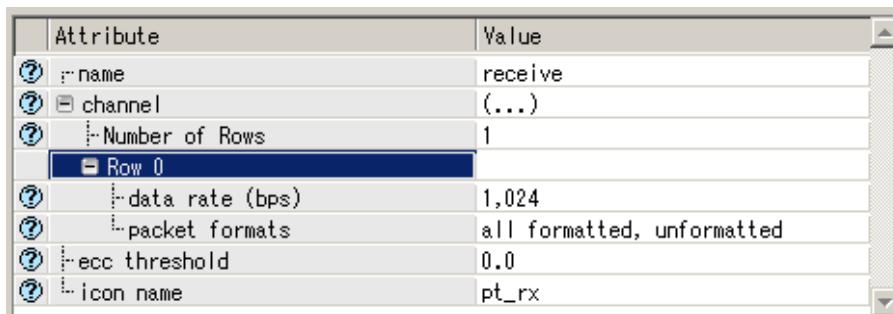


图2-30. OPNET点对点接收机属性设置

(1) 误码率门限(ecc threshold)

单位为 ecc/bit。默认设置为 0.0。代表最大容许的数据包中的比特错误比例。取值为 0.0~1.0。0.0 代表不能出现错误比特，不然无法接收，1.0 代表全部错误仍可以接收。

2.3.4、定向天线

(1) 定向天线在节点模型中的位置

定向天线只能和无线收发信机联合使用。收发信机可以连接同一个定向天线。多个收发信机可以连接同一个定向天线。连接图如图 2-31 所示。

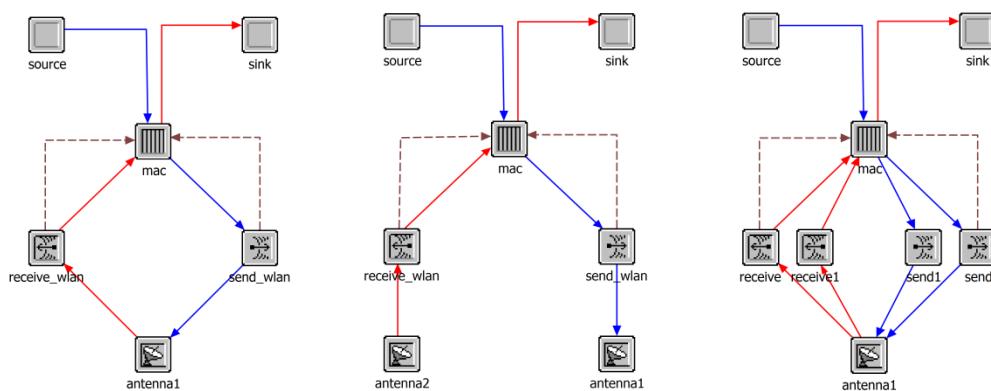


图2-31. OPNET定向天线连接示意图

(2) 定向的参数设置

OPNET 中定向天线模块的使用通过三组参数进行设置：天线方向图、天线指向、主瓣方向。

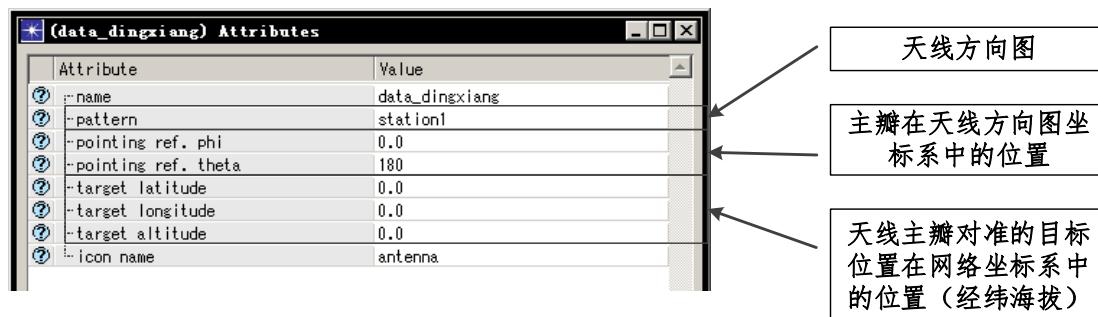


图2-32. OPNET定向天线属性设置

天线方向图坐标系使用三维坐标系，其中相对于坐标原点的方向由 phi 和 theta 两个参数确定，相对于坐标原点的增益由距离决定。

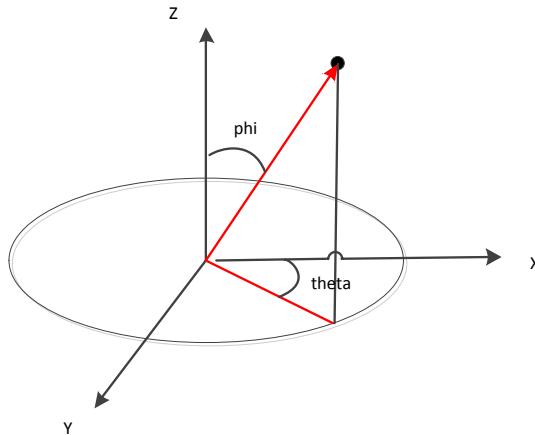


图2-33. OPNET定向天线天线方向图坐标

(3) 主瓣方向设置

主瓣方向的参数并不会根据使用者绘制的方向图自动选择最大增益方向，使用者需要手动设定主瓣方向。所以我们习惯先设置主瓣方向参数再根据主瓣方向参数绘制天线方向图。主瓣方向参数，即在天线方向图坐标系中的位置一般采用默认设置 pointing ref. phi = 0.0, pointing ref. theta = 180。即天线方向图的 z 轴方向为主瓣方向，我们在设计天线方向图时，将按照主瓣在 z 轴的角度去设置各方向增益。

(4) 天线指向

在仿真应用中，我们常常需要变更天线方向的指向，即天线主瓣指向的物体在网络中的经纬度海拔坐标。天线指向设置中需要经纬度海拔参数，因此需要使用 op_ima_node_pos_get 函数获取自己或目标的经纬度海拔信息。（注意在代码中变量不能使用中文）

```
op_ima_node_pos_get(目标节点 ID, &目标纬度, &目标经度, &目标海拔, &目标坐标 x, &目标坐标 y, &目标坐标 z);
int 定向天线模块 ID = op_id_from_name(本节点 ID, OPC_OBJTYPE_ANT, "定向天线名称");
op_ima_obj_attr_set(定向天线模块 ID, "target latitude", 目标纬度);
op_ima_obj_attr_set(定向天线模块 ID, "target longitude", 目标经度);
op_ima_obj_attr_set(定向天线模块 ID, "target altitude", 目标海拔);
```

定向天线常用指向操作如表 2-7 所示。

表2-7. 定向天线常用操作

固定节点指向固定节点	初始化代码中获取目标节点的经纬度海拔作为天线指向
固定节点指向固定方向	初始化代码中获取自己的经纬度海拔，并在自己的经纬度海拔值上加减作为目标经纬度海拔
固定节点指向非固定节点	在周期性执行的状态中获取目标节点的经纬度海拔作为天线指向
固定节点指向非固定方向	在周期性执行的状态中获取自己的经纬度海拔，并设置好每次对数值的操作，作为目标经纬度海拔

移动节点指向固定节点	初始化代码中获取目标节点的经纬度海拔作为天线指向
移动节点指向固定方向	在周期执行的状态中获取自己的经纬度海拔，并在此数据基础上加减作为目标经纬度海拔。从当前到下一次更新天线指向期间，天线始终指向该目标位置。所以目标的设置要尽量远离节点本身，使得在移动过程中有更好的同一指向效果。
移动节点指向移动节点	在周期执行的状态中获取目标节点的经纬度海拔作为天线指向
移动节点指向移动方向	在周期执行的状态中获取自己的经纬度海拔，并在此数据上修改作为目标经纬度，目标经纬度尽量远离当前节点的经纬度位置。

例如：卫星在移动过程中希望天线始终垂直指向地面。

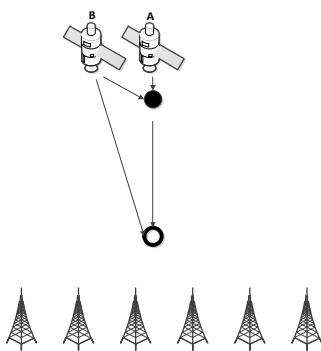


图2-34. 卫星高速移动中的定向天线设置

在卫星处于 A 的位置时更新天线指向，若天线指向设置中的目标经纬度处于实心球的位置，则在卫星进行下一次更新前，虽已经处于 B 的位置，但是仍然指向实心球的位置，已经严重偏离垂直向下的指向。若卫星在 A 处更新天线指向时，将天线指向空心球的位置，则在到达 B 时，虽然仍指向空心球，有些偏离，但是偏离程度不大，所以在指向方向时，设置的目标经纬度要尽可能远离源模块，或更新频率尽可能高，但是要注意更新频率提高引起的程序运行缓慢问题。

(5) 天线方向图

通过 File—New—antenna Pattern—OK，打开天线方向图设置窗口。

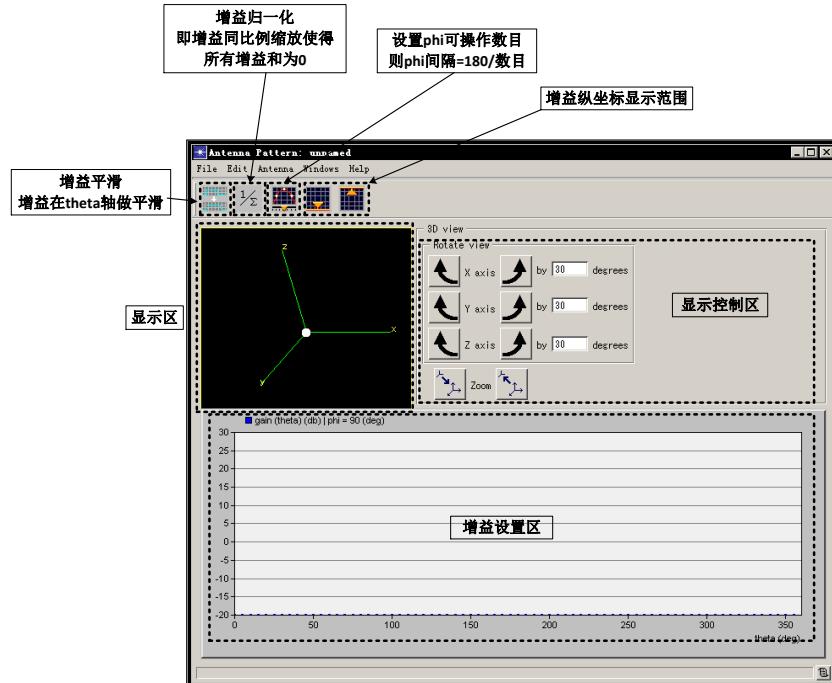


图2-35. 天线方向图设置窗口

天线方向图是三维图像。OPNET 采用多层二维增益设置的方式构建三维方向图。Theta 的取值为 0° ~ 359° 每个整数。phi 的取值范围为 0° ~ 180° ，但并不是 phi 每个值都会获取一遍，具体取值根据设置的 phi 值的数目不同而不同。Phi 值从 0° 开始，间隔为 180° 除以 phi 取值数量后取整，例如默认 phi 值的数目为 36 个，即 Phi 值的分布默认间隔 5° ，即 phi 的取值分别为 0.0° , 5.0° , 10° ..., 175° 。不同的 phi 取值，都对应一个增益设置区。比如上图为 phi= 90° 时，theta 在 0° ~ 359° 上的增益分布。需要全部设置完每一个 phi 值对应的增益分布，才算完成天线方向图。每两个 phi 取值之间的范围的增益分布与小于自己的 phi 取值的增益分布相同。例如在 phi= 0.0° ~ 5.0° 之间的范围的增益分布与 phi= 0.0° 时的增益分布相同。在 phi= 5.0° ~ 10.0° 之间的范围的增益分布与 phi= 5.0° 时的增益分布相同。

(6) 单一 phi 值时的增益分布

Phi 值固定时，增益的设置为纵坐标增益在横坐标 theta 上的分布。首先选择一个 phi 的值，然后就可以设置此 phi 值时的增益随 theta 的变化了。一般关于主瓣方向（这里使用 z 轴）对称的天线，增益在横坐标 theta 为恒定值。打开窗口时 phi= 90° 。首先确定增益的分布范围，如增益为 -100db。为了更加精确的绘制增益分布，我们设置纵坐标的显示下边界为 -101db，上边界为 -99db。



图2-36. 天线增益上下限设置

然后我们看到增益设置区域的纵轴范围变成 -101 到 -99。在 theta=0，增益 gain=-100 的地方单击一下，在 theta=359，增益 gain=-100 的地方单击一下，其他 theta 地方的增益会自动按直线填充，如下图所示。单一 phi 值的增益分布就设置好了。

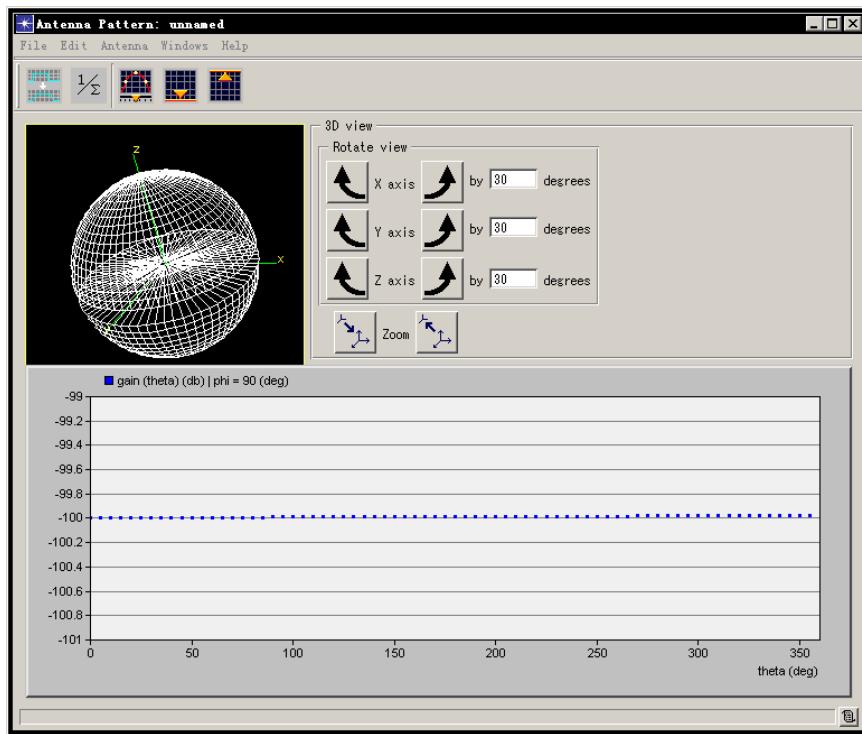


图2-37. 天线固定phi值时天线增益设置

(7) Phi 值的变更

当设置完一个 phi 值的增益分布时，需要切换 phi 值设置下一个 phi 值时的增益分布。在增益设置区域单击右键，弹出 phi 值变更快捷菜单如图 2-38 中左图所示。Increase Phi Plane 为增大一个 phi 间隔的 phi 值，Decrease Phi Plane 为减小一个 phi 间隔的 phi 值，Set Phi Plane 直接进入 phi 值的选择界面如图 2-38 中右图所示。在 phi 值的选择界面可以直接选中进入对应此 phi 值的增益分布设置区。



图2-38. 天线方向图phi值变更

由于要全部的 phi 值都要设置，这就会存在 phi 值分布过密造成操作量大，或者 phi 值分布过于稀疏造成天线图不精确的问题。这种问题可通过设置 phi 值数量解决，点击菜单栏 Antenna，选择 Set Phi Plane Count 进入 phi 值数量设定。



图2-39. 天线方向图phi值数量设置

phi 值的间隔为 180 度除以 phi 值的数量。当我们觉得 phi 值分布过密时，我们减小 phi 值的数量，增大 phi 值的间隔。当我们觉得 phi 值分布过于稀疏时，我们增大 phi 值的数量，减小 phi 值的间隔。当主瓣或旁瓣分布角度大，但是主瓣与旁瓣的边界又需要精细处理的话，我们可以先减少 phi 值的数量，使得 phi 值的间隔大，设置好大范围的 phi 值后再增大 phi 值的数量做精细处理。定向天线制作完成后，天线增益的使用在物理层功率中的效果参考 2.6.2.5 无线链路发送天线增益管道阶段函数的使用。

示例：参考实例 5

2.4、进程模型

2.4.1、进程概念

进程域是最底层的域也是最难上手的域，如果全部使用 OPNET 自带的进程模块，那么我们永远只能是个 OPNET 使用者，但如果你想要搭建系统实现自己的一些想法，那么我们必须得学会进程的编写，开发自己的进程模块，上升到软件开发的层次。

一个进程可以被认为近似于一个执行程序。在 Windows 系统中一个进程都有一串数字作为一个进程的唯一标识，我们成为进程句柄。在 OPNET 中使用 Prohandle 类型表示。OPNET 中的进程是基于进程模型，他们是在进程编辑器中定义的。一个进程模型中的元素包含初始状态、强制态、非强制态、状态转移线、状态转移条件、执行函数、状态变量、临时变量、全局变量、自定义函数等。状态变量、临时变量、全局变量的学习请参考 3.1 变量章节。

进程组由许多进程组成，这些进程都是在同一个处理器模块或者队列模块中执行。当仿真开始的时候，每个模块只能有一个进程，称之为根进程。这个根进程之后能够创建新的子进程，他们之后也能再创建其他的子进程。在仿真中被创建的子进程称之为动态进程。

一个进程只能按顺序执行，同一时刻只能处于一个状态中。当一个进程开始执行后，我们说这个进程被调用了。当一个进程调用另一个进程时，源调用进程被暂时挂起直到被调用进程被阻止。一个进程如果完成了它当前调用的处理就将被阻止。当被调用进程被阻止时，源调用进程就将从它挂起的地方继续执行。进程 A 如图 2-40 所示，在绿色态中使用代码调用激活了某子进程 B，进程 A 会立刻暂停。进程 B 会先执行直到进程 B 遇到它自己的红色阻塞态。之后进程 A 会从调用进程 B 的地方继续执行到红色态。

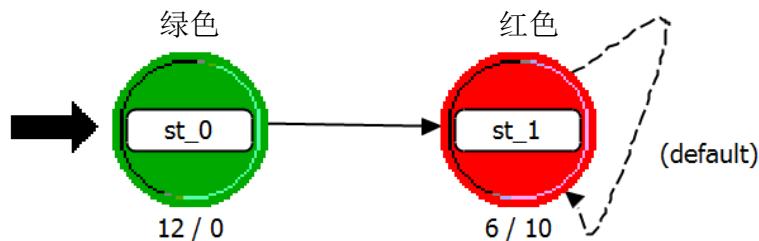


图2-40. 多进程示例A进程模型

OPNET 中的进程能够对中断或者调用产生响应。中断可能产生自进程组之外的源，或者进程组中的其他成员，或者这个进程自己。中断一般对应于一些事件，包括消息到达、计时器到时、资源释放或者其他模块的状态改变等等。

OPNET 的进程编辑器使用 Proto-C 的语言来描述进程模型。Proto-C 基于状态转移图 (State Transition Diagrams STD)、一个高级指令库 (kernel Procedures) 和 C 或 C++ 的通用部件。STD 定义了模型的各个进程所处的状态，以及使进程在状态之间转移的条件。这种条件称之为 transition。高级指令库主要一系列的函数包组成，这些包详见参考文档。

a) 状态变量 (State Variables)。进程可以拥有一些私有状态变量，这些状态变量可以是任意的数据类型，包括 OPNET 专有的、通用的 C/C++、用户定义类型等等。这种能力使得进程能够灵活的控制 计数器、路由表、与性能相关的统计量、需要转发的数据包。任意组合的状态变量可以在一个进程所有的动作和决定中使用。

b) 状态执行 (State Executives)。通过 C/C++语言描述了进程进入和离开状态时的操作，典型的操作包括：修改状态信息，创建或接收消息，更新发送消息的内容，更新统计数据，设置计时器以及对计时器作出响应。

c) 转移条件 (Transition Conditions)。通过 C/C++语言描述布尔变量，要涉及中断的属性以及状态变量的组合。

d) 转移执行 (Transition Executives)。转移时可能会定义一些通用的操作。

进程中常用的函数包括如表 2-8 所示。

表2-8. 进程操作常用函数

Prohandle op_pro_self(); 获取当前进程的进程句柄
int op_pro_id (Prohandle pro_handle); 根据进程句柄获取进程 ID。
Prohandle op_pro_root (Objid mod_objid); 根据模块 ID 获取模块根进程句柄。
Boolean op_pro_equal (Prohandle h1, Prohandle h2); 判断两个进程是否相等。
Objid op_pro_mod_objid (Prohandle pro_handle); 获取当前运行此进程的模块 ID
Boolean op_pro_valid (Prohandle pro_handle) 判断一个进程句柄指向的进程是否存在
void * op_pro_svar_get (Prohandle prohandle, const char * svar_name); 获取一个进程中的指定状态变量
int* child_int_svar_ptr = op_pro_svar_get (child_pro_handle, "int_var");

```
Compcode op_pro_tag_set (Prohandle pro_handle, const char* tag_string);
设置一个进程的描述语。
```

2.4.2、多进程

在仿真开始时，节点对象中的处理器模块、队列模块或者外部系统模块中都只能包含一个进程。该进程被称为模块的“根进程”，是仿真核心自动创建的。该进程是模块的“进程模型”属性所指定的模型的实例。“根进程”模型可以用来管理所有的任务，并提供所期望的功能。但是在很多情况下，单独用一个进程模型来实现所有的功能很复杂。而将其分解成由多个相互协调的进程促成的进程集，则可以提高模块化程度，并降低设计的复杂度。

进程集中除了根进程外，其他进程都需要调用核心函数来创建。这时，根进程所在模块就成为容纳进程集的容器，进程集中的各进程都必须经由它所在的模块建立和仿真核心之间的接口（事件、数据流等）。

使用多进程有两条规则：

1 至少模块的部分功能确实需要管理动态的任务，这种任务的数量和出现时机是预先不确知的。在这种情况下，每一个出现的任务都对应一个新进程。任务完成后进程可以被销毁。由于模块可以包含的动态进程是没有限制的，因此并发任务的数量也是没有限制的。

2 模块需要管理若干并行、并发、相互独立的任务，这时模块功能用多个异步运行的进程表达比较方便。所需的进程可以在仿真开始时就创建好，并可以在其后的整个仿真过程中保持不变，各进程相互协作以支持整体功能的实现。

需要注意的是，虽然进程可以创建任何类型的进程，包括它自己的类型，但是这些类型必须在仿真运行前预先声明为自“子进程”，否则创建未声明类型的进程可能会导致运行错误。“子进程”的属性会被逐级提升，直至根进程所在的模块，成为模块属性列表的一部分。由于循环声明，来自相同进程模型的属性只会显示一次，除了来自根进程的属性，其他属性会在名称前添加所属进程模型名，这样能防止可能出现的属性同名。

(1) 进程树

上述进程集中各进程根据创建关系可以构成一个进程树。树根是根进程，树枝是创建的关系，树上的其他节点为动态进程。

进程树中的根进程是起点，并且是唯一不变的，树可以在仿真过程中生长变化，但根进程保持不变。不过需要注意的是，树在生长过程中是一个联通的整体，然而由于动态进程可以随机销毁，并且它创建的进程可以在销毁后还继续存在，所以进程树可能由于某动态进程的销毁而成为森林。在图 2-41 中，虽然根进程与 2 级子进程间不能联通，但实际上是可以管理的。根进程可以通过共享内存等方式获得其后代进程节点的句柄信息。

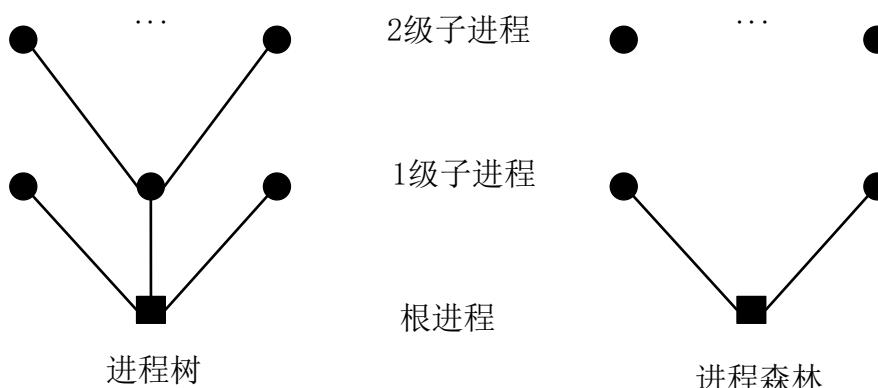


图2-41. 进程模型进程树与进程森林

OPNET 中针对进程树常用的函数如表 2-9 所示。

表2-9. OPNET 常用进程函数

创建进程	创建进程前，首先保证已经设计好了需要创建的进程模型。
函数	Prohandle op_pro_create (const char* model_name, void* ptc_mem_ptr);
介绍	创建一个新进程，并设置父子进程之间共享内存块。当不需要共享内存时，内存指针使用 OPC_NIL。
例	Prohandle mypro_handel = op_pro_create ("model_name", OPC_NIL);

调用进程	调用进程前，首先在进程模型界面点击 File-Declare Packet Process Models 声明需要运用的子进程。
函数	Compcode op_pro_invoke (Prohandle pro_handle, void* argmem_ptr);
介绍	调用进程，在使用此函数后，系统将自动进入子进程，并等待到子进程返挂起后才返回原进程继续执行。内存地址不需要时使用 OPC_NIL。
例	op_pro_invoke (mypro_handel, OPC_NIL);

销毁进程	在进程不在需要时，可以销毁进程。
函数	Compcode op_pro_destroy (Prohandle pro_handle);
介绍	通过进程句柄，销毁内存。
例	op_pro_destroy (pro_handle);

销毁进程	销毁进程时可能也希望同时取消此进程中被调度的事件。
函数	Compcode op_pro_destroy_options (Prohandle pro_handle, int options);
介绍	当 Options 是 OPC_PRO_DESTROY_OPT_KEEP_EVENTS 时，系统不会取消被销毁进程中的所有事件，此时等价于 op_pro_destroy 当 Options 是 OPC_PRO_DESTROY_OPT_NONE 时，系统会取消被销毁进程中的所有事件
例	op_pro_destroy_options (op_pro_self(), OPC_PRO_DESTROY_OPT_NONE);

父进程	通过子进程获取父进程
函数	Prohandle op_pro_invoker (Prohandle pro_handle, int invnode_ptr);
介绍	获取进程是被系统激活还是被其他进程激活，返回父进程的句柄，并通过指针返回间接模式。
例	invoker_ph = op_pro_invoker (op_pro_self (), &inv_mode);

父进程	通过子进程获取父进程
函数	Prohandle op_pro_parent (Prohandle pro_handle);

介绍	通过子进程句柄获取父进程句柄。
例	parent_ph = op_pro_parent (op_pro_self());

(2) 多进程间数据共享机制

有三种机制用来作为多进程中各个进程之间的通信方式：

a) 模块共享内存：

通过函数 `op_pro_modmem_install()` 和 `op_pro_modmem_access()` 访问。为了保证进程间通信机制，各个进程应当遵循共享内存的数据类型，这就要求各进程都要知道，因而共享内存的数据结构定义应当放在外部定义".h"文件中，并包含在每个进程的头模块中。共享内存一开始是没有的，是由各进程来决定什么时候分配以及分配多大，这些通过 `op_pro_modmem_access()` 来完成。内存的分配一般是通过 `op_prg_meme_alloc()` 来完成。

b) 父子共享内存：

只有以父子关系联系在一起的进程才能访问的私有共享内存。这种共享内存只能在子进程由 `op_pro_create()` 产生时由 `op_prg_mem_alloc()` 分配，且不能被替换。通过 `op_pro_parmem_access()` 访问。通过 `op_pro_invoke()` 通知对方对共享内存的内容进行的修改，以及对内容的检查。

c) 参数内存(argument memory)

将内存地址作为 `op_pro_invoke()` 的参数传给别的进程用以通信，通过 `op_pro_argmem_access()` 来完成访问。与前两个不同的是，这部分内存不是永恒的。

多进程内存控制中常用的函数如表 2-10 所示。

表2-10. 多进程内存控制常用函数

void op_pro_modmem_install (void* mem_ptr);	创建模块内存
void* op_pro_modmem_access ();	读取模块内存
void* op_pro_parmem_access ();	读取亲自内存
void* op_pro_argmem_access();	参数内存的接入

2.4.3、状态

一个进程对应一个进程模型，即一个状态转移图。一个状态转移图（进程）内的程序只能按顺序执行，不能同时多处被执行。同一个处理器模块内的多个进程同一时刻也只能有一个能处于执行状态，要想多个进程同时运行，需要创建多个处理器模块来实现。这与 C++语言编程的不同处是，C++可以开启线程，使一个进程内，多个线程同时运行。一个状态转移图包含初始状态，阻塞态（红色），非阻塞态（绿色），状态转移线。

(1) 初始状态(Initial States):

初始状态是进程被第一次调用时的起始位置。通过 `set initial state` 或 `make initial state` 来设置。`begin simulation interrupts` 是一个模块属性，开启后用来完成对初始状态的进入。通过模块的 `begsim intrpt` 属性来选择。当然也可以不选择使用开始编

译中断而使用普通的中断(不推荐)。

(2) 强制和非强制状态(Forced and Unforced States):

进程在任意时刻只能处在一个状态下。进程可以根据它收到的中断在状态之间转移。每个状态的执行过程分为两个部分。入指令(enter executives)和出指令(exit executives)，分别在进入和离开该状态的时候执行。

进程定义了两种状态，称之为强制状态(forced states)(非阻塞态)和非强制状态(unforced states)(阻塞态)，分别用绿色和红色表示。非强制状态允许进程在入指令和出指令之间暂停。一旦进程执行完非强制态下的入指令，就被挂起，并将控制权交还给调用它的其他进程。直到下一个新的调用产生使得它进入当前状态的出指令。强制状态是不允许进程等待的。所以一般它的出指令是空白的。这是它与强制状态的最大区别。

(1) 代码执行顺序 (状态内):

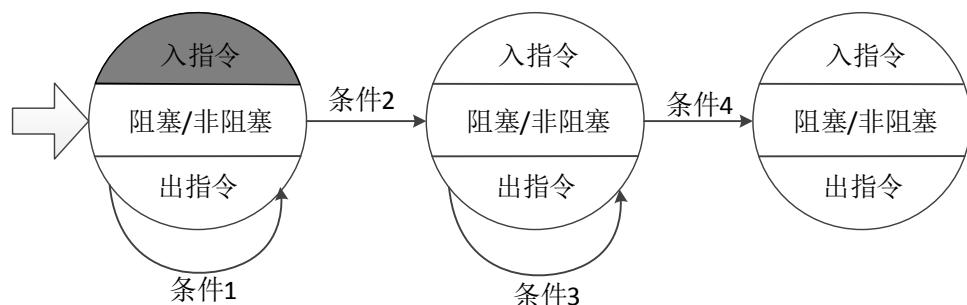


图2-42. 状态内代码执行图a

进程首先执行初始状态的“入指令”，“入指令”和“出指令”可以理解为一个自定义函数的函数体。当执行完“入指令”后，进程进入当前状态的“阻塞/非阻塞”位置。

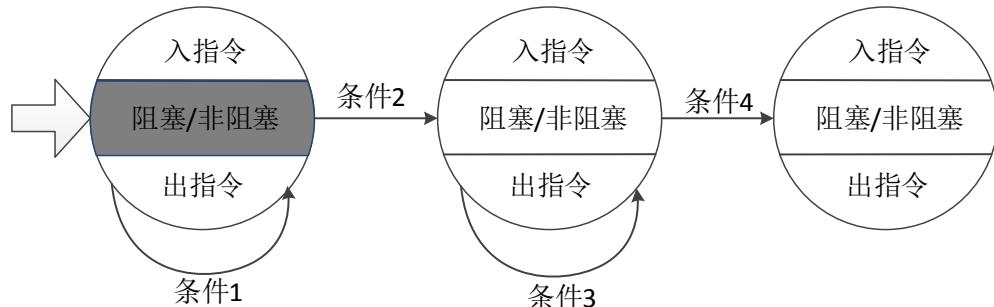


图2-43. 状态内代码执行图b

进程处于“阻塞/非阻塞”位置时，若当前状态为绿色“非阻塞”状态时，进程直接跳出“阻塞/非阻塞”位置，执行当前状态的“出指令”。若当前状态为红色“阻塞”状态时，进程挂起，不再向后执行，进程暂停，当进程接收到中断后，进程恢复，执行当前状态的“出指令”。因此中断可以被认为是使进程恢复继续运行的功能函数。

所以可以看出，进程经过绿色“非阻塞”状态时，并不会暂停，执行完入指令后，立刻执行出指令，而两个指令又同时独立函数体。将两个函数体都写在“入指令”中或者都写在“出指令”中，效果是一样的。因此我们在编程中常看到设计者仅在绿色“非阻塞”状态的“入指令”中写有代码。

同样绿色“非阻塞”状态，进程经过时，不暂停，状态仅相当于一个函数体，而这个函数体又可以直接写到红色“阻塞”态中。因此绿色“非阻塞”态，并非必须有的。绿色“非

“阻塞”态时为了更加直观的理解状态的转移流程，和分担红色“阻塞”态代码过多的压力。

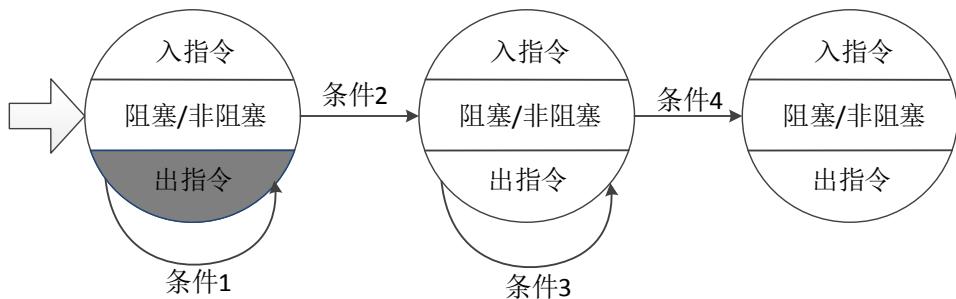


图2-44. 状态内代码执行图c

进程执行“出指令”，同代码执行“入指令”一样，运行一段函数体。当函数运行完“出指令”后，面临状态跳转选择。

(2) 代码执行顺序 (状态间):

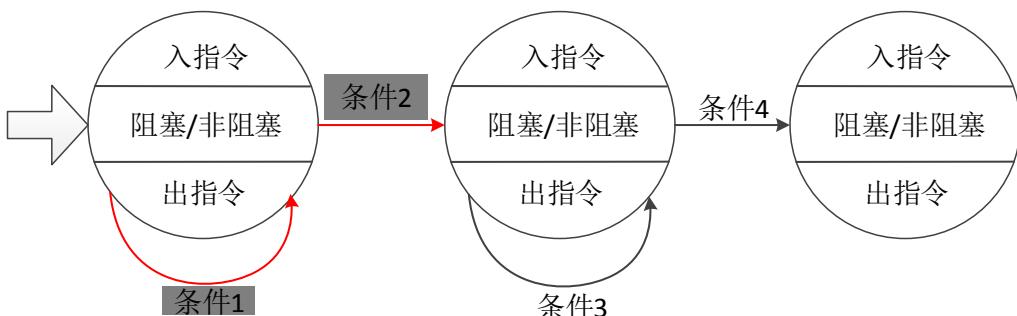


图2-45. 状态间代码执行图a

状态转移线上的条件可以通过右键单击转移线，在编辑属性窗口中设置。

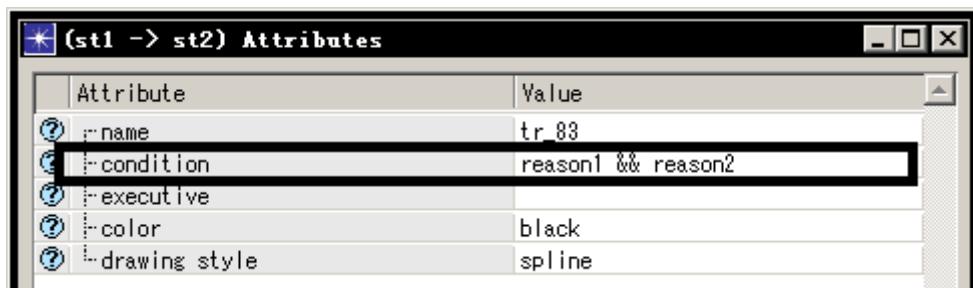


图2-46. 状态转移属性设置

转移条件的值为布尔型的变量或公式，可以理解为 if(条件 1) 或者 if(条件 2)，使用者尽量使用 OPNET 使用的布尔型 Boolean，不要使用 c 语言 bool 类型。

转移条件的值可以是头模块中的宏定义，也可以是状态变量或全局变量。我们在使用时，把状态转移条件的值，理解为 if 语句的判别语句来处理。

例：

在头模块中宏定义

```
#define reason1 (!tra_busy && fin_send)
#define reason2 (!rec_busy && fin_receive)
```

则状态转移条件 reason1 && reason2 等价于

```
(!tra_busy && fin_send) && (!rec_busy && fin_receive)
```

OPNET 要求状态的状态转移必须“有出路，不重复”。既满足条件 1||条件 2 的值为 TRUE 有满足条件 1&&条件 2 的值为 FALSE。OPNET 定义了状态转移值“default”，等级与 c 语言中的 else 功能。当不满足其他状态转移条件时，执行“default”指向的状态转移。

状态转移线，只是连接状态与状态之间，与状态内部的执行没有关系。如图 2-45 所示，当进程执行完第一个状态，判断当前条件 1 和条件 2 哪个为真，若条件 1 为真，则进程重新执行状态 1，按照状态内部执行顺序运行函数，若条件 2 为真，则进程执行状态 2，按照状态内部执行顺序运行函数。

(3) 初始状态的设定

初始状态是程序开始执行的地方，可以是阻塞态，也可以是非阻塞态。右键单击需要设置为初始态的状态，点击选择 Make Initial State。

或者左键单击选中想设置为初始态的状态，点击工具栏设置初始状态按钮 ，即可将一个状态设置为初始状态。

(4) 阻塞态、非阻塞态转换

右键单击需要改变状态类型的状态图标，当图标是绿色“非阻塞”（强制）状态时，点击选择 Make State Unforced。当图标是红色“阻塞”状态时，点击选择 Make State Forced。

最后使用一个进程模型的例子，解释状态转移的方法。

例：

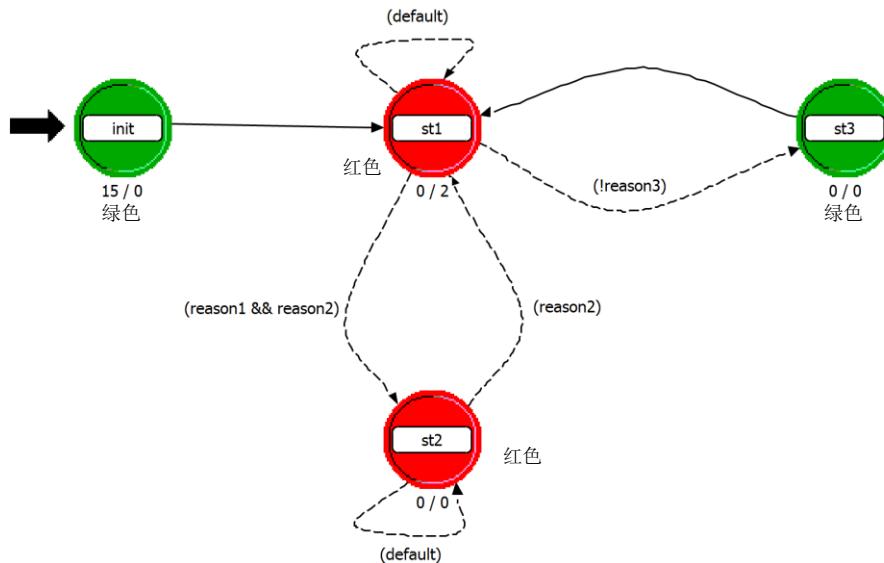


图2-47. 状态转移流程示例

进程执行流程：

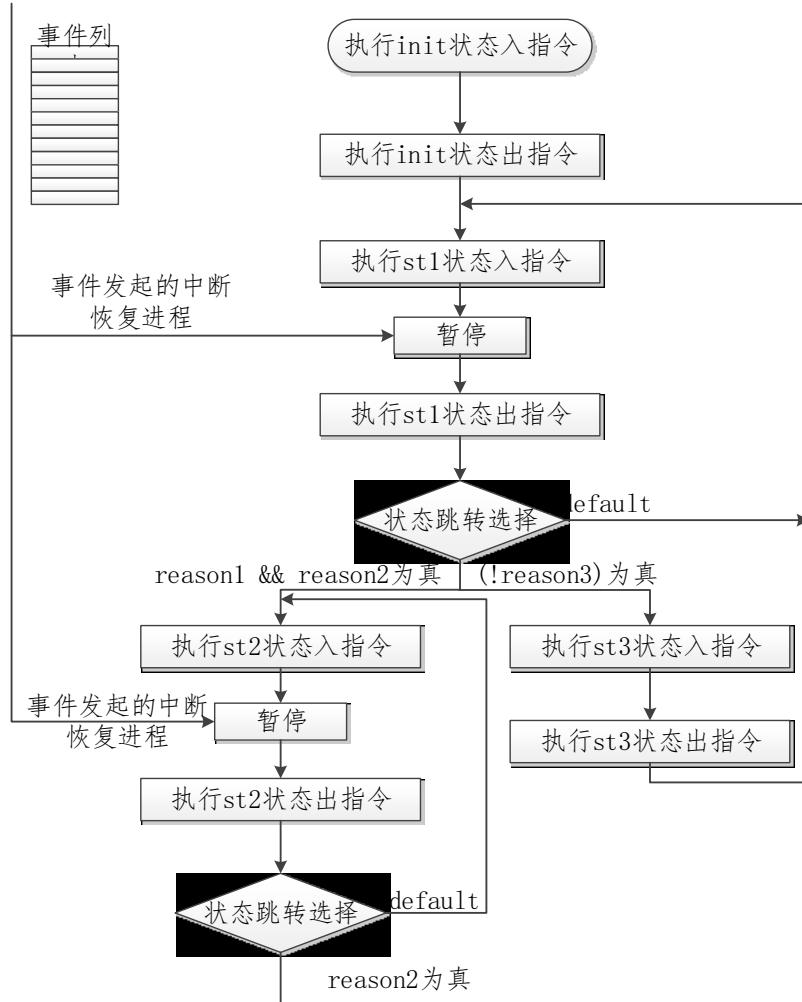


图2-48. 状态转移流程

2.5、轨迹轨道

卫星节点和移动节点具有可移动性，可以根据需求设置移动轨道。移动节点使用自定义绘制轨迹，卫星节点需要导入STK轨道文件。

2.5.1、轨迹的定义

在子网模型界面，点击菜单栏 topology 按钮，选择 Define Trajectory，进入轨迹定义窗口，如图 2-49。在轨迹定义界面，可以设置轨迹的名称 Trajectory name，轨迹类型 Trajectory type，初始海拔高度 Initial altitude，初始等待时间 Initial wait time，初始俯仰角 Initial pitch（移动节点沿前进方向的上下角度），初始航向角 Initial yaw（移动节点沿前进方向的左右角度），初始滚动角 Initial roll（移动节点沿前进方向的旋转角度）

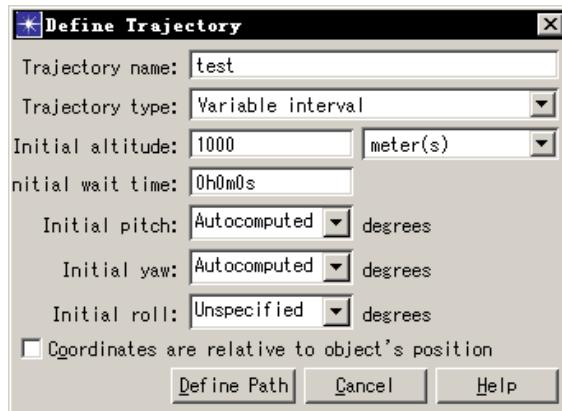


图2-49. OPNET轨迹定义窗口

设置完毕后，点击 Define Path，开始绘制轨迹，会出现轨迹状态窗口，同时鼠标变成一个连有一段线段和“start”字样的箭头。

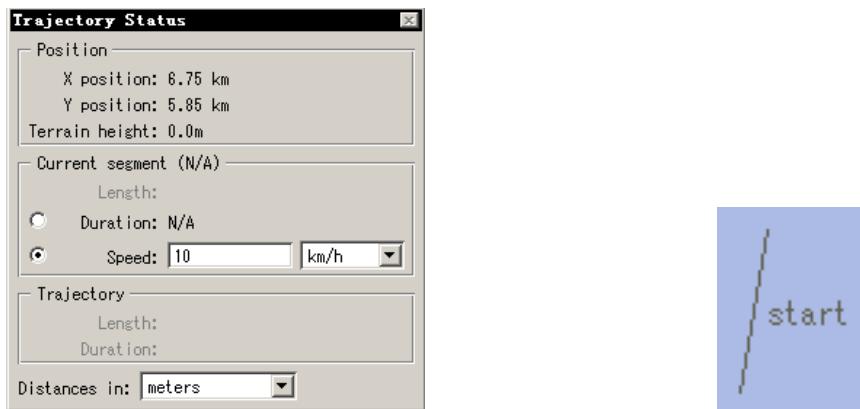


图2-50. OPNET轨迹状态窗口

我们在轨迹的起点点击后松开鼠标，鼠标会拉出一条线段，线段上标有 segment 1，为第一段轨迹，在第一段轨迹的结尾再点击一下鼠标左键，轨迹固定，并转变为红色，同时弹出此段轨迹的信息，包含速度，海拔高度，三维角度，开始时间起点。若轨迹绘制完毕，点击 Complete 完成绘制。若要继续绘制第二段轨迹，选择 Continue，鼠标在第一段轨迹的末尾拉出一条线段，线段上标有 segment 2，为第二段轨迹。依次继续绘制。

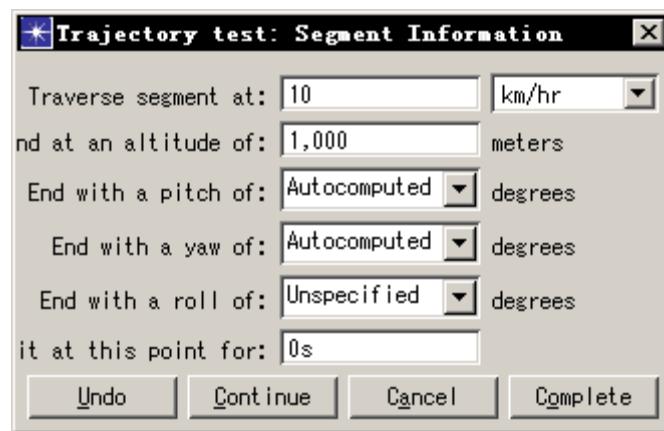


图2-51. OPNET轨迹段信息

轨迹在绘制以后会自动保存成.trj 文件。

2.5.2、轨迹的应用

在需要应用轨迹的移动节点的属性编辑界面点击属性 trajectory 值的地方，选择刚刚存储的轨迹模型。则绘制的轨迹就会出现在子网模型中，在仿真中，节点就会按照轨迹的位置移动。

在移动网络中，尤其是大范围移动通信网，如卫星网络时，往往需要获取节点位置，以实现定位功能，进而实现基于位置的网络协议。表 2-11 为 OPNET 中较多使用的对象位置信息函数。在 3.4.3 对象属性章节也有介绍。

表2-11. OPNET 对象位置信息常用函数

op_ima_obj_pos_get (Objid site_objid, double* lat_ptr, double* long_ptr, double* alt_ptr, double* x_ptr, double* y_ptr, double* z_ptr); 获取节点或子网的位置信息，其中获取的海拔高度单位为 m，在获取通过节点的 altitude 属性获取的节点海拔高度单位与子网尺度单位相同。 例 op_ima_obj_pos_get (node_id,& lat,& lon,& alt,& x,& y,& z);
op_ima_obj_pos_get_time (Objid site_objid, double time, double* lat_ptr, double* long_ptr, double* alt_ptr, double* x_ptr, double* y_ptr, double* z_ptr); 获取节点或子网模块指定仿真时刻的位置信息 例 op_ima_obj_pos_get_time (node_id, op_sim_time(),& lat,& lon,& alt,& x,& y,& z);
op_ima_obj_pos_set_geocentric (Objid objid, double x, double y, double z); 设置对象到地心坐标系下指定位置
op_ima_obj_pos_set_geodetic(Objid objid, double lat, double lon, double alt); 设置对象到大地坐标系下指定位置
op_ima_traj_info_get (Objid objid, double time, double * ground_speed_ptr, double * ascent_rate_ptr, double * end_time_ptr) 获取对象指定时间的水平速度和上升速度。时间必须为当前仿真时间及以后

但是需要注意的是，节点的移动与绘制的轨迹可能并不完全相同，这是由于轨迹的定义是通过轨迹点定义的原因。绘制的轨迹，由于仅能记录轨迹点，真实轨迹会在按照真实两轨迹点间直线运动。因此当轨迹线跨度范围大时，仿真时轨迹线会与绘制轨迹线有较大区别。如图 2-52 所示，我们希望移动节点沿纬度线移动，轨迹距离有 2000km，而实际中我们是希望移动节点沿纬度线曲线运动。OPNET 由于只能记录两个轨迹点，所以会自动在这两点间直线运动。以至于在仿真动画中我们看到的节点移动路线为途中虚线。

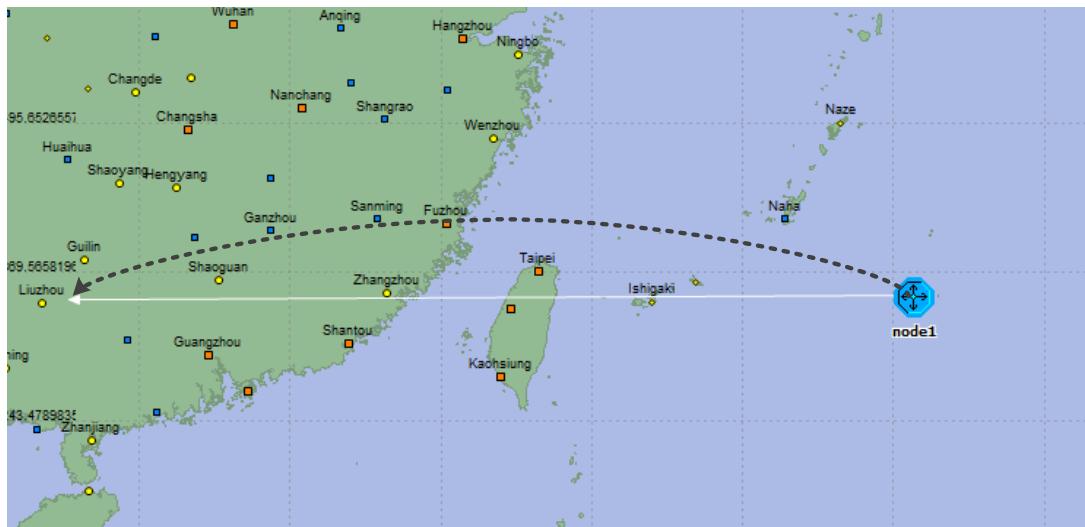


图2-52. 移动节点实际移动与绘制显示区别

由于在长距离移动时，使用轨迹点设置的移动路径不能完美的模拟真实路径。所以为了更加真实，我们往往需要添加多个轨迹点，使节点的移动更加符合我们的预想。也正是因此卫星节点的移动，只能使用 STK 导入轨道文件。如图 2-53 所示，真实运动轨迹为实线部分，子网模型中显示为虚线部分。

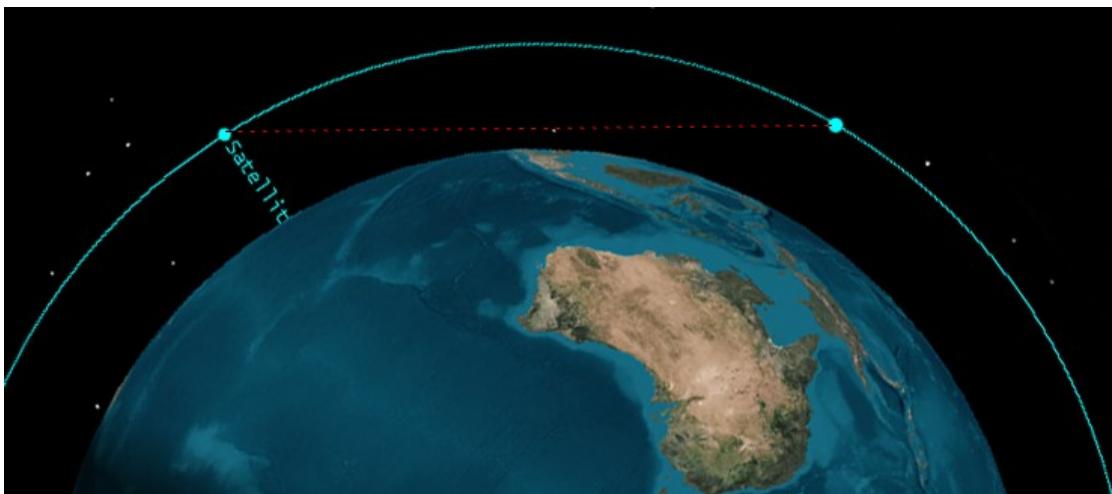


图2-53. STK卫星轨道

2.5.3、轨迹修改

右键单击轨迹线，弹出仅有 Edit Trajectory 选项的快捷选项，点击 Edit Trajectory 修改轨迹设置。在轨迹信息界面，会显示所有轨迹点。第一个轨迹点为轨迹的起点。各轨迹点信息包括轨迹 X, Y 坐标，与起点的距离，轨迹点海拔高度，移动所需时间，移动速度，等待时间，三维角度等信息。

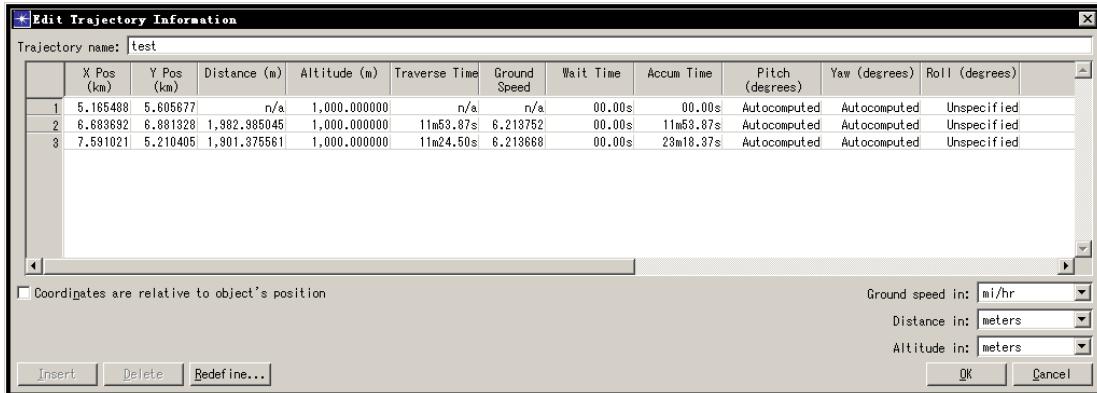


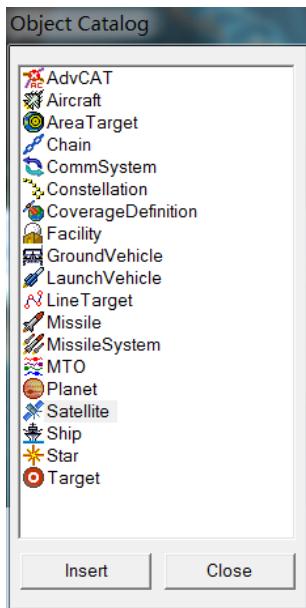
图2-54. OPNET轨迹编辑窗口

在轨迹信息界面可以直接修改轨迹点的相关参数，可以修改速度单位，距离单位，海拔高度单位，也可以直接点击 Redefine，重新定义轨迹。

2.5.5、卫星轨道的使用

对于卫星节点，节点的移动符合一定的物理定律。为了真实的模拟卫星轨迹，OPNET 需要导入 STK 制作的.sa 格式的轨道文件。由于 OPNET14.5 无法识别 STK10 的轨道文件，这里以 STK8 为例讲述轨道文件制作方法。

(1) STK 制作卫星轨道。



打开 STK 8 软件，点击 File-New，创建新场景，默认名称为 Scenario1。点击菜单栏 Insert-New，进入对象插入窗口，如图 xx 所示。

选择 satellite，点击 Insert 插入，自动弹出轨道设置对话框。直接取消，可以在轨道设计界面自行设计修改。

图2-55. STK卫星节点插入

右键单机新插入的 satellite1，选择 properties Browser 进入卫星属性设置界面。在 Basic-Orbit 中设置卫星轨道，在轨道参数中修改后点击下方的 Apply 可以在 2D 和 3D 视图中查看轨迹，其中 Apogee Altitude 为远地点海拔，Perigee Altitude 为近地点海拔，Step Size 决定采样精度，Orbit Epoch 和场景属性中的 Start 时间共同决定卫星开始运行时间，决定了轨迹的起点，Inclination 为运行轨迹所能达到的维度范围，90 度时能达到南北极，根据自己的需求设置这些参数后，选择 Apply。

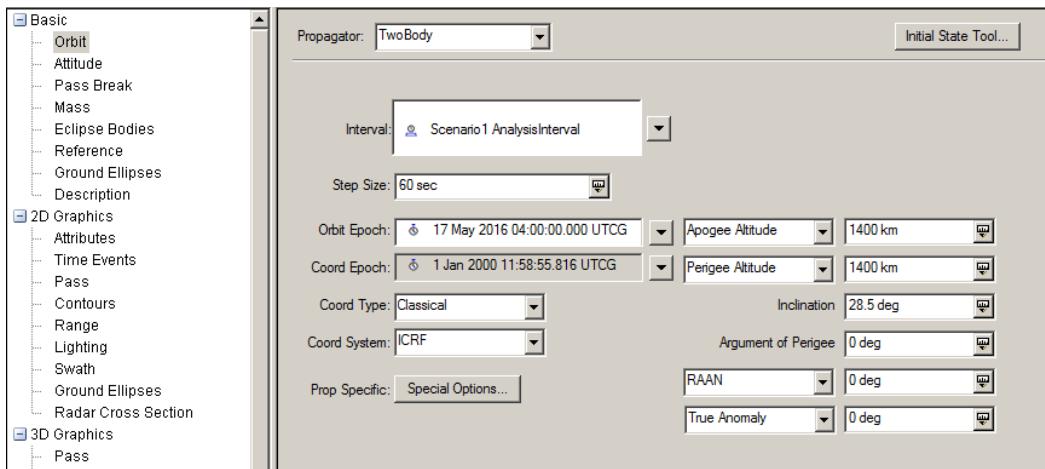


图2-55. STK卫星轨道设置

右键单击场景，选择 Properties Browser 进入场景属性设置界面。在 Basic-Time Period 中场景运行开始时间。这个时间和卫星轨道的 Epoch 时间共同决定了仿真开始的卫星的位置。如图 2-57 所示。

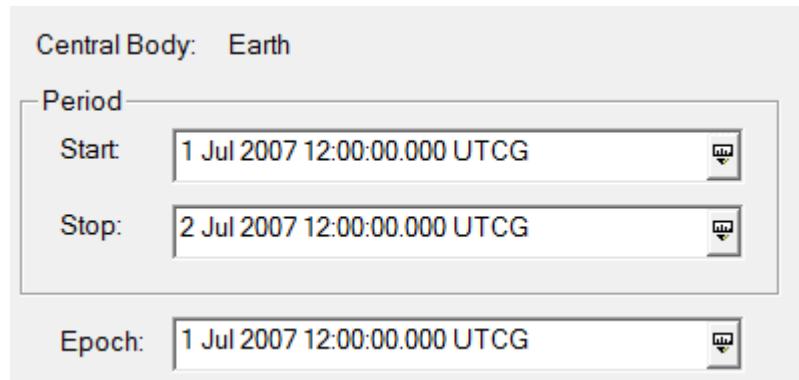


图2-56. STK场景属性设置

点击 ，运行方仿真场景，点击 File-Save 保存项目。在保存的项目文件夹下有与项目同名的 xx.sa 文件，即为轨道文件。

(2) 卫星节点应用卫星轨道

右键编辑卫星节点属性，在属性编辑窗口，可以看到比普通节点多出 Import STK Orbit 按钮，点击按钮，弹出卫星轨道导入窗口。点击 Browse 选择 STK 项目下的. sa 文件，点击确认即可。

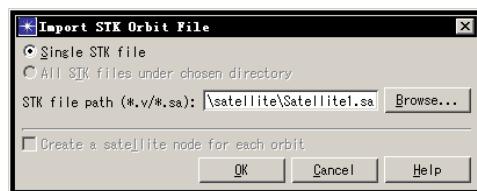


图2-57. OPNET导入卫星轨道文件

点击确认后. sa 文件自动转化为. orb 文件，在卫星节点 orbit 属性选择刚刚导入的轨道。若中途需要更改轨道文件，在卫星节点属性窗口，可以修改 orbit 属性的值。轨道视图

只能在顶层“top”网中可以看到。STK 中轨道如图 2-59 所示。OPNET 轨道如图 2-60 所示。

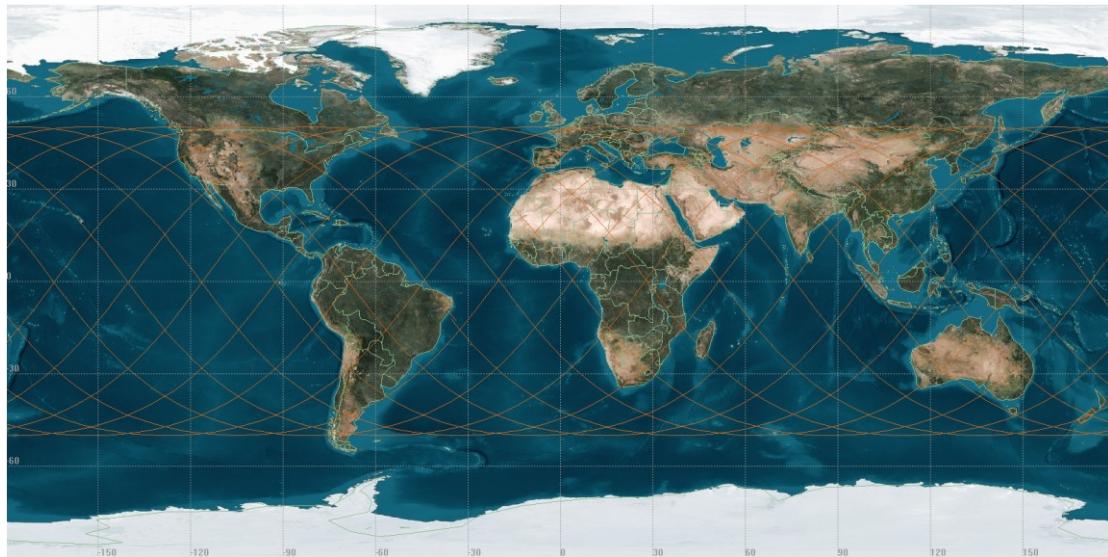


图2-58. STK卫星轨道视图

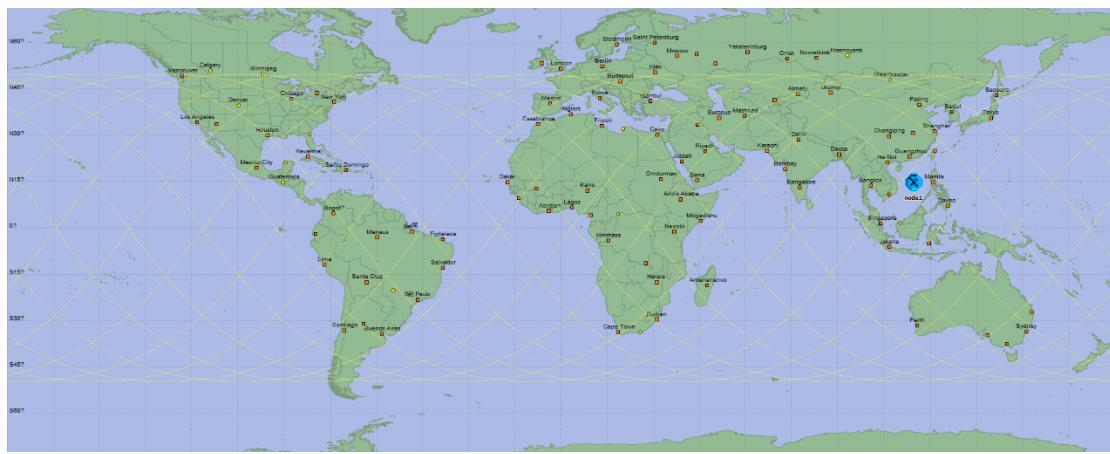


图2-59. OPNET卫星轨道视图

2.5.6、轨迹模块定义

除了前面讲述的分段移动方式，还包含向量移动方式和使用移动配置器的方式。

(1) 设置向量 (Vector) 轨迹方式

在物件的高级属性中设置方向 (bearing)、地面速度 (ground speed) 以及上升速度 (ascent rate)，只要地面速度和上升速度不为 0，物件就持续不断的运动，在仿真过程中也可以动态修改这 3 个参数，使物件运动状态发生改变。

(2) 使用移动配置器 (Mobility Config)

移动自组网的节点运动情况符合随机路点 (Random Way Point) 移动模型；在该模型中，每个移动节点被随机分配一个初始位置；随后节点在区域范围内随机选择一个目的点，然后以速度 v 从当前位置向该目的地运动，到达目的地后，随机停留一段时间 t_{pause} ，之后重复以上过程。通常情况下，速度大小 v 在区间 $[v_{min}, v_{max}]$ 上满足均匀分布；停留时间 t_{pause} 在 $[0, t_{pause}_{max}]$ 上满足均匀分布。

移动配置器收集在物件拼盘的 utility 工具箱中，用来设定无线移动的方式。在 Topology→Random Mobility→set mobility profiling……，移动配置器属性对话框如下图 6 所示。按照属性对话框中参数设置要求，得到节点运动轨迹如图 7 所示。

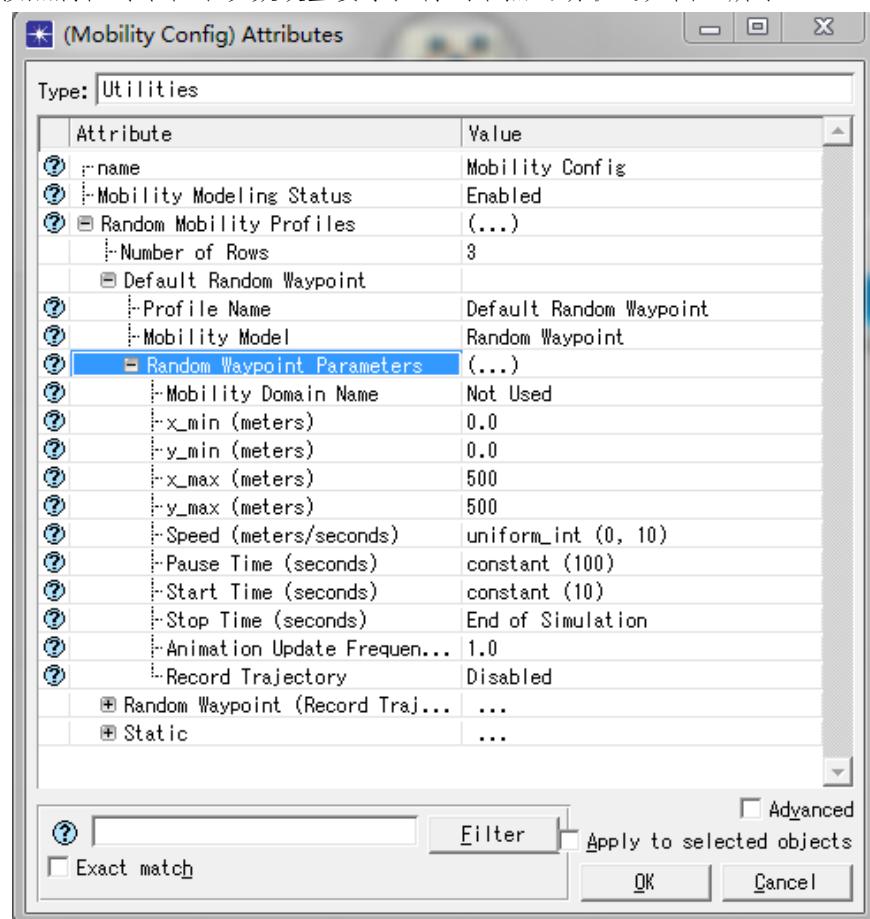


图2-60. 移动配置器属性对话框

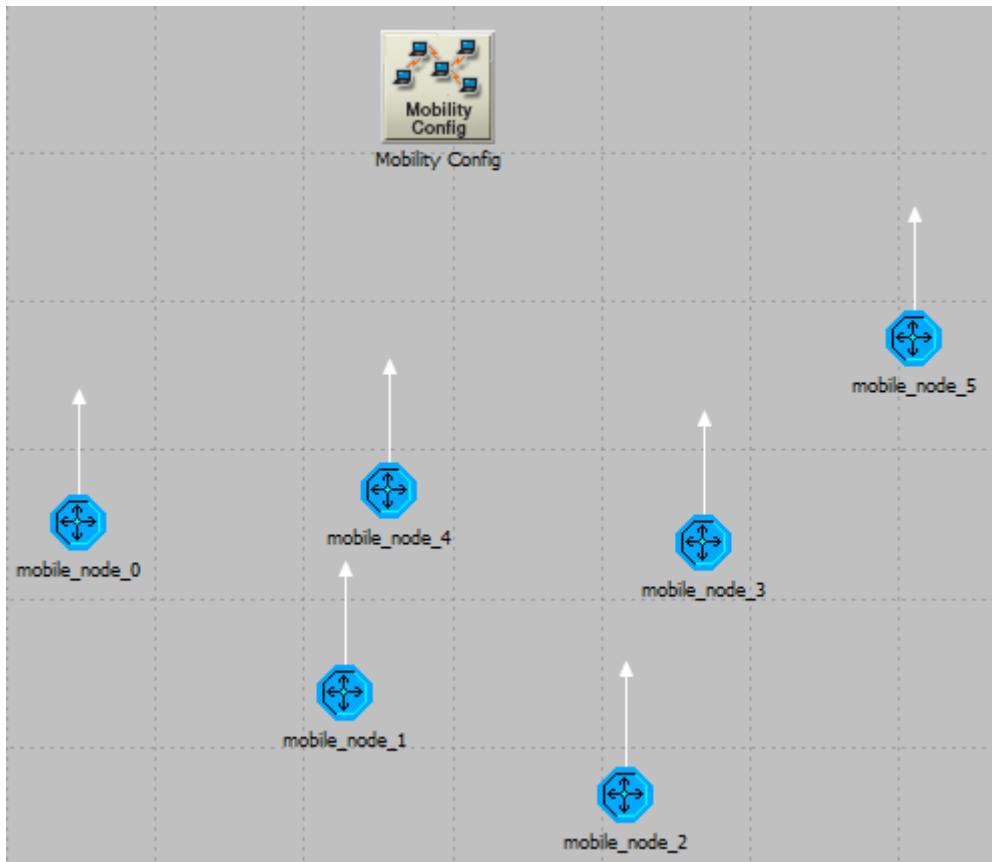


图2-61. 节点运动轨迹图

2.6、链路模型

链路模型主要用于模拟信道物理层协议，包括时延、噪声、信噪比、调制等过程，由信道参数、调制方式、链路模型、管道函数等组成。链路参数设置位置如表 2-12 所示。有线链路的信道参数在有线收发信机中设置，有线链路的链路模型和管道函数在链路模块中设置。无线信道中，由于没有链路模块，所以不包含像有线链路一样的链路模型，无线链路的信道参数、调制方式、管道函数均在无线收发信机中设置。有线链路和无线链路的信道参数和调制方式参考 2.3.3 收发信机章节。

表2-12. 收发信机可用参数设置

参数	设置位置	
	有线	无线
信道参数	有线收发信机	无线收发信机
调制	X	无线收发信机
链路模型	链路模块	X
管道函数	链路模块	无线收发信机

点对点链路：只能在固定节点间使用。

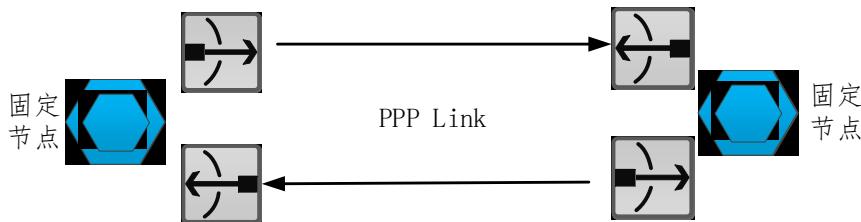


图2-62. 点对点链路传输示意图

总线链路：只能在固定节点间使用。

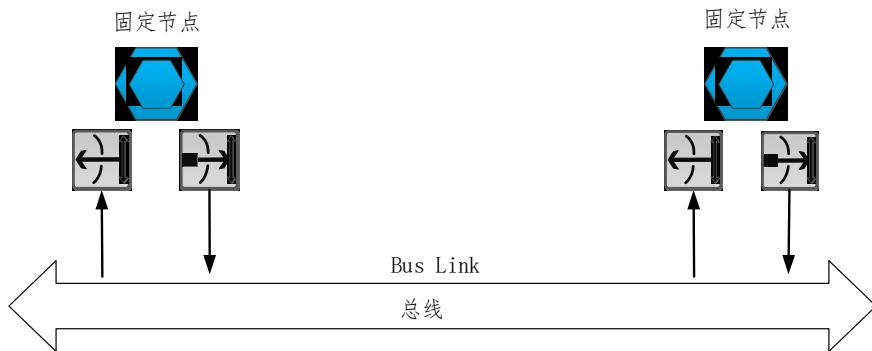


图2-63. 总线链路传输示意图

无线链路，可以在固定，移动，卫星节点间使用。无线链路不显示，物理层链路通过无线收发信机上的管道模型设置。无线链路可以在任何两个无线收发信机间建立。

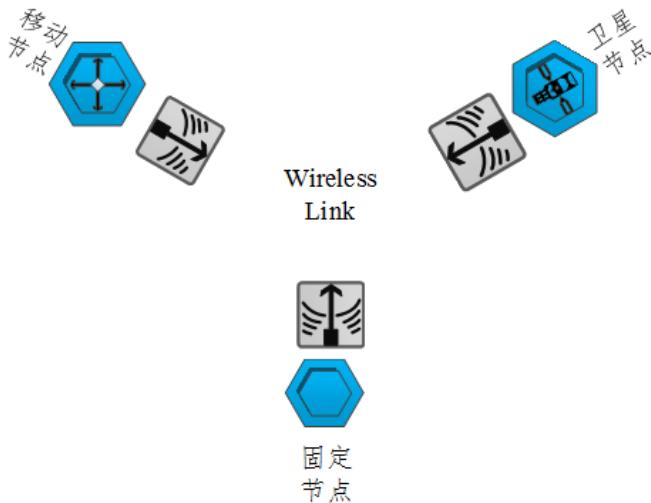


图2-64. 无线链路传输示意图

2.6.1、有线链路的链路模型

不同类型的链路模型应用于不同物理层传输模式的网络，在添加完节点模块后，需要添加链路模块来实现节点与节点间的传输。点击工具栏上模型选择按钮, 进入模型选择窗口。我们可以添加双工链路 Duplex Link Models, 单工链路 Simplex Link Models, 总线 Bus Models, 总线抽头 Bus Tap Models。

在添加有线链路模块以后往往需要声明较多的外部文件，这是因为在链路模型中调用了较多的外部函数。我们一般使用或修改 OPNET 中自带的链路模型。点击 File-New，选择 Link Model 可以进入链路模型编辑器，创建新的链路模型，存储为.lk.m 文件，点击 File-Open，选择 Link Model File (*.lk.m) 可以打开链路模型编辑器，修改链路模型文件。

在链路模型中对各个链路对象进行了说明。对于不同的链路对象，每一类链路都包含了特有的属性接口、注释以及表示方法。在项目编辑器中创建的链路是链路模型的特定实例，因此在对链路模型的属性进行修改时，链路实例会自动的继承修改后的属性。

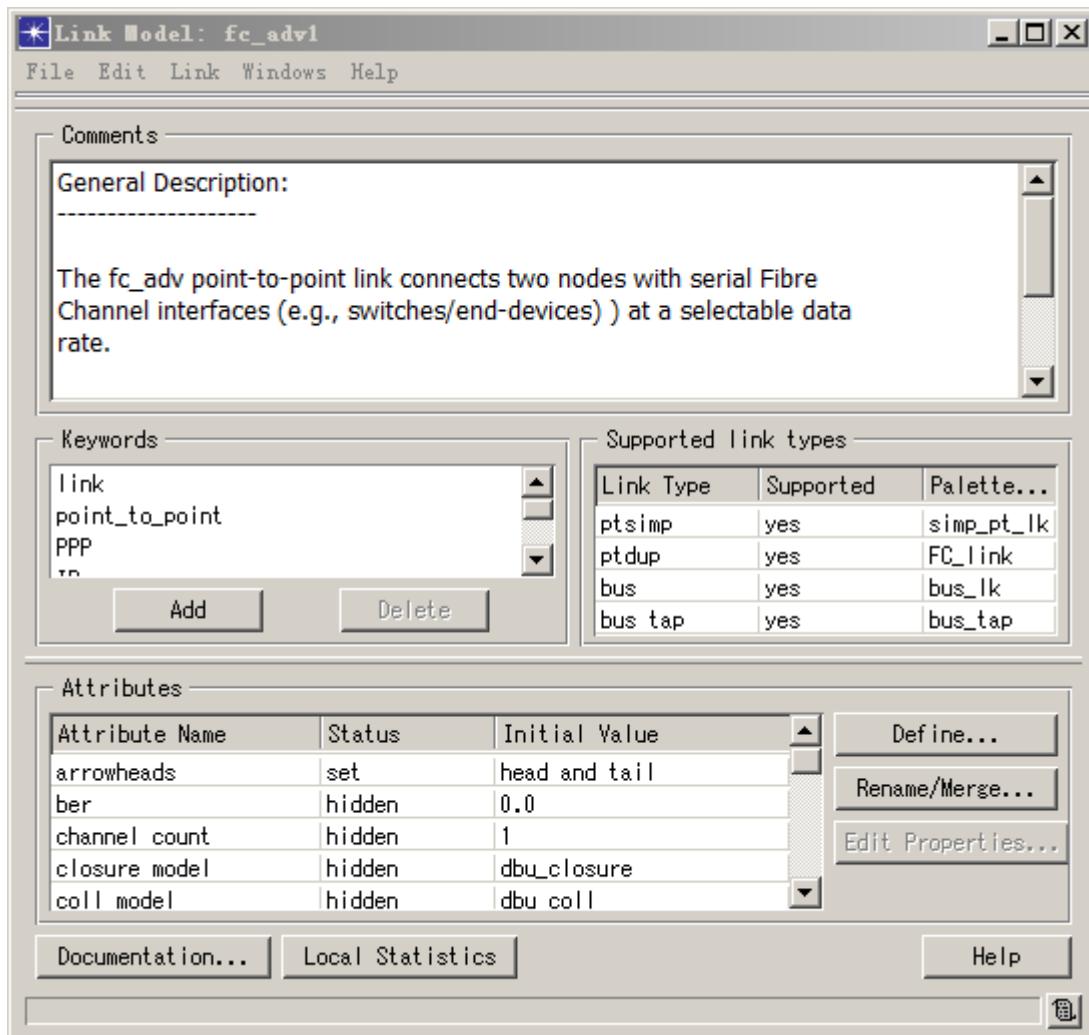


图2-65. 链路模型编辑器窗口

在链路编辑器对话框中规定了如下信息：

Support link Types (所支持的链路类型) 每一链路模型可以支持四种基本的链路类型中的一种或多种，即 ptsimp (点对点双工链路)、ptdup (点对点单工链路)、bus (总线链路) 和 bus tap (总线分接链路)。请注意，无线链路的设计不包含在链路编辑器中，它是由仿真内核 (Simulation Kernel) 经过动态定义生成的与结点相对位置、传输及运行环境中诸多因素相关的管道函数。

Keywords (关键字) 链路模型的关键字允许有选择的在项目编辑器对象面板中显示链路模型。在配置对象面板时，OPNET 将关键字与所请求的关键字进行比较，以此决定是否将此模型作为选择。此机制可减少在对象面板中的模型数，而只显示那些与当前应

用相关的模型。

Comments（模型注释）在链路模型中包含了一系列注释，这些注释描述了链路的特性、潜在应用和用户可能涉及的任何信息。因为有些用户无权访问链路模型内部，注释就成为此类用户可利用的主要信息。通过将模型接口的相关文档作为模型自身固有的部分嵌入到模型中，OPNET 为用户访问信息提供了便利。

Attribute(属性接口)结点和进程模型可以分别影响结点和模块的属性表达和使用，同样，链路模型为项目编辑器中链路对象的属性提供了规范说明。链路模型和链路之间的关系与进程模型和模块间的相互作用最为类似。和进程模型相似，链路模型中不包含可以提升属性的对象。因而，链路可以提升的唯一属性是“链路模型属性”。和进程模型一样，链路模型可以通过属性预分配、属性隐藏、属性重命名和改变属性优先级为链路对象的内嵌属性规定配置信息。在链路模型中可以修改 packet formats 属性，使得链路支持所有类型的数据包格式。

Local Statistics（本地统计量）可以为链路模型添加本地统计量。添加统计量的方法同 3.7 章节统计量的介绍。

2.6.2、管道函数

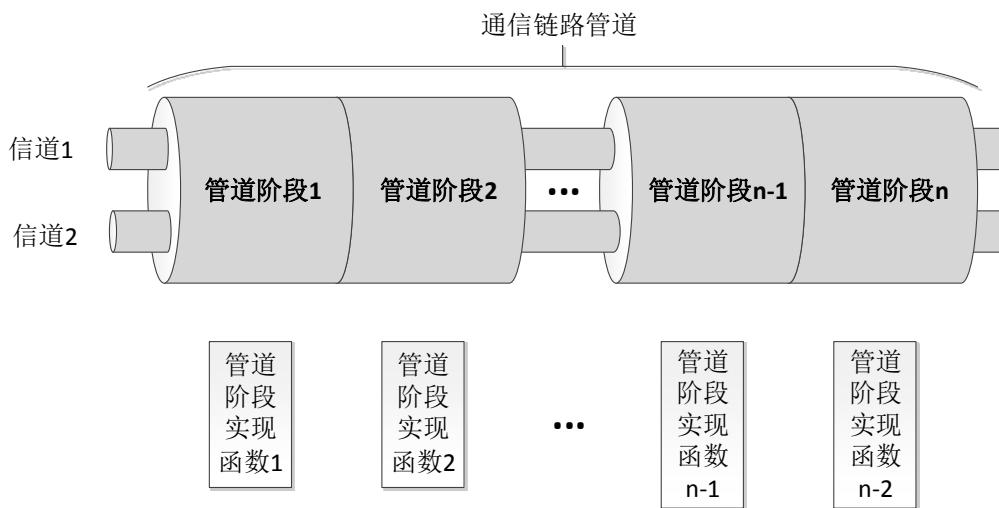


图2-66. OPNET管道阶段函数示意图

在 OPNET 模型中，当包被传送到发信机请求发送后，实际中的情况是：包将立即被发送到通信信道上进行传输，因此 OPNET 必须对通信信道进行建模，也就是在模型中要实现物理层的特征，以便将信道对包产生的传输效果考虑进整个网络模型。OPNET 将信道对包产生的传输效果建模为若干个计算阶段（称为 pipeline stage），最终来判断该包能否被接收到。

管道 (Pipeline) 阶段函数的典型输入参数是一个包指针，也就是说，管道阶段函数是针对每个包来计算它在物理信道上的传输效果的。为了记录管道阶段函数所需或计算的信道参数，每个包都包含着由 TDA (transmission data attribute) 的一组值构成的存储区，当包的传输效果计算进入某一管道阶段时，系统内核为 TDA 分配初始值或者根据计算结果来设置 TDA 值。这一组 TDA 值可以为后续的管道阶段提供计算的依据。用户可以对缺省的管道阶段函数进行修改以适应用户所需的信道类型：用户可以在管道里定义自己的 TDA，还可以调用系统内核里的支持对 TDA 进行操作的内核过程 (KP) 来编程实现自己的信

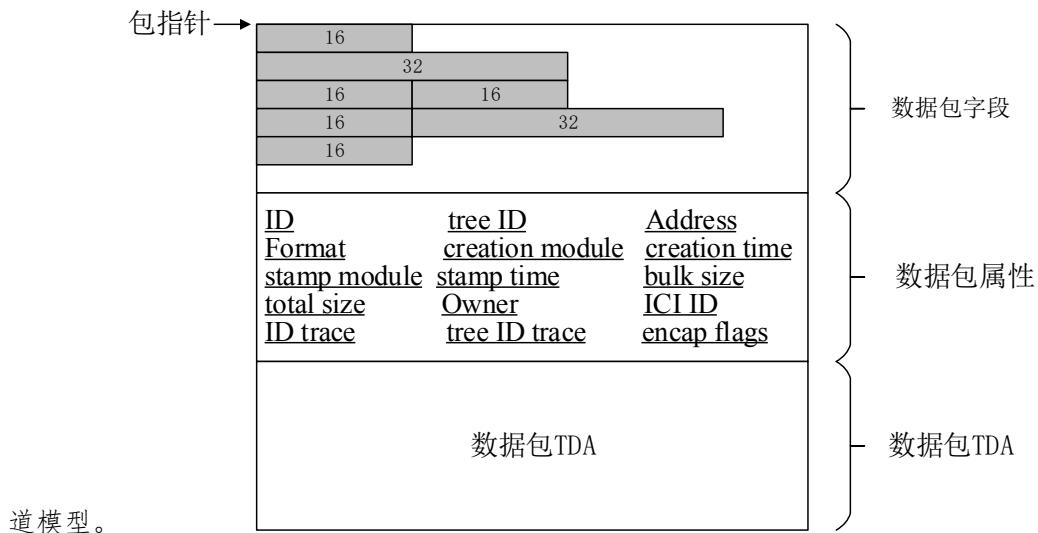


图2-67. OPNET数据包相关域

2.6.2.1、管道函数的创建修改

OPNET 中缺省的管道阶段函数文件后缀名为.ps.c，经编译后形成的目标文件后缀名为.ps.o。所有的三种信道的缺省管道阶段函数文件都存储在<opnet 目录>/<版本目录>/models/std/links/文件夹下面。用户若要自己编写管道阶段函数来代替缺省函数，则需先编写.ps.c 后缀的 c 或 c++文件，然后编译形成.ps.o 目标文件。修改方式如下所示。

默认存储位置：<opnet 目录>/<版本目录>/models/std/links/

函数前缀：

- *dpt_**: 点到点管道
- *dbu_**: 总线管道
- *dra_**: 无线管道

修改前最好将需要修改的文件复制到自己的项目中，不要修改原始文件。管道模型文件的创建、修改、编译都需要在 OPNET 中完成。自定义函数文件名要和文件名相同。管道函数文件为 xx.ps.c。

(1) 创建管道函数文件

点击 File-New 选择 PipeLine Stage (C code) 点击 OK 按钮。进入管道函数编辑界面。

```

1  #include "opnet.h"
2
3  void
4  dra_chanmatch_fdma (OP_SIM_CONTEXT_ARG_OPT_COMMA Packet * pkptr)
5  {
6      double tx_freq, tx_bw, tx_drate, tx_code;
7      double rx_freq, rx_bw, rx_drate, rx_code;
8      Vartype tx_mod;
9      Vartype rx_mod;
10
11     /* Determine the compatibility between transmitter and receiver channels. */
12     FIN_MT (dra_chanmatch (pkptr));
13
14     /* Obtain transmitting channel attributes. */
15     tx_freq = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_FREQ);
16     tx_bw = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_BW);
17     tx_drate = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_DRATE);
18     tx_code = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_CODE);
19     tx_mod = op_td_get_ptr (pkptr, OPC_TDA_RA_TX_MOD);
20
21     /* Obtain receiving channel attributes. */
22     rx_freq = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_FREQ);
23     rx_bw = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_BW);
24     rx_drate = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_DRATE);
25     rx_code = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_CODE);
26     rx_mod = op_td_get_ptr (pkptr, OPC_TDA_RA_RX_MOD);
27
28     /* For non-overlapping bands, the packet has no */
29     /* effect; such packets are ignored entirely. */
30     if ((tx_freq > rx_freq + rx_bw) || (tx_freq + tx_bw < rx_freq))
31     {
32         op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_IGNORE);
33         FOUT
34     }
35
36     if((tx freq > rx freq) && (tx freq+tx bw < rx freq+rx bw))

```

Opened File: (C:\Users\Administrator\op_models\mytdma.project\dra_chanmatch_fdma.ps.c) Line: 16

图2-68. OPNET管道阶段函数编辑界面

管道函数如图 2-67 所示，在编辑完后点击 Compile-compile 编译函数文件。无错后即可在链路设置中调用了。

(2) 修改管道函数文件

点击 File-Open，选择 pipeline stage(c code) File (*.ps.c) 文件类型，打开将要修改的管道函数文件。如图 2-67 所示。修改后点击 Compile-compile 编译函数文件。无错后即可在链路设置中调用了。

2.6.2.2、有线无线中的管道类别

在有线和无线链路中均存在管道阶段函数，但是由于链路的区别，他们并不包含相同的管道函数，表 2-13 给出了不同管道阶段在点到点通信、总线型通信和无线通信中的情况。空白表示不存在此管道阶段函数。

表2-13. 有线链路与无线链路管道模型区别

管道阶段名称	点到点有 线通信链 路	总线型有 线通信链 路	无线通 信链路	管道阶段的用途
收信机分组			1	初步筛选同类型接收机，用于减小计算量，与误码计算本身无关
传输时延	1	1	2	计算数据包的第一个比特到达发信机与最后一个比特发送完之间的时间差，即发信机处理数据包所用的时间

链路闭合		2	3	计算收发信机之间无线电信号的物理可达性
信道匹配		4		计算收发信机之间是否存在匹配的信道,或由于不完全匹配而产生的干扰的信道。
发信机天线增益		5		计算发信机在收发信机连线方向上的天线增益
传播时延	2	3	6	计算数据包第一个比特离开发信机天线后到达收信机天线前在空中传播过程中经历的时延。
碰撞		4		判断数据包传输过程中是否发生了冲突
收信机天线增益		7		计算收发信机在收发信机连接方向上的天线增益
接收信号功率		8		根据信道的数学模型计算数据包到达收信机时的有效功率
干扰噪声		9		计算包括背景噪声、信道不完全匹配、针对性干扰等原因导致的噪声,这里也需要使用信道的数学模型。
背景噪声		10		计算包括环境噪声和背景噪声在内的背景噪声功率
信噪比		11		根据信号功率和噪声功率,计算比值,影响后续误比特率的判断
误比特率		12		根据信噪比的动态变化计算数据包受影响部分的误比特率
误码分配	3	5	13	根据误比特率随机产生数据包的误比特数并累计
纠错	4	6	14	对包含误码的数据包进程纠错处理

2.6.2.3、点对点链路管道阶段函数

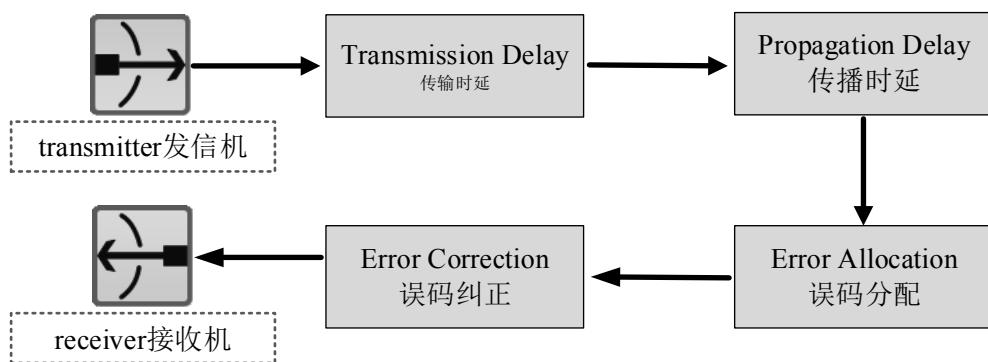


图2-69. 点对点链路阶段函数流程

(1) 传输时延阶段

函数功能	传输时延描述的是第一个比特发送时间到最后一个比特发送时间之间的时间间隔
------	-------------------------------------

	隔。
典型模型	dpt_txdel.ps.c
计算方法	<p>从包里读取传输该包的链路的标志号 (ID) ; <code>tx_ch_objid = op_td_get_int (pkptr, OPC_TDA_PT_TX_CH_OBJID);</code></p> <p>有了链路 ID 后，即可读取信道的数据速率； <code>op_ima_obj_attr_get (tx_ch_objid, "data rate", &tx_drate);</code></p> <p>读取包的长度； <code>pklen = op_pk_total_size_get (pkptr);</code></p> <p>传输时延=包长/数据速率； <code>tx_delay = pklen / tx_drate;</code></p> <p>把计算而得的传输时延值写到包的 TDA 里。 <code>op_td_set dbl (pkptr, OPC_TDA_PT_TX_DELAY, tx_delay);</code></p>

(2) 传播时延阶段

函数功能	传播时延描述的是第一个比特开始发送时间到第一个比特到达目标之间的时间间隔。
典型模型	dpt_propdel.ps.c
计算方法	<p>从包里读取传输该包的链路标志号 (ID) ; <code>link_objid = op_td_get_int (pkptr, OPC_TDA_PT_LINK_OBJID);</code></p> <p>有了链路 ID，即可读取链路的"delay"属性值； <code>op_ima_obj_attr_get (link_objid, "delay", &prop_delay);</code></p> <p>把该传播时延值写进包的 TDA 中； <code>op_td_set dbl (pkptr, OPC_TDA_PT_PROP_DELAY, prop_delay);</code></p>

(3) 误码数目分配阶段

函数功能	计算一条链路中一个数据包发生误码的数目。
典型模型	dpt_error.ps.c
计算方法	<p>读取链路的标志号 (ID) ; <code>link_objid = op_td_get_int (pkptr, OPC_TDA_PT_LINK_OBJID);</code></p> <p>读取链路的误码率"ber"属性值，即单个比特可能误码的概率； <code>op_ima_obj_attr_get (link_objid, "ber", &pe)</code></p> <p>读取包长； <code>seg_size = op_pk_total_size_get (pkptr);</code></p> <p>计算"正好发生 k 个比特误码"的概率 P(k)，那么可以得到"至多发生 k 个比特误码"的概率 P=P(0)+P(1)+……+P(k)；</p> <p>产生一个在{0, 1}内平均分布的随机数 r； <code>r = op_dist_uniform (1.0);</code></p> <p>如果随机数 r 小于等于"至多发生 k 个比特误码"的概率 P，那么就"认定"k 就是这个包在信道上传输的误码数目；</p> <p>如果 r 大于 P，那么就将 k 的值加 1，反复计算以得到算法能够接受的误码数目；</p> <p>将误码数目写进包的 TDA 里。 <code>op_td_set_int (pkptr, OPC_TDA_PT_NUM_ERRORS, ((int)num_errs))</code></p>

(3) 纠错阶段

函数功能	计算能纠正的码数目，并与误码数目比较，确定包能否被正确接收
典型模型	dpt_ecc.ps.c
计算方法	<p>读取接收器的标志号 (ID)； <code>rx_objid = op_td_get_int (pkptr, OPC_TDA_PT_RX_OBJID);</code></p> <p>读取接收器能纠正的误码数目门限值"ecc threshold"属性值； <code>op_ima_obj_attr_get (rx_objid, "ecc threshold", &ecc_thresh)</code></p> <p>读取包的长度； <code>pklen = op_pk_total_size_get (pkptr);</code></p> <p>读取前面计算的错误数目； <code>num_errs = op_td_get_int (pkptr, OPC_TDA_PT_NUM_ERRORS);</code></p> <p>将错误数目与纠错门限"ecc threshold"比较，判决该包是否能被正确接收； <code>accept = (((double) num_errs) / pklen) <= ecc_thresh) ? OPC_TRUE : OPC_FALSE;</code></p> <p>将判断结果写进包的 TDA 里。 <code>op_td_set_int (pkptr, OPC_TDA_PT_PK_ACCEPT, accept);</code></p>

2.6.2.4、总线型管道模型

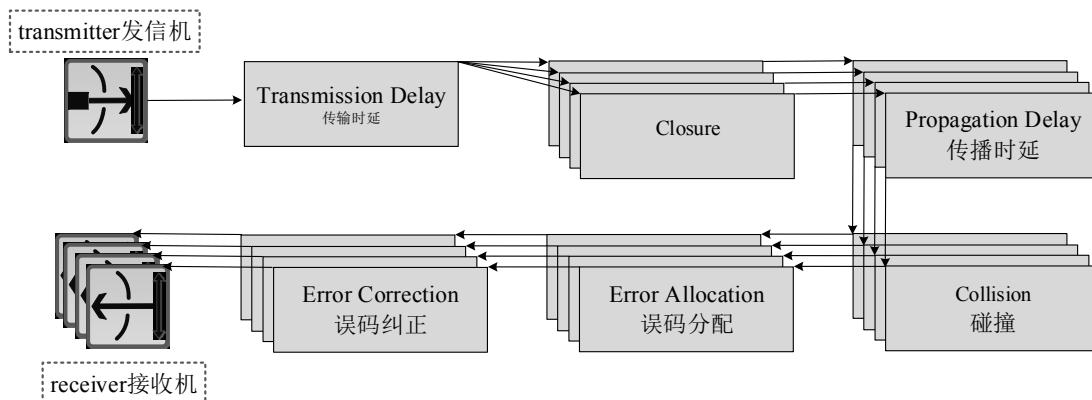


图2-70. 总线链路阶段函数流程

由六个缺省的 pipeline stage 组成，其中第一个阶段针对每个传输只计算一次，而后面的五个阶段针对各个可能接收到这次传输的接收器分别计算一次。

(1) 封闭性计算阶段

函数功能	这个阶段的意义在于判断各个接收器节点是否能够接收到这次传输，即链路的封闭性。针对每个接收器都有一个判断结果。有了这个结果以后系统内核就可以决定是否再为该接收器执行后面的计算进程。这个判断的好处是提高了仿真效率，因为若已知某接收器不能接收到这次传输，就不必为其计算传输时延，冲突等值，避免了进行不必要的计算。
------	---

典型模型	dbu_closure.ps.c
计算方法	<p>缺省认为所有 bus 上的站点都能接收到这次传输,因此直接把判断值写进包的 TDA 里。</p> <pre>op_td_set_int (pkptr, OPC_TDA_BU_CLOSURE, OPC_TRUE);</pre>

(2) 传输时延阶段

函数功能	传输时延描述的是第一个比特发送时间到最后一个比特发送时间之间的时间间隔。
典型模型	dbu_txdel.ps.c
计算方法	与点对点链路情况一致

(3) 传播时延阶段

函数功能	传播时延描述的是第一个比特开始发送时间到第一个比特到达目标之间的时间间隔。
典型模型	dbu_propdel.ps.c
计算方法	<p>读取链路的标志号 (ID)； <code>bus_objid = op_td_get_int (pkptr, OPC_TDA_BU_LINK_OBJID);</code></p> <p>读取链路的单位距离的传播时延 "delay" 属性值。注意在这里的 delay 属性与点对点链路的 delay 属性意义不一样。这里指的是单位距离的传播时延, 而点对点链路中的 delay 直接指的是总传播时延。因为点对点只涉及到单条链路的传播时延, 而总线链路要针对不同接收器即不同的传播距离计算出多个传播时延； <code>op_ima_obj_attr_get (bus_objid, "delay", &unit_delay)</code></p> <p>读取收发器之间的距离间隔； <code>prop_distance = op_td_get dbl (pkptr, OPC_TDA_BU_DISTANCE);</code></p> <p>二者乘积值即为传播时延, 将其写进包的 TDA 里。 <code>prop_delay = unit_delay * prop_distance;</code> <code>op_td_set dbl (pkptr, OPC_TDA_BU_PROP_DELAY, prop_delay);</code></p>

(4) 冲突检测阶段

函数功能	在某个包的整个接收时间内 (第一个比特到达时间到最后一个比特到达时间之间的时间间隔), 可能会发生多次传输事件, 于是对于该包来说, 可能要遭遇多次冲突事件。在 OPNET 中, 每当发生一次冲突事件, 就调用本 pipeline stage 一次, 以记录这次冲突事件。这个 pipeline stage 对每个包传输不是总要调用, 它只是在发生冲突时调用, 而是否发生冲突是由系统内核来判别的。这个计算进程区别于其他的 pipeline stage, 有两个包指针参数: 第一个是先到的分组, 第二个是后到的分组 (就是触发冲突事件的那个)。
典型模型	dbu_coll.ps.c
计算方法	<p>如果前一个包刚好在后一个包开始传输时结束了接收, 则不考虑为一次冲突。因此读取前一个包的结束时间, 将其与当前仿真时间进行比较。</p> <p>如果相等则不认为冲突。</p> <pre>op_td_get dbl (pkptr_prev, OPC_TDA_BU_END_RX) != op_sim_time ()</pre>

	<p>如果不等，则将前后两个包的记录冲突次数 TDA 都加一。</p> <pre>op_td_set_int (pkptr_prev, OPC_TDA_BU_NUM_COLLs, op_td_get_int (pkptr_prev, OPC_TDA_BU_NUM_COLLs) + 1); op_td_set_int (pkptr_arriv, OPC_TDA_BU_NUM_COLLs, op_td_get_int (pkptr_arriv, OPC_TDA_BU_NUM_COLLs) + 1);</pre>
--	--

(5) 误码数目分配阶段

函数功能	计算一条链路中一个数据包发生误码的数目。
典型模型	dbu_error.ps.c
计算方法	与点对点链路的计算方法一致，根据误码率计算误码数目

(6) 纠错阶段

函数功能	包能被正确接收的判断标准与点对点链路稍有不同。首先是要求包未经冲突，然后将误码数目与纠错门限比较判断可正确接收与否。
典型模型	dbu_ecc.ps.c
计算方法	<p>读取包的冲突数目；</p> <pre>op_td_get_int (pkptr, OPC_TDA_BU_NUM_COLLs)</pre> <p>如果冲突数目不为 0 或节点被 disabled，则直接判断为不能正确接收；</p> <pre>(op_td_get_int (pkptr, OPC_TDA_BU_NUM_COLLs) != 0) (op_td_is_set (pkptr, OPC_TDA_BU_ND_FAIL))</pre> <p>将误码数目与纠错门限比较以决定能否正确接收，将判断结果写进包的 TDA 里。</p> <p>具体步骤与点对点链路一致。</p>

2.6.2.5、无线链路管道阶段函数

无线链路的管道阶段相对于有线链路来说较为复杂。对于无线链路网络仿真或多或少的牵扯到物理层网络协议，这也是无线网络管道阶段复杂的原因。即便具有这么多无线管道阶段，OPNET 对物理层技术的仿真仍然十分有限，需要与 MATLAB 进行联合调试才能更加完美的实现物理层技术。

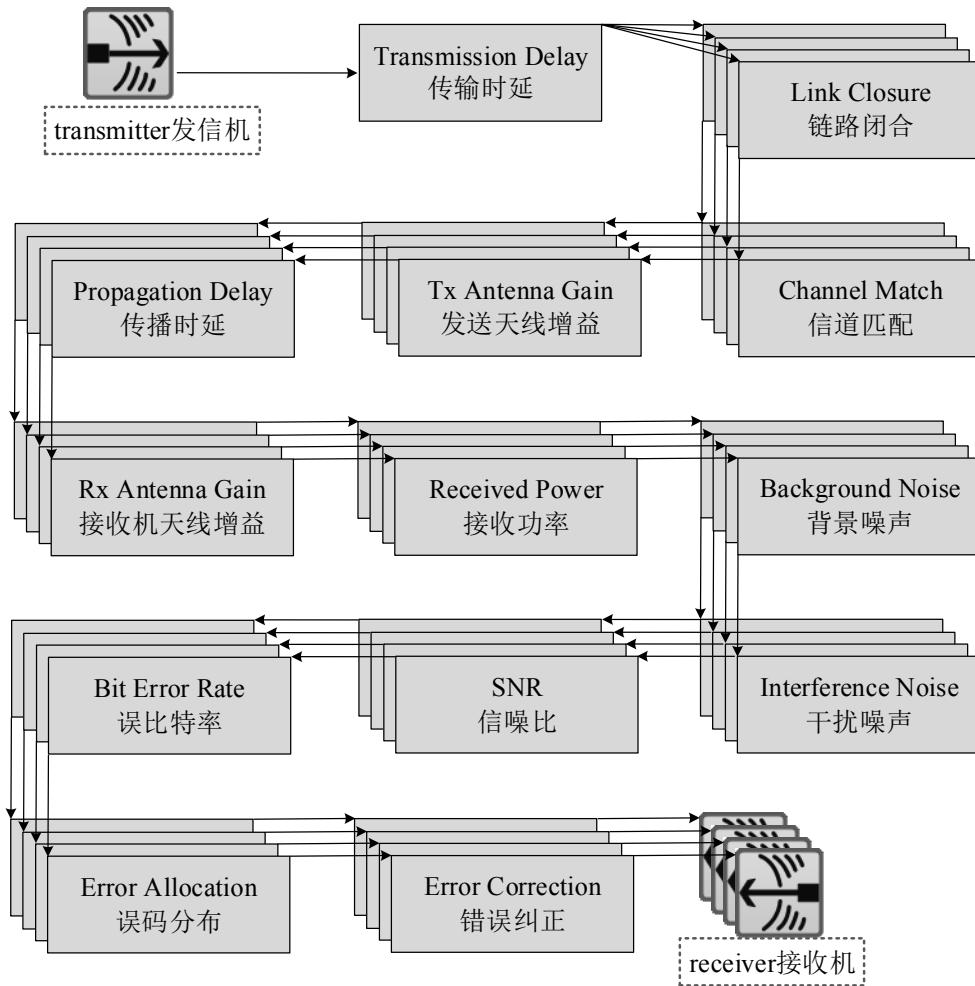


图2-71. 无线链路阶段函数流程

在无线收发信机中使用的管道阶段函数必须和文件名一致，否则无法编译通过。OPNET 使用的无线链路管道阶段函数对应的内嵌属性、所属模块、参考设置如表 2-14 所示。

表2-14. 无线链路收发信机管道函数分布

管道阶段名称	对应属性名	所属模块	参考设置值
收信机分组	rxgroup model	发信机	dra_rxgroup
传输时延	txdel model		dra_txdel
链路闭合	closure model		dra_closure
信道匹配	chanmatch model		dra_chanmatch
发信机天线增益	tagain model		dra_tagain
传播时延	propdel model		dra_propdel
收信机天线增益	ragain model	收信机	dra_ragain
接收信号功率	power model		dra_power
干扰噪声	inoise model		dra_inoise
背景噪声	bkgnoise model		dra_bkgnoise
信噪比	snr model		dra_snr
误比特率	ber model		dra_ber
误码分配	error model		dra_error

纠错	ecc model		dra_ecc
----	-----------	--	---------

由于无线链路是一种广播媒介，每一次传输都可能影响到整个网络模型中的多个接收机。另外，对于某个特定包的发送，每个接收机的无线链路可能呈现不同的行为和定时特征。因此每个接收机都必须执行不同的管道阶段。

无线收发管道包括 14 个管道阶段，大多数必须在每个接收机中执行。然而，第一个阶段(接收机组)在每对收发机中只执行一次，以建立发送与接收机信道之间的静态绑定。第二个阶段(发送时延)用来计算一个对所有目的结点都适用的数值，因此每次发送可以只执行一次。最后，每个管道序列可以也可完全不需要执行，这依赖于第三个阶段(闭合阶段)的结果，因为该阶段负责判断收发机之间是否可以通信。类似的，第四个阶段(信道匹配)考虑收发信机之间的信道匹配的影响，进行接收可达性的判断，这样避免了管道序列必须执行到最后一个管道。

必须注意到无线链路最后的几个阶段对于每个接收机可能执行多次，这是由于可能有多个包同时发送。为了检测在同一接收机信道的多个包的接收，仿真内核为每个无线接收机信道保留了两个“当前”数据包的表单。第一个表只包含信道匹配阶段检测是“有效”的数据包；第二个表包含了“无效”的数据包。通常，这要求发射与接收机管道之间存在匹配的特性，也可以是接收机信道能够与数据包信号保持同步。在不同的表单中保留“无效”和“有效”包，主要是因为这样使得仿真内核可以只对“有效”的包执行特定的管道阶段和计算特定信道的统计数据。例如，对于接收机不接收的包来说，就不需要计算信噪比、比特错误率或者错误分布参数。通常，接收功率阶段之后的所有阶段都只适用于“有效”的数据包。

管道阶段 9 到阶段 12 用来估计链路的状况，以及相应信号状态的变化。通常至少调用阶段 10 到阶段 12 其中之一来估计有效数据包持续时间内的性能。然而，当干扰包到达时，需要调用阶段 9 到 12 之间的每个阶段，以计算新的信号状况。由于无线链路不作为物理对象而存在，所以用来进行特定的无线传输的管道阶段必须与形成链路的发射机与接收机联系起来。

第一部分：无线发信机管道阶段流程

无线链路的管道阶段函数在无线收发信机的属性中配置。右键单击无线发信机，可以进行无线发信机的信道 channel 参数和管道函数进行配置。

Attribute	Value
rn-name	send
channel	(...)
Number of Rows	1
Row 0	
-data rate (bps)	1,024
-packet formats	all formatted, unformatted
-bandwidth (kHz)	10
-min frequency (MHz)	30
-spreading code	disabled
-power (W)	100
-bit capacity (bits)	infinity
-pk capacity (pkts)	1,000
-modulation	bpsk
-rxgroup model	dra_rxgroup
-txdel model	dra_txdel
-closure model	dra_closure
-chanmatch model	dra_chanmatch
-tagain model	dra_tagain
-propdel model	dra_propdel
-icon name	ra_tx

图2-72. 无线发信机属性设置窗口

数据包在无线发信机阶段经历的过程如图 2-71 所示。其中在发信机上每一个数据包副本都各自经历自己的管道阶段函数。

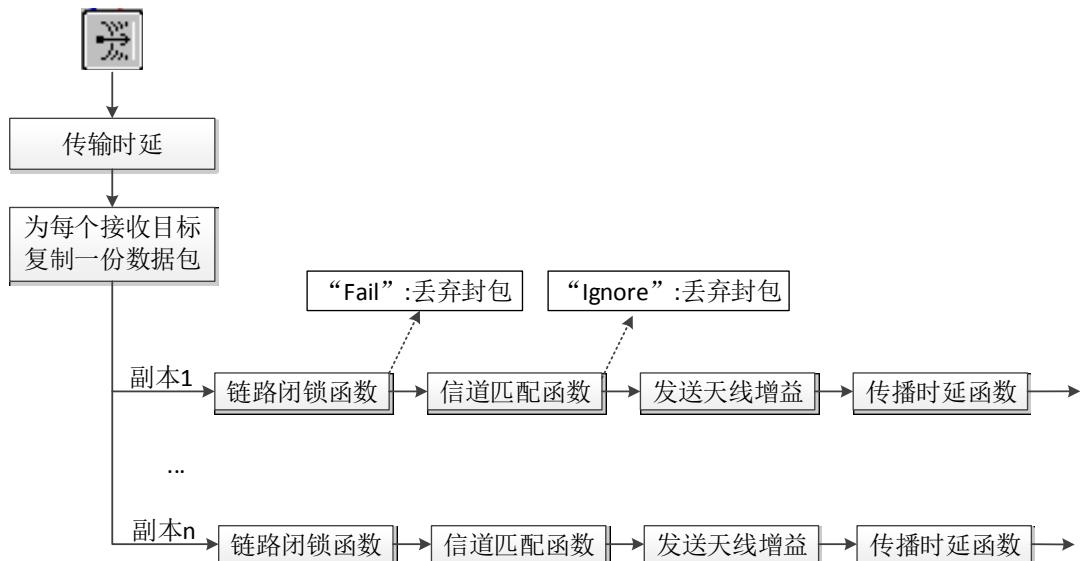


图2-73. 无线发信机管道函数运行流程

流程说明：

数据包在无线发信机阶段经历的过程较为简单。首先无线发送模块会为每个接收模块复制一个副本，以使每个接收模块对数据包的处理都是各自独立的。每一个数据包副本在每个链路上运行及设置相应的 TDA 参数。无线发信机管道阶段首先计算收发节点间是否允许通信。节点间是否被地球阻拦。默认情况使用的为 dra_closure 模型，即节点间被地球阻拦时不允许通信。dra_closure_all 模型为节点间被地球阻拦时允许通信。如果不允许通信则丢弃包，不进行后面的函数。在包通过链路闭合函数后，执行信道匹配函数，信道匹

配函数将包处理成有效包、忽略包。这里不能直接丢弃包，因此这些包有可能在接收机内影响其他包的接收。后面会通过发信机定向天线模型计算发送天线增益。没有定向天线时，默认为全向天线，各方向增益为 0。随后计算传播时延，即数据包在空间链路上需要的时间，依次来确定什么时间开始启动接收机管道函数。

第二部分：无线接收机管道阶段流程

Attribute	Value
② r-name	receive
② ↴ channel	(...)
② ↴ Number of Rows	1
② ↴ Row 0	
② ↴ ↴ data rate (bps)	1,024
② ↴ ↴ packet formats	all formatted, unformatted
② ↴ ↴ bandwidth (kHz)	10
② ↴ ↴ min frequency (MHz)	30
② ↴ ↴ spreading code	disabled
② ↴ ↴ processing gain (dB)	channel_bw/dr
② ↴ ↴ modulation	bpsk
② ↴ ↴ noise figure	1.0
② ↴ ↴ ecc threshold	0.0
② ↴ ↴ ragain model	dra_ragain
② ↴ ↴ power model	dra_power
② ↴ ↴ bkgnoise model	dra_bkgnoise
② ↴ ↴ inoise model	dra_inoise
② ↴ ↴ snr model	dra_snr
② ↴ ↴ ber model	dra_ber
② ↴ ↴ error model	dra_error
② ↴ ↴ ecc model	dra_ecc
② ↴ ↴ icon name	ra_rx

图2-74. 无线接收机属性设置窗口

数据包在无线接收机阶段经历的过程如图 2-73 所示。

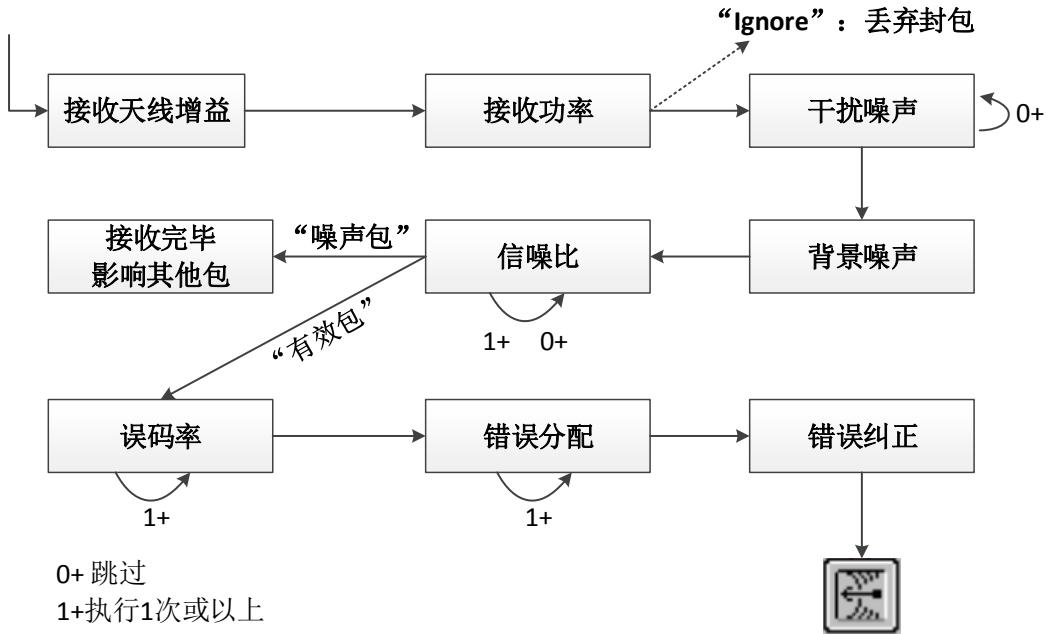


图2-75. 无线接收机管道函数运行流程

流程说明：

数据包进入无线接收机以后不管是否是有效包，都会先计算接收天线增益和接收功率。

1 数据包刚开始接收时，若不存在前面的数据包正在接收，则经历：（接收天线增益 → 接收功率 → 背景噪声 → 信噪比 → 暂停），若接收中没有后面的数据包冲突，则在接收完成后进入（误码率，误码数目，纠错阶段），若接收中有后面的数据包冲突，则和后面的数据包一起进入（干扰噪声阶段）。

2 数据包刚开始接收时，若存在前面的数据包正在接收，则经历（接收天线增益 → 接收功率 → 干扰噪声）

在前面1、2两种情况中的进入干扰噪声后，若噪声包在有效包接收范围内接收开始或接收结束两个时刻，有效包在经历了（背景噪声 → 信噪比）的基础上都要重新经历（误码率 → 误码数目 → 信噪比）。

在数据包接收完成时，计算误码率，误码数目，纠错阶段。

接收功率阶段会重写包的有效性，干扰噪声阶段不重写数据包的有效性。

第三部分：无线链路管道阶段函数

阶段0：接收机组

接收机组阶段实际上并不是处理发送的动态管道的一部分。但是，由于它能计算出结果，影响无线传输的行为特征，它被认为是部分管道，由无线发射机的“rxgroup model”属性表示。

当无线发送开始时，仿真内核通过在发送信道与一组接收机信道之间实现多个无线链路，来模拟无线链路的广播特性。每个发送机信道都保留着其自身的信道“接收机组”，这是可能的接收传输的候选信道。接收机组阶段的目的是为每个发射机信道创建一个初始

的接收机组。仿真内核检查每种可能的发送接收机信道对，并为每个发射机信道构建一个接收机组。

然而这样引发了一个问题：由于收发机特性在仿真中动态变化，通常在开始时很难判定哪个接收信道与发送信道匹配对应。因此，接收机组阶段包括了一个接收信道，除非它能够事先判定发送信道永远不与该接收信道对应。仿真过程中，随后的管道阶段可以动态的判定接收信道能否接收传输的包。仿真中使接收信道不能接收的可能的原因包括：

- (1) 分离的频带。频带由其基频和带宽描述。如果发送信道与接收信道的频带不交叠，那么发送信道的发送不会影响接收信道一或者作为有效的信号或者当成噪声。
- (2) 物理分离。发送信道与接收信道之间可能相隔很远，而建立一条无线链路将没有足够的信号强度。无线链路的建立也取决于发射机接收机天线高度、发送信号功率以及频率等因素。
- (3) 天线失效。管道阶段分别在阶段 4 和阶段 6 对发射机与接收机天线的增益模型进行建模。当应用方向性天线时，这些阶段计算出的结果可能极大的衰减信号功率——以至于仿真可以合理的忽略发送对接收信道的影响。在某些网络模型中，仿真内核可以判定发送接收信道对之间是完全不能进行通信的。这种情况下，将接收信道从发射机信道的接收机组中移去可以加速仿真，因为这样减少运行管道阶段，而不产生影响。虽然接收机组在仿真开始之前被调用，但是结果不应取决于可能在仿真中变化的因素。例如两个移动阶段之间最初的距离以及分配给发送和接收信道初始的频率(假设模块能动态改变这些频率)。缺省情况下，这些情况下 OPNET 包含接收信道，所有的接收机组在整个仿真中保持不变；随后的管道阶段于是可以利用动态标准来估计发送接收机信道之间的连接性。

仿真内核要求接收机组阶段程序分别接收发送和接收信道的“对象 ID”。程序应该返回一个整数值 (OPC_TRUE 或 OPC_FALSE) 给内核；该数值表明了接收机信道是否为一个合适的目的端，而且应当包含在接收机组中。

可以利用核心函数(KP) 中的无线函数包来改变缺省的接收机组性质，以及针对仿真事件动态的改变和重新计算接收机组。例如，如果在仿真进程中接收机结点被屏蔽掉，可以从接收机组中移去接收信道。同样可以使用函数 op_radio_txch_rxgroup_compute()，在仿真中的任何时间计算或者重新计算给定信道接收机组(本质上，这是重新调用接收机组管道机阶段)。甚至可以完全“跳过”接收机组阶段，接着按需要构造并更新接收机组。(跳过这个阶段涉及到设置“rxgroup model”为“dra_no_rxgroup”而不是缺省值

“dra_rxgroup”，这样就为所有的发送信道创建了空接收机组)。动态的更新接收机组可以进行更快速的仿真，尤其是对高无线通信量的网络模块。“跳过”管道阶段以及按需要创建接收机组也可加速有大量收发信道的网络的仿真。

在接收机组中的一个常用设置就是当收发信机完全匹配时，希望自己发送的节点不被自己接收，在 dra_rxgroup.ps.c 中函数体 FRET (OPC_TRUE) 前加入

```
if(op_topo_parent(op_topo_parent(op_topo_parent(PRG_ARG_UNUSED(tx_obid))))==op_topo_parent(op_topo_parent(op_topo_parent(rx_obid))))
{
    FRET(OPC_FALSE);
}
```

接收机分组函数典型实现方法：

函数功能	这个阶段只在仿真一开始时调用一次，以评估每一对收发器信道之间的连通性。不同于其他 pipeline stage 的是，这个阶段不是针对包进行操作的。它是针对每对收发器信道进行判断连通性的。
典型模型	dra_rxgroup.ps.c

计算方法	缺省认为所有的接收器信道都是任一发送器潜在的目的站，即任何一对收发器信道间都默认为是连通的。
可选函数	dra_rxgroup、wlan_rxgroup、wimax_rxgroup、umts_dra_rxgroup

阶段 1：传输时延

发送时延阶段数值上是无线管道的第二个阶段，但当发送新的数据时却是首先执行的。它由无线发射机的“txdel model”属性描述，开始发送数据包时立即执行。这是唯一的阶段——只执行一次而对所有的管道都适用。

调用该管道来计算整个包完成发送所需要的时间。这个时间是第一个比特开始发送的时间与最后一个比特完成发送之间的仿真时间。仿真内核利用该阶段返回的结果在发送包的发送信道中调度一个结束发送事件。当该事件发生时，发射机可以在信道内部的队列中开始发送下一个包；或者发送信道进入空闲状态。另外，发送时延与传播时延结合起来计算包被链路目的模块接收的时间（例如，最后一个比特到达的时间就是它发送完成的时间加上在链路上的传播时延）。

仿真内核要求发送时延阶段函数将包地址作为其唯一的输入参数。计算出的发送时延由内核符号常量 OPC_TDA_RA_TX_DELAY 表示。分配值应当是非负的双精度浮点值。

传输时延函数典型实现方法：

函数功能	传输时延描述的是第一个比特发送时间到最后一个比特发送时间之间的时间间隔。
典型模型	dra_txdel.ps.c
计算方法	<p>读取信道的传输速率；（这里与点对点和总线不一样，只要直接从包里的 TDA 读取就可以了，无须在程序里取得链路的标志号 ID，再读链路属性值。因为无线链路不存在独立的链路实体，因此传输速率不可能设在链路属性里，而是设在无线发送器的信道属性里，包括频率，带宽，数据速率等。系统内核已经将这些参数写进包里。）；</p> <pre>tx_drate = op_td_get dbl (pkptr, OPC_TDA_RA_TX_DRATE);</pre> <p>读取包长；</p> <pre>pklen = op_pk_total_size_get (pkptr);</pre> <p>传输时延=包长/传输速率，并写进包的 TDA 里。</p> <pre>tx_delay = pklen / tx_drate;</pre> <pre>op_td_set dbl (pkptr, OPC_TDA_RA_TX_DELAY, tx_delay);</pre>
可选函数	dra_txdel、dpt_txdel、dbu_txdel、wimax_txdel、docsis_txdel、wlan_txdel、umts_dra_txdel、txdel_zero

阶段 2：链路闭合

闭合阶段是管道的第三个阶段，由无线发射机的“closure model”属性描述。它在发送信道目的信道设置中所指定的接收信道中只执行一次，在发送时延阶段返回后立即执行，期间没有仿真时间过去。该阶段的目的是判断一个特定的接收信道是否受发送的影响。发送到达接收信道称之为发送信道与接收信道之间的闭合，正如该阶段的名字。

无线链路的封闭性计算是依据通视性来决定的。算法测试连接发送器和接收器之间的线段是否和地球表面相交。若存在交点，那么认为接收器不可达，即其不可能接收到这次传输。

在节点创建时，节点的海拔高度属性默认为 0。创建无线发信机时，链路闭合模型默认选择的是 dra_closure，即不可穿透地球。这样两个无线节点间海拔都是 0，节点间距离又不为 0，节点又不能穿透地球发射信号，所以节点间不能通信。只要将节点海拔高度设置 10m 或 100m 即可，或者选择 dra_closure_all 可穿透地球发送信号的链路闭合模型。但同时需要注意信号可穿透地球产生的其他情况。

在 opnet 中地球半径取 6378000m，放置节点时，节点海拔默认为 0，链路闭合模型默认为 dra_closure，即信号不可穿透地球，无论节点间距离有多近，只要不在同一个节点位置，则此时两个节点间是不可通信，如图 xx 中 A 图所示。当需要计算设置多大高度时，根据三角形正余弦定理计算，例如当收发节点天线高 32m 时，两节点视距为 40Km。

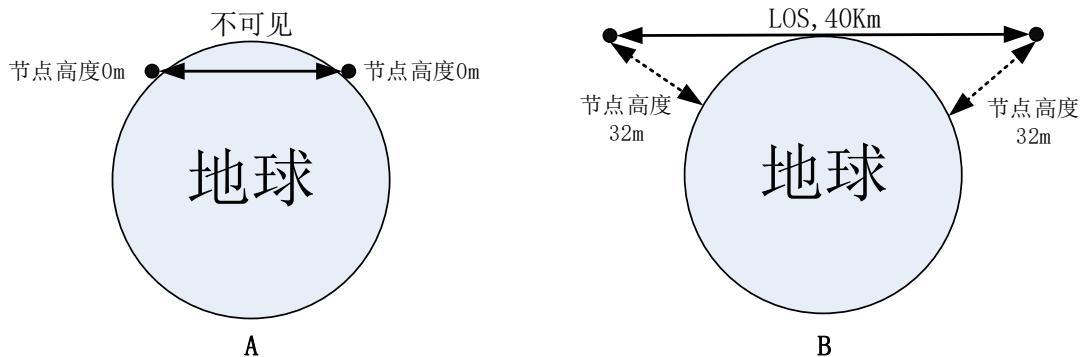


图2-76. 链路闭合管道模型示意图

注意到闭合阶段的目的不是判断是否某个传输对特定的信道是有效的或恰当的，而是发射信号能否得到候选的接收信道；通常，该阶段所进行的计算大多数基于物理上的考虑，例如障碍物、地球的表面等。

仿真内核要求该阶段得到一个整数值。内核利用其判断是否执行下面的无线管道阶段。如果该整数值等于符号常量“OPC_TRUE”，那么就建立闭合，信号在发送与接收机之间的传送就可能；否则符号常量 OPC_FALSE 用以表明不可以。

仿真内核要求该阶段将包地址作为其唯一的输入参数。闭合标志由内核符号常量 OPC_TDA_RA_PROP_CLOSURE 表示。

链路闭合函数典型实现方法：

函数功能	无线链路的封闭性计算是依据通视性来决定的。算法测试连接发送器和接收器之间的线段是否和地球表面相交。若存在交点，那么认为接收器不可达，即其不可能接收到这次传输。因此导致与该包相关的剩下的所有 pipeline stage 不再执行。判断结果将写进包的相应 TDA 里。
典型模型	dra_closure.ps.c
计算方法	收发器在地心坐标系统上的坐标由用户在收发器属性里预先设定。系统内核已自动将其写进包里。 读取包里已有的收发器坐标，由一定算法计算是否与地球表面相交。 将链路可达性判断结果写进包的 TDA 里。
可选函数	dra_closure、umts_dra_closure、dbu_closure、docsis_closure、dra_closure_all

阶段 3：信道匹配

信道匹配阶段包括载频匹配、带宽匹配、速率匹配、编码匹配、模式匹配。

根据载频和带宽的关系或速率、编码和模式的关系，可将数据包分为三种情况：忽略、干扰、有效。

信道匹配阶段是管道的第四个阶段，由无线发射机的“chanmatch model”属性指定。它在每个满足链路闭合的接收机信道中都执行一次。在链路闭合阶段返回后立即触发该阶段，期间没有仿真时间过去。该阶段的目的是根据接收信道对发送进行分类。三种可能的种类之一必须指定给包，如下所示：

(1) 有效性。该类的包被认为是与接收信道匹配，可以被接收并转发到接收节点中的别的模块，只要其不受过多错误的影响。将数据包分类为有效的通常取决于是否发送接收机之间关键参数有至少一项符合。

(2) 噪声。这种分类用来表示数据内容不能接收的包，但对接收信道产生干扰而影响其性能。发射与接收信道配置的不匹配，就将包分类为有噪声的。

(3) 忽略。如果一次发送决定对接收信道的状态或性能都不产生影响，那么它应被分为此类。仿真内核将停止执行发送信道与接收信道之间的管道阶段(未来的信道之间的传输将不会再被阻止)。

默认情况下 dra_chanmatch 在参数完全匹配时信号接收正常，存在重叠带宽时认为是噪声，频带不重叠，丢弃数据包。但是当需要仿真实现 FDMA 形式的多频多用户同时通信时，如下图所示。就需要修改信道匹配的管道模型，添加进入当接收主节点的带宽包含从节点带宽时也认为接收成功。

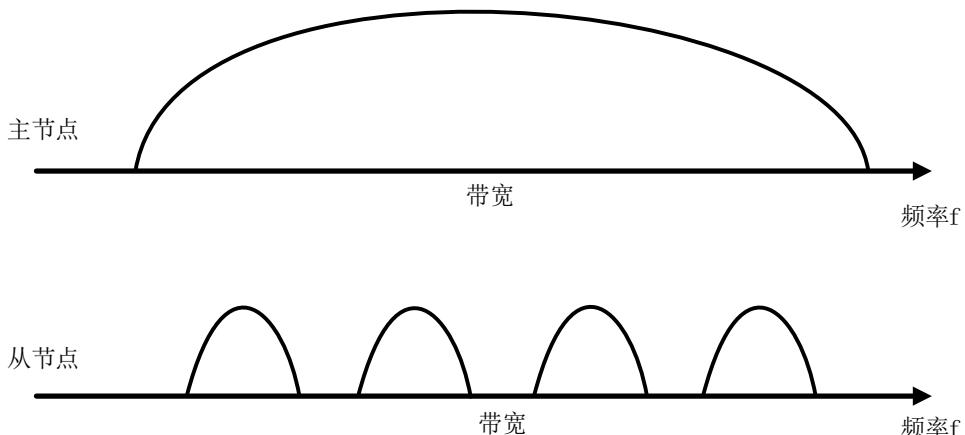


图2-77. 频分多址带宽占用示意图

仿真内核要求信道匹配阶段函数将包地址作为其唯一的输入参数。分类由内核符号常量

OPC_TDA_RA_MATCH_STATUS 表示。符号常量 OPC_TDA_RA_MATCH_VALID, OPC_TDA_RA_MATCH_NOISE, 或 OPC_TDA_RA_MATCH_IGNORE 分别用来表示有效、噪声和忽略种类。

信道匹配函数典型实现方法：

函数功能	<p>这个计算阶段针对每条可能存在的链路（不可达的链路已在前面的计算阶段里打上标记，因此已被排除在外）来执行。根据发送器和接收器的频率，带宽，数据速率，传输编码，和调制方式等五个属性来判断传输结果。根据判断结果将包标上三种标记中间的某一个：</p> <p><code>valid</code>: 接收器和发送器属性完全匹配，接收器能正确接收解码这个传输包； <code>interference</code>: 带内干扰。接收器和发送器频率，带宽等属性有重叠的部分，因此包虽然不能被解码或利用，但是这个包影响接收器接收其他的包。</p>
------	---

	ignored: 带外。也即接收器的频率等属性与发送器属性完全不一致，这个包既不能被接收器接收，也不会对接收器接收其他包产生干扰。
典型模型 典型模型 计算方法	dra_chanmatch.ps.c 读取收发器的各五个属性； tx_freq = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_FREQ); tx_bw = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_BW); tx_drate = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_DRATE); tx_code = op_td_get_db1 (pkptr, OPC_TDA_RA_TX_CODE); tx_mod = op_td_get_ptr (pkptr, OPC_TDA_RA_TX_MOD); rx_freq = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_FREQ); rx_bw = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_BW); rx_drate = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_DRATE); rx_code = op_td_get_db1 (pkptr, OPC_TDA_RA_RX_CODE); rx_mod = op_td_get_ptr (pkptr, OPC_TDA_RA_RX_MOD); 如果频带不交叉，则将包置为 ignore; (tx_freq > rx_freq + rx_bw) (tx_freq + tx_bw < rx_freq)); op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_IGNORE); 如果频带交叉，但其他属性存在不匹配情况，则将包置为 noise (interference) ; (tx_freq != rx_freq) (tx_bw != rx_bw) (tx_drate != rx_drate) (tx_code != rx_code) (tx_mod != rx_mod)); op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_NOISE); 完全匹配则置为 valid。 op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_TDA_RA_MATCH_VALID);
可选函数	dra_chanmatch、wimax_chanmatch、wlan_chanmatch、umts_ue_dra_chanmatch、 umts_utran_dra_chanmatch

阶段 4：发送天线增益

OPNET 中天线建模主要是设计天线在空间中各个角度的增益。归一化后使整个天线模型的总增益为 0。因此：全向天线即为在各个角度增益全都为 0 的一种天线；定向天线为在各个角度上增益不同的一种天线。

信号通过天线发送时，OPNET 只是从天线模型中空间一点获得增益值，存入到相应属性中。定向天线的使用参照 2.3.4 定向天线

发射天线增益是无线收发管道的第五个管道，由无线发射机的“tagain model”属性指定。其对每个目的信道分别执行，除了那些链路闭合阶段失败以及那些信道匹配阶段将传输分类为“忽略”的信道。发射天线增益阶段在信道匹配阶段返回后立即调用，期间没有仿真时间过去。发射天线增益阶段的目的是计算发射机关联的天线的增益，基于发射机到接收机向量的方向。仿真内核自己不用该结果，当其通常被分解用以阶段 7 计算接收功率。

天线增益刻画了发射信号能量放大或衰减的现象。发射功率的“整形”是基于天线结构的物理特性以及可能的在某个天线上的相位。

在任何方向上都没有增益的天线称为“全向”天线，因为其对于所有可能的信号路径都有最优的对称性。特定方向上的天线增益测量用来与全向天线做比较。它定义为给定距离的天线的信号功率与同样距离的全向天线之比。增益是无单位的量，通常表示为分贝的形式(dB)。

通过测量或者计算一系列方向上的增益，可以构建三维的天线增益模型，它描述了不同发送方向上的天线的效果。OPNET 不提供计算天线增益模型的工具，但是天线编辑器可以根据经验形式得到天线模型的数据(例如，模型必须已经知道，并且可以表示为数字形式)。另外，EMA 程序接口可以用 C 语言程序描述天线模型，允许用分析的方法形成模型，或者是数据库的检索(例如，别的面向天线的程序产生模型数据，这些数据能被 OPNET 所利用)。不考虑用来捕获天线模型数据的方法，天线编辑器还可以展示 3D 的天线模型。

通常，发射天线增益阶段首先计算方向向量分离发射机与接收机，接着利用天线模型的信息来决定发射的天线增益。该管道阶段也指明了天线的方向。方向由天线的一组属性控制，包括“pointing ref. phi”和“pointing ref. theta”，这两个属性指明了天线模型的一个特定部分在空间中指向一个选定的点。管道阶段可以通过 TDA:

OPC_TDA_RA_TX_BORESIGHT_PHI 和 OPC_TDA_RA_TX_BORESIGHT_THETA 来取得

发射天线的这些属性。另外，天线属性“target latitude”，“target longitude”以及“target altitude”属性定义了空间中天线的参考点。管道阶段如果需要的话可以直接从天线对象获得这些属性值。然而，仿真内核提供了来源于这些属性的信息，在大多数情况下对天线增益的计算是足够的；这些信息包含在 TDA “OPC_TDA_RA_TX_PHI_POINT”和“OPC_TDA_RA_TX_THETA_POINT”中，它们表示分离发射机和目标位置的向量的方向角 ϕ 和 θ 。这些角度与用来定义天线模型的 ϕ 和 θ 没有混淆。相反的，这些目标方向角是相对于笛卡儿坐标系而定义的。

仿真内核保留了一个 TDA，由符号常量 OPC_TDA_RA_TX_GAIN 表示，来存储发射天线增益阶段的结果以便后面的管道阶段使用。仿真内核要求阶段的函数将包地址作为其唯一的输入参数。

发送天线增益函数典型实现方法：

函数功能	<p>天线模型问题提出：在信道 pipeline 里之所以要考虑天线的模型，是因为天线在各个方向上对进行传输的包功率的衰减程度不一样，因此直接影响了包的接收功率，进而影响信噪比和误码数目。举例来说，对于一个有主瓣旁瓣的天线模型，在主瓣方向潜在链路上传输的包功率，比旁瓣方向潜在链路上传输的包功率要来得大。所谓的天线模型，它描述了不同方向上的天线增益值。天线模型函数基于的坐标系统是地心坐标系统，用两个角度来表示在地心三维坐标系上的一个方向向量 (x, y 平面角称为 θ，x, z 平面角称为 ϕ)，而对应与这个向量，由用户指定唯一的信号衰减 db 值与之相对应。这个函数（也即天线的数学模型）可以由用户在天线编辑器里来描绘。若天线模型为球型，则认为天线是各向同性的，也就是说，各个方向上传输的包里的记录天线增益的 TDA 都将被赋值 0 db。</p> <p>天线模型的属性：</p> <p>pattern 属性：即用户指定的天线模型；</p> <p>target 目标特征：天线指向目标的经纬度和海拔 (target latitude, longitude, altitude)；</p> <p>pointing ref. phi 和 pointing ref. theta : 参考的 θ 和 ϕ 角的值。一</p>
------	---

	般情况下，这两个角的意义是天线最大增益的方向，即主瓣方向。
典型模型	dra_tagain.ps.c
计算方法	<p>从包里读取发送天线采用的天线模型表的存储位置指针，判断如果是空表，那么表示是各向同性，于是不进行任何计算，直接在包的 TDA 里设置发送天线增益是 0 db。如果不是空表则进行下列步骤。</p> <p>读取收发器位置的六个坐标值。计算这条链路的方向向量；</p> <p>读取天线实际指向向量：phi 和 theta 值（已经由系统内核基于天线的目标经纬度，海拔属性和发送器的位置属性计算出，并写入包里）；</p> <p>读取天线的最大增益方向向量：phi 2 和 theta2 值（已经由系统内核基于天线的 pointing ref.phi 和 pointing ref.theta 属性得到，并写入包里）；</p> <p>我们要计算的是天线模型在目前链路方向向量上的增益是多少。天线模型的坐标系已经旋转到使得天线的最大增益方向对准 target 方向，因此应该将链路向量投影到旋转以后的坐标系上，才能查表获得链路方向上的衰减值。</p> <p>可以知道，坐标系旋转方式是 x, z 平面旋转 phi2 - phi 角度，x, y 平面旋转 theta 2 - theta 角度。因此链路向量投影到旋转后坐标系上的新的 phi 和新的 theta 角度可以计算出来，从而调用内核过程执行查表操作得到衰减值。</p> <p>将衰减值写进 packet 的 TDA 里。</p>
可选函数	dra_tagain、wimax_tagain、umts_dra_tagain

阶段 5：传播时延

传播时延阶段是无线收发机管道的第六个阶段，由无线发射机的“propdel model”属性表示。每个接收信道成功通过链路闭合和信道匹配阶段后调用该阶段。该阶段的调用发生在发射天线增益阶段返回后，期间没有仿真时间过去。该阶段的目的是计算包信号从无线发射机到接收机所需要的时间。内核利用这个时间在接收信道中设置一个开始接收事件。另外，传播时延值用来与发送时延结合起来计算包接收完成所需的时间（例如，最后一比特到达的时间就是包开始发送的时间加上传播时延与发送时延之和）。

仿真内核要求传播时延阶段函数将包地址作为其唯一的输入参数。计算的传播时延由内核符号常量 OPC_TDA_RA_START_PROPDEL 和 OPC_TDA_RA_END_PROPDEL 表示。指定的值应当是非负的双精度浮点值。

传播时延函数典型实现方法：

函数功能	在无线链路里，由于节点可能发生移动，在包传输过程中，传输距离可能发生变化。因此，在这个阶段里，计算两个时延结果：传输开始时的传播时延和传输结束时的传播时延。计算基于收发器之间的距离和常量电磁波传播速率来进行。
典型模型	dra_propdel.ps.c
计算方法	<p>读取开始距离和结束距离；</p> <pre>start_prop_distance = op_td_get dbl (pkptr, OPC_TDA_RA_START_DIST); end_prop_distance = op_td_get dbl (pkptr, OPC_TDA_RA_END_DIST);</pre> <p>计算两个时延，并写入包的 TDA 里。</p> <pre>start_prop_delay = start_prop_distance / PROP_VELOCITY; end_prop_delay = end_prop_distance / PROP_VELOCITY; op_td_set dbl (pkptr, OPC_TDA_RA_START_PROPDEL, start_prop_delay); op_td_set dbl (pkptr, OPC_TDA_RA_END_PROPDEL, end_prop_delay);</pre>
可选函数	dra_propdel、wimax_propdel、dpt_propdel、dbu_propdel、wlan_propdel、

	propdel_zero
--	--------------

阶段 6：接收天线增益

接收天线增益阶段是无线收发机管道的第七个阶段。它是与无线接收机而不是发射机关联的最早阶段，由接收机的“ragain model”属性描述。每个符合条件的目的信道分别执行该阶段，发生在包到达接收机时(传播时延过去之后)。

接收天线增益阶段的目的是计算接收机所关联的天线增益，基于从接收机到发射机向量的方向。仿真内核自己不应用该计算结果，但通常将其分解用于阶段 7 计算接收功率。

接收天线增益的概念与发射天线增益的概念完全相同，除了接收天线增益是与接收机所关联的天线的物理配置及实现有关的。

仿真内核保留了一个 TDA，由符号常量 OPC_TDA_RA_RX_GAIN 表示，来存储该阶段的数据结果以为后来的管道阶段使用。仿真内核要求该阶段的函数将包地址作为其唯一的输入参数。

接收天线增益函数典型实现方法：

函数功能	接收天线增益与发送天线增益的函数功能相同。在接收端增强信号功率
典型模型	dra_ragain.ps.c
计算方法	计算方法与发送天线增益类似，只是参数不一样。
可选函数	dra_ragain、wimax_ragain、umts_dra_ragain

阶段 7：接收功率

在仿真中默认情况下使用的是自由空间路径损耗模型：

$$\text{路径损耗} = \lambda^2 / (4 * \pi * 4 * \pi * d^2)$$

$$\text{接收功率} = \text{带内发射功率} * \text{发射天线增益} * \text{路径损耗} * \text{接收天线增益}.$$

计算过程中主要包含如下的函数：

$$L_p = \left(\frac{\lambda}{4\pi D} \right)^2, \quad \lambda = \frac{C}{f_c}, \quad P_i = \frac{P_{tx}(f_{max} - f_{min})}{B}, \quad P_{rx} = P_i G_{tx} L_p G_{rx}$$

其中 L_p 为路径损耗， λ 为波长， D 为距离， C 为光速， f_c 为中心载频， P_i 为带内频率， P_{tx} 为发送功率， f_{max} 为最大频率， f_{min} 为最小频率， B 为带宽， P_{rx} 为接收功率， G_{tx} 为发送天线增益， G_{rx} 为接收天线增益。

接收功率阶段是无线收发机的第八个阶段，由无线接收机的“power model”属性描述。对每个合格的目的信道分别调用该阶段；调用发生在接收天线增益返回后，期间没有仿真时间过去。该阶段的目的是计算到达数据包信号的接收功率(瓦)。

对分类为有效的包而言，接收功率值在判定接收机是否正确的获取包中的信息时是一个关键的因素。对于分类为噪声的包，通常必须估计接收功率以计算有效的和噪声的包的相对强度。

通常，计算接收功率是基于诸如这些因素如发射功率，发射机与接收机之间的距离，发射频率，发射和接收天线增益等。仿真内核要求接收功率阶段函数将包地址作为其唯一输入参数。计算的接收功率由内核符号常量“OPC_TDA_RA_RCVD_POWER”指定。指定值应当是一个非负的双精度浮点值，不允许别的值。

接收功率=带内发射功率*发射天线增益*路径损耗*接收天线增益。发送功率在发送机信道模型中设置，根据距离实时计算。路径损耗为 $32.44 + 20\log_{10}^{d(\text{km})} + 20\log_{10}^{f(\text{MHz})}$ 其中 d 为传

输距离，频率的单位以 MHz 计算。默认接收机灵敏度-95dBm。

接收功率函数典型实现方法：

函数功能	这个计算阶段缺省地支持"信号锁 (signal lock)"概念。其意义是指接收器认定先到达的包是应该接收的包，而在那个包的接收期间，置信道的信号锁为 1，表明信道已经正在被占用，其他到达的包被认为是干扰。相对的概念是"功率锁 (power lock)"概念。其意义是指接收器认定功率最大的包是应该接收的包，功率小于该包的其他到达的包被认为是干扰，而不管包到达的先后顺序。但"功率锁"概念不是缺省支持的处理方式。
典型模型	dra_power.ps.c
计算方法	<p>读取包的"match"标志（在信道匹配阶段计算得到的，包在那个阶段被标志上 valid, interference, 或 ignore 标志中间的任一个），如果 match 标志为 valid，则执行以下操作，进行进一步的划分 valid 与 noise 包：</p> <pre>op_td_get_int(pkptr, OPC_TDA_RA_MATCH_STATUS)==OPC_TDA_RA_MATCH_VALID</pre> <p>读取信道的标志号 (ID)；</p> <pre>rx_ch_objid = op_td_get_int (pkptr, OPC_TDA_RA_RX_CH_OBJID);</pre> <p>读取信道的"信号锁"标记；</p> <p>若信道已被加锁，则将包标记为 noise (写入 TDA)；若信道为空闲状态，则将信道的"信号锁"置为 1，表示信道从现在开始"忙"；而不管包的"match"标志是不是 valid，都要计算接收功率：</p> <p>读取包的发送功率；</p> <pre>tx_power = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_POWER);</pre> <p>读取发送器基准频率和带宽；</p> <pre>tx_base_freq = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_FREQ); tx_bandwidth = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_BW);</pre> <p>读取接受器 基准频率和带宽，从而可以得到收发器重叠的带宽；</p> <pre>rx_base_freq = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_FREQ); rx_bandwidth = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_BW); bind_min;band_max;</pre> <pre>in_band_tx_power = tx_power * (band_max - band_min) / tx_bandwidth;</pre> <p>读取发送天线和接收天线的增益；</p> <pre>tx_ant_gain = pow (10.0, op_td_get_dbl (pkptr, OPC_TDA_RA_TX_GAIN) / 10.0); rx_ant_gain = pow (10.0, op_td_get_dbl (pkptr, OPC_TDA_RA_RX_GAIN) / 10.0);</pre> <p>由频率计算发送波长，读取传播距离，可以利用公式计算自由空间的电磁波功率传播损耗。</p> <pre>tx_center_freq = tx_base_freq + (tx_bandwidth / 2.0); lambda = C / tx_center_freq; prop_distance = op_td_get_dbl (pkptr, OPC_TDA_RA_START_DIST); path_loss = (lambda * lambda) / (SIXTEEN_PI_SQ * prop_distance * prop_distance);</pre> <p>接收功率=发送功率*(重叠带宽/发送带宽)*发送天线增益*传播损耗*接收天线增益。计算出结果并写入到包的 TDA 里。</p>

	<pre>rcvd_power = in_band_tx_power * tx_ant_gain * path_loss * rx_ant_gain; op_td_set_dbl (pkptr, OPC_TDA_RA_RCVD_POWER, rcvd_power);</pre>
可选函数	dra_power、wimax_power、mil_power、wlan_power、zigbee_dra_power、umts_utran_dra_power、umts_ue_dra_power、umts_repeater_dra_power

阶段 8：背景噪声

计算背景噪声的函数主要包括：

$$T_{rx} = (NF - 1.0) * 290.0, \quad T_{bk} = 290.0, \quad k = 1.379E^{-23}, \quad N_b = (T_{rx} + T_{bk}) B_{rx} k,$$

$$N_a = B_{rx} (1.0E^{-26}), \quad N = N_b + N_a$$

其中 NF 为噪声指数， T_{rx} 为接收机温度， T_{bk} 为背景温度， k 为波尔兹曼常数， B_{rx} 为接收带宽， N_b 为背景噪声， N_a 为环境噪声， N 为噪声。

背景噪声阶段是无线收发机阶段的第九个阶段，由无线接收机的“bkgnoise model”属性描述。它在接收功率阶段返回后立即执行，期间没有仿真时间过去。该阶段的目的是表示所有噪声源的影响性，除了别的同时到达的传输包—由干扰噪声阶段表示。预期的数值是其余的噪声源功率之和，这些功率是在接收机处和接收信道通带处测量。典型的背景噪声源包括了从临近电子元件或者未建模型的无线电发射的热噪声或射电噪声（例如，商业无线电台，业余无线电，电视，这取决于频率）。

仿真内核自身不利用该阶段的数值结果，但它却保留一个 TDA—由符号常量 `OPC_TDA_RA_BKGNOISE` 表示—来存储数值结果并传递给随后的管道阶段。通常，在信噪比阶段，背景噪声值是后来加到别的噪声上来计算总的噪声。内核要求背景噪声阶段将包地址作为其唯一的输入参数。

背景噪声函数典型实现方法：

函数功能	<p>在 OPNET 里，将背景噪声功率建模为以下两部分组成：</p> <p>环境噪声：系统提供环境噪声（ambient noise）的功率谱密度 <code>AMB_NOISE_LEVEL</code>。因此环境噪声功率=带宽*功率谱密度 <code>AMB_NOISE_LEVEL</code>；</p> <p>累计热噪声：由有效的背景温度和有效的设备温度构成。计算公式是：</p> <p>累计热噪声功率=带宽*波尔兹曼常数*（背景温度+设备温度）；</p>
典型模型	<code>dra_bkgnoise.ps.c</code>
计算方法	<p>读取接收器的 <code>noise_figure</code>；</p> <p><code>rx_noisefig = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_NOISEFIG);</code></p> <p>假设操作温度 290 开尔文，计算得设备温度= <code>(noise_figure-1) *290.0</code></p> <p><code>rx_temp = (rx_noisefig - 1.0) * 290.0;</code></p> <p>背景温度 = 常数 <code>BKG_TEMP</code>；</p> <p><code>bkg_temp = BKG_TEMP;</code></p> <p>背景噪声功率 = 环境噪声功率+背景热噪声功率，计算出结果并写入包的 TDA 里。（在 OPNET 的背景噪声模型里，没有对接收放大器增益造成的噪声效果直接建模。）</p> <p><code>bkg_noise = (rx_temp + bkg_temp) * rx_bw * BOLTZMANN;</code></p>

	amb_noise = rx_bw * AMB_NOISE_LEVEL; op_td_set dbl (pkptr, OPC_TDA_RA_BKGNOISE, (amb_noise + bkg_noise));
可选函数	dra_bkgnoise 、 wimax_bkgnoise 、 umts_utran_dra_bkgnoise 、 umts_ue_dra_bkgnoise、 umts_repeater_dra_bkgnoise

阶段 9：干扰噪声

干扰噪声阶段是无线收发机的第十个阶段，由无线接收机的“inoise model”属性描述。对于数据包而言，有两种情况可能需要执行该阶段：（1）包是有效包并且到达目的信道同时另外一个包已经接收到了；（2）包是有效包并且已经被接收到了，当另外一个包（无论有效还是无效）到达时。很明显，大多数情况下对每个包出现第一种情形，第二种情形可能出现多次，这取决于模块中别的发射机的传输状况。注意到如果两个包都是有效的，这两个包可以共享一次调用干扰噪声阶段（调用的语法如下所示，提供两个包的地址给干扰噪声阶段以估计共同的影响）。

该阶段的目的是说明同时到达同一接收信道的发送之间的影响。仿真内核保留了一个TDA（由符号常量OPC_TDA_RA_NOISE_ACCUM表示）来存储当前从所有的干扰传输来的噪声值。由于通常不需要对有噪数据包进行链路状况估计，所以只是对有效的包（由信道匹配阶段决定）才保留这样的累加器。这样，干扰噪声阶段利用干扰包的接收功率在每一有效包中增加该累加器的值。当一个包接收完成时，内核自动地在仍旧在信道中的包的噪声累加器中减去接收功率值，累加器只反映了当前地噪声值。

内核要求干扰噪声阶段函数将两个包地址作为参数。第一个包地址代表最早到达的包，第二个表示新到的并触发干扰噪声阶段的包。

除了OPC_TDA_RA_NOISE_ACCUM这个TDA外，内核还提供了一个TDA来记录每个包所经历的碰撞次数。这个TDA由符号常量OPC_TDA_RA_NUM_COLL表示，在开发面向应用的管道阶段时提供了便利，因为当判定在最后一个管道阶段中是否接收或丢弃包时需要用到该结果。虽然干扰阶段是更新这个TDA的一个合适的位置，但内核并不应用这个TDA，因此也不需要保证其精确性。

干扰噪声函数典型实现方法：

函数功能	这个 pipeline stage 的调用发生在特定时机：当一个包正在被接收器接收且还未完成时，另一个包又到达该接受器，此时将调用这个计算阶段来计算干扰功率。当然干扰功率只对 valid 包（即被接收器认为是相匹配发送器发送的包，并且在接收功率计算阶段未被打上 noise 标记的包。在那里，如果一个 valid 包到达接收器时信道已经被锁定了，则该 valid 包仍然认为是 noise）来计算。
典型模型	dra_inoise.ps.c
计算方法	<p>读取前一个包的接收完成时间，与当前仿真时间（即后一个包的到达时间）进行比较，如果相等，则不认为这两个包产生了互相干扰，因此不计算干扰功率；如果不相等，则进行下列操作；</p> <pre>op_td_get dbl (pkptr_prev, OPC_TDA_RA_END_RX) != op_sim_time () 将两个包的冲突次数都加1； op_td_increment_int (pkptr_prev, OPC_TDA_RA_NUM_COLL, 1); op_td_increment_int (pkptr_arriv, OPC_TDA_RA_NUM_COLL, 1); 读取两个包的"match"标志和接收功率； prev_match = op_td_get_int (pkptr_prev, OPC_TDA_RA_MATCH_STATUS);</pre>

	<pre> arriv_match = op_td_get_int (pkptr_arriv, OPC_TDA_RA_MATCH_STATUS); 若后一个包的标志为"valid", 则将其噪声功率再加上前一个包的接收功率作为它的噪声功率值; prev_rcvd_power = op_td_get_dbl (pkptr_prev, OPC_TDA_RA_RCVD_POWER); op_td_increment_dbl(pkptr_arriv,OPC_TDA_RA_NOISE_ACCUM,prev_rcvd_power); 若前一个包的标志为"valid", 则将其噪声功率再加上后一个包的接收功率作为它的噪声功率值; arriv_rcvd_power = op_td_get_dbl (pkptr_arriv, OPC_TDA_RA_RCVD_POWER); op_td_increment_dbl(pkptr_prev,OPC_TDA_RA_NOISE_ACCUM,arriv_rcvd_power); </pre>
可选函数	<pre> dra_inoise 、 wimax_inoise 、 dra_inoise 、 umts_utran_dra_inoise 、 umts_ue_dra_inoise、 wlan_inoise </pre>

阶段 10: 信噪比 (snr model)

信噪比(SNR)阶段是无线收发机管道的第十一个阶段，由无线接收机的“snr model”属性描述。对于有效包，在三种情形下被执行：(1) 包到达目的信道；(2) 一个包已经接收到而另外一个(无论有效还是无效)到达时；(3) 一个包已经接收到而另外一个(无论有效还是无效)完成接收。很明显，第一种情形对每个包只出现一次，第二和第三个可能发生任意多次，取决于模型中其余的发射机传输状况。这三种类型的调用描述了包的平均 SNR 被认为是常数的时间间隔(当然，当存在移动性时这只是一个近似，因为 SNR 将会联系变化)。

SNR 阶段的目的是计算到来的数据包的 SNR 值，通常它是基于早期阶段获得的数值，包括了接收功率、背景噪声以及干扰噪声。包的 SNR 值是一个重要的性能量度，用来判定接收机是否正确的收到包的内容。内核利用该阶段计算的结果来更新接收信道的输出值，通常管道后面的阶段也用到该值。

仿真内核要求 SNR 阶段函数将包地址作为其唯一的输入参数。计算的 SNR 值由内核符号常量 OPC_TDA_RA_SNR 来表示。指定的值应当是一个双精度浮点值，以分贝(dB)表示。信噪比计算阶段主要包含如下公式： $SNR = 10 \log_{10}^{[P_r/(P_b+P_i)]}$ ，其中 P_r 为接收功率， P_b 为背景噪声， P_i 为干扰噪声。

在计算有效信噪比中，除了信号信噪比，还有处理增益 $10 \log_{10}^{(b_w/d_r)}$ ，其中 b_w 为带宽 d_r 为速率。有效 SNR=信号 SNR+处理增益。

信噪比函数典型实现方法：

函数功能	虽然背景噪声功率对于每个包的传输来说，只进行估算一次，但是干扰噪声功率却可能要计算多次。因为在一个包的整个接收过程当中，可能有多次的其他包的到达，形成了新的干扰功率，每形成一次干扰，都要重新对信噪比评估一次。一个包在两次评估信噪比的时间间隔里传输的那一段数据称为一个 segment。
典型模型	dra_snr.ps.c
计算方法	读取接收功率； <code>rcvd_power = op_td_get_dbl (pkptr, OPC_TDA_RA_RCVD_POWER);</code>

	<p>读取干扰噪声功率和背景噪声功率；</p> <pre>accum_noise = op_td_get_db1 (pkptr, OPC_TDA_RA_NOISE_ACCUM); bkg_noise = op_td_get_db1 (pkptr, OPC_TDA_RA_BKGNOISE);</pre> <p>计算信噪比，写入包的 TDA 里；</p> <pre>op_td_set_db1 (pkptr, OPC_TDA_RA_SNR, 10.0 * log10 (rcvd_power / (accum_noise + bkg_noise))); 将仿真时间也写入包里，以记录本次信噪比计算的时间点。</pre> <pre>op_td_set_db1 (pkptr, OPC_TDA_RA_SNR_CALC_TIME, op_sim_time());</pre>
可选函数	dra_snr、wimax_snr

阶段 11：误比特率

要得到 BER 涉及到两个主要参数：SNR，调制方式。

1、首先计算 SNR，有效 SNR 分为两部分：信号 SNR 和处理增益。

$$\text{信号 SNR} = 10 \log_{10} (\Pr / (\Pr + \Pi))$$

其中 \Pr 为接收功率， Π 为背景噪声， Π 内部干扰噪声。

处理增益 = $10 \log_{10} (\text{bw}/\text{dr})$ ，bw——带宽，dr——速率

有效 SNR = 信号 SNR + 处理增益。

Ber 计算是根据有效 SNR，调制方式（对应到调制方式的误码率性能曲线中）得到误码率。

误比特率(BER)阶段是无线收发信机管道的第 12 个阶段，由无线接收机的“ber model”属性描述。对有效包（由信道匹配阶段决定）在 3 种情形下可能调用该阶段：

(1) 包在目的信道完成接收；(2) 一个包已经接收到而另外一个包（无论有效还是无效）到达时；(3) 一个包已经接收到而另外一个包（无论有效还是无效）完成接收时。这几种情形对应于包的 SNR 值为常数。

BER 阶段的目的是从过去的 SNR 值为常数的阶段中得到比特错误概率。这不是根据经验的比特错误率，而是基于 SNR 的预期的值。通常，该阶段的比特错误率也是用于发送信号的调制类型的函数。

仿真内核要求 BER 阶段函数将包地址作为其唯一的输入参数。计算出的 BER 由内核符号常量 OPC_TDA_RA_BER 表示。指定的值应当是一个介于 0 和 1（包括 1）之间的双精度浮点值。

误比特率函数典型实现方法：

函数功能	<p>在无线链路里，误码率计算不是基于整个包来计算的，而是基于每一个 segment 段来计算的。因为在这个包的传输过程中，信噪比不是固定不变的，从而导致误码率也不是固定不变的。计算方法是根据信噪比和调制函数来计算误码率，在调制函数和信噪比已知的情况下，误码率可以被唯一确定。调制函数可以在 modulation curve editor 里来描述。</p>
典型模型	dra_ber.ps.c
计算方法	<p>读取信噪比和接收器的处理增益；</p> <pre>snr = op_td_get_db1 (pkptr, OPC_TDA_RA_SNR); proc_gain = op_td_get_db1 (pkptr, OPC_TDA_RA_PROC_GAIN);</pre> <p>二者相加得到有效的信噪比；</p> <pre>eff_snr = snr + proc_gain;</pre> <p>读取调制函数表的地址指针；</p>

	<pre>modulation_table = op_td_get_ptr (pkptr, OPC_TDA_RA_RX_MOD); 调用系统内核过程 op_tbl_mod_ber() 来计算出误码率 ber，并写入包的 TDA 里。 ber = op_tbl_mod_ber (modulation_table, eff_snr); op_td_set dbl (pkptr, OPC_TDA_RA_BER, ber);</pre>
可选函数	dra_ber、wimax_ber、umts_utran_dra_ber、umts_ue_dra_ber、wlan_ber

阶段 12：误码数目

错误分布阶段的计算函数包含：

$$P_k = p^k (1-p)^{N-k} \binom{N}{k}, \quad \sum_{k=0}^N P_k \geq r, \text{ 其中 } P_k \text{ 为 } k \text{ 个错误的概率, } p \text{ 为一个码元错误的}$$

概率, N 为数据包长度, r 为随机产生一个 $0 \sim 1$ 之间的数, k 为误码数量。

错误分布阶段是无线收发信机的第十三个阶段, 由无线接收机的“error model”属性描述。通常在比特错误率阶段返回后立即执行。错误分布阶段的目的是估计数据包中的一段的比特错误数目, 这一段的比特错误概率已经计算并且为常数。如果比特错误概率在包的接收过程中没有变化, 那么段可以是整个包。比特错误数目的估计通常基于比特错误概率(从阶段 11 中获得)和段的长度。

仿真内核要求错误分布阶段函数将包地址作为其唯一的输入参数。内核只在包中保留一个比特错误累加器 TDA; 因此, 管道阶段将新发生的错误数目加到整数 TDA 的已存在的数值中, 用符号常量 OPC_TDA_RA_NUM_ERRORS 表示。增加的数值应当介于 0 和受影响的段的长度之间。另外, 内核要求错误分布阶段在数据包段上提供经验式的比特错误率; 这个错误率可以通过将段中的比特错误数除以段的大小获得。经验的比特错误率应当置于一个双精度的 TDA 中, 它由符号常量 OPC_TDA_RA_ACTUAL_BER 表示。内核利用这个数值来更新接收信道的比特错误率。

Ber 计算式根据有效 SNR, 调制方式(对应到调制方式的误码率曲线中)得到误码率。

误码数目函数典型实现方法:

函数功能	获取数据包的误码数目
典型模型	dra_error.ps.c
计算方法	已知每一段(segment)的误码率, 即可计算出每一段的误码数目, 计算方法和其他两种链路的一样。然后将其累积起来即可得到总的误码数目, 并写进包的 TDA 里
可选函数	dra_error、wimax_error、umts_utran_dra_error、umts_ue_dra_error、dpt_error、dbu_error、wlan_error、error_zero_err

阶段 13：错误纠正

计算一个包长内错误位数与包长的比, 如果超过门限则接收失败, 否则包接收成功。判断: num_errs / pklen 与 ecc_thresh 的大小。ecc_thresh 在无线接收机的属性中设置, 如下图所示。

Attribute	Value
r-name	receive
channel	(...)
modulation	bpsk
noise figure	1.0
ecc threshold	0.0
regain model	dra_regain
power model	dra_power
bkgnoise model	dra_bkgnoise
inoise model	dra_inoise
snr model	dra_snr
ber model	dra_ber
error model	dra_error
ecc model	dra_ecc
icon name	ra_rx

图2-78. Ecc门限设置

错误纠正阶段是管道的第十四个阶段，由无线接收机的“ecc model”表示。当包完成接收，在错误分布阶段返回后立即被触发执行，期间没有仿真时间过去。对每个被认为是有用的数据包(由信道匹配阶段判定)只调用该阶段一次。该阶段的目的是判定是否接收到达的包并通过信道对应的输出流转发到接收机相邻的模块中，这通常取决于是否包经历了碰撞、在错误分布阶段所计算的结果以及接收机纠正错误的能力。依据该阶段所作出的判断，内核或者销毁该包，或者允许其继续发送到目的阶段。另外，它也影响接收信道收集的错误和吞吐量结果。

仿真内核要求错误纠正阶段函数将包地址作为其唯一的输入参数。判断接收或丢弃包由符号常量 OPC_TDA_RA_PK_ACCEPT 表示。所指定的值应当是一个等于常量 OPCTURE 的整数，表示接收；否则应当指定整数值 OPC_FALSE，表示丢弃。

误码纠正函数典型实现方法：

函数功能	包的可接受标准有两条：一个是因为源端的问题导致包没有完整发送；另一个是比较误码数目和接收器的纠错门限值。
典型模型	dra_ecc.ps.c
计算方法	<pre> 读取节点使能标志，如果节点被 disabled，则置包为不可接受； if (op_td_is_set (pkptr, OPC_TDA_RA_ND_FAIL)) accept = OPC_FALSE; 否则就读取误码数目和纠错门限，比较二者的值，以判断是否能接受该包； ecc_thresh = op_td_get_dbl (pkptr, OPC_TDA_RA_ECC_THRESH); num_errs = op_td_get_int (pkptr, OPC_TDA_RA_NUM_ERRORS); accept = (((double) num_errs) / pklen) <= ecc_thresh) ? OPC_TRUE : OPC_FALSE; 将判断结果写入包的 TDA 里。 op_td_set_int (pkptr, OPC_TDA_RA_PK_ACCEPT, accept); 无论如何接收的链路都不能被锁住 rxch_state_ptr = (DraT_Rxch_State_Info *) op_ima_obj_state_get (op_td_get_int (pkptr, OPC_TDA_RA_RX_CH_OBJID)); rxch_state_ptr->signal_lock = OPC_FALSE; </pre>
可选函数	dra_ecc、wimax_ecc、umts_dra_ecc、dpt_ecc、dbu_ecc、mil_ecc、wlan_ecc、ecc_zero_err、umts_repeater_dra_ecc

2.7、拓扑

在进行网络仿真我们总是要最先确定网络拓扑结构，明确网络属于总线网 Bus、网状态 Mesh、环状网 Ring、星状网 Star、树型网 Tree 的哪一种。OPNET 给出了一种快速拓扑配置的方法。

从 Topology 菜单中选择 Rapid Configuration，进入拓扑结构选择界面，如图 2-78 所示。

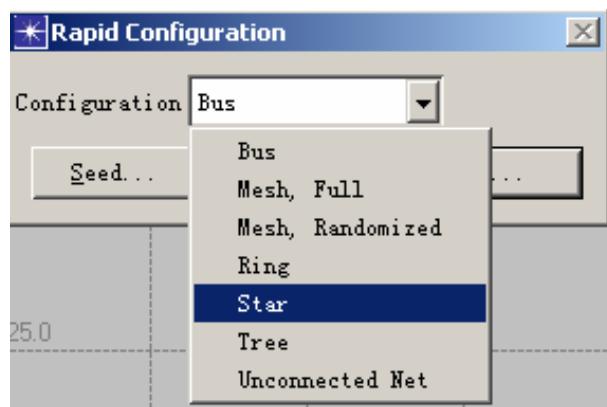


图2-79. 快速拓扑配置窗口

从配置下拉列表中选择自己网络的拓扑结构，这里选择 Star，单击 OK，出现如图 2-79 所示的配置界面。这里进入的是星状网的配置界面

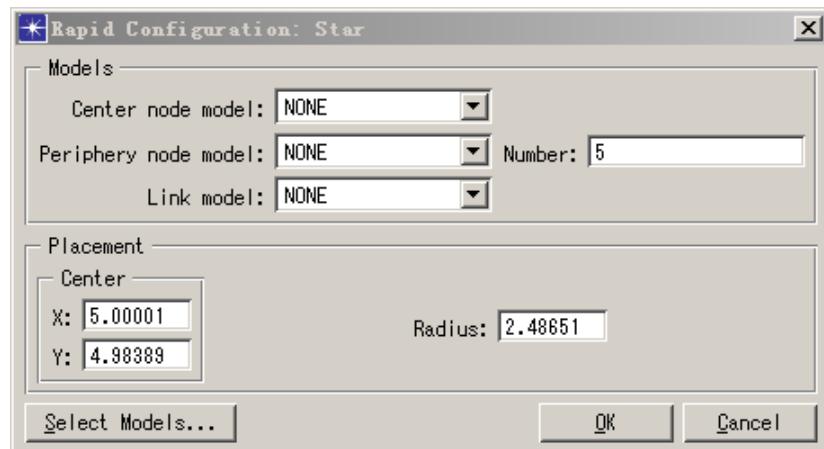


图2-80. 星状网拓扑快速配置

在配置界面可以设置节点模型，链路模型，网络位置，范围，节点数目等。按照自己的需求点击 OK，子网场景中会自动创建好网络拓扑。如图 2-80 所示。

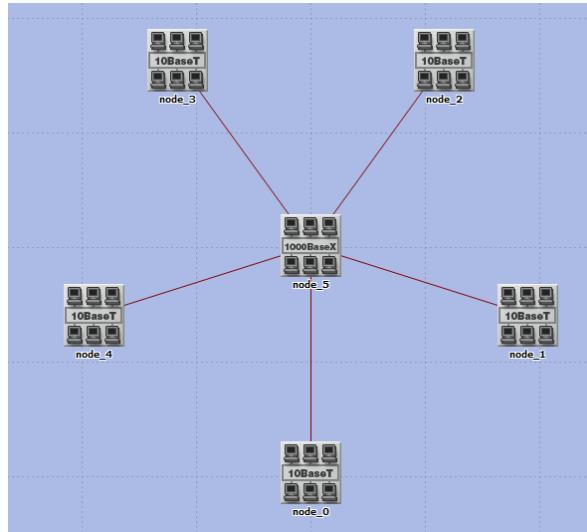


图2-81. 星状网络拓扑示意图

这样一个子网的拓扑就配置好了。如果网络不复杂，我们也可以通过手动拖动设置。

示例：参考实例 7

第三章 代码实现

3.1、变量

OPNET 支持 C/C++ 编译，所以支持 C/C++ 语言的变量类型，不过需要在进程模型编译前，点击进程模型界面 Compile 菜单，选择 Enable C++ Code Generation。OPNET 本身的语言被称为 Proto-C。除了 OPNET 自带的变量类型和 C/C++ 的类型，在不同的协议中，OPNET 也使用了很多的自定义结构体类型，用户也可以根据自己的需要定义结构体类型。所以除了下表中的比较常规的变量类型，OPNET 还可以使用很多的变量类型。尤其需要注意的是 bool 属于 C/C++ 变量类型，如果定义了 bool 类型的变量需要开启 C++ 编译模式，Proto-C 中的布尔型为 Boolean。

表3-1. OPNET 常用变量类型

变量类型	类型名称	变量类型	类型名称
int	整型	OpT_Packet_Size	数据包大小
Boolean	布尔型	OpT_uInt32	无符号 32 位整型
Compcode	C 代码	OpT_uInt64	无符号 64 位整型
double	双浮点型	Prohandle	进程句柄
Distribution	分布	Packet*	包指针
float	浮标行	Stathandle	统计量句柄
char*	字符型指针	List*	列表指针
Const char*	字符串	Boolean	布尔型
Evhandle	事件句柄	OpT_Packet_Id	包 ID
Objid	对象 ID	Log_Handle	记录句柄
OpT_Int32	32 位整型	Ici*	ICI 指针
OpT_Int64	64 位整型	OpT_Int64*	64 位指针

(1) Animation Entity: 动画集由操作中特定动画实体的 ID 号表示。之所以采用 ID 号来代替中的指针，是因为对于动画观察函数 op_vuanim，ID 号通信超过了仿真范围。尽管 ID 号只是存储在规则的 C/C++ 整型变量中的简单整数值，但 OPNET 也声明了特定的数据类型来准确标记 ID 参数和变量。三种基于 ID 号的动画实体包括浏览器 (Viewer)、宏 (Macro) 和图画 (Drawing)。

(2) Boolean: 核心函数通过返回布尔值来表示结果是否正确。布尔值可与符号常量 OPC_TRUE 和 OPC_FALSE 进行比较。

(3) Compcode: 核心函数通过返回 Compcode 值来表示操作是否正确完成。Compcode 的值可与符号常量 OPC_COMPCOED_SUCCESS 和 OPC_COMPCOED_FAILURE 进行比较。

(4) Distribution: Distribution 是一种与概率密度函数 (PDF) 一致的数据结构，它描述了随机数到特定数字输出的映射。Distribution 包含一张对映射进行编码的数字表，指出完成该映射的算法。对于基于表格的 Distribution，数据从 PDF 编辑器的 PDF 模型文件中读入。这些结构均由 Dist 函数集中核心函数操作。

(5) Event Handle: 事件句柄是惟一一种确定未决仿真事件 (中断) 的数据结构。该结构主要在 Intrpt 核心函数集中使用，因此可通过它们处理预设的中断。注意，事件句柄是一种数据结构，而不是整型或指针。因此不能把它存储在整型或指针变量中。

(6) Statistic Handle: 统计量句柄是一种确定动态产生的全局和局部统计量的数据结构。统计量句柄的数据类型为 Stathandle, 获得统计量句柄的唯一方法是通过核心函数的 Stat 函数集来注册统计量。注册统计量时将为其指定一个唯一的名称，并和时间一起存储在一个输出矢量中。局部统计量用在特定处理器或队列中；全局统计量由仿真模型中的实体共享，每个实体分布式地作用于输出矢量。

(7) ICI: ICI (Interface Control Information, 接口控制信息) 是与仿真中断相关的结构化数据的集合用于进程间通信机制，传输分层协议接口的控制信息。ICI 由 ici 函数集中的核心函数操作。

(8) List: List 是存储在双向链表中的数据元素的集合。List 中的元素可按照从简单的 C/C++ 数据类型到复杂的数据结构进行排列。List 主要用于临时存储数据结构组，可包含各种不同类型的元素，但通常并不这样使用。对 List 的大小没有限制，可在其任意位置插入或移除元素。List 由 Prg 函数集的 List 子函数集操作。

(9) Object ID: 对象 ID 唯一地确定了一个仿真对象。通过使用 Objid 数据类型声明该标识符，供 Id、Ima、Topo 和 Pk 函数集使用。

(10) Packet: Packet 是数据封装和传输建模中的基本仿真实体。它由 Pk 函数集中的核心函数操作。

(11) Memory Object Type: 某些建模需要为其动态分配内存来存储各种信息。每个相同大小数据组成的集合记为一个池，内核为每个池分配大量的数据条目以提高标准内存分配器的效率。每个汇聚池中的内存对象必须通过调用核心函数 op_prg_pmo_define() 来创建，该函数将返回一个汇聚内存对象句柄来标识池，用 Pmohandle 表示。创建汇聚内存对象时都为其分配了一个唯一的名称，仿真模型中的实体可以共享汇聚内存对象。

(12) Log Handle: 当在仿真调试或数据分析中创建仿真日志时，日志句柄对于每个日志项非常必要。

(13) Procedure: 某些核心函数将 C/C++ 函数指针作为参数，但并不声明这些参数作为指向返回整型值的函数的指针，而是定义了一种特殊的数据类型——Procedure。

(14) Process Handle: 进程句柄是唯一一种标识仿真中活动进程的数据结构，由 Pro 函数集中的核心函数使用。需注意进程句柄是数据结构，而不是整形或指针，因而不能将它们存储在整型或指针变量中。

(15) Sar Buffer Handle: Sar 缓冲句柄是唯一一种标识 Sar (Segmentation & Reassembly, 分段与重组) 缓冲区的数据结构。Sar 缓冲区缓存包序列，并可对包进行分段和重装。Sar 缓冲区由 Sar 函数集中的核心函数创建，该函数返回访问新缓冲区的 Sar 缓冲句柄。Sar 函数集函数利用 Sar 缓冲名柄来处理被标识的 Sar 缓冲区。与其他 OPNET 数据结构一样，不能将 Sar 缓冲句柄分配到整型或指针变量中。

3.1.1、变量区别

OPNET 中有三种变量全局变量，状态变量和临时变量。打开进程模型，在进程模型编辑器界面的工具栏，可以看到图 。SV 用来状态变量定义，TV 为临时变量定义，HB 为头模块，用来全局变量定义、结构体定义、引用、宏定义、函数声明等，FB 为自定义函数块，DB 为诊断模块，TB 为终止模块。

(1.1) 全局变量：

全局变量生存期最长，作用范围最大，它在仿真的任何时刻，在仿真系统的任何进程都是可见的，因而它经常被用来定义各个进程想要共享公用的一些变量。全局变量是在进程的头模块中定义的，它在某一个进程里被主声明，在其它需要调用它的进程中用“extern”进

行外部声明，extern 可省略不写。两个处理器模块无论是因为使用了相同的进程模型，还是因为进行了外部声明而可以同时操控同一个变量，只要全局变量的值被修改了，这个全局变量在所有进程模型中的值都被修改了。所以全局变量只适用于读取，需要在函数中修改了，尽量放在状态变量或者节点属性中实现。

由于进程中外部声明变量时可以省略 extern，所以 OPNET 进行全局变量定义时遵守如下规则。当一个进程模型头模块中，只定义了全局变量，没有进行初始化时，OPNET 认为此进程或者是定义全局变量或者是外部声明；当一个进程模型头模块中，定义并初始化了一个全局变量，OPNET 认为此进程一定定义了一个全局变量。当有两个进程模型同时定义了一个相同的全局变量时，OPNET 会报重定义错误。当只有一个定义了全局变量时，其他进程模型中被认为时外部声明。

(1.2) 状态变量：

状态变量是专属于该进程的，只要该进程被调用它就存在，但是别的进程不能直接访问它，当然通过调用一些函数它还是能够被获知的，它在状态变量中被定义。节点的一些统计变量一般采用状态变量。一般情况下，进程模型中进行编程操作时均采用状态变量。

(1.3) 临时变量：

临时变量生成期最短，它不需要在进程的两次调用之间保持不变，比如 for 循环中自加变量 i，因为只是使用上的需要，并不对它运行的结果关注所以用了临时变量。所以临时变量也往往可以在状态的出入指令中现场定义。当进程进入到阻塞态状态，挂起时，临时变量就会被恢复初始值。

3.1.2、变量定义

(2.1) 状态变量：

在进程编辑器中，点击 SV，进入状态变量定义窗口，如图 3-1 所示。

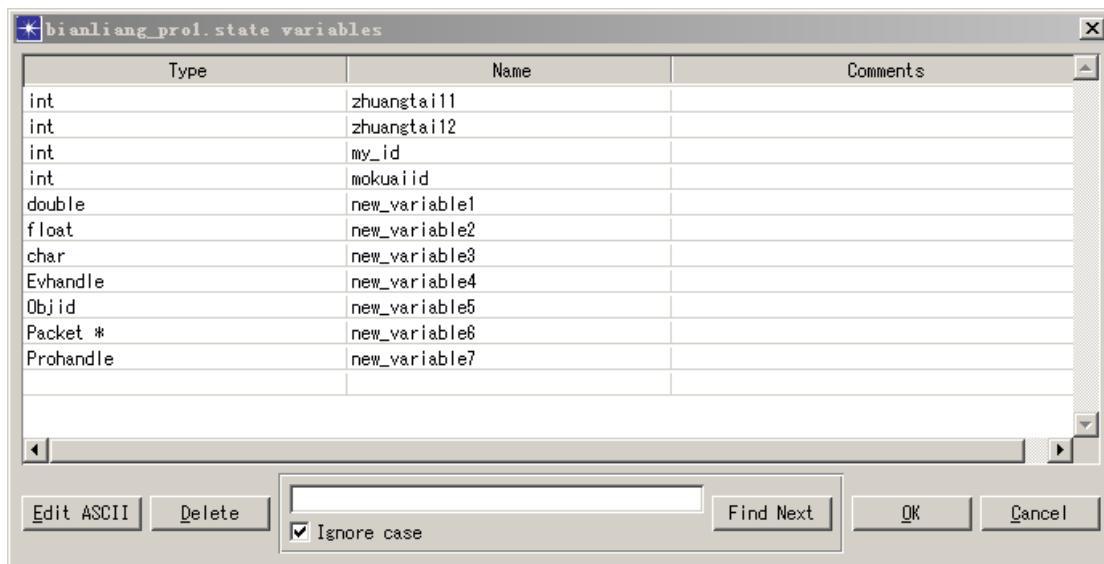


图3-1. OPNET 状态变量定义窗口

在 Type 栏选择变量类型，在 Name 栏输入变量名，也可以点击 Edit ASCII 进入变量输入源码边界界面。

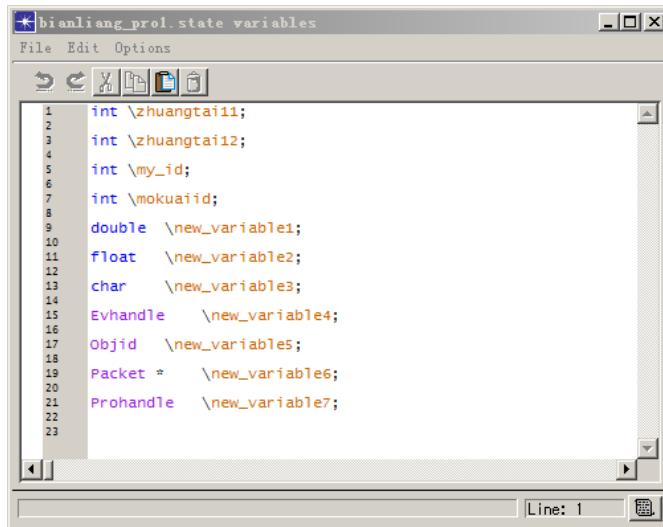


图3-2. OPNET 状态变量 ASCII 定义窗口

在这个界面我们可以看到状态变量的定义使用（变量类型 \变量名;）的方式，且状态变量无法初始化。这个界面在多个变量的复制时非常有用，所以也就成为我们主要使用的定义方式。

(2.2) 临时变量：

在进程编辑器中，点击 TV，进入临时变量定义窗口，如图 3-3 所示。

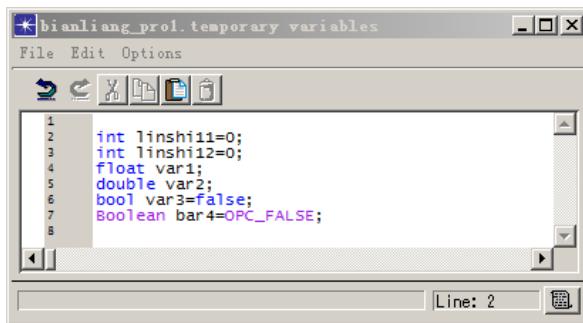


图3-3. OPNET 临时变量定义窗口

临时变量由于作用范围有限，当遇到进程阻塞时，就会恢复初始值，所以临时变量也可以在需要的状态代码中进行定义。临时变量定义时可以直接进行初始化，当没有进行初始化而直接使用时，以默认初始值执行。

(2.3) 全局变量：

全局变量在头模块中定义，也可以讲所需要的变量定义在.h 文件中，在需要引用变量的进程模型中引用此.h 文件。

3.1.3、随机变量分布

随机变量跟仿真时间和随机种子有关，仿真时间和随机种子不变，随机变量不变。因此注意在设置根据随机变量控制状态转移时间时，注意如果状态总是根据变量值跳转到本身时，如果某次变量值为 0，则立即重新跳转到状态本身，重新生成随机数，随机数会保持不变，因为仿真时间没有改变。因此我们可以采用在跳转前判断随机数的值是否为 0，或先生成一个随机分布序列，每次从随机序列中取出值再使用。

OPNET 自带变量分布类型包括三种定义方式, 概率群分布函数(PMF), 概率密度函数(PDF)和累积分布函数(CDF)。其主要包含如下的变量分布类型。

表3-2. OPNET 自带变量分布函数

贝努力概率群分布函数: $p(x) = \begin{cases} (1-p), & x=0 \\ p, & x=1 \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = p$ 方差: $\delta^2 = p(1-p)$	二项式分布概率群分布函数: $f(x) = \binom{n}{x} p^x (1-p)^{n-x}$ 均值: $E(x) = np$ 方差: $\delta^2 = np(1-p)$
卡方分布概率密度函数: $f(x) = \begin{cases} \left[\left(\frac{n}{2} - 1 \right)! \right]^{-1} \left(\frac{x}{2} \right)^{\frac{n}{2}} e^{-\frac{x}{2}}, & x > 0 \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = n$ 方差: $\delta^2 = 2n$	平稳概率群分布函数: $p(x) = \begin{cases} 1, & x = C \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = C$ 方差: $\delta^2 = 0$
尔兰概率密度分布函数: $f(x) = \begin{cases} \frac{(\frac{x}{b})^{c-1} e^{-\frac{x}{b}}}{b(c-1)!}, & x > 0 \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = bc$ 方差: $\delta^2 = cb^2$	指数分布概率密度函数: $f(x) = \begin{cases} ae^{-ax}, & x > 0 \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = a^{-1}$ 方差: $\delta^2 = a^{-2}$
伽马分布概率密度函数: $f(x) = \left(\frac{x}{b} \right)^c \times \frac{e^{-\frac{x}{b}}}{b\Gamma(c)}$ 均值: $E(x) = bc$ 方差: $\delta^2 = b^2 c$	几何分布概率群分布函数: $f(x) = p(1-p)^x$ 均值: $E(x) = \frac{1-p}{p}$ 方差: $\delta^2 = \frac{1-p}{p^2}$
拉普拉斯概率密度函数: $f(x) = \frac{1}{2b} e^{-\frac{ x-a }{b}}$ 均值: $E(x) = a$ 方差: $\delta^2 = 2b^2$	增长分布概率密度函数 $f(x) = \frac{e^{\frac{x-a}{b}}}{b(1+e^{\frac{x-a}{b}})^2}$ 均值: $E(x) = a$ 方差: $\delta^2 = \frac{\pi^2 b^2}{3}$
对数正态概率密度函数: $f(x) = \begin{cases} \frac{1}{x\delta\sqrt{2\pi}} e^{\frac{-(\ln x - \mu)}{2\delta^2}}, & x > 0 \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = e^{\mu + \frac{b^2}{2}}$ 方差: $\delta^2 = (e^{\mu + b^2/2})(e^{b^2/2} - 1)$	正态分布概率密度函数: $f(x) = \frac{1}{\delta\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\delta^2}}$ 均值: $E(x) = \mu$ 方差: $\delta^2 = \sigma^2$

帕累托概率密度函数: $f(x) = \frac{ca^c}{x^{c+1}}, \quad a \leq x, c > 2$ 均值: $E(x) = \frac{ca^2}{c-1}$ 方差: $\delta^2 = \frac{ca^2}{(c-1)^2(c-2)}$	泊松概率密度函数: $f(x) = \frac{a^x}{e^a x!}, \quad a > 0, x \text{ 为非负整数}$ 均值: $E(x) = a$ 方差: $\delta^2 = a$
幂函数概率密度函数: $p(x) = \frac{cx^{c-1}}{b^c}, \quad 0 \leq x \leq b$ 均值: $E(x) = \frac{bc}{c+1}$ 方差: $\delta^2 = \frac{b^2 c}{(c+2)(c+1)^2}$	瑞利概率密度函数: $p(x) = \frac{x}{b^2} e^{-\frac{x}{4b^2}}, \quad 0 \leq x$ 均值: $E(x) = b\sqrt{\frac{\pi}{2}}$ 方差: $\delta^2 = (2 - \frac{\pi}{2})b^2$
三角分布概率密度函数: $p(x) = \begin{cases} \frac{(x-a)}{(b-a)^2}, & a \leq x \leq \frac{a+b}{2} \\ \frac{(b-x)}{(b-a)^2}, & \frac{a+b}{2} \leq x \leq b \end{cases}$ 均值: $E(x) = \frac{a+b}{2}$ 方差: $\delta^2 = \frac{(a-b)^2}{24}$	均匀分布概率密度函数: $p(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = \frac{a+b}{2}$ 方差: $\delta^2 = \frac{(a-b)^2}{12}$
整数均匀概率群分布函数: $p(x) = \begin{cases} \frac{1}{(1+b-a)}, & a \leq x \leq b \\ 0, & \text{else} \end{cases}$ 均值: $E(x) = \frac{a+b}{2}$ 方差: $\delta^2 = \frac{(a-b)(2+b-a)}{12}$	威布尔累积分布函数: $p(x) = \begin{cases} 1 - e^{-\left(\frac{x}{\beta}\right)^\alpha}, & x > 0 \\ 0, & \text{else} \end{cases}$

其中各分布函数在 OPNET 中的使用如表 3-3 所示，每中分布对应不同的文件，使用时传入不同的参数。

表3-3. OPNET 分布函数对应文件及参数

分布名称	Pdf 文件	参数 1	参数 2
贝努里分布	bernoulli.pd.s	均值 p	
二项式分布	binomial	取样次数 n>0 正整数	成功接收概率 p=[0,1]
卡方分布	chi_square	均值 n>0 正整数	
平稳分布	constant	均值 C	
尔兰分布	erlang	比例 b	形状 c
指数分布	exponential	均值 1/a	
极限分布	extreme	位置 a	比例 b
伽马分布	gamma	比例 b	形状 c

几何分布	geometric	成功接收概率 p	
拉普拉斯	laplace	均值 a	比例 b
增长	logistic	均值 a	比例 b
对数正态	lognormal	均值	方差
正态分布	normal	均值 u	方差
帕累托分布	pareto	位置 a	形状 c
泊松分布	poisson	均值 a	
幂函数分布	power	形状 c	比例 b
瑞利分布	rayleigh	比例 b	
三角分布	triangular	位置下限 a	位置上限 b
均匀分布	uniform	取值下限 a	取值上限 b
整数均匀分布	uniform_int	取值下限 a	取值上限 b
威布尔分布	weibull	形状 a	比例 b

常用函数如表 3-4 所示：

表3-4. OPNET 变量分布常用函数

double op_dist_uniform(double limit); 产生 0-limit 之间的一个随机数，包含 0，不包含 limit
double op_dist_exponential(double mean); 返回一个指数分布的随机数，返回值在 0 到正无穷 例：Rand_num = op_dist_exponential(0.5);
Distribution* op_dist_load (const char* dist_name, double dist_arg0, double dist_arg1); 加载一个概率密度函数文件，生成一个随机分布序列。当参数不需要时会自动忽略，发生错误返回 OPC_NIL 例：next_dist = op_dist_load("exponential", ia_time, 0.0); 例：job_type_dist = op_dist_load("uniform_int", 1, job_type_range);
op_dist_unload (Distribution* dist_ptr); 释放一个分布指针的内存，如果出错的话，返回 OPC_DBL_INVALID
double op_dist_outcome(Distribution* dist_ptr); 从随机分布序列中取出一个值

3.2、结构体与列表

3.2.1、结构体

结构体对信息整合具有很好的效果，所以在 OPNET 仿真中使用普遍。同 C 语言一样，结构体的类型定义使用 `typedef struct`。结构体在头模块中定义，不同进程模块或相同进程模块的头模块可以定义相同的结构体。在 OPNET 自带协议编程中，出现很多的变量类型，就是使用的结构体进行的类型定义。

表3-5. OPNET 结构体使用方法

功能描述	代码操作	备注
结构体定义	<pre>typedef struct struct_information { Packet* send_frame; Int receive_mac_address; Boolean have_send; Evhandle get_ack_timeout; } struct_information;</pre>	结构体在头模块中定义，定义后，可在状态变量中用结构体定义的类型来定义变量。
变量定义	struct_information * var_struct;	无法对结构体进行初始化
结构体变量 开辟内存	var_struct = (struct_information *)op_prg_mem_alloc(sizeof(struct_information));	指针必须先开辟内存才能使用
结构体变量 参数赋值	var_struct -> receive_mac_address = 1;	在不进行复制时，使用默认初始化值
结构体变量 清空内存	op_prg_mem_free(var_struct);	内存清空后不再能查询到变量内部数据。

3.2.2、列表

列表能灵活的管理大量的同类数据，在OPNET仿真中使用也很普遍。列表可以理解为一个动态长度的数组，里面只能存储指针类型的变量。使用列表为数据包排队，能使处理器模块具有队列模块一样的排队效果。

列表定义	List* mlist;
列表初始化	mlist = op_prg_list_create();

列表的定义和初始化都比较简单，下面重点讲述一下列表的使用。

表3-6. OPNET列表常用函数

op_prg_list_insert:

函数体	void op_prg_list_insert (List* list_ptr, void* element_ptr, int pos_index);
功能	在某列表的指定位置插入指针，pos_index 从 0 开始为列表头，pos_index 还可以是 OPC_LISTPOS_HEAD 列表头或 OPC_LISTPOS_TAIL 列表尾
示例	op_prg_list_insert (list_ptr, i_ptr, pos_index); //在 list_ptr 列表的 pos_index 位置插入 i_ptr 指针，插入后 i_ptr 指针在列表中的下边为 pos_index。

op_prg_list_remove:

函数体	void* op_prg_list_remove (List* list_ptr, int pos_index);
功能	移除指定列表中指定位置的指针并返回，pos_index 从 0 开始为列表头，pos_index 还可以是 OPC_LISTPOS_HEAD 列表头或 OPC_LISTPOS_TAIL 列表尾。移除指针，不清除指针内存。
示例	mypacket = op_prg_list_remove (list_ptr, pos_index); //将 list_ptr 列表下标为 pos_index 位置上的指针移除赋值给 mypacket

op_prg_list_access:

函数体	void* op_prg_list_access (List* list_ptr, int pos_index);
-----	---

功能	读取指定列表中指定位置的指针并返回, pos_index 从 0 开始为列表头, pos_index 还可以是 OPC_LISTPOS_HEAD 列表头或 OPC_LISTPOS_TAIL 列表尾。读取列表, 仅复制指针, 不复制内存。
示例	mypacket = op_prg_list_access (list_ptr, pos_index); //读取 list_ptr 列表下标为 pos_index 位置上的指针, 将指针赋值给 mypacket

op_prg_list_size:

函数体	int op_prg_list_size (List* list_ptr);
功能	获取列表中元素的个数
示例	Mypacket_num = op_prg_list_size (list_ptr); //返回 list_ptr 列表中元素的个数给 mypacket_num。

op_prg_list_free:

函数体	void op_prg_list_free (List* list_ptr);
功能	清空列表中的指针, 同时清空指针指向的内存。
示例	op_prg_list_free(list_ptr); //清空 list_ptr 列表, 清空队列内存。

op_prg_list_clear:

函数体	void op_prg_list_clear (List* list_ptr);
功能	清空列表中的指针, 但不清空指针指向的内存。
示例	op_prg_list_clear(list_ptr); //清空 list_ptr 列表。

op_prg_list_elems_copy:

函数体	void op_prg_list_elems_copy (List* from_list_ptr, List* to_list_ptr);
功能	复制列表, 仅复制元素指针, 不复制元素指针指向的内存。
示例	op_prg_list_elems_copy (list_ptr1, list_ptr2); //复制列表 list_ptr1 到列表 list_ptr2。

op_prg_list_elems_find:

函数体	void* op_prg_list_elem_find (List* list_ptr, List_Compare_Proc test_func, void* elem_ptr, int* low_bound_pos_ptr, int* high_bound_pos_ptr);
功能	在列表中查询元素并返回。
示例	found_info = op_prg_list_elem_find (redirect_lptr, redirect_list_compare_proc, redirect_info, &low_index, &high_index); // 在 redirect_lptr 列表的 *low_index 到 * high_index 范围内查找 redirect_list_compare_proc 并范围指针给 redirect_info。

op_prg_list_map:

函数体	void op_prg_list_map(List* list_ptr, List_Map_Proc mapfun, void* state);
功能	为指定列表中的每个元素应用用户自定义的函数运算。
示例	op_prg_list_map (routing_list_ptr, printf, "%s\n"); //将 routing_list_ptr 列表中的每个元素打印输出

3.3、常量

3.3.1、对象常量

表3-7. OPNET 对象常量及描述

常量名称	值	英文描述	描述
OPC_OBJTYPE_ANT	23	antenna object (requires Wireless module)	定向天线
OPC_OBJTYPE_BURX	20	bus link receiver object	总线链路接收机
OPC_OBJTYPE_BURXCH	29	bus link receiver channel object (comp. attr. subobject)	总线链路接收信道
OPC_OBJTYPE_BUTX	19	bus link transmitter object	总线链路发送机
OPC_OBJTYPE_BUTXCH	30	bus link transmitter channel object (comp. attr. subobject)	总线链路发送信道
OPC_OBJTYPE_COMP	44	compound attribute object	复合属性
OPC_OBJTYPE_DEMAND_CONN	11	connection object	连接
OPC_OBJTYPE_DEMAND_FLOW	12	flow object	流量
OPC_OBJTYPE_ESINTERFACE	47	esys interface	外部系统接口
OPC_OBJTYPE_ESYS	46	esys module object	外部系统模块
OPC_OBJTYPE_GENERIC	45	subobject of a user-defined compound attribute object	子对象
OPC_OBJTYPE_LKBUS	8	bus link object	总线链路
OPC_OBJTYPE_LKDUP	7	duplex point-to-point link object	双工点对点链路
OPC_OBJTYPE_LKSIMP	6	simplex point-to-point link object	单工点对点链路
OPC_OBJTYPE_LKTAP	9	tap object (attaches to bus link)	抽头
OPC_OBJTYPE_NODE_FIX	3	fixed node object	固定节点
OPC_OBJTYPE_NODE_MOB	4	mobile node object (requires Wireless module)	移动节点
OPC_OBJTYPE_NODE_SAT	5	satellite node object (requires Wireless module)	卫星节点
OPC_OBJTYPE_PATH	10	path object	路径
OPC_OBJTYPE_PB_STAN		statistic animation probe object	统计量动画探针
OPC_OBJTYPE_PBAA	38	automatic animation probe object	自动动画探针
OPC_OBJTYPE_PBCA	39	custom animation probe object	自定义动画探针
OPC_OBJTYPE_PBCP_NODE	37	coupled node probe object	节点对探针
OPC_OBJTYPE_PBGS	41	global statistic probe object	全局统计量探针
OPC_OBJTYPE_PBSA	40	simulation attribute probe object	仿真属性探针
OPC_OBJTYPE_PBSTD_DEMAND	36	demand probe object	业务探针
OPC_OBJTYPE_PBSTD_LINK	34	link probe object	链路探针
OPC_OBJTYPE_PBSTD_NODE	33	node probe object	节点探针
OPC_OBJTYPE_PBSTD_PATH	35	path probe object	路径探针
OPC_OBJTYPE_PROC	13	processor object	处理器模块
OPC_OBJTYPE_PTRX	18	point-to-point link receiver object	点对点接收机

OPC_OBJTYPE_PTRXCH	27	point-to-point link receiver channel object	点对点接收信道
OPC_OBJTYPE_PTTX	17	point-to-point link transmitter object	点对点发信机
OPC_OBJTYPE_PTTXCH	28	point-to-point link transmitter channel object	点对点发送信道
OPC_OBJTYPE_QUEUE	14	queue object	队列模块
OPC_OBJTYPE_RARX	22	radio link receiver object	无线接收机
OPC_OBJTYPE_RARXCH	31	radio link receiver channel object	无线接收信道
OPC_OBJTYPE_RATX	21	radio link transmitter object (requires Wireless module)	无线发信机
OPC_OBJTYPE_RATXCH	32	radio link transmitter channel object	无线发送信道
OPC_OBJTYPE_STATWIRE	25	statistic wire object	统计线
OPC_OBJTYPE_STRM	24	packet stream object	包流
OPC_OBJTYPE_SUBNET_FIX	0	fixed subnet object	固定子网
OPC_OBJTYPE_SUBNET_MOB	1	mobile subnet object	移动子网
OPC_OBJTYPE_SUBNET_SAT	2	satellite subnet object	卫星子网
OPC_OBJTYPE_SUBQ	26	subqueue object	子队列
OPC_OBJTYPE_WDOMAIN	48	wireless domain object	无线域

3.3.2、中断类型常量

表3-8. OPNET 中断常量及描述

常量名称	值	英文描述	中文描述
OPC_INTRPT_ACCESS	11	access interrupt	接入中断
OPC_INTRPT_BEGSIM	8	begin simulation interrupt	开始编译中断
OPC_INTRPT_ENDSIM	9	end simulation interrupt	编译结束中断
OPC_INTRPT_ESYS_INTERFACE	14	external system interrupt	外部系统中断
OPC_INTRPT_ESYS_INTERFACE_EXT	15	external system interrupt (from cosimulator)	
OPC_INTRPT_FAIL	1	node or link failure interrupt	节点或链路失效中断
OPC_INTRPT_MCAST	12	multicast interrupt	多播中断
OPC_INTRPT PROCEDURE	2	procedure call interrupt	
OPC_INTRPT_PROCESS	13	process interrupt	进程中断
OPC_INTRPT_RECOVER	0	node or link recovery interrupt	节点或链路恢复中断
OPC_INTRPT_REGULAR	5	regular interrupt	周期中断
OPC_INTRPT_REMOTE	7	remote interrupt	远程中断
OPC_INTRPT_REMOTE_END_RCV	17	end of packet arrival at receiver	数据包接收完成中断
OPC_INTRPT_REMOTE_START_RCV	16	beginning of packet arrival at receiver	数据包接收开始中断
OPC_INTRPT_REMOTE_START_XMT	18	beginning of packet transmission when the split start radio transmission preference is TRUE	数据包发送开始中断
OPC_INTRPT_SELF	3	self interrupt	自中断
OPC_INTRPT_SELF_END_XMT	19	end of packet transmission	数据包发送完成中断

OPC_INTRPT_STAT	6	statistic interrupt	统计量中断
OPC_INTRPT_STRM	4	stream interrupt	流中断

3.4、对象属性

OPNET 通过对对象<—>子对象，对象<—>属性，记录整个网络的体系结构。通过对象 ID 区别对象模块。

(1) 对象 (Object):

对象是模型的一部分，而模型又是上层模块对象的一个属性。对象可以在模型中扮演下面的功能：定义行为、创建信息、储存和管理信息、处理、修改、转发信息、对事件作出响应、包含其他的对象。

(2) 属性 (Attribute):

对象的属性和一些允许访问属性或者使属性生效的程序组成了对象的接口。这些程序可以是 OPNET 自动生成的，也可以是用户编写的。包含有子对象的对象称之为复合对象 (compound object)。

(3) 模型属性 (Model Attribute) 和属性提升 (Attribute Promotion)

属性除了可以描述对象外，还可以用在模型上用来表示模型的参数。模型的属性机制可以提高模型的可重用性。具体地说，模型的属性被定义为模型的一部分，但同时，他们也出现在对象里，他们是在对象的模型被规定后被对象获得的。这是对象的本能操作。

类似这种模型属性机制，对象也可以被向上传给模型，这种机制就是所谓的属性提升。提升导致对象属性不再有值，而是作为模型的属性出现在上层属性中。对于一路提升到所有模型之上的 属性，我们可以把它看作是仿真系统的属性。从而把研究的系统看成是这些属性的函数。

3.4.1、对象拓扑

OPNET 通过对象的三层建模机制向上获取父对象，向下获取子对象。对象 ID 用于区别对象。表 3-9 为拓扑的常用函数，在仿真代码中经常使用。进程模型的父对象为处理器或队列模块，处理器或队列模块的父对象为节点模块，节点模块的父对象为子网模块。

表3-9. OPNET 拓扑常用函数

拓 扑 函 数	Objid op_topo_parent(Objid child_objid); 获得指定实体 ID 的父实体的 ID
	Objid op_topo_assoc(Objid objid, int direction, int objmtype, int index); 通过对象 ID，流动方向、关联、关联索引来获取相连的对象的 ID。其中流量方向取值 OPC_TOPO_ASSOC_IN 输入关联、OPC_TOPO_ASSOC_OUT 输出关联、OPC_TOPO_ASSOC_THROUGH 通过 关联。 例 strm_objid = op_topo_assoc (mymod_objid, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_STRM, 0);
	int op_topo_assoc_count (Objid objid, int direction, int objmtype); 获取与指定对象以指定连接方向和指定连接类型向关联的对象数目，连接方向取 OPC_TOPO_ASSOC_IN 或 OPC_TOPO_ASSOC_OUT

	<pre>例 num_out = op_topo_assoc_count (mod_objid, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_STRM);</pre>
	<pre>Objid op_topo_child (Objid parent_objid, int child_type, int child_index);</pre> <p>获取某对象下指令对象类型的第 x 个对象的 ID。</p>
	<pre>例 node_id = op_topo_child (subnet_id, OPC_OBJMTYPE_NODE, 0);</pre>
	<pre>int op_topo_child_count (Objid parent_objid, int child_type);</pre> <p>获取对象下某类型的所有子对象的数目</p>
	<pre>例 num_subqs = op_topo_child_count (subq_objid, OPC_OBJTYPE_SUBQ);</pre>
	<pre>Objid op_topo_connect (Objid src_objid, Objid dst_objid, int conn_objmtype, int conn_index);</pre> <p>通过源对象 ID, 目的对象 ID, 连接类型, 连接索引获取关联器的 ID</p>
	<pre>例 link_id = op_topo_connect (src_node_id, dest_node_id, OPC_OBJTYPE_LKDUP, 0);</pre> <p>获取连接 src_node_id 和 dest_node_id 之间的通过点对点双工链路连接的第 1 个链路的 ID</p>
	<pre>int op_topo_connect_count (Objid src_objid, Objid dst_objid, int conn_objmtype);</pre> <p>获取源对象与目的对象之间指定连接类型的连接的个数。</p>
	<pre>Objid op_topo_object (int objmtype, int index);</pre> <p>根据对象类型和索引获取对象 ID</p>
	<pre>例 ch_id = op_topo_object (OPC_OBJTYPE_BUTXCH, 1);</pre> <p>获取所有总线型发送信道中的第 2 个信道的 ID</p>
	<pre>int op_topo_object_count (int objmtype);</pre> <p>根据对象类型获取对象数目</p>
	<pre>例 num_nodes = op_topo_object_count (OPC_OBJTYPE_NDFIX);</pre> <p>获取固定节点的数目</p>
	<pre>Objid op_topo_parent (Objid child_objid);</pre> <p>获取父对象的对象 ID</p>
对 象 ID 函 数	<pre>Boolean op_id_check (Objid objid);</pre> <p>查询某 ID 对象实体是否存在</p>
	<pre>Objid op_id_from_hierarchical_name (const char * hname);</pre> <p>通过层次化名称获取对象 ID</p>
	<pre>例: node_objid = op_id_from_hierarchical_name ("top.net_name.node_name");</pre>
	<pre>Objid op_id_from_name (Objid parent_objid, int obj_type, const char* obj_name);</pre> <p>根据父对象的 ID, 当前指定对象的类型, 当前对象的名称获取指定对象的 ID。对象类型使用对象常量, 参考 3.3.1 常量</p>
	<pre>例: queue_objid = op_id_from_name (rem_node_objid, OPC_OBJTYPE_QUEUE, "buffer_a");</pre>
	<pre>Objid op_id_from_userid (int parent_objid, int obj_type, int userid);</pre> <p>通过父对象 ID, 指定对象类型, 用户 ID, 获取指定对象的对象 ID (仅在子网模型中的节点模块都有 user id 的属性, 可能会被隐藏, 但可以使用)</p>
	<pre>Objid op_id_self();</pre> <p>返回调用当前进程的处理器或队列 ID</p>
	<pre>int op_id_to_type (Objid objid);</pre> <p>获取指定对象的类型, 类型获取后与对象常量进行比较</p>

3.4.2、属性

对象通过属性设置仿真参数和运行机制选择。表3-10为OPNET中较多使用的属性函数。

表3-10. OPNET 属性常用函数

属性 读 取	Compcode op_ima_obj_attr_get (objid objid, const char* attr_name, void* value_ptr); 获得对象 ID 的属性名称的值，赋值到变量指针中。 例 op_ima_obj_attr_get (my_id, "user id", &my_user_id);
	Boolean op_ima_obj_attr_exists (Objid objid, const char* attr_name); 查看某对象是否存在指定属性。
	PrgT_Color op_ima_obj_attr_get_color (Objid objid, const char* attr_name, PrgT_Color* color_ptr); 获取一个对象颜色属性的值 例 op_ima_obj_attr_get_color (link_objid, "color", &mcolor); 获得某链路对象 color 属性的值，赋值给 mcolor
	Compcode op_ima_obj_attr_get_dbl (Objid objid, const char* attr_name, double * value_ptr); 获取对象中一个类型为 double 的属性的值到 double 变量地址中。返回结果为 OPC_COMPCODE_SUCCESS 成功或者 OPC_COMPCODE_FAILURE 失败
	Compcode op_ima_obj_attr_get_int32(Objid objid, const char* attr_name, int * value_ptr); 获取对象中一个类型为 int 的属性的值到 int 变量地址中。返回结果为 OPC_COMPCODE_SUCCESS 成功或者 OPC_COMPCODE_FAILURE 失败
	Compcode op_ima_obj_attr_get_objid(Objid objid, const char* attr_name, Objid* value_ptr); 获取对象中一个类型为 objid 的属性的值到 objid 变量地址中。返回结果为 OPC_COMPCODE_SUCCESS 成功或者 OPC_COMPCODE_FAILURE 失败
	Compcode op_ima_obj_attr_get_str (Objid objid, const char* attr_name, unsigned int buffer_size, char * value_buffer); 获取对象中一个类型为字符串的属性的值到字符指针中。返回结果为 OPC_COMPCODE_SUCCESS 成功或者 OPC_COMPCODE_FAILURE 失败 例 p_ima_obj_attr_get_str (tx_nodeid, "name", 128, tx_nodename);
	Compcode op_ima_obj_attr_get_str_sized (Objid objid, const char* attr_name, size_t* size_ptr, char* value_buffer); 获取对象字符串某字符串变量到指定大小的字符指针中。字符串大小包含结尾\0 例 op_ima_obj_attr_get_str_sized (tx_nodeid, "name", &size, tx_nodename);
	Compcode op_ima_obj_attr_get_toggle(Objid objid, const char* attr_name, int* value_ptr); 获取对象中一个类型为 toggle 的属性的值到 int 变量地址中。返回结果为 OPC_COMPCODE_SUCCESS 成功或者 OPC_COMPCODE_FAILURE 失败
	Compcode op_ima_obj_selfdesc_characteristic_get (Objid objid, const char * characteristic_name, int max_characteristic_len, char * value_ptr,

	<pre>OpT_Sim_Selfdesc_Characteristic_Type * type_ptr)</pre> <p>获取节点、链路、路径、业务模块的自我描述</p>
	<pre>Compcode op_ima_obj_hname_get (Objid objid, char * hname, int max_len);</pre> <p>获取一个对象的名称 例: char name[10]; op_ima_obj_hname_get (op_id_self (), name, 10)</p>
	<pre>Compcode op_ima_obj_attr_set (Objid objid, const char* attr_name, void* value);</pre> <p>设置对象属性的值 例 p_ima_obj_attr_set (txch_objid, "min frequency", freq_tx);</p>
	<pre>Compcode op_ima_obj_attr_set_color(Objid jid, const char* attr_name, PrgT_Color value);</pre>
属性设置	<pre>Compcode op_ima_obj_attr_set_dbl(Objid jid, const char* attr_name, double value);</pre>
	<pre>Compcode op_ima_obj_attr_set_int32(Objid jid, const char* attr_name, int value);</pre>
	<pre>Compcode op_ima_obj_attr_set_objid(Objid jid, const char* attr_name, objid value);</pre>
	<pre>Compcode op_ima_obj_attr_set_str(Objid jid, const char* attr_name, const char* value);</pre>
	<pre>Compcode op_ima_obj_attr_set_toggle(Objid jid, const char* attr_name, int value);</pre>
	<p>设置个类型的属性的值与获取时只是更改了最后一个输入参数类型。</p>

3.4.3、对象位置

在移动网络中，尤其是大范围移动通信网，如卫星网络时，往往需要获取节点位置，以实现定位功能，进而实现基于位置的网络协议。表 3-11 为 OPNET 中较多使用的对象位置信息函数。

表3-11. OPNET 对象位置信息常用函数

	<pre>Compcode op_ima_obj_pos_get (Objid site_objid, double* lat_ptr, double* long_ptr, double* alt_ptr, double* x_ptr, double* y_ptr, double* z_ptr);</pre> <p>获取节点或子网的位置信息，其中获取的海拔高度单位为 m，在获取通过节点的 altitude 属性获取的节点海拔高度单位与子网尺度单位相同。 例 op_ima_obj_pos_get (node_id,& lat,& lon,& alt,& x,& y,& z);</p>
	<pre>Compcode op_ima_obj_pos_get_time (Objid site_objid, double time, double* lat_ptr, double* long_ptr, double* alt_ptr, double* x_ptr, double* y_ptr, double* z_ptr);</pre> <p>获取节点或子网模块指定仿真时刻的位置信息 例 op_ima_obj_pos_get_time (node_id, op_sim_time(),& lat,& lon,& alt,& x,& y,& z);</p>
	<pre>Compcode op_ima_obj_pos_set_geocentric (Objid objid, double x, double y, double z);</pre> <p>设置对象到地心坐标系下指定位置</p>
	<pre>Compcode op_ima_obj_pos_set_geodetic (Objid objid, double lat, double lon, double alt);</pre>

设置对象到大地坐标系下指定位置

```
Compcode op_ima_traj_info_get (Objid objid, double time, double * ground_speed_ptr,
double * ascent_rate_ptr, double * end_time_ptr)
```

获取对象指定时间的水平速度和上升速度。时间必须为当前仿真时间及以后

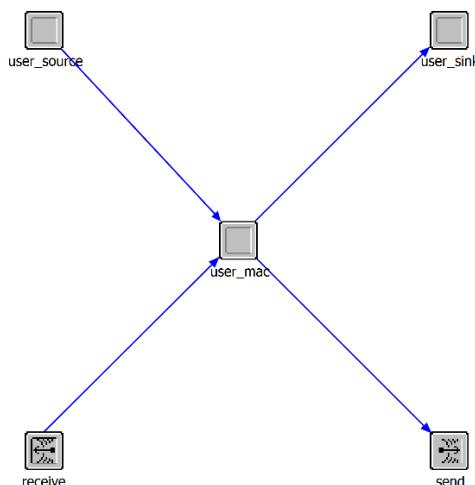
3.4.4、对象操作

在移动网络中，尤其是大范围移动通信网，如卫星网络时，往往需要获取节点位置，以实现定位功能，进而实现基于位置的网络协议。表 3-12 为 OPNET 中较多使用的对象位置信息函数。

表3-12. OPNET 对象操作常用函数

Compcode op_ima_obj_command (Objid objid, const char* cmd_name); 指定对象执行指定命令 例 op_ima_obj_command (tx_channel_objid, "abort");
op_ima_obj_event_notify (objid, event_type, start_service_time, end_service_time, num_bits, num_packets)
Compcode op_ima_obj_state_set (Objid objid, void* state_ptr); 为指定对象绑定一个结构体或值，将指针绑定在对象上
void* op_ima_obj_state_get (Objid objid); 获取绑定在对象上的结构体或值。
void* op_ima_obj_svar_get (Objid objid, const char* svar_name); 获取指定对象中的指定状态变量名称的状态变量的值。对象可以为处理器模块或队列模块 prio_thresh_ptr = (int *) op_ima_obj_svar_get (rem_queue_objid, "prio_thresh");

实例：



我们使用如上图所示的节点模型进行举例。

1 我们希望在 user_source 模块的进程模型中，获取 user_mac 的一个属性 user_id 属性值。
在 user_source 的进程模型中编写：

```
user_source_id=op_id_self(); // 获取当前进程所在模块 id，即 user_source 模块的 ID
node_id=op_topo_parent(user_source_id); // 获取节点 id，即节点模块的 ID
```

```
//根据名称，获取处理器模块 id
user_mac_id = op_id_from_name(node_id, OPC_OBJTYPE_PROC, "user_mac");
op_ima_obj_attr_get (user_mac_id, "user id", &user_id); //获取对象属性
```

2 在 user_mac 模块中设置无线发信机的最低频率为 30.0。

在 user_mac 模块的进程模型中编写：

```
user_mac_id=op_id_self(); //获取当前进程所在模块 id, 即 user_mac 模块的 ID
node_id=op_topo_parent(user_mac_id); //获取节点 id, 即节点模块的 ID
tx_id = op_id_from_name(node_id, OPC_OBJTYPE_RATX, "send"); //获取发信机的对象 ID
op_ima_obj_attr_get (tx_id, "channel", &comp_attr_objid); //获取复合属性 channel 的对象 ID
//获取第 1 个子对象
comp_attr_row_objid = op_topo_child (comp_attr_objid, OPC_OBJTYPE_RATXCH, 0);
//获取复合属性 Row 0 的对象 ID
op_ima_obj_attr_get (comp_attr_row_objid, "Row 0", &comp_attr_objid);
//获取第 1 个子对象
comp_attr_row_objid = op_topo_child (comp_attr_objid, OPC_OBJTYPE_RATXCH, 0);
//设置属性名时不包含变量单位名。
op_ima_obj_attr_set (comp_attr_row_objid, "min frequency", 30.0);
```

3.5、事件

表3-13. 事件结构表

时间	执行 ID	产生顺序	源对象 ID	目的对象 ID	类型	参数码	触发包	附加 ICI	附加状态
1.23	1200	1500	45	26	remote	12	NULL	0x456ef	NULL

我们知道 OPNET 仿真运行都是以事件的形式。OPNET 中定义事件使用 Evhangle 类型。可以理解为 OPNET 是在按照事件列表的顺序依次在指定的时间发起指定的中断，进程接收到中断后，安装状态转移进入指定状态，运行出入指定函数，后继续挂起。当事件被执行完后将会被从列表中删除，无法再通过之前获取的事件句柄获取事件中的参数值了。

事件列表的排列顺序先按照时间顺序，当时间参数相同时，再按照事件优先级排序。事件的表达形式则是以中断方式通知给进程。OPNET 中事件操作的常用函数如表 3-14 所示。

表3-14. 事件操作常用函数

Compcode op_ev_cancel(Evhangle env); 取消中断事件
Compcode op_ev_cancel_if_pending (Evhangle evhandle); 如果事件未执行，则取消事件
Evhangle op_ev_current (); 获理当前执行事件的句柄，所谓当前事件即为发起本次中断的事件。

int op_ev_type (Evhandle evhandle);	获得当前执行事件的类型。事件类型即为中断类型
int op_ev_code (Evhandle evhandle);	获取当前时间的中断码
int op_ev_count ();	获取剩余事件数量
int op_ev_count_local ();	获取发送给当前模块的事件数目。
OpT_uInt64 op_ev_current_execid();	获取当前事件的执行 ID。
Boolean op_ev_equal (Evhandle evhandle_a, Evhandle evhandle_b);	判断两个事件是否相等。
OpT_Ev_Id op_ev_id(Evhandle env);	获取事件 ID
Ici* op_ev_ici (Evhandle evhandle);	获取事件的 ICI 指针
Objid op_ev_dst_id (Evhandle evhandle);	获取事件目的模块 ID
Objid op_ev_src_id (Evhandle evhandle);	获取事件源模块 ID
Evhandle op_ev_next (Evhandle evhandle);	获取下一个事件的事件句柄
Evhandle op_ev_next_local (Evhandle evhandle);	获取当前模块下一个事件的事件句柄
Evhandle op_ev_seek_time (double time, int flag);	获取与指定时间最近的事件的事件句柄。其中 flag 可取值 OPC_EVSEEK_TIME_POST (指定时间后的第一个事件)、OPC_EVSEEK_TIME_POSTINC (指定时间或指定时间以后的第一个事件)、OPC_EVSEEK_TIME_PRE (指定时间之前的最后一个事件)、OPC_EVSEEK_TIME_PREINC (指定时间或指定时间之前的最后一个事件)。
int op_ev_stat (Evhandle evhandle);	获取与指定事件相关联的输入统计量索引。
double op_ev_time (Evhandle evhandle);	获取事件的时间属性
Boolean op_ev_valid (Evhandle evhandle);	判断事件是否存在。
Boolean op_ev_pending (Evhandle evhandle);	判断事件是否未执行。
int op_ev_strm (Evhandle evhandle);	获取事件相关联的流索引
Compcode op_ev_id_str_get (Evhandle evhandle, char * buffer);	获取一个事件的陈述

3.5.1、事件附件状态

附加状态是一组用户自定义结构的信息，其使用与在事件上附加 ICI 非常相似，只要设置状态指针，获取状态指针就可以了。由于不需要创建 ICI，设置 ICI 属性，获取 ICI，获取 ICI 属性等步骤，所以比 ICI 更方便。但是由于状态信息的结构完全由用户负责，且使用状态指针的双方都需要对信息结构有正确的认识，另外必须在编译前确定信息结构，不能在运行中更改结构定义，需要用户自己注意释放内存以防止内存泄露，状态信息的使用比 ICI 缺少灵活性。OPNET 中事件附加状态的常用函数如表 3-15 所示。

表3-15. 事件附件状态操作常用函数

void* op_ev_state (Evhandle evhandle); 获取事件附加状态
void* op_ev_state_clear (Evhandle evhandle) 清除事件附加状态
void* op_ev_state_install(void* client_state_ptr, SimT_Ev_State_Print_Meth state_print_proc); 向同一进程中的所有事件安装附加状态

3.5.2、中断

中断是进程模块处理的消息信息的机制。进程模型中，状态转移过程中，当遇到红色“阻塞”态时，在执行完“阻塞”态“入指令”后，会暂停函数运行，进程挂起。当接收到中断事件时，进程恢复，继续执行“阻塞”态“出指令”，并按照状态转移条件转入执行新的状态函数。在接收到中断时，同时能接收到中断事件携带的信息，包括中断码、ICI 等。中断类型及相关内容如表 3-16 所示。

表3-16. OPNET 各中断类型信息

中断类型	中断类型常量	值	中断源	被触发者	触发条件	启动方法
自中断	OPC_INTRPT_SELF	3	进程	进程自己	发起者指定期间到	op_intrpt_schedule_self()
进程中断	OPC_INTRPT_PROCESS	13		目标进程		op_intrpt_schedule_process()
过程调用中断	OPC_INTRPT PROCEDURE	2		指定函数		op_intrpt_schedule_call()
外部系统进程中断(出)	OPC_INTRPT_ESYS_INTERFACE_EXT			外部系统的回调函数		
流中断	OPC_INTRPT_STRM	4		模块	数据包到达	op_pk_send...(), op_pk_deliver...()

存取中断	OPC_INTRPT_ACCESS	11			发起请求	<i>op_intrpt_access()</i>
远程中断	OPC_INTRPT_REMOTE	7			发起者指定的时间到	<i>op_intrpt_schedule_remote()</i> 、 <i>op_intrpt_force_remote()</i>
多播中断	OPC_INTRPT_MCAST					<i>op_intrpt_schedule_mcast_global()</i>
统计量中断	OPC_INTRPT_STAT	6			统计量值发生改变	<i>op_stat_write()</i>
定期中断	OPC_INTRPT_REGULAR	5			周期性时间到	<i>intrpt interval</i> 属性
失效中断	OPC_INTRPT_FAIL/ OPC_INTRPT_DISABLED					<i>failure intrpts</i> 属性
恢复中断	OPC_INTRPT_RECOVER	0				<i>recovery intrpts</i> 属性
仿真开始中断	OPC_INTRPT_BEGSIM	8			仿真开始	<i>begsim intrpt</i> 属性
仿真结束中断	OPC_INTRPT_ENDSIM	9			仿真结束	<i>endsim intrpt</i> 属性
外部系统进程中断(入/出)	OPC_INTRPT_ESYS_INTERFACE			外部系统模块	写接口结束	
包发送开始	OPC_INTRPT_REMOTE_STA RT_XMT			收信机	管道模块计算的延时到	
包发送结束中断	OPC_INTRPT_SELF_END_X MT			收信机		
包开始接收	OPC_INTRPT_REMOTE_STA RT_RCV			接收机		
包接收完毕	OPC_INTRPT_REMOTE_END _RCV					

OPNET 中以事件机制进行建模，而事件以中断形式存在，所以对中断的控制成为网络协议仿真的主要操作，在 OPNET 中主要的中断操作如表 3-17 所示，以及后面介绍的各中断的发起。

表3-17. OPNET 中断控制常用函数

int op_intrpt_code(); 获取中断码
int op_intrpt_type(); 获取中断类型
int op_intrpt_strm();

获取流中断源的索引
int op_intrpt_stat ();
获取统计中断源的索引
op_intrpt_clear_self ();
取消所有未执行的自动断
op_intrpt_disable (int type, int code, Boolean next); 取消指定类型置顶中断码的中断,当 next 取 OPC_TRUE 时,只取消下一次这种中断,当 next 取 OPC_FALSE 时,取消今后所有这种中断。
op_intrpt_priority_set (int type, int code, int priority); 设置中断优先级
op_intrpt_priority_set_next (type, code, priority); 设置下一个事件的优先级
int op_intrpt_priority_get (int type, int code); 获取中断优先级
Objid op_intrpt_source (); 发起中断的模块 ID
op_intrpt_enable(int type,int code) 开启指定类型, 指定中断码的中断。
op_intrpt_enable_all() 开启所有中断类型
op_intrpt_disable (int type, int code, Boolean next); 取消指定类型, 指定中断码的中断, next 为 OPC_TRUE 表示只取消下一次, next 为 OPC_FALSE 表示永久取消。
int op_intrpt_esys_interface() 获取与当前中断相关联的外部接口的 ID
op_intrpt_port_register (int port_type, int port_index, Prohandle pro_handle); 注册一个进程为一个指定端口下到来的流中断和统计中断的接收者。端口类型为 OPC_PORT_TYPE_STRM (流中断) 或者 OPC_PORT_TYPE_STAT (统计中断)
void * op_intrpt_state_ptr_get() 获取与当前时间相关联的客户附加状态
op_intrpt_type_register (int intrpt_type, Prohandle pro_handle); 设置指定进程为当前模块未来指定中断的接收者。这是因为一个模块可以有多个进程,但是只能有一个进程能接收到外部发来的中断事件。默认为根进程接收。

3.5.2.1、多进程下的中断设置

由于中断只能被一个进程所接收,而很多进程的作用范围在模块级别,模块内可能含有很多个子进程,默认接收中断的进程为模块根进程。当需要接收中断的进程为子进程时,需要声明子进程并注册子进程为中断的接收者。

(1) 声明子进程

当存在多进程时首先声明某进程为当前进程的子进程。打开进程模型界面,点击“File”,选择 Declare Child Process Models,然后弹出声明进程选择窗口,选中需要添加的子进程,点击 OK。即声明成功。

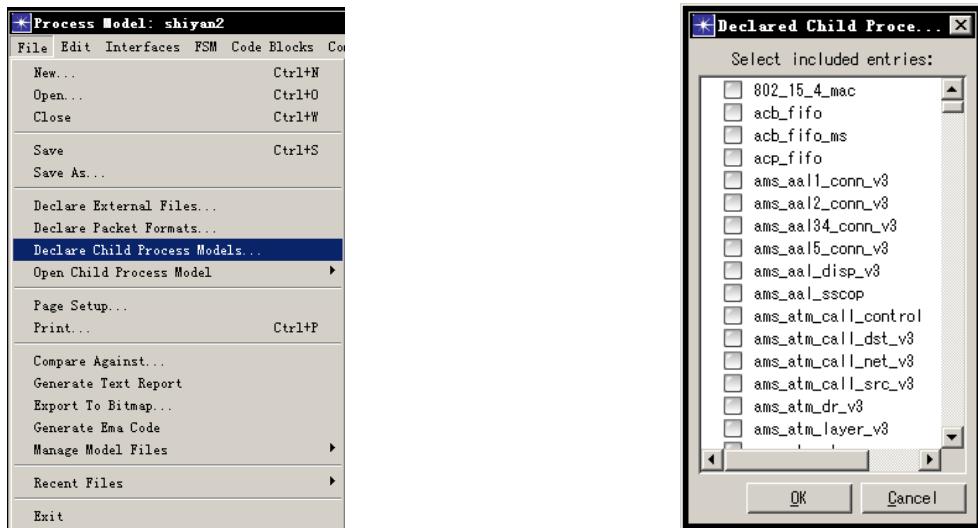


图3-4. OPNET 进程模型中声明子进程

(2) 创建子进程

```
Prohandle module_pro = op_pro_create ("子进程名称", OPC_NIL);
```

(3) 注册子进程为中断接收者

```
op_intrpt_type_register (OPC_INTRPT_REMOTE, module_pro);
op_intrpt_type_register (OPC_INTRPT_STRM, module_pro);
op_intrpt_type_register (OPC_INTRPT_STAT, module_pro);
```

(4) 激活进程

```
op_pro_invoke (module_pro, OPC_NIL);
```

其他关于进程的使用参考 2.4 进程模型

3.5.2.2、仿真开始中断

“仿真开始”中断设置简单，但却非常必要，也是非常容易忘记的。一个进程如果希望一开始就运行初始化状态，必须开启“仿真开始”中断。否则，进程是不会运行的。当新建进程后，第一件事情就是要记得开启这个中断。

启动方法：首先进入进程模型界面（状态转移图界面），点击菜单栏 Interfaces 按钮，点击 Process Interfaces 选项。

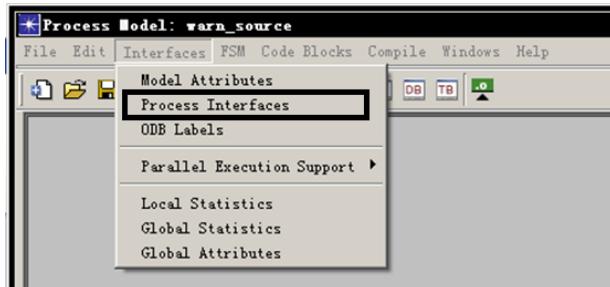


图3-5. OPNET 进程接口

进入进程接口设置界面，只需要将 begsim intrpt 的初始化值设置为 enabled，即开启了“开始编译”中断。可以看到，“结束编译”中断，“编译失败”中断，“周期性”中断，

“恢复”中断也是在 Process Interfaces 中设置，将属性值不要设置为 disabled。

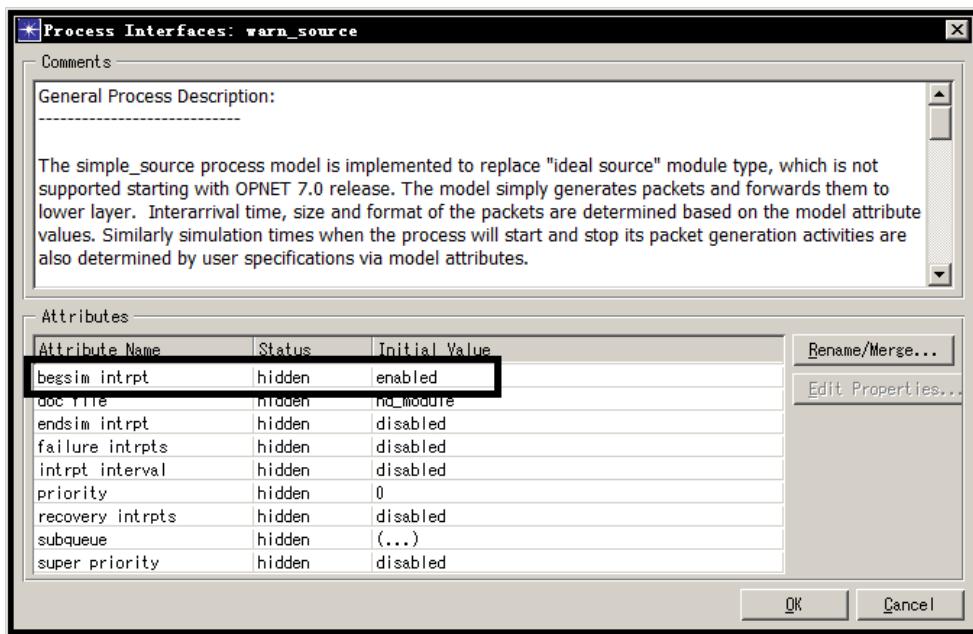
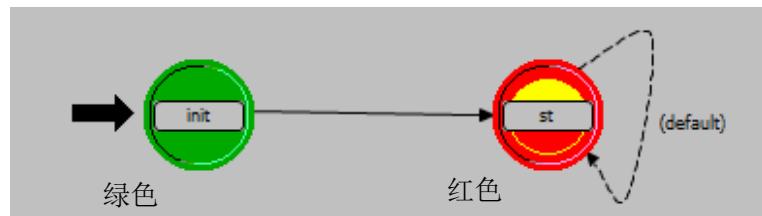


图3-6. OPNET 启动开始编译中断

开启“开始编译”中断后，程序才能自动启动，执行初始化状态代码。

如图所示，添加了“开始编译”中断，在没有其他中断时，程序自动运行到阻塞态。



如图所示，没有添加“开始编译”中断，在没有其他中断时，程序不运行。



3.5.2.3、自中断

表3-18. 自中断使用方法

启动方法	Evhandle evh= op_intrpt_schedule_self(double sim_time, int sim_code); 在仿真时间为 sim_time 秒时，当前进程接收到一个自中断，中断码为 sim_code	
中断常量	OPC_INTRPT_SELF	3
触发源	进程中代码	

被触发者	进程本身	
触发条件	进程指定的时间到	
参数	仿真时间	op_sim_time() + 向后推迟时间 T 非真实时间, 与程序运行缓慢无关
	中断码	用于区别每次的中断事件

自中断主要功能:

1 时间限制 (处理超时, 处理延时)

例: 设置指定事件后处理某事。

(1) 设定端

```
double      time_out=2.0;
int        TIME_OUT = 102;
Evhandle   time_out_evh = op_intrpt_schedule_self(op_sim_time() + time_out, TIME_OUT);
```

(2) 接收端

```
int    intrpt_type = op_intrpt_type();
int    intrpt_code = op_intrpt_code();
if(intrpt_type == OPC_INTRPT_SELF && intrpt_code == TIME_OUT)
{
    ...
}
```

若接收到数据包, 取消中断事件

```
op_ev_cancel(time_out_evh);
```

2 中断码区分 (识别中断目标)

例: 为 5 个用户设置剩余通话截止的中断

(1) 设定端

```
int      ii;
double   user_left_time[5];
for(ii=0;ii<5;ii++)
{
    user_left_time[ii]=ii*2;
    op_intrpt_schedule_self(op_sim_time() + user_left_time[ii], ii);
}
```

(2) 接收端

```
int intrpt_type = op_intrpt_type();
int intrpt_code = op_intrpt_code();
if(intrpt_type == OPC_INTRPT_SELF)
{
    switch(intrpt_code)
    {
        case 0:... break;
```

```

    case 1:... break;
}
}

```

3.5.2.4、进程中断

表3-19. 进程中断使用方法

启动方法	<pre> int 处理器 id = op_id_self(); int 节点 id = op_topo_parent (处理器 id); int 模块 id=op_id_from_name(节点 id,模块类型常量, "模块名称"); Prohandle 进程句柄 = op_pro_root(模块 id); Evhandle evh = op_intrpt_schedule_process(进程句柄,仿真时间,中断码); 向当前节点的某模块根进程发起进程中断。 </pre>	
中断常量	OPC_INTRPT_PROCESS	13
触发源	进程中代码	
被触发者	其他进程	
触发条件	进程代码中指定的时间到	
参数	目标节点 id	可以向当前节点内模块进程或其他节点内模块进程发起中断
	目标模块 id	目标节点下的某个某块，目标进程所属模块。
	模块类型常量	OPC_OBJTYPE_PROC 处理器模块 13 OPC_OBJTYPE_QUEUE 队列模块 14 其他参见“常量”章节
	模块名称	通过模块名称获取模块 id 更加直观，且更适合编程修改
	进程句柄	多进程时进程句柄并不一定是根进程
	仿真时间	op_sim_time() + 向后推迟时间 T 非真实时间，与程序运行缓慢无关
	中断码	用于区别每次的中断事件

进程中断主要功能：

1 功能通知（根据不同的中断码和变量共享，实现模块间消息传递）

例：设置共享变量和进程通知，共享变量在节点属性中设置，通知其他模块添加共享变量

(1) 设定端：

```

int user_address=12;
int user_id=3;
int my_id = op_id_self();
int my_node_id = op_topo_parent (my_id);
op_ima_obj_attr_set (my_node_id, "user_address",user_address);
op_ima_obj_attr_set (my_node_id, "user_id",user_id);
int module_objid = op_id_from_name(my_node_id, OPC_OBJTYPE_PROC , "source");
Prohandle module_pro = op_pro_root(module_objid);
op_intrpt_schedule_process(module_pro,op_sim_time(), 10);

```

(2) 接收端

```

int    intrpt_type = op_intrpt_type();
int    intrpt_code = op_intrpt_code();
if(intrpt_type == OPC_INTRPT_PROCESS && intrpt_code == 10)
{
    op_ima_obj_attr_get (my_node_id, "user_address",&user_address);
    op_ima_obj_attr_get (my_node_id, "user_id",&user_id);
    //添加用户信息的代码
}

```

3.5.2.5、远程中断

表3-20. 远程中断使用方法

启动方法	int 处理器 id = op_id_self(); int 节点 id = op_topo_parent (处理器 id); int 子网 id = op_topo_parent (节点 id); int 目标节点 id = op_id_from_name(子网 id, 节点类型常量, "目标节点名称"); int 目标模块 id = op_id_from_name(目标节点 id, 模块类型常量, "模块名称"); Evhandle evh = op_intrpt_schedule_remote(仿真时间, 中断码, 目标节点 id); 向当前子网下的指定节点指定模块下发起远程中断。	
中断常量	OPC_INTRPT_REMOTE	7
触发源	进程中代码	
被触发者	节点内模块	
触发条件	进程代码中指定的时间到	
参数	子网 id	可以向子网内选择目标节点发起远程中断
	目标节点 id	选择的远程中断的目标节点
	节点类型常量	OPC_OBJTYPE_NODE_MOB 移动节点 4 OPC_OBJTYPE_NODE_FIX 固定节点 3 OPC_OBJTYPE_NODE_SAT 卫星节点 5 其他参见“常量”章节
	模块类型常量	OPC_OBJTYPE_PROC 处理器模块 13 OPC_OBJTYPE_QUEUE 队列模块 14 其他参见“常量”章节
	目标节点名称	通过节点名称获取及节点 id 更加直观, 且更适合编程修改
	仿真时间	op_sim_time() + 向后推迟时间 T 非真实时间, 与程序运行缓慢无关
	中断码	用于区别每次的中断事件

远程中断用于模块间传递信息, 无需经过流或链路连接传递。目的模块可以与源进程不处于相同模块。远程中断只能到达目标模块, 不能进入模块内的具体进程。一个模块只能有一个进程接收到远程中断, 默认接收远程中断的进程为目标模块根进程。当目标模块存在多进程时, 且希望目标接收者为某子进程, 首先声明子进程, 然后将子进程注册为远程中断的接收者。另一种远程中断模式为强制远程中断, 使用代码

op_intrpt_force_remote (int code, Objid mod_objid); 实现，被激活进程会立即执行，源进程被暂时挂起，被激活进程挂起后，源进程继续执行。即产生了一个更高优先级的远程中断。

远程中断主要功能：

1 功能通知（根据不同的中断码和变量共享，实现模块间消息传递）

例，设置共享变量和进程通知，共享变量在节点属性中设置，通知其他模块添加共享变量

(1) 设定端

```
int user_address=12;
int user_id=3;
int my_id = op_id_self();
int my_node_id = op_topo_parent (my_id);
op_ima_obj_attr_set (my_node_id, "user_address",user_address);
op_ima_obj_attr_set (my_node_id, "user_id",user_id);
int module_objid = op_id_from_name(my_node_id, OPC_OBJTYPE_PROC , "source");
Prohandle module_pro = op_pro_root(module_objid);
op_intrpt_schedule_process(module_pro,op_sim_time(), 10);
```

(2) 接收端

```
int intrpt_type = op_intrpt_type();
int intrpt_code = op_intrpt_code();
if(intrpt_type == OPC_INTRPT_PROCESS && intrpt_code == 10)
{
    //添加用户信息的代码
}
```

3.5.2.6、流中断

表3-21. 流中断使用方法

启动方法	Packet* 包指针 = op_pk_create (数据包大小); op_pk_send(包指针, 包流输出索引); 或者 op_pk_send_delayed(包指针, 包流输出索引, 延迟时间); 或者 op_pk_send_forced(包指针, 包流输出索引); 发送无格式数据包。在节点内沿包流线传输、在节点间沿链路传输。	
中断常量	OPC_INTRPT_STRM	4
默认中断码	1	
触发源	进程中代码	
被触发者	模块	
触发条件	数据包接收完成	
参数	包指针	包指针指向一块内存区域，当包被发送后，源进程将不再能读取包中内容。
	包流线出口	包流出口，决定包的去向

数据包发送方式：

1 计划发送（相同时间多个中断时，优先处理其他中断）

```
op_pk_send(包指针, 包流输出索引);
op_pk_send_delayed(包指针, 包流输出索引, 延迟时间);
```

2 静默发送（目标模块接收不到流中断，数据包缓存到包流中）

```
op_pk_send_quiet(包指针, 包流输出索引);
```

3 强制发送（相同时间多个中断时，优先处理此中断）

```
op_pk_send_forced(包指针, 包流输出索引);
```

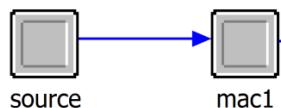
数据包接收：Packet* frame = op_pk_get(op_intrpt_strm()); //从当前流中断口的缓存中读取数据包

注意事项：

由于数据包传输是缓存在包流中的，包流中可以缓存多个包。当数据包到来时，先把数据包缓存在包流中，再向目的模块发起流中断，如果目的模块没有通过 op_pk_get() 将数据包取走，则包会一直缓存到包流中，知道下次使用 op_pk_get() 获取数据包。所以即使我们知道不可能存在流中断的时间端接收到流中断，也要把数据包从包流中取走，销毁。

(1) 模块间循环发送数据包的操作。

举例：



循环发送数据包时使用 op_pk_send() 函数，先执行所有的发送函数，再执行所有的接收函数。而这些过程都是在同一仿真时间完成的。

source 循环发送数据包

```
for(ii=0;ii<3;ii++)
{
    pk = op_pk_create (120);
    op_pk_send (pk,0);
    printf("source send %d packet\n",ii);
}
```

mac 接收数据包

```
intrpt_type = op_intrpt_type();
if(intrpt_type == OPC_INTRPT_STRM)
{
    printf("mac receive %d stream intrpt. time is %fs\n",receive_num,op_sim_time());
    receive_num++;
}
```

运行结果

```
source send 0 packet
source send 1 packet
source send 2 packet
mac receive 0 stream intrpt
mac receive 1 stream intrpt
```

```
mac receive 2 stream intrpt
```

循环发送数据包时使用 op_pk_send_forced() 函数，能在发送一个数据包后立即执行流中断接收函数。然后再发送下一个数据包。而这些过程都是在同一仿真时间完成的。

source 循环发送数据包

```
for(ii=0;ii<3;ii++)
{
    pk = op_pk_create (120);
    op_pk_send_forced(pk,0);
    printf("source send %d packet\n",ii);
}
```

mac 接收数据包

```
intrpt_type = op_intrpt_type();
if(intrpt_type == OPC_INTRPT_STRM)
{
    printf("mac receive %d stream intrpt\n",receive_num);
    receive_num++;
}
```

运行结果

```
mac receive 0 stream intrpt
source send 0 packet
mac receive 1 stream intrpt
source send 1 packet
mac receive 2 stream intrpt
source send 2 packet
```

(2) 无线链路多包冲突

节点间发送数据包沿链路方向，或在无线链路中广播。传播时延由链路模型或无线链路的传播时延管道阶段函数决定。在链路中包的传输就涉及数据包冲突问题。实际中当两个数据包传输时间上有重叠，则认为两个包都接收失败。在 OPNET 中数据包的相互干扰是通过误码数目，纠错码数目，以及误码门限三个参数决定。在误码门限等于 0 时，表示只要发生碰撞或其他形式产生误码就不能接收。也就不会产生流中断了。在误码门限等于 1 时，表示无论误码数目是多少，都能完整接收数据包。而数据包是否碰撞是在无线接收机管道阶段的噪声干扰函数中设定的。在部分 OPNET 噪声干扰模型中，后来的数据包会影响先到数据包的接收，使得接收不到先到数据包的流中断，也没有缓存存在。但如果后来数据包没有被再后来数据包干扰，即使这个数据包干扰了之前数据包的接收，他也能被接收到，也会有流中断产生。这与实际中不同。

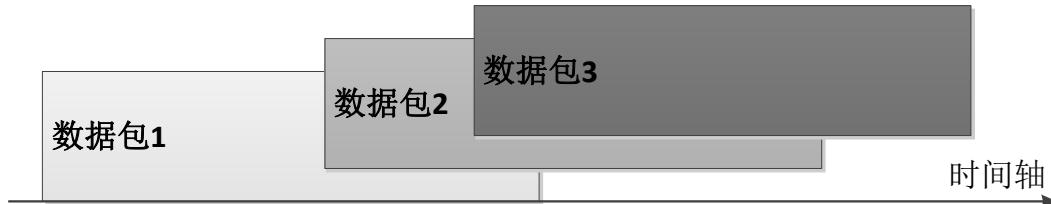


图3-7. 多包接收冲突示意图

如图所示的3个数据包依次发送到接收机，数据包1和数据包2不能被接收到，数据包3可以被接收到。但实际生活中，这3个数据包都不能被接收到，所以需要添加统计量中断先做判断。而流中断的事件优先级低于统计量中断的事件优先级。所以在同一时间上进程先获得统计量中断，再获得流中断。

(3) 模块与发信机间的包传输

仿真常常希望源模块发送数据包给目标模块，当目标模块接收到流中断时，使用 `receive_frame = op_pk_get(op_intrpt_strm());` 读取数据包，以此来完成数据包的流动。

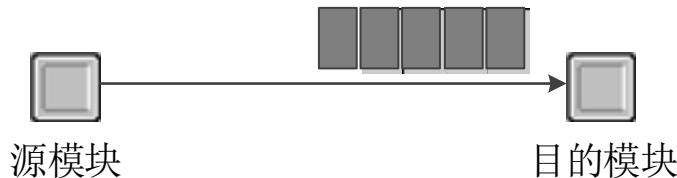


图3-8. 多包接收冲突示意图

源模块发送数据包给目标模块，先等待设定的时延结束后，源模块根据发送模式发起流中断，同时将数据包缓存到目的模块。目的模块接收到流中断以后，通过代码从缓存中获取数据包。但在此过程中有时会存在几个常见使用错误。

- 第一个错误。源模块发送了数据包到缓存上，目标模块却没有接收到流中断。
- 第二个错误。目标模块接收到流中断，却没有把数据包从缓存中取走，以至于下次再接收到流中断，读取的数据包为上一次流中断的数据包。
- 第三个错误，数据包在模块中丢失，是可能缓存数据包已经超过模块容量。当数据包过多时，模块会自动删除后来的数据包。

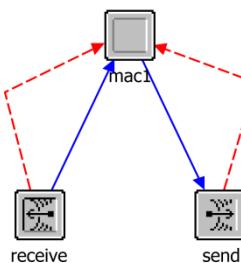
3.5.2.7、统计量中断

表3-22. 统计中断使用方法

启动方法	统计量中断通过统计线设定，主要在无线收发信机与模块间。 <code>op_stat_write()</code>	
中断常量	OPC_INTRPT_STAT	6
默认中断码	0	
触发源	统计线源头	
被触发者	统计线指向（节点内模块）	
触发条件	统计量值发生改变	
参数	中断接收者	统计量中断线连接促发模块和被促发模块。
	中断号索引	用于目标模块存在多个统计量中断源时区分源头
	触发条件	仅在统计量变化瞬间触发中断。
常用函数	<code>int 统计量索引=op_intrpt_stat(); //获取统计量索引</code> <code>double status=op_stat_local_read(统计量索引); //统计的值为事件后的值</code>	

在统计量中断时，我们常常希望获取当前统计中断源，和某一统计量的值。OPNET 使用 int op_intrpt_stat () 获取当前统计中断。使用 double op_stat_local_read(int index) 获取某个统计中断索引的统计量的值。

统计量中断设置：



如上图中虚线所示为统计中断线，右键单击统计量中断线，进入统计量中断属性设置窗口。

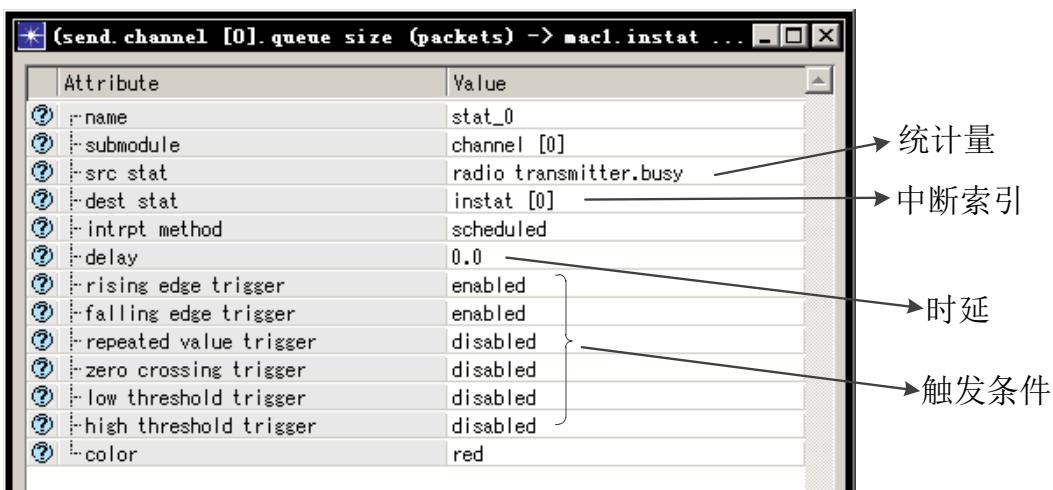


图3-9. OPNET 统计量中断线属性设置窗口

在统计中断属性设置窗口，可以设置统计中断的统计量与触发条件。属性使用如表 3-23 所示。

表3-23. OPNET 统计量中断设置

属性含义	取值	备注
src stat	统计量	不同模块的统计量不同
dest stat	中断索引	当同一个目标模块，有多个触发源时，需要判断中断来源
delay	时延	中断时延
rising edge trigger	上升沿	统计量值由低升高时有效，如由 0 变成 1。disabled 不能触发统计量中断，enabled 能触发统计量中断。
falling edge trigger	下降沿	统计量由高变低时有效，如由 3 变成 2。disabled 不能触发统计量中断，enabled 能触发统计量中断。
repeated value trigger		
zero crossing trigger	过 0	统计量由非 0 变成 0 时触发。disabled 不能触发统计量中断，enabled 能触发统计量中断。
low threshold trigger	低门限值	当低于低门限值时，触发统计量中断。disabled 不能触发统计量中断，使用需要设置具体值。

high threshold trigger	高门限值	当高于高门限值时，触发统计量中断。disabled 不能触发统计量中断，使用需要设置具体值。
---------------------------	------	--

点击 src stat 属性值设置时，弹出触发模块所具有的统计量。

无线发信机所具有的统计量，如图 3-10 所示。

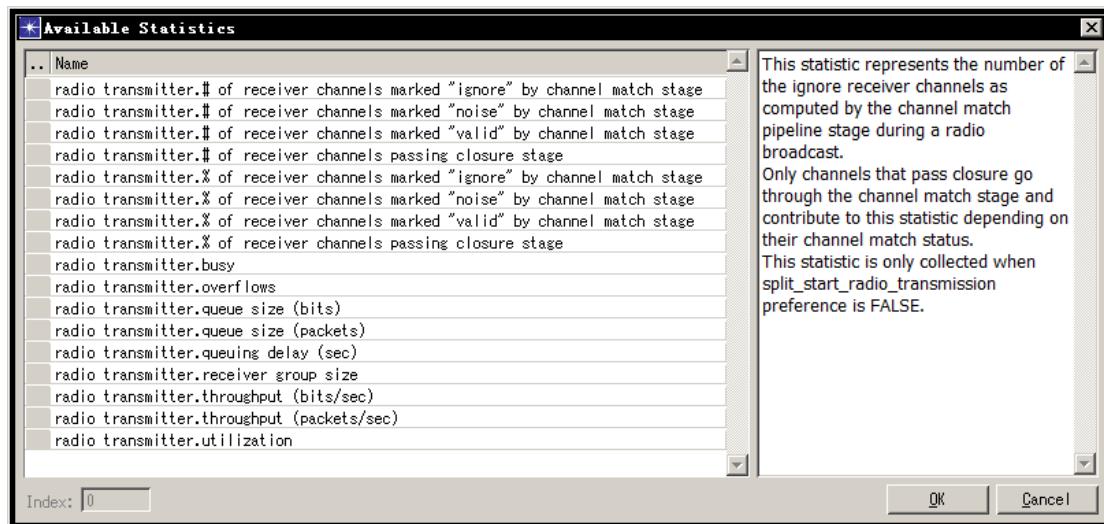


图3-10. OPNET 无线发信机统计量描述窗口

图3-11. OPNET 无线发信机统计量

名称	含义	使用
radio transmitter.busy	发信机忙	当发信机开始发送数据，统计量值变成 1.0，当发信机发送完毕，统计量值变成 0.0
radio transmitter.overflow	发信机数据溢出	当缓存比特数(包)超出比特(包)缓存容量时，统计值变成 1。需要注意，当超过时，发信机会自动删除后来包。
radio transmitter.queue size(bits)	发信机比特容量变化	当发信机模块接收完其他模块发来的包时，统计量值增加接收包的比特数，当发信机发送完一个数据包时，统计量的值减少发送包的比特数。
radio transmitter.queue size(packets)	发信机包容量变化	当发信机模块接收完一个其他模块发来的包时，统计量值增加 1，当发信机发送完一个数据包时，统计量的值减少 1。
radio transmitter.queueing delay(sec)	数据包等待发送时延	当发信机完全发送完一个数据包时，统计量的值为此数据包的等待发送时延(第一个比特进入到最后一个比特离开的时间)
radio transmitter.receiver group size	发信机接收机分组数目	无线信道接收机分组的数目，在无线链路管道模型第一个函数阶段中表述了接收机分组的含义
radio transmitter.throughput(bit/sec)	发信机平均每秒的	无线发信机每秒成功传输的比特数目，查看结果时，使用 time average 模式观

	比特流量	看
radio transmitter.throughput (packets/sec)	发信机平均每秒的包流量	无线发信机每秒成功传输的包数目，查看结果时，使用 time average 模式观看
radio transmitter.utilization	发信机信道利用率	无线发信机每秒传输的数据比特数与信道带宽的比值，以 100.0 表示 100% 利用。查看结果使用 time average 表示

无线接收机所具有的统计量，如图 3-12 所示。再无线接收机的中断均是在流中断之前发起。即连接无线接收机的模块先接收到统计量中断，再接收到流中断。

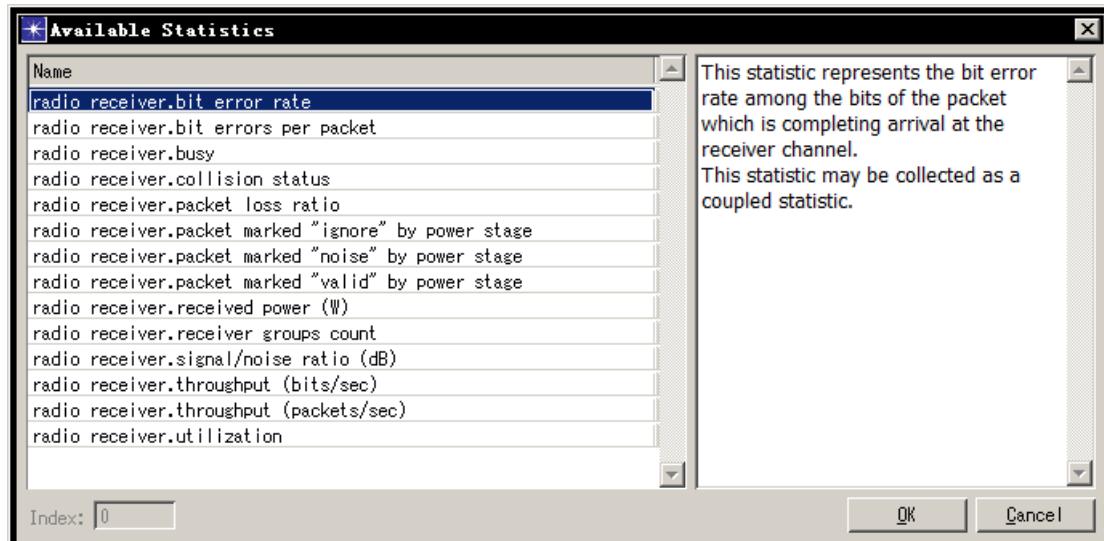


图3-12. OPNET 无线接收机统计量描述窗口

图3-13. OPNET 无线接收机统计量

名称	含义	使用
radio receiver.bit.error.rate	数据包误码率	当接收机接收完一个数据包后，统计量的值变成此数据包的误码率。
radio receiver.bit.errors.per.packet	数据包误码数目	当接收机接收完一个数据包后，统计量的值变成此数据包的误码数目。默认误码数目为 0
radio receiver.busy	接收机忙	当接收机正在接收数据包，统计量为 1，接收完成，恢复 0
radio receiver.collision.status	接收冲突	当接收机中开始发生冲突，统计量值变成 1。当多个冲突数据包最后一个接收完毕，统计量变成 0。
radio receiver.packet.loss.ratio	数据包丢失	当接收机接收完一个数据包，如果抛弃，统计量值为 1，如果接收，统计量为 0
radio receiver.received.power(W)	接收机接收	并不是实际中接收机当前接收到

	某一数据包时的功率值	的功率值，而是针对当期数据包的功率分量。
radio receiver.received groups count	接收机分组数目	接收机开始接收数据包时，属于同一无线接收信道的接收机分组数目
radio receiver.signal/noise ratio(dB)	信噪比	接收机开始接收数据包时
radio receiver.throughput(bits/sec)	比特流密度	接收机每秒成功接收到的比特数目，查看时使用 time average 方式。
radio receiver.throughput(packets/sec)	包流密度	接收机每秒成功接收到的包数目，查看时使用 time average 方式。
radio receiver.utilization	信道利用率	接收机每秒传输的比特数目与带宽的比值，查看使用 time average 方式。

统计量中断与流中断同一时间同时产生时，按照先后顺序，被触发模块，先接收到统计量中断，再接收到流中断。例如，设置无线接收机统计量属性为下图。在接收机功率由 3.16E-013w 之下变化为 3.16E-013（包括）之上时，会触发中断。当接收机功率由 0 之上变化为 0（包括）之下，会触发中断。

Attribute	Value
name	stat_1
submodule	channel [0]
src stat	radio receiver.received power (W)
dest stat	instat [1]
intrpt method	scheduled
delay	0.0
rising edge trigger	disabled
falling edge trigger	disabled
repeated value trigger	disabled
zero crossing trigger	disabled
low threshold trigger	0.0
high threshold trigger	3.16E-013
color	red

图3-14. OPNET 无线接收机统计中断设置示意图

则在接收到如图所示的两个数据包到来时。

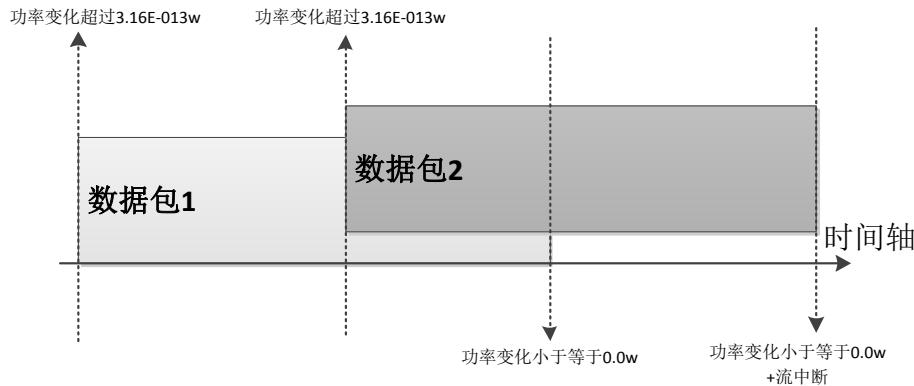


图3-15. OPNET 无线接收机发起的中断示意图

产生如下 4 个统计量中断和 1 个流中断

第 1 个统计量中断	统计量目标模块在第 1 个数据包到来时会接收到统计量中断，使用函数 double status=op_stat_local_read(统计量索引)；获取的结果是一个大于 0 的功率值，但是由于功率值以 w 为单位，数值有时比较小，要显示到小数点后 20 才能看的清楚。
第 2 个统计量中断	目标模块在第 2 个数据包到来时接收到统计量中断，这时使用函数 double status=op_stat_local_read(统计量索引)；获取的结果为第 2 个数据包在接收机上的功率值。而不是两个数据包在接收机上的功率和。
第 3 个统计量中断	目标模块在第 1 个数据包接收结束时接收到统计量中断，但是接收不到流中断，因为此时有其他数据包在接收。使用函数 double status=op_stat_local_read(统计量索引)；获取的结果为 0，而不是接收机正在接收的第 2 个数据包的分量值，这是因为功率值与实际生活中的综合功率值有区别，OPNET 里接收机功率值都是针对每个数据包的功率分量。这是 OPNET 的事件处理机制产生的。
第 4 个统计量中断	目标模块在第 2 个数据包接收结束时接收到统计量中断，能接收到流中断，因为 OPNET 只能向后检测判决数据包是否冲突。使用函数 double status=op_stat_local_read(统计量索引)；获取的结果为 0
流中断	在某些情况下，在接收到第 2 个数据包发送结束时的统计量中断后会立刻接收到流中断。为第 2 个数据包。

在上面的冲突过程中，OPNET 并不能完美的判决数据包冲突，需要我们在仿真中设置。比较简单的方法为接收到数据包后，先读取数据包 TDA 中的冲突次数标志位，当冲突次数标志位的值大于 0，则销毁数据包。否则继续运行。

3.5.2.8、其他中断

(1) 多播中断：

多播中断会同时到达仿真系统中的所有进程，提供了进程间多播通信的途径，只是目前目标的针对性比远程中断还差。

(2) 定期中断：

定期中断常用于安排自动的、周期性的行为，比如输入的轮询、状态消息的传输、标志

总线周期开始等周期性活动。

(3) 失效中断、恢复中断：

失效中断用于异步指示某节点或链路的失效，恢复中断则用于异步指示他们从失效中恢复。自动重新配置路由表的功能通常需要这两个指示。进程模型可以在接口上定义接收仿真系统中所有节点和链路的失效/恢复中断（Network Wide），或者只接受本进程所处节点的失效/恢复中断（Local Only）

(4) 仿真结束中断

仿真结束中断核心是仿真核心在仿真结束时刻产生的仿真结束的通知，通常用于触发结束时的信息记录，如标量统计量、自定义报表等，在与外部程序进行联合仿真时使用。

(5) 外部系统进程中断(入)

外部系统进程中断（入）时由协同仿真中的外部系统代码向外部模块的接口写数据引起的，通常用于指示外部模块读取这些数据并将数据转交到仿真系统中的合适模块。

(6) 外部系统进程中断(出)

外部系统进程中断（出）用于通过外部模块的接口向外部系统发送信息。

(7) 包头到达收信机中断、包尾到达收信机中断、包传输开始中断、包传输结束中断

这些中断都是核心用来推挤管道模型本运作的事件，进程模型不用关心。

3.5.3、ICI

通过表 3-13 可以看到每个事件都包含一个 ICI 字段。ICI 是基于接口控制信息（Interface Control Information）的通信机制，类似于基于包的通信机制，并且 ICI 数据结构也类似于包数据结构，但是它比包结构更简单，只包含用户自定义的域，而不存在封装的概念。

广义 ICI 是与事件关联的用户自定义的数据列表。如果某个事件希望传递信息给予它相隔一段时间的将来某个事件，可以将 ICI 绑定在将来这个事件中，等到它将来发生时就可以取出 ICI 信息。由于 ICI 是以事件为载体，所以它可以用在各种有关事件调度的场合，基于 ICI 的通信适用于任何事件，而且常和流事件一起使用，虽然流事件源于包的传输，但是如果需要传输额外的信息又想避免使用包本身，这时可以用 ICI。例如协议栈中的高层协议模块在向底层传输包的同时可以通过 ICI 携带这个包相应的服务等级和目的地址。

3.5.3.1、ICI 格式定义

点击 File-New，选择 ICI Format 进入 ICI 格式定义窗口。ICI 格式定义主要包含数据的属性域（属性名：Attribute Name、数据类型：Type 和默认值：Default Value），如图 3-16 所示。

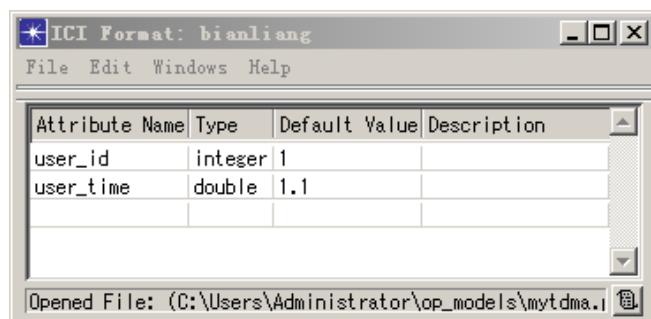


图3-16. ICI 格式定义

属性名是读写 ICI 数据的依据，它的作用和包域名称一样，以属性名作为输入参数可以对相应数据进行设置 (op_ici_attr_set)、读取 (op_ici_attr_get) 和存在性判断 (op_ici_attr_exists) 等操作。

3.5.3.2、ICI 操作

ICI 是仿真中进程动态创建的对象。以 ICI 格式文件名为输入参数，调用 op_ici_create() 可以返回一个相应的 ICI 指针，它作为所有后续操作的依据。OPNET 提供了专门针对 ICI 操作（创建，设置和读取属性，绑定和销毁）的核心函数还有其他一些有关 ICI 的函数，现归纳如表 3-24 所示。

表3-24. ICI 常用函数

Ici * op_ici_create (const char * fmt_name);	创建一个指定格式的 ICI
op_ici_install (Ici * iciptr);	为进程安装一个 ICI
Boolean op_ici_attr_exists (Ici * iciptr, const char* attr_name);	判断某个 ICI 是否包含某个属性字段
op_ici_attr_get (Ici * iciptr, const char* attr_name, void* value_ptr);	获取 ICI 中某个属性字段的值，存在 value_ptr 中，在使用时类型要匹配。
op_ici_attr_set (Ici * iciptr, const char* attr_name, void * value);	设置某个 ICI 属性字段的值，在使用时类型要匹配。
op_ici_format (Ici * iciptr, char* format_name);	获取一个 ICI 的格式
op_ici_id (Ici * iciptr);	获取 ICI 的 ID
op_ici_id_str_get (Ici * iciptr, char* buffer);	获取一个 ICI 的描述
op_ici_print (Ici * iciptr);	输出 ICI 内容
op_ici_destroy (Ici * iciptr);	销毁一个 ICI

3.5.3.3、ICI 与事件

为了将一个 ICI 与一个事件关联，仿真核心采用一种称为绑定 (Installation) 的机制。在任意时刻每个进程一次最多只能绑定一个 ICI，具体来说，如果进程多次调用 op_ici_install() 绑定 ICI，最后一个才是真正起作用的。绑定 ICI 后，对于进程生成的新事件，仿真核心自动将绑定的 ICI 地址与该事件相关联，对于后续事件也做相同处理，直到进程绑定另一个 ICI（称为 ICI 更新）。

一般来说，某个 ICI 只针对特定事件，而对于后续事件，该 ICI 是没有意义的，但是默认情况下仿真核心仍会将后续事件与之关联，为了避免这种情况可以调用 op_ici_install(OPC_NIL) 拆除当前 ICI 的绑定（绑定空指针即拆除）。实际上，如果某个事件不需要 ICI，但是意外地与 ICI 关联，也不会对仿真产生任何负面影响。事件在执行之后会被清除，但是 ICI 内存不会被清除。

ICI 的安装与事件是相对独立的。一个进程只保留一个 ICI 指针，一个进程默认的 ICI 内容为空。当安装了新的 ICI 以后，进程将清除旧的 ICI 内容，只保留当前最新的 ICI 内容。事件在创建时（不是发生时），会自动复制进程 ICI 内容到自己的 ICI 区域中。这样即使在事件创建以后进程 ICI 更改了，已创建事件的 ICI 仍不会变。示意图如图 3-17 所示。

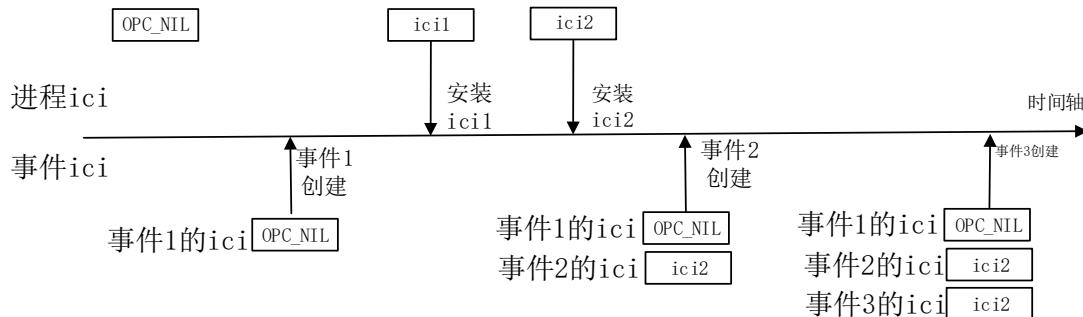


图 3-17. OPNET 事件 ICI 变化示意图

例：

(1) ICI 创建状态

设置状态变量 Evhndl \evh1;

```
Ici* myici1 = op_ici_create ("ici1"); // 创建名称为 ici1 的 ici 指针
op_ici_attr_set(myici1, "user_id", 2); // 设置 ici 的 user_id 参数值
op_ici_install(myici1); // 安装 ici 到后面的事件
evh1= op_intrpt_schedule_self(op_sim_time() + 10, 100); // 设置事件
op_ici_install(OPC_NIL); // 清除 ici
```

(2) ICI 接收状态

```
int myuser_id = 0;
if (op_intrpt_code () == 100)
{
    Ici* iciptr = op_intrpt_ici (); // 读取当前事件的 ici
    // Ici* ici_ptr = op_ev_ici (evh1); // 获取指定事件，可以没发生或的
    // 查看当前事件的 ici 中是否含有 user_id 参数
    Boolean exit = op_ici_attr_exists(iciptr, "user_id");
    if (exit)
    {
        op_ici_attr_get (iciptr, "user_id", &myuser_id);
    }
}
```

3.5.3.4、ICI 与数据包

ICI 除了可以绑定到事件上，同样可以绑定到数据包中。只要将 ici 绑定到数据包上，接收到数据包后读取即可。

(1) 数据包发送端

```
Ici* myici1 = op_ici_create ("ici1");           //创建 ici
Packet* pkptr = op_pk_create (100);    //创建数据包
op_pk_ici_set (pkptr, myici1);           //绑定 ici 到数据包
op_pk_send(pkptr,0);                     //发送数据包
```

(2) 数据包接收端

```
int intrpt_type=op_intrpt_type();           //获取中断类型
int myuser_id=0;
if(intrpt_type==OPC_INTRPT_STRM)
{
    Packet* pkptr = op_pk_get (0); //读取数据包
    Ici* myici1 = op_pk_ici_get (pkptr); //读取数据包中的 ici
    Boolean exit = op_ici_attr_exists(myici1, "user_id"); //判断是否存在某字段
    if(exit)
    {
        op_ici_attr_get (myici1, "user_id", &myuser_id); //读取 ici 中字段的值
    }
}
```

3.6、数据包

数据包的交互是网络协议中的主要流程，因此数据包的使用是了解一个网络协议的重要方法。数据包的使用包含数据包格式的定义、数据包的创建、数据包字段和属性的设置、数据包发送，数据包捕获，数据包属性和字段的读取，数据包复制、销毁等操作。

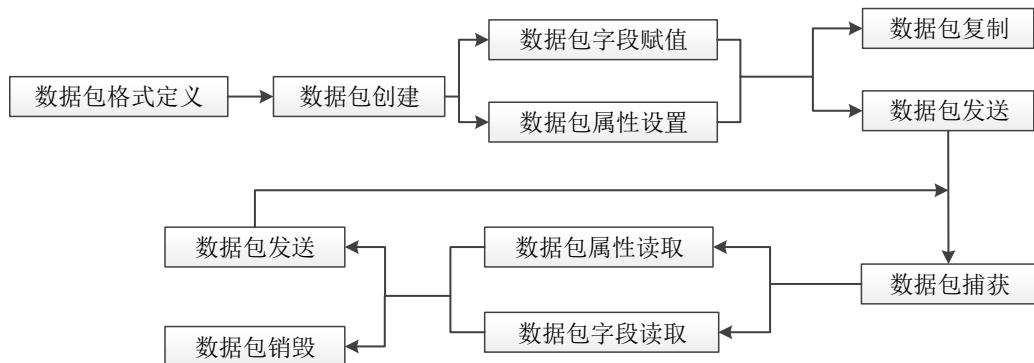


图3-18. 数据包传输流程

由于网络的流程主要以数据包的交互为主，在数据包中除了用户定义的字段外还包括其他信息，如图 3-19 所示。通过数据包指针，还可以获得数据包属性域和 TDA 数据。

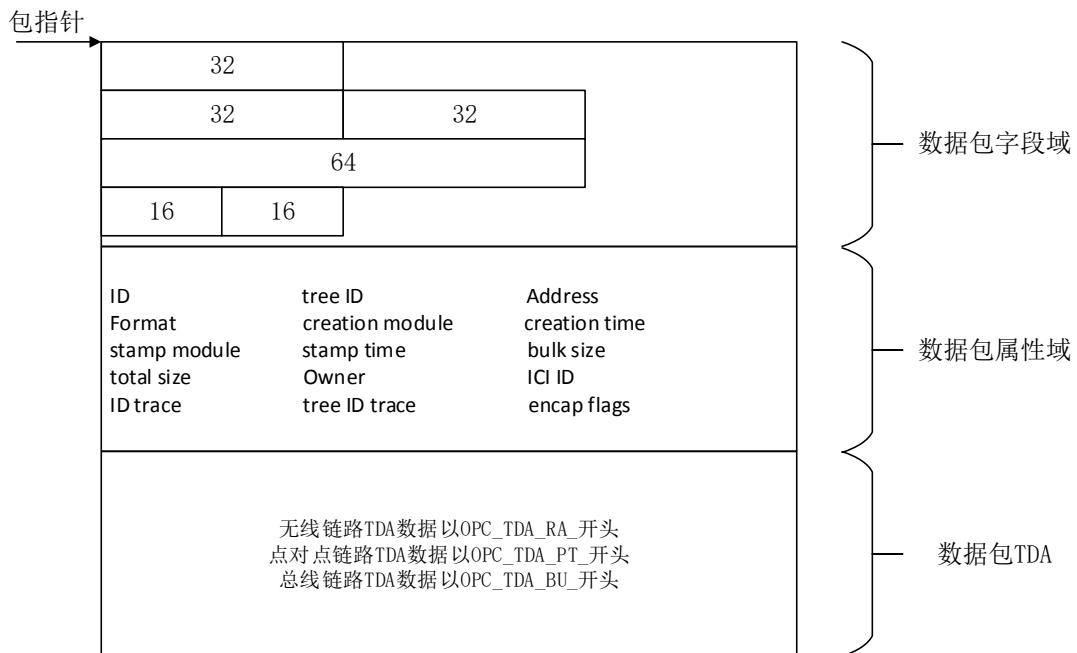


图3-19. 数据包各数据域

3.6.1、数据包格式的定义

OPNET 中包含有格式数据包和无格式数据包。OPNET 中定义了一系列的数据包格式，如 802.11 中的 Beacon 帧、data 帧、ACK 帧等。当我们想调用 OPNET 中自带的一些数据包格式时，我们要在进程模型界面点击 File—Declare Packet Formats 声明外部数据包格式。这样在相应的进程模型中才能使用此数据包格式。当我们想自己定义数据包格式时，点击菜单栏 File，选择 New，在创建内容中选择 Packet Format，点击 OK，进入数据包定义窗口。在数据包定义窗口，点击 Fields 菜单，选择 Create Field F3，可以添加一个 32bit 的数据块。也可以通过工具栏 按钮添加数据块。

右键单击数据块，可以编辑数据块属性。如图 3-20 所示。



图3-20. 数据包数据块属性编辑窗口

在数据包数据块属性设置窗口，可以修改字段名称 name，在代码中读写数据包字段参数时，需要借助字段名称完成。可以修改字段类型 type，包含 integer、floatint point、structure、packet、information、integer(64bit)、packet id、object id。可以修改字段大小 size，必须为整数。可以修改默认值 default value，并选择在数据包创建时该字段是否设置默认值。可以设置字段排放顺序 conversion order 其他数据块使用相同的设置方法，其中数据块可以拖动更换位置。字段排放顺序的值从 1 开始，字段索引为字段排放顺序减 1 的值，即从 0 开始。

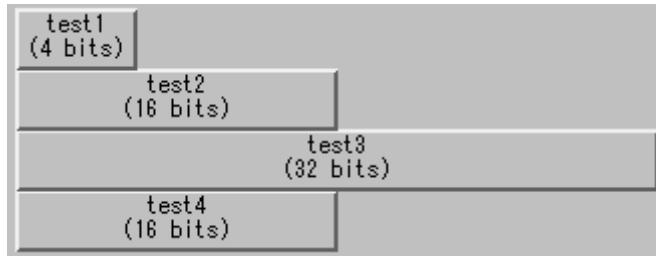


图3-21. 数据包格式示意图

点击保存按钮，将数据包文件存储，设置好文件名，文件名在创建数据包时要用到，我们这里保存成 test_pk.pk.m。由于不能保存到当前目录，我们选择保存在其他地方，再手动剪切到项目文件夹中。

3.6.2、数据包的创建

在 OPNET 中数据包只用数据包指针进行控制，Packet*，因而数据包存在创建和销毁的小队过程。在状态的出入指令中创建无格式数据包时使用代码：

```
Packet* op_pk_create (OpT_Packet_Size bulk_size);
```

创建一个无格式的数据包，大小为 bulk_size. (bulk_size 可以是 int 类型) 返回值：返回指向新创建的数据包的指针，or OPC_NIL。

例：Packet* mypacket = op_pk_create(200);

在状态的出入指令中创建自定义数据包时使用代码：

```
Packet* op_pk_create_fmt (const char* format_name);
```

新建一个先前定义好的格式数据包。返回指向新数据包的指针，失败返回 OPC_NIL

例：Packet* mypacket = op_pk_create_fmt ("packet1");

但需要注意的是，由于新的文件在创建以后需要重启 OPNET 才能被识别到，在刚定义保存过数据包文件后重启 OPNET，然后进入需要创建数据包的进程模型相应代码位置。而且有格式的数据包在创建时的默认大小就是调用的自定义数据包的大小。

3.6.3、数据包字段域

3.6.3.1、数据包字段的设置：

数据包创建之后需要对数据包内的字段进行赋值。字段赋值，可通过数据包字段索引或根据数据包字段名称两种方法。

(1) 通过字段名称找到字段区域进行赋值，函数以 op_pk_nfd_set 开头的函数，主要函数如下所示：

```
Compcode op_pk_nfd_set(Packet* pkptr, const char* fd_name, void* value);
将 pkptr 所指向的包中域名为 fd_name 域赋值为 value。
例：op_pk_nfd_set(f_pkptr, "fd_int_1", 2511);
```

对有格式数据包字段进行赋值时也可以根据字段的变量类型直接使用不同类型的赋值函数，如表 3-25 所示，分别为不同字段类型的字段设置函数，能更好的帮助我们纠正错误。

表3-25. 数据包字段名称操作函数

op_pk_nfd_set dbl (Packet* pkptr, const char* fd_name, double value);
op_pk_nfd_set_info (Packet* pkptr, const char* fd_name);
op_pk_nfd_set_int32 (Packet* pkptr, const char* fd_name, int value);
op_pk_nfd_set_int64 (Packet* pkptr, const char* fd_name, OpT_Int64 value);
op_pk_nfd_set_objid (Packet* pkptr, const char* fd_name, Objid value)
op_pk_nfd_set_pkid (Packet* pkptr, const char* fd_name, OpT_Packet_Id value);
op_pk_nfd_set_pkt (Packet* pkptr, const char* fd_name, Packet* value);
op_pk_nfd_set_ptr (Packet* pkptr, const char* fd_name, void* value, SimT_Pk_Fd_Struct_Copy_Proc copy_proc, SimT_Pk_Fd_Struct_Dealloc_Proc dealloc_proc, size_t alloc_size)

(2) 我们也可以通过字段索引找到字段区域进行赋值，这种方便相较于通过字段名称麻烦了一些，函数以 op_pk_fd_set 开头的函数，函数使用如下所示：

```
op_pk_fd_set(Packet* pkptr, int fd_index, int type, void* value, int/double/OpT_Int64
size);

作用是设定 pkptr 所指向的数据包的字段索引，字段数据类型，字段值，及大小。可用的 type 有：
OPC_FIELD_TYPE_INTEGER     、      OPC_FIELD_TYPE_INT64      、      OPC_FIELD_TYPE_DOUBLE      、
OPC_FIELD_TYPE_PACKET_ID、OPC_FIELD_TYPE_OBJECT_ID、OPC_FIELD_TYPE_PACKET。
```

或者通过不同字段类型的相应函数设置字段的值。

表3-26. 数据包字段索引操作函数

op_pk_fd_set dbl(Packet* pkptr, int field_index, double value, OpT_Int64 field_size);
op_pk_fd_set_int32(Packet* pkptr, int field_index, int value, OpT_Int64 field_size);
op_pk_fd_set_int64(Packet* pkptr, int field_index, OpT_Int64 value, OpT_Int64 field_size);
op_pk_fd_set_objid(Packet* pkptr, int field_index, Objid value, OpT_Int64 field_size);
op_pk_fd_set_pkid(Packet* pkptr, int field_index, OpT_Packet_Id value, OpT_Int64 field_size);
op_pk_fd_set_pkt(Packet* pkptr, int field_index, Packet* value, OpT_Int64 field_size);
op_pk_fd_set_ptr (Packet* pkptr, int field_index, void* value, OpT_Int64 field_size, SimT_Pk_Fd_Struct_Copy_Proc copy_proc, SimT_Pk_Fd_Struct_Dealloc_Proc dealloc_proc, size_t alloc_size);

3.6.3.2、数据包字段读取

数据包相对的字段设置的操作当然就是数据包字段的读取了。

(1) 通过**字段名称**读取指定字段的值，函数以 op_pk_nfd_get 开头的函数，包含如下基本函数：

```
op_pk_nfd_get (Packet* pkptr, const char* fd_name, void* value_ptr)
通过字段名称获取数据包某字段的值赋值给本地变量
例 op_pk_nfd_get (my_packet, "mac_address", &my_address);
//读取数据包 my_packet 中的 mac_address 字段的值到本地 my_address 变量中。
```

除此之外同样可以针对不同类型的字段使用不同的变量读取函数。

表3-27. 数据包字段名称操作函数

op_pk_nfd_get dbl (pkptr, fd_name, value_ptr);
op_pk_nfd_get_int32 (pkptr, fd_name, value_ptr);
op_pk_nfd_get_int64 (pkptr, fd_name, value_ptr);
op_pk_nfd_get_objid (pkptr, fd_name, value_ptr);
op_pk_nfd_get_pkid (pkptr, fd_name, value_ptr);
op_pk_nfd_set_pkt (pkptr, fd_name, value);
op_pk_nfd_set_ptr (pkptr, fd_name, value, copy_proc, deallocate_proc, alloc_size);

(2) 通过**字段索引**读取指定字段的值，函数以 op_pk_fd_get 开头的函数，包含如下基本函数：

```
op_pk_fd_get (Packet* pkptr, int field_index, void* value_ptr)
通过字段索引获取数据包某字段的值赋值给本地变量
例 op_pk_fd_get (pkptr, 3, &ds_ptr);
//读取数据包 pkptr 中的第 4 个字段的值到本地 ds_ptr 变量中
```

或者直接通过相应字段类型的函数。

表3-28. 数据包字段索引操作函数

op_pk_nfd_get dbl (Packet* pkptr, const char* fd_name, double* value)
op_pk_nfd_get_int32 (Packet* pkptr, const char* fd_name, int* value)
op_pk_nfd_get_int64 (Packet* pkptr, const char* fd_name, Opt_Int64* value)
op_pk_nfd_get_objid (Packet* pkptr, const char* fd_name, Objid* value)
op_pk_nfd_get_pkid (Packet* pkptr, const char* fd_name, Opt_Packet_Id* value)
op_pk_nfd_get_pkt (Packet* pkptr, const char* fd_name, Packet** value)
op_pk_nfd_get_ptr (Packet* pkptr, const char* fd_name, void** value_ptr)

3.6.3.3、字段域操作

除了对数据包字段的设置与读取外，OPNET 还提供了其他对数据包字段的操作，主要包括查询有无，查询类型，设置大小等。同样字段域的其他操作同样包含使用**字段索引**和使用**字段名称**两种方法

(1) 使用**字段名称**可以进行的操作主要包括如下函数：

表3-29. 数据包字段名称常用函数

double op_pk_nfd_default (Packet* pkptr, const char* fd_name);	获取一个字段的默认值。当返回值不是 double 时，可进行强制类型转换
例: def_opcode = (int) op_pk_nfd_default (pkptr, "opcode");	
OpT_Boolean op_pk_nfd_exists (Packet* pkptr, const char* fd_name);	判断某字段是否存在
Boolean op_pk_nfd_is_set (Packet* pkptr, const char* fd_name);	判断某字段是否被设置了。数据包字段块的属性 set at creation 中有相关设置
int op_pk_nfd_name_to_index (Packet* pkptr, const char* fd_name);	获取某字段的字段索引
OpT_Packet_Size op_pk_nfd_size (Packet* pkptr, const char* fd_name);	获取某字段的内存大小
op_pk_nfd_size_set (Packet* pkptr, const char* fd_name, OpT_Packet_Size size);	设置某字段的内存大小
op_pk_nfd_strip (Packet* pkptr, const char* fd_name);	从一个数据包中删除某字段。
int op_pk_nfd_type (Packet* pkptr, const char* fd_name);	获取某字段的类型
int op_pk_name_to_index (const char * format_name, const char * field_name);	获取某种数据包中某字段的字段索引。

(2) 使用**字段索引**可以进行的操作主要包括如下函数：

表3-30. 数据包字段索引常用函数

op_pk_fd_index_to_name (Packet* pkptr, int fd_index, char* fd_name);	获取指定索引字段的字段名称。
Boolean op_pk_fd_is_set (Packet* pkptr, int fd_index);	判断某字段是否被设置了。数据包字段块的属性 set at creation 中有相关设置。
int op_pk_fd_max_index (Packet* pkptr);	获取数据包的最大字段索引值
OpT_Packet_Size op_pk_fd_size (Packet* pkptr, int fd_index);	获取指定索引字段的字段内存大小。
op_pk_fd_strip (Packet* pkptr, int fd_index);	删除数据包指定索引的字段
int op_pk_fd_type (Packet* pkptr, int fd_index);	获取数据包指定索引的字段的字段类型

(3) 其他关于字段域的操作。

表3-31. 数据包其他字段域操作函数

int op_pk_num_fds (Packet* pkptr);	获得数据包中字段的个数
op_pk_num_fields_get (Packet* pkptr, int* num_fmd_fields, int* num_unfmd_fields);	获得数据包有格式和无格式字段的个数分别存储在 num_fmd_fields 和 num_unfmd_fields

中。

3.6.4、数据包属性域

数据包属性域中包含的数据包属性方便使用者在调试中掌握数据包运行情况，在管道模型阶段也会被用到来设置数据包的状态。数据包属性包含如表 3-32 所示的内容。

表3-32. 数据包属性域

数据包属性	取值类型	取值示例
ID	整型	3381
tree ID	整型	351
Address	整型	0x04389778
Format	字符串	log_in_ack
creation module	整型	top. CampusNetwork. satellite1. control_channel [Objid=3625]
creation time	双浮点型	1. 670743947487sec. [1s . 670ms 743us 947ns 487ps]
stamp module	整型	top. Campus Network. satellite1. control_channel [Objid=3625]
stamp time	双浮点型	1. 670743947487 sec. [1s . 670ms 743us 947ns 487ps]
bulk size	整型	0 bits
total size	整型	56 bits
owner	整型	top. Campus Network. node44. control_channel [Objid=1616]
ICI ID	整型	NONE
ID trace	布尔型	off
tree ID trace	布尔型	off
encap flags		NONE

(1) ID:

数据包属性中的 ID 为当前数据包的 ID，同一个内存块的数据包的 ID 相同，即同一个数据包在不同模块间传输时包 ID 不变，但是当包具有了新的内存块，则就会具有新的包 ID。所以数据包 ID 唯一标志数据包内存。如图 3-22 所示，在节点内一个数据包传输时，为同一内存块，即始终是同一个包，在一个无线发信机向多个发信机发送时，除了将原始的数据包发给其中一个接收机外，会为其他每个接收器开辟新的内存，复制副本，因为其他数据包不是原始数据包，虽然内容相同，因而包 ID 也就跟着更新了。在动画显示时，区分数据包也是通过包 ID 实现的。如图 3-22 所示。

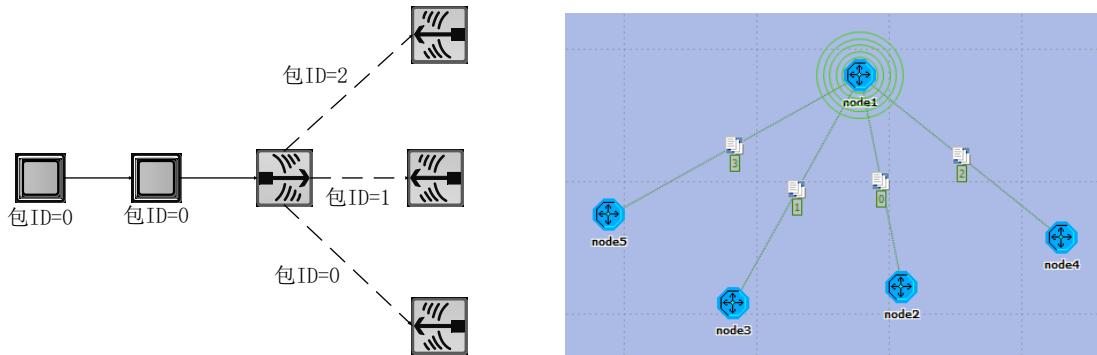


图3-22. 数据包 ID 变化示意图

数据包 ID 的设置是仿真核心来实现的。操作函数：

<code>OpT_Packet_Id op_pk_id (Packet* pkptr);</code>
来获取某数据包的 ID
<code>op_pk_id_str_get (OpT_Packet_Id pkid, char* buffer);</code>
通过函数根据数据包 ID 获取数据包的描述

(2) Format:

网络协议的运作流程通过包交互实现，传递信息在不同类型的包中，所以模块在接收到包后，总是先判断包的类型，再根据不同类型的包格式获取包中字段。操作函数：

<code>op_pk_format (Packet* pkptr, char* fmt_name);</code>
获取数据包的格式

(3) creation module:

每有一个新的数据包内存被建立，数据包的 ID 就会变动一次，但是数据内容不变，所以当进行包跟踪时，往往需要获知产生数据包的模块。操作函数：

<code>op_pk_creation_mod_set (Packet* pkptr, Objid objid);</code>
重写产生数据包的模块 ID
<code>Objid op_pk_creation_mod_get (Packet* pkptr);</code>
通过产生数据包的模块 ID

(4) creation time:

在网络仿真时，需要仿真网络端到端时延性能，这是就要获取数据包的产生时间。需要注意的是，在有重传的协议中，数据包发送需要发送副本，副本的数据包产生时间为复制的时间。操作函数：

<code>op_pk_creation_time_set (Packet* pkptr, double time);</code>
重写数据包的产生时间
<code>double op_pk_creation_time_get (Packet* pkptr);</code>
获取数据包的产生时间

(5) stamp module:

将数据包坐上标记以在后续的操作中获取此标记的相关信息。数据包标记的操作函数。

<code>op_pk_stamp (Packet* pkptr);</code>
标记数据包。会自动设置数据包的标记模块和标记时间
<code>Objid op_pk_stamp_mod_get (Packet* pkptr);</code>
获取标记数据包的模块
<code>op_pk_stamp_mod_set (Packet* pkptr, Objid objid);</code>
设置某数据包的标记模块

(6) stamp time:

数据包标记时间的操作函数如下：

```
op_pk_stamp_time_set (Packet* pkptr, double time);
```

设置数据包的标记时间属性

```
double op_pk_stamp_time_get (Packet* pkptr);
```

获取数据包的标记时间属性

(7) bulk size:

数据包的大小属性中包含批量数据大小属性和总大小属性两个属性。批量数据是总数据的一部分。

```
op_pk_bulk_size_set (Packet* pkptr, OpT_Packet_Size bulk_size);
```

设置数据包批量数据的大小，以比特为单位。批量数据只是数据包的一部分数据

```
OpT_Packet_Size op_pk_bulk_size_get (Packet* pkptr);
```

获取数据包批量数据的大小，以比特为单位。批量数据只是数据包的一部分数据

(8) total size:

数据包的大小，即数据包的总大小是影响传输的关键参数，在计算传输时延时，使用的是总大小的属性。操作函数如下：

```
op_pk_total_size_set (Packet* pkptr, OpT_Packet_Size total_size);
```

设置数据包总的大小，以比特为单位。

```
OpT_Packet_Size op_pk_total_size_get (Packet* pkptr);
```

获取数据包总数据的大小，以比特为单位。

(9) owner:

数据包的创建者只有一个，但是数据包的所有者却随着数据包的流动而变化。数据包的拥有者每经过一个模块也就更新一次。owner 属性即为当前获取数据包的模块 ID。

(10) total size:

数据包的大小，即数据包的总大小是影响传输的关键参数，在计算传输时延时，使用的是总大小的属性。操作函数如下：

```
op_pk_total_size_set (Packet* pkptr, OpT_Packet_Size total_size);
```

设置数据包总的大小，以比特为单位。

```
OpT_Packet_Size op_pk_total_size_get (Packet* pkptr);
```

获取数据包总数据的大小，以比特为单位。

(11) ICI:

数据包可通过 ICI 传递信息，不占用字段内存，但同样可以传递信息。可参见 3.5.3 ICI 章节，操作函数如表

```
op_pk_ici_set (Packet* pkptr, Ici* iciptr);
```

设置数据包的 ICI

```
Ici* op_pk_ici_get (Packet* pkptr);
```

获取数据包的 ICI。

(12) encapsulation:

在多层网络协议中，至上而下逐层封装。encap 的属性用于标记封装了其他包的包，用于捕获数据包后的检测。

```
Boolean op_pk_encap_flag_is_set (Packet* pkptr, int index);
```

检测封装包的标志位是否被设置了，index 取值为 0-31。

```
op_pk_encap_flag_set (Packet* pkptr, int index);
```

为包设置一个标志位，使得封装了装个包的所有数据包对这个标志位均可见。

```

op_pk_encap_pk_access (Packet* pkptr, const char* format, Packet** encaps_pk_pptr);
从一个封装了其他包的包中读取指定格式的原始包

op_pk_encap_pk_get (Packet* pkptr, const char* format, Packet** encaps_pk_pptr);
从一个封装了其他包的包中复制指定格式的原始包

```

3.6.5、数据包传输

数据包在发送后，原始模块就无法再通过指针获取数据包的内容了，也就没有办法设置、读取、发送之前的数据包了。在点对多点的包传输中，源模块会复制 $n-1$ 个副本，其中 n 为目的模块的个数。将源数据包和 $n-1$ 个副本，发送给 n 个目的模块，每个模块处理各自的数据包。数据包复制以后，生成新的包内存，就会具有新的数据包 ID。无线链路数据包流通如图 3-23 所示。

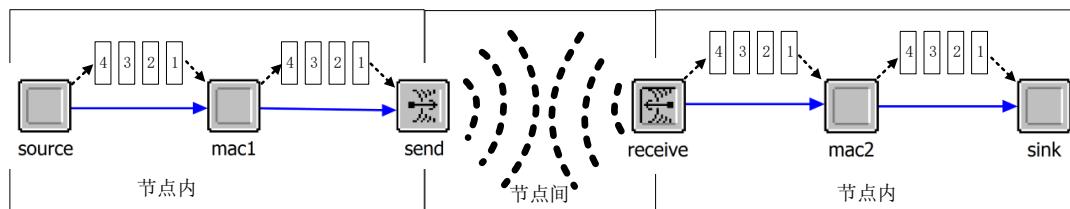


图3-23. 无线通信包交互示意图

(1) 数据包在节点内传输

如果包流的源模块是进程模块，则可以通过 `op_pk_send (Packet* pkptr, int outstrm_index)` 及其演变的 3 种方式将包发送至目的模块输入流。

(1) 常用的发送方式是调用 `op_pk_send()`，当包沿着源模块输出流到达目的模块输入流时立即向目的模块触发流中断。整个过程时延由包流的“delay”属性指定，所以包到达的时刻为包发送的时刻加上包流“delay”属性的值。

(2) 与第一种方式相比，如果要模拟包在包流传输过程的额外延时，以此来仿真模块有限的处理速度，这时可以调用 `op_pk_send_delayed()` 函数，包将滞后指定的时间达到目的模块。

(3) `op_pk_send_forced()` 产生的事件不需要在仿真核心的事件列表中排队，而是插队到事件列表的队首立刻执行，并且包不需要经历从源模块输出流到目的模块输入流的延时，直接到达目的模块。

前面 3 种传输方式对于目的模块来说是被动的，因为包的到达会强加一个流中断通知它接收。如果目的模块希望隔一定的时间间隔主动地去从队列中取出一个包，此时包到达引起的时间上不规则的中断显得无意义。考虑到目的模块的这种要求，源模块应该调用 `op_pk_send_quiet()` 函数，采取一种静默的方式发送包。

为了支持以上各种包传输模式，还必须设置相应的包流“中断模式”(intrpt mode) 属性，它有三种可选值，分别是 scheduled、forced 和 quiet。选择 scheduled 对应采用 `op_pk_send()` 和 `op_pk_send_delayed()` 传输包，这时可以设置包流的“delay”属性，如图所示。

gen_0 [0] -> dp [0]	
Attribute	Value
name	strm_0
src stream	src stream [0]
dest stream	dest stream [0]
intrpt method	scheduled
delay	0.0
color	RGB003

图3-24. 包流属性设置

选择 forced 对应采用 op_pk_send_forced() 传输包；选择 quiet 对应采用 op_pk_send_quiet() 传输包。

数据包在节点内模块间的传输是通过包流线实现。数据包在节点内模块间传输时，需要在发送时指定包流输出索引。一条发送语句只能发送一个数据包往一条包流上。源模块在发送了数据包指针指向的数据包以后就无法再通过指针访问数据包内容或者再发送一次了。但是仿真设计中往往需要重传机制，所以仿真中可以设计先复制一份数据包，发送时发送副本，等接收到 ACK 回应或者其他方式确认数据包被正确接收时，再销毁原始数据包。

包流线可以设置数据包传输时延和流中断方式（强制，调度，静默），强制方式发送数据包会在目的模块触发一个优先级别高流中断，此中断会排在相同时间事件列表的前面执行。调度方式发送数据包会触发一个优先级别低的流中断，此中断会排在相同事件列表的后面执行。静默方式发送数据包，目的模块不会触发流中断。详解参照 3.5.2 中断章节

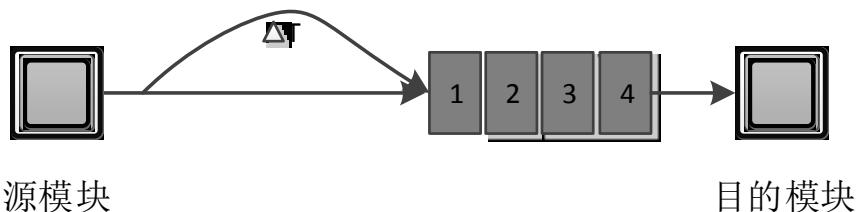


图3-25. 包流属性设置

源模块在发送数据包以后，数据包会缓存到目的模块中，并根据不同的流中断方式，触发目的模块产生流中断。目的模块通过代码主动去捕获缓存在本地的数据包。

由于向目的模块缓存数据包时是缓存到数据包列表的尾部，而从缓存中捕获数据包时是捕获的数据包列表的头部。所以当缓存中存在不仅一个缓存数据包时，发送的数据包和捕获的数据包并不一定是同一个数据包。

目的模块在接收到流中断后，应先获取产生当前流中断的包流源索引 op_intrpt_strm()，再通过 Packet* pkptr = op_pk_get (包流索引) 读取缓存中的数据包；

数据包在从源模块开始发送到完全缓存到目的模块中需要的时间根据包流属性中的 delay 设置，默认为 0 不需要损耗时间。同样目的模块从缓存中读取数据包到完全读取至目的模块也是不需要消耗时间的。所以当多个源模块同时向一个模块发送数据包，目的模块可以同时接收到。

(1) 处理器（队列）模块间循环发包的步骤。

由于默认情况下源模块到目的模块间的发包时延为 0。则在连续发包时不同的发包方式会产生不同的中断结果，参照 3.5.2.6 流中断章节

(2) 模块发包速度大于发信机速度

当发信机接收处理器模块发来的数据包频率大于发送数据包到其他节点的频率时，数据包会被暂时缓存在发信机模块缓存中。发信机会先捕获缓存中的一个数据包发送，发送完成后，自动捕获缓存中的下一个数据包发送。当缓存中不再存在数据包则不再发送。所以发信机不会按照接收内部数据包的速度发送数据包。接收机则按照发信机的发送速率接收到数据包。接收机接收完成第一个数据包后，同时开始接收第二个数据包。

(2) 数据包在节点间传输

数据包在节点间传输时通过收发信机和通信链路完成。数据包在移动节点或卫星节点间传输时是通过无线发信机，经过发信机物理层管道模型，经过收信机物理层管道模型，最后经过无线收信机到达移动接收节点。数据包使用有线收发信机传输时，先通过有线发信机，然后通过链路模型和有线链路管道模型，到达目的节点的有线接收机，被有线接收机接收后数据包成功到达目的节点。同样数据包在发送之后，就不能在访问，也不能再销毁或再发送了。请参照 2.6 章节链路和管道模型

在数据包接收中都是先发起流中断，然后模块主动获取的方式。在数据包接收中存在接收冲突的问题。当接收机在接收机忙时，又接收到一个数据包，则只能接收到后来的数据包（但实际中应该两个数据包都不能接收到）。当接收机先接到一个不匹配的数据包立刻结束。不影响紧随其后的正常包的接收。数据包的接收参考 3.5.2 中断章节。

(3) 数据包在节点间传递

包流只支持包在“同一节点模型”中不同模块间的包传输。在某些情况下，要求包能够在节点模型之间直接传输而又不希望将这些节点模型通过链路连接（即节点间没有物理连接），这时可以用到“包传递”的方法，如图所示。

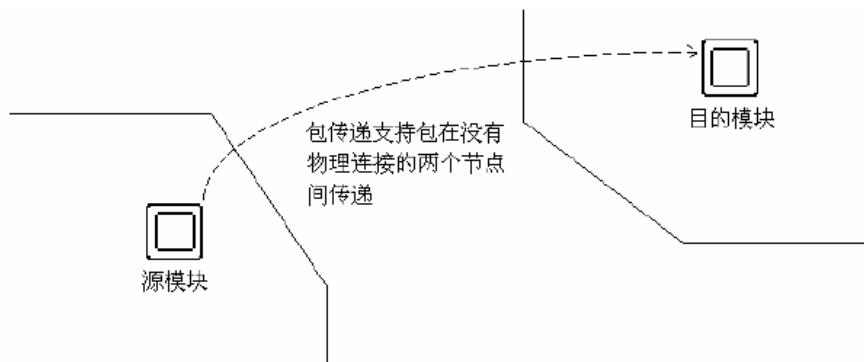


图3-26. 数据包节点间传递

与包发送的四种方式 `op_pk_send()`、`op_pk_send_delayed()`、`op_pk_send_forced()` 和 `op_pk_send_quiet()` 相对应，包传递也有四种方式，分别是 `op_pk_deliver()`、`op_pk_deliver_delayed()`、`op_pk_deliver_forced()` 和 `op_pk_deliver_quiet()`，但是与包发送不同的是包传递需要指定目的模块的 Objid。由于没有包流的参与，包传递没有指向目的模块的依据，所以只能通过指定 Objid 的方法来定位目的模块。

3.6.6、数据包其他操作

(1) 数据包复制

函数	Packet* op_pk_copy (Packet* pkptr);
备注	复制数据包会复制所有字段的信息，所以在销毁时也要各自销毁原本和副本。但是属性域和 TDA 的数据并不复制

(2) 数据包销毁

函数	Void op_pk_destroy (Packet* pkptr);
备注	销毁包，释放内存空间。

(3) 获取数据包类型

函数	int op_pk_type (Packet* pk_ptr);
备注	这里的类型，不是包格式。返回值为 OPC_PACKET_TYPE_FORMATTED（有格式）、OPC_PACKET_TYPE_UNFORMATTED（无格式）、OPC_PACKET_TYPE_VVEC（值向量）

(4) 设置包优先级

函数	void op_pk_priority_set (Packet* pkptr, double value); double op_pk_priority_get (Packet* pkptr);
备注	默认为不存在优先级，优先级属性的值为 OPC_PRIO_NOT_SET

(5) 包转移

函数	op_pk_transfer (Packet* from_pkptr, Packet* to_pkptr);
备注	将数据包的内容从一个数据包转移到另一个

3.6.7、TDA

数据包中除了字段域和属性与还有 TDA 域，在每个数据包中设置了很多 TDA 属性，主要用来在管道阶段进行信道模拟和数据包处理。不同链路类型有不同数量和种类的 OPNET 自带 TDA，相同名称的 TDA 属性在不同链路类型中的值和含义也不尽相同。所以推荐使用 OPNET 中定义的 TDA 常量来读写数据包 TDA 属性。无线链路 TDA 参数以 OPC_TDA_RA_开头，点到点链路以 OPC_TDA_PT_开头，总线链路以 OPC_TDA_BU_开头。

表3-33. 无线链路 TDA 参数

名称	类型	含义	存取限制
OPC_TDA_RA_ACTUAL_BER	double	数据包或其中一段的误比特概率	误码分配阶段可读写
OPC_TDA_RA_BER	double	数据包或其中一段的误比特概率	误比特阶段可读写
OPC_TDA_RA_BKGNOISE	double	其他未建模信源产生的背景噪声 (W)	背景噪声阶段可读写
OPC_TDA_RA_CLOSURE	integer	收发信机间的可达性	链路闭合阶段可读写
OPC_TDA_RA_ECC_THRESH	double	纠错容许的数据包中的最大误比特比例	核心设定，各阶段可读写
OPC_TDA_RA_END_DIST	double	传送结束时收发信机间的距离 (m)	核心设定，各阶段可

			读写
OPC_TDA_RA_END_PROPDEL	double	发信机发射结束时无线信号在收发信机传递所需的时间	传播时延阶段可读写
OPC_TDA_RA_END_RX	double	数据包接收完毕的仿真时间 (=START_TX+TX_DELAY+END_PROPDELAY)	所有阶段可读写
OPC_TDA_RA_END_TX	double	发信机完成数据包发送的仿真时间	所有阶段可读写
OPC_TDA_RA_MATCH_STATUS	integer	收发信机间的兼容情况,有合法、干扰、忽略三种	信道匹配阶段和读写
OPC_TDA_RA_ND_FAIL	double	接收过程中接收节点发生失效的最早时间	各阶段可读写
OPC_TDA_RA_ND_RECov	double	接收过程中节点恢复的最晚时间	各阶段可读写
OPC_TDA_RA_NOISE_ACCUM	double	数据包或其中一段上累积的噪声功率	干扰噪声阶段可读写
OPC_TDA_RA_NUM_COLLs	integer	数据包累积遭遇的冲突数目	冲突阶段可读写
OPC_TDA_RA_NUM_ERRORS	integer	数据包中误比特的数目	误码分配阶段可读写
OPC_TDA_RA_PK_ACCEPT	integer	是否接收数据包的判决	纠错阶段可读写
OPC_TDA_RA_PROC_GAIN	double	收信机处理增益(dB),用以计算信噪比	核心设定,各阶段可读写
OPC_TDA_RA_RCVD_POWER	double	收信机收到信号的带内功率(W)	接收信号功率阶段可读写
OPC_TDA_RA_RX_BORESIGHT_PHI	double	发信机天线的参考点(度)	
OPC_TDA_RA_RX_BORESIGHT_THETA			
OPC_TDA_RA_RX_BW	double	收信机信道带宽	核心设定,各阶段可读写
OPC_TDA_RA_RX_CH_INDEX	integer	收信机信道索引	核心设定,各阶段可读写
OPC_TDA_RA_RX_CH_OBJID	integer	收信机信道对象 ID	核心设定,各阶段可读写
OPC_TDA_RA_RX_CODE	double	收信机使用的扩频码或扩频序列	核心设定,各阶段可读写
OPC_TDA_RA_RX_DRATE	double	收信机信道的数据速率(bps)	核心设定,各阶段可读写
OPC_TDA_RA_RX_FREQ	double	收信机信道频率下界	核心设定,各阶段可读写
OPC_TDA_RA_RX_GAIN	double	收信机天线提供的增益	收信机天线增益阶段可读写
OPC_TDA_RA_RX_GEO_X	double	接收机节点的球面坐标(m)	
OPC_TDA_RA_RX_GEO_Y	double		
OPC_TDA_RA_RX_GEO_Z	double		
OPC_TDA_RA_RX_LAT	double	接收机节点的球心坐标	
OPC_TDA_RA_RX_LONG	double		

OPC_TDA_RA_RX_ALT	double		
OPC_TDA_RA_RX_MOD	pointer	收信机调制曲线表对象的指针，用作 op_tbl_mod_ber() 的参数	
OPC_TDA_RA_RX_NOISEFIG	double	收信机的噪声系数	
OPC_TDA_RA_RX_OBJID	integer	收信机模块的对象 ID	各阶段可读写
OPC_TDA_RA_RX_PATTERN	pointer	天线方向图的指针，用做 op_tbl_pat_gain() 的参数	
OPC_TDA_RA_RX_PHI_POINT	double	接收天线“target”属性决定的指向角（度）	
OPC_TDA_RA_RX_THETA_POINT	double		
OPC_TDA_RA_RX_REL_X	double	接收节点在所在子网坐标系中的相对位置	
OPC_TDA_RA_RX_REL_Y	double		
OPC_TDA_RA_SNR	double	接收机处理数据包或其中一段的信噪比 (dB)	信噪比阶段可读写
OPC_TDA_RA_SNR_CALC_TIME	double	上次计算 SNR 的仿真时间	
OPC_TDA_RA_START_DIST	double	传输开始时收发信机间的距离 (m)	
OPC_TDA_RA_START_PROPDEL	double	开始传输时无线信号在收发信机传播的时延 (s)	传播时延阶段可读写
OPC_TDA_RA_START_RX	double	数据包接收开始的仿真时间	所有阶段可读写
OPC_TDA_RA_TX_BORESIGHT_PHI	double	发信机天线的参考点（度）	
OPC_TDA_RA_TX_BORESIGHT_THETA			
OPC_TDA_RA_TX_BW	double	发信机信道的带宽 (Hz)	
OPC_TDA_RA_TX_CH_INDEX	integer	发信机信道索引	所有阶段可读写
OPC_TDA_RA_TX_OBJID	integer	发信机信道的对象 ID	所有阶段可读写
OPC_TDA_RA_TX_CODE	double	发信机使用的扩频码或扩频序列	
OPC_TDA_RA_TX_DELAY	double	数据包传输时延	传输时延阶段可读写
OPC_TDA_RA_TX_DRATE	double	发信机信道的数据速率 (bps)	
OPC_TDA_RA_TX_FREQ	double	发信机信道的频率下界 (Hz)	
OPC_TDA_RA_TX_GAIN	double	发信机信道提供的增益	发信机天线增益
OPC_TDA_RA_TX_GEO_X	double	发信机节点的球面坐标 (m)	
OPC_TDA_RA_TX_GEO_Y	double		
OPC_TDA_RA_TX_GEO_Z	double		
OPC_TDA_RA_TX_LAT	double	发信机节点的球心坐标 (度)	
OPC_TDA_RA_TX_LONG	double		
OPC_TDA_RA_TX_ALT	double		
OPC_TDA_RA_TX_MOD	pointer	发信机调制曲线表对象的指针，常用于 op_tbl_mod_ber() 的参数	
OPC_TDA_RA_TX_OBJID	integer	发信机模块的对象 ID	所有阶段可读写
OPC_TDA_RA_TX_PATTERN	pointer	收信机天线方向图对象的指针，常用于 op_tbl_pat_gain() 的参数	
OPC_TDA_RA_TX_PHI_POINT	double	发射天线“target”属性决定的指向角（度）	
OPC_TDA_RA_TX_THETA_POINT	double		
OPC_TDA_RA_TX_POWER	double	发射功率 (w)	

OPC_TDA_RA_TX_REL_X	double	接收节点在所在子网坐标系中的相对位置	
OPC_TDA_RA_TX_REL_Y	double		

表3-34. 点对点链路 TDA 参数

OPC_TDA_PT_RX_OBJID	integer	发信机模块的对象 ID	所有阶段可读写
OPC_TDA_PT_RX_OBJID	integer	收信机模块的对象 ID	
OPC_TDA_PT_LINK_OBJID	integer	链路的对象 ID	
OPC_TDA_PT_TX_CH_OBJID	integer	发信机信道的对象 ID	
OPC_TDA_PT_CH_INDEX	integer	发信机信道的对象 ID	
OPC_TDA_PT_ND_FAIL	double	接收过程中接收节点发生失效的最早时间	
OPC_TDA_PT_ND_RECV	double	接收过程中接收节点恢复的最晚时间	
OPC_TDA_PT_TX_DELAY	double	数据包的传输时延	
OPC_TDA_PT_PROP_DELAY	double	数据包的传播时延	
OPC_TDA_PT_NUM_ERRORS	integer	数据包中误比特的数目	
OPC_TDA_PT_PK_ACCEPT	integer	是否接收数据包的判决	纠错阶段可读写

表3-35. 总线链路 TDA 参数

OPC_TDA_BU_RX_OBJID	integer	接收机模块的对象 ID	各阶段只写
OPC_TDA_BU_RX_OBJID	integer	接收机模块的对象 ID	
OPC_TDA_BU_LINK_OBJID	integer	链路的对象 ID	
OPC_TDA_BU_TX_CH_OBJID	integer	发信机信道的对象 ID	
OPC_TDA_BU_CH_INDEX	integer	收发信机的信道编号	
OPC_TDA_BU_ND_RECV	double	接收过程中接收节点恢复的最晚时间	
OPC_TDA_BU_TX_DELAY	double	数据包传输时延	
OPC_TDA_BU_PROP_DELAY	double	数据包传播时延	
OPC_TDA_BU_NUM_ERRORS	integer	数据包中误比特的数目	
OPC_TDA_BU_PK_ACCEPT	integer	是否接收数据包的判决	
OPC_TDA_BU_RX_TAP_OBJID	integer	发信机与总线间的连接头对象 ID	各阶段可读写
OPC_TDA_BU_RX_TAP_OBJID	integer	收信机与总线间的连接头对象 ID	
OPC_TDA_BU_DISTANCE	double	收发信机间的距离 (m)	
OPC_TDA_BU_END_RX	double	接收完成的时间	
OPC_TDA_BU_NUM_COLL	integer	数据包累积遭遇的冲突次数	冲突阶段可读写

OPC_TDA_BU_CLOSURE	integer	收发信机间的可达性	链路闭合阶段可读写
--------------------	---------	-----------	-----------

3.7、统计量

统计量主要分为局部统计量和全局统计量。局部统计量必须与特定对象相关，对于这种统计量必须制定具体对象。支持局部统计量的对象可以时是节点、链路、模块。全局统计量提供整体性能的相关信息。

在使用局部或全局统计量时，并没有限制使用者使用统计量收集什么内容。所以全局统计量在写入时使用代码控制，同样可以作为局部统计量使用。

统计量是数据源，探针是收集统计量的方法，统计量可以用作探针和统计线的源。统计量的使用包含统计量的设定，统计句柄的定义，统计量的注册，数据收集，数据显示。

3.7.1、统计量的定义

(1) 全局统计量：

在需要收集统计量的进程模型界面，选择菜单栏“Interfaces”按钮，选择 Global Statistics 快捷项，进入全局统计量编辑窗口。

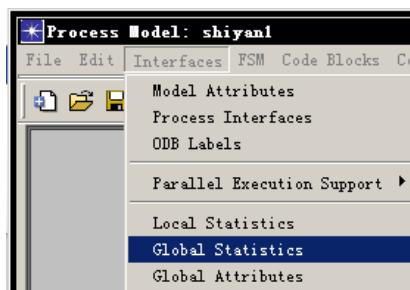


图3-27. 全局统计量进入方法

全局统计量编辑窗口如下图 3-28 所示

Stat Name	Mode	Count	Description	Group	Capture Mode	Draw Style	Low Bound	High Bound
test1(global)	Single	N/A		group1	bucket/default total/sample mean	linear	0.0	disabled
test2(global)	Single	N/A		group1	bucket/default total/sum_time	linear	0.0	disabled
test3(global)	Single	N/A		group1	bucket/default total/sum	linear	0.0	disabled
test4(global)	Single	N/A		group1	bucket/default total/time average	linear	0.0	disabled
test5(global)	Single	N/A		group1	bucket/default total/max value	linear	0.0	disabled
test6(global)	Single	N/A		group1	bucket/default total/min value	linear	0.0	disabled
test7(global)	Single	N/A		group1	bucket/default total/count	linear	0.0	disabled
test1(global)	Single	N/A		group2	bucket/default total/sum_time	discrete	0.0	disabled
test2(global)	Single	N/A		group2	bucket/default total/sum_time	sample-hold	0.0	disabled
test3(global)	Single	N/A		group2	bucket/default total/sum_time	bar	0.0	disabled
test4(global)	Single	N/A		group2	bucket/default total/sum_time	bar chart	0.0	disabled
test5(global)	Single	N/A		group2	bucket/default total/sum_time	square-wave	0.0	disabled

图3-28. 全局统计量声明窗口

(2) 局部统计量：

在需要收集统计量的进程模型界面，选择菜单栏“Interfaces”按钮，选择 Local Statistics 快捷项，进入局部统计量编辑窗口，如图 3-27 选择 Local Statistics。局部统计量与全局统计量声明相同，编辑窗口相同。只是定义的统计量作用范围不同。

在统计量声明窗口中，各字段的用途如下所述。

- Stat Name: 统计量名称，在显示时看到的记录名称
- Mode: 指示统计量是否是多维的。Single 为单维，Dimensioned 为多维。单维统计量至产生一系列以时间为横轴的记录。在显示时，横坐标为时间，纵坐标为统计量记录。多维统计量即多个单维统计量，各维数据之间相互独立，都可以独立的为探针或统计线用做源。多维统计量中的各维具有相同的统计量名称，但通过数组下标区分。这一机制对管理大量相似的统计量很有用。当需要统计若干组相同的统计量时，只需要声明一个多维统计量，这比分别声明多个单维统计量方便得多，当需要增加统计量时，只要修改维数的定义即可，进程模型也会因此获得更好的可扩展性。
- Count: 指示多维统计量的维数，单维统计量时用 N/A
- Description: 关于统计量的简要描述。
- Group: 指示统计量的统计分组。用于更好的管理统计量。
- Capture Mode: 指示统计量的默认捕获模式。
- Draw Style: 指示统计量的默认绘制形式。
- Low Bound: 统计量显示时坐标系的参考下界。如果收集到的数据没有低于该值，则图形按该值设定坐标，否则按实际情况设定。
- High Bound: 统计量显示时坐标系的参考上届。如果收集到的数据没有高于该值，则图形按该值设定坐标，否则按实际情况设定。

3.7.2、统计量的捕获模型 Capture Mode

点击统计量的捕获模型编辑标签，弹出捕获模型编辑窗口。



图3-29. 统计量捕获模式设置

图3-30. 统计量捕获模式中各参数含义

模式	作用
----	----

all values	不加更改的记录所有统计值
sample	只记录特定的统计值，忽略其他统计值，支持两个子模式，1 获取每隔 n 个的值 2 获取每隔 T 秒的值。
bucket	默认模式，将一系列值看做一个“桶”，并记录一个代表桶的值，可以通过时间间隔或连续的统计值个数划分桶。通过下列函数计算桶内的值以得到代表桶的值：和/时间、最小值、最大值、采样平均、时间平均以及计数。
Glitch removal	当同一时间有多个统计值时，只记录最后一次更新。

为了能对收集到的数据做处理并显示在坐标系中，我们使用默认的 bucket 模式

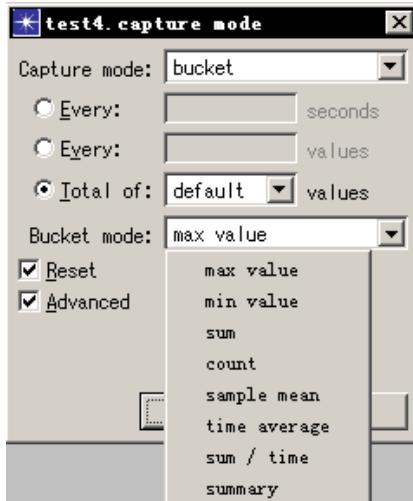


图3-31. 统计量捕获模式设置

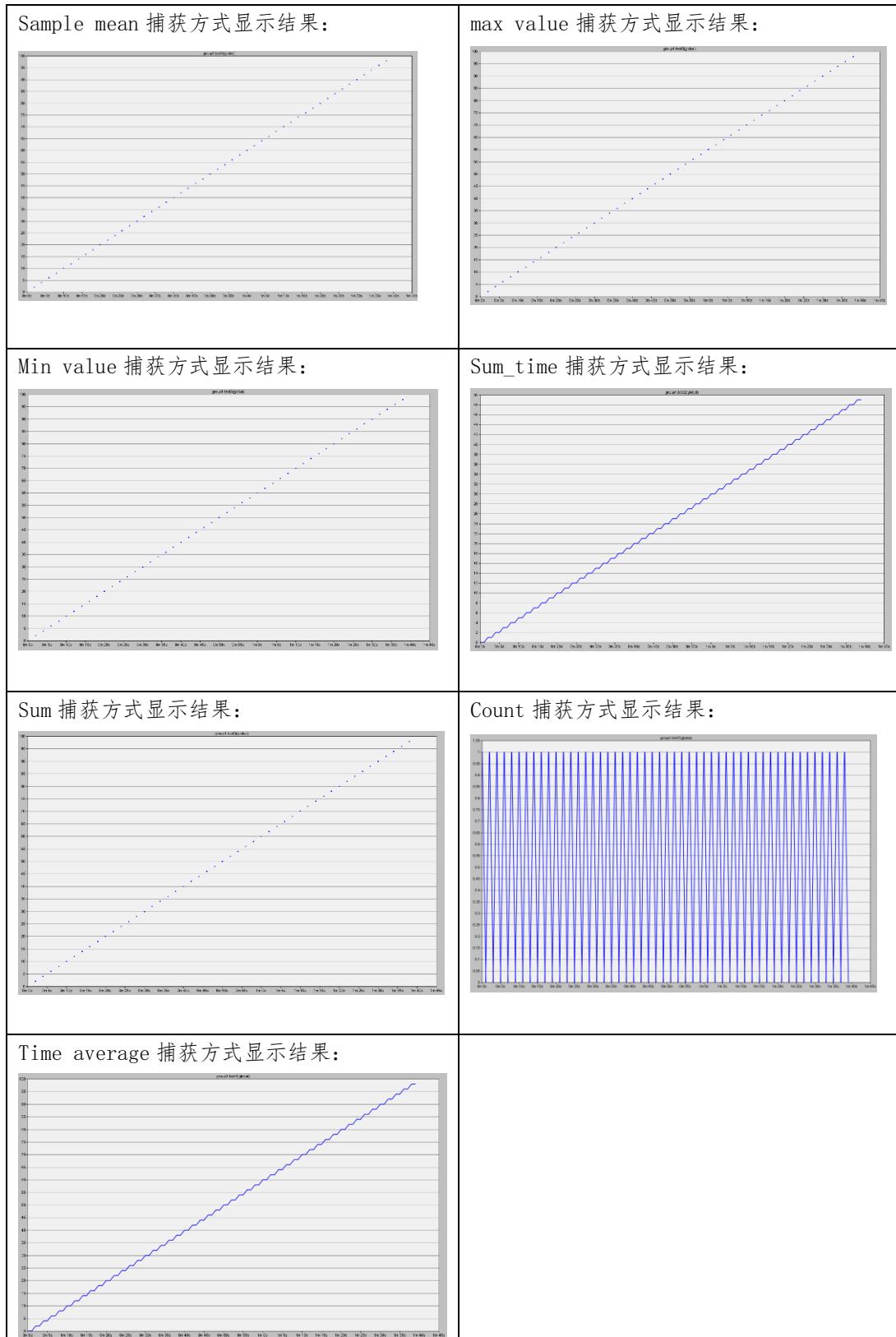
在介绍 Bucket mode 之前先将介绍一下“计算时间窗”。在写入统计量时可能时随机时间也可能时周期性的。仿真时长是用户根据自己的需求设定的。仿真时长设定了坐标系的时间横轴就设定了。系统会自动确定在横轴上显示多少个统计点，则每个统计点之间的时间长度就成为“计算时间窗”，比如设定仿真时间为 100s。统计量收集周期在 0.1s 附近。系统为了仿真结果的显示，在 100s 的时间横轴上要显示 50 个点。则“计算时间窗”长度为 2s。在每个 2s 内收集的所有统计量做函数运算，获取的一个代表值，作为当前 2s 的值显示在纵坐标上。所以 Bucket mode 中的函数，都是针对一个“计算时间窗”内的统计量的计算结果。而这个时间窗长度又是不定长的。所以相同的仿真数据，仿真时间长度不同时，显示的结果就不同。使用 sum/time 时由于对时间求了均值，所以不受时间窗长度的影响。

表3-36. Bucket 模式各参数含义

函数名	功能
max value	计算时间窗内所有统计量的最大值
min value	计算时间窗内所有统计量的最小值
sum	计算时间窗内所有统计量的和
count	计算时间窗内所有统计量的个数
Sample mean	计算时间窗内所有统计量的样本均值
Time average	计算时间窗内所有统计量的时间均值
Sum / time	计算时间窗内所有统计量与时间窗长度的比值
summary	

示例，分别采用每种捕获方式每隔 2.0s 记录一次当前仿真时间。共仿真 20s 记录数据如图

记录时间	0.0s	2.0s	4.0s	6.0s	8.0s	10.0s	...
记录值	0.0	2.0	4.0	6.0	8.0	10.0	...



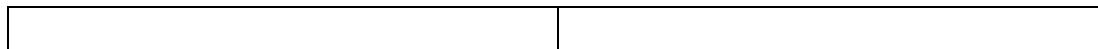


图3-32. 统计量各种捕获模式下示意图

3.7.3、绘制风格 Draw style

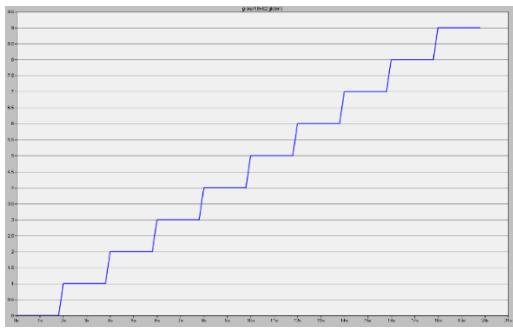
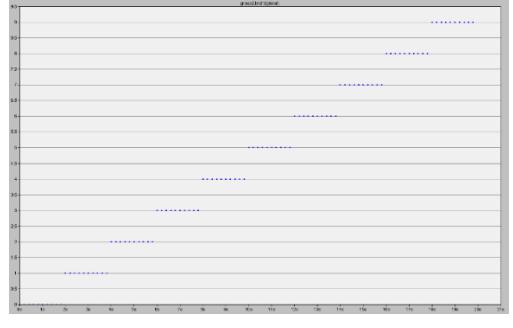
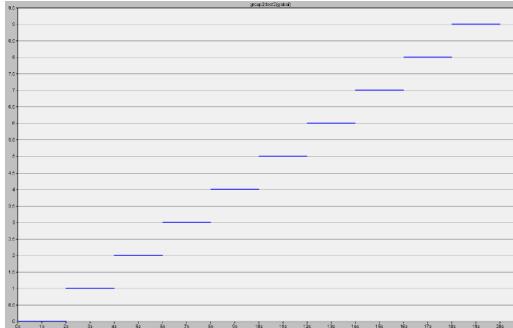
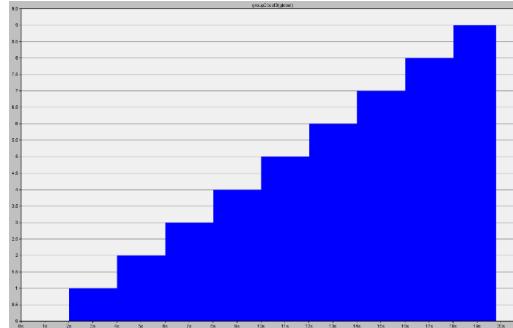
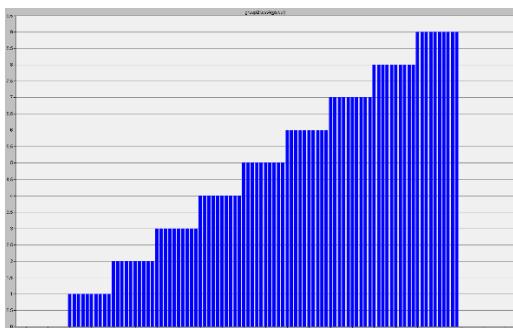
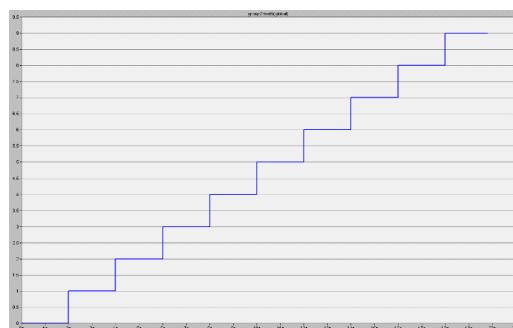
<p>Linear 风格, 每个计算时间窗的代表值以点的形式显示在坐标系中, 并使用直线连接起来。</p> 	<p>Discrete 风格, 每个计算时间窗的代表值仅仅以点的形式显示在坐标系中</p> 
<p>Sample-hold 将相邻且相同取值的统计量点连接起来</p> 	<p>Bar 使用柱坐标显示, 每个计算时间窗内的值使用柱体表示。柱体宽度占满整个时间窗</p> 
<p>Bar chart 使用柱坐标显示, 每个计算时间窗内的值使用柱体表示。柱体宽度不占满整个时间窗</p> 	<p>Square-wave 以矩形波的形式展示</p> 

图3-33. 统计量各绘制风格下示意图

3.7.4、注册统计量

(1) 定义统计量句柄:

在定义了统计量之后，仿真中需要向统计量写入很多数据，统计量的名称其实质是一个队列的名称，统计量句柄是指向该队列的一个指针，要想收集统计量的值，需要向这个队列写入值，就需要使用队列指针来完成。所以在状态变量中定义统计量句柄，变量类型为Stathandle；

(2) 注册统计量:

注册统计量就是把统计量句柄与统计量绑定。注册全局统计量时使用
统计量句柄 = op_stat_reg ("统计量分组.统计量名称",OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);

注册局部统计量时使用

统计量句柄 = op_stat_reg ("统计量分组.统计量名称",OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);

3.7.5、数据收集

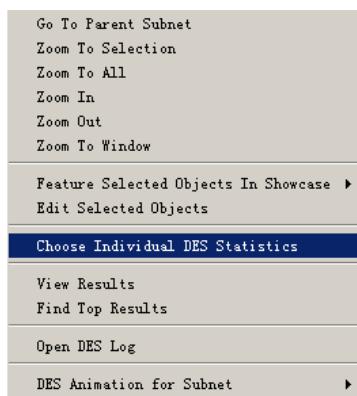
对数据的收集使用 op_stat_write (统计量句柄, 数值) 函数。写入数值时均是以 double 类型写入。写入时，系统会自动向指定统计量句柄下，写入统计量值和时间。

3.7.6、统计量的函数计算

对统计量的计算在选择 Bucket mode 时即设定好，无需人工编写，使用只用进行数据收集，软件会自动根据设定的函数和收集的数据，计算每个计算时间窗的代表值。

3.7.7、选择想要收集的统计量

(1) 全局统计量的选择收集:



定义了统计量、统计句柄，注册了统计量之后，记得要在进程界面先编译保存统计量才能在统计量列表中找到。统计量设置完毕后，需要在场景设置中选择收集此统计量，程序编译运行时才会收集。这是因为收集统计量需要较长时间，拖慢程序运行，如果默认全部收集会影响运行效果，且容易产生死机，所以 OPNET 让使用者自己选择要收集的统计量。在子网模型节点单击右键，如图 3-34，选择 Choose Individual DES Statistics，进入全局统计量的选择窗口。也可以在 DES 菜单栏中

图3-34. 全局统计量收集选择入口
选择 Choose Individual Statistics 进入统计量选择。

在选择窗口我们看到在进程模型界面定义的全局统计量。选中收集，不选中则不收集，统计量收集的数目越少，仿真速度越快。选中统计量时，可在窗口右半边设置统计量的绘制风格和收集模式。点击 OK，保存设置

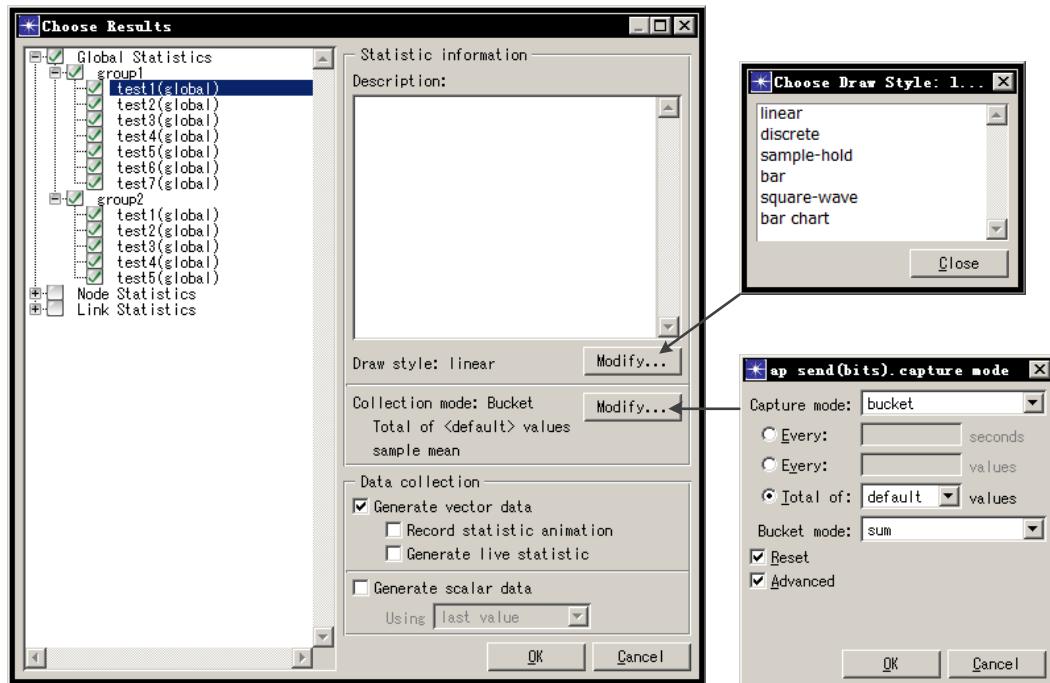


图3-35. 统计量收集选择设置窗口

(2) 局部统计量的选择收集

右键单击注册了局部统计量的节点，选择 Choose Individual DES Statistics，进入局部统计量编辑界面。

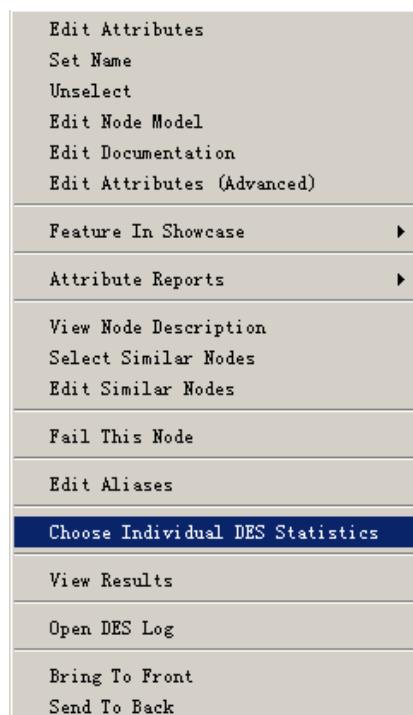


图3-36. 局部统计量收集选择入口

则局部变量界面可以看到除了在进程 mac1 模型界面设置的局部统计量外，还有无线发信机 send 模块和无线接收机 receive 模块自带的局部统计量。同样选中统计量可以编辑绘制风格和收集模式。

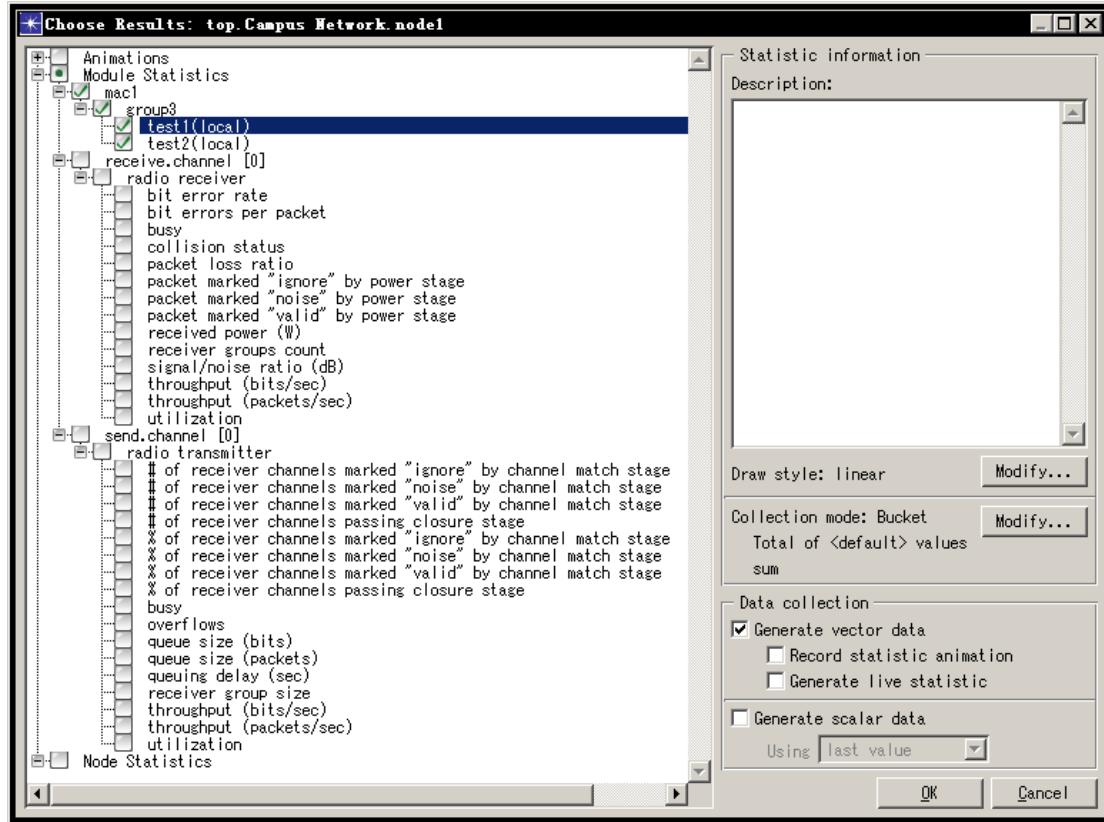


图3-37. 局部统计量收集选择设置窗口

3.7.8、数据显示及导出

这里仍使用 3.7 章节的统计量数据。每隔 2 秒记录一次当前时间作为统计量，仿真时间为 100s。仿真运行完毕，点击工具栏查看仿真结果按钮 ，查看统计量收集结果的显示。

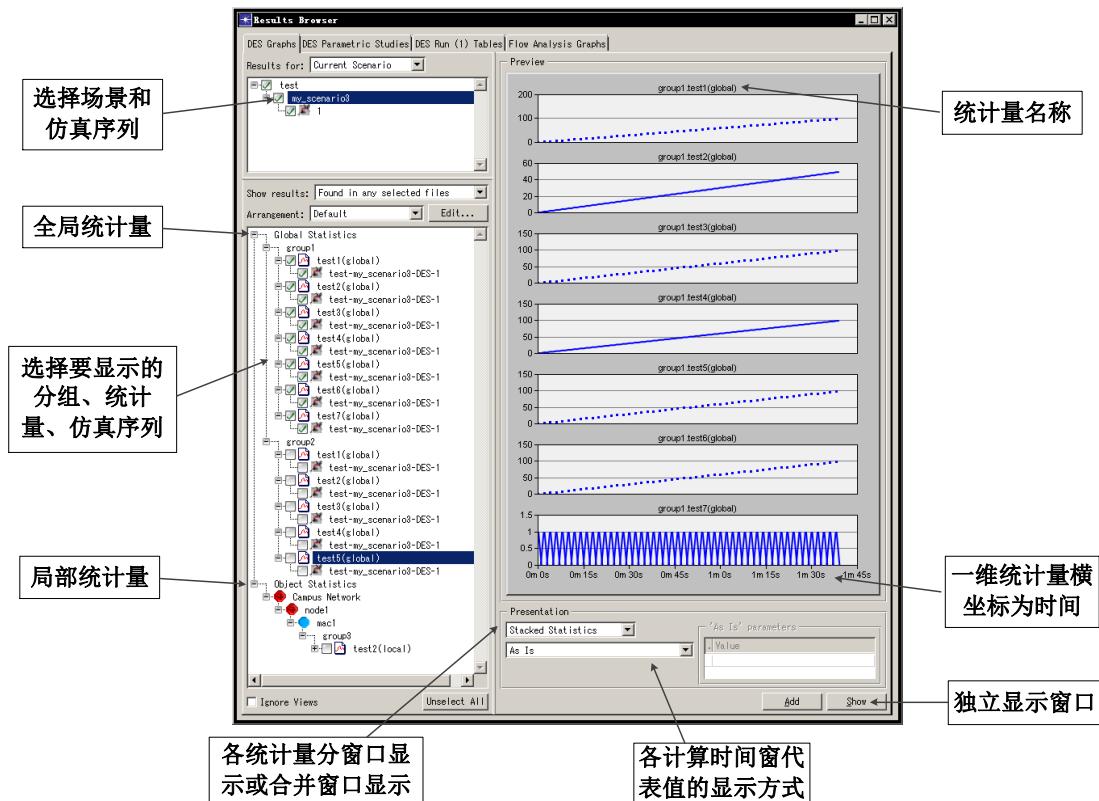


图3-38. 数据显示设置窗口

每一个场景的每一个随机种子都对应一次仿真结果。通过仿真结果可以掌握 Bucket mode 中函数的意义。在记录数据时，我们统计每隔 2s 记录一次当前时间。

记录时间	0.0s	2.0s	4.0s	6.0s	8.0s	10.0s	...
记录值	0.0	2.0	4.0	6.0	8.0	10.0	...

(1) 项目、场景、仿真序列的选择：

在场景和仿真序列选择区域，当有多个仿真场景或多个仿真序列需要对比时，需要选择希望展示的不同项目或场景下的仿真结果。

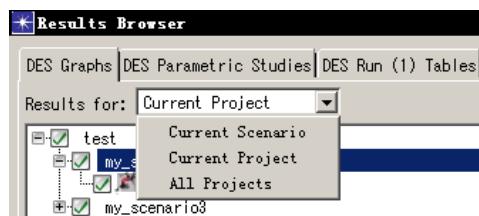


图3-39. 选择场景和仿真序列

网络仿真中有时需要使用不同的随机种子做很多次实验，将实验结果进行平均以消除随机性对系统性能的影响，所以常常一个统计量下的随机序列往往有多个。

(2) 显示统计量的选择

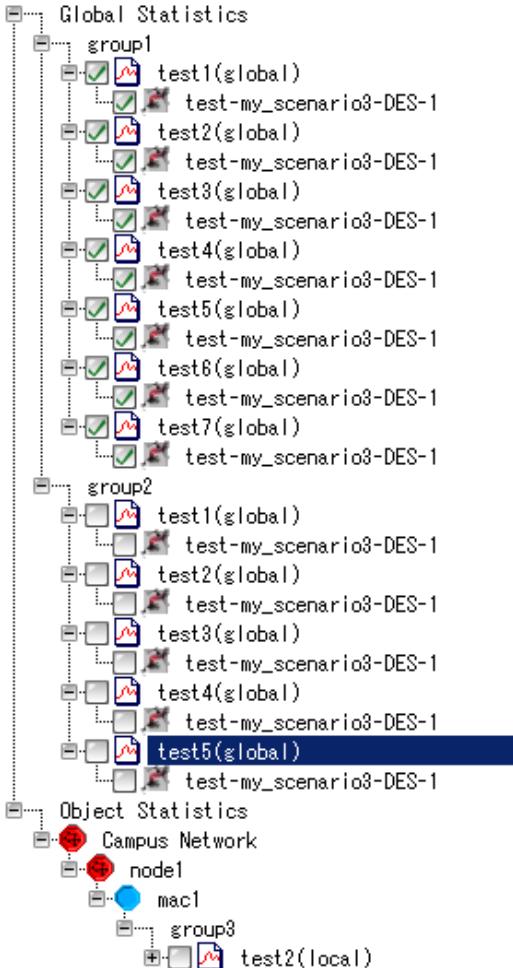
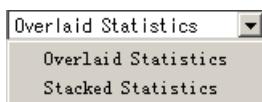


图3-40 显示统计量选择

我们可以选择自己当前希望显示在窗口中的全局或局部统计量。也可以选择统计量下不同的仿真序列对应的结果。有时由于代码编写的问题，造成有些随机种子的结果显示有明显错误，我们可以通过这样找到不适用于当前程序的随机种子，并在这些随机种子下进行调试，以便找到错误。

(3) 各统计量显示控制

在各统计量显示控制下拉框中。



当选择 Stacked Statistics 时，各统计量在不同的坐标系下显示。适用于显示不同类型的统计量，如图 3-41 所示

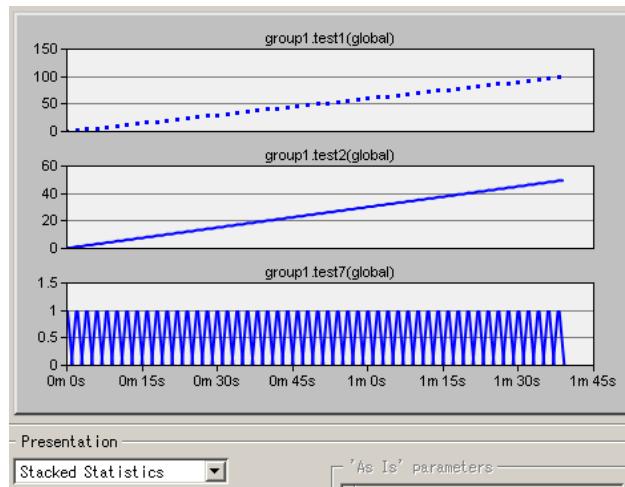


图3-41. 统计量分窗口显示

当选择 Overlaid Statistics 时，各统计量在同一坐标系下显示。适用于显示相同类型的统计量。当显示在同一坐标系的各统计量取值差距较大时，取值较小的统计量不能很好的在坐标系下反应变化情况。如图 3-42 所示。

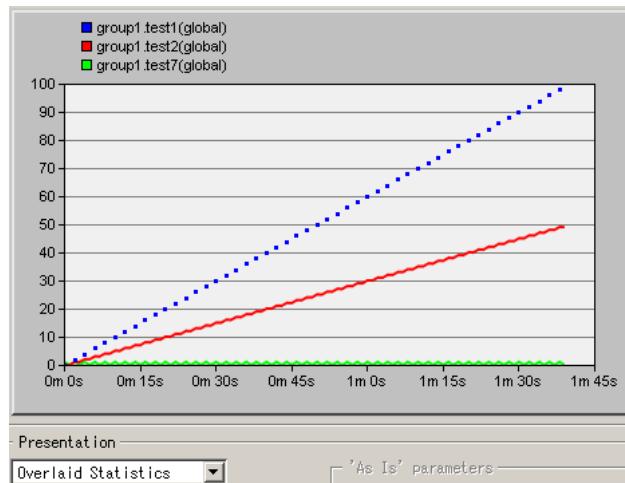


图3-42. 统计量合并窗口显示

(4) 各计算时间窗值的显示控制

默认情况，在坐标系中每个取值点上的纵坐标值均是一个计算时间窗内所有统计量经过设定 Bucket mode 后的代表值。

```

As Is
Probability Density (PDF)
Cumulative Distribution (CDF)
Probability Mass (PMF)
Histogram (Sample Distribution)
Histogram (Time Distribution)
abscissa_filter
adder
average
constant_shift
delay_element
differentiator
exponentiator
gain
glitch_notch
integrator
limiter
logarithm
moving_average
multiplier
reciprocal
sample_sum
time_average
time_window
value_notch

```

图3-43. 统计量窗口代表值处理函数

表3-37. 统计量窗口代表值处理函数方法说明

关联方式	含义
As Is	各计算窗口独立显示。
Probability Density (PDF)	概率密度，用以得出结果取值的概率密度函数
Cumulative Distribution (CDF)	累积分布函数，用以得出结果取值的累积分布函数，这一函数与概率密度函数的关系一般视为积分、微分的关系
Probability Mass (PMF)	概率，用以得出结果取值的概率密度函数，它与概率密度间的关系一般视为对 PDF 做逐段的积分
Histogram (Sample Distribution)	取样分布的柱状图，它对结果的纵轴取值进行分段统计，得出各分段中取值数量，分段数目为偶数，默认为 100，可以根据结果规模和分布进行自动调整
Histogram (Time Distribution)	时间分布的柱状图，它对结果的纵轴取值的持续时间进行统计，得出各种值在时间轴（横轴）上占用的时间总数
abscissa_filter	取结果的横坐标为输出结果的纵坐标

adder	将两个结果加和成一个结果
average	算数平均值
constant_shift	将结果延纵轴移动一定量
delay_element	水平平移
differentiator	求导
exponentiator	指数函数，对结果进行求 p 次方
gain	按一定比例放大结果
glitch_notch	重复值打孔，去除数据中横纵坐标完全相同的点。
integrator	积分
limiter	限幅，切除结果中纵轴数量超出限制的部分
logarithm	对数函数，对结果求 log10
moving_average	移动平均，目的是通过移动窗口对结果进行连续的局部窗口平均形成新曲线。局部窗口平均是按一定尺寸的窗口选择，并对他们进行持续时间加权平均得到一个值的计算。
multiplier	将两个结果相乘为一个结果
reciprocal	求原结果的倒数，原结果为 0 除外
sample_sum	自身求和
time_average	持续时间加权平均
time_window	限时间，切除结果中横轴数量超出限制的部分
value_notch	数值打孔，将结果中落于一定范围的数值去除，范围为 $(x-v, x+v)$ x 是滤波器参数，而 v 由 value_notch_filter_tolerance 偏好决定

点击右下角的 show 按钮可以分离坐标系显示窗口。如图 3-44 所示。

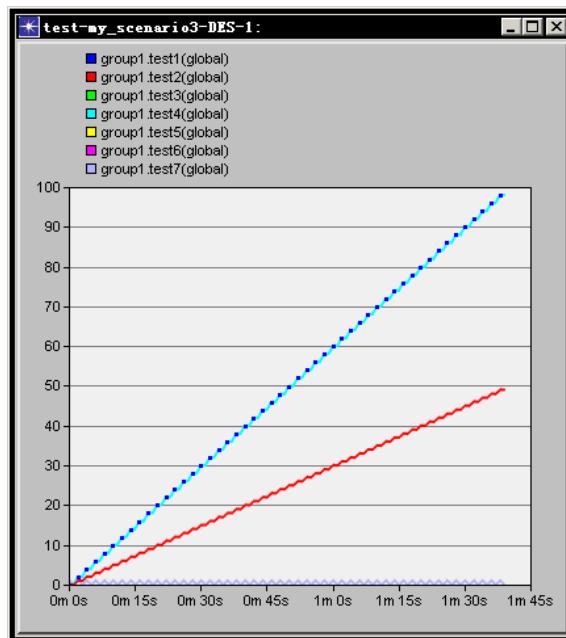


图3-44. 统计量显示窗口分离

在分离窗口中坐标系外点击右键可以设置坐标系配置。



图3-45. 统计量窗口控制

表3-38. 统计量窗口控制说明

设置	功能
Edit Panel Properties	设置分离窗口的显示控制
Show Statistic Data	查询修改统计量数据
Duplicate This Panel	复制窗口
Add Graph	在当前窗口添加一个坐标系
Use Same Vertical Scale	多个坐标系使用相同的纵坐标范围
Full Horizontal Scale	横坐标显示全部范围
Export All Graph Data to Spreadsheet	输出所有坐标系中的数据到表格
Make Panel Template	复制当前的坐标系，去除显示数据
Load Data Into Template	加载数据到坐标系中
Hide This Panel	隐藏分离窗口
Make Panel Annotation In Network	
Time Axis	时间轴显示，可选择以秒为单位，或分秒同时显示
Chart Style	坐标系显示风格

其中点击 Edit Panel Properties 进入分离窗口属性设置界面。

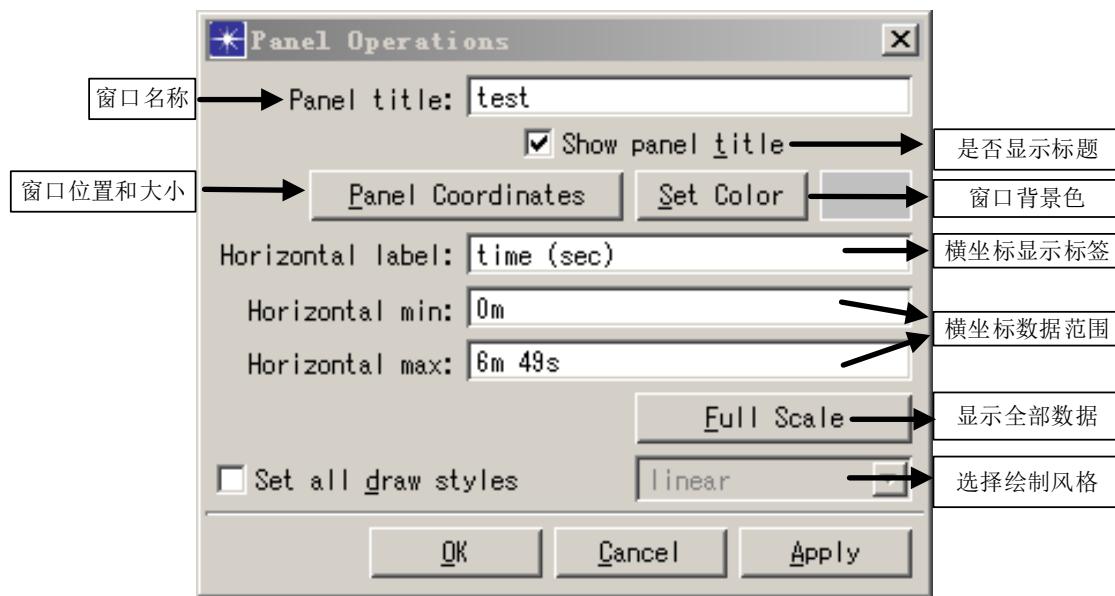


图3-46. 统计量显示及数据操作窗口

点击 Show Statistic Data 进入统计量数据查看修改界面，如图 3-47 所示。选择 Statistic Data 可以直接查看到统计量的统计量存储位置、名称、类型长度等信息，以及统计量具体数值。在这个窗口中可以直接修改统计量的值，点击 Build New Statistic 可以直接根据新的数据生成新的坐标系显示图。

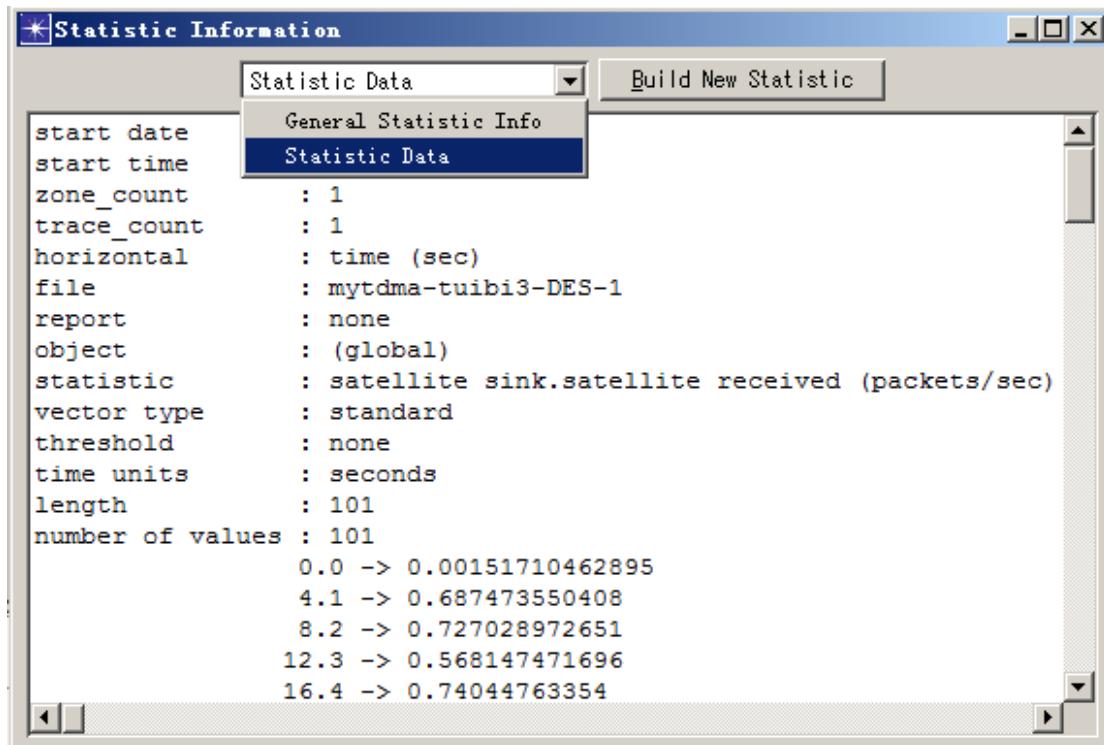


图3-47. 统计量数值查看修改窗口窗口

在分离窗口中坐标内点击右键可以设置显示配置。



图3-48. 统计量坐标系控制窗口

表3-39. 统计量坐标系操作说明

设置	功能
Edit Graph Properties	坐标系显示配置窗口
Edit Panel Properties	窗口显示
Add Statistic	添加统计量
Full Vertical Scale	显示全部纵坐标范围
Full Horizontal Scale	显示全部横坐标范围
Full Scale	显示全部横纵坐标范围
X Grid	是否显示 x 轴网格线, Disabled 为不显示, Slid 显示实线, Dshed 为显示虚线
Y Grid	是否显示 y 轴网格线, Disabled 为不显示, Slid 显示实线, Dshed 为显示虚线
Draw Thickness	统计线粗细
Draw Style	统计量绘制风格
Symbol Size	
Show Trend Line	显示变化趋势
Use Log Scale	使用 log 坐标系
Export Graph Data to Spreadsheet	导出数据到 excel
Make Graph Template	提取坐标系
Load Data Into Template	加载数据到坐标系
Generate Distribution from Trace	根据统计量数据生成随机分布

Remove Trace	删除与统计量的关联，去除显示统计量，不删除文件
--------------	-------------------------

点击 Edit Graph Properties, 进入坐标系显示配置窗口

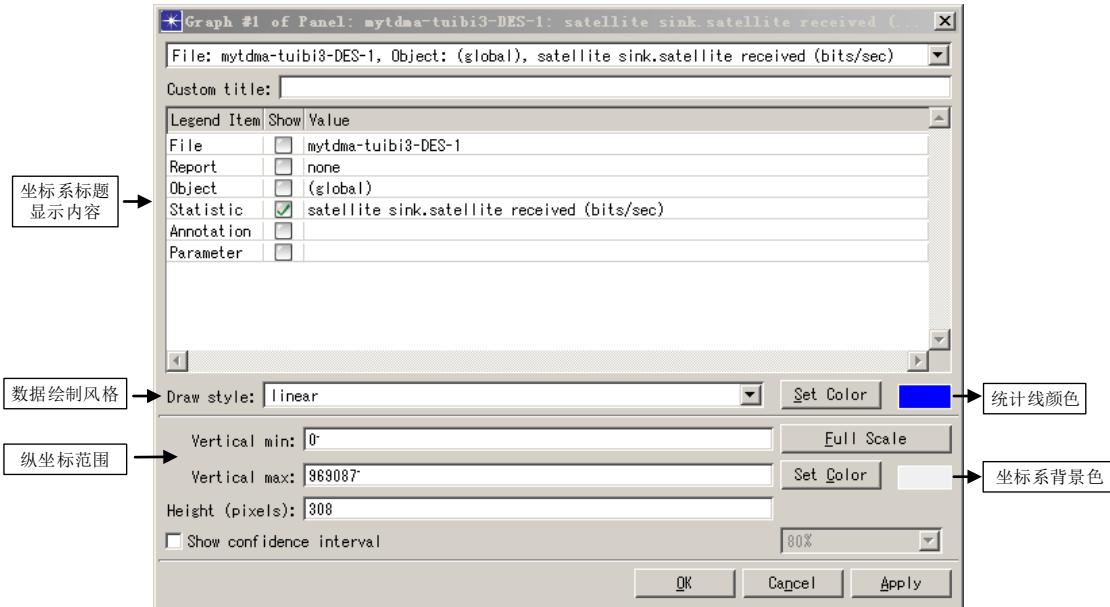


图3-49. 统计量坐标系修改窗口

点击 Export Graph Data to Spreadsheet, 可以直接将统计量的值导出到 excel，通过 excel 进行处理，导入到 MATLAB 进行绘图，能绘制出更加美观的结果图来。

第四章 调试演示

4.1、调试

4.1.1、仿真设置

在系统模型搭建与代码编写实现协议功能完成后，或完成过程中可进行及时的调试。进行调试前，必须在系统中将用到的每个进程模型中进行进程编译，在进程模型中点击 ，编辑进程，保证进程没有语法错误。当每个进程编译成功。在进行系统仿真调试。

在子网模型节点，点击 ，会弹出仿真调试设置窗口。如图 4-1 所示。

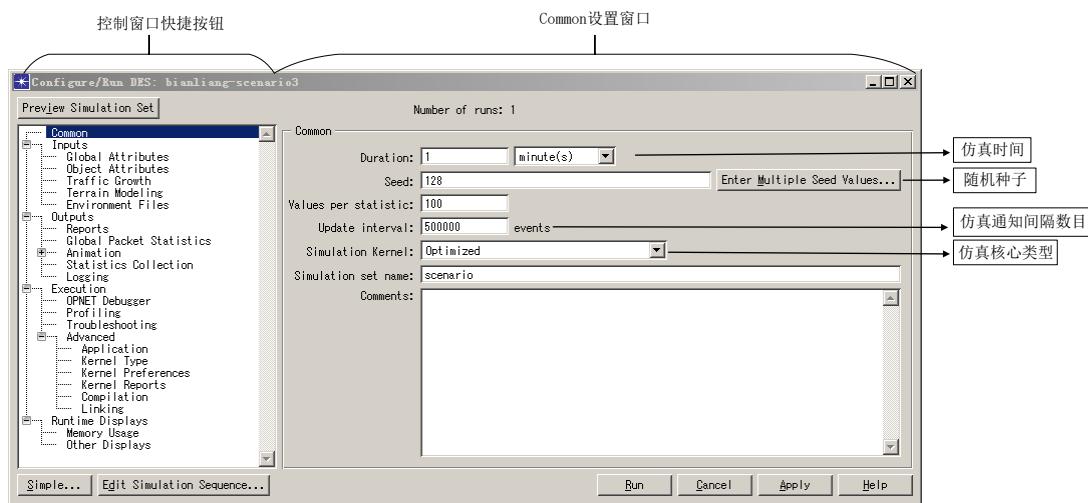


图4-1. 仿真常规窗口设置

在 Common 窗口可以设置仿真时间，随机数，仿真更新通知间隔事件数目，仿真核心类型等。首先介绍一下仿真核心的类型，在 Simulation Kernel 中可以选择的仿真核心类型包括 Based on ‘kernel_type’ preference、Development、Optimized。前两种仿真类型类似，我们这里只讲述 Development、Optimized 两种仿真核心。

(1) 仿真输入设置：

在 Inputs 选项下有 Global Attributes、Object Attributes、Traffic Growth、Terrain Modeling、Environment Files 几个设置界面。

点击 Object Attributes 进入提升的对象属性设置窗口。如图 4-2 所示。点击 Add 按钮，可以看到所有提升到顶层的对象属性，点击每个属性前的 Add? 下的空白表格单元格，可以添加到仿真设计中。在对象属性窗口的 Attribute 中便可以看到添加到本次仿真的提升到顶层的对象属性。在 Value 栏中填入相应的值，点击 Enter Multiple Values 可以设置某属性一系列参数值，在仿真中即会以这些值作为参数分别进行仿真。如图 4-2 所示，设置此次仿真，子网模块的 priority 属性值为 2 或 3 或 4，节点 1 的 user_id 属性的值为 1,2,3,4,5。

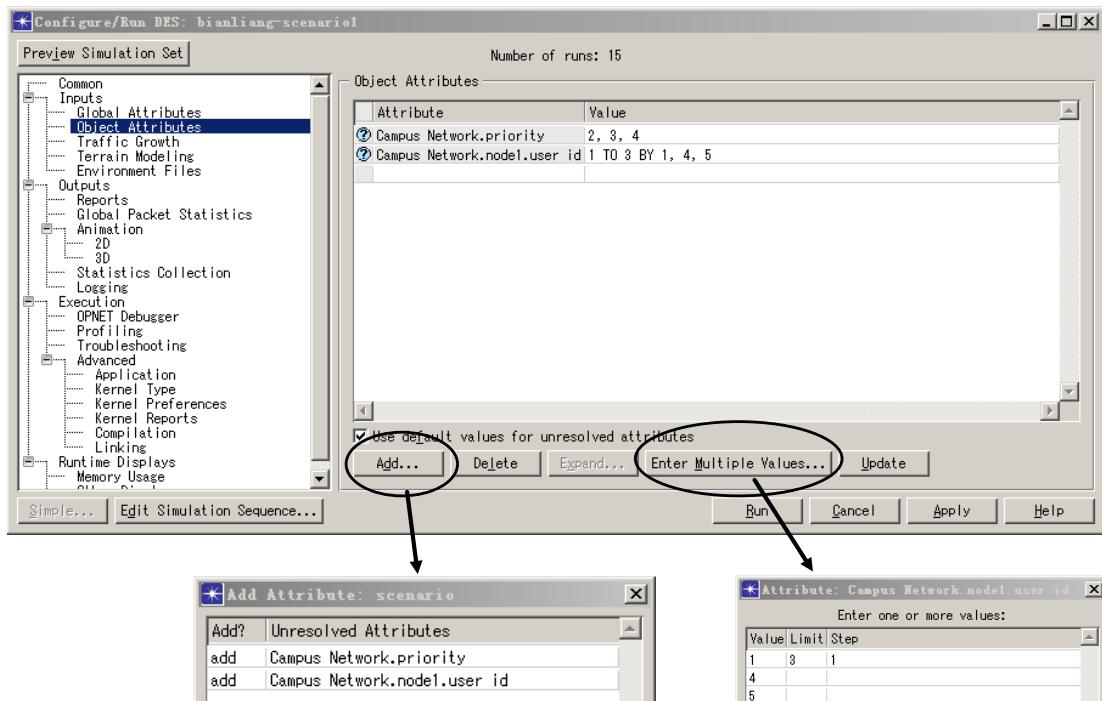


图4-2. 仿真对象属性设置

点击 Traffic Growth 进入流量增长属性设置窗口。如图 4-3 所示。在这个窗口可以设置迭代仿真次数，每次仿真间隔的时间，每次仿真流量增长比例。此设置必须要在 Outputs 的 Reports 中选中一个报告才能应用。

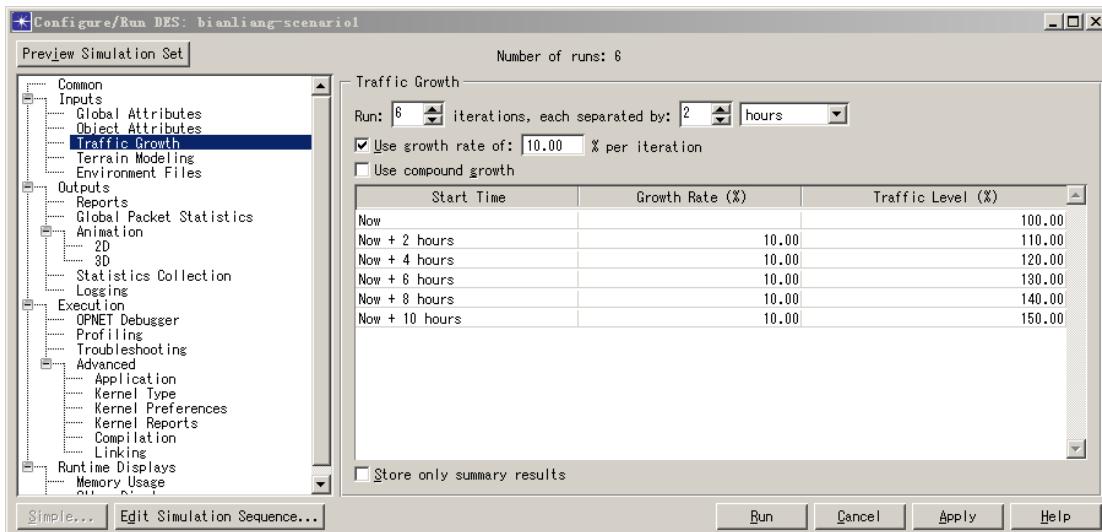


图4-3. 仿真流量阶段性增长设置

点击 Terrain Modeling 用于设置地形数据。点击 Environment Files 用于设置环境文件。

(2) 仿真输出设置：

在 Outputs 选项下有 Reports、Global Packet Statistics、Animation、Statistics Collection、Logging 几个设置界面。

点击 Reports 进入输出报告设置界面。如图 4-4 所示。在输出报告设置界面点击 Define Statistics Report 定义统计量报告，在定义界面选择 New report 创建新的报告，会弹出

统计量选择界面。选中自己想要生成报告的统计量，点击 Save 保存，弹出报告名称定义。输入报告名称点击 OK。在统计量报告栏中变出现刚定义的新的报告。选中点击 Apply 应用。输出报告设置完成。

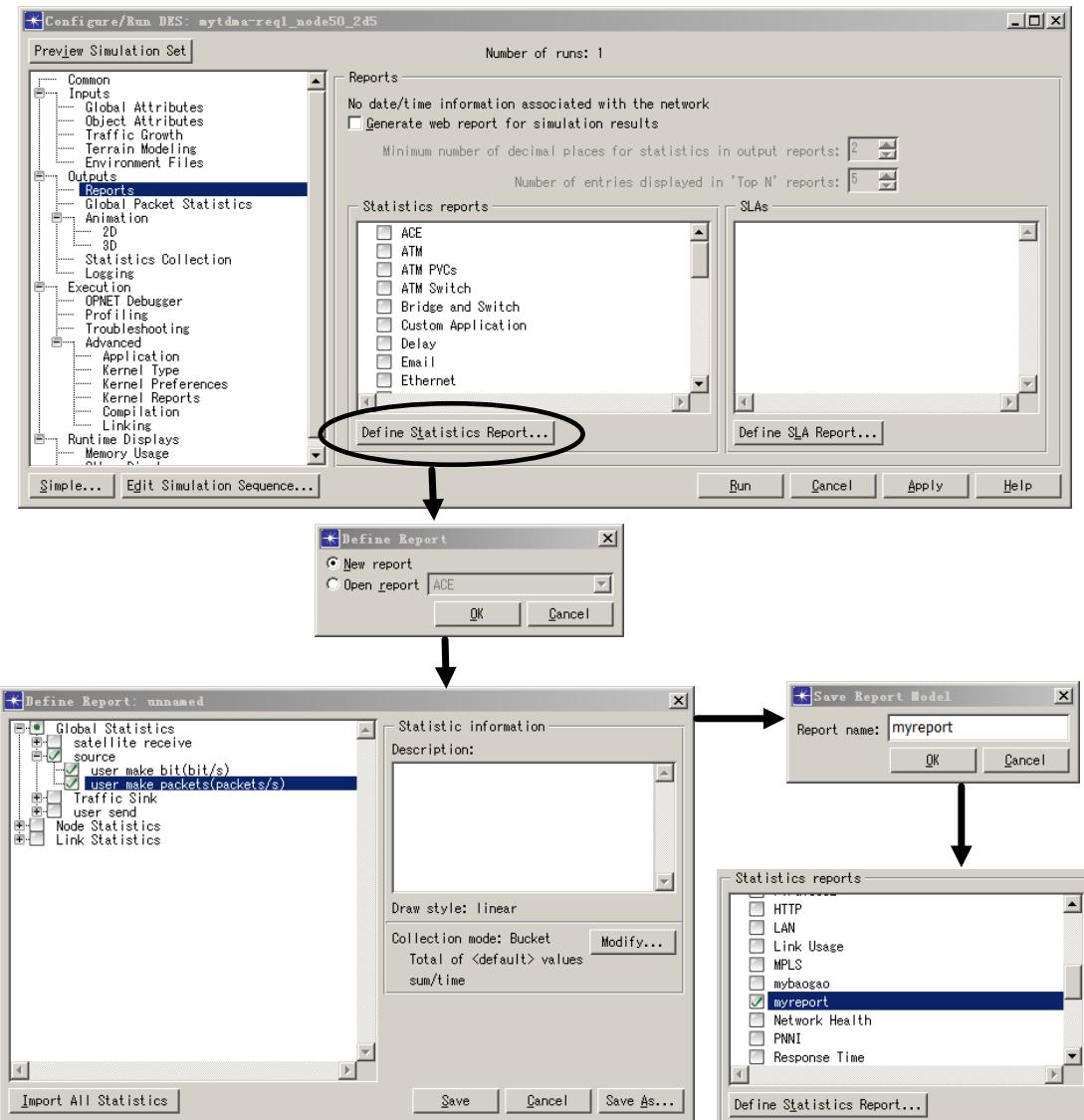


图4-4. 仿真输出报告设置

点击 Animation 可以选择 2D、3D 动画的收集。这与仿真中查看动画演示是不一样的。Statistics Collection 用于统计量收集开始结束时间设置，默认为全程均进行数据收集，例如使用者向错过出入网时间，仅收集系统稳态后的数据统计情况，可以设置这里。Logging 用于日志记录收集设置，是用于收集仿真中出现的错误的。

(3) 仿真执行设置：

在 Execution 选项下有 OPNET Debugger、Profiling、Troubleshooting、Advanced 几个设置界面。

当 Common 界面选择 Development 开发者模式时，在 OPNET Debugger 设置界面选中 Use OPNET Simulation Debugger (ODB)，即可开启调试模式。

在 Advanced 高级设置的 Kernel Type 核心类型中可以设置仿真核心使用开发者模式还是最优性能模式，使用串行仿真还是并行仿真，以及使用的地址空间。地址空间选择 32 位。

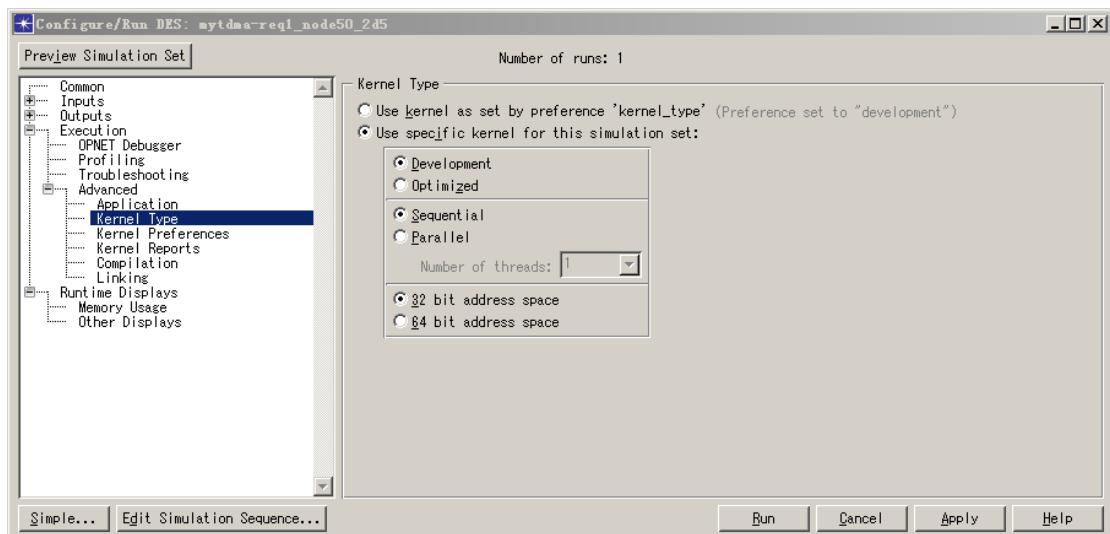


图4-5. 仿真核心类型设置

(4) 运行显示:

在 Runtime Displays 选项下有 Memory Usage、Other Displays 两个设置界面。在 Other Displays 中可以选择是否绘制事件进程的速度，显示每次运行的参数设置，显示统计量面板等，如图 4-6 所示。

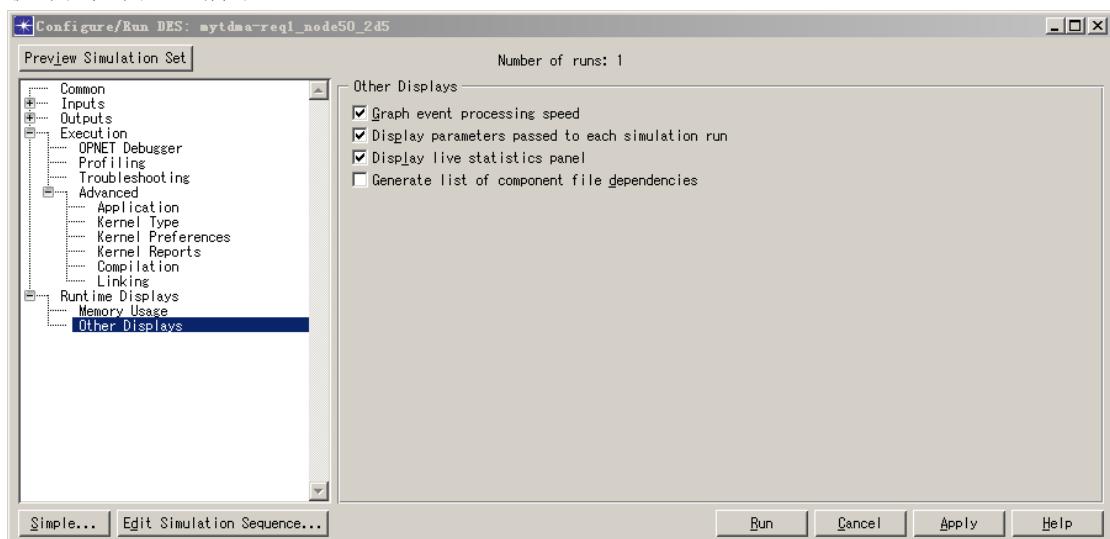


图4-6. 仿真输出显示设置

4.1.2、ODB 调试

在仿真中最长使用的仿真核心类型是调试模式和性能最优模式。其中调试模式 ODB 能输出方仿真中打印的信息，观看报文交互动画，中途断点调试，查看包信息，与事件信息。所示在未完全正确实现协议前均是使用 ODB 调试模式。在全完成网络设计后，使用 ODB 调试无误后，以后便可以使用 Optimized 模式，加快仿真速度。

在仿真窗口的 Common 设置界面 Simulation Kernel 仿真核心类型中选择 Development 调试模式，在 Execution 的 OPNET Debugger 中选中 Use OPNET Simulation Debugger(ODB) 即可开启调试模式。点击 Run，运行程序进入，调试窗口。如图 4-7 所示。

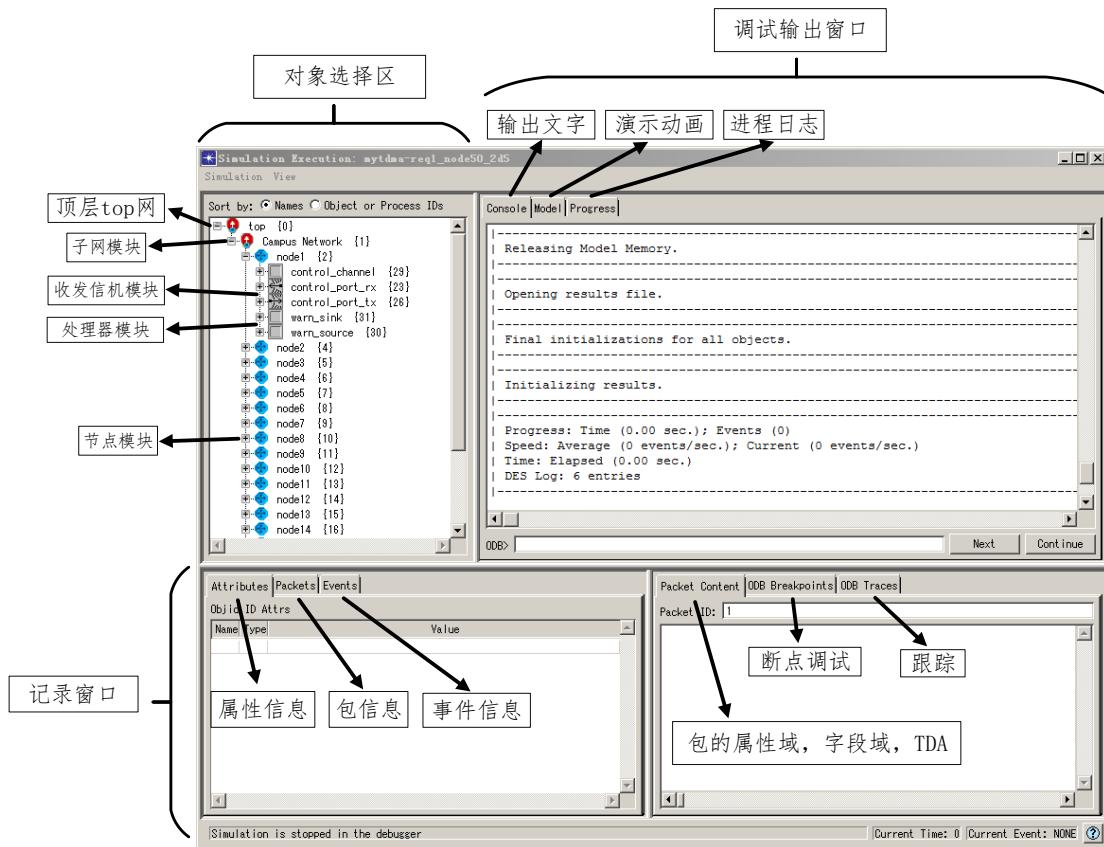


图4-7. ODB 调试窗口图

4.1.2.1、调试输出窗口

4.1.2.1.1、控制台输出窗口

在调试输出窗口可以输出打印信息，演示动画，进程日志。输出打印信息界面如图 4-8 所示。在界面下方，是 OPNET 命令输入框。Next 表示程序运行下一步，Continue/break 表示继续/暂停。在命令行输入 q 回车表示结束仿真程序。在打印窗口会跟随仿真程序的进行，打印输出程序中 printf 输出的内容。

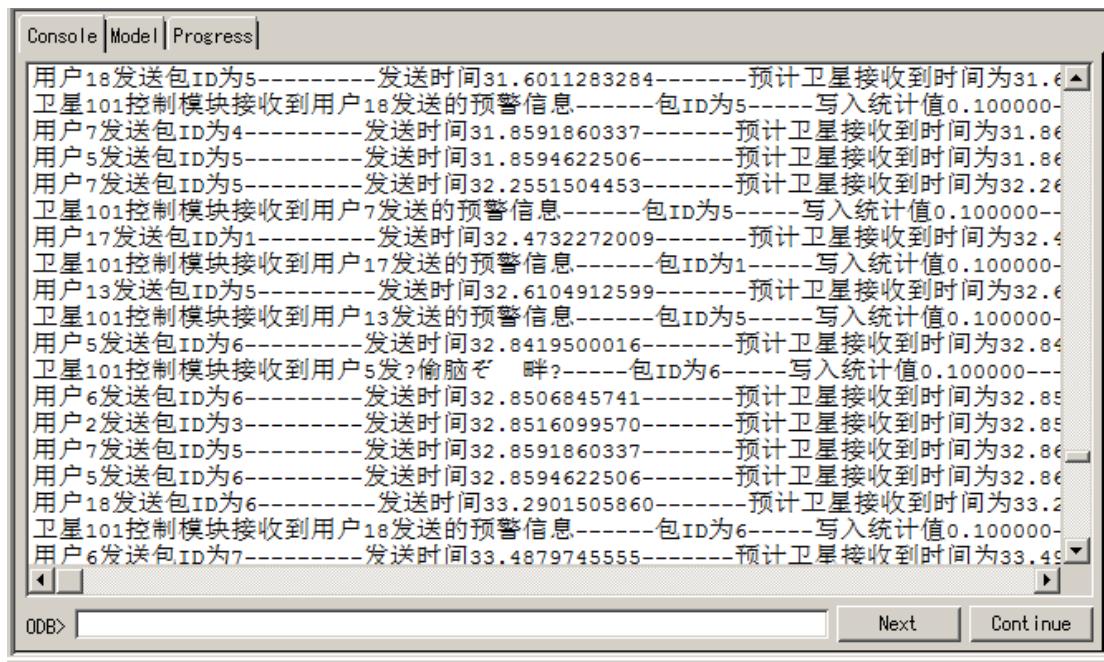


图4-8. 控制台窗口图

在命令框中输入仿真设置命令，进行相应的仿真需求，主要包括断点控制命令、触发行管理、执行控制、对象遍历、属性信息、包相关信息、进程相关信息、内存信息、跟踪信息、事件信息。

(1) 断点控制命令

与一般的符号调试器一样，断点是程序暂停执行的位置。断点控制有设置、查询、挂起、恢复、删除五种活动。

- 用户可以设定多种断点，包括：
 - ◆ 在时间附近暂停执行的断点，命令时“tstop[<abs_time>|+<rel_time>]”，如
 - “tstop 100.005”表示在绝对时间大于等于“100.005”的事件处暂停；
 - “tstop+100.005”表示绝对时间大于等于“当前时间+100.005”的事件处暂停；
 - ◆ 在某事件处暂停执行的断点，命令是“evstop<exec_id>|+<n>|#<sched+id>”，如
 - “evstop 12”表示在全局第12个事件处暂停；
 - “evstop+12”表示当前事件之后的第12个事件暂停
 - “evstop#12”表示在调度号为12的事件处暂停，调度号可以通过“evprint”命令得到。
 - ◆ 在操作某数据包时暂停执行的断点，命令是“pkstop<pk_id>”；
 - ◆ 某模块发生某类型事件时暂停执行的断点，命令是“instop<objref><intrpt>或intstop<objref>stream[<stream_index>[<packet_id>]]”，如
 - “intstop subnet1.node0.CPU regular”，表示在节点“subnet1.node0”的“CPU”模块发生周期中断时仿真执行暂停。
 - “intstop subnet1.node0.CPU stream”，表示在节点“subnet1.node0”的“CPU”模块发生流中断事件（接收到数据包）时仿真执行暂停。

- “intstop subnet1.node0.CPU stream 1”，表示在节点“subnet1.node0”的“CPU”模块的1号输入流接收到数据包时仿真执行暂停。
- “intstop subnet1.node0.CPU stream 1, 105”，表示在节点“subnet1.node0”的“CPU”模块的1号输入流接收到105号数据包时仿真执行暂停。

另外<intrpt>参数可以指定任何可以发生在进程中的事件类型，中断类型与参数值对应关系如表4-1所示。

表4-1 中断类型与参数值对应表

中断类型	参数值	中断类型	参数值
存取中断	access	进程中断	process
仿真开始	begsim	恢复中断	recovery
仿真结束	endsim	定期中断	regular
失效中断	failure	远程中断	remote
外部系统进程中断	interface	自中断	self
多播中断	multicast	统计线中断	statwire
过程调用中断	procedure	流中断	stream

- ◆ 某模块发生任何事件时都暂停执行的断点，命令是“mstop<objref>”，如“intstop subnet1.node0.CPU regular”；
- ◆ 某模块发生任何事件时都暂停执行的断点，命令是“prostop<process_id>”，如“prostop 95”；
- ◆ 链路上的收发过程的断点，命令是“comstop <link_objref>|<tx_objref>:<rx_objref>[<ps_name>]”设置断点，如
 - “comstop “top.servers.switch<->server_with_comp””，表示在链路“switch<->server_with_comp”上，从“switch”到“server_with_comp”节点方向的任何管道阶段都暂停执行。
 - “comstop top.servers.switch.hub_tx_0:*”，表示由“top.servers.switch.hub_tx_0”发射机模块到任何接收机的通信链路的任何管道阶段都暂停执行。
 - “comstop top.servers.switch.hub_tx_0:* propdel”，表示由“top.servers.switch.hub_tx_0”发射机模块到任何接收机的通信链路的传播时延管道阶段都暂停执行。
 - “comstop *”，或者“comstop *:*”，表示在任何链路的任何管道阶段上都暂停执行，即只要发生链路上的数据包收发就执行中断。

另外，<ps_name>参数可以选择的阶段与参数值的对应关系如表4-2所示

表4-2 不同链路可选阶段与参数值对应表

管道阶段名称	参数值	点到点链路	总线型链路	无线
传输时延	txdel	有	有	有
传播时延	propdel			
误码分配	error			
纠错	ecc			
链路闭合	closure	无		

碰撞	coll		
信道匹配	chanmatch		
发信机天线增益	tagain		
收信机天线增益	ragain		
接收信号功率	power		
干扰噪声	inoise		
背景噪声	bkgnoise		
信噪比	snr		
误比特率	ber		

由于无线管道模型的收信机分组阶段是为了减少计算量而设计的，在这里不视为真正的通信过程，所以不能作为可中断的链路阶段。

- ◆ 在两个事件之间执行的代码中暂停执行的断点，也成为标签断点，标签的预定义由代码中的“op_prg_odb_bkpt”完成，激活由命令“lstop<label>”完成，如“lstop”DataReq”，表示在其后执行到所有“op_prg_odb_bkpt(“DataReq”)”语句时都中断；
- ◆ 作用范围只局限在某模块范围内的标签断点，命令是“mlstop<objref><label>”，断点设置对模块内的所有进程都有效
- ◆ 作用范围只局限在某模块范围内的标签断点，命令是“prolstop <process_id><label>”，断点只对具体进程有效。
- ◆ 在协同仿真中，数据到达外部系统接口时暂停执行的端点，命令是“interfacestop<objref>”。
- 查询已设置的断点，可以使用命令“status”，命令会分类显示各断点、跟踪、行为、内存标签的序号和状态
- 暂时取消某个断点，可以使用命令“suspstop <bkpt_id>|all”来挂起某个中断，bkpt_id可以通过“status”查询，被挂起的断点会被跳过，即使条件满足也不会暂停仿真执行。
- 恢复挂起的断点，可以使用命令“actstop<bkpt_id>|all”完成。
- 完全取消某个断点，可以使用命令“delstop <bkpt_id>|all”删除断点，删除后断点不能恢复。

(2) 触发行为管理命令

有事用户希望在某些事件发生的时候执行一些 ODB 命令，一百年输出一些辅助信息，比如每 100 个事件执行一次“snapshot”，以便形成统计快照。这是就需要使用触发行为了，触发行为了可以是任何 ODB 命令，可以根究需要带参数。

- 可以使用“evaction[<exec_id>|#<sched_id>|+<n>]<action>”命令来指定，<exec_id>、#<sched_id>、+<n>都与“evstop”中相同参数的含义一致，%<n>表示从当前事件起每第 n 个被执行的事件，<action>中的指令如果带有参数需要用引号包围起来。
- 如果希望暂时取消某些触发行为了，可以使用“suspaction<action_id>|all”命令，<action_id>可以通过“status”命令查询到。
- 如果需要恢复某些触发行为了，可以使用“actaction<action_id>|all”命令。
- 如果需要完全取消某些触发行为了，可以使用“delaction<action_id>|all”命令。

(3) 执行控制命令

ODB 作为调试工具，需要控制程序执行的推进节奏，该快的时候快，该慢的时候慢，是程序的执行按照调试的需要走走停停，为此提供了前进、退出、紧急暂停等功能。为了输出的便捷，还提供了一些快速指令。但即便 ODB 提供了这么多实用的命令，如果每次调试都需要逐字输入仍很麻烦，尤其是在某个问题点附近反复调试时更加如此，这时的断点配置、信息输出需求都基本相同，反复输入多条相同的命令很繁琐。为此 ODB 还提供了与 Windows 的控制台批量处理功能类似的脚本功能。

- 推进节奏控制包括：
 - ✧ “continue”，指令全速运行直至终止或下一个断点处。
 - ✧ “next[<num_step>|time]”，指示运行至几个事件或指定时间，其中：
 - “next” 表示运行一个事件后停止，类似一般的符号调试器的单步运行。
 - “next 100” 表示运行 100 个事件后停止，与“evstop +100” “continue”的组合使用效果。
 - “next time” 表示运行到下一个时间，在当前时间与下一个时间之间的任何事件都被全速执行通过。
- 在这一命令执行后，如果在停止之前遇到断点，会先停止在断点处。
 - ✧ “exit”，指示强行退出，不产生输出向量文件，临时输出片段 (*.of) 文件和动画记录文件都不会正常关闭。
 - ✧ “quit”，指示正常退出，产生输出向量文件，并且正确关闭临时输出片段 (*.of) 文件、动画记录文件。
 - ✧ “esastop”，指示协同仿真在当前时间执行完后返回到外部系统中继续执行。
- 快读指令包括：
 - ✧ “history[<number>]”，查询已经执行过的指令，例如
 - “history”，显示所有已经执行过的指令，指令按执行的先后次序编号，起始编号为 0。
 - “history 4”，显示最后 4 条已经执行过的指令。
 - ✧ “![<cmd_index>]”，用以执行之前已经执行过的命令，<cmd_index>可以通过“history”命令查询。
 - ✧ “alias<name>[<value>]”，用以对常用的长指令设定一个便捷的简化指令，例如
 - “alias get_size attrset #80 “pk size pdf””，表示用“get_size”代表“attrset #80 “pk size pdf””。
 - “alias” 显示所有定义的简化指令定义
 - “alias get_size” 显示简化指令“get_size”的定义。
 - ✧ “unalias<name>”，用以取消简化指令。
- 脚本功能包括：
 - ✧ 脚本是存放在文本文件中的命令，可以用“script_play<filename>[noprompt]”命令来执行，例如
 - “script_play wdx”，表示执行“wdx.scr”文件中的所有指令。
 - “script_play wdx noprompt”，表示执行“wdx.scr”文件中的所有指令，同时取消所有提示信息的回显。
 - ✧ 脚本可以手工编辑，也可以在交互的时候记录，记录时可以用“script_start”标志需要记录的起点，而用“script_save <filename>[all]”保存，“script_save”有两种保存方式，例如
 - “script_save wdx”，表示记录从“script_start”到“script_save”

之间所有的执行过的命令。

- “script_save wdx all”，表示记录所有执行过的命令，包括“script_start”和“script_save”。
- 记录的文件名是指定名称+“.scr”。
- ✧ 对已有的脚本可以用“script_view<filename>”命令查看
- ✧ 可以用“setvar[<name>|<name><value>]”命令设置命令变量，供脚本使用，例如
 - “setvar my_attr #80 “pk size pdf””表示设置命令变量“my_attr”为字符串“#80 “pk size pdf””。
 - “setvar my_attr”表示查看命令变量“my_attr”的设置
 - 命令变量与宏定义类似，每当 ODB 见到形如“\$var”和“%var%”的单词，都会将他们换成 var 对应的字符串
- ✧ 对命令变量设置的取消，可通过“unsetvar<name>”完成。

(4) 对象遍历命令

在命令行中可以通过命令遍历网络拓扑、节点内部模块结构、节点或模块的组合属性等。

- 通过命令“objassoc<objref><assoc_type>”可以枚举与“<objref>”指定对象相连的“<assoc_type>”类型的对象。
- 通过命令“objcount”可以汇报分类统计的网络中各种对象的数目。
- 通过命令“objid<objref>”可以返回“<objref>”指定对象的对象标志号，这一编号会随网络中的对象的增删发生变化，但“<objref>”使用的是对象的名称，只要对象自身名称及所在的层级结构不变，就不会变。
- 通过命令“objmap”来枚举满足要求的对象，“objmap”命令有两种格式“objmap<onjref>[parents|children]”或者“objmap<type>[<name>]”，例如
 - ✧ “objmap all”，会列出所有对象，显示能表明其层级关系的名称，对象标识、对象类型、父对象标识号等相关信息。
 - ✧ “objmap subnet”，会列出所有子网对象，记忆相关信息
 - ✧ “objmap subnet1.node0.CPU parents”，会列出对象 subnet1 子网下 node0 节点内 CPU 模块的父对象的相关信息。
- 通过命令“objprint<objref>[detail]”能返回<objref>指定对象的相关状态信息，不同对象的状态信息会有区别，例如
 - ✧ “objprint subnet1.node0.CPU”，会返回进程模块对象“subnet1.node0.CPU”的当前状态信息，包括属性、活动进程、注册的中断、中断优先级、输入数据包流状态、输出数据包流状态
 - ✧ “objprint subnet1.node0.hub_rx”，会返回接收机模块对象“subnet1.node0.hub_rx”的当前状态信息，包括属性、信道、输出数据包流状态。
 - ✧ “objprint subnet1.node0.hub_rx detail”，会在“objprint subnet1.node0.hub_rx”返回信息的基础上，加上附加的管道阶段结果信息。

在对象遍历命令中的参数可以选择的类型及对应的参数值，如表 4-3 所示。

表4-3 不同链路可选阶段与参数值对应表

类型	参数值	类型	参数值	类型	参数值
所有	all	所有模块	module	统计线连接	statwire

所有节点和子网	site	所有处理器、队列、外部系统模块	qps	子队列	subq
各种子网	Subnet			外部系统接口	esinterface
固定子网	Fixsubnet	处理器模块	proc	点到点发送信道	pttxch
移动子网	Mobsubnet	队列模块	queue	点到点接收信道	ptrxch
卫星子网	Satsubnet	外部系统模块	esys	总线发送信道	butxch
各种节点	Node	理想发生器	igen	总线接收信道	burxch
固定节点	Fixnode	恒定发生器	cgen	无线发送信道	ratxch
移动节点	Modnode	发射机模块	xmit	无线接收信道	rarxch
卫星节点	Satnode	点到点发射机	pttx	路径统计量探针	pathprobe
链路	Link	总线发射机	butx	自动动画探针	aaprobe
单向点到点链路	Simplink	无线发射机	ratx	耦合的节点统计量探针	cprobe
双向点到点链路	Duplink	接收机模块	recv	自定义动画探针	caprobe
总线型链路	Bus	点到点接收机	ptrx	需求统计量探针	demandprobe
总线接头	tap	总线接收机	burx	全局统计量探针	gsprobe
路径	path	无线接收机	rarx	链路统计量探针	lkprobe
所有需求对象	demand	天线模块	ant	实况统计量探针	rtsprobe
需求连接	dconn	节点内各种连接	conn	节点统计量探针	sprobe
需求流	dflow	流连接	stream	仿真属性探针	saprobe

(5) 属性信息命令

在调试中可以按需要查询设置更改对象的属性值。

- 对于一般对象属性值的查询，可以使用命令“attrget<objref><attr_name>”。
- 如需查看一般对象的属性信息，可以使用命令“attrprint<objref><attr_name>”，命令将显示属性名、类型、当前值、缺省设置、符号映射等信息。
- 对于仿真属性值的查询，可以使用命令“simprint<pattern>”，其中“<pattern>”是正则表达式，用以描述所需查询的属性名的特征。
- 对于一般对象属性值的修改，则可以使用命令“attrset<objref><attr_name><value>”。
- 设置外部接口的值，则需要使用命令“interfaceset<objref>[all|index<n>]<value>”，比如“interfaceset subnet1.node0.esys1.in 15”，表示将外部接口“subnet1.node0.esys1.in”的值设置为15，当设置的外部接口为输出接口时，则数据到达外部代码，若输入接

口，数据到达外部系统定义，但此时不会通知外部系统该值发生了变化

(6) 包相关信息命令

数据包的传输是网络的核心工作，其正确性是网络正常工作的保证。在调试中经常需要查看数据包的内容、流转过程等相关信息。常见的操作有查询、跟踪、自身内容显示、传输数据属性值显示、值向量显示、相关 ICI 显示等。

- 通过“pkmap[<pk_id>|all|stats|<pk_format>|#<objid>|<objid_hname>]”命令，可以查找所有具有指定特征的，当前仍未被销毁的包的基本信息，包括包标识号、包树示号、所有者名称等，例如
 - ✧ “pkmap all”，表示列举所有包的基本信息。
 - ✧ “pkmap stats”，表示按所有者统计包的数目。
 - ✧ “pkmap ethernet_v2”，表示列举所有格式的“Ethernet_v2”的包。
 - ✧ “pkmap #2872”，表示列举标识号为 2872 的对象所拥有的包。
 - ✧ “pkmap top.servers.switch.hub_tx_0”，表示列举对象“top.servers.switch.hub_tx_0”拥有的格式为“Ethernet_v2”的包的数量。
- 如果不能确定包的标识号、所有者、格式，还可以通过时间来限定查找范围，命令是“pktimemap<time>”，例如
 - ✧ “pktimemap 10”，表示查询最近 10s 创建的，当前仍未被销毁的包。
 - ✧ “pktimemap -10”，表示查询存活时间超过 10s，当前仍未被销毁的包，通常用于查询未被销毁的包。
- 如果需要显示包的自身属性内容、传输数据属性、值向量，可以使用“pkprint<pk_id>[tda|vvec|long]”命令，例如
 - ✧ “pkprint 1”，表示显示 1 号数据包的自身内容，包括标识、格式、创建模块、创建时间、尺寸、当前所有者内容，以及包的各字段名称、类型、尺寸、值。
 - ✧ “pkprint 1 tda”，表示显示 1 号数据包的自身内容的同时，附带显示传输数据属性值
 - ✧ “pkprint 1 vvec”，表示显示 1 号数据包的自身内容的同时，附带显示值向量数据
 - ✧ “pkprint 1 long”，表示显示 1 号数据包的自身内容的同时，附带显示传输数据属性值和值向量数据
- 包标识号可以用“pkmap”或“pktimemap”来查询。
- 使用“iciprint_pk<pk_id>”命令查看包相关联的 ICI 信息，包标识号可以用“pkmap”或“pktimemap”来查询。

(7) 进程相关信息命令

由于模块中进程会随动态进程的创建与销毁，构成一个动态变化的进程树，所以通常需要查看动态进程树是否正常。即使没有动态变化，进程的状态也是常需要关心的重要信息，因为进程的状态反应了状态机的情况。由于进程不是为一个对象，所以不能通过对象遍历访问到，为了获得进程的相关信息需要用到下列命令。

- 查询进程基本信息使用“promap[<objref>|<object_hname>|all]”命令，基本信息包括进程标识号、进程模型名、进程标签（在进程代码中通过 op_pro_tag_set 函数设置）、所有者标识号，例如

- ◆ “promap 15”，表示查询标识号为 15 的模块所拥有的进程。
- ◆ “promap subnet1.node0.CPU”，表示查询节点“subnet1.node0”中“CPU”进程模块中包含的所有进程。
- ◆ “promap all”，表示查询所有进程。
- 打印进程实例的详细信息使用“preprint <process_id>”命令，详细信息包括所属模块、父进程、子进程、当前状态、状态变量块地址。
- 打印用户自定义信息需要使用“prodiag<process_id>”命令或“proldiag<process_id><label>”命令，“prodiag”会调用进程模块中诊断块中的所有代码，而“prodiag”更会附加一个标签条件，该标签可以在诊断块中通过函数“op_prg_odb_ltrace_active”判断，这就相当于能在诊断块中划分出多个信息输出函数。

(8) 内存相关信息命令

内存泄露是一种常见且很难定位的错误，ODB 提供了分类内存和内存标签两种功能和信息，以辅助完成这类错误的定位。

- ODB 可以通过核心进程直接创建或间接分配的内存分类，并统计显示各类内存的使用情况。
 - ◆ 命令“memsnk<mem_category>”，显示指定分类的内存释放记录
 - ◆ 命令“memsrc<mem_category>”，显示指定分类的内存分配记录
 - ◆ 命令“memstats[long|data|diff|kernel|model|catmem|poolmem|general|private]<kbyte_threshold>|<pattern>”，显示当前使用中的内存情况
- 内存标签可以对内存使用进行更细的分辨，并有助于发现各类内存的使用细节。
 - ◆ 命令“mtagprint[<specifications>][<limit>]”可以用来显示分类，其中<specifications>参数指示显示标签内存信息的同时显示的附加信息，可以选用的值包括
 - “C”，同时显示其分类
 - “E”，同时显示分配时的事件标识号
 - “T”，同时显示分配事件
 - “M”，同时显示所属模块名
 - “O”，同时显示所属模块标识号
 - “P”，同时显示所属进程名
 - “I”，同时显示所属进程标识号
 - ◆ 命令“mtagfilter[all|<category_name>]”可以设定需要作标签的内存分类。
 - ◆ 命令“mtagon”打开标签功能
 - ◆ 命令“mtagoff”关闭标签功能，关闭后除了“mtagon”命令外不能执行其他内存标签的相关命令
 - ◆ 命令“mtagflush”清除所有已生成的标签信息。

(9) 跟踪信息命令

跟踪是一组在事件或某些行为发生时被调用的函数，能用于观测进程逻辑在动态上下文中的运行过程，以便验证进程调用的函数的参数、返回值的正确性。

用户可以控制跟踪信息格式、设定、查询、暂停、删除自定义跟踪、全局跟踪功能的

开启和关闭。

- 设置跟踪信息格式需要使用“tracefmt[indented|flat]”命令，“indented”表示以缩进方式展示函数调用信息，“flat”表示以扁平方式函数调用信息，信息全部左对齐。
- 用户可以设定的自定义跟踪包括。
 - ◆ 模块跟踪，用“mtrace<objref>”命令设置，能指示模块中各进程的激活、进入或退出进程状态、核心过程调用，以及仿真核心的活动。
 - ◆ 进程跟踪，用“protrace<process_id>”命令设置，能指示进程触发（调用）、进入或退出进程状态、核心过程调用、仿真核心的内部活动。
 - ◆ 包跟踪，用“pktrace<pk_id>”命令设置，能指示操作指定数据包的核心过程，以及仿真核心的活动。
 - ◆ 包树跟踪，用“pttrace<pk_tree_id>”命令设置，能指示对制定数据包树进程操作的核心过程，以及仿真核心的活动。
 - ◆ 全局标签跟踪，用“ltrace<label>”命令设置，能对所有进程有效，执行之后进程代码中调用核心过程“op_prg_odb_ltrace_active(<label>)”能返回1。
 - ◆ 模块标签跟踪，用“mltrace<objref><label>”命令设置，与全局标签跟踪的区别在于作用范围局限与某指定模块。
 - ◆ 进程标签跟踪，用“proltrace<process_id><label>”命令设置，与全局标签跟踪的区别在于作用范围局限于某指定进程。
 - ◆ 外部系统跟踪，用“interfacetrace<objref>”设定，能显示操作指定外部系统接口的核心过程和仿真核心的内部活动。
- 全局跟踪功能包括
 - ◆ 封装跟踪，用“encaptrace”命令打开或者关闭，打开时在某个包被设置为另一个包的字段内容时会打印信息。
 - ◆ 执行跟踪，用“exectrace”命令打开或关闭，打开时会在每个事件执行之后打印附加信息，包括最高的包标识号、事件表中的事件数、已执行事件数等，可以用来考察不同版本的Modeler对相同的仿真模型的影响。
 - ◆ 完全跟踪，用“fulltrace”命令打开或关闭，完全跟踪能显示所有核心过程的调用过程，同时相当于启动了所有的标签跟踪。
- 用“status”能查询已经设定的用户自定义跟踪。
- 挂起某些跟踪使用“susptrace<trace_id>|all”命令，挂起的跟踪暂停输出信息。
- 用“acttrace<trace_id>|all”命令激活被挂起的跟踪，使指定跟踪继续输出信息。
- “where”命令能显示断点处的函数调用栈。
- 如果不知道进程有哪些标签跟踪可以设定，可以使用“lmap[<match>|all]”命令来查询，其中<match>是正则表达式描述的查询条件。

(10) 事件信息命令

一般情况下，断点处的提示标题上都有当前事件的详细信息，但有时也需要查询其他事件的信息或者当期事件的相关ICI内容。

- 可以用“evprint +<n>|all|#<sched_id>|<objref>[+<n>]|@<time>|@<time1>-<time2>”命令查询其他事件的详细信息，例如
 - ◆ “evprint +3”，表示列出后3个事件的信息
 - ◆ “evprint all”，表示列出事件列表中的所有事件的信息。

- ◆ “evprint #300”，表示显示调度号为 300 的事件的信息。
- ◆ “evprint subnet1.node0.CPU”，表示显示事件表中“subnet1.node0.CPU”模块的所有事件。
- ◆ “evprint subnet1.node0.CPU +3”，表示显示事件表中“subnet1.node0.CPU”模块的后 3 个事件。
- ◆ “evprint @30.5”，表示显示事件表安排在 30.5s 出执行的事件信息。
- ◆ “evprint @30.5 34.0”，表示显示事件表安排在 30.5~34.0s 之间执行的事件信息。
- 可以用“icprint_ev”显示当前事件的相关 ICI 内容

事件信息中包含两个标识号，一个是执行标识号，一个是调度标识号。执行标识号表示事件最终被执行时，按执行的先后次序编的号，而调度标识号是按事件进入事件队列的先后次序编的号，所以在一次仿真中，调度标志号是固定的，而执行标识号是可变的。

4.1.2.1.2、动画输出窗口

在调试输出窗口中 Model，进入动画演示窗口。在动画演示页面，根据左侧对象浏览树中选择的模块所属层级显示网络层级。在左侧选择子网下某节点，动画演示窗口显示子网模型内的包交互动画。如图 4-9 所示。

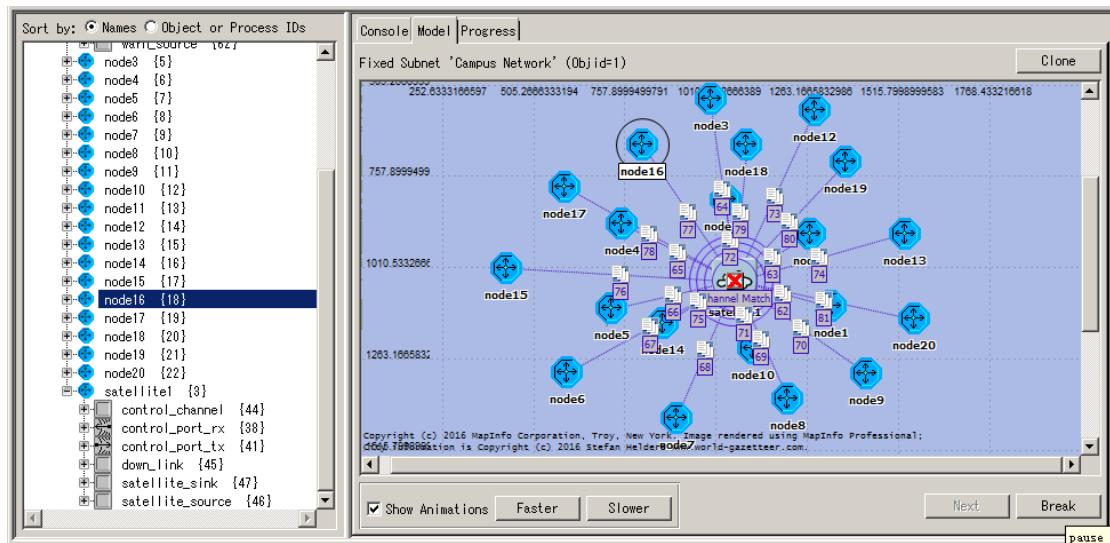


图4-9. OPNET 调试子网模型动画演示

在左侧选择某节点下模块，动画演示窗口显示节点模块内的包交互动画。如图 4-10 所示。

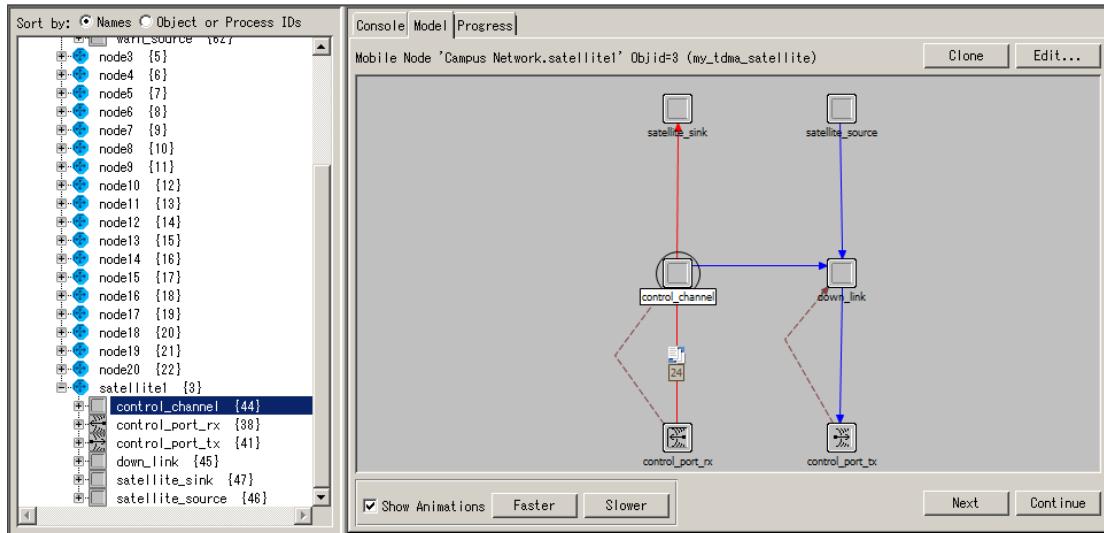


图4-10. OPNET 调试节点模型动画演示

在左侧选择某处理器模块或队列模块下的进程，动画演示窗口显示此进程模型的状态转移变化情况。如图 4-11 所示。

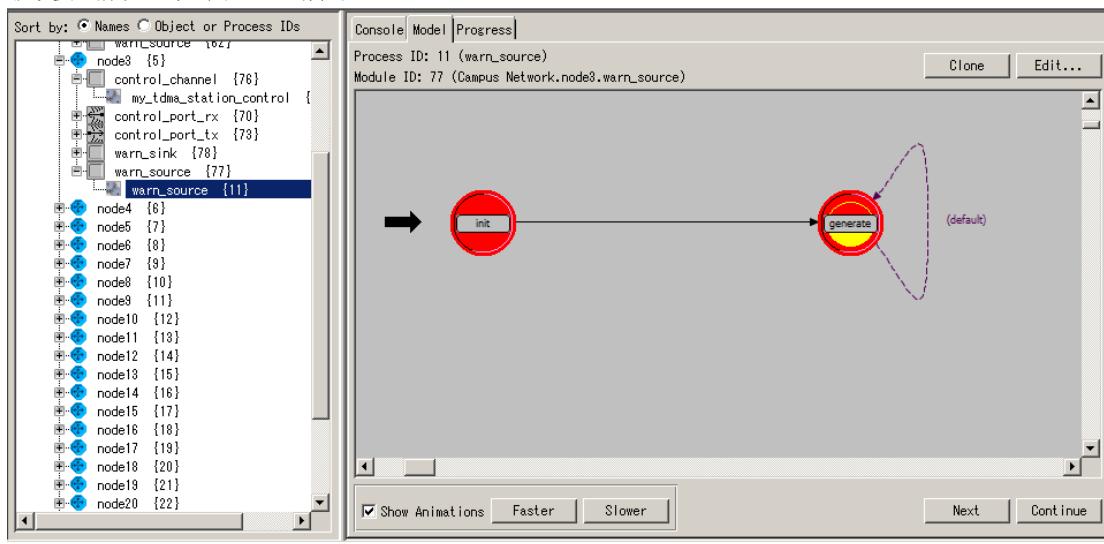


图4-11. OPNET 调试进程模型动画演示

在动画演示界面，只有选中 Show Animations 才能运行动画。点击 Faster 和 Slower 分别时加快和减速动画演示效果。如果不选中 Show Animations，仿真核心则自动以在控制台的转态快速运行。演示动画以后，仿真的运行速度将变的非常缓慢。在调试和包跟踪时，我们往往通过观看包交互动画来了解协议的运行以及查找协议问题。

在动画演示阶段需要注意的由于动画演示是基于时间的，而 OPNET 建模仿真是基于事件的。所以动画演示和事件仿真并不是非常接近。比如，在节点 A 向节点 B 发送数据包，节点 B 接收到以后返回另一个数据包。在 B 节点内部模块看已经接收到了 A 节点发来的数据包，并已经开始返回另一个数据包。而在子网模型中看到数据包还没有完全被 A 节点接收到，就已经开始发送了数另一个数据包。这是正常现象。

4.1.2.1.3、仿真进度窗口

在调试界面中点击 Progress，显示进程进度界面，在这里与使用 Optimized 仿真核心类型具有类似的效果。可以显示仿真事件，事件数目，仿真错误等。在控制台输出界面同样显示仿真错误提示。

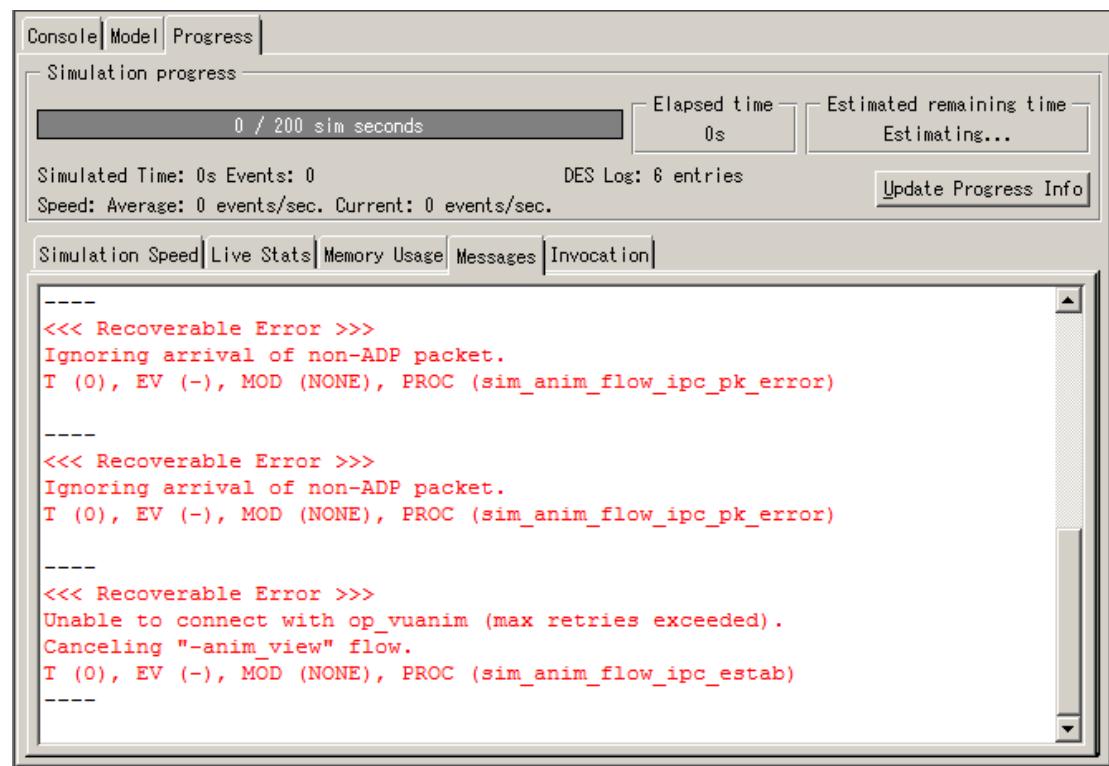


图4-12. OPNET 调试仿真进度及错误信息

4.1.2.2、记录窗口

记录窗口位于调试窗口界面的左下角，包含属性记录窗口、数据包记录窗口、事件记录窗口。

点击 Attributes 进入属性记录窗口。在对象浏览树中选中不同的模块对象，在属性记录窗口会显示该对象当前属性的值。通过这个可以实时看到对象属性的变化。如图 4-13 所示为 node4 节点的属性。

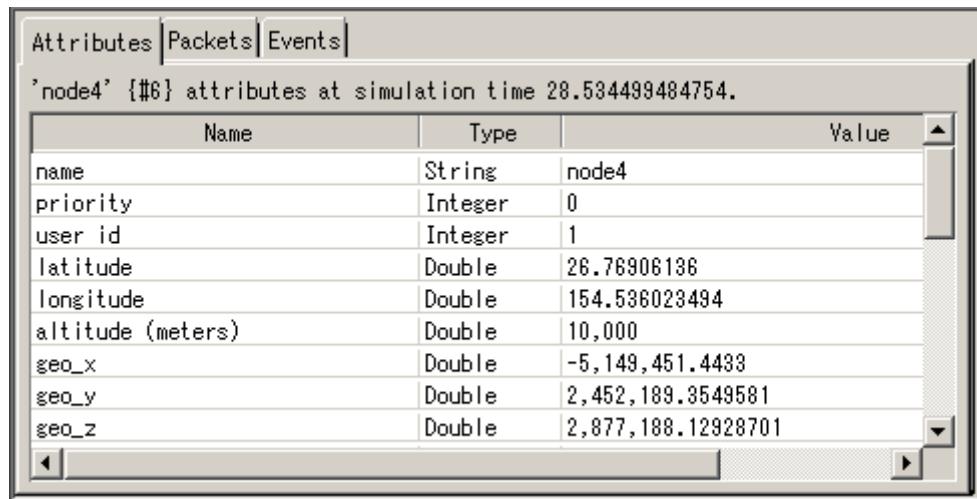


图4-13. OPNET 调试对象属性记录窗口

点击 Packets 进入数据包记录窗口，如图 4-14 所示。在数据包记录窗口可以根据 Show 框内显示设置，选择性显示数据包信息。

- “All” 为显示所有数据包。数据包的信息包括数据包 ID, 树 ID, 数据包格式, 创建者, 拥有者, 创建时间, 批量数据大小, 数据包总大小, ICI 的 ID 等信息。相当于命令 “pkmap all”
- “With Tree ID” 为枚举属于指定数据包树的数据包，相当于命令 “ptmap <pk_tree_id>”。
- “Owned by” 表示枚举由标识号指定的对象所有的数据包，相当于 “pkmap #<objid>” 命令。
- “Owned by Kernel” 表示枚举由核心所有的数据包，没有对应的控制台命令。
- “Created by” 表示枚举由标识号指定的对象创建的数据包，没有对应的控制台命令。
- “Up to” 复选框用于限制显示的条目数量。
- “Updata” 按钮用于按指定条件更新枚举结果

点击数据包，会在右侧 Packet Content 框内显示此数据包的字段域、属性域、TDA 信息。可用于观察此数据包的信息，验证是否符合预想结果。

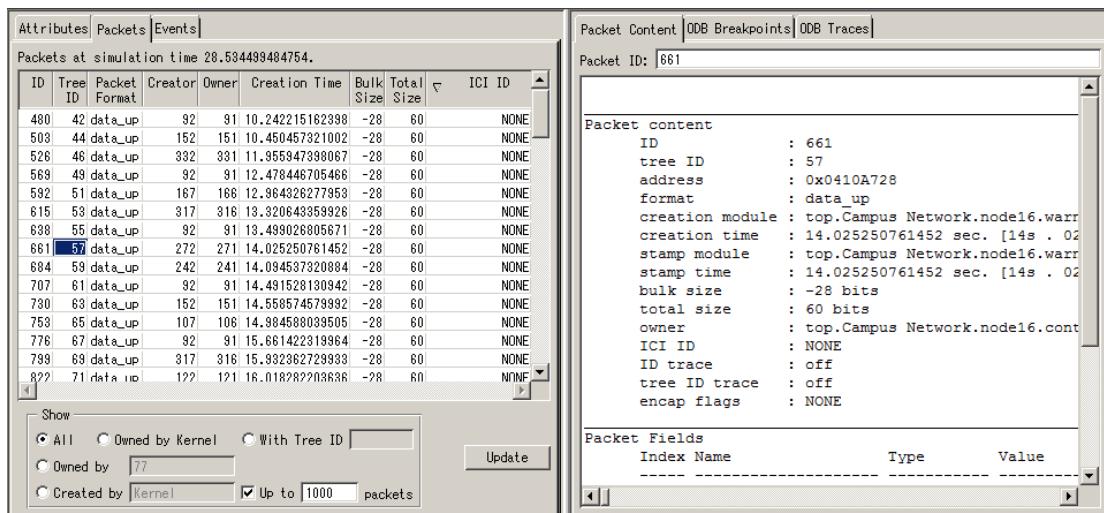


图4-14. OPNET 调试数据包记录窗口

点击 Events 进入事件记录窗口，如图 4-15 所示。在事件记录窗口可以根据 Show 框内显示设置，选择性显示事件信息。

- “All” 为显示所有事件。事件的信息包括调度 ID，源执行 ID，源模块 ID，目的模块 ID，中断事件类型，响应事件，ICI 的 ID，附加状态指针，中断码类型，中断码的值等信息。通过事件记录窗口可以掌握事件的顺序是否符合预想，以此来判断系统错误的原因。
- “Event type” 表示枚举事件表中指定类型的事件，没有对应的控制台命令
- “Source module ID” 表示枚举源对象标识号为指定值的事件，没有对应的控制台命令。
- “Target module ID” 表示枚举目的对象标识号为指定值的事件，没有对应的控制台命令。
- “Up to” 复选框用于限制显示的条目数量
- “Update” 按钮用于指定条件更新枚举结果。

The screenshot shows the OPNET Events window. At the top, there are three tabs: Attributes, Packets, and Events. The Events tab is selected. Below the tabs, a message says "Scheduled interrupts at simulation time 28.534499484754.". A table lists several interrupt events:

Schedule ID	Source Execution ID	Source Module ID	Target Module ID	Type	Time	ICI ID	State Ptr	Primary Data Type	Primary Data Value
#12152	12037	302	302	self_intrpt	28.864463640922	NONE	0x0	code	101
#12564	12450	182	182	self_intrpt	31.269700042393	NONE	0x0	code	101
#12566	12451	181	181	self_intrpt	28.736673313067	NONE	0x0	code	-10,006
#12681	12566	136	136	self_intrpt	28.859186033727	NONE	0x0	code	-10,004
#12687	12569	106	106	self_intrpt	28.859462250626	NONE	0x0	code	-10,005
#12700	12586	62	62	self_intrpt	32.851608957032	NONE	0x0	code	101
#12702	12587	61	61	self_intrpt	28.945040435836	NONE	0x0	code	-10,002

Below the table, there is a "Show" section with the following settings:

- All Event type: access_intrpt
- Source module ID: Kernel
- Target module ID: 77
- Up to 1000 events
-

图4-15. OPNET 调试事件记录窗口

4.1.3、调试技巧

下面介绍两种调试技巧。对于使用者，进行 OPNET 仿真时的主要步骤包括以下几步。

第一步为场景搭建，包括子网域，节点域与进程域的模块拖放、对象连接、属性设置等。这一步骤根据使用者具体的场景来设定。

第二步为协议编程，主要为进程模型代码的编写和链路模型代码的编写。并进行进程模型代码编译，保证语法无错。

第三步为逻辑调试，这是最消耗时间的。因为 OPNET 并不提示错误，但是却系统的运行却不符合预想，由不知从何下手。

第四步为仿真结果的处理。在 3.7 统计量章节进行了介绍。

下面我们介绍进行模型语法编译和仿真协议的逻辑调试两个主要内容。

4.1.3.1、语法调试技巧

进程模型和链路模型函数的代码调试主要语法为主，相较于仿真中的逻辑调试较为简单。但是由于 OPNET 提供的使用  按钮进行调试的方法，弹出的错误并不一定是关键问题，也并不一定是最关键的错误问题。所以下面较少一种作者常用的语法错误调试方法。在读者无法确定语法错误原因时可以使用。

在介绍前，请先确认一般性错误原因，包括没有开启 C++ 编译，使用了中文的分号，逗号引号，语句后没有添加分号结尾等。在 OPNET 中一个中文占两个数据位，需要按两次删除键才能删除一个中文，如果只删除了一个也会报错。

下面距离较少一种比较本的方法。例如确定某状态代码中是否有错，及错误位置。状态代码如下：

```
语句 1;
语句 2;
语句 3;
...
语句 i;
...
语句 n;
```

首先再代码首尾添加注释符 /* */，来注释所有代码，编译看是否有错误，判断此状态是否有错。

```
/*语句 1;
语句 2;
语句 3;
...
语句 i;
...
语句 n;*/
```

如果添加注释后，编译代码不再有错误则此段代码包含错误的地方，下一步缩小注释范围，重新编译，一次来找到出错的语句。

4.1.3.2、逻辑调试技巧

在语法调试无误后需要进行系统仿真的逻辑调试，在 OPNET 中最困难也是最耗时间的莫过于逻辑调试。最实际有效的莫过于在代码中添加较多的 printf，及时输出你想了解的变量的值。通过调试窗口打印输出的内容，寻找不符合预想的地方。作者这里也介绍一种查找错误地方的方法。

首先使用 Optimized 调试模式进行仿真，仿真中会中断出现错误的时间和事件数，例如错误的时间为 5.123s，时间为事件 2543。

关闭调试窗口，重新使用 Development 重新打开仿真界面，先不要点击 Continue，在命令输入框中输入 evstop 2543，表示设定事件断点在事件 2543 处，输入 con 回车，仿真程序自动运行到 2543 前的事件，输入 fulltrace 回车表示详细跟踪。输入 next 回车，会出来下一步的详细运行步骤，一直 next 直到仿真出错中断。查看详细的运行打印信息寻找出错的地方。同样读者可以使用时间参数设定中断位置为 tstop 5.123。

4.2、动画

动画的观看在调试中已经有过介绍，在 ODB 调试模式中的调试输出窗口，选择 Model 可以很方便的观察数据包的交互或状态转移。当需要回看某些动画时，读者除了可以重新仿真外，也可以讲动画文件存储下来，使用 OPNET 的动画播放观看。

首先在子网模型界面菜单栏点击 DES，选择 Choose Statistics (Advanced) 进入探针模型窗口。如图 4-16 所示，其中 Global Statistic Probes 全局统计量探针，这里显示之前我们在进程模型中设置的全局统计量。Automatic Animation Probes 为自动动画探针。点击菜单栏 Objects 下的 Create Automatic Animation Probes 创建自动动画探针。如图 4-16 所示，已经设置了 pb2 探针。

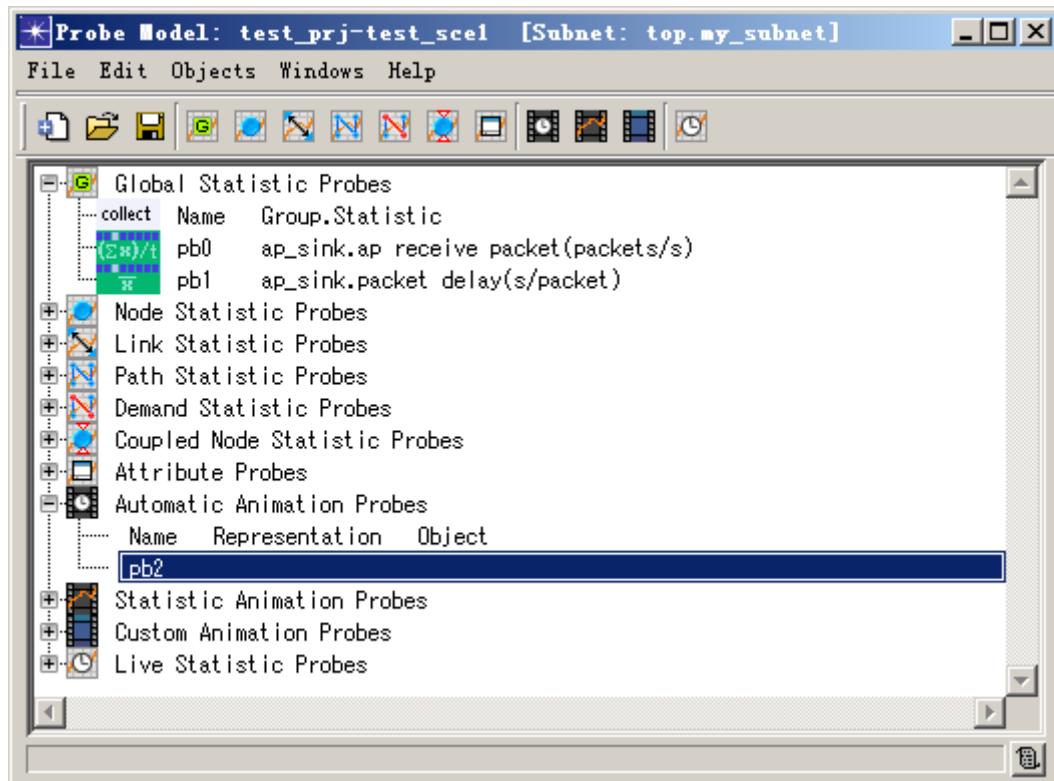


图4-16. 各类探针设置窗口

在创建的探针上单击右键，出现 Edit Attributes 和 Choose Probed Object 两个快捷选项。先选择 Choose Probed Object 进行探针对象选择，再选择 Edit Attributes 编辑探针属性。

先选择Choose Probed Object进行探针对象选择。

再选择Edit Attributes进入属性编辑，设置representation属性的值

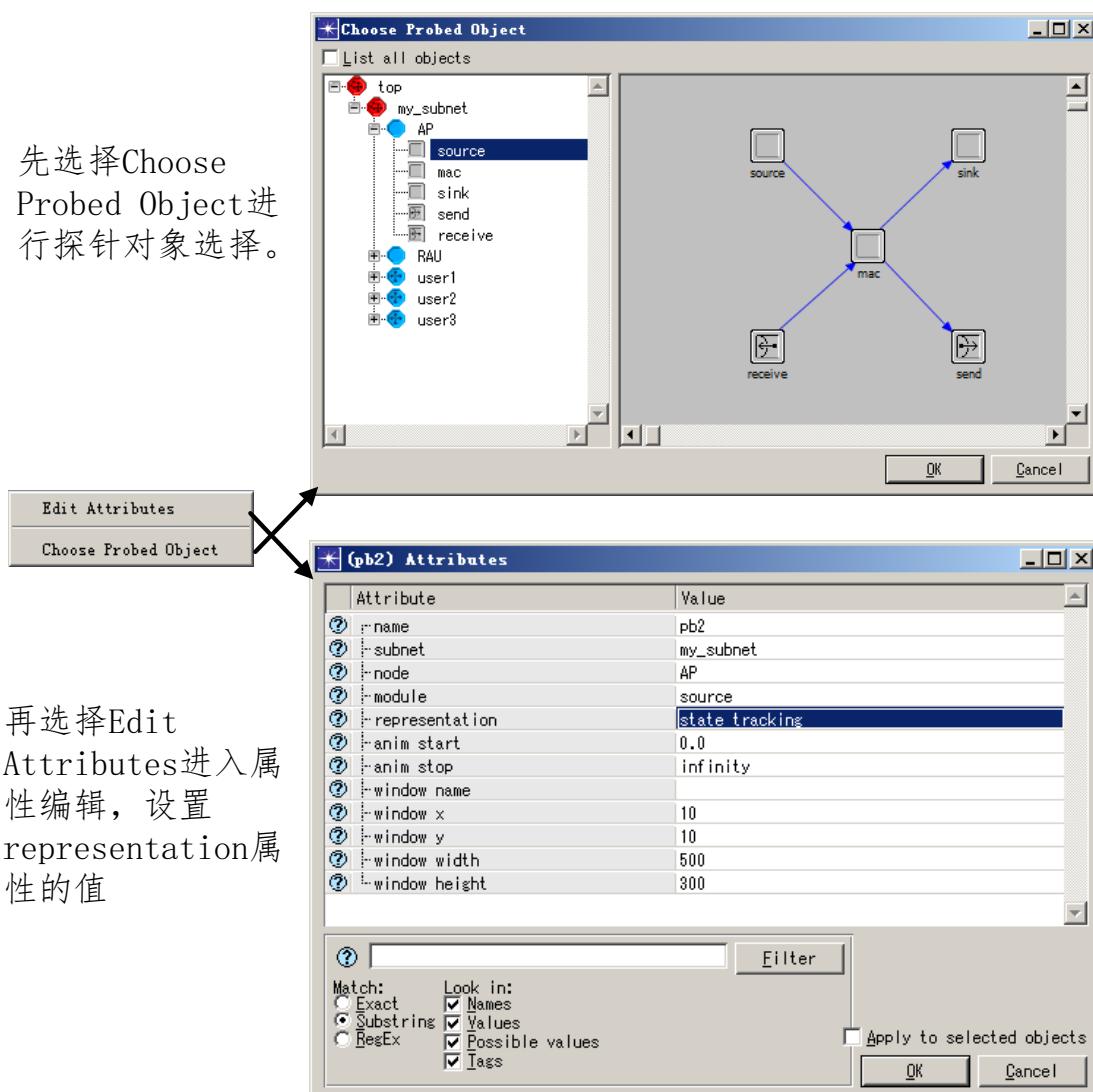


图4-17. 动画探针设置过程

设置完探针后，保存。点击子网模型 DES 菜单按钮选择 Configure/Run Discrete Event Simulation，进入仿真设计界面，在 Outputs-Animation-2D 下选中 Sent animation to history file，设置保存仿真动画到文件，单击 Run 运行仿真程序。

运行完仿真程序后，点击子网模型 DES 菜单下 Play 2D Animation 会弹出动画演示窗口。点击动画演示窗口菜单栏的 File-Open，选择自己的动画，即可以查看之前存储的演示动画了。如图 4-18 所示。

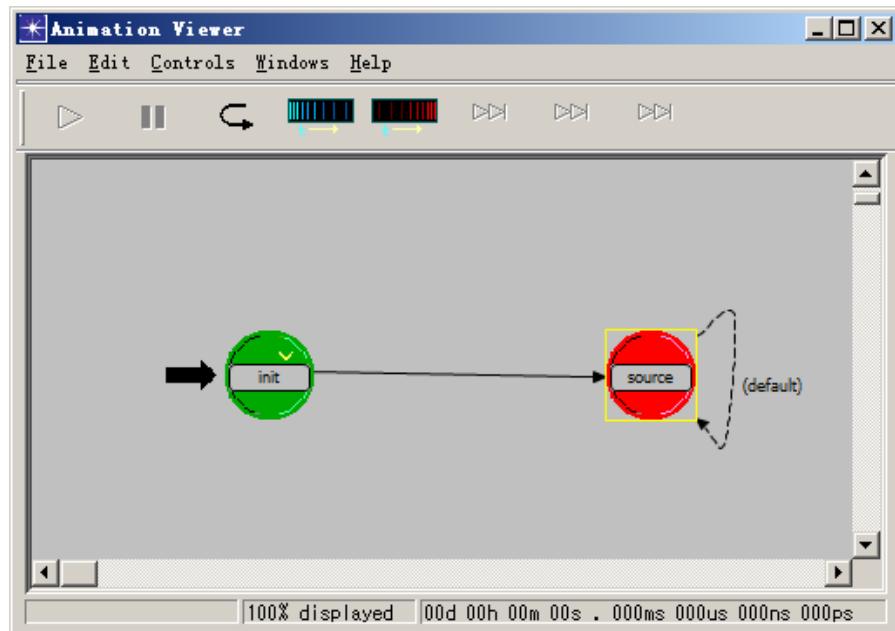


图4-18. 记录动画演示窗口

第五章 常见问题及错误

5.1、常见错误

1、目的模块使用 `op_pk_get(op_intrpt_strm())` 获取的数据包不是源模块当前发送过来的数据包，而是源模块之前发送过来的数据包。

解答：

`op_pk_get(op_intrpt_strm())` 是从当前流中断端口获取数据包，但是这个端口可能排了很多数据包，此函数只是获取缓存中的第一个数据包，并不是获取当前来的数据包。

2、源模块发送数据包，目的模块没有接收到流中断。

解答：

数据包的发送中断模式有多种，静默发送不产生流中断，而且调度发送的流中断优先级不高，相同时间高优先级事件先执行。

3、正在接收数据包时，打印输出接收功率的值为 0

解答：

接收功率以 w 为单位，数值为 double 类型，取值非常小，需要打印输出到小数点后 20 位才能精确输出。

4、能否在进程模型中做并行进程

解答：

一个进程从初始状态开始按顺序执行，在进程中可以创建子进程，各进程之间并行工作。一个进程模型只有一个进程，对应一个进程模型文件。一个模块属性设置中的进程为根进程，根可创建多个子进程并行运行。

5、模块间发包太快造成目的模块接收不到流中断。

解答：

6、进程模型中循环执行一个状态、仿真开启单不运行，仿真死机

解答：

(1) 在一个阻塞态中的入指令添加了在当前仿真时间自中断。

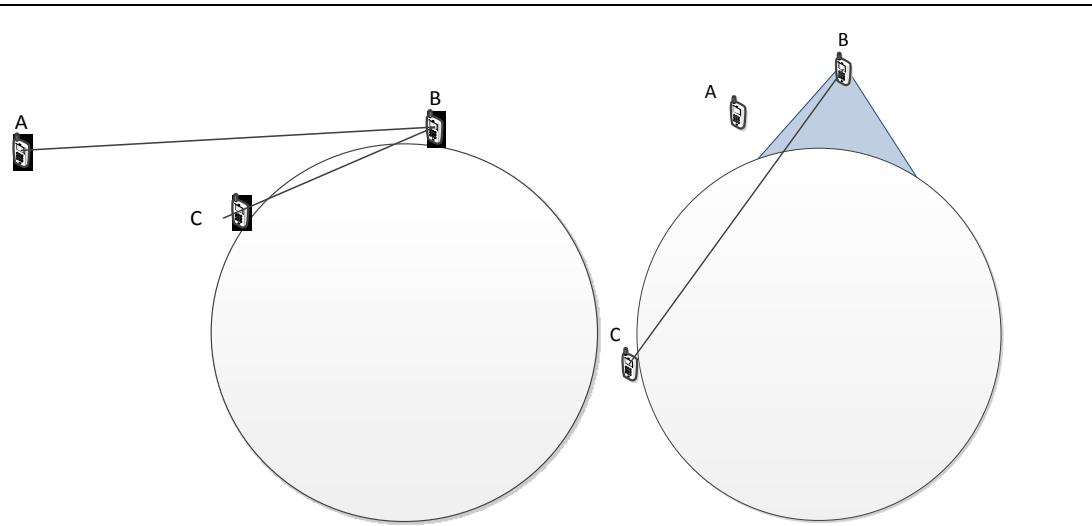
`op_intrpt_schedule_self(op_sim_time(), 中断码)`；然后根据状态转移指向状态又跳转到本身，就会出现循环执行自中断的情况。或者自中断的延迟时间为随机变量，变量为 0 了。

(2) `while` 或 `for` 无法跳出循环

7、在远距离通信时，相同的节点属性，近处节点无法通信，远处节点能通信。

解答：

当管道模型，链路闭合参数选择不可穿透地球时，发送功率足够大，可能存在左图所示的情况，当链路参数选择可穿透时，又使用了定向天线，存在右图所示的情况。



8、收发信机参数不匹配还存在干扰

不同节点的收发信机，或者同一节点内的收发信机，只要覆盖频率范围有重叠就会存在干扰，就会影响数据包的接收。

9、Process model (xxx) is undeclared

建立了子进程 xxx，但是没有在子进程里声明，所以造成了错误。

5.2、常见问题

1、OPNET 中的函数 FIN, FRET 以及 FOUT 都是什么功能？

为了使一个用户定义的函数被执行，该函数必须与一个特殊的堆栈跟踪代码相连。堆栈跟踪技术靠在函数的入口点和出口点插入预处理器宏指令完成（一个函数 只有一个入口点，但可以有多个出口点（由 C 语言的 return 声明决定）。这些宏指令为：FIN、FOUT 和 FRET。FIN 被插入到函数的入口点，FOUT 被插入到函数的出口点，但却不返回任何值，FRET 被插入到函数的出口点，返回一个值。注意这些宏指令不需要以分号结束（它们自我包含），FIN 的参数中也不需要双引号。

opnet 提供的所有的示例模型都包含了这些宏指令，并且建议用户定义的函数也包含这些宏指令。如果 FIN、FOUT、和 FRET 被正确插入了用户代码中，我们就可以使用 op_vuerr 来找出程序错误的位置，哪怕是在一个嵌套的模型函数调用中。

2、Objid 和 int 的区别？

Objid 是系统分配的，全局唯一的整数。int 变量是自己可以设置的，可以不唯一。在使用时可以混合使用，都是整型

3、如何将模块添加到 OPNET 中，被 OPNET 识别？

edit-> preferences -> mod_dirs 添加模块路径即可。

4、为什么每次新建一个 project 都给我保存在 c:\op_models 目录下，我想换一个地方，怎么设置？

edit -> preferences -> mod_dirs 中，新建一个路径，并作为第一路径即可。

5、想查找一个变量的使用场合，包括不同 process, 不同 node 中的 header 和 function，如何做？

在 OPNET 中变量是在一个 PROCESS 中存在的。不同的 PROCESS 之间则是通过进程之间

的通信机制来共享信息的。因此你查找变量的作用范围应该是在一个 PROCESS 内的。编译后每个 PROCESS 会产生一个 C 或 C++文件。在那个文件里就可以查到变量的应用地方。不同的进程可能具有相同的属性。而为了减少节点的属性数目可以采用 rename/merge 属性的方式。这时这些属性具体对应到各个进程的哪个属性可以通过节点接口菜单下的 rename/merge 按钮下找到找到。

6、OPNET 怎样将图导出来？

(1) 保存子网场景：在子网场景编辑界面，选择菜单栏 scenarios-generate scenario bitmap 出现保存设置界面。

(2) 保存节点模型或进程模型：在节点模型或进程模型编辑界面，点 file-export to bitmap 出现保存设置界面。

(3) 保存分析曲线：对于分析出来的曲线，按鼠标右键，其中有个 Export Graph Data to Spreadsheet，然后会有提示 说你文件保存在什么地方，一般缺省是保存在 c:\op_admin\tmp 目录下。文件你可以用 UltraEdit 打开来看，是两列数据，一列是仿真时间，一列是仿真数据，然后你就可以想用什么工具画图就无所谓了。

7、opnet 中关于时延的问题。

数据速率是用来和包长结合计算传输时延的，而“delay”属性是用来描述电波的传播时延的。在点到点链路属性里，“delay”就是总传播时延；在多点链路里，“delay”指单位距离的传播时延。用户可以修改传播时延的计算方法，“Distance Related”表示在自定义的传播时延 pipeline stage 里基于距离计算传播时延。

8、在 opnet 中关于统计一些速率方面的参数。

统计流速率的时候，首先应该在 Local Statistics 中将这个统计项的 Capture Mode 设成 sum/time，然后在程序中每次收到一个数据包，就将这个包的长度 len 写入，比如 op_stat_write(handle, len)，随后再马上调用一个 op_stat_write (handle, 0) 来结束这次写入，就可以了。

9、用 VC 调试的时候，state variable 的值无法看到，怎么办？

用 op_sv_ptr 这个指针。它指向了所有的状态变量。

10、关于消耗时间和仿真时间？

一个是仿真程序运行的时间，反映仿真程序执行的速度。而另一个是所仿真的系统的时间进度，反映当前的仿真执行的进度。仿真时间的修改是通过事件的发生来进行的。譬如说你在 0s 时作一件事持续时间为 5 秒，5 秒钟结束后会触发一个事件，这个事件将系统的仿真时间改为 5s。你使用 OPNET 的模型，它在接收到事件时会进行相应的仿真时间的更新。而你自己也可根据需要更新仿真时间。你采用 op_intrpt_schedule_self(op_sim_time() + 延迟时长，中断码)，就可以在当前时刻的所需的时间以后产生一个中断，从而触发一个事件，系统的仿真事件也就被更新为此时间。OPNET 中数据的收集方式是可选的，可以选择为逐点的，也可以选择按照漏斗进行平滑的根据自己的需要而定。

11、OPNET 运行时无法进行 C/C++代码编译的解决办法？

当你出现这种情况时，OPNET 总是提示说 comp_msvc 不能执行，因为 Visual C++ 没有正确安装，这时你需要修改系统的环境变量。参照 1.2 章节设置环境变量。在进程模型中点击 compile-enable C++ code generation。

12、请问 opnet 中的移动台的 trajectory (即运动轨迹) 能否用一个专门的代码来生成，而不是用鼠标事先画出？如何实现 opnet 与此段代码的交互？

要做到运动轨迹的交互性，你得修改一些 process 和 pipeline。运动的结果无非是和基站的距离变化，然后利用衰落模型得到 snr，ber 等参数，所以你可以修改

dra_propdel.ps.c, umts_ue_dra_power, umts_dra_snr 等 process. 如在 dra_propdel.ps.c 中把 start_pro_distance用自己的距离函数代替即可。

13、OPNET 关于字符数组？

OPNET 中使用字符串代替名称获取时，字符串数组的最后一个必须是\0，否则容易出错。

14. 请问 OPNET 里如何提取统计信息作为反馈控制变量？例如将丢失率提取出来后，通过 函数将其反回馈回模型中进行控制。

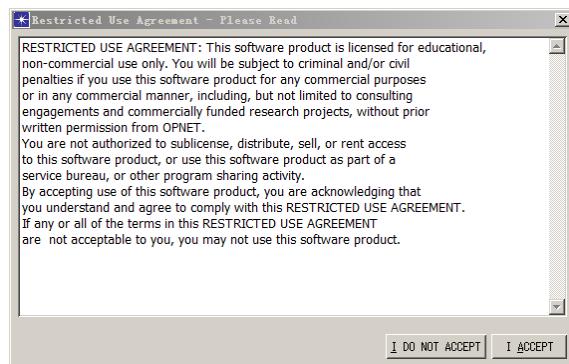
可以试试 stat_intrpt 函数。？

15. 怎样在其他模块获取在 pipeline stage 中计算的某些参数的数值，如接收功率的数值？

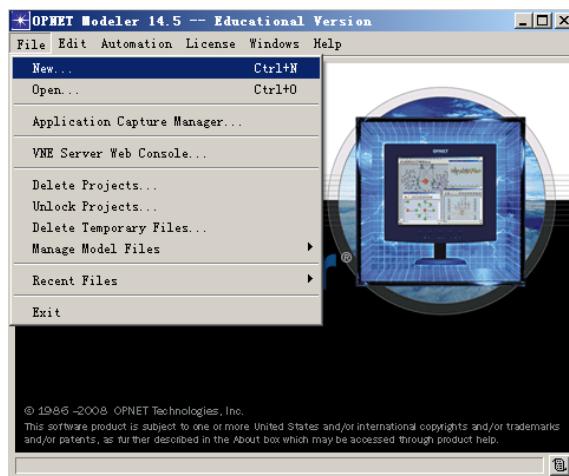
可以读取数据包 TDA 属性，用 pwr = op_td_get dbl (pkptr, OPC_TDA_RA_RCVD_POWER) 获取。

实例 1：创建项目与场景

(1) 打开 OPNET 运行程序, 选择 I ACCEPT



(2) 点击 File 按钮, 选择 New 选项



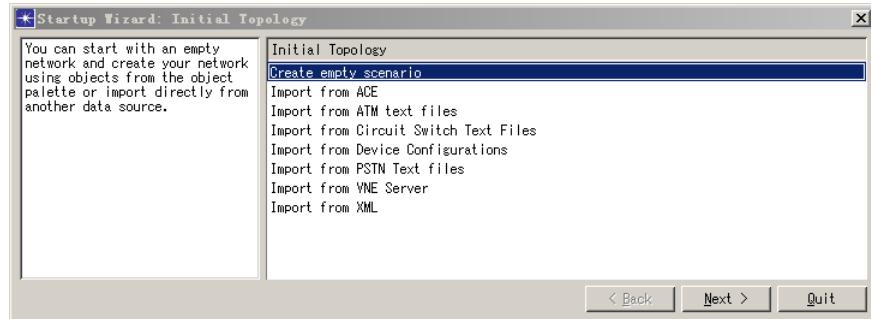
(3) 在创建内容中选择 Project (在不同界面默认选项不同), 点击 OK 按钮



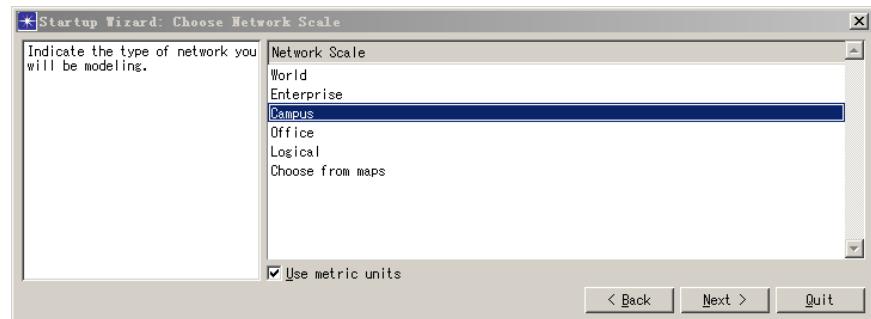
(4) 输入项目名称和场景名称, 点击 OK



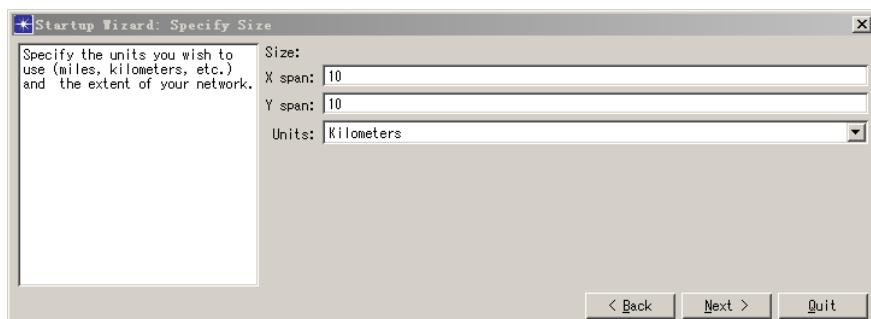
(5) 选择创建一个空场景, 点击 Next 按钮。



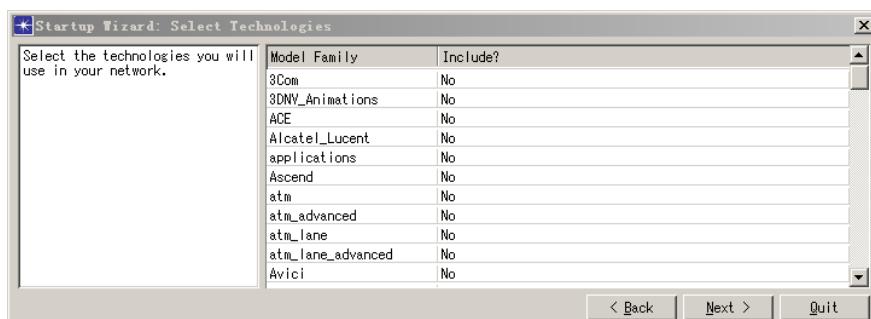
(6) 选择 Campus 网络比例，点击 Next



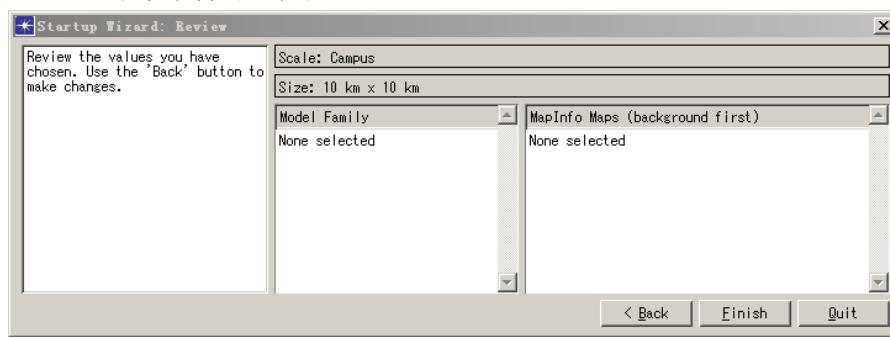
(7) 设置网络大小，采用 10km*10km 的网络，点击 Next



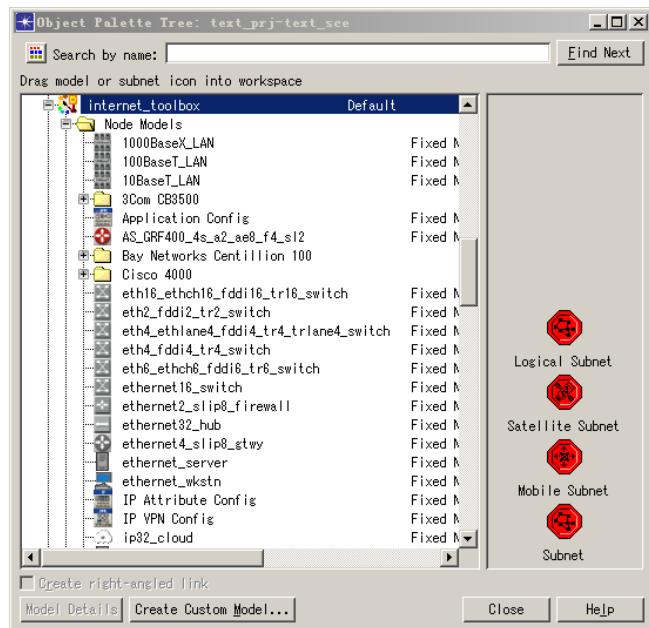
(8) 选择包含模型库，这里先不选择，点击 Next



(9) 点击 Finish，完成项目创建。



(10) 完成后会自动弹出添加对象的窗口，关闭此窗口即可。

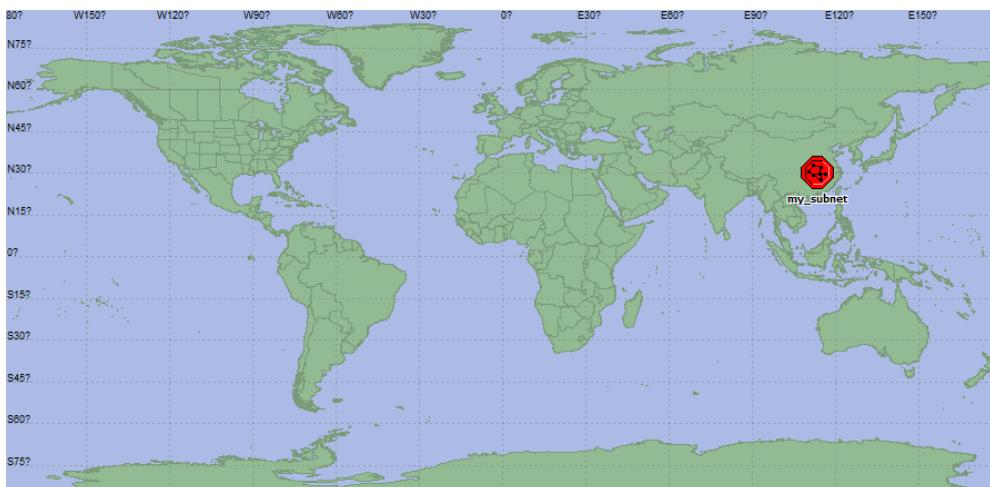


(11) 选择 File-Save，将 prj 文件存放在本项目的文件夹下，覆盖原有 prj 文件即可。到此一个空的场景创建完毕。

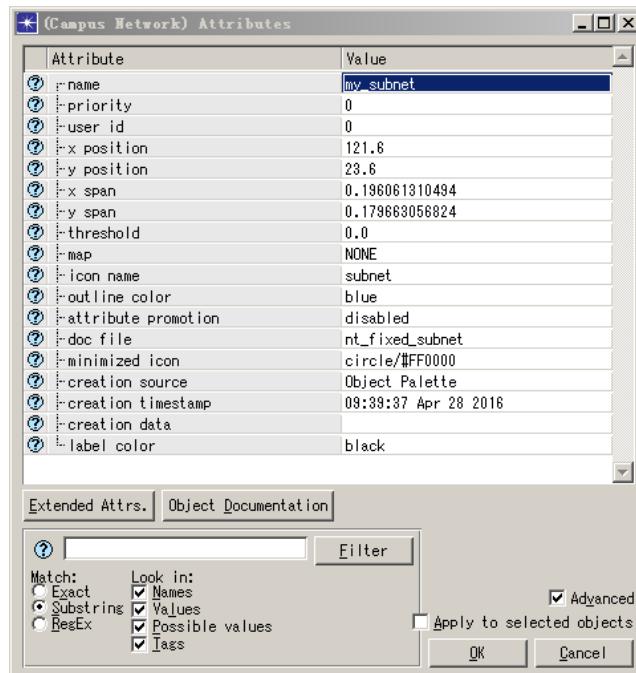
实例 2：分布式子网搭建

在实例 1 的基础上，本实例实现 20km*20km 的分布式子网搭建。

(1) 点击  按钮，回到 top 网，点击拖动鼠标，移动子网位置到中国海域。

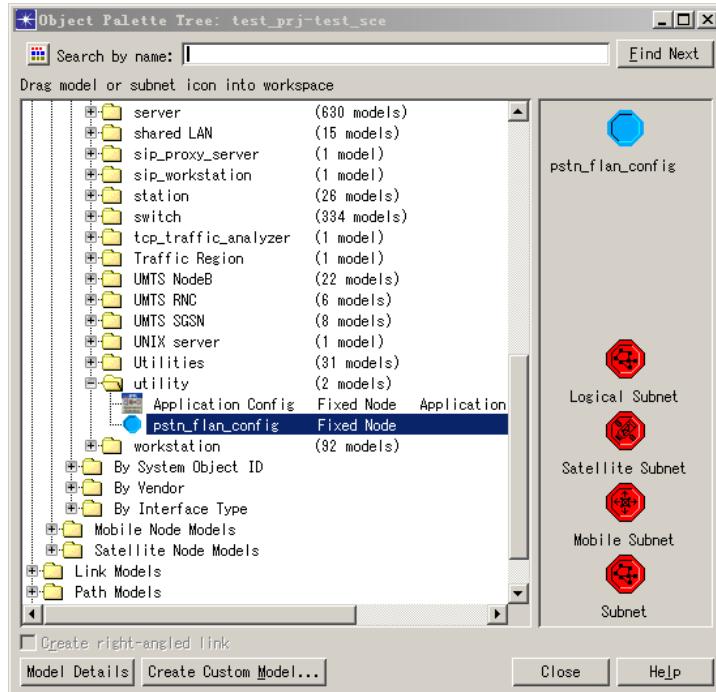


(2) 右键单击子网模块，选择高级属性编辑，进入属性编辑界面，编辑 name 值为 my_subnet，原子网大小为 10km，现在设置为 20km，将 x span 的值乘以 2，y span 的值乘以 2，点击 OK 按钮

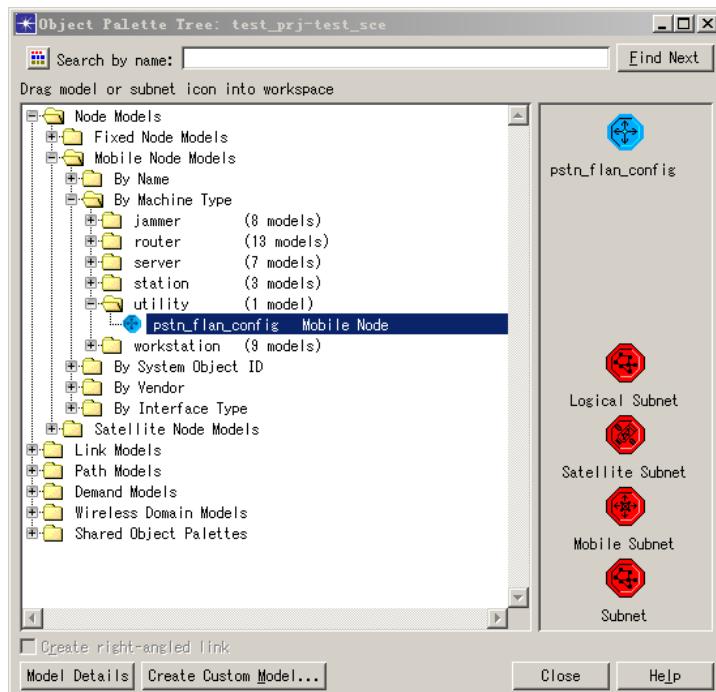


(3) 双击子网模块，进入子网场景布局。陆地背景为绿色，海洋背景为蓝色。

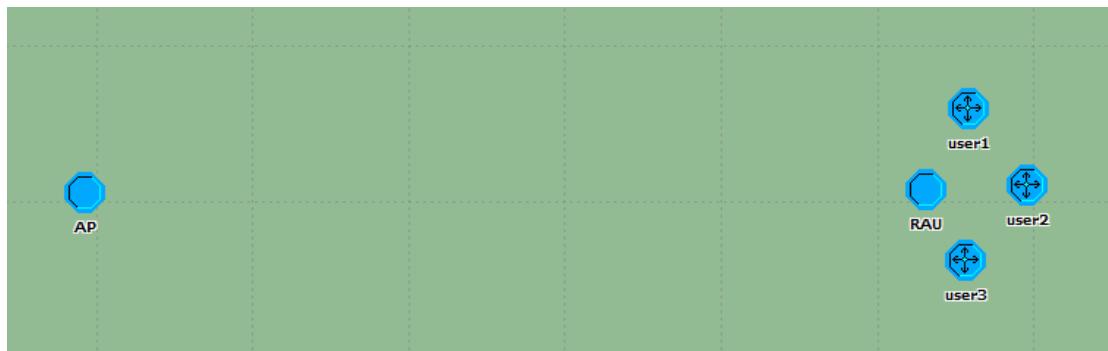
点击  按钮，选择模块对象，找到 Node Models-Fixed Node Models-By Machine Type-utility-pstn_flan_config，拖动对象图标到子网中。



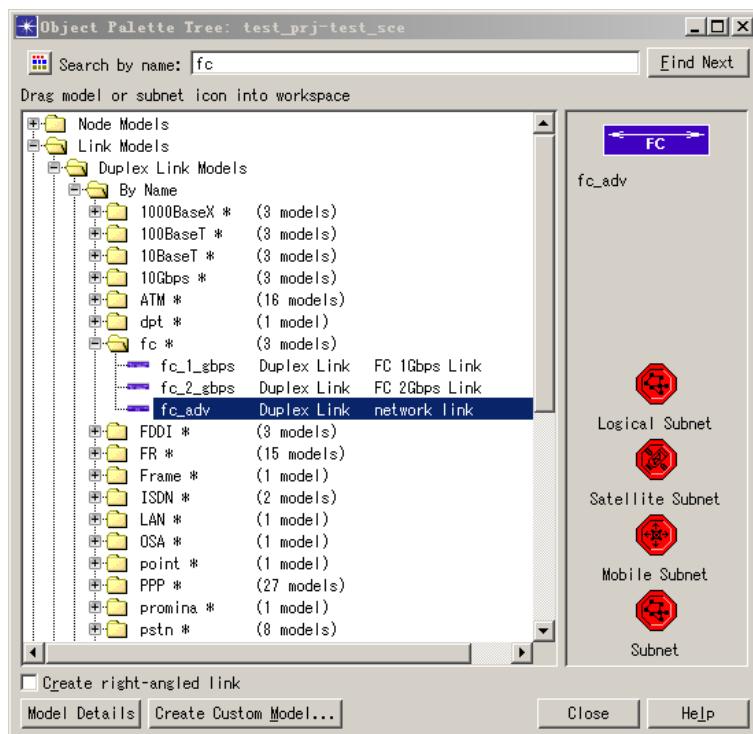
(4) 同样方法，找到 Node Models-Mobile Node Models-By Machine Type-utility-pstn_flan_config，拖动对象图标到子网中。



(5) 构建图中所示子网节点布局，右键单击节点，编辑节点属性，设置名称如图中所示。每个节点的海拔为 0.02。这里设置海拔是因为使用默认的链路闭合管道阶段函数时，每个节点分布在不同位置，海拔均为 0 时，节点间链路被地球阻拦，不可直接通信，所以设置海拔为 0.02km，使节点间可视。



(6) 点击 按钮，选择模块对象，找到 Link Models-Duplex Link Models-By Name-fc*-fc_adv，拖动对象图标到子网中。连接节点 AP 和 RAU1 以及 AP 和 RAU2。



效果图如图所示



(7) 右键单击链路，选择 Edit Link Model，进入链路编辑窗口。在链路编辑窗口中设置链路支持的数据包格式为全部支持。

Attributes			
Attribute Name	Status	Initial Value	
financial cost	set	0.00	
line style	set	solid	
packet formats	hidden	all formatted, unformatted	
propdel model	set	dpt_propdel_bgutil	
role	set		
symbol	set	none	
tag	promoted		
thickness	set	1	

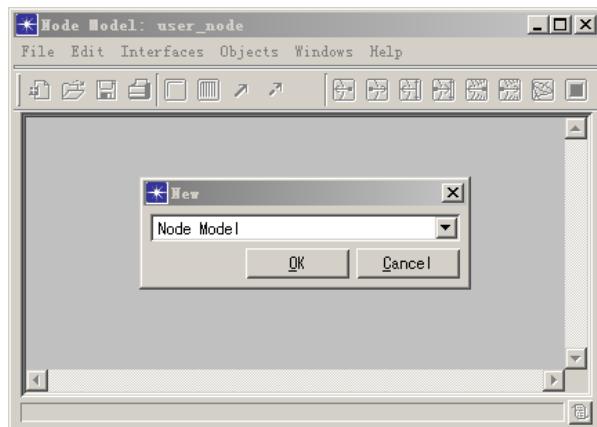
(8) 选择 File - declare external files, 进入外部文件声明, 选中 fc_supp, oms_gw, ams_basetraf_supp, ams_support_v2, apptrack_support, oms_basetraf, oms_data_def, oms_ext_file_support, oms_ot_support, oms_pipeline, oms_pr, oms_rr, oms_bgutil, oms_sim_attr_cache, oms_sv, oms_tan, oms_string_support, oms_dist_support。调用这些外部文件是由于光线链路模型中调用了一系列的外部函数。

到此子网配置设置完毕

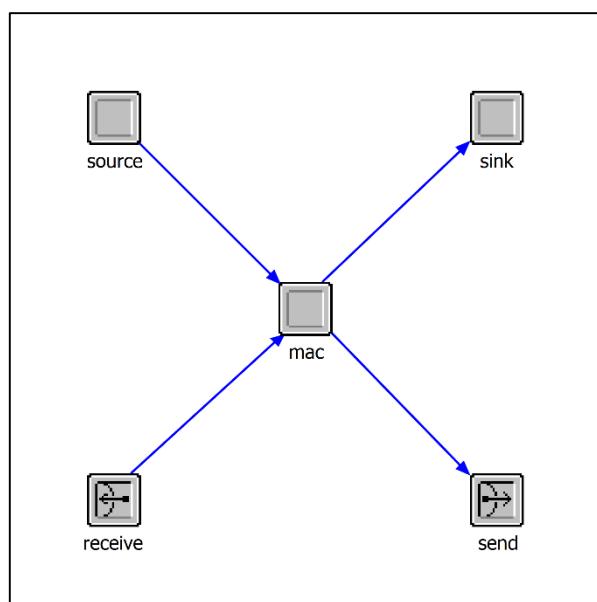
实例 3：为 AP 和用户创建节点模型

在实例 2 的基础上，实例 3 为 AP 节点、RAU 节点、用户节点创建节点模型。

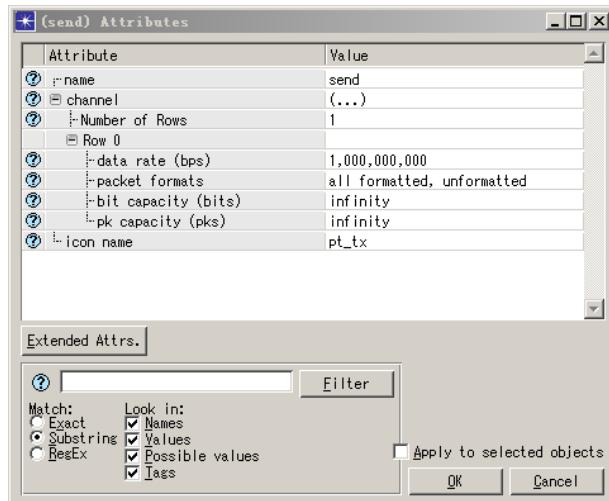
(1) 双击 AP 节点模块，进入 AP 节点模型设置界面，在节点模型界面点击 File-New 选择 Node Model 点击 OK，即创建了一个未命名的节点模型。



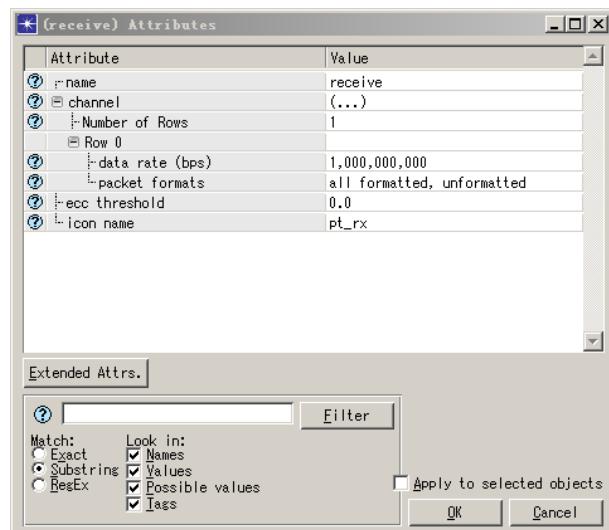
(2) 在新创建的节点模型中，按照如下图设置 AP 的节点模型



(3) 设置 AP 节点模型中点对点发信机模块属性如图所示



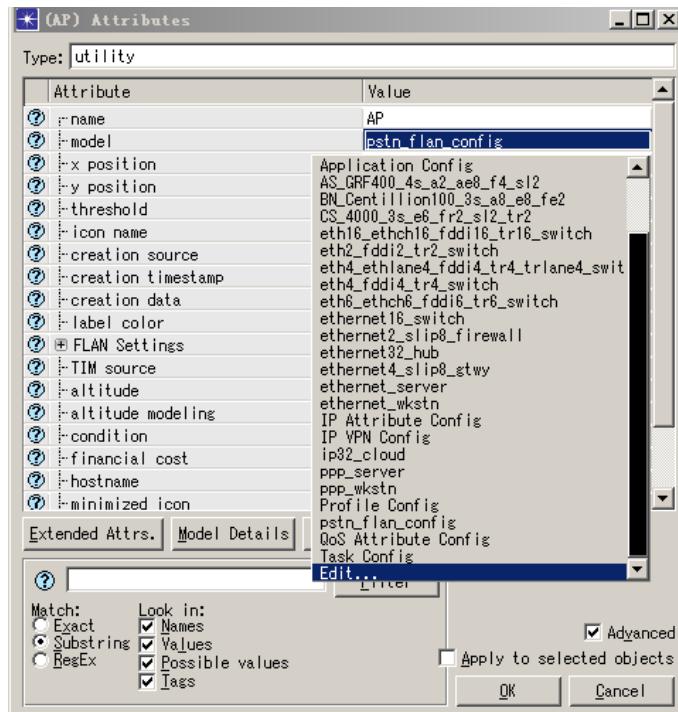
(4) 设置 AP 节点模型中点对点收信机模块属性如图所示



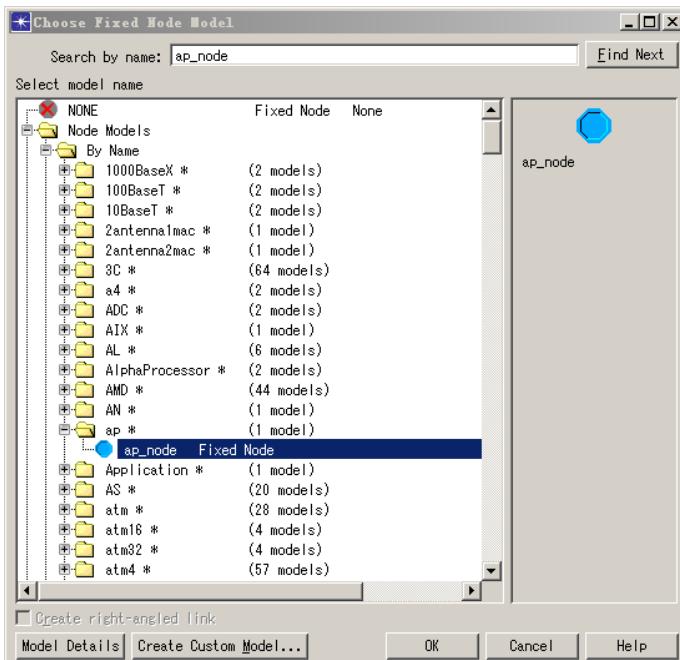
(5) 点击 File-Save，保存名称为 ap_node.nd.m 保存路径到项目文件夹外面，然后关闭项目，将 ap_node.nd.m 文件剪切到项目文件夹内。重新打开项目，右键单击 AP 节点模块，选择高级属性设置。



(6) 修改 model 属性值，选择 Edit...进入节点模型选择界面

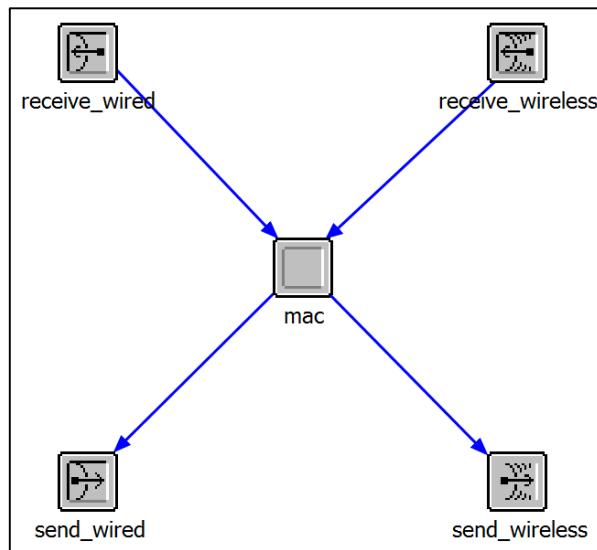


(7) 在节点模型选择界面，搜索 ap_node 选择 Find Next

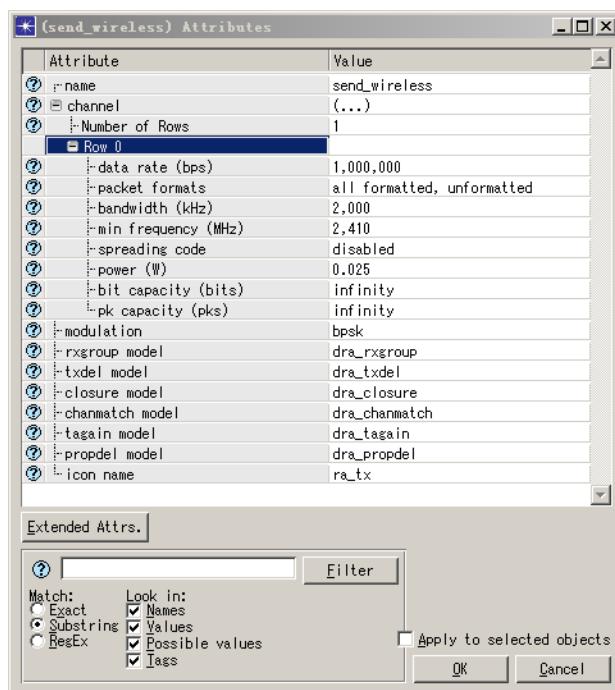


(8) 双击 ap_node，在属性界面点击 OK，则成功设置了 AP 节点模块的节点模型，双击 AP 节点模块即可进入刚刚创建的 ap_node 节点模型。

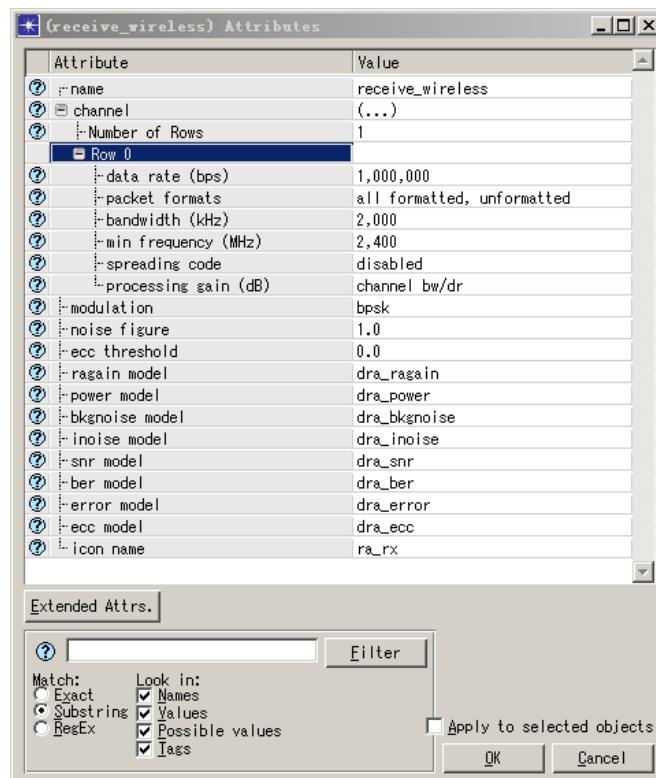
(9) 同样方法创建 RAU 的节点模型如下图所示



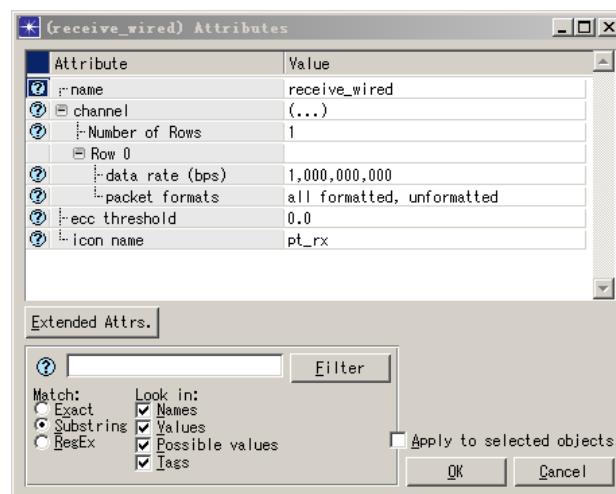
(10) 设置 RAU 节点模型中无线发信机模块属性如图所示。



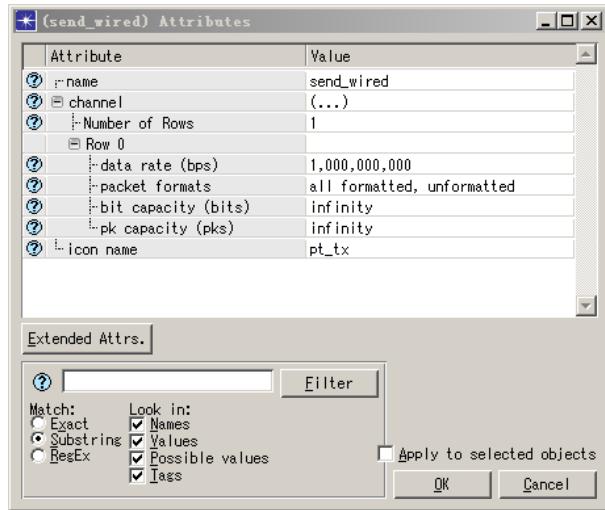
(11) 设置 RAU 节点模型中无线收信机模块属性如图所示。



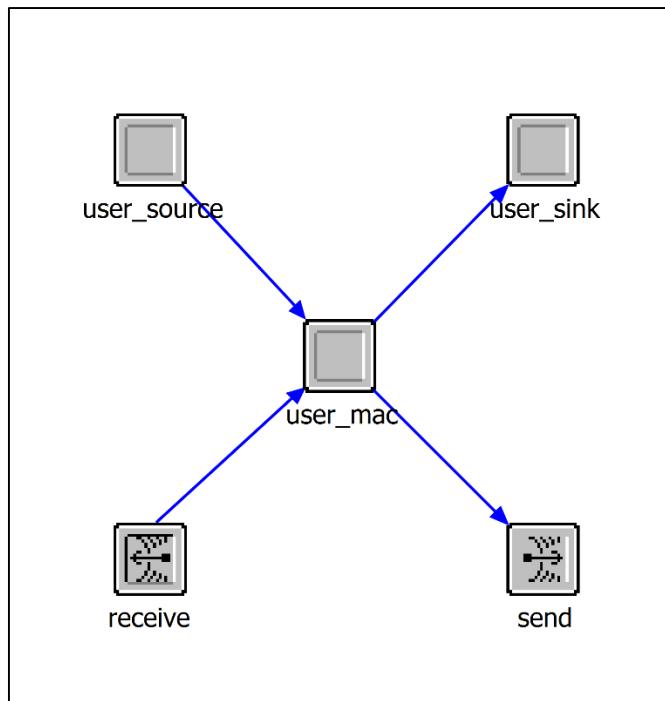
(12) 设置 RAU 节点模型中点对点有线接收机模块属性如图所示



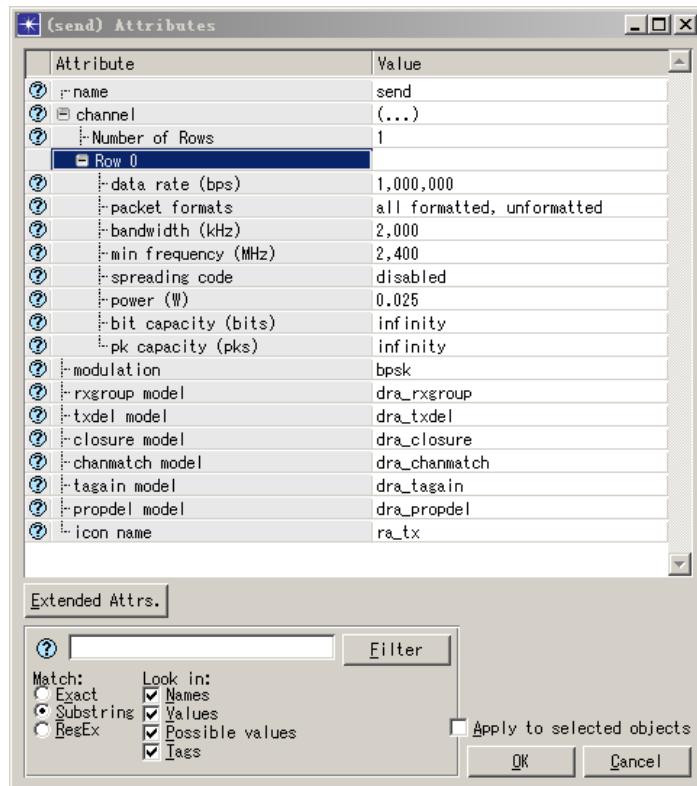
(13) 设置 RAU 节点模型中点对点有线发信机模块属性如图所示



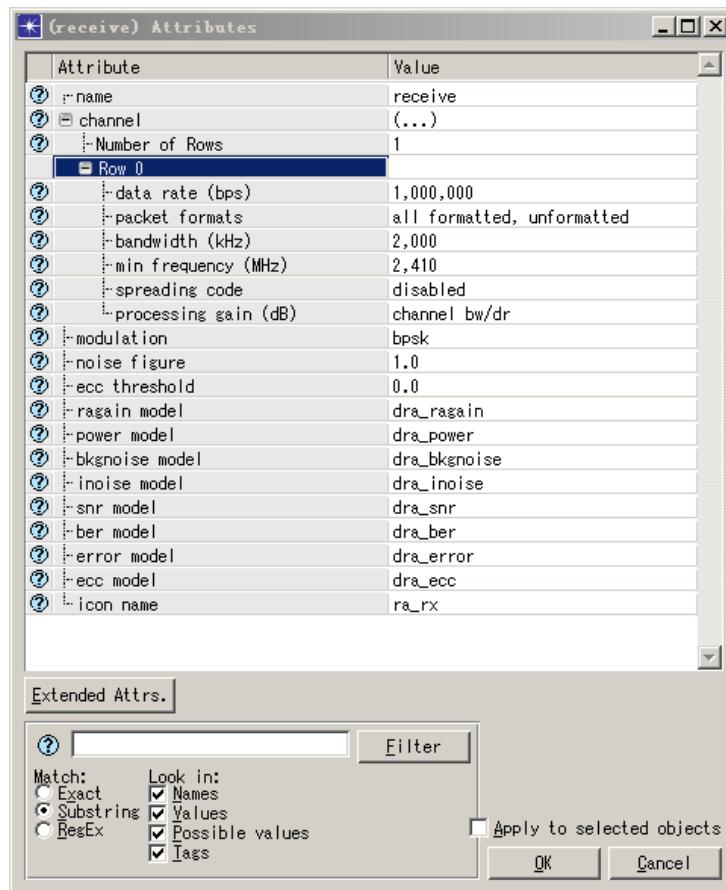
(14) 将 RAU 节点模型保存成 rau_node.nd.m, 设置 RAU 节点模块的节点模型设为 rau_node。同样设置 user 节点模型如图所示。



(15) 设置 user 节点模型中无线发信机模块属性如图所示



(16) 设置 user 节点模型中无线收信机模块属性如图所示



(17) 将 user 节点模型保存文件名为 user_node.nd.m, 将所有的 user 节点模块的节点模型都设置为 user_node。

到此所有节点模块的节点模型创建完毕。

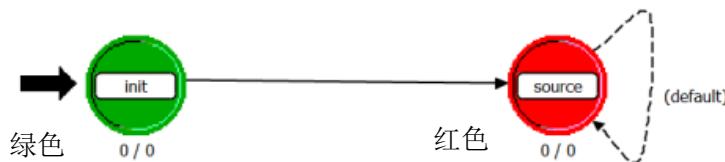
实例 4：创建数据包接收进程模型

在实例 3 的基础上，为 user 节点设计 source, sink, mac 进程模型，为 RAU 节点设计 mac 进程模型，为 AP 节点设计 source, sink, mac 进程模型。

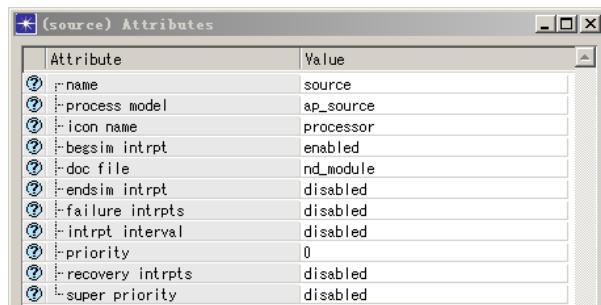
(1) 首先创建 user 节点的 source 进程模型。点击 File-New，选择创建进程模型



(2) 创建如下图所示的进程模型



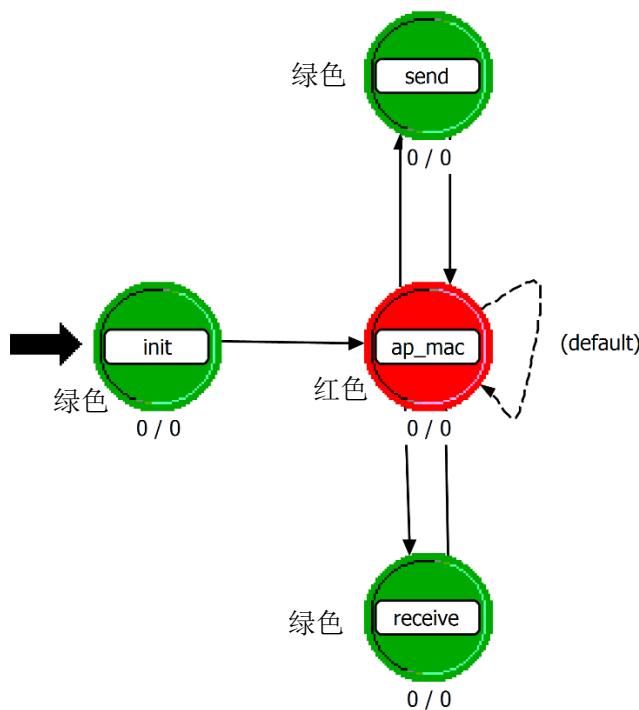
(3) 完成后保存文件为 ap_source_pro.pr.m，保存在项目文件夹外部，关闭 OPNET 软件，将 ap_source_pro.pr.m 文件剪切到项目文件夹内，重新打开 OPNET 软件，双击打开 AP 节点的节点模型，右键单击 source 处理器模块，设置如下图所示。将 process model 修改为刚创建的 ap_source_pro，开启 begsim intrpt 中断



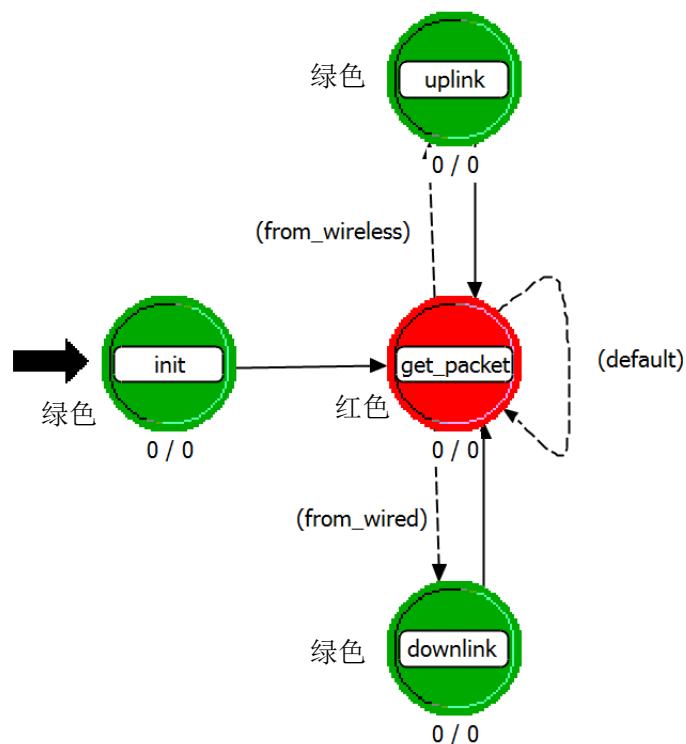
(4) 同样的方法，设置 AP 节点 sink 进程模型，如下图所示，存储成 ap_sink_pro.pr.m



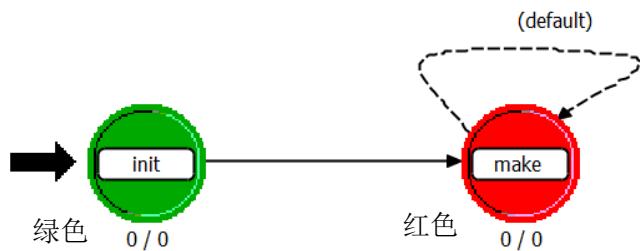
(5) 同样的方法，设置 AP 节点 mac 进程模型，如下图所示，存储成 ap_mac_pro.pr.m



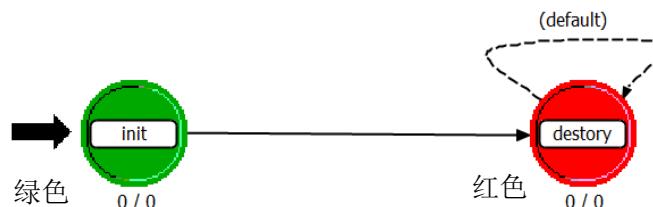
(6) RAU 节点的 mac 进程模型，如下图所示，存储成 rau_pro.pr.m



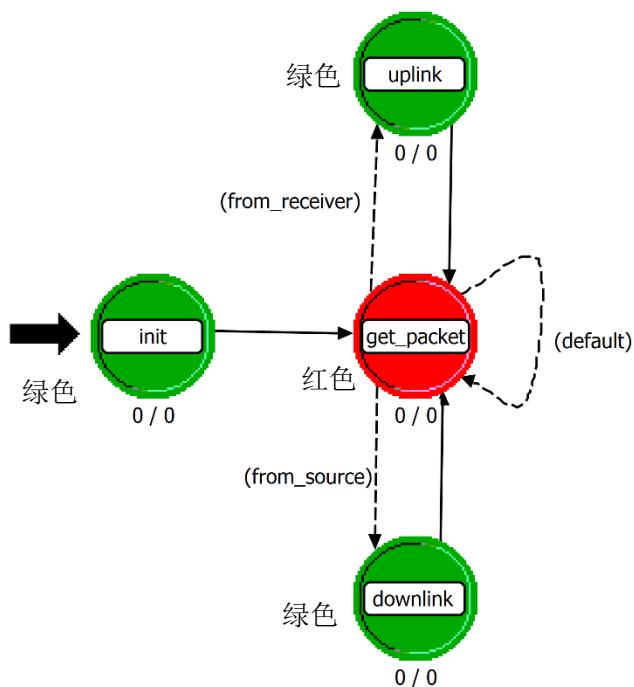
(7) user 节点的 user_source 进程模型，如下图所示，存储成 user_source_pro.pr.m



(8) user 节点的 user_sink 进程模型，如下图所示，存储成 user_sink_pro.pr.m



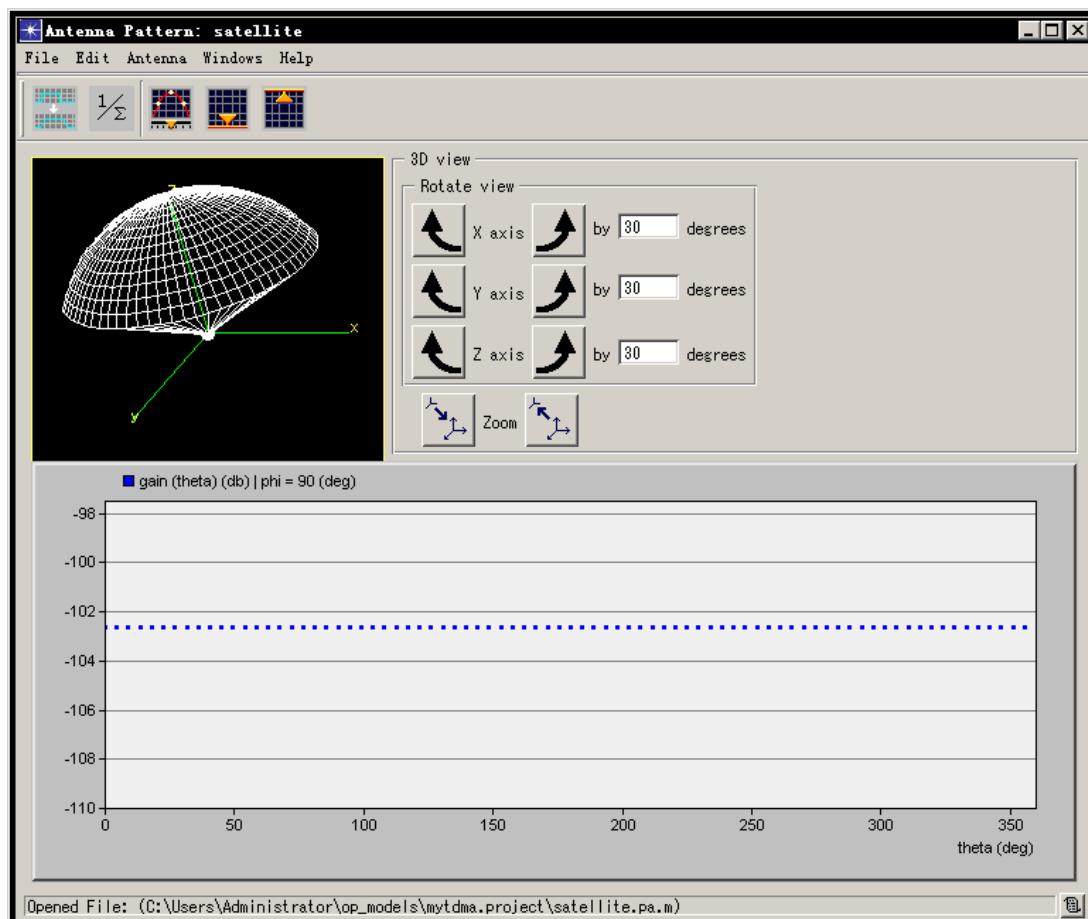
(9) user 节点的 mac 进程模型，如下图所示，存储成 user_mac_pro.pr.m



在每个进程模型中的 Interfaces 下的 Process Interfaces 下设计进程接口，将 begsim intrpt 设置为 enabled。并选中 Compile 下的 Enable C++ Code Generation，使得进程可以编译 C++ 代码结构。

实例 5：定向天线方向图

本实例任务完成如下图所示的天线方向图，主瓣增益为 20db，旁瓣增益为-100db。主瓣张角为 120° 。则需要设置在 $\text{phi}=0^\circ \sim 70^\circ$ 时，增益在 $\text{theta}=0^\circ \sim 360^\circ$ 上均为 20db。在 $\text{phi}=70^\circ \sim 180^\circ$ 时，增益在 $\text{theta}=0^\circ \sim 360^\circ$ 上均为-100db。

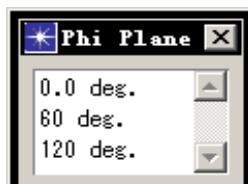


(1) 设置 phi 值的取值数量，减少操作。我们设置 phi 的取值数量为 3。使得 phi 的值分布分为 $0-60^\circ$ 、 $60-120^\circ$ 、 $120-180^\circ$ 。菜单栏 Antenna—Set Phi Plane Count—输入 3—OK。



(2) 设置三个 phi 取值时的增益分布。

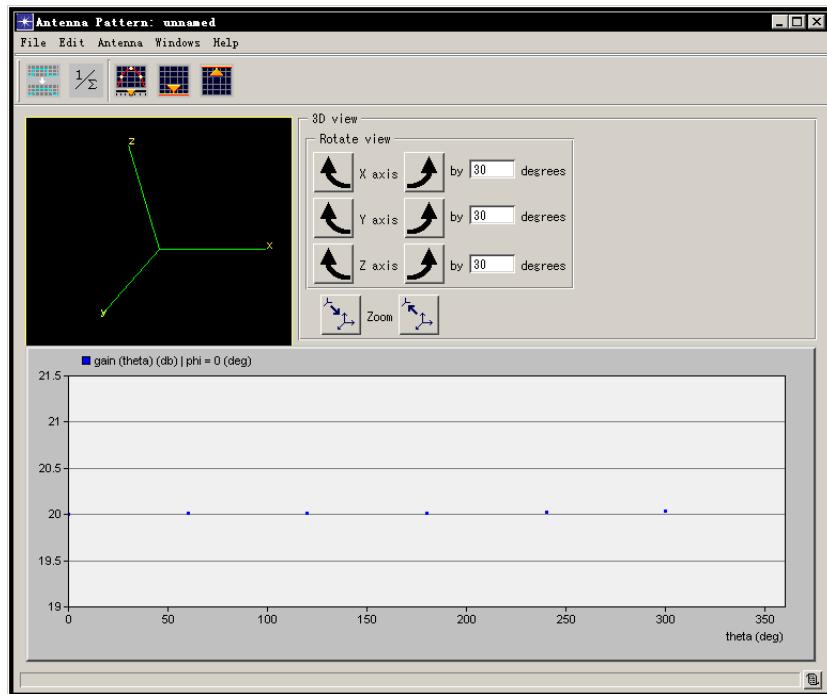
首先右键单击增益分布区域选择 Set Phi Plane，进入 phi 选择界面。选择 $\text{phi}=0.0^\circ$ ，进入 $\text{phi}=0.0^\circ$ 时的增益分布。



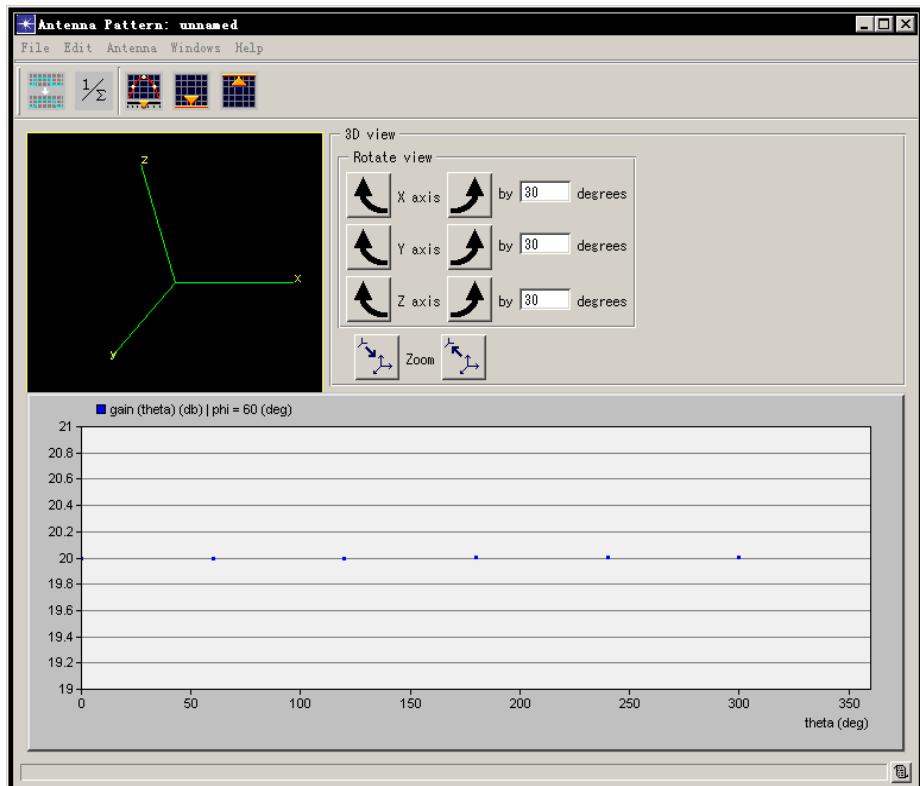
点击 设置上边界为 21，如图所示。点击 设置下边界为 19，如图所示。当原上边界小于将要设置的新的下边界时，应先设置上边界，当原下边界大于新的上边界时，应先设置下边界，否则在设置时容易失败。



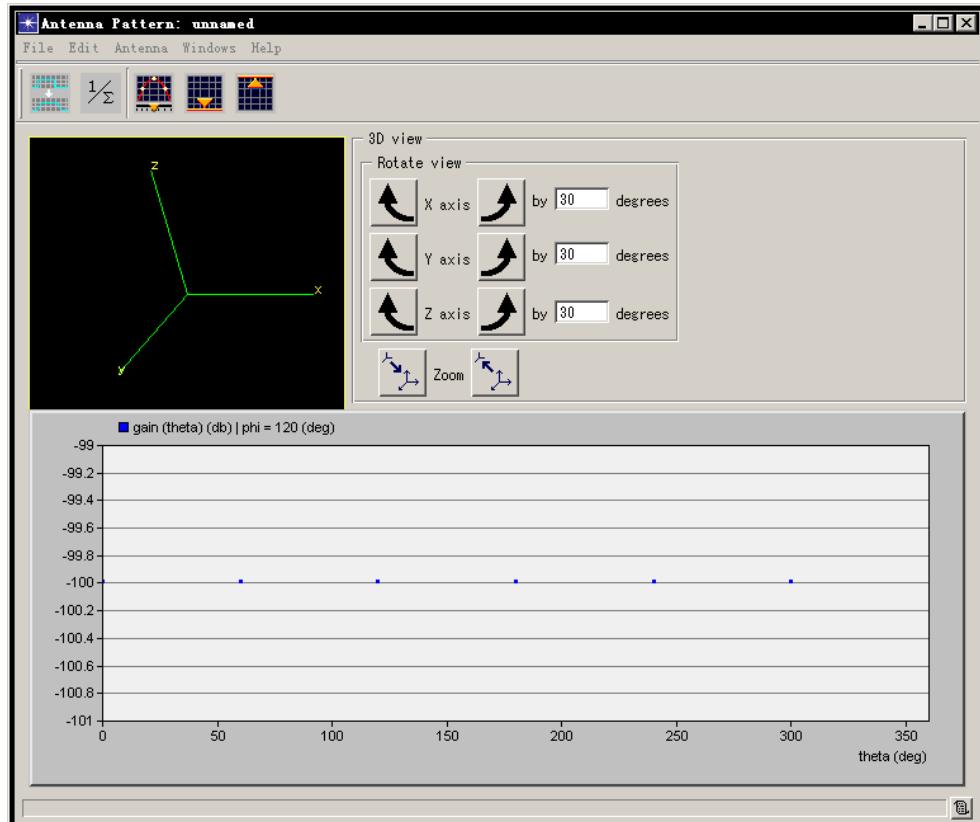
在增益分布区域最左边 20db 和最右边 20db 处分别点击一下，则在增益分布区域自动填充。



右键单击增益分布区域选择 Set Phi Plane，进入 phi 选择界面。选择 phi=60deg，进入 phi=60.0 时的增益分布。同上设置上边界为 21，下边界为 19。在增益分布区域最左边 20db 和最右边 20db 处分别点击一下，则在增益分布区域自动填充。



右键单击增益分布区域选择 Set Phi Plane, 进入 phi 选择界面。选择 phi=120deg, 进入 phi=120 时的增益分布。设置上边界为-99, 下边界为-100。在增益分布区域最左边-100db 和最右边-100db 处分别点击一下，则在增益分布区域自动填充。

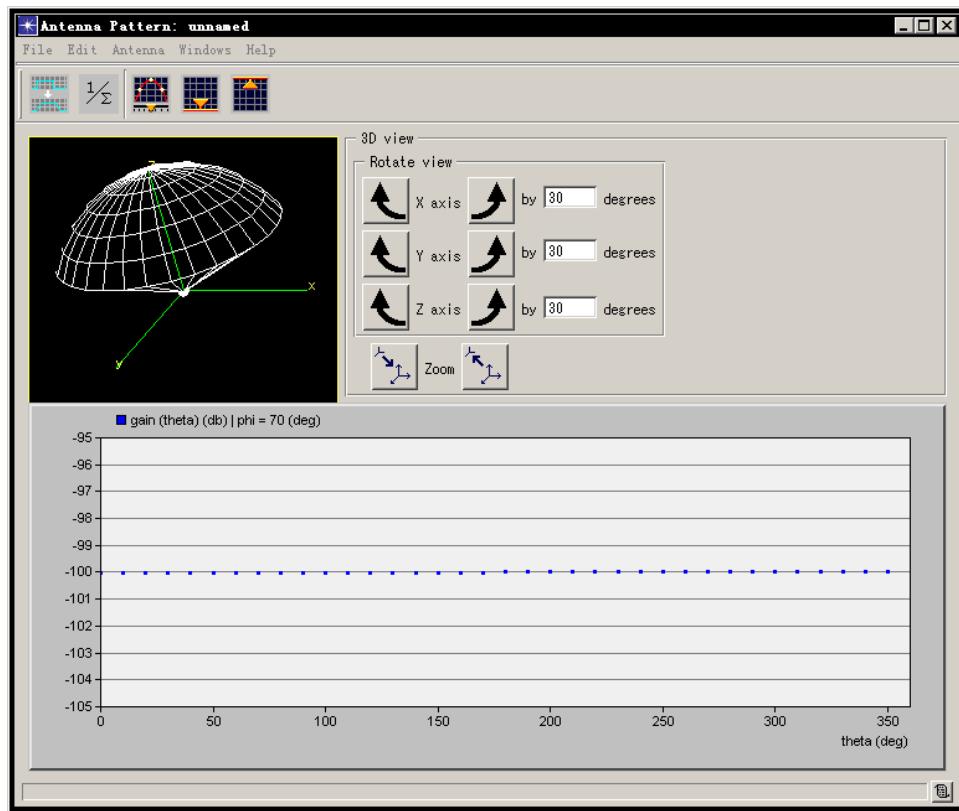


(3) 修改 phi 值的数目为 18。则 phi 的取值变成 0、10、20、30…170。

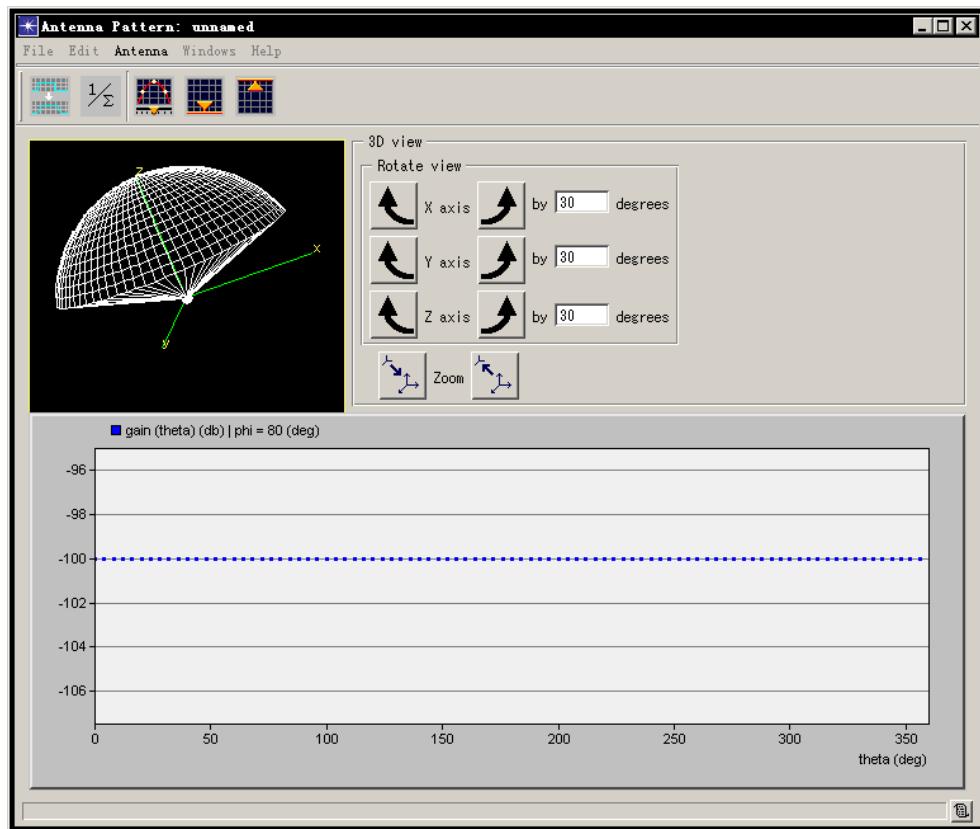
可以发现在 phi=0、10、20…110 时，增益均为 20db。Phi=120、130、170 时，增益均为-

100db。

我们按照步骤 2 中，设置单一 phi 值时增益分布的方法，设置 phi=70、80、90、100、110 处的天线增益。效果如图所示



(4) 有时需要截图，希望天线方向图的曲线更加紧密，我们可以继续增多 phi 取值的数量。同时设置旋转角度使得天线方向图看起来更直观。

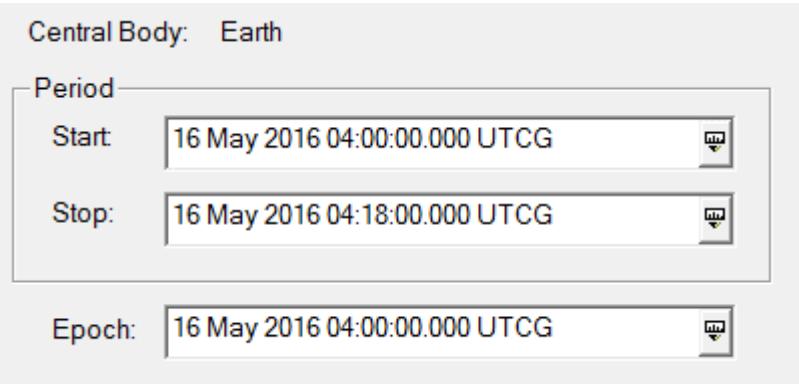


至此，定向天线方向图创建完成

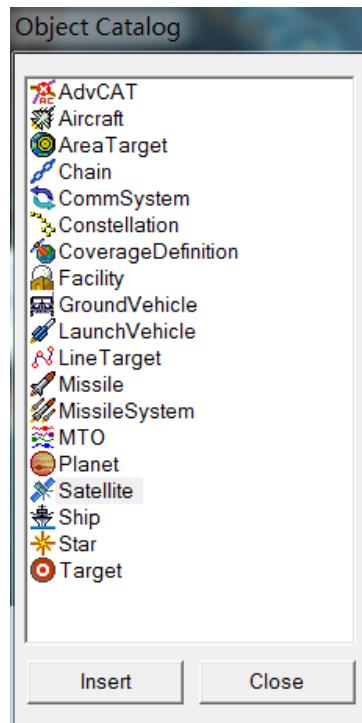
实例 6：卫星轨迹

本实例的任务是为一个卫星节点创建一个在途径中国海域的低轨卫星轨道。

(1) 首先打开 STK 软件，这里使用的是 STK 8 版本。点击菜单栏 File-New，创建项目场景。右键单击场景，Properties Browser 进入场景属性设置界面。选择 Start 时间为 16 May 2016 04:00:00.000 UTCG，Stop 时间为 16 May 2016 04:18:00.000 UTCG，Epoch 时间为 16 May 2016 04:00:00.000 UTCG，如图中所示。点击下方的 Apply 应用场景属性设置。



(2) 点击菜单栏 Insert-New 弹出对话插入框，如图所示



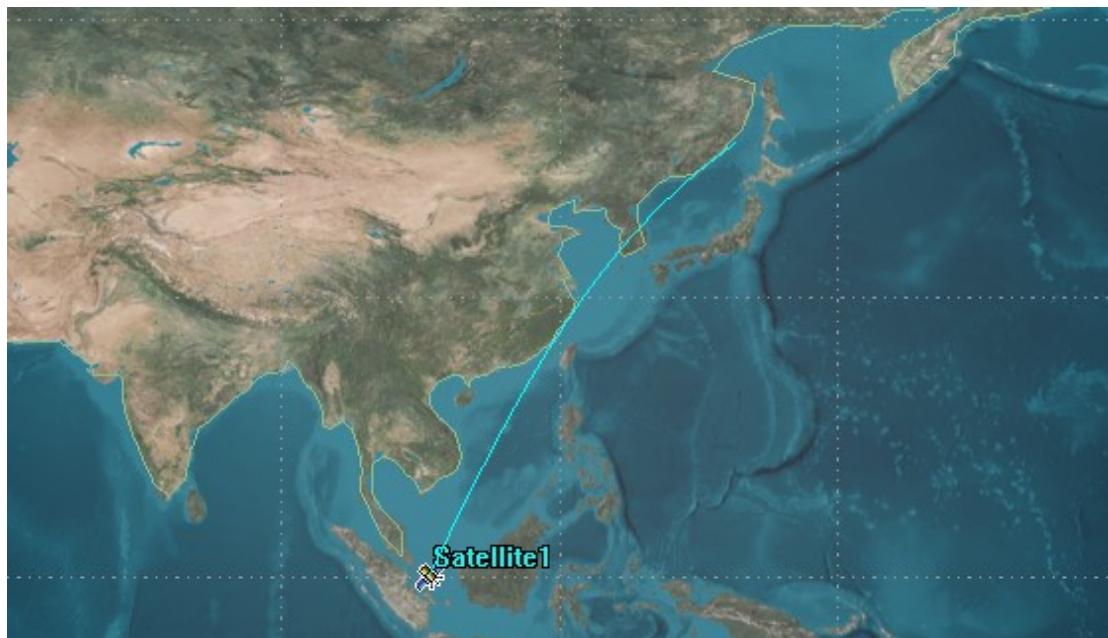
选择 Satellite，点击 Insert，插入卫星节点到场景中。在弹出的轨道设置中选择 Cancel，取消轨道设置。

(3) 在左侧对象浏览窗口中，右键单击插入的卫星节点选择 Properties Browser，进入卫星对象属性设置界面。如果没有对象浏览框，在菜单栏 View 下选择 Object Browser 即可打开对象浏览框。在卫星对象属性设置界面选择 Basic 下 Orbit 设置内容，进入卫星轨道设置界面，如图所示。

Start Time:	16 May 2016 04:00:00.000 UTCG	<input type="button" value="..."/>				
Stop Time:	16 May 2016 04:18:00.000 UTCG	<input type="button" value="..."/>				
Step Size:	60 sec	<input type="button" value="..."/>				
Orbit Epoch:	16 May 2016 04:00:00.000 UTCG	<input type="button" value="..."/>	Apogee Altitude	<input type="button" value="..."/>	1400 km	<input type="button" value="..."/>
Coord Epoch:	1 Jan 2000 11:58:55.816 UTCG	<input type="button" value="..."/>	Perigee Altitude	<input type="button" value="..."/>	1400 km	<input type="button" value="..."/>
Coord Type:	Classical	<input type="button" value="..."/>	Inclination	60 deg	<input type="button" value="..."/>	
Coord System:	J2000	<input type="button" value="..."/>	Argument of Perigee	0 deg	<input type="button" value="..."/>	
Prop Specific:	N/A	<input type="button" value="..."/>	RAAN	<input type="button" value="..."/>	40 deg	<input type="button" value="..."/>
			True Anomaly	<input type="button" value="..."/>	0 deg	<input type="button" value="..."/>

(4) 在卫星轨道设置界面中设置 Step Size 为 60 sec, Apogee Altitude (远地点) 为 1400km, Perigee Altitude (近地点) 为 1400km, Inclination 为 60deg, RAAN 为 40deg。点击设置界面下方的 Apply，应用设置。

设置完成后，点击 2D Graphics (2 维平面地图)，可以看到如图所示，卫星轨道分布在中国海域，卫星其实地点在中国南海，结果地点在中国东北附近。

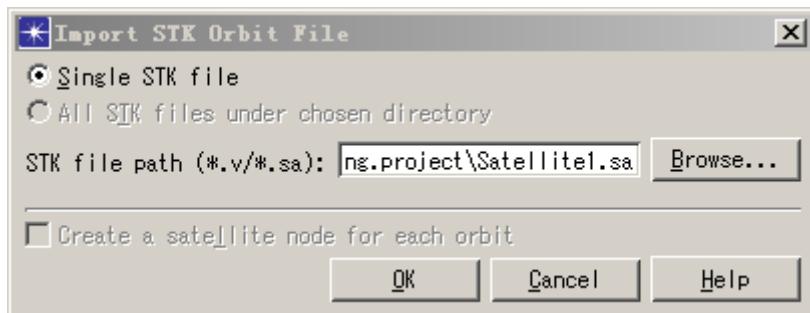


点击 3D Graphics (3 维立体地图)，可以看到如图所示，卫星为低轨轨道，仿真轨道从中国南海到中国东北。



点击工具栏中的 ，开始运行仿真，我们可以看到卫星移动起来。保存项目，在项目目录中找到 Satellite1.sa 轨道文件。将其复制到 OPNET 项目中。

(5) 在 OPNET 中右键单击需要添加卫星轨迹的卫星节点模块，选择属性编辑，必须是卫星节点才能添加卫星轨道。在属性编辑窗口，点击 Import STK Orbit，导入刚才复制过来的轨道文件，如图所示，点击确定。在菜单栏 Topology 中同样可以选择 Import STK Orbit。



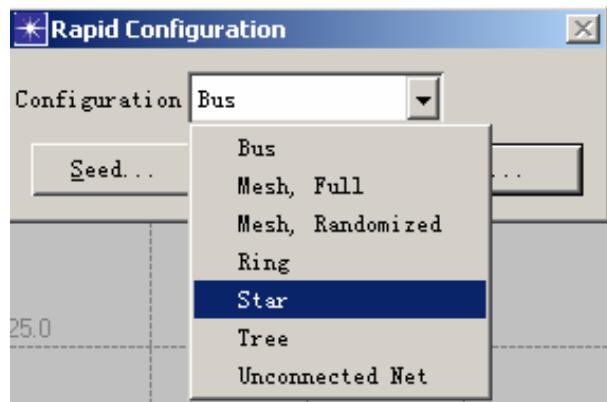
这样轨道文件就导入了仿真系统中。右键单击卫星节点，在属性设置窗口 orbit 属性值中选择刚导入的轨道文件，就成功将轨道应用到了此卫星节点上。在仿真中，卫星节点便按照这个设定轨道进行变化。

在 OPNET 的“top”顶层网络下会显示卫星轨道如图所示。



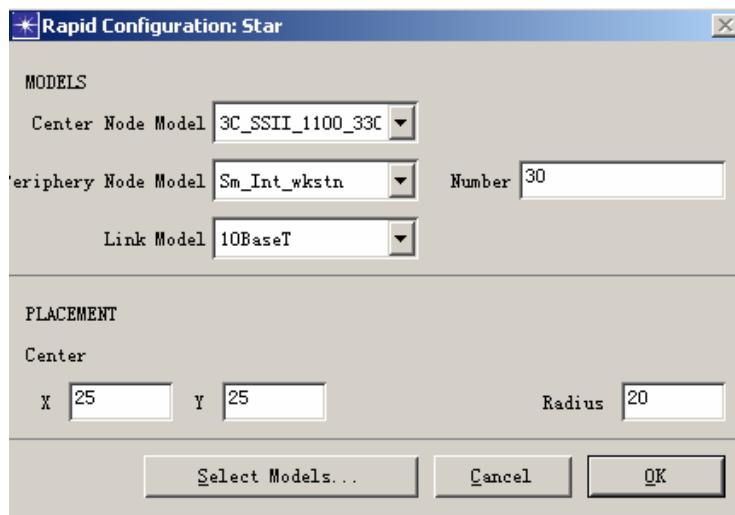
实例 7：配置星状网实例操作

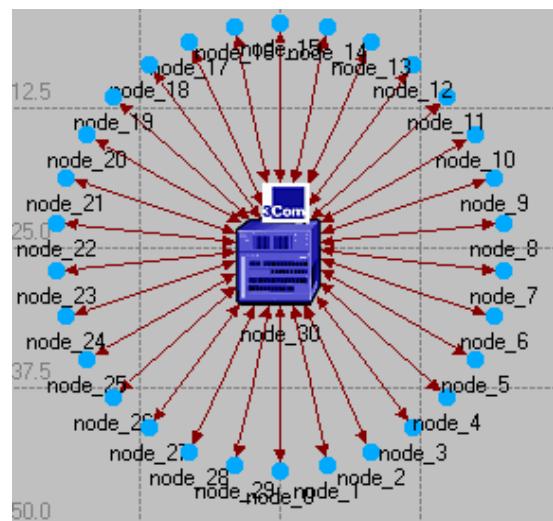
- (1) 从 Topology 菜单中选择 Rapid Configuration。
- (2) 从配置下拉列表中选择 Star, 单击 OK, 如下图所示。



- (3) 选择中心节点模型为 3C_SSII_1100_3300_4s_ae52_e48_ge3。这是 3Com 公司的交换机。
- (4) 选择周边节点模型为 Sm_Int_wkstn，并设置节点个数为 30。
- (5) 选择链路模型为 10BaseT
- (6) 指定网络在工作空间中放置的位置：

设置中心的 X 和 Y 轴坐标为 25。设置局域网的半径范围为 20。





至此，一个星状网配置完成

实例 8：对象属性设置

在实例 4 的基础上，实现 AP、user 节点获取本地地址。

(1) 在 user_source_pro 进程模型中添加状态变量

```
Packet* \mypacket;
int \my_address;
int \user_source_id;
int \user_node_id;
int \ii;
char \name[7];
```

在 user_source_pro 的 init 状态入指令中添加

```
user_source_id=op_id_self();
user_node_id=op_topo_parent(user_source_id);
for(ii=0;ii<7;ii++)
{
    name[ii]=0;
}
op_ima_obj_hname_get(user_node_id,name,7);
my_address=0;
for(ii=0;ii<7;ii++)
{
    if(name[ii]>=48 && name[ii]<=57)
    {
        my_address = my_address*10+name[ii]-48;
    }
}
printf("user%d source initialize succeed\n",my_address);
```

(2) 在 user_mac_pro 进程模型中添加状态变量

```
Packet* \mypacket;
int \frame_type;
Boolean \from_source;
Boolean \from_receiver;
int \receive_address;
int \my_address;
int \user_mac_id;
int \user_node_id;
int \ii;
char \name[7];
```

在 user_mac_pro 的 init 状态入指令中添加

```
user_mac_id=op_id_self();
user_node_id=op_topo_parent(user_mac_id);
for(ii=0;ii<7;ii++)
{
    name[ii]=0;
```

```

}

op_ima_obj_hname_get(user_node_id, name, 7);
my_address=0;
for(ii=0;ii<7;ii++)
{
    if(name[ii]>=48 && name[ii]<=57)
    {
        my_address = my_address*10+name[ii]-48;
    }
}
printf("user%d mac initialize succeed\n",my_address);

```

(3) 在 rau_pro 进程模型中添加状态变量

```

Boolean \from_wireless;
Boolean \from_wired;
Packet * \mypacket;

```

在 rau_pro 的 init 状态入指令中添加

```
printf("rau mac initialize succeed\n");
```

(4) 在 AP 节点模块属性中添加自定义属性 ap_address，类型为 ap_address，默认值为 1

(5) 在 ap_source_pro 进程模型中定义状态变量

```

Packet* \mypacket;
int \my_address;
int \ap_source_id;
int \ap_node_id;
int \ii;

```

在 ap_source_pro 进程模型的 init 状态下添加

```

ap_source_id=op_id_self();
ap_node_id=op_topo_parent(ap_source_id);
op_ima_obj_attr_get (ap_node_id, "ap_address", &my_address);
printf("ap source initialize succeed,address = %d\n",my_address);

```

(6) 在 ap_mac_pro 进程模型中定义状态变量

```

Packet* \mypacket;
int \my_address;
int \ap_mac_id;
int \ap_node_id;
int \ii;
Boolean \from_source;
Boolean \from_receiver;
int \send_address;
int \receive_address;
int \frame_type;

```

在 ap_mac_pro 的 init 状态入指令中添加

```

ap_mac_id=op_id_self();
ap_node_id=op_topo_parent(ap_mac_id);
op_ima_obj_attr_get (ap_node_id, "ap_address", &my_address);

```

```
printf("ap mac initialize succeed,address = %d\n",my_address);
```

(7) 在 ap_sink_pro 进程模型中定义状态变量

```
Packet* \mypacket;
```

在 ap_sink_pro 的 init 状态入指令中添加

```
printf("ap sink initialize succeed\n");
```

属性设置到这里就成功了,点击  运行程序,在 Common 界面 Simulation Kernel 选项中选择 Development,在 Execution 的 OPNET Debugger 界面中选中 Use OPNET Simulation Debugger(ODB)。点击 run 运行程序。点击 Continue 开始运行(调试方案可参照 4.1 章节)。弹出如下图所示的仿真输出。

```
ODB> continue
ap source initialize succeed,address = 1
ap mac initialize succeed,address = 1
ap sink initialize succeed
rau mac initialize succeed
user2 source initialize succeed
user2 mac initialize succeed
user3 source initialize succeed
user3 mac initialize succeed
user1 source initialize succeed
user1 mac initialize succeed
```

实例 9：发送回应的包交互实现

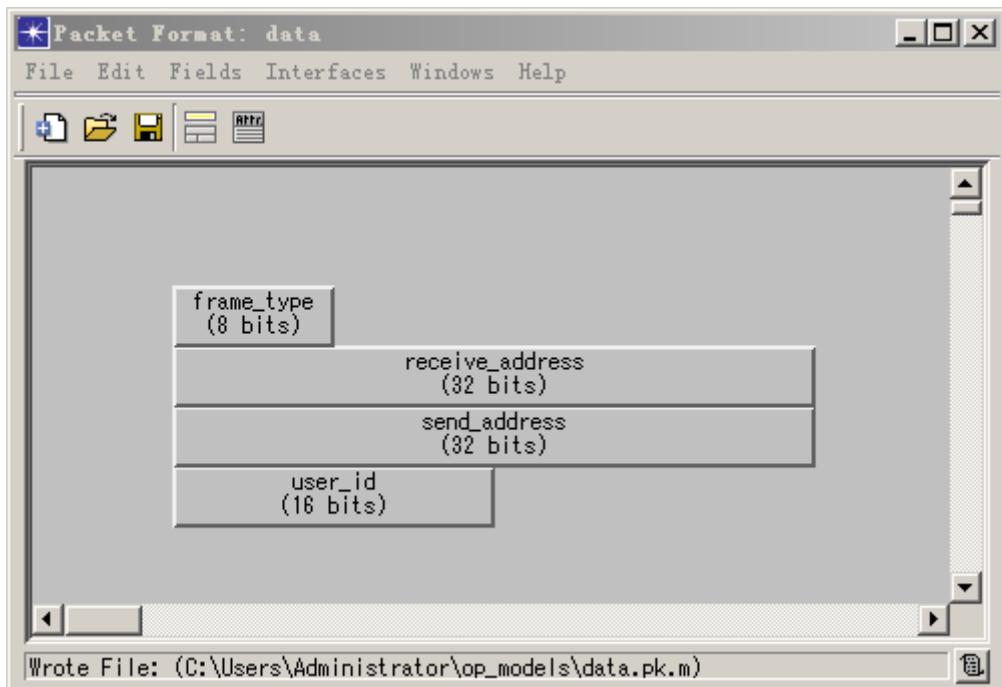
在实例 8 的基础上，实现 user 通过 RAU 转发向 AP 发送数据包，AP 接收到数据包后回应 ACK 帧。

(1) 定义 data 数据包格式与 ACK 帧格式。

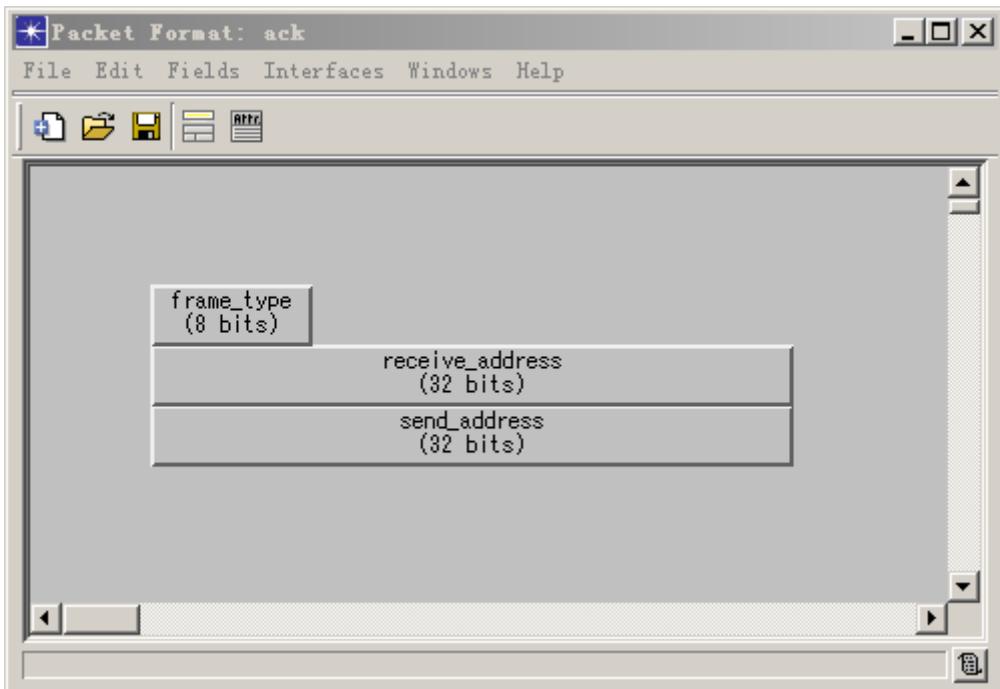
点击菜单栏 File-New，选择 Packet Format，创建数据包格式。



创建如图所示的数据包格式，包含 frame_type 字段，integer 整型，8 位；receive_address 和 send_address 字段，integer 整型，32 位；user_id 字段，integer 整型，16 位。保存成 data 文件名。并复制文件到项目文件夹下。



同样方法创建如图所示的 ACK 帧格式，包含 frame_type 字段，integer 整型，8 位；receive_address 和 send_address 字段，integer 整型，32 位。保存成 ack 文件名。并复制文件到项目文件夹下。



将实例 8 中的所有 printf 函数注释掉。

(2) user_source 模块产生数据包，并发送到 user_mac 模块。

在 user_source_pro 的 make 状态入指令中添加

```
mypacket = op_pk_create_fmt("data");
op_pk_total_size_set (mypacket,5000);
op_pk_nfd_set(mypacket, "frame_type", 1);
op_pk_nfd_set(mypacket, "receive_address", 1);
op_pk_nfd_set(mypacket, "send_address", my_address);
op_pk_send_delayed(mypacket,0,my_address); //0 是输出包流索引号，通过 source 的模块关联可以查看
printf("user%d source make data packet,time=%f\n",my_address,op_sim_time());
```

(3) user_mac 模块接收 user_source 发送的数据包，并发送到无线发信机，接收来自无线接收机的数据包进行判断和销毁。

在 user_mac_pro 进程模型的 get_packet 状态出指令添加

```
if(op_intrpt_type()==OPC_INTRPT_STRM)
{
    mypacket = op_pk_get (op_intrpt_strm());
    if(op_intrpt_strm()==0)
    {
        from_receiver =OPC_FALSE;
        from_source = OPC_TRUE;
    }
    else if(op_intrpt_strm()==1)
    {
        from_receiver = OPC_TRUE;
        from_source = OPC_FALSE;
        op_pk_nfd_get(mypacket, "receive_address",&receive_address);
    }
}
```

```

    if(receive_address!=my_address)
    {
        op_pk_destroy(mypacket);
        from_receiver = OPC_FALSE;
        from_source = OPC_FALSE;
    }
}
}

```

在 uplink 状态入指令中添加

```

op_pk_nfd_get(mypacket, "frame_type",&frame_type);
if(frame_type==2)
{
    op_pk_destroy (mypacket);
    printf("user%d receive ack packet and destroy it, time = %f\n",my_address,op_sim_time());
}

```

在 downlink 状态入指令中添加代码

```

op_pk_nfd_get(mypacket, "frame_type",&frame_type);
if(frame_type==1)
{
    op_pk_send (mypacket,0); //0 是输出包流索引号，通过 user_mac 的模块关联可以查看
    printf("user%d mac receive data packet from source and send to ap, time = %f\n",my_address,op_sim_time());
}

```

(4) RAU 节点 mac 模块设置转发数据包

在 rau_pro 进程模块的 get_packet 状态出指令添加

```

if(op_intrpt_type()==OPC_INTRPT_STRM)
{
    mypacket = op_pk_get (op_intrpt_strm());
    if(op_intrpt_strm()==0)
    {
        from_wired = OPC_TRUE;
        from_wireless = OPC_FALSE;
    }
    else if(op_intrpt_strm()==1)
    {
        from_wired = OPC_FALSE;
        from_wireless = OPC_TRUE;
    }
}

```

在 uplink 状态入指令添加

```

op_pk_send(mypacket,1); //1 是输出包流索引号，通过 rau_mac 的模块关联可以查看
printf("rau receive packet from wireless and send it to ap, time = %f\n",op_sim_time());

```

在 downlink 状态入指令添加

```
op_pk_send(mypacket,0); //0 是输出包流索引号，通过 rau_mac 的模块关联可以查看
printf("rau receive packet from wired and send it to user,time = %f\n",op_sim_time());
```

(5) AP 的 mac 模块接收到数据包，将数据包发送到 ap_sink 模块，同时回复 ACK 帧发送到 AP 的有线发信机

ap_mac_pro 进程模型 ap_mac 状态出指令添加

```
if(op_intrpt_type()==OPC_INTRPT_STRM)
{
    mypacket = op_pk_get (op_intrpt_strm());
    if(op_intrpt_strm()==0)
    {
        from_receiver = OPC_FALSE;
        from_source = OPC_TRUE;
    }
    else if(op_intrpt_strm()==1)
    {
        from_receiver = OPC_TRUE;
        from_source = OPC_FALSE;
        op_pk_nfd_get(mypacket, "receive_address",&receive_address);
        if(receive_address!=my_address)
        {
            op_pk_destroy(mypacket);
            from_receiver = OPC_FALSE;
            from_source = OPC_FALSE;
        }
    }
}
```

在 receive 状态入指令添加

```
op_pk_nfd_get(mypacket, "frame_type",&frame_type);
op_pk_nfd_get(mypacket, "send_address",&send_address);
if(frame_type==1)
{
    op_pk_send(mypacket,1); //1 是输出包流索引号，通过 ap_mac 的模块关联可以查看
    printf("ap mac receive data packet and send it to sink,time = %f\n",op_sim_time());
    mypacket = op_pk_create_fmt("ack");
    op_pk_nfd_set(mypacket, "frame_type", 2);
    op_pk_nfd_set(mypacket, "receive_address", send_address);
    op_pk_nfd_set(mypacket, "send_address", my_address);
    op_pk_send(mypacket,0); //0 是输出包流索引号，通过 ap_mac 的模块关联可以查看
    printf("ap send ack packet,time = %f\n",op_sim_time());
}
```

(6) ap_sink 模块接收到 ap_mac 发来的数据包销毁

在 ap_sink_pro 进程模型 destory 状态出指令添加

```

if (op_intrpt_type() == OPC_INTRPT_STRM)
{
    mypacket = op_pk_get (op_intrpt_strm());
    op_pk_destroy(mypacket);
    printf("ap sink destroy packet\n");
}

```

代码添加完成后再次确认无线有线收发信机至此的包格式和光纤链路支持的包格式均为全部支持。并在每个进程模型中点击  按钮，编译进程模型，保证每个进程编译无误。

数据包交互代码到这里就成功了，点击 ，运行程序，在 Common 界面 Simulation Kernel 选项中选择 Development，在 Execution 的 OPNET Debugger 界面中选中 Use OPNET Simulation Debugger(ODB)。点击 run 运行程序。点击 Continue 开始运行（调试方案可参照 4.1 章节）。弹出如下图所示的仿真输出。

```

ODB> continue
user2 source make data packet,time=0.000000
user3 source make data packet,time=0.000000
user1 source make data packet,time=0.000000
user1 mac receive data packet from source and send to ap,time = 1.000000
rau receive packet from wireless and send it to ap,time = 1.005005
ap mac receive data packet and send it to sink,time = 1.005055
ap send ack packet,time = 1.005055
ap sink destroy packet
rau receive packet from wired and send it to user,time = 1.005099
user1 receive ack packet and destroy it,time = 1.005176
user2 mac receive data packet from source and send to ap,time = 2.000000
rau receive packet from wireless and send it to ap,time = 2.005005
ap mac receive data packet and send it to sink,time = 2.005055
ap send ack packet,time = 2.005055
ap sink destroy packet
rau receive packet from wired and send it to user,time = 2.005100
user2 receive ack packet and destroy it,time = 2.005178
user3 mac receive data packet from source and send to ap,time = 3.000000
rau receive packet from wireless and send it to ap,time = 3.005005
ap mac receive data packet and send it to sink,time = 3.005054
ap send ack packet,time = 3.005054
ap sink destroy packet
rau receive packet from wired and send it to user,time = 3.005099
user3 receive ack packet and destroy it,time = 3.005176

```

实例 10：进程中断

在实例 9 的基础上，实现 user 接收到 ACK 帧以后停留 5 秒钟继续产生数据包发送。

(1) 在 user_mac 模块接收到 ACK 帧后发起进程中断给 user_source 模块即可。

在 user_mac_pro 的 uplink 状态入指令原有代码的后面添加

```
if(frame_type==2)
{
    int user_source_id = op_id_from_name(user_node_id,OPC_OBJTYPE_PROC, "user_source");
    Prohandle user_source_pro = op_pro_root(user_source_id);
    op_intrpt_schedule_process(user_source_pro,op_sim_time() +5,1);
}
```

代码即添加完成，点击  按钮，编译进程模型。

数据包交互代码到这里就成功了，点击 ，运行程序，在 Common 界面 Simulation Kernel 选项中选择 Development，在 Execution 的 OPNET Debugger 界面中选中 Use OPNET Simulation Debugger(ODB)。设置仿真时间为 12second(s)，点击 run 运行程序。点击 Continue 开始运行（调试方案可参照 4.1 章节）。弹出如下图所示的仿真输出。

```
ODB> continue
user1 source make data packet,time=0.000000
user2 source make data packet,time=0.000000
user3 source make data packet,time=0.000000
user1 mac receive data packet from source and send to ap,time = 1.000000
rau receive packet from wireless and send it to ap,time = 1.005005
ap mac receive data packet and send it to sink,time = 1.005055
ap send ack packet,time = 1.005055
ap sink destroy packet
rau receive packet from wired and send it to user,time = 1.005099
user1 receive ack packet and destroy it,time = 1.005176
user2 mac receive data packet from source and send to ap,time = 2.000000
rau receive packet from wireless and send it to ap,time = 2.005005
ap mac receive data packet and send it to sink,time = 2.005055
ap send ack packet,time = 2.005055
ap sink destroy packet
rau receive packet from wired and send it to user,time = 2.005100
user2 receive ack packet and destroy it,time = 2.005178
user3 mac receive data packet from source and send to ap,time = 3.000000
rau receive packet from wireless and send it to ap,time = 3.005005
ap mac receive data packet and send it to sink,time = 3.005055
ap send ack packet,time = 3.005055
ap sink destroy packet
rau receive packet from wired and send it to user,time = 3.005100
user3 receive ack packet and destroy it,time = 3.005177
user1 source make data packet,time=6.005176
user1 mac receive data packet from source and send to ap,time = 7.005176
user2 source make data packet,time=7.005178
rau receive packet from wireless and send it to ap,time = 7.010181
ap mac receive data packet and send it to sink,time = 7.010231
ap send ack packet,time = 7.010231
ap sink destroy packet
rau receive packet from wired and send it to user,time = 7.010276
user1 receive ack packet and destroy it,time = 7.010352
user3 source make data packet,time=8.005177
user2 mac receive data packet from source and send to ap,time = 9.005178
rau receive packet from wireless and send it to ap,time = 9.010183
```

在 12 秒钟内每个 user 节点均向 ap 节点发送了两次数据。

实例 11：随机分布

在实例 10 的基础上，实现 user 接收到 ACK 帧以后退避一个在 2~5 秒之间均匀分布的随机秒数，然后继续产生数据包发送。

(1) 在 user_mac 模块接收到 ACK 帧后发起进程中断给 user_source 模块即可。

在 user_mac_pro 的 uplink 状态入指令原有代码中的

```
op_intrpt_schedule_process(user_source_pro, op_sim_time() + 5, 1);
```

更改为

```
double delay_time = op_dist_uniform(3) + 2;
op_intrpt_schedule_process(user_source_pro, op_sim_time() + delay_time, 1);
```

代码即添加完成，点击  按钮，编译进程模型。点击 ，运行程序，设置仿真时间为 12second(s)，点击 run 运行程序。点击 Continue 开始运行。弹出如下图所示的仿真输出。

```
rau receive packet from wireless and send it to ap, time = 6.610780
ap mac receive data packet and send it to sink, time = 6.610830
ap send ack packet, time = 6.610830
ap sink destroy packet
rau receive packet from wired and send it to user, time = 6.610875
user2 receive ack packet and destroy it, time = 6.610952
user3 mac receive data packet from source and send to ap, time = 8.920506
rau receive packet from wireless and send it to ap, time = 8.925511
ap mac receive data packet and send it to sink, time = 8.925561
ap send ack packet, time = 8.925561
ap sink destroy packet
rau receive packet from wired and send it to user, time = 8.925606
user3 receive ack packet and destroy it, time = 8.925683
user1 source make data packet, time = 8.981077
user1 mac receive data packet from source and send to ap, time = 9.981077
rau receive packet from wireless and send it to ap, time = 9.986081
ap mac receive data packet and send it to sink, time = 9.986131
ap send ack packet, time = 9.986131
ap sink destroy packet
rau receive packet from wired and send it to user, time = 9.986176
user1 receive ack packet and destroy it, time = 9.986253
user2 source make data packet, time = 11.517354
user3 source make data packet, time = 11.777117
```

根据数据包的产生时间可以看出每个 user 节点接收到 ACK 帧到产生数据之间的时间不再固定。

实例 12：结构体与列表

在实例 11 的基础上，实现 ap 节点记录在网用户以及用户的相关信息。

(1) 在 ap_mac_pro 进程模型中的头模块中添加结构体定义。

```
typedef struct user_information
{
    Packet*    user_send_packet;
    int        user_address;
    int        user_packet_num;
}user_information;
```

(2) 在 ap_mac_pro 的状态变量定义中添加列表变量和结构体变量定义

```
List*    \user_list;
user_information* \user_struck;
```

(3) 在 ap_mac_pro 的 init 状态入指令中添加列表初始化代码

```
user_list = op_prg_list_create();
```

(4) 在 ap_mac_pro 的 receive 状态入指令中代码更改为

```
op_pk_nfd_get(mypacket, "frame_type", &frame_type);
op_pk_nfd_get(mypacket, "send_address", &send_address);
if(frame_type==1)
{
    bool user_have_in = false;
    for(ii=0; ii<op_prg_list_size(user_list); ii++)
    {
        user_struck = (user_information *)op_prg_list_access(user_list, ii);
        if(user_struck->user_address==send_address)
        {
            user_have_in=true;
            user_struck->user_packet_num = user_struck->user_packet_num+1;
            break;
        }
    }
    if(!user_have_in)
    {
        user_struck = (user_information *)op_prg_mem_alloc(sizeof(user_information));
        user_struck->user_address=send_address;
        user_struck->user_packet_num=0;
        op_prg_list_insert(user_list, user_struck, OPC_LISTPOS_TAIL);
    }
}
if(frame_type==1)
{
    op_pk_send(mypacket, 1);
    printf("ap mac receive data packet and send it to sink, time = %f\n", op_sim_time());
    mypacket = op_pk_create_fmt("ack");
```

```
op_pk_nfd_set(mypacket, "frame_type", 2);
op_pk_nfd_set(mypacket, "receive_address", send_address);
op_pk_nfd_set(mypacket, "send_address", my_address);
op_pk_send(mypacket,0);
printf("ap send ack packet,time = %f\n",op_sim_time());
}
```

代码即添加完成，点击  按钮，编译进程模型。

实例 13：统计量的使用

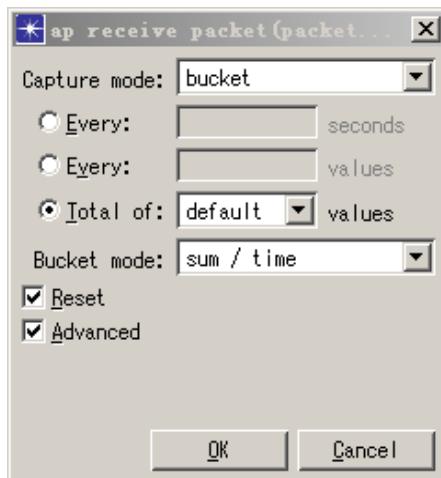
在实例 12 的基础上，收集一定时间内的 AP 节点接收到数据包的吞吐量与端到端时延。

(1) 添加统计量定义

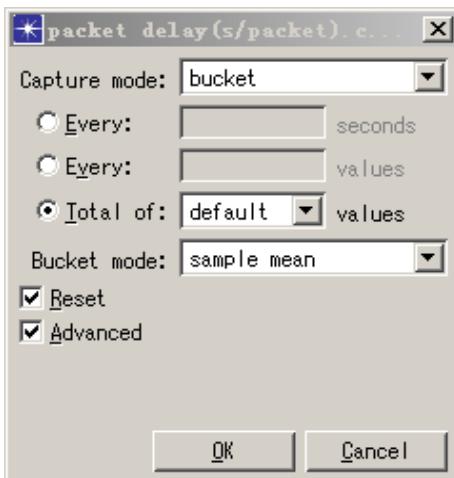
进入 ap_sink_pro 进程模型，菜单栏点击 Interface，选择 Global Statistics，进入全局统计量定义界面，创建如图所示的两个全局统计量。这里使用 ap receive packet(packets/s) 表示吞吐量统计量。使用 packet delay(s/packet) 表示端到端时延统计量。

Stat Name	Mode	Count	Description	Group	Capture Mode	Draw Style	Low Bound	High Bound
ap receive packet(packets/s)	Single	N/A		ap_sink bucket/default	total/sum_time	linear	0.0	disabled
packet delay(s/packet)	Single	N/A		ap_sink bucket/default	total/sample mean	linear	0.0	disabled

其中 ap receive packet(packets/s) 统计的捕获模式设置如图所示。



其中 packet delay(s/packet) 统计的捕获模式设置如图所示。



(2) 定义统计量句柄。

在 ap_sink_pro 的状态变量中添加

```
Stathandle \ap_receive_num;
Stathandle \delay_packet;
```

(3) 注册统计量。

在 ap_sink_pro 进程模型的 init 状态入指令

```
ap_receive_num=op_stat_reg("ap_sink.ap    receive    packet(packets/s)",OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
```

```
delay_packet = op_stat_reg("ap_sink.packet delay(s/packet)", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
```

(4) 统计量数据收集

在 ap_sink_pro 进程模型的 destroy 状态出指令中将原有代码

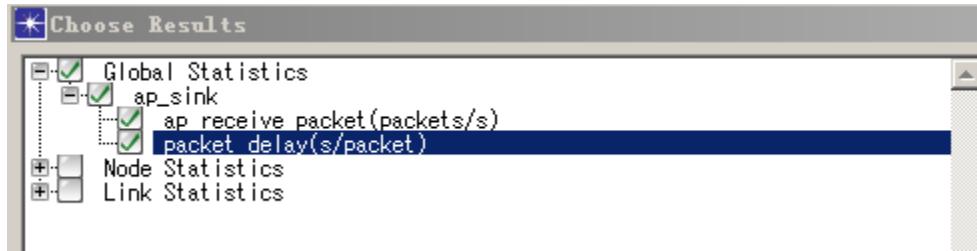
```
if(op_intrpt_type()==OPC_INTRPT_STRM)
{
    mypacket = op_pk_get (op_intrpt_strm());
    op_pk_destroy(mypacket);
    printf("ap sink destroy packet\n");
}
```

更改为代码

```
if(op_intrpt_type()==OPC_INTRPT_STRM)
{
    mypacket = op_pk_get (op_intrpt_strm());
    op_stat_write(ap_receive_num,1);
    double delay_time =op_sim_time()- op_pk_creation_time_get (mypacket);
    op_stat_write(delay_packet,delay_time);
    op_pk_destroy(mypacket);
    printf("ap sink destroy packet\n");
}
```

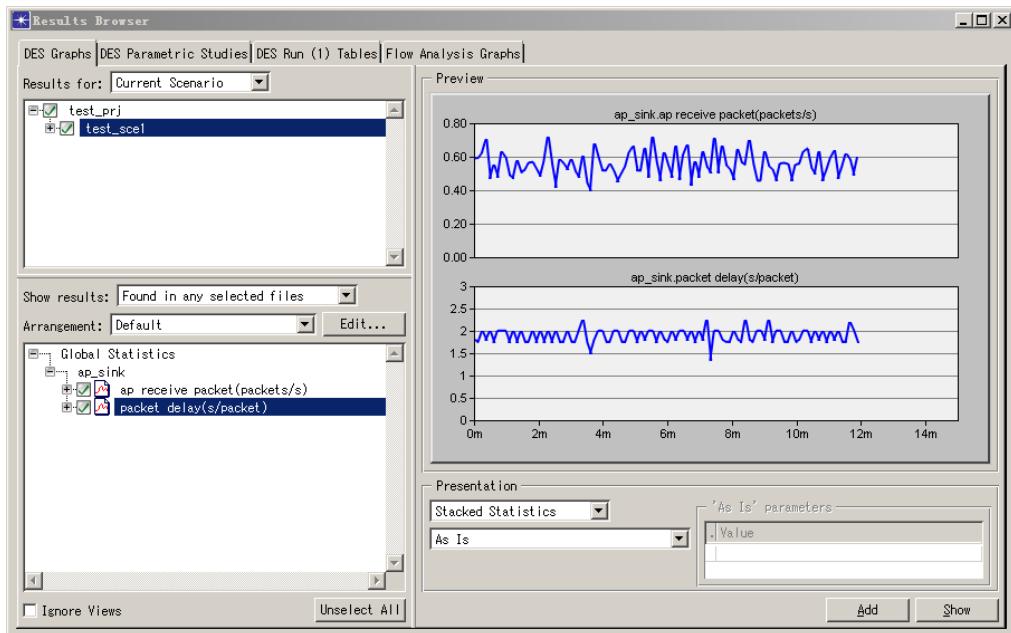
代码编写部分完成，点击  按钮，编译进程模型。

回到子网模型界面，点击菜单栏 DES 下选择 Choose Individual Statistics 进入统计量选择界面，也可以在子网模型空白处单击右键选择 Choose Individual DES Statistics。选中刚定义的两个统计量如图所示。点击 OK 后关闭。



点击 ，运行程序，设置仿真时间为 12 minute(s)，点击 run 运行程序。点击 Continue 开始运行。运行完毕后关闭调试窗口。

点击 ，打开仿真结果视图，选中刚收集的两个统计量，可以在右侧窗口中看到两个统计量的值。如图所示。



实例 14：TDA 使用

在实例 13 的基础上，学习使用数据包 TDA 域的操作。

(1) 加速 user_source_pro 发包频率，修改 user_source_pro 进程模型，make 状态入指令中

```
op_pk_send_delayed(mypacket, my_address);
```

修改为

```
op_pk_send_delayed(mypacket, 0, 0.001*my_address);
```

编译，设置仿真时间为 12second，运行仿真，查看仿真的打印输出如图。

```
user1 source make data packet,time=0.000000
user2 source make data packet,time=0.000000
user3 source make data packet,time=0.000000
user1 mac receive data packet from source and send to ap,time = 0.001000
user2 mac receive data packet from source and send to ap,time = 0.002000
user3 mac receive data packet from source and send to ap,time = 0.003000
```

可以看出每个用户发送的数据包都没有被 RAU 节点接收。

(2) 在 (1) 的基础上，修改 RAU 节点无线接收机的误码门限。

修改 RAU 节点 receive_wireless 模块属性 ecc_threshold 为 1.0，即表示无论发生什么冲突，发生多少误码均能接收。因为 OPNET 只是通过记录误码数目和可恢复数目结合可接受误码率，来确定是否接收该数据包，并不对数据包进行码元变换。

编译，设置仿真时间为运行仿真，查看打印输出结果如图

```
user1 source make data packet,time=0.000000
user2 source make data packet,time=0.000000
user3 source make data packet,time=0.000000
user1 mac receive data packet from source and send to ap,time = 0.001000
user2 mac receive data packet from source and send to ap,time = 0.002000
user3 mac receive data packet from source and send to ap,time = 0.003000
rau receive packet from wireless and send it to ap,time = 0.006005
ap mac receive data packet and send it to sink,time = 0.006055
ap send ack packet,time = 0.006055
ap sink destroy packet
rau receive packet from wired and send it to user,time = 0.006099
user1 receive ack packet and destroy it,time = 0.006176
user1 source make data packet,time=2.921505
user1 mac receive data packet from source and send to ap,time = 2.922505
rau receive packet from wireless and send it to ap,time = 2.927510
ap mac receive data packet and send it to sink,time = 2.927560
ap send ack packet,time = 2.927560
ap sink destroy packet
rau receive packet from wired and send it to user,time = 2.927605
user1 receive ack packet and destroy it,time = 2.927681
user1 source make data packet,time=7.288960
user1 mac receive data packet from source and send to ap,time = 7.289960
rau receive packet from wireless and send it to ap,time = 7.294964
ap mac receive data packet and send it to sink,time = 7.295014
ap send ack packet,time = 7.295014
ap sink destroy packet
rau receive packet from wired and send it to user,time = 7.295059
user1 receive ack packet and destroy it,time = 7.295136
```

可以看出虽然用户节点在发包时在 RAU 节点处冲突了，但是 RAU 仍然能完全接收到数据包。

(3) 读取接收数据包的冲突标志位，查看一个数据包在接收到的时候已经冲突了多少次，如果没有冲突则保留，如果冲突了则直接销毁。将 rau_mac_pro 进程模型 uplink 状态入指令代码替换成

```
double colls = op_td_get_int (mypacket, OPC_TDA_RA_NUM_COLLSS);
```

```
if(coll>0)
    op_pk_destroy(mypacket);
else
{
    op_pk_send(mypacket,1);
    printf("rau receive packet from wireless and send it to ap,time = %f\n",op_sim_time());
}
```

代码即添加完成，编译运行程序。仿真输出结果为

```
user1 source make data packet,time=0.000000
user2 source make data packet,time=0.000000
user3 source make data packet,time=0.000000
user1 mac receive data packet from source and send to ap,time = 0.001000
user2 mac receive data packet from source and send to ap,time = 0.002000
user3 mac receive data packet from source and send to ap,time = 0.003000
```

可以看出 RAU 节点又不可以接收冲突的数据包了。