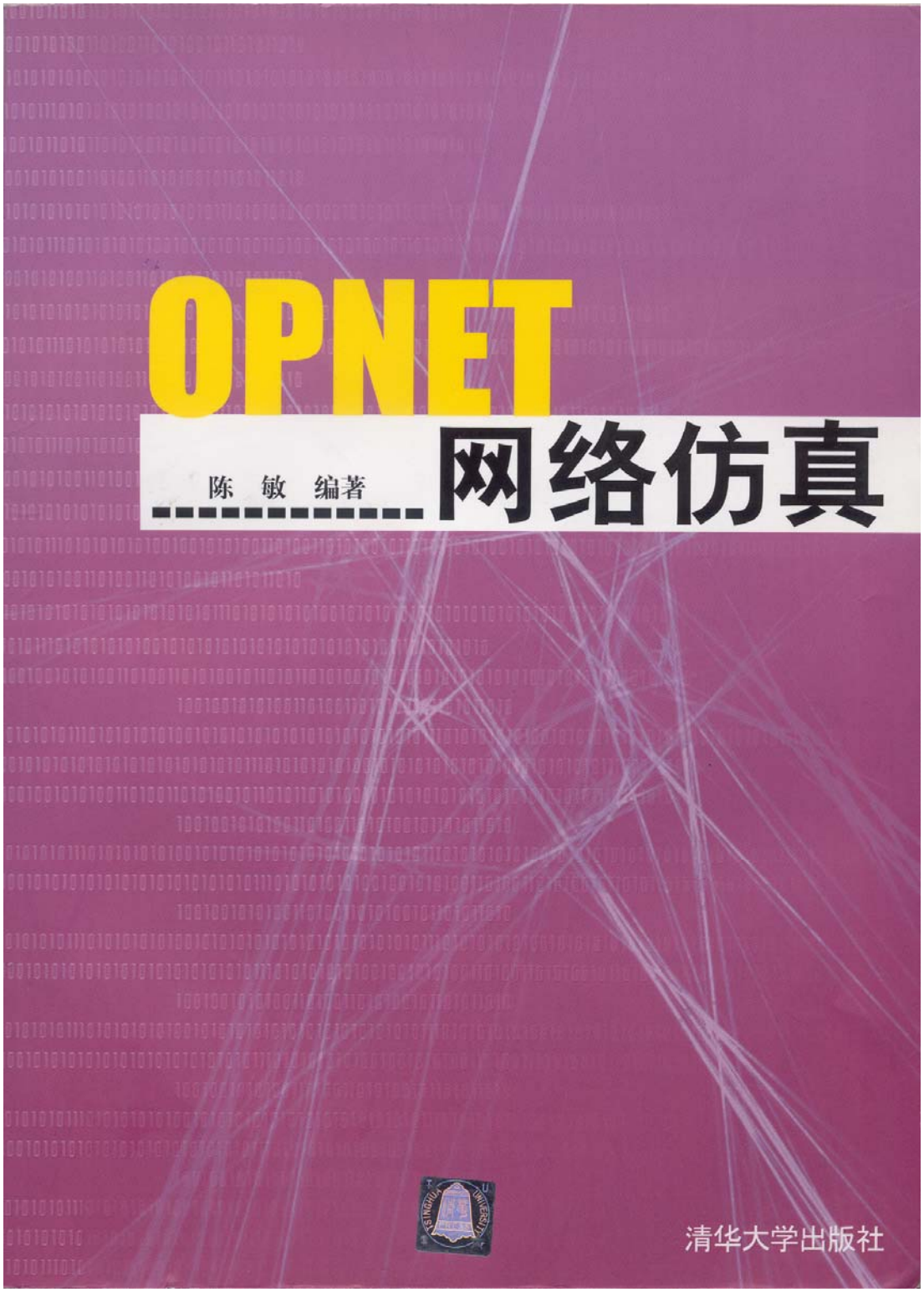


作者简介：**Min Chen** was born on Dec. 1980. He received BS, MS and Ph.D degree from Dept. of Electronic Engineering, South China University of Technology, in 1999, 2001 and 2004, respectively. He is a postdoctoral fellow in Communications Group, Dept. of Electrical and Computer Engineering, University of British Columbia since from 2006 to 2008 http://www.ece.ubc.ca/~minchen/. He was a postdoctoral fellow in School of Computer Science and Engineering, Seoul National University in 2004 and 2005 http://mmlab.snu.ac.kr/~mchen/. His current research interests include wireless sensor network, wireless ad hoc network, and video transmission over wireless networks.

引　　言

近年来，由于网络日趋复杂、网络规模日趋庞大，网络仿真技术应用于网络规划和设计的需求日渐强烈。我国网络仿真虽然起步较晚，但是网络的迅猛发展必将强劲地拉动网络仿真技术的研究和应用。未来数年将是网络仿真技术蓬勃发展的时期，今后网络仿真必将成为数据网络规划设计不可缺少的环节。

如何有效逼真地对复杂的通信系统进行建模仿真是个富有挑战性的课题，网络仿真的种种困难随着 OPNET、NS2 等网络仿真工具的诞生迎刃而解，而 OPNET 是目前网络仿真工具中的很好的一个，但是由于操作复杂，使用者一般需要半年或以上时间的培训和自行研究才能熟练地掌握。针对这个问题，本书基于对 OPNET 软件的研究，由浅入深地介绍了用其进行通信仿真系统建模的整个过程。

本书第 1 章针对暂时没使用过 OPNET 的读者，侧重于对 Modeler 仿真环境和工具的纲要性介绍。通过一个简单的例子说明如何建立网络模块，包括定义拓扑，设定业务流量，收集统计量，分析仿真结果；第 2 章介绍 OPNET Modeler 的安装；作为 Modeler 用户，要创建自己新的协议模块，首先必须了解 OPNET 采用何种仿真机制，第 3 章介绍了一些 OPNET 主要的通信机制，比如基于事件出发的模拟方式如何，封包是怎样传输的；第 4 章介绍 OPNET 中所有的编译器；第 5 章介绍如何收集、查看、分析及发布仿真结果的有关操作；第 6 章读者将简单地建立网络模型、节点模型和进程模块，并收集统计量及分析仿真结果，从而了解 OPNET Modeler 编程的基本流程，接着描述了 OPNET 各类核心函数功能如何，在何处用及怎么用，之后介绍了动态进程的编程技巧；以后的章节重点放在 OPNET 使用的高级技巧及高级应用上，这也是本书的主要内容所在。当把网络模型建立起来后，却运行出错，或者仿真结果不是所要，第 7 章将告诉我们如何使用 OPNET 自带的调试功能，如何与 VC 结合联调；第 8 章介绍如何对业务进行模拟，如何选择并组合不同的业务建模技巧，如何在仿真精确度和时间上达到一个最好的平衡点；第 9 章介绍无线建模的功能和所支持的移动方式有哪些，无线链路中各种各样的影响在 OPNET 中如何体现，应该

如何修改。由于无线固有的广播传输方式带来的仿真时间急剧增加，那么如何缩短仿真时间和提高仿真效率；第 10 章重点介绍了当前热门的无线 IEEE 802.11 模块；针对某些读者对动画的强烈兴趣，第 11 章将介绍如何对动画进行编程；第 12 章将通过一个自定义流体协议教程将读者带入 OPNET 应用层的内部构架；第 13 章介绍如何建立一个简单的 IP 协议；第 14 章和第 15 章对 EMA 和 HLA 的应用性讲解相信会受到某些读者的强烈关注。本书最大特点是尽量配合 Modeler 实际操作并参杂作者大量的实际经验，从而使读者可以较好地理解 OPNET 当中的运作。

陈敏

2004 年 于 广州

目　　录

第 1 部分　OPNET Modeler 简介

第 1 章　OPNET 仿真概述

1.1　网络仿真简介

在今天的信息技术时代，网络结构和规模日趋复杂庞大，表现在多种类型的网络日益走向融合，业务种类增加，网络负载日益繁重，新的网络技术也层出不穷，因此如何对现有网络进行优化设计和规划是个非常富有挑战性的课题。

对于企业网络，在建设网络、开展网上业务之前，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。对于运营商网络，面对用户的增加，新业务的推出以及新的网络技术的出现，技术人员和网管如果需要知道可能给网络带来瓶颈的原因是什么，是业务过于繁重、网络带宽不够还是服务器处理速度不高。如果网络上增设新的业务，对网络性能有什么影响。如果拟采用新的网络技术对网络进行升级，网络的性能会有多大幅度的改善，相比之下投入是否值得，新技术的引进是否会给网络性能带来负面影响；对于从事新协议的研发机构，如何有效逼真地模拟协议各种行为细节，如何构建接近真实有代表性的网络环境和业务，使得测试结果能够公正地评判新协议的性能?

无论是构建新网络，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。传统网络设计和规划方法主要靠经验，对复杂的大型网络，很多地方由于无法预知而抓不住设计的要点。因此越来越需要一种新的网络规划和设计手段。在这种情况下网络仿真作为一种新的网络规划和设计技术应运而生，它以其独有的方法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。网络仿真技术目前已经逐渐成为网络规划、设计和开发中的主流技术。

在国外，网络仿真技术的研究和应用已经有十多年的历史。以前主要用于网络协议和网络设备的开发和研究，使用者大都是大学和研究所的研究和开发人员，近年来网络仿真软件生产厂商近年来纷纷把应用和开发重点转向网络规划和设计方面，将用户由原来的研究开发人员转向网络规划和设计人员，另一方面网络仿真规划设计软件的使用和操作相当复杂，还远没有达到一般网络规划设计人员经过短时间培训就能够熟练使用的目标，因此国外网络仿真软件厂家正致力于简化软件界面和操作流程，强化软件的项目应用能力，特别是加强了与网络管理软件厂商的合作，开发与网管软件的接口，使得网络模型的建立逐步自动化，加快网络建模的速度。我国的网络仿真技术的研究 1999 年刚刚起步，这主要有两个原因，一个是我国数据网络的发展较晚，对网络仿真技术的需求相对不是十分迫切；另一个原因是主流的网络仿真软件基本上产自美国，而其高端产品在 1998 年以前一直是对包括中国在内的社会主义国家禁运。近年来，特别是 1998 年以来，由于我国数据网络的迅猛发展的拉动和美国解除高端网络仿真软件出口的限制的刺激，我国的网络仿真研究和应用逐步起步。

具体来说，网络仿真技术是一种通过建立网络设备、链路和协议模型, 并模拟网络流量的传输，从而获取网络设计或优化所需要的网络性能数据的仿真技术。从应用的角度上看，网络仿真技术有以下特点：（1）全新的模拟实验机理，使其具有在高度复杂的网络环境下得到高可信度结果的特点。网络仿真的预测功能是其他任何方法都无法比拟的；（2）使用范围广，既可以用于现有网络的优化和扩容，也可以用于新网络的设计，而且特别适用于大中型网络的设计和优化；（3）初期应用成本不高，而且建好的网络模型可以延续使用，后期投资还会不断下降。

网络仿真技术在网络规划设计方面的应用在我国刚刚起步，即使是在国外，也还只是处于初级阶段，特别是在大型网络和复杂网络的应用方面，尚存在不少重要的项目技术问题需要解决，部分已解决的问题也仍需进一步深入研究和探讨。

1.2 OPNET Modeler 仿真平台简介

OPNET 最早是在 1986 年由麻省理工大学的两个博士创建的，并发现网络模拟非常有价值，因此于 1987 年建立了商业化的 OPNET。目前共有大概 2700 个 OPNET 用户，包括企业、网络运营商、仪器配备厂商，以及军事、教育、银行、保险等领域。OPNET 近几年赢得的大量奖项是对其在网络仿真中所采用的精确模拟方式及其呈现结果的充分肯定。在设备制造领域，企业界如 Cisco，运营商如 AT&T，采用 OPNET 做各种各样的模拟和调试。在国防领域，主要被美国广泛采用，其他国家大多低调处理。在 OPNET 各种产品中，Modeler 几乎包含其他产品的功能，针对不同的领域，它表现出不同的用途：（1）对于企业网的模拟，Modeler 调用已经建好的标准模型组网。在某些业务达不到服务质量要求的情况下，如网上交易、数据库等业务响应时间慢于正常情况，Modeler 捕捉重要的流量进行分析，从业

务、网络、服务器三方面找出瓶颈；（2）对于比企业网更复杂的运营商（ISP）网的模拟，Modeler 焦点放在整个业务层、流量的模拟，使运营商有效查出业务配置中产生的错误，例如有哪个服务器配置不好，让黑客容易进攻，有哪些业务的参数配置不合适等情形；（3）针对研发的需求，Modeler 提供了一个开放的环境，使用户能够建立新的协议和配备，并且能够将细节定义并模拟出来。本书侧重于使用 Modeler 进行研发的场合，使得其能将深层次的细节完全精确模拟的特点体现出来，这是传统方式不足以做到的。

Modeler 所能应用的各种领域包括端到端结构（End to End Network Architecture Design）、系统级的仿真（System Level Simulation for Network Devices）、新的协议开发和优化（Protocol Development and Optimization）、网络和业务层配合如何达到最好的性能（Network Application Optimization and Deployment Analysis）。举例来说，在端到端结构上的应用中，从 IPv4 网络升级为 IPv6，采用哪种技术方式对转移效果比较好；新协议的开发，如目前流行的 3G 无线协议。在系统级的仿真中，分析一种新的路由或调度算法如何使路由器或者交换机达到 QoS；在网络和业务之间如何优化方面，可以分析新引进的业务对整个网络的影响，网络对业务的要求，实际中网络和业务是对矛盾，通过 Modeler 模拟来查找网络和业务之间所能达到最好的指标。

Modeler 采用阶层性的模拟方式（Hierarchical Network Modeling），从协议间关系看，节点模块建模完全符合 OSI 标准，业务层->TCP 层->IP 层->IP 封装层->ARP 层->MAC 层->物理层；从网络物件层次关系看，提供了三层建模机制，最底层为进程（Process）模型，以状态机来描述协议；其次为节点（Node）模型，由相应的协议模型构成，反映设备特性；最上层为网络模型。三层模型和实际的协议、设备、网络完全对应，全面反映了网络的相关特性。

Modeler 采用面向对象模拟方式（Object-oriented Modeling），每一类节点开始都采用相同的节点模型，再针对不同的对象，设置特定的参数。例如，配置多个 WLAN 工作站，它们采用相同的节点模块，界面上，可以设置不同的 IP 地址和 WLAN 参数。

基于事件出发的有限状态机建模（Finite State Machine Modeling），避免以时间出发，变成以事件出发的建模。采用离散事件驱动（Discrete Event Driven）的模拟机理，与时间驱动相比，计算效率得到了很大提高。例如在仿真路由协议时，如果要了解封包是否到达，不必要每隔很短时间去周期性地查看一次，而是收到封包，事件到达才去看。每一时刻，FSM 将停留在特定状态，之后收到事件，完成事件并跳转状态。例如路由协议要做的事有获取周边节点地址，建立拓扑信息，之后路由表稳定下来，在收到封包将其转发到下一个节点，这些事件中断将引起相应的状态转移。

在 Modeler 中所有代码，各种协议的代码都是完全公开（Total Openness），每一个代码的注释也是非常清楚，使得用户更容易理解协议的内部运作。

采用混合建模机制，把基于包的分析方法和基于统计的数学建模方法结合起来，既可得到非常细节的模拟结果，也大大提高了仿真效率。

仿真引擎的效率方面，Modeler 10.0 使整个仿真速度更加提高，同样一个仿真节省大约一半时间，同时引入并行仿真使得无论无线还是有线的仿真更加快速。

在物件拼盘中，包含了详尽的模型库（设备、链路及详细的协议），包括：路由器、交换机、服务器、客户机、ATM 设备、DSL 设备、ISDN 设备等，还有其他厂商提供的配备，随着 OPNET 版本的提高模型库也不断增加。

Modeler 也提供了多种业务模拟方式，具有丰富的收集分析统计量，查看动画和调试等功能。它可以直接收集常用的各个网络层次的性能统计参数，能够方便地编制和输出仿真报告，如图 1-1 所示。

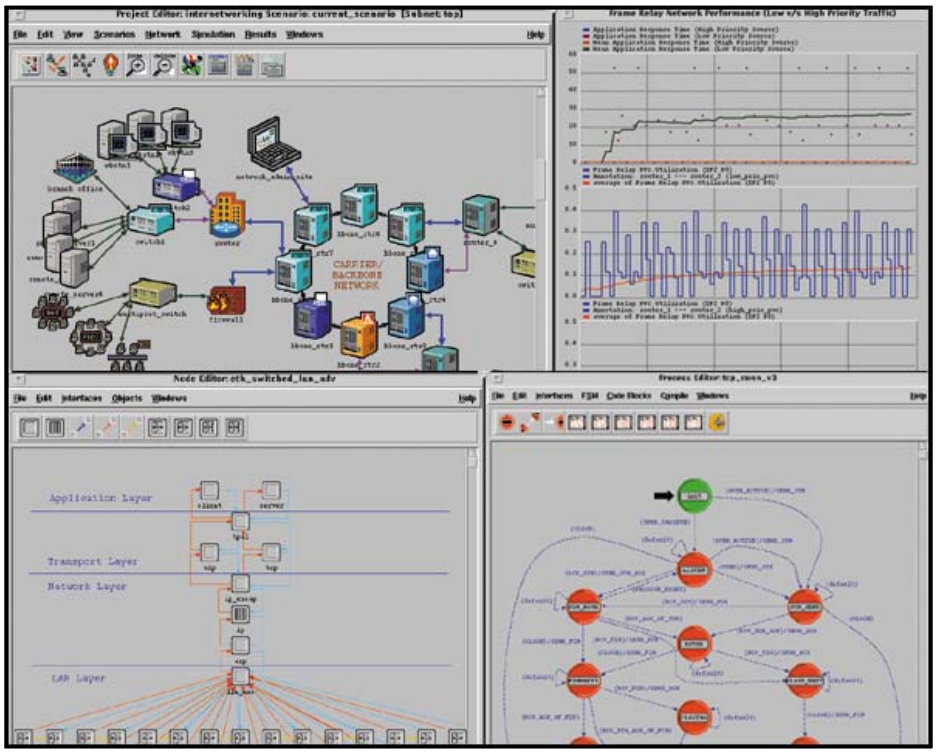


图 1-1　Modeler 层次化的建模机制和集成的建模、仿真和分析环境

使用 Modeler 仿真可以大体分成 6 个步骤，分别是配置网络拓扑（Topology），配置业务（Traffic），收集结果统计量（Statistics），运行仿真（Simulation），调试模块再次仿真（Re-simulation），最后发布结果和拓扑报告（Report），这些步骤可以总结为两个 t 两个 s 和两个 r。

为了让 Modeler 表现更加出色，功能更强，OPNET 提供了需要额外费用的附加的模组，在此对它们简单介绍：ACE（Application Characterization Environment）用来发现业务存在的问题，例如银行上网交易时，刷卡需要 5 秒的延时，ACE 能够发现每个封包的细节，并对各种业务协议的行为分解，以发现造成业务延时过大的原因；TMM（Terrain Modeling）提供无线仿真中对地形模拟的支持，导入电子地图可加入地形信息，将对无线传播模型进行更进一步地考虑，例如移动台经过一个山谷时，接收信号所受的影响；HLA（High Level Architecture）主要用于军事领域，采用分布式仿真方式使得多个异构网联成一个大系统；

MVI（Multi-Vendor Import）多厂商引进，用在企业和运营商设定复杂设备参数的场合，支持直接导入各种厂商的配置，包括设备参数和业务流量，不需要手动建立。假设有 100 台路由器，每个路由器有 10 个端口，每个端口参数不相同，这样手动一拉一点的参数输入方式可能需要 2 天时间，而完全自动导入只需几分钟时间；ESP（Expert Service Prediction）提供对网络性能预知的功能，例如运营商声称其 ADSL 网在 99%情况下提供 400kbps 以上的带宽，必需有一个凭据，为什么在不论多拥塞的情况下能够达到这种服务等级（SLA, Service Level Agreement）；Flow Analysis 流分析，收集流量和拓扑，分析流量阻塞，配合 Net-doctor（网络医生）检测 IP 地址配置，如果某个端口原本应该封闭而被打开，网络医生可以对业务进行安全性分析。

另外针对某些客户群建立了特别的模型库，例如 UMTS 和 IPv6 等模型，由于在整个开发过程中是和其他企业互相配合建立的，因此需另外加费。

OPNET 用户可以向一年一度的 OPNETWORK 会议投稿，论文投稿截止日期在每年的 8 月底。于 9 月底在美国的华盛顿召开，期间有各种各样免费培训，吃住免费并且不用交注册费。

1.3　OPNET 网络环境

这一节我们将初步接触 OPNET 工程编辑器，通过它将更好地了解 OPNET 网络环境，这将涉及两个相关的重要概念，工程（Project）与场景（Scenario）。在任何打开 OPNET 时候，最高层次永远为一个工程，每个工程底下的场景代表网络模块，每个场景都是具体的，当进行建模时，即使只有单独一个网络模块，也需要创建一个工程包含该场景。

具体来说，一个工程就是一组仿真环境，一个场景就是其中的一个具体网络仿真环境配置方案。场景是网络的一个实例，一种配置，如拓扑结构、协议、应用、流量以及仿真属性等设置。在 7.0 以前版本没有工程的概念，它的提出初衷是方便对不同的场景的仿真结果进行比较。工程提供场景复制功能，可以对场景进行备份，备份后的场景所有的配置及结果都相同，通过改变其中一个的参数，查看参数改变后对结果的影响，也可以使不同场景侧重系统的不同方面，验证系统在不同场合下的性能及是否存在瓶颈。

工程编辑器最开始用来新建一个工程，指定工程名字和第一个场景名字后，网络配置小精灵（Startup Wizard）就出来了，如图 1-2 所示。

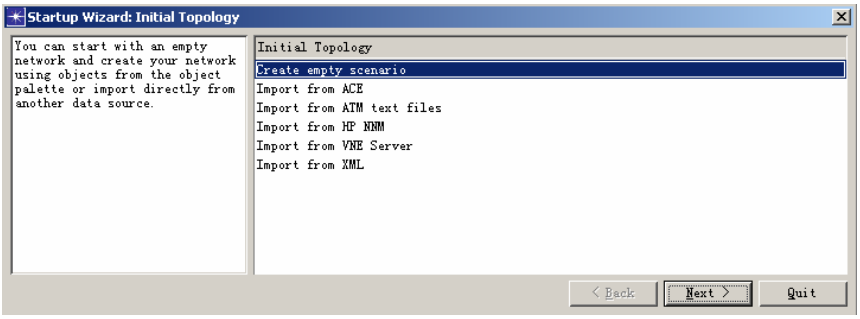


图 1-2　OPNET 中配置网络的小精灵

我们可以选择手动建立网络，或者可以从特殊格式文件导入。从设备配置文件中直接导入，需要多厂商引进的模块；从 XML 导入，可以将拓扑信息完全析取出来。我们一般建立一个空的场景，接下来根据网络的规模，选择全球网、企业或者校园网络，如图 1-3 所示。

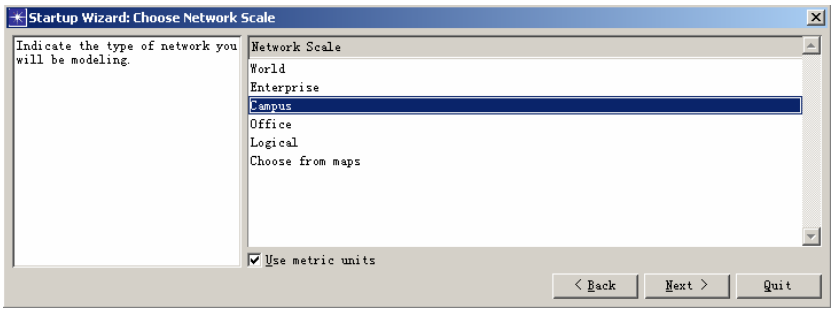


图 1-3　选择网络的规模

之后设定网络的范围，同时也可以指定度量单位，可以是经纬度、米、公里、英尺、英里等，如图 1-4 所示。

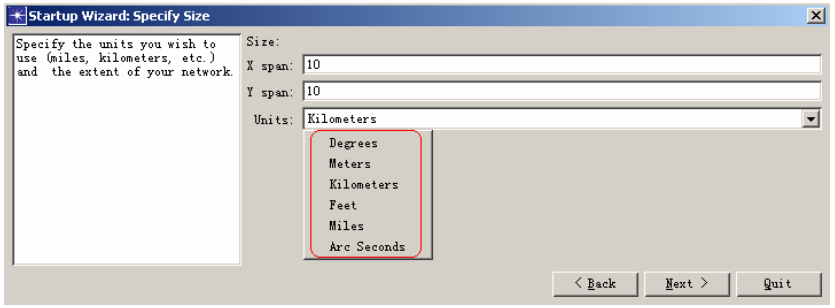


图 1-4　设定网络的范围

接着选择常用的模块家族（Model Family），把它们包含（include）进去后，Modeler 将自动建立新的物件拼盘  ，其中包含的内容就是所 include 的模块家族，里面所有物件将合并到物件拼盘中，如图 1-5 所示，它们作为构建网络的候选组件。

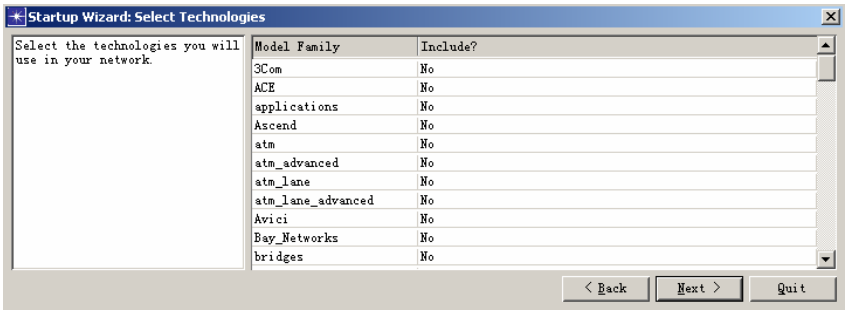


图 1-5　选择所需的模块家族

接下来看看任何一个网络拓扑中的背景信息及设置。就拿我们刚才设定的 10 平方公里规模的校园网为例。我们点击查看上一级拓扑按钮，可以看到在全球地图里放了一个子网，子网根据刚才设定的网络规模被命名为 Campus Network，如图 1-6 所示。

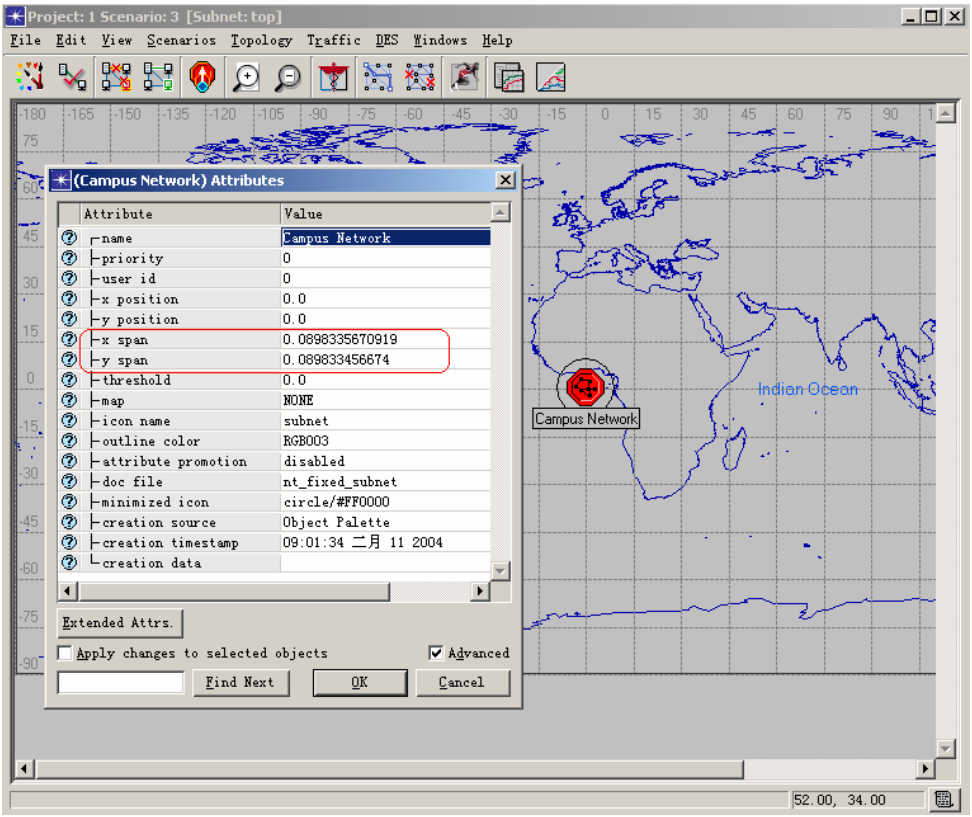


图 1-6　每个场景的最高层网络——全球网

子网所处在的经纬度都为 0。在子网里，我们已经指定了长宽 x span 和 y span 为 10km 的范围，而由于这一层是最高一层，它以经纬度为单位衡量子网的大小，从子网的属性看它的长宽为 0.089 经纬度，如图 1-6 所示，而换算成公里正好是 10km。

有益提示 全球级（world）网络是每个场景最高层的网络，称为 top，其度量单位一定是基于经纬度的，而其他底层网络可以选择不同的单位。

如果把子网移去不同地方，其经纬度也相应改变，双击它，就进入子网内部，可以知道整个子网的范围。

我们还可以用放大镜  对子网不断放大，这将看到蓝色的边框，它标识了整个子网的范围，如图 1-7 所示。

我们也可以设置背景的显示分辨率（Resolution：pixels/degree）和背景网格的解析率（Division）。分辨率如果调大一倍，则背景也相应变大一倍；解析率如果设为 0.1，则网格标识的单位精确到 0.1 度，如图 1-8 所示。

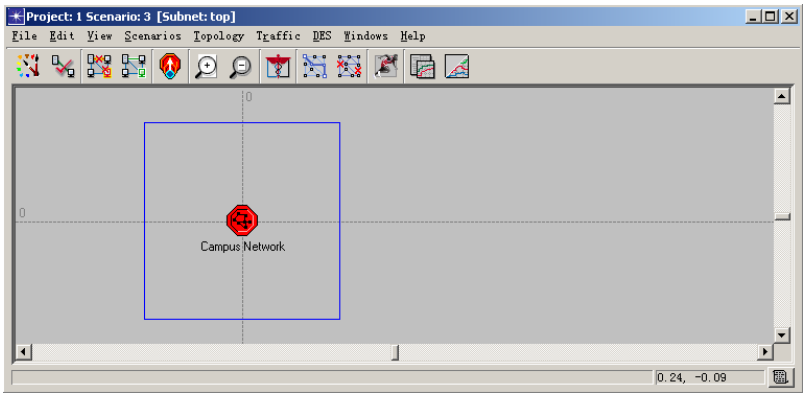


图 1-7　使用放大镜看到子网的边界

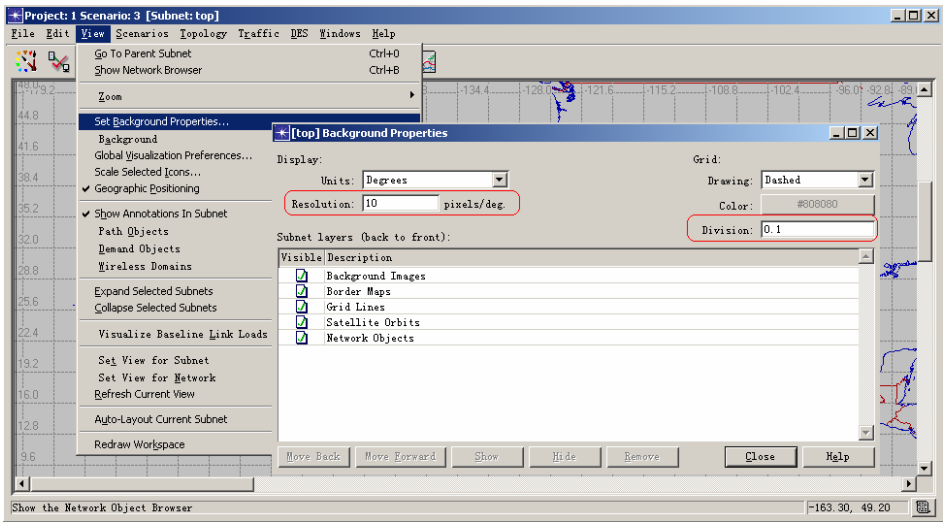


图 1-8　设定背景的分辨率和解析率

在背景中，也可以引进 OPNET 自带的地图。例如从在 View 菜单下选择 Add Image Map 可以导入一些带有经纬度信息的卫星地图（geotiff），它将按实际比例映射在全球背景中，

我们需要不断放大才能找到，图 1-9 所示为导入芝加哥地图的实例。

也可以选择 View -> Add MIF Map 导入海岸线或高速公路信息，如图 1-10 所示为导入 asia 海岸线的结果，在 Modeler 中可以看到这片区域变成绿色，如果放大可能看得到代表高速公路的线条。

所有背景图形，只作为参考，并不含有地理地形信息。放置地图时注意 View -> Background Properties 中设定的距离单位。

虽然 Modeler 中放大缩小操作非常简单，但是有时放大缩小之后，本来是个很清楚的图，但是要看到更大的背景范围时节点却叠加起来了，这是由于放大缩小操作只改变背景图形的分辨率，而节点图标分辨率保持不变，如图 1-11 所示。

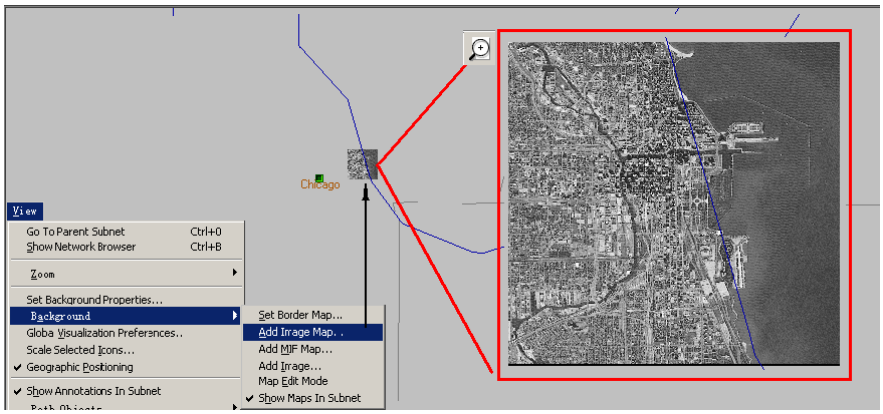


图 1-9　导入带经纬度信息的 geotiff 地图

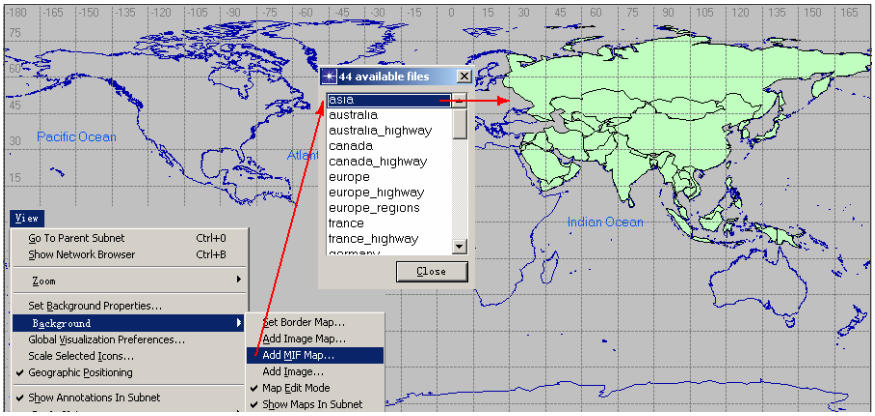
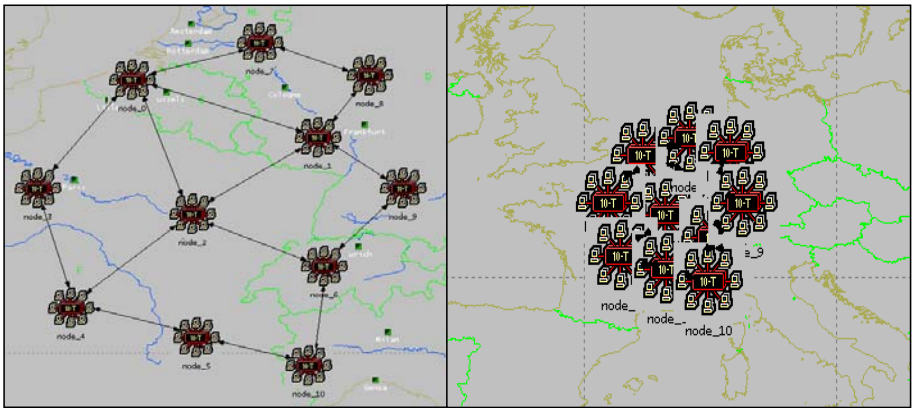


图 1-10　导入 MIF 地图



背景分辨率：80 pixel/degree　　背景分辨率：15 pixel/degree

物件显示的极限参数：10　　物件显示极限参数：10

图 1-11　场景被缩小后背景和节点图标的分辨率不匹配

这时我们需要调整物件显示的极限参数（Threshold），我们可以在物件的高级属性中（Advanced Edit Attributes）找到它。我们可以一次性选择所有节点（Select Similar Nodes），将极限设成 40，如图 1-12 所示。Threshold 只是一个相对值，不一定一次设置就能达到最好的显示效果，需要不断尝试，只要显示清楚就可以，其实节点图标重叠对仿真没有任何影响，只是看起来感觉混淆。

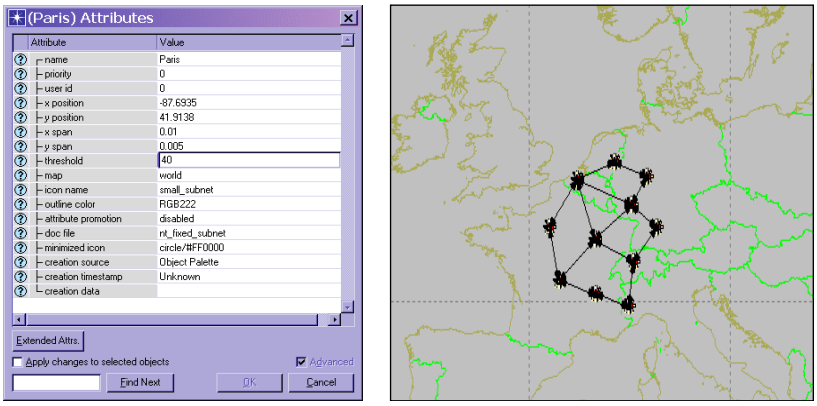


图 1-12　调整物件 Threshold 属性后的情形

有时为了加快仿真速度，可以在拓扑方面将子网简化为一个节点：

如图 1-13 所示，局域网由 10 个工作站、10 根有线链路、1 个交换机，共 21 个物件组成，如果仿真只关心整个局域网的性能，而不关注子网内工作站的表现，则可以用一个 LAN 节点代替，这样可以减少仿真事件数量，缩减仿真时间。

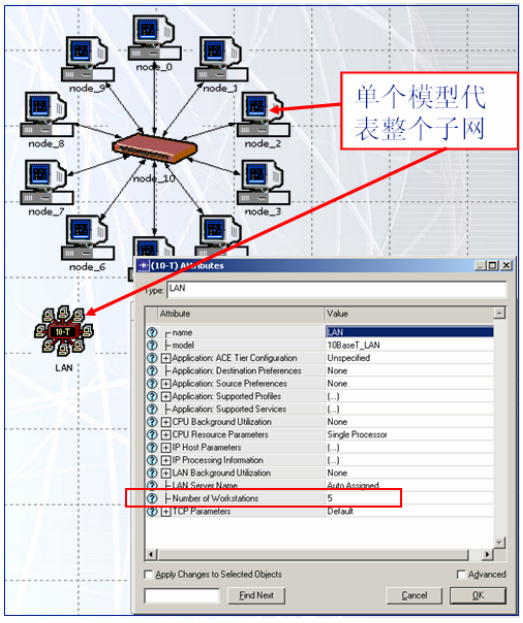


图 1-13　模块的简化

另外当我们进行端对端仿真时，看封包经历广域网后的延时，广域网包含很多服务器，但是我们不关心其内部结构，或者根本不知道其结构如何，而其内部也可以不完全模拟，这时 OPNET 常用一个 IP 云朵来代替， 云朵有丢包率和封包延时属性，代表所有业务经过广域网后产生的影响。

虽然我们可以通过一拉一点的方式去熟悉网络环境并定位到我们感兴趣的物件，但是对于一个上万个节点规模的网络，是否有更快的方式查找物件呢？如图 1-14 所示，在 View 菜单下选择 Show Network Browse 或使用 ctrl+B 快捷键将弹出场景物件浏览器，我们可以直接输入物件名称来查找。

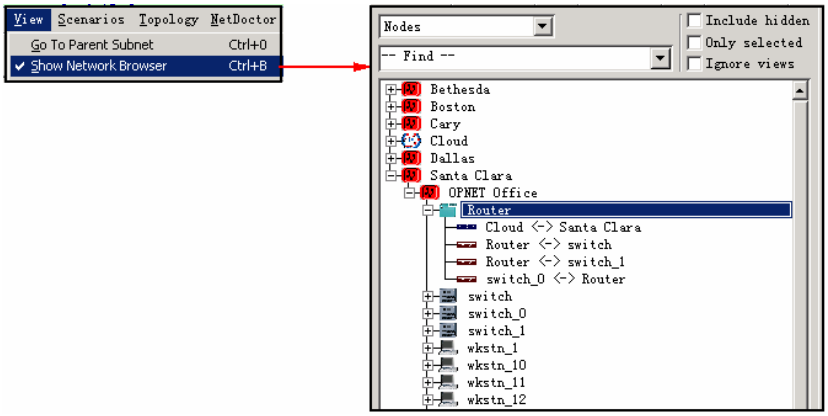


图 1-14　OPNET 的物件浏览器

有时候我们希望找出两个相似场景的细微差别，找出引起仿真结果不同的可能原因所在。OPNET 提供了场景比较的功能，如图 1-15 所示，从 Scenarios 菜单下选择 Network Differences，将生成网络配置区分的报告。

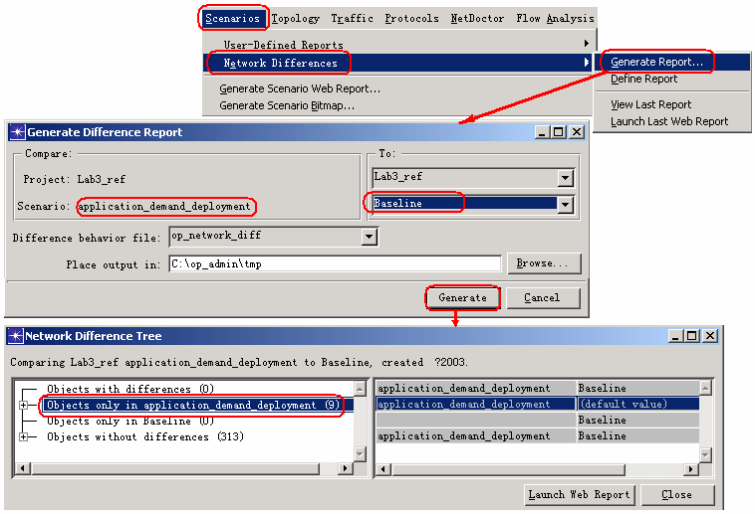


图 1-15　比较两个场景

1.4　OPNET 编辑器简介

当我们在 OPNET 菜单中新建或打开一个文件，将看到如图 1-16 所示的文件类型列表，这节对它们的特点进行概要性描述。

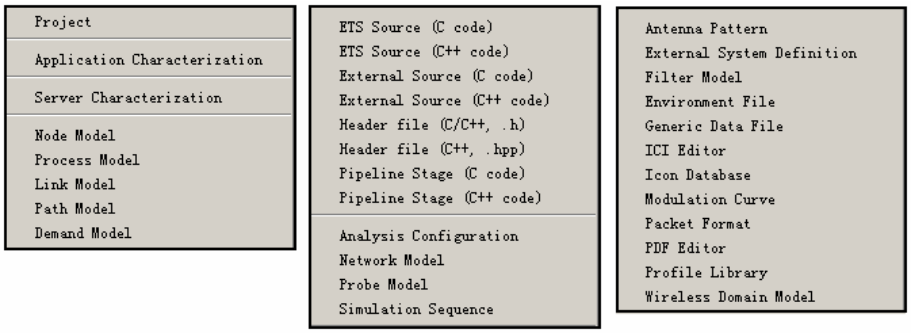


图 1-16　文件类型列表

（1）Project、Node Model、Process Model 分别为工程编辑器、节点编辑器和进程编辑器，分别对应与 OPNET 建模的三个阶层；

（2）Link Model 有线链路模型编辑器，设定链路的传输速率、支持的封包格式及采用哪些管道阶段来描述链路的物理特性；

（3）Path Model 用来显示流经过的路径，当我们配置了一个背景流量，仿真完成之后，会出现与路径模型相对应的路由表，通过路径模块显示一个流是怎么路由。在 Protocol 菜单中选择 IP -> demands -> display router for configure；

（4）Demand model 背景流量型，用来配置应用背景流或网络背景流，使用方法参见 8.4.2 与 8.4.3 节，另外有一个称为 demands 的物件拼盘，里面包含各种已经定义好的背景流模型。

（5）ETS Source 外部工具支持（External Tool Support）文件，代表为 OPNET 界面额外附加的一些功能键，由于 OPNET 现有菜单和工具栏提供的功能有限，如果不喜欢 OPNET 预定的界面，希望加入自定义的菜单和按钮，或需要开发新仿真环境，完全自定义一套新的仿真开发平台和分析工具，例如做光网规划可能涉及无线网环境构建的特殊功能，另外可以加入系统评估的功能，这就需要自己开发一个和 Modeler 完全不一样的仿真平台，而 ETS 用来定义这些平台。

（6）External Source 外部文件，其中包含若干个外部函数，主要用在进程模块中。一般进程模块只是本身用到函数在函数块（FB）中定义，如果某个函数在两个或以上的进程模块中用到，就应该定义成外部函数，使用的时候通过申明（declare external file）的方式将其包含进来。比如说 ip_support.ex.c 外部文件在 ip_dispatch 和 ip_icmp 进程中同时用到，也可以共享于多个进程模型之中，需要用时就申明。

（7）Head file 收集了所有头文件，使得对头文件进行修改和重建更加方便。

（8）Pipeline Stage 为管道阶段模型，描述物理层的表现，OPNET 的三种链路：有线点到点、总线和无线链路都是由相应管道阶段组成。

（9）Analysis configuration 结果分析编辑器，一般它所收藏的结果是在场景中隐藏起来的，点击 Hide/Show Graph Panels可以看到上次隐藏的结果。另外它有一个很重要的作用是查看两组结果之间的关系，新建一个结果分析文件，同时加载两个统计量结果，一个作为自变量，另一个作为因变量，也可以选择 Create a graph of two scalars with a third parameter 看三维显示。

（10）Network Model 网络模型文件，可以直接打开，在 OPNET 6.0 版本没有工程的概念时采用这种方式浏览网络环境，在 OPNET 7.0 以后的版本提出工程与场景的概念后，可在工程中导入该文件将成为一个仿真场景。

（11）Probe model 探针模型用来收集统计量。我们选择统计量的第一种方式是在工程上右点键，在 Node Statistics 中有一系列已经分好组的统计量可供选择，其实它们原本的名字 99%是从进程模块衍生出来，后来被提升到节点模块中来。如图 1-16 所示，首先我们选择统计量组别，例如全部有关 TCP 的统计都归为同一类，它们所属的类别是在节点模型中定义的，在统计量选择中看到的名字是提升后的名字。

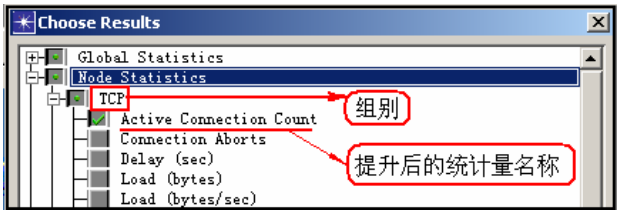


图 1-16　选择统计量组别与提升后的统计量

如果要收集未提升到网络层的统计，可以使用探针模型（Probe Model），它可以加入各种各样的统计，具体来说，点击相应的统计功能键可收集节点、链路、路径、背景流量、配对物件（Coupled Node）、属性的统计。其中配对物件表示逻辑连接上为一对，一般用于无线网络的统计，例如可指定收集节点 A 的第一个发送器和节点 B 的第一个接收器之间收发包的量。一般的统计量以时间为自变量，而属性探针可以使属性变量成为自变量，例如可观察节点发包率　对延时属性的影响。另外自定义动画也需要在探针模型中设置。

（12）Simulation Sequence 高级仿真配置文件，用来定义更加全面的仿真，该文件与高级仿真属性对话框的设定是互相关联的。可以创建一个仿真序列，其中可能包含多个仿真子集（set），仿真子集以蓝二横杆图标标识，每个仿真子集又包含多个仿真。当仿真正在进行中时，点击 Stop Sequence 将所有仿真停滞，Stop Set 停滞仿真组，Stop Run 停止当前仿真。

（13）Antenna Pattern 天线模型和 Modulation Curve 调制曲线用于无线模型，我们会在第 9 章对它们详述。

（14）Extern System Definition 定义 OPNET 与外部系统交互的信息格式，例如外部系

统传输数据的类型，以及从哪个系统成员到另一个成员，它作为模块形式放在节点模型中，看上去像是回字型的图标。

（15）Filter Model 自定义结果过滤器。OPNET 已经提供了 20 多种基本过滤器，用来对已得到结果进一步加工，使得结果更能突出某方面或更容易被理解。而过滤器模型支持创建新的过滤模块，我们可以对基本过滤器进行组合，也可定义全新的过滤器，对它编译之后就能在过滤器选择列表中看到它。

（16）Environment File 对应高级仿真属性 Files 选项卡，包含运行仿真的各种环境设定，主要有两个用途，一方面，如果下一个仿真要用到上一个仿真完全一样的设定就可以不需要手动设定而直接拷贝文件就行了；另一方面，如果只有一个 Modeler license，并且又在运行仿真，则不能对模型做编辑工作，这时可以采用执行带环境变量文件的命令行方式仿真：op_runsim -net_name <网络模型名称> -duration <仿真持续时间> -ef <环境变量文件名>，它只占用仿真执行 license，而不占用 Modeler 的 license，并且这样也可以提高仿真速度，最后还可以存成静态仿真文件*.sim。

（17）Generic Data File 是一个文本文件，OPNET 有一类标准的核心函数专门针对这种特殊文件的读取，并且可以检验内存和做一些预处理工作。

（18）ICI Editor，接口控制信息格式编辑器，ICI 可想象成是一种特别的封包格式，在一个节点模型中，ICI 可以将一些控制信息从一个进程模块捎带到另一个进程模块，如握手信息。

虽然通常比较建议能用包格式传输信息时尽量使用封包，但是封包局限在必需使用封包流，而 ICI 可与任何类型的中断进行绑定。另外在 OPNET 中很多的标准协议都采是用 ICI 交换握手信息，以逼近实际情况。

（19）Icon Database 图标库文件，支持自定义图标来做网络物件的标志。

（20）Packet Format 封包格式编译器，定义封包的域，包括域的类型、域的颜色等设定。

如果设定 information 类型，则该域只作为承载封包信息的标签，不记入包长；如果设定为 Packet 类型，则可实现包的封装，该域的大小为从上层传过来的封包大小；另外域还可以封装自定义的结构体 structure，它的 comments 属性可能保存了结构体的定义代码。

封包的读写根据编辑器设定的有名字的标签来操作，这样很方便。有时包的长度计算令人混淆，例如 structure 类型的域，其长度设定为 32bit，而实际数据量可能大于设定的比特数，或者将一个整型类型的域设定为 0bit，最后包长的计算完全不理会这些与实际情况有些偏差的设定，只是将所有由编辑器设定的域的大小相加（除了包的封装域大小根据实际包大小来定），为了更加逼近实际，我们在获取包的长度的时候，可能还要加上校验值（bulk size）。

（21）PDF Editor 为概率分布函数编辑器，OPNET 定义业务时通常使用了各种各样分布概率，OPNET 提供了一系列标准的概率分布函数，而该编辑器支持用户自定义概率分布函数。

一种是离散概率分布，如图 1-17 所示，假设封包的目的地有 4 种可能的选择，图中 50%

的封包发往第 0 个节点，15%的封包发往第 1 个节点，25%发往第 2 个节点，15%发往第 3 个节点；第二种连续概率分布。有时候可能一拉一点的设置不够精确，可以导成 EMA 文件进行再次编辑，它完全脱离 OPNET 工作环境，具体做法参见第 14 章，其实本节提及的所有模型都可以产生相应的 EMA 文件。

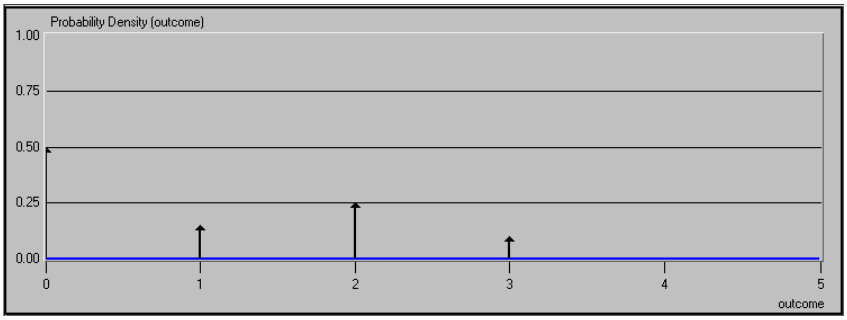


图 1-17　PDF Editor 编辑的离散概率分布

（22）Profile Library 对应高级仿真属性中的 Profiling 选项卡。

（23）Wireless Domain Model 无线区域模型，用来划分接收主询，是 OPNET 9.0 以后版本增添的新功能，参见 9.4.5 节。

1.5　实例讲解——配置一个简单的网络

目标　本课将使用 OPNET Modeler 快速创建一个网络，收集反映网络性能的统计结果，运行仿真，并且分析这些结果解决网络中存在问题。本课侧重于项目编辑器(Project Editor)的使用，展示 Modeler 建模和分析的功能。学会用 OPNET 来模拟仿真的基本过程。

在本例程的开始，我们列出一些需要用到的重要概念：

1．项目（Project）与场景（Scenario）

一个项目就是一组仿真环境，一个场景就是其中的一个仿真环境。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。在 Modeler 仿真时，最高层次永远是一个项目，每个项目底下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。项目的提出初衷是方便对不同的场景的仿真结果进行比较。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 子网和 TCP/IP 的子网不是同一个概念。OPNET 的子网是将网络中的一些元素抽象到一个对象中去。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。一个简单的例子，如运营商的骨干网，例如把骨干网上的所有路由器放到一个视图里，十分凌乱，不如按照省份将同一省份的路由器都放到同一个子网中，然后以省份的名称来命名每个子网的名字，构建成的网络看上去比较有条理。

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．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。

5．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。

6．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有限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。而仿真是基于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。

因此，从分层的角度来看仿真比模块的属性高。同样，仿真属性具备比模块属性更高的等级，代表最高级别的参数。

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。这种用法的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。

属性的提升至仿真属性有一个特殊的用途，就是可以成为序列仿真的输入变量。举一个例子来说明，有一个节点的模块属性为“traffic load”，如果需要通过改变网络负载来寻找最佳的网络吞吐量性能，这时有两种做法：

（1）每次改变“traffic load”参数，运行一次仿真，仿真完毕后记录这次的吞吐量结果。这种手动式的做法显然很麻烦，因为重复进行多次仿真，并且最后还要手动组合结果。

（2）将模块属性“traffic load”提升为仿真属性，设置需要测试的各种取值，然后运行仿真序列（Simulation Sequence）。这时 OPNET 会根据设定值的个数运行相应次数的仿真，每次仿真对应一种参数设置。这种自动式的做法只要运行一次仿真即能达到我们的目的，而且结果可以集合到一个文件中。

9．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，可以不惟一。

10．模型（model）、模块（module）与对象（object）

模型通常指的是进程模型、节点模型和网络模型等。

模块具有实在的物理含义，例如进程模块（processor），就是节点模型里面的小方块。

对象分为两种，一种是抽象对象，如复合属性；第二种是具体对象：例如模块（module）、节点、收信机、发信机。在 OPNET 中对象提出的目的是设置和获取它的属性，因此对象需要有它的对象 ID 号 Objid，作为程序获取对象属性的依据，一般通过 IMA 核心函数（以 Objid 为输入参数）获取或设置对象的属性。

值得一提的是进程模型（process model）没有 Objid，它不是一个对象，而是抽象概念，代表协议行为的逻辑关系，在没有被激活成进程时，对仿真没有任何意义。

因此模型（model）和对象（object）没有必然的联系。

1.5.1 定义问题

试想一下，你需要为公司内部互联网的扩展制定一个合理的方案。目前，公司在办公楼的第一层有一个星型拓扑的网络，现在要在第二层增加另一个星型拓扑网络。这时一个典型的“what-if”问题，所要解决的是确保增加的网络不会导致整个网络的连通失败，如图 1-18 所示。

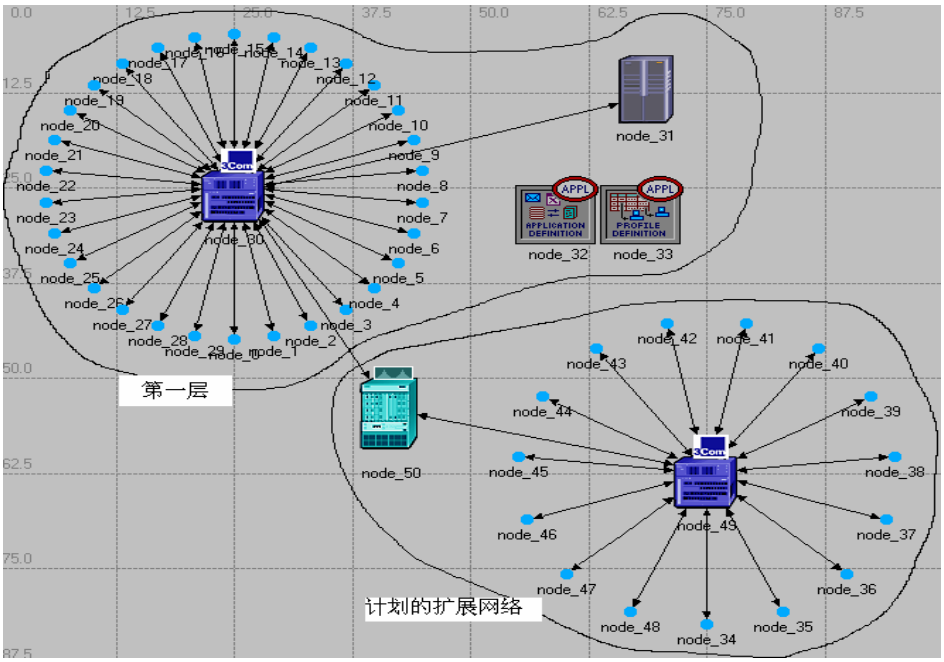


图 1-18　计划中扩展后的网络模型

1.5.2　建立网络拓扑结构

要创建一个新的网络模型，首先你需要创建一个新的项目和一个新的场景。

关键概念　一个项目为一系列场景的集合，而每个场景对网络建模的侧重点不同。

采用开始建立向导（Startup Wizard）来建立一个新的项目和一个新的场景。开始建立向导有以下几个步骤：

（1）选择网络拓扑类型。

（2）设定网络的范围和大小。

（3）设定网络背景图。

（4）选择对象模型家族。

有益提示　在创建一个新项目时会自动弹出开始向导，它可以用来设置网络的一些背景环境。

开始建立一个场景步骤如下：

（1）打开 Modeler。

（2）从 File 菜单中选择 New...。

（3）从弹出的下拉菜单中选择 Project 并单击 OK。

（4）将你的项目命名为<initials>_Sm，场景命名为<initials>_first_floor。

- <initials>用来区分同一项目的不同版本，比如你可以将项目命名为 1_sim。将场景命名为 1_first_floor。

（5）单击 OK 按钮。

- 这时出现开始向导，创建新的背景拓扑图，如图 1-19 所示。

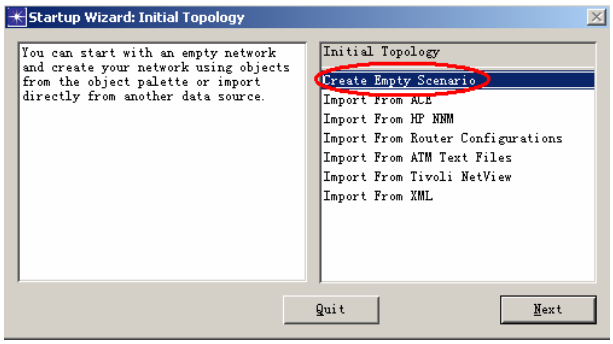


图 1-19 开始向导：创建新的背景拓扑图

- 选定网络的范围，哪图 1-20 所示。

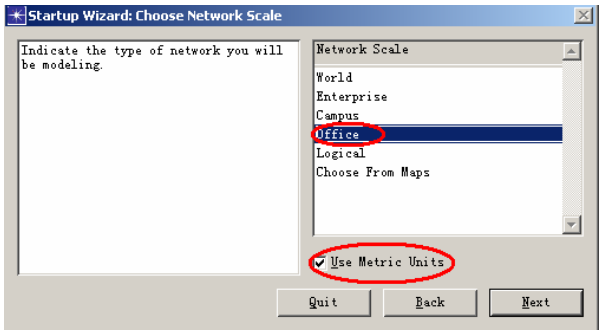


图 1-20 开始向导：选择网络范围

- 指定网络的大小，如图 1-21 所示。

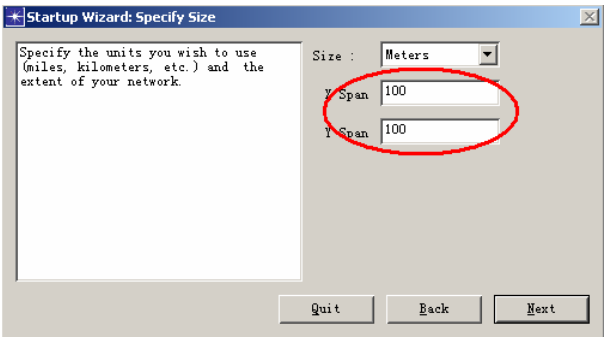


图 1-21 开始向导：指定网络大小

❑ 选择 OPNET 自带的对象模型家族种类，如图 1-22 所示。

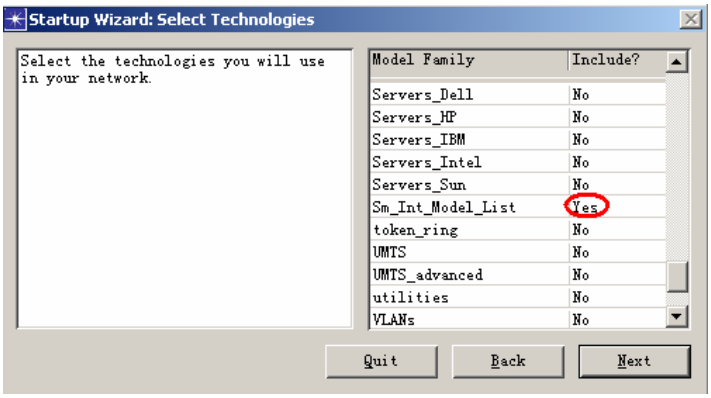


图 1-22　开始向导：选择对象模型家族种类

❑ 再次确认环境设置，如图 1-23 所示。

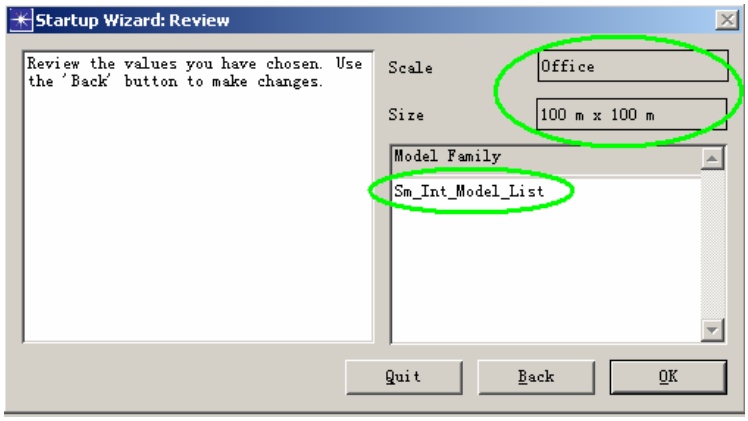


图 1-23　设置完毕的开始向导

这时出现大小和规格如同所指定的工作空间，同时弹出一个对象模板（包含刚刚选定的对象模型家族的所有模型）。

关键概念　通过对象模板中的节点和链路模型来创建网络模型

- ❑ 节点模型：代表实际的设备。
- ❑ 链路模型：代表连接设备的物理媒质，可以是电缆或者光缆。

可以通过对象模板中的图标直观地看出节点模型和链路模型。

关键概念　可以使用以下三种方法之一创建网络拓扑：

- ❑ 导入拓扑图。
- ❑ 从对象模板中选择模型并放置在工作空间中。
- ❑ 使用快速拓扑配置工具（Rapid Configuration）。

快速拓扑配置通过指定参数（节点模型和链路模型），一次性创建规则的拓扑结构：

（6）从 Topology 菜单中选择 Rapid Configuration。

（7）从配置下拉列表中选择 Star，单击 OK...，如图 1-24 所示。

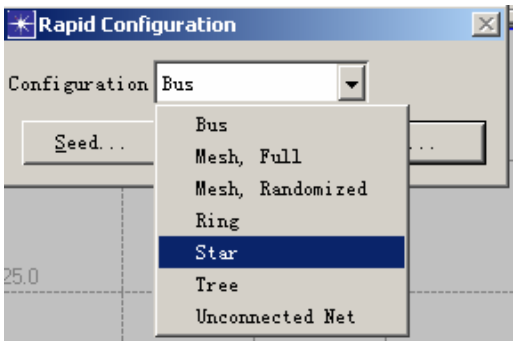


图 1-24　拓扑结构类型选择

接下来需要为网络指定节点模型和链路模型

OPNET 中标准模型的命名规则为：

<protocol1>_..._<protocoln>_<function>_<mod>其中<protocol>为模型用到的协议，可能同时用到几个协议<function>代表模型的大致功能<mod>模型派生类别。

（8）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。

（9）选择周边节点模型为 Sm_Int_wkstn，并设置节点个数为 30。

（10）选择链路模型为 10BaseT

（11）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。

（12）设置好单击 OK 按钮，如图 1-25 所示。

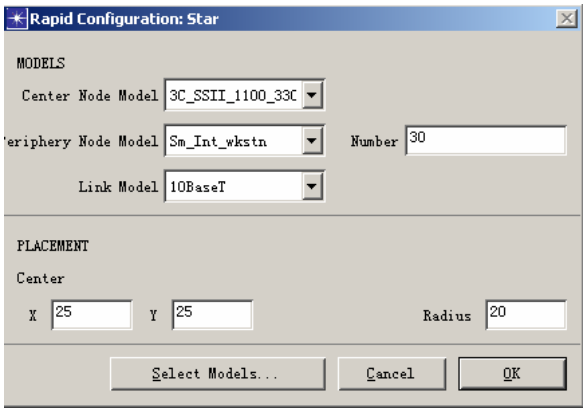


图 1-25　快速拓扑配置对话框

❑　>项目编辑器中出现如图 1-26 所示的网络拓扑。

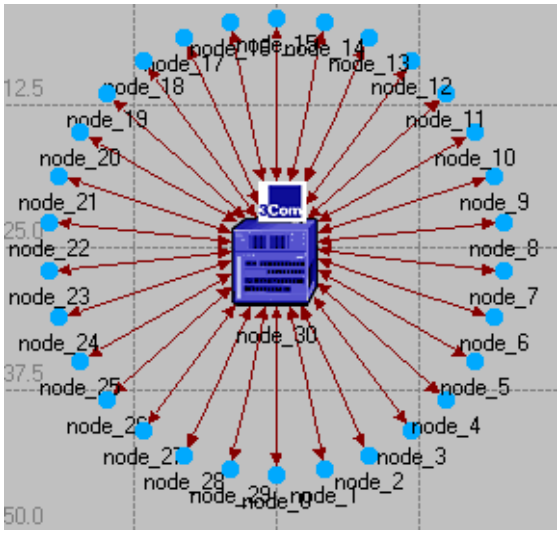


图 1-26　快速拓扑配置后的网络模型

接下来，需要扩展这个网络，首先增加一个服务器。这时将用到第二种创建网络模型的方法：在对象模板中选择模型并放置在工作空间内。

（13）打开对象模板。

（14）找到 Sm_Int_server 对象，并将它放置在工作空间中。

如果找不到该模型， 可能前面的操作不正确，你需要从左上角的下拉列表中选择 Sm_Int_Model_List 模型家族。

（15）单击右键，结束节点放置。

如果需要你可以多次单击鼠标左键，放置多个节点。

接下来，需要连接服务器和星型网络：

（16）在对象模板中找到 10BaseT 链路对象。

（17）在服务器上单击鼠标左键，移动光标，再单击星型网络的中心节点。

这时出现连接两个节点对象的链路。

（18）单击鼠标右键结束链路创建。

最后需要为网络配置业务，包括应用定义（Application definition）和业务规格定义（Profile definition），设置业务涉及的内容较复杂，本例程不作要求，因此模板中应用定义对象和业务规格定义对象的参数已经配置好（为 Light database 业务），只要将他们放置在工作空间中即可。

（19）在对象模板中找到 Sm_Application_Config 对象并将其放置在工作空间中。

（20）单击右键，光标重新移到对象模板中，单击 Sm_Profile_Config，并将其放置在工作空间中，单击鼠标右键。

（21）关闭对象模板。

这时得到如图 1-27 所示网络拓扑图。

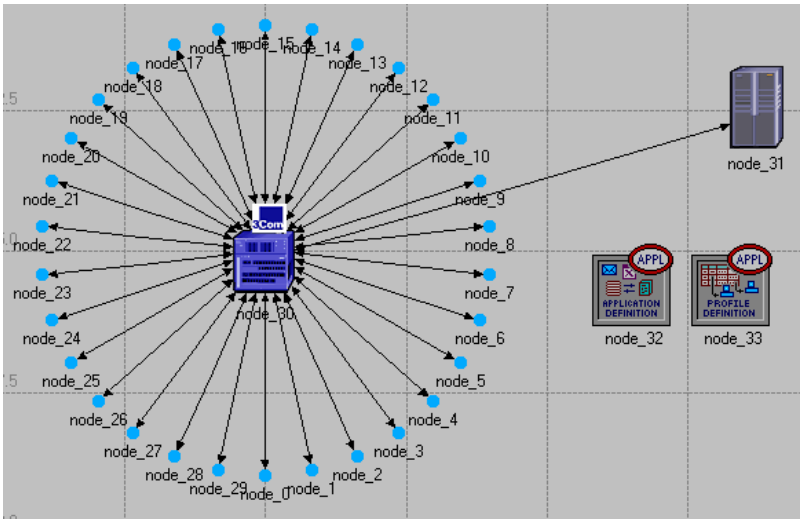


图 1-27　配置完毕的网络模型

接下来，需要收集统计结果。

首先，打开节点模型编辑器和进程模型编辑器。

关键概念

OPNET 采用三层建模机制，最底层为进程(Process)模型，以状态机来描述协议；其次为节点(Node)模型，由相应的协议模型构成，反映设备特性；最上层为网络模型。三层模型和实际的网络、设备、协议层次完全对应，全面反映了网络的相关特性；

每个网络对象（链路除外）都是一个节点模型，它由一个或多个模块(Modules)组成，模块与模块之间通过包流(Packet streams)或状态线相连。而模块实际上为进程模型，它通过状态转移图(STD, State Transition Diagram)来描述模块的行为。

现在让我们来看看第一层网络服务器的结构：

（22）在项目编辑器中鼠标双击 node_31(服务器节点）

这时打开一个新的节点模型编辑器窗口

如图 1-28 所示为以太网服务器的内部结构，它由几个模块以及连接模块的包流和状态线组成。

在仿真过程中，来自客户端的数据包被收信机 hub_rx_0_0 接收，然后由下至上穿过协议栈到 application 模块。经过处理后，又沿原路返回至发信机 hub_tx_0_0，最后被传输到客户端，如图 1-29 所示。

接下来，我们来看看传输适应层 tpal 模块的内容。

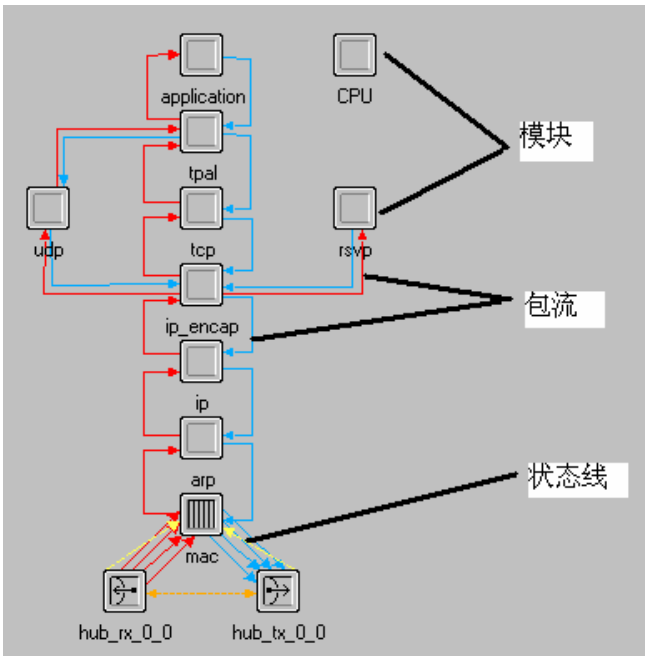


图 1-28　以太网服务器节点模型

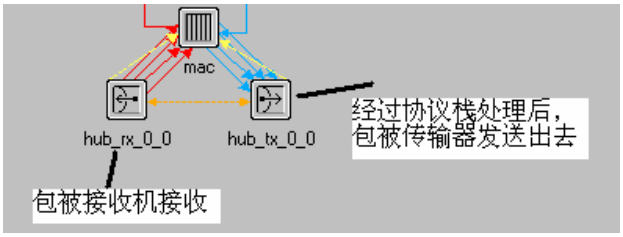


图 1-29　包的输入输出接口模块——收发信机

（23）在节点模型编辑器中的 tpal 模块  上双击鼠标。

这时打开一个新的进程模型编辑器，如图 1-30 所示。

（24）在 init 状态的上半部双击鼠标，打开它的入口代码。

（25）在 init 状态的下半部双击鼠标，打开它的出口代码。

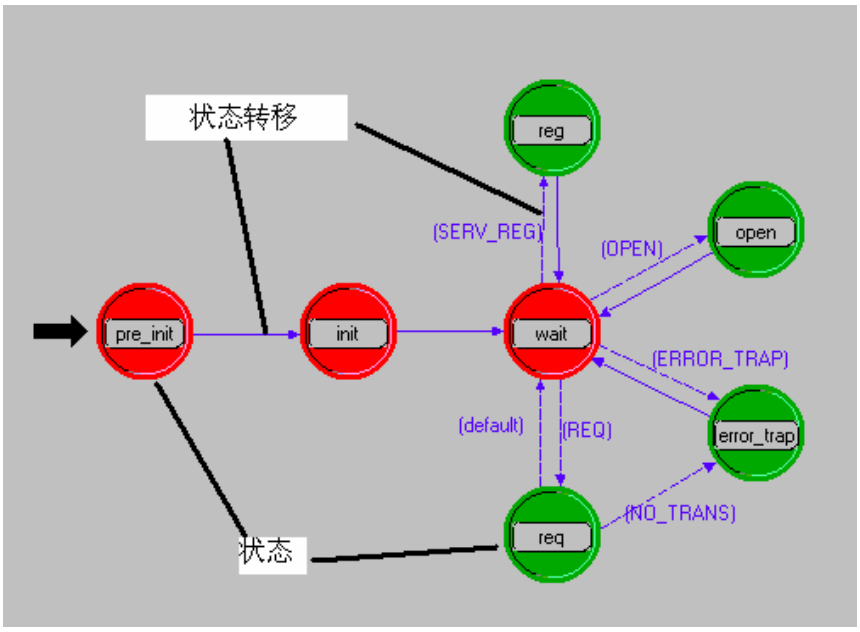
进程中的每个状态（图中红色的或绿色的圆圈）都包含一个入口代码（enter executive）和一个出口代码（exit executive），它们由 C/C++代码组成。入口代码在进入状态时执行，出口代码在离开状态时执行，如图 1-31 所示。

（26）关闭这两个代码窗口。

关键概念　状态与状态之间通过转移线 transitions 相连。转移线可以是带条件的（必须满足条件才能转移）或者无条件的（直接转移）。

图 1-32 包含两条转移线，一条是从 wait 状态到 open 状态的条件转移线（虚

线表示）。虚线中间的 OPEN 条件必须满足，wait 状态才能转移到 open 状态。然而，从 open 状态到 wait 状态的转移（实线表示）是无条件的，因此当执行完 open 状态的代码后立即转移到 wait 状态。



1-30　tpal 进程模型

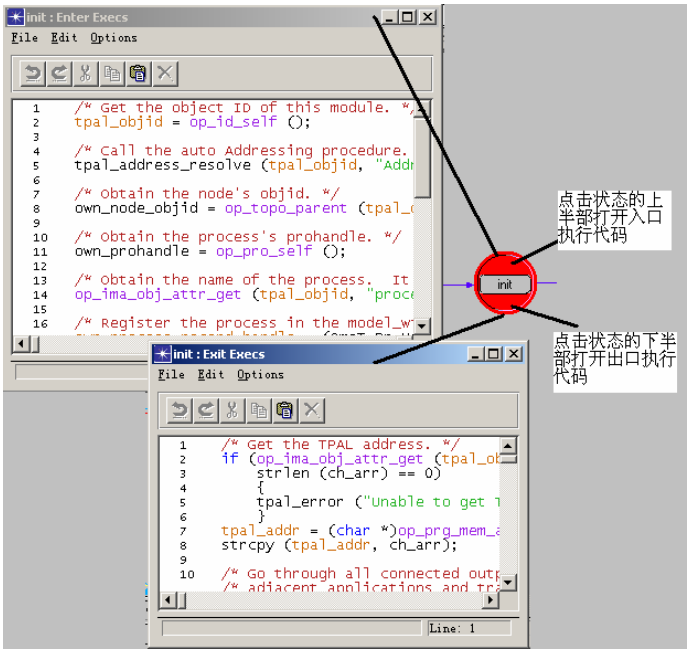


图 1-31　Init 状态的入口执行代码和出口执行代码

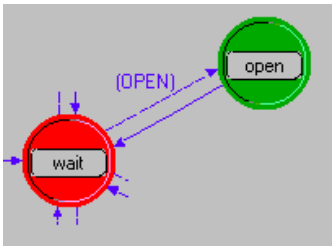


图 1-32 状态之间的条件转移

（27）关闭节点和进程模型编辑器。

1.5.3　收集统计量

关键概念　在网络模型中可以对单个对象收集统计量（Object statistics），也可以对整个网络收集全局统计量（Global statistics）。

到现在为止，已经建好了网络模型，现在要根据本教程最开始提出的问题决定收集哪些统计量：

（1）服务器有能力处理扩展网络的额外业务负载吗？

（2）一旦与扩展网络连接，整个网络的延时性能还能够接受吗？

为了找到这些问题的答案，需要选择一个对象统计量：Server Load 和一个全局统计量：Ethernet Delay。

服务器负载（Server Load）是整个网络的性能瓶颈。下面来收集与服务器负载相关的统计量：

（1）在服务器节点（node_31）上单击鼠标右键，从弹出的菜单中选择 Choose Individual Statistics。

这时出现 node_31 的选择统计量对话框，如图 1-35 所示。

有益提示　统计量对话框以树型结构显示统计量，可以清楚地了解它们的隶属关系。

（2）单击 Node Statistics->Ethernet，选择 Load(bits/sec)统计量，如图 1-35。

（3）单击 OK 关闭对话框。

全局统计量可以用来收集整个网络的信息。下面，我们通过选择全局 Delay 统计量来查看整个网络的延时性能。

（4）在网络编辑器的工作空间（避免指到对象）上单击鼠标右键，从弹出的菜单中选择 Choose Individual Statistics。

（5）单击 Gobal Statistics 树型结构，找到并点开 Ethernet 节点统计量。

（6）选中 Delay(sec)统计量。

（7）单击 OK 按钮关闭对话框。

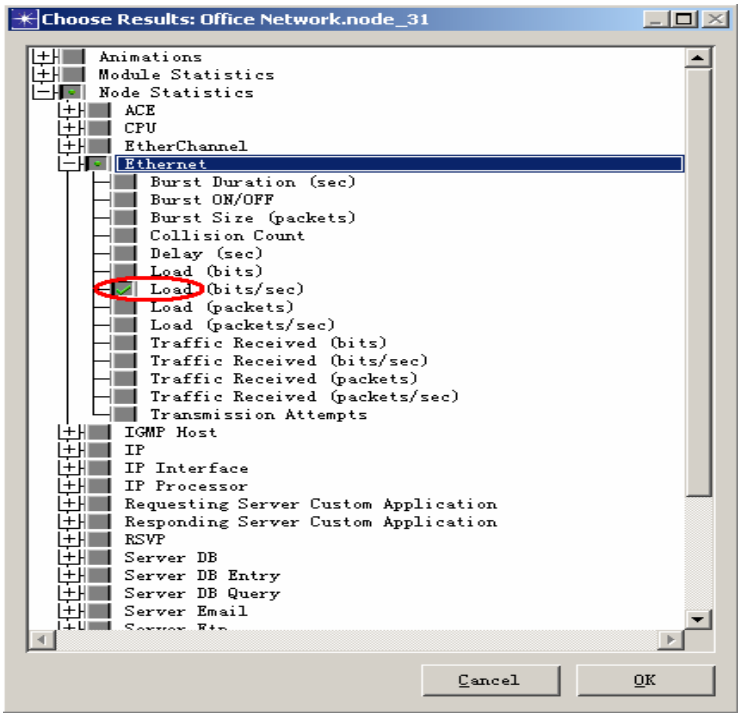


图 1-35　选择结果统计量

1.5.4　保存项目

接下来需要保存项目（最好养成经常保存项目的好习惯）。在 File 菜单中选择 Save（在前面已经给项目命名，因此不需要重命名）。

1.5.5　运行仿真

下一步，可以准备运行仿真了。首先，需要确定 repositories 属性设置正确：

（1）在 Edit 菜单中选择 Preferences。

（2）在查找文本框中输入“repositories”，单击 Find 按钮。

（3）在弹出的对话框的左下角单击 Insert 按钮，在文本框中输入 stdmod，然后回车。

（4）单击 OK 关闭 repositories 和 Preferences 对话框。

有益提示　优化仿真核心：

仿真核心有 development（调试）和 optimized（优化）两种。调试状态的仿真核心会收集仿真信息，这些信息可用来调试模块。而优化仿真核心使运行速度加快。

设置优化仿真的方法如下：

在 Edit 菜单中选择 Preferences，在查找文本框中输入 kernel_type，单击 Find 按钮。将对应的 value 设置为 optimized。

接下来运行仿真：

（1）在 Simulation 菜单中选择 Configure Simulationl...，或者在工具栏中选择运行仿真按钮。

（2）将仿真时间 Duration 设置为 0.5，即模拟执行半小时的仿真，如图 1-36 所示。

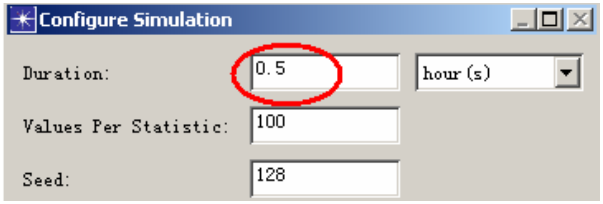


图 1-36　配置仿真属性

（3）单击 Run 按钮运行仿真。

（4）运行完毕后单击 Close 按钮关闭对话框。

1.5.6　查看结果

关键概念　可以从项目编辑器中弹出的菜单中选择 View Results 查看结果。

查看服务器 Ethernet load 结果：

（1）在服务器节点（node_31）上单击鼠标右键，从弹出的菜单中选择 View Results，这时出现查看结果对话框，如图 1-37 所示。

（2）然后选中 Load(bits/sec)。

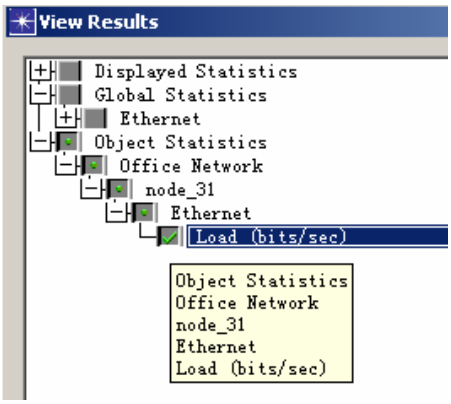


图 1-37　查看结果

（3）单击 Show 按钮，这时在项目编辑器上出现如图 1-38 所示的结果。

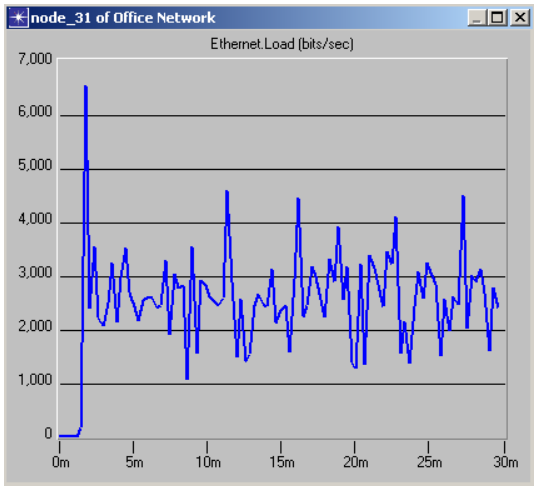


图 1-38　网络负载结果

- 不同的实验曲线走势应该是大致相同，当然具体的取值会因为节点放置的位置和链路长度不同而有微弱的差别。
- 注意到负载最大值为 6,000 bits/second。这个场景是我们想得到的值，用它和后面扩展网络后的结果进行比较，关闭对话框。

现在来查看 Ethernet Delay 的结果，这是一个全局统计量。

（4）在工作空间中单击鼠标右键，从弹出的菜单中选择 View Results。

（5）选择 Global Statistics→Ethernet→Delay(sec),然后单击 Show 按钮。

注意到网络收敛时的延时大约为 0.4 微秒，如图 1-39 所示。

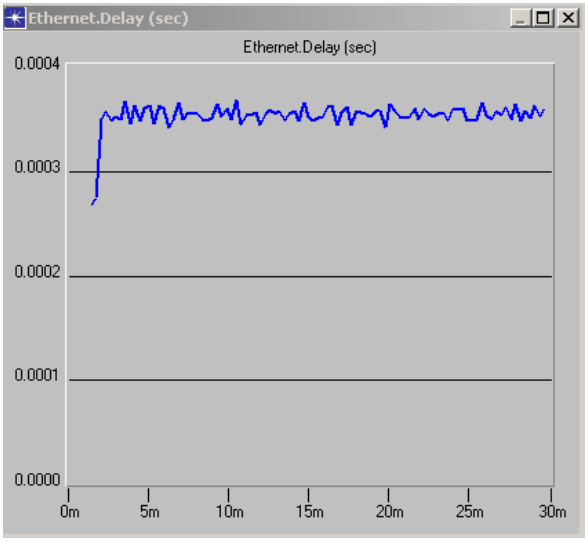


图 1-39　网络延时结果

（6）关闭结果图。

1.5.7　复制场景并扩展网络

你已经创建了一个基本的网络，并且收集了相关结果。现在可以扩展该网络并且验证在增加额外负载下，网络仍然能够很好地工作。

为了保留刚才的网络场景，以便和扩展的网络场景的仿真结果相比较，需要复制场景：

（1）在 Scenarios 菜单中选择 Duplicate Scenario...。

（2）命名新的场景为 expansion。

（3）单击 OK 按钮。

这时出现和刚才网络模型一模一样的场景。

接下来，需要构建网络的另一部分。

（4）从 Topology 菜单中选择 Rapid Configuration。

（5）从配置下拉列表中选择 Star，单击 OK...。

- 选择中心节点模型为 3C_SSII_1100_3300_4s_ae52_e48_ge3。
- 选择周边节点模型为 Sm_Int_wkstn，并设置节点个数为 15。
- 选择链路模型为 10BaseT。
- 指定网络在工作空间中放置的位置：中心的 X 轴坐标为 75 和 Y 轴坐标为 62.5。局域网的半径范围为 20。

（6）设置好以后单击 OK 按钮，这时项目编辑器中出现另一个局域网。

连接这两个局域网：

（7）单击对象模板工具按钮  。

（8）选中 Cisco 2514 路由器并将它放置在两个局域网之间。单击鼠标右键结束放置。

（9）在对象模板中选中 10BaseT 链路图标，在项目编辑器中分别连接 node_30 和 node_50（Cisco 路由器），以及 node_49 和 node_50。

（10）单击鼠标右键。

（11）关闭对象模板。

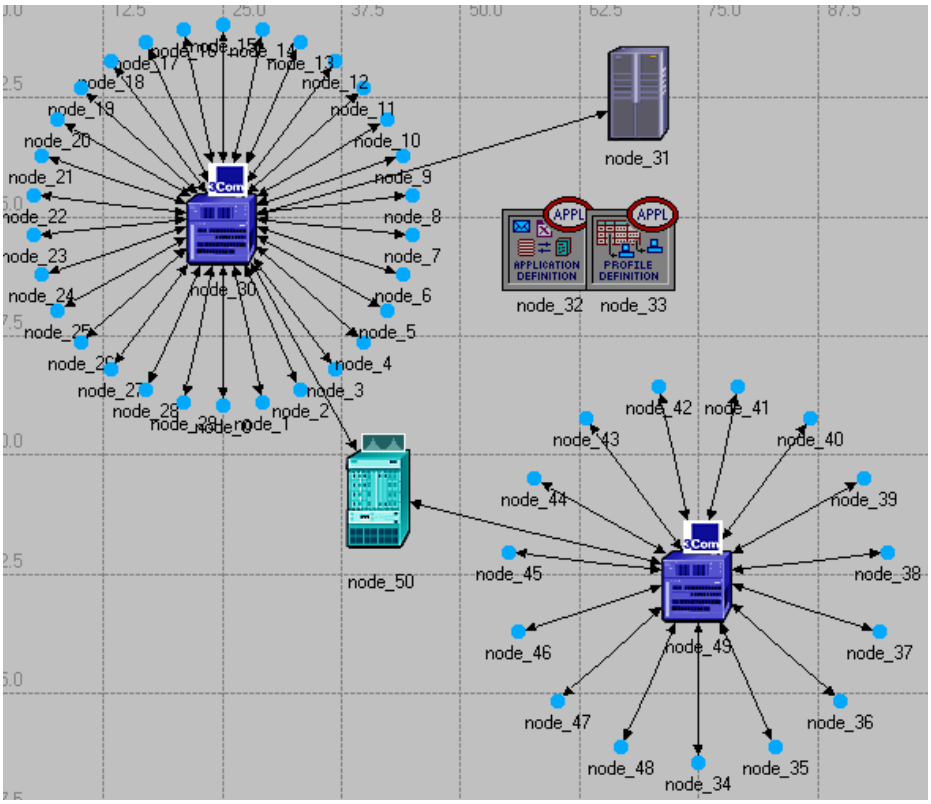
（12）在 File 菜单中选择 Save 保存项目。

这时整个网络建好了，如图 1-40 所示。

1.5.8　再次运行

现在可以运行仿真了。

（1）在 Simulation 菜单中选择 Configure Simulationl...，或者在工具栏中选择运行仿真按钮  。



1-40 扩展后的网络模型

（2）将仿真时间 Duration 设置为 0.5，即模拟执行半小时的仿真。

（3）单击 Run 按钮运行仿真。

（4）运行完毕后单击 Close 按钮关闭对话框。

1.5.9 比较结果

为了回答最开始提出的问题，需要将这两个网络的仿真结果进行比较：

（1）在服务器节点（node_31）上单击鼠标右键从弹出的菜单中选择 Compare Results。

（2）选中 Office Network.node_31→Ethernet→Load(bits/sec)结果统计量，并在比较结果对话框的右下角的下拉列表中选择 All Scenarios，如图 1-41 所示。

（3）单击 show 查看比较的结果。

图 1-42 中曲线抖动很厉害，为了更加清楚两条曲线的走势，我们可以改变结果的收集模式，从 Compare Results（如图 1-41）对话框中间下面的下拉列表中选择 time average，单击 show，这时出现图 1-43 的结果，可以看出抖动被平滑了。

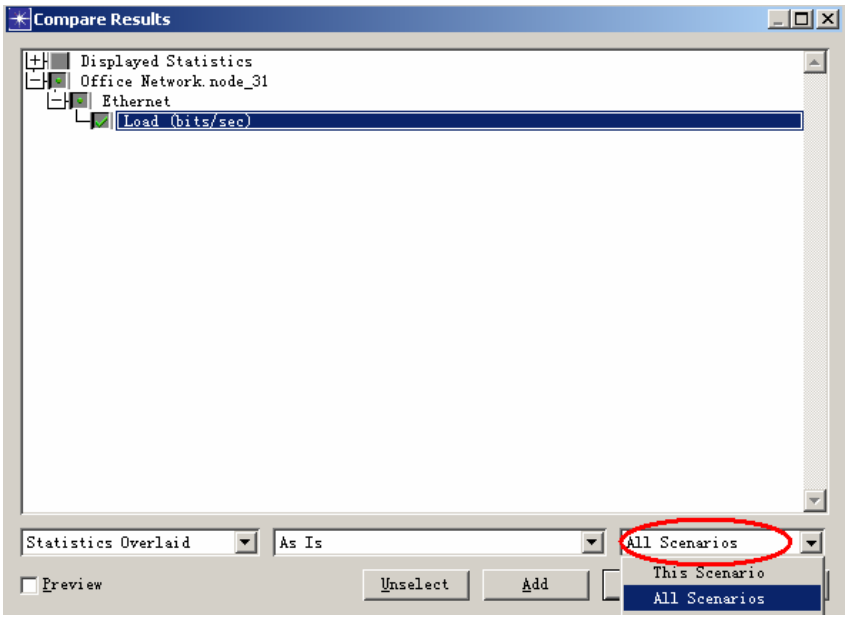


图 1-41　比较负载结果

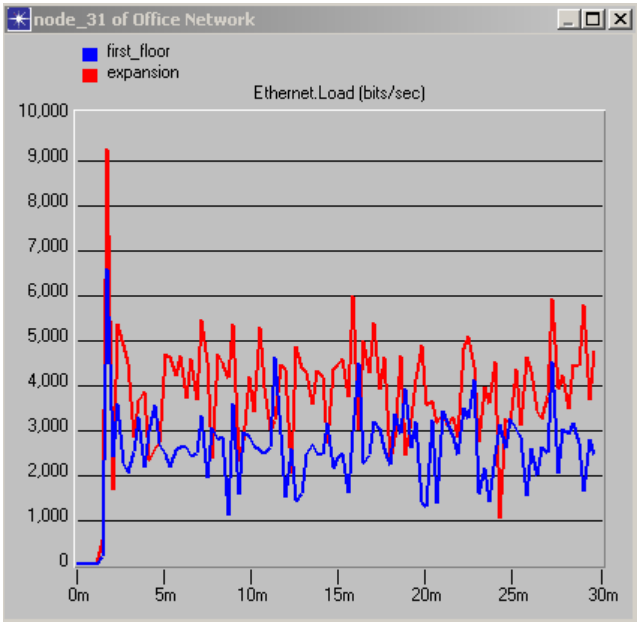


图 1-42　负载结果比较图

（4）关闭 server 的比较结果对话框。

最后，我们来查看增加第二个网络对网络的延时性能的影响。比较这两个场景的 Ethernet delay 结果：

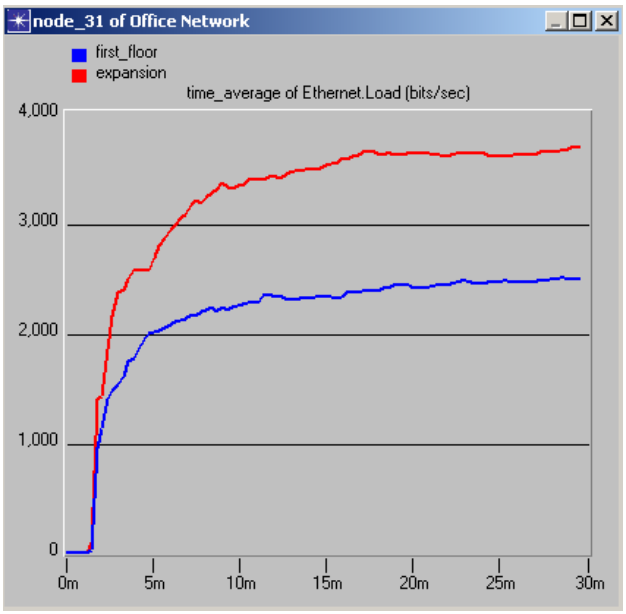


图 1-43　平均负载比较图

（5）在工作空间中单击鼠标右键，从弹出的菜单中选择 Compare Results。

（6）选择 Global Statistics→Ethernet→Delay(sec)统计量。

（7）单击 show 显示比较结果，如图 1-44 所示。

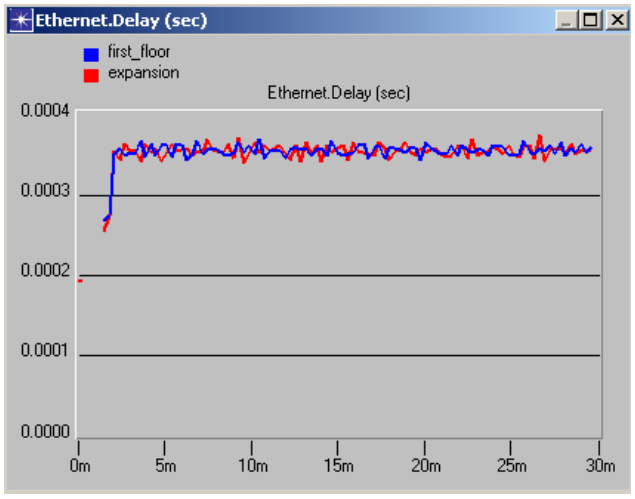


图 1-44　端对端延时比较

从图中可以看出以太网延时性能并没有因为服务器负载的增加而受影响。

（8）关闭比较结果对话框。

（9）从 File 菜单中选择 Close，保存并关闭项目文件。

第 2 章　OPNET Modeler 环境变量的设置及文件管理

2.1　OPNET Modeler 环境变量的设置

Modeler 全部功能得以正常运作有赖于相关环境变量的正确设置。OPNET 支持 SUN、HP、IBM、SGI 工作站和一般 PC 等硬件设备，常用的操作系统为 UNIX 和 Windows 2000，本节分别给出这两操作系统下有关 OPNET 环境变量的设定。需设置的内容包括与操作系统相关的用户环境变量和在 OPNET 中单独设置的与 c 编译器相关的一些环境变量。

2.1.1　Windows 2000 下环境变量的设置

在 Windows 下，可以采用以下两种方法打开如图 2-1 所示的用户环境变量设置对话框：

- 控制面板→系统→高级→环境变量。
- 在“我的电脑”上单击鼠标右键→属性→高级选项卡→环境变量，如图 2-1 所示。

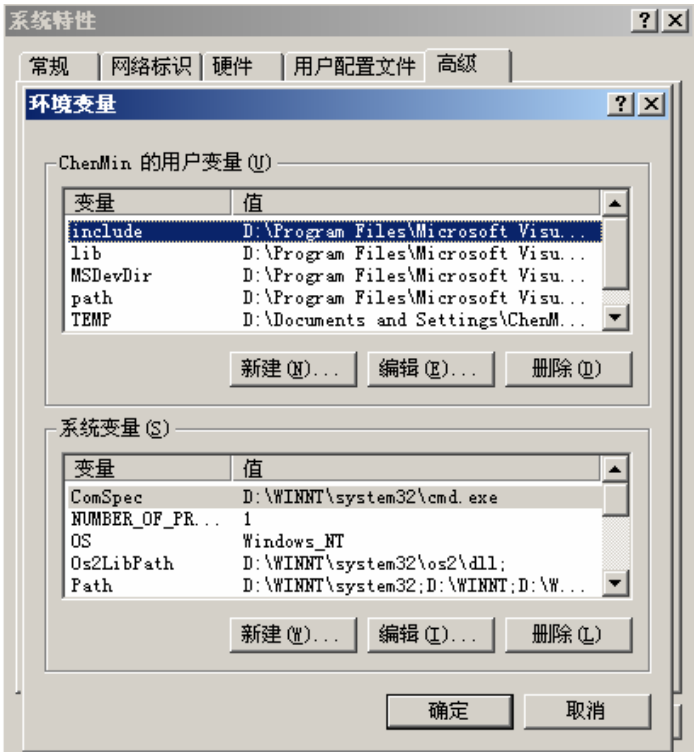


图 2-1 操作系统环境变量对话框

下面列出需要设置的环境变量，其中<opnet_dir>表示 OPNET 安装目录，<version_num>表示版本号：

（1）include(包含的头文件目录，包括三类)

- VC 的 include 文件：

C:\Program Files\Microsoft Visual Studio\VC98\atl\include；

C:\Program Files\Microsoft Visual Studio\VC98\mfc\include；

C:\Program Files\Microsoft Visual Studio\VC98\include；

- OPNET 的 include 文件：

<opnet_dir>\<version_num>\sys\include；

<opnet_dir>\<version_num>\models\std\include；

- 自定义头文件目录：

（2）Lib（包含的库文件目录）：

- VC 的库文件：

C:\Program Files\Microsoft Visual Studio\VC98\mfc\lib；

C:\Program Files\Microsoft Visual Studio\VC98\lib；

- OPNET 的库文件：

<opnet_dir>\<version_num>\sys\lib；

<opnet_dir>\<version_num>\sys\pc_intel_win32\lib；

（3）path（路径文件目录）：

- VC 的路径文件目录：

C:\Program Files\Microsoft Visual Studio\Common\Tools\WinNT；

C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin；

C:\Program Files\Microsoft Visual Studio\Common\Tools；

C:\Program Files\Microsoft Visual Studio\VC98\bin；

- OPNET 的路径文件目录：

<opnet_dir>\<version_num>\sys\pc_intel_win32\bin。

有益提示

VC 安装过程中选择自动注册环境变量，运行 OPNET 如果出现编译错误“bind_so_msvc: Unable to execute bind program (Win32 error code: 2)”则需要检查 Path 环境变量是否设置，新设定的环境变量需重新启动计算机才能生效。

除了设定以上与操作系统相关的用户环境变量外，还有一些需要单独在 OPNET 的 Edit->Preference 下设置的与 c 编译器相关的一些环境变量，它们用来实现 OPNET 与 VC 的联合调试，具体内容在 7.3.2 节有详细描述。

2.1.2　Unix 下环境变量的设置

（1）Unix 下主要设置用户环境变量文件“.cshrc”，它是每次打开一个用户终端（console）时系统执行的批处理文件。可以在.chsrc 文档中直接添加下述命令行，然后注销当前用户，再次登录，或者在 console 下输入这些命令行，其中<opnet_dir>表示安装目录，<version_num>表示版本号：

- set path=(/bin /usr/bin /usr/local/bin /usr/sbin /usr/ucb /export/home/gcr/op_models /<opnet_dir>/<version_num>/sys/unix/bin /etc .)

设置类似于 DOS 下的 path 命令，当一个指令在当前目录下找不到时将在这些 path 目录下搜索，其中空格键将多个 path 路径分隔开。

- setenv LD_LIBRARY_PATH /usr/local/lib

设置通用 c 支持库的路径。

- setenv LD_LIBRARY_PATH /<opnet_dir>/<version_num>/sys/sun_sparc_solaris/lib:$LD_LIBRARY_PATH

这条指令相当于设置 windows 底下的<opnet_dir>\<version_num>\sys\pc_intel_win32\bin

另外符号 :$ 表示增加一条新记录，否则当前记录将覆盖上一条记录。

（2）除了设定以上用户环境变量外，还需设置一些 OPNET 自身的环境变量，如表 2-1 所示：

表 2-1　OPNET 的 Edit->Preferences 中需要设置的环境变量

变量名	变量值	意义
comp_prog	comp_gcc 或 comp_g++	指定 c 编译器
comp_prog_cpp	comp_gcc 或 comp_g++	指定 c++编译器
bind_shobj_prog	bind_so_gcc 或 bind_so_g++	指定绑定共享库的程序
bind_static_prog	bind_gcc 或 bind_g++	指定连接静态库的程序

上表中 gcc 代表 Unix 下的一种常用的 c 编译器，g++为其 c++版本。

2.2　OPNET 常用文件格式

OPNET 仿真能够运行所必须使用的文件可以分为两类：

（1）以*.m 结尾的文件，如*.nt.m、*.pb.m、*.nd.m、*.pr.m、*.seq、*.prj、*.pk.m、*.ic.m 等；

（2）自定义的文件，如*.h、*.ex.c、*.ps.c、*.ex.cpp、*.ps.cpp、*.gdf。

表 2-2 列出 OPNET 常用的文件后缀及其说明。

表 2-2　OPNET 常用的文件后缀及其说明

后 缀 名	扩展描述	意 义	文件格式
.ac	Analysis Configuration	分析配置文件	二进制文件
.ad.m	Public Attribute Description	公共属性描述	二进制文件
.ah	Animation History	动画文件	二进制文件
.as	Animation Script	动画描述	ASCII 数据
.bkg.i	Background Image	背景图片	二进制文件
.cds	Cartographic Data Set	绘图数据集	二进制文件
.cml	Custom Model List	自定义模型列表	ASCII 数据
.csv	Comma Separated values	分栏数据文件	ASCII 数据
.ef	Environment File	环境文件	ASCII 数据
.em.c	Ema C code	EMA C 代码	C 代码
.em.cpp	Ema C++ code	EMA C++代码	C++代码
.em.o	Ema code object	EMA 目标文件	目标代码
.em.x	Ema Application executable	EMA 可执行文件	可执行程序
.ets	External Tool Support File	外部工具支持文件	ASCII 数据
.ets.c	External Tool Support C code	外部工具支持 C 代码	C 代码
.ets.o	External Tool Support object	外部工具支持目标文件	目标代码
.ets.cpp	External Tool Support C++ code	外部工具支持 C++代码	C++代码
.ex.c	External C Code	外部 C 代码	C 代码
.ex.cpp	External C++ Code	外部 C++代码	C++代码
.ex.h	External Code Include File	外部头文件	C/C++代码
.ex.o	External Code object	外部目标文件	目标代码

续表

后　缀　名	扩展描述	意　　义	文件格式
.fl.m	Filter Model	过滤器模型文件	二进制文件
.fl.x	Filter Model	过滤器模型可执行文件	可执行程序
.gdf	General Purpose Data File	通用数据文件	ASCII 数据
.h	Header File	头文件	C 代码
.hlp	Help File	帮助文件	文本文件
.ic.m	ICI Format	ICI 模型文件	二进制文件
.icons	Icon Database	图库文件	ASCII 数据
.lk.d	Derived Link Model	派生的链路模型	二进制文件
.lk.m	Link Model	链路模型	二进制文件
.map.i	Image Map	地图文件	二进制文件
.md.m	Modulation Function	调制曲线文件	二进制文件
.nd.d	Derived Node Model	派生的节点模型	二进制文件
.nd.m	Node Model	节点模型	二进制文件
.nt.m	Network Model	网络模型	二进制文件
.nt.so	Network Repository shared library	集成的网络目标文件	共享库文件
.orba	Satellite Node Orbit	卫星轨道文件	二进制文件
.os	Output Scalars	输出标量文件	二进制文件
.ov	Output Vectors	输出矢量文件	二进制文件
.path.d	Derived path object	派生的路径模型	二进制文件
.path.m	Path object	路径模型	二进制文件
.pa.ma	Antenna Pattern	天线模型	二进制文件
.pb.m	Probe List	探针模型	二进制文件
.pd.m	PDF/ editable form	概率密度函数	二进制文件
.pd.s	PDF / simulation loadable form	概率密度函数	二进制文件
.pk.m	Packet Format	包格式模型	二进制文件
.pr.c	Process Model C	进程 C 代码	C 代码
.pr.cpp	Process Model C++	进程 C++代码	C++代码
.pr.m	Process Model	进程模型	二进制文件
.pr.o	Process Model object	进程模型目标文件	目标代码
.prj	Project Model	项目文件	二进制文件
.ps.c	Pipeline Stage Model C	管道阶段 C 文件	C 代码
.ps.cpp	Pipeline Stage Model C++	管道阶段 C++文件	C++代码
.ps.o	Pipeline Stage Model object	管道阶段目标文件	目标代码
.scfa	Satellite Configuration	卫星配置文件	二进制文件
.pbs.m	Service Level Agreement Prob e Model	服务等级保证探针模型（用于 ESP 附加模块）	二进制文件
.sd	Simulator Description	仿真描述	文本文件
.seq	Simulation Sequence	仿真序列	二进制文件

续表

后 缀 名	扩展描述	意 义	文件格式
.sim	Simulation Executable program	可执行的仿真	可执行文件
.trja	Mobile Node Trajectory	移动节点轨迹	二进制文件

2.3 OPNET 文件管理

当我们创建了一个新的文件目录后，其中包含的模型文件要想在 OPNET 下能够打开需要将文件目录加入到 OPNET 源路径中，如图 2-2 所示，在 File 菜单下选择 Models Files -> Add Model Directory，找到相关目录，如果想让它成为工作目录（所有新创建的文件将保存在该目录中），那么要选中 Make this the default directory。另外所选目录包含很多子目录时，选择 Include all subdirectories 能够将所有目录添加到 OPNET 源路径中，并且在 Edit->Preference->mod_dirs 中，最高层目录将浮动到所有添加目录的最高行，但并不一定是工作目录。之后在 OPNET 就可以打开里面的文件。

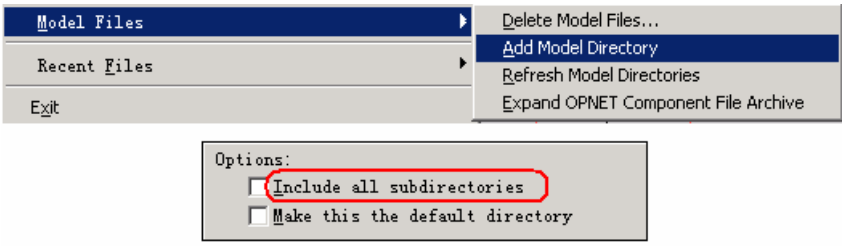


图 2-2 添加 OPNET 源路径

另外在 C:\op_admin 目录下，包含的一些特殊的文件及子目录有些特殊的用途，有时需要对它们进行一些管理：

（1）session_log 文件，记录了有关 OPNET 的所有操作，如对模型的复制，创建，删除等。

（2）err_log 文件，为错误日志文件，它包含了出错情况下完整的函数调用堆栈信息，我们可以从函数阶层性的调用关系中精确找到出错位置。

（3）env_db 文件，为 OPNET 环境文件，所有在 Edit->Preference 下的改动都会保存到这个文件中，同时环境变量的文本化给移植带来很大方便。

（4）op_admin\tmp 目录，保存仿真中间结果，时间一长，该目录包含的文件可能占用大量的硬盘空间，注意定时清空该目录。

（5）op_admin\bk 目录，备份文件，在 Edit->Preference 下可设置 backup_interval 来指定多长时间备份一次，以及通过设置 backup_max_count 指定备份的最大次数，它和 tmp 目录相似，长时间不清空可能占用很多硬盘空间

最后如果碰到 OPNET 使用上的问题，可以从 Help 菜单中选择打开联机帮助文档，它

为*.pdf 格式，需要用 Adobe Acrobat 软件打开，在 Edit->Preference 下可指定浏览器软件的路径，在 vudoc_prog 属性中输入 C:\Program Files\Adobe\Acrobat 5.0\Acrobat\Acrobat.exe。由于 OPNET 共有 412 个帮助文档，首先必须通过全局搜索来定位，如图 2-3 所示，然后在当前页查找相关内容。

图 2-3　OPNET 在线帮助的全局搜索与局部搜索

第 2 部分　OPNET Modeler 使用（基本篇）

第 3 章　OPNET 的通信仿真机制

3.1　离散事件仿真机制

3.1.1　OPNET 中的事件推进机制

OPNET 采用离散事件驱动的模拟机理（Discrete event driven）(其中“事件”是指网络状态的变化)也就是说，只有网络状态发生变化时，模拟机才工作，网络状态不发生变化的时间段不执行任何模拟计算，即被跳过。因此，与时间驱动相比，离散事件驱动的模拟机计算效率得到很大提高。仿真核心实际上为离散事件驱动的事件调度器（Event Scheduler），它对所有进程模块希望完成的事件和计划该事件发生的时间进行列表和维护。

如图 3-1 所示，事件调度器主要维护一个具有优先级的队列，它按照事件发生的时间对其中的工作排序，并遵循先进先出（FIFO，First In First Out）顺序执行事件。而各个模块之间的通信主要依靠传递包的方式来实现。

OPNET 采用的离散事件驱动模拟机理决定了其时间推进机制：仿真核心处理完当前事件 A 后，把它从事件列表(Event List)中删除，并且获得下一事件 B（这时事件 B 变为中断

B，所有的事件都渴望变成中断，但是只有被仿真核心获取的事件才能变成中断，事件有可能在执行之前被进程销毁），如果事件 B 发生的时间 t2 大于当前仿真时间 t1，OPNET 将仿真时间(Simulation time)推进到 t2，并触发中断 B；如果 t1 等于 t2,仿真时间将不推进，直接触发中断 B。

值得注意的是，OPNET 推进是仿真时间，和逝去时间(Elapsed time)有着本质的区别。逝去时间是仿真程序运行的时间，是真实的时间,反映了仿真程序执行的速度，由机器的硬件速度决定。而仿真时间是系统仿真的时间进度，反映当前仿真执行的进度,是一个抽象的时间,它的推进是根据仿真的逻辑来定。

仿真时间的推进随着事件的发生而单调递增。具体来说，在 0 秒时执行一个事件，机器运行 5 秒种（逝去时间），之后仿真核心接着触发下一个事件，随着这个事件的执行，系统的仿真时间推进到 5 秒（仿真时间）。在进程模型中，可以通过调度将来的某个时刻的事件来更新仿真时间，例如当前时刻执行语句 op_intrpt_schedule_self(op_sim_time()+ 仿真推进的时间 T，中断码)后，下一个事件的执行将使仿真时间推进 T 秒。在上例中如果等于 0 秒，则下一事件没有对仿真时间的推进做任何贡献。

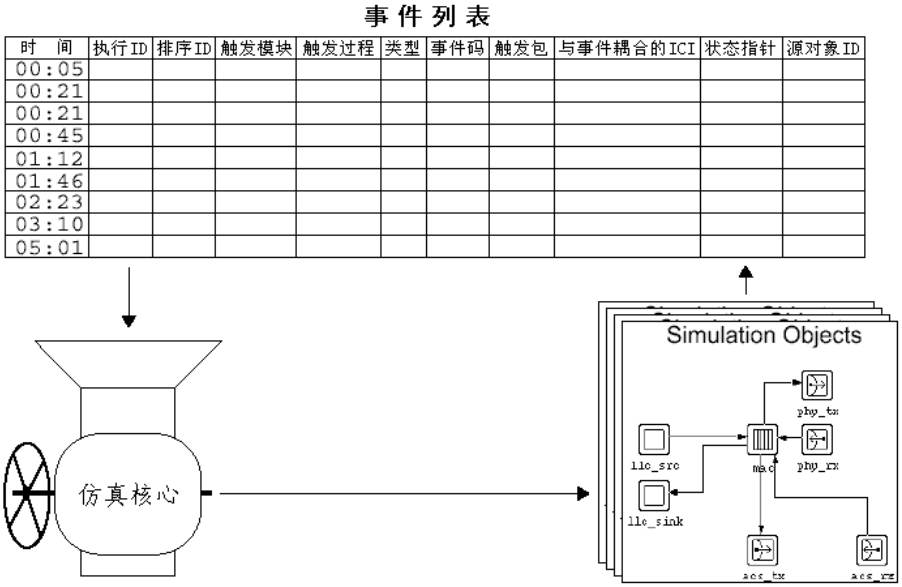


图 3-1 事件调度

有时可能会出现仿真时间始终停留在某个时间点上，这肯定是由于程序的逻辑错误导致的，具体来说，在某个时刻循环触发事件，例如，在某个循环语句中执行了以下程序 op_intrpt_schedule_self (op_sim_time() , 中断码))，这样仿真核心永远处理不完当前时刻的时间，因此仿真总是无法结束。仿真结束条件有两个：

（1）Event List 为空。

（2）仿真时间推进到所设定的时间。

总之，执行事件不需要任何时间，事件和事件之间可能跨越仿真时间，但是不消耗物

理时间，事件执行过程直至事件执行完毕，仿真时间不推进，但需要物理时间，这个物理时间是受机器 CPU 的限制。

3.1.2　同一时刻事件优先级的界定

上节提到执行事件不需要任何时间，假如同一时刻有多个事件存在仿真核心事件列表中，那么它们将按照先进先出 FIFO 的顺序被仿真核心处理，我们很难确定这些事件执行的优先级。当我们在时间上不能区分事件优先级，那只好手动设定事件优先级来区分同一时间内事件执行的顺序，OPNET 提供了三种方法，分别是（1）在进程界面上设置事件优先级；（2）编程指定特定事件优先级；（3）增加冗余的红色状态。

（1）如图 3-2 所示，在进程模型的 Process Interfaces 中设定优先级（priority）属性值，这个值越大代表优先级越高。设定之后所有由该进程产生的事件都采用这个优先级，因此它也可以称为进程优先级。

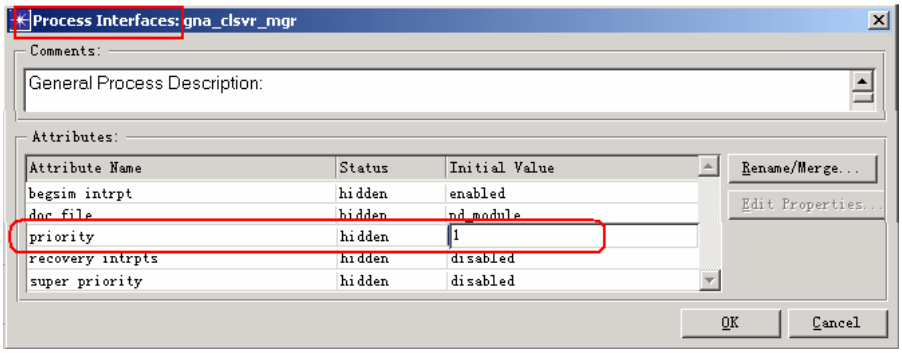


图 3-2　在进程界面上设置优先级

（2）编程实现 op_intrpt_priority_set (事件类型，事件代码，事件优先级);

（3）增加冗余的红色状态，这种方法在初始化是最常用到，也可以称为零时刻多次触发事件。

在上节提到同一进程模型中某时刻多次触发事件有可能导致逻辑错误，但也是一种编程的技巧，一般用在多个协议需要协同初始化的场合，OPNET 中许多标准协议的进程模型都使用了该技巧。

图 3-3 为无线局域网 MAC 层接口进程模型（wlan_mac_interface.pr.m），在“init”状态中的入口执行代码中有语句 op_intrpt_schedule_self(op_sim_time(), 0)，而出口执行代码中也有语句 op_intrpt_schedule_self(op_sim_time(), 0)，两句零时刻调度语句和两个红色的非强制(unforced)状态（init 和 init2 状态）互相抵消，因此 init2 状态看似冗余状态，其实不然，它起着为多个进程模块协同初始化的作用。

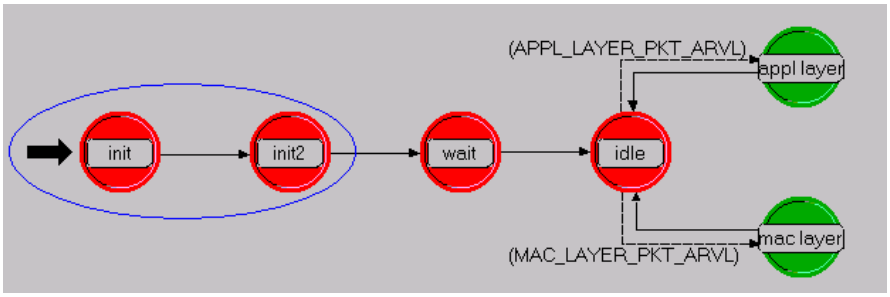


图 3-3　无线局域网 MAC 层接口进程模型

因为在仿真开始时候（零时刻），许多模块需要通过仿真核心触发仿真开始事件（begsim intrpt）来进行初始化，然而有些模块的初始化依赖于其他模块初始化的结果，换句话说，它需要等待其他模块初始化完毕后才能进一步设置参数或做出某种决定。因为都是同一时刻的事件，仿真核心没有能力安排它们合理的顺序，因此通过引入冗余的非强制状态来界定同一时刻事件的发生顺序。

对于 OPNET 编写的标准 OSI 协议栈（application、tpal、ip 等模块），它们是一个整体，互相关联，缺一不可而且一般情况下不允许出现重复的协议模块，为了在仿真开始时验证协议栈的完整性和兼容性，仿真初始化时存在一个协议注册（register）和发现（discover）的过程。为了保证在协议发现之前其他协议模块都已完成注册，这时需要添加一个冗余状态，这样协议发现事件将发生在协议注册事件之后。

3.2　基于包的通信

OPNET 采用基于包的建模机制（Simulation on packet level）来模拟实际物理网络中包的流动，包括在网络设备间的流动和网络设备内部的处理过程；模拟实际网络协议中的组包和拆包的过程，可以生成、编辑任何标准的或自定义的包格式，利用调试功能；还可以在模拟过程中察看任何特定包的包头（Header）和净荷（Payload）等内容。

包这个术语起源于通信领域，也是在 OPNET 建模环境中交流最广泛的信息，它可以使用于不同应用类型场合。包是 OPNET 为支持基于信息源（Message-oriented）通信而定义的数据结构。包被看作是对象，可以动态创建、修改、检查、拷贝、发送、接收和销毁。

1．包结构

每个包含有一些存储信息的区域。包的类型可以是有格式（formatted）或无格式（unformatted）的。一个有格式包中每个域以名字标识，作为访问（设置或者读取）包域的依据，而无格式包只为每个域指定索引号。包域可以存储不同类型的信息，如整型和双精度型用来存储数字数据；包结构类型用来封装另一个包；结构体用来内嵌用户自定义的数据结构。

除了这些包域，所有的包自带一些系统预定义的信息，如优先级、创建时间和地点。

为了支持收发机管道阶段的建模，包还隐含着一些为不同管道阶段交流数据的存储区域，请参见 6.2.15 节传输数据类核心函数和第 9 章无线信道建模。

2．包流

包流是支持包在同一节点模型的不同模块间传输包的物理连接，具体来说，它是源模块的输出端口和目的模块输入端口间的物理连接。包流通常分为源模块的输出流（Output stream)和目的模块的输入流（Input stream）。

虽然连接到模块的包流（输入流和输出流）的个数没有限制，但是 OPNET 不允许群收（Fan-in）和群发（Fan-out）模式，具体来说，每个输入流只能是一个包的唯一接收者，相对应地，每个输出流只能是每个包的唯一发送者。

在节点模型中的包流上单击鼠标右键，可以看到源模块输出流和目的模块输入流的流索引号（stream index），分别为 src stream [m]和 dest stream [n]中的 m 和 n。如图 3-4 所示。

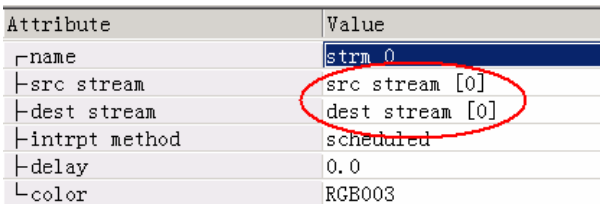


图 3-4　包流属性

3．输入流中的排队

OPNET 为目的模块设置了一个包队列，允许包在没有被移除之前在队列中积压。值得注意的是，包队列是隶属于模块，而不隶属于某个包流，因此连接模块的包流可以有多个，而包队列只有一个。仿真核心不限制该队列的大小。队列采用先进先出（FIFO）模式管理包，位于队首的包才能被目的模块通过 op_pk_get(stream index)获取并移除。如图 3-5 所示。

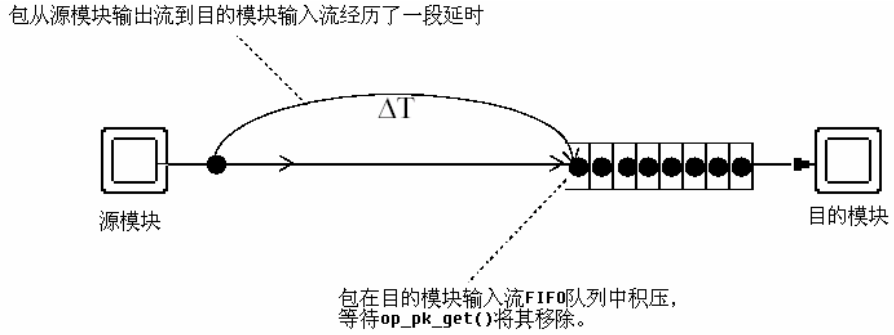


图 3-5　包流中的排队

4．中断模式

如果包流的源模块是进程模块，则可以通过 op_pk_send()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。

（4）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 3-6 所示。

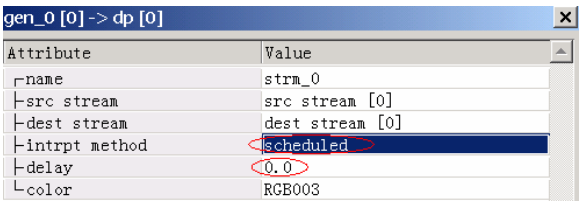


图 3-6　包流属性对话框

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。

5．包传递（Packet delivery）

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 3-7 所示。

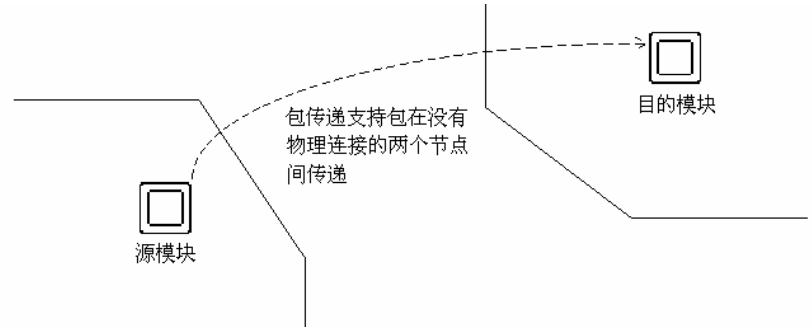


图 3-7　包在模块间的传递

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。

3.3　使用接口控制信息进行通信

基于接口控制信息（ICI，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。

1．ICI 的应用场合

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，因此比包的应用范围更广，如同一节点模型的不同模块之间、不同节点模型之间以及同一节点模型的相同模块内。

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 捎带这个包相应的服务等级和目的地址。

2．ICI 的内容

在使用 op_ici_create()创建 ICI 之前必须先编辑其格式，它定义了 ICI 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 3-8 所示。

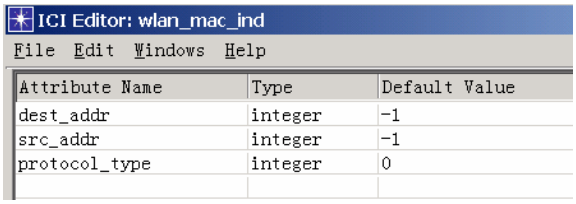


图 3-8　ICI 编辑器中的属性域

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。属性的数据类型支持整型、双精度性和结构体。结构体可以封装用户自定义的数据，一般需要通过 op_prg_mem_aloc()为它分配动态内存，如以下代码所示：

```
/* 分配内存给自定义结构体"command"，它将被封装在ICI的一个属性域中 */
command_ptr = (Command*) op_prg_mem_alloc (sizeof (Command));
/* 给"command"结构体赋值*/
command_ptr->type = COMMAND_TYPE_A;
command_ptr->priority = COMMAND_PRIORITY_LOW;
command_ptr->request = SET_COMPRESSED_MODE;
/* 创建一个新的ICI*/
ici_ptr = op_ici_create ("link_control");
/* 将“command”结构体写入ICI的属性域中*/
op_ici_attr_set (ici_ptr, "command", command_ptr);
```

当包被销毁时，所有包域数据所占内存会被自动清空，ICI 这点与之不同，ICI 被销毁时，其封装的数据所占内存需要手动清空。

3．ICI 通信原理

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（有关进程和进程模块的区别请参见 7.3 节）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。

4．基于 ICI 的操作

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（请参见 7.2.4 节），还有其他一些有关 ICI 的函数，现归纳如表 3-1 所示。

表 3-1　与 ICI 相关的核心函数表

与 ICI 相关的核心函数	函数功能简介
op_ici_attr_exists()	判断某个名字所对应的属性栏是否存在 ICI 格式中
op_ici_attr_get()	指定属性名，获取对应的数据值
op_ici_attr_set()	指定属性名，设置对应的数据值
op_ici_create()	创建一个指定格式的 ICI，返回 ICI 指针
op_ici_destroy()	释放 ICI 占用的内存
op_ici_format()	获取 ICI 的格式，即 ICI 格式文件名称
op_ici_install()	为进程绑定一个 ICI
op_ici_print()	将 ICI 包含的内容打印在 ODB 窗口中
op_intrpt_ici()	获取与事件关联的 ICI 指针
op_pk_ici_get()	获取与包关联的 ICI 指针之前必须通过 op_pk_ici_set()将该 ICI 与包绑定

3.4　点对点和总线管道阶段

OPNET 支持 3 种链路形式，分别是点对点链路、总线链路和无线链路，为了描述它们

的物理特性上的各个特点，分别采用一系列管道阶段去模拟。这一节我们只介绍点对点和总线链路，无线链路管道阶段请参考 9.3 节。

对于有线链路，它们都需要设定所支持的封包格式，并且要和收发信机支持的封包格式一致。建好拓扑后通常需要验证链路间的连接性，如果支持封包不匹配将导致连接失败。在有线链路编辑器对话框中选定链路的类型将决定有线链路是点对点单工链路（ptsimp）、点对点双工链路（ptdup）、总线链路（bus 或 bus tap），如图 3-9 所示。

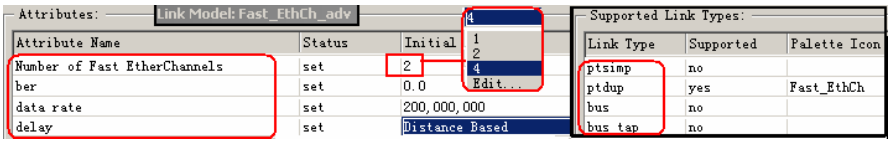


图 3-9　有线链路对话框中需要设定的参数

如图 3-10 所示，点对点链路经历 4 个管道阶段计算：

（1）传输延时阶段计算传播封包所用的处理延时，根据链路处理速度（data rate）来定，如图 3-9 所示。

（2）传播延时，可以指定固定延时，也可以根据链路的长度来定，如图 3-90 所示。

（3）错误分配，需要根据 ber 属性设定的错误分配概率来决定封包有多少位出错，如图 3-9 所示。

（4）错误纠正纠错，根据错误分配阶段计算的比特错误位数和 ecc 纠错极限来决定是否丢包，例如封包的大小为 1000bit，其中有 1 位出错，但是纠错极限为 0，则需要丢包。值得注意的是，该阶段并没有真正的纠错能力，只是做一个简单的判断而已。

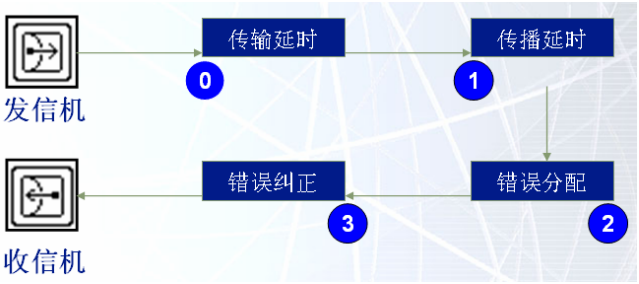


图 3-10　点对点链路管道阶段

除了以上设定链路参数为管道阶段提供计算支持外，我们还需要指定管道阶段模型的名称，默认为 txdel model（传播延时模型，默认为 dpt_txdel），prodel model（传播延时模型，默认为 dpt_prodel，另外针对的特殊链路，dpt_prodel_bgutil 管道阶段还可设置背景利用率），error model（错误分配模型，默认为 dpt_error），ecc model（错误纠正模型，默认为 dpt_ecc）。封包从发信机出来后，经过这些阶段的计算，最后将结果写入包的 TDA 属性中。

注意管道阶段文件名和函数名必须完全相同，否则编译时会出现无可解决的外部函数错误。另外为了对特殊链路的支持，还可以设定多个信道，如图 3-9 所示的 Fast_EthCh_adv

链路模型的 Number of Fast Ethen Channel 属性可以设定多个信道，我们打开以太网工作站模块，如图 3-11 所示，可以看到 MAC 层与收发信机间有 4 个信道连线，它为链路的多信道特定提供支持，否则多信道的功能不能够体现。

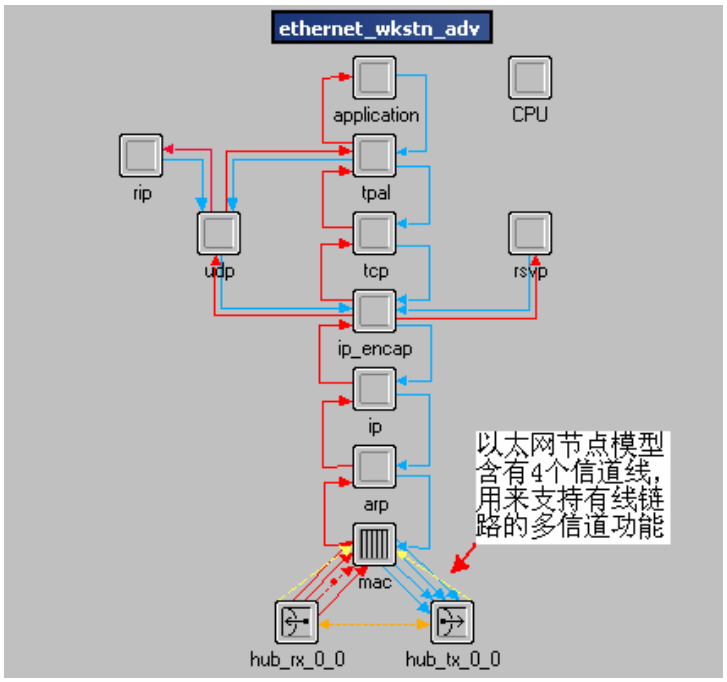


图 3-11　以太网工作站节点模块中设定的多个信道

如图 3-12 所示，总线共有有 6 个管道阶段模块，与点对点链路相比，总线最大的特点是可供多个收信机同时接收信号，而发信机端的传输时延计算一次。

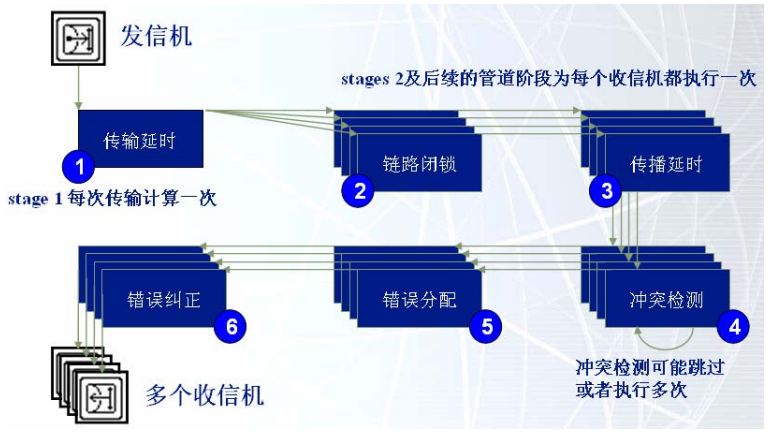


图 3-12　模拟总线链路物理特性的 6 个管道阶段

如图 3-13 所示，A 向链路发送数据，对于每个包在经历传输延时之后，将判断链路闭锁，看有没有潜在的目的地，如有多个闭锁则计算多次后续的管道阶段，如图 3-13 所示，

A 发送的每个封包将复制 4 份，分别发网节点由 B、C、D、E。之后对于每个目的地，将单独计算传播延时，因为每个目的地离 A 地距离可能不同，并且由于总线共享链路的特性，有可能存在冲突，必须进行冲突检测，之后的错误分配和错误纠正阶段与点对点链路管道阶段计算类似。

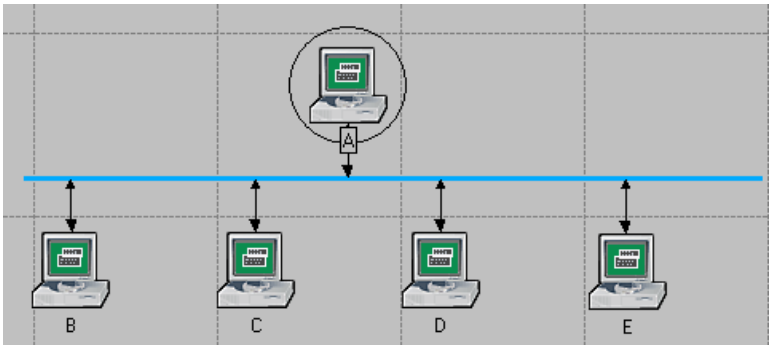


图 3-13　节点 A 通过总线链路向 BCDE 发送数据

第 5 章　收集、查看、导出以及发布仿真结果

5.1　收集统计量

OPNET 提供两种类型的统计量：矢量统计量（Vector）和标量统计量（Scalar），对应的输出文件为矢量文件（*.ov）和标量文件（*.os）。收集统计量就是将统计量写入统计量输出文件中。

矢量文件包含动态的，基于事件的十进制数据，这些数据为统计量随时间变化的情况，每个数据点都是在某个时刻仿真核心自动将矢量统计量写入矢量文件生成的。一个矢量文件只能包含一次仿真的数据，换句话说，仿真过程中不能将新的数据加入以前仿真创建的矢量输出文件中。每个矢量统计量在隶属的探针模型（Probe Model）文件中都对应一个探针，而标量统计量不需要在探针编辑器中定义探针。

标量输出文件可以收集由许多仿真共同产生的结果，具体来说，对于一系列仿真，每次仿真更新一次参数得出一个新的结果，我们希望将每次仿真的参数与其对应的结果画成一条曲线，这时就可以采用将结果写入标量输出文件的方法。与矢量文件相比，标量文件

只包含静态的数据。标量文件以数据块的方式组织数据，每一次仿真的所有标量数据被写入一个相应的数据块中。

5.1.1　收集矢量统计量

基于结果收集的范围，矢量统计量可以分成本地统计量(local statistics)和全局的统计量(global statistics)两种，本地统计量只针对某个模块，其结果只反映单个模块的行为。全局统计量针对整个网络模型，关注整个网络的行为和性能，例如，对网络包的端对端延时性能的测试，它并不关心某个包的源和目的地，只关心所有包的延时性能的统计结果。如果一个节点模型发送一系列数据包，希望统计发送包的个数，这时可以编程将包数分别写入一个本地统计量和全局统计量中，假如在项目中用到了两个这样的节点，那么本地统计量是查看每一个节点发送的数据包数，而全局统计量则是这两个节点共同发送的数据包数。

OPNET 提供多种矢量统计量收集模式（Capture Mode）的选择，这将给用户提供多角度观察网络性能的支持。收集模式有四种，分别是：

（1）all value：收集所有的值。

（2）sample：采样收集，每间隔多少秒钟收集一次。

（3）bucket：桶状收集，在一定范围内将结果迭加后平均。

（4）glitch removal：去除毛刺，两个结果在同一时刻发生，往往只要最后一个值，或舍去值过大过小的点，使结果平滑，在无线上比较常用。

OPNET 中选择统计结果的高级方法：

（1）建立模拟场景后，Simulation→Choose Statistics（Advanced）。

（2）这里出现 8 大类可以统计的结果。

（3）Global Statistic Probes 的使用。

单击鼠标右键，弹出属性对话框后，选择 Statistic，然后配置需要的统计量。

（4）Node Statistic Probes 的使用。

右键弹出属性框后，先后输入 subnet、node 属性值，然后选择 Statistic 配置需要的统计量。

5.1.2　收集标量统计量

标量文件的收集是由用户手动完成的，因为对于一次仿真，一个标量统计量只有一个值，所以一般将某个仿真属性设置为多个取值，然后运行仿真序列（Simulation Sequence）。这时 OPNET 会根据设定值的个数运行相应次数的仿真，每次仿真对应一种参数设置并产生一个结果值。在进程模型中每次仿真结束时将这些单个结果值写入标量文件中，多个仿真就有一系列值。

例如在仿真的一个参数取值从 M~N，采样 R 次，仿真完成后将生成 R 个输出结果，

最终写入到一个标量文件中。标量统计量一个重要的用处是查看两组结果之间的关系，通过分析配置工具（analysis configuration）同时加载两个标量统计量就能产生一个结果随着另一个结果变化的曲线。

5.2 查看和导出仿真结果

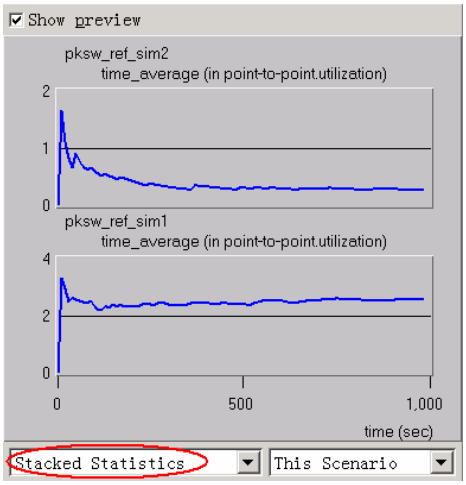
仿真结束后，就可以查看 OPNET 统计结果，关于统计结果的显示 OPNET 提供多种可选择的方案，分别为：

（1）视觉效果选择有三种：individual statistic（一幅图只显示一个结果）；stacked statistics（一幅图包含多个结果子图，如图 5-1 所示）；overlaid statistics（一幅图重叠显示多个结果，如图 5-2 所示）。

（2）图形面板的选择，包括选择横轴纵轴是否显示，线条大小以及为虚线还是实线。

（3）可以设置结果显示的风格，可以选择线性图、离散图或是柱状图。

（4）提供多种结果显示模式，常用的有 As is：不做任何处理；Average：对曲线取值做平均；Time_average：对曲线取值做时间平均。



5-1 结果分开显示

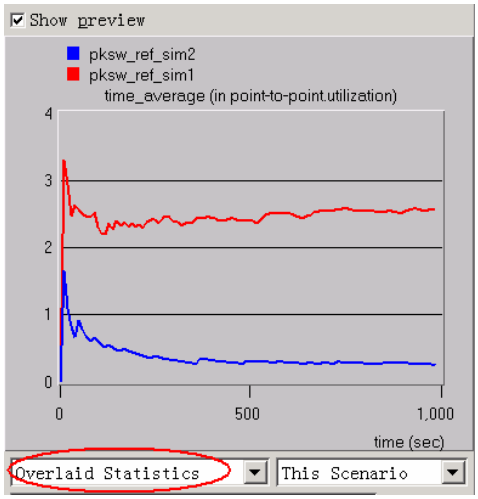


图 5-2 结果重叠显示

查看结果之后，一般需要将制作仿真报告，将结果图拷贝到文档中，这里提供几种方法，列举如下。

（1）Ctrl + Print Screen 抓屏。

（2）选中所需的图，Ctrl+t，然后会弹出一个保存文件对话框，将要保存的图命名后就可以在相应目录中用画图板打开了。但是这种方法结果图的标题栏的风格会被修改。

（3）OPNET 的结果显示效果局限于颜色和显示风格的调整，由于没有提供特殊图标支持，因此打印在黑白的纸张中很难比较结果的差别。这时前面两种方法不能满足要求，

我们希望将结果图导出，转成原始数据。可以采用以下方法：在要导出的图上按鼠标右键，从弹出的菜单中选择 Export Graph Data to Spreadsheet，然后会有提示说文件保存在什么地方，一般默认是保存在 c:\op_admin\tmp 目录下。可以查找最新的文件找到它，并用剪贴板或 UltraEdit 等工具打开来看，是两列或两列以上数据，第一列是仿真时间，其他列是仿真数据，然后就可以用喜欢的软件画。

除了导出仿真结果，有时需要导出网络拓扑图和节点进程模型结构图，可以从项目编辑器的 Topology→Export Topology→…导出 Project 的几种图形，有 bitmap，html 等格式。节点和进程模型可以从 File 中的 Export Bitmap 导出拓扑图。

5.3　发布仿真结果

当我们设计完拓扑，建好模块，并成功运行仿真后，如何将设计的拓扑结果、仿真结果以及模块向外界发布呢？OPNET 提供一些特殊的功能产生拓扑或结果报告，将模块打包使其方便地在网上传送。

（1）在 Scenarios 菜单下选择 Generate Scenario Web Report…产生拓扑信息报告，如图 5-3 所示，它由一系列链接好的 HTML 文件组成，当我们在 IE 浏览器上点击网络物件时，就可以进入其内部查看其细节，就好比使用工程编辑器一样。

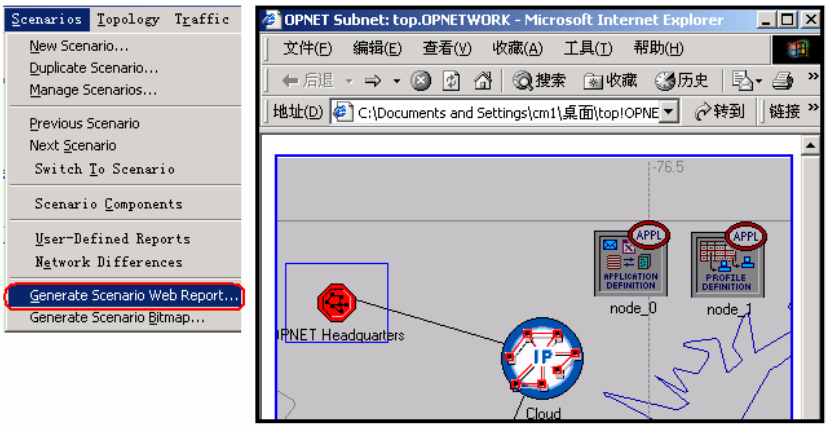


图 5-3　生成拓扑报告

（2）在仿真属性中选择 Generate web report for simulation results，如图 5-4 所示，仿真完毕将自动生成结果报告（web report），里面将所有的结果统计量进行分类显示，它被保存在 C:\op_admin\ ace_web_reports 目录下.

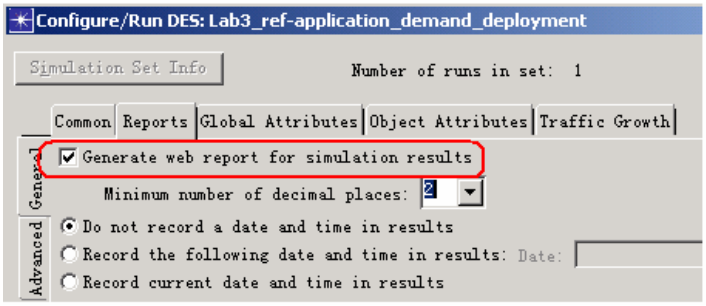


图 5-4 生成结果报告

（3）有时候由于模块包含的文件数量过多并且文件总大小较大，非常不适合电子邮件传输，这时我们可将它们打包，在 Files 菜单下点击 model files -> package project components 选择需要传输的文件打成一个包，对方接收到后，在 Files 菜单下点击 expand opnet component file archive -> select opnet package 把包解开。

第 6 章 OPNET Modeler 编程基础

6.1 从例程开始——创建一个包交换网络

6.1.1 概述

该例程将仿真一个简单的包交换网络。它包括四个周边节点和 1 个中心节点，周边节点产生业务，而中心节点将这些业务转交给相应的目的节点（四个周边节点中的 1 个）。网络拓扑结构如图 6-1 所示。

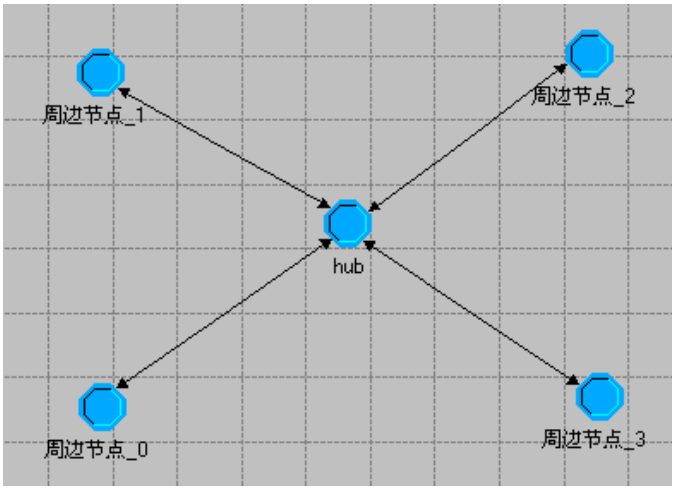


图 6-1　简单的包交换网络拓扑结构

构建该网络模型的同时，我们还将接触到一些新的核心函数，学会如何使用包和链路编辑器，以及如何自定义统计结果。

最后通过观察网络的包交换行为，我们将更加熟悉节点和进程模型及其如何在网络模型中运作。实验完毕，将得到业务的端对端延时结果，进而评估网络的性能。

6.1.2　开始建立

在开始构建该网络之前，我们先熟悉它的物理通信机制和各个节点的功能：。

网络的物理通信机制——如图 6-2 所示，每个节点至少包含一对点对点收发机，并且通过一条有线双工链路和另一对点对点收发机构成一个收发机组。每个这样的收发机组可以支持数据的双向传输。在中心交换节点中，配置了四对点对点收发机，从而在物理上能够支持与四个周边节点互连互通。

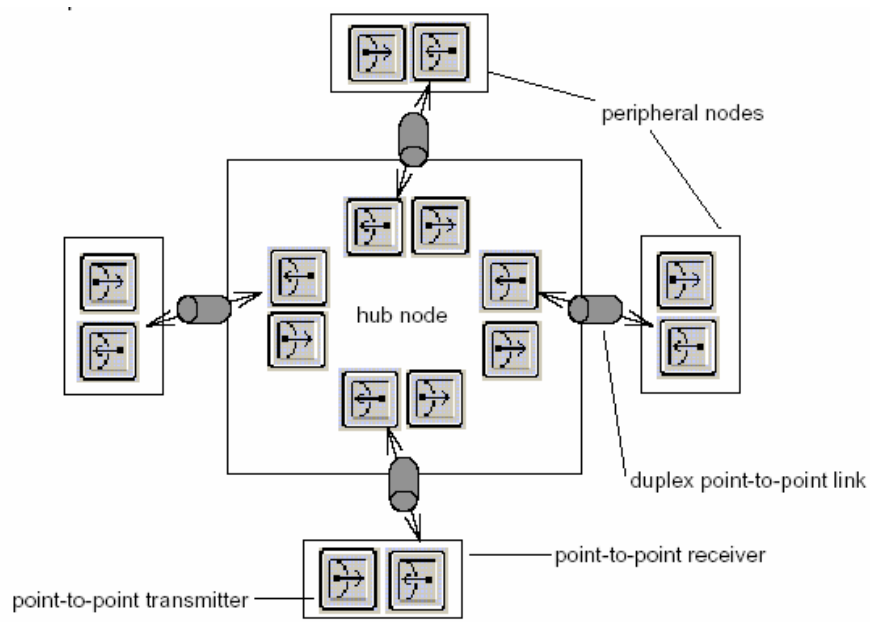


图 6-2　网络的物理通信拓扑结构

网络功能概述——拓扑结构包含两种类型节点模型，它们分别是周边节点和中心交换节点。本例程的目的是仿真一个周边节点发出的业务能够通过中心交换节点路由至另一个目的周边节点。从中心交换节点中看，我们假设，包是以随机的方式来自四个周边节点，每个包包含目的地址，目的地址可以用一个整数来表示不同的目的周边节点，中心节点接收到包后通过对目的地址的解析最后选择一个合适的发信机将包送往目的地。

中心交换节点如何实现寻址和包交换——每个有向包流（以某个进程模型为参考，某个包流或者进入该进程或者离开该进程，因此称之为有向包流）有一个惟一的索引号。

这个索引号总是和某个收信机（对应进入包流）或者某个发信机（对应离开包流）惟一对应，而收信机和发信机又和某个周边节点惟一对应，因此可以直接用流索引号作为交换包的依据。当然为了增强网络的稳健性，我们也可以建立一个目的地址和流索引（可以看作是物理地址）的映射表。为了简单起见，采用前一种方法实现寻址和包交换，如图 6-3 所示。

周边节点的功能——作为网络的业务源，周边节点产生包（用标准的业务生成模块实现），然后为每个包分配一个目的地址并且通过点对点发信机传输出去（自定义模块实现）。同时作为网络的业务终端，周边节点接收包并且统计其端对端延时（在同上的自定义模块中实现），如图 6-4 所示。

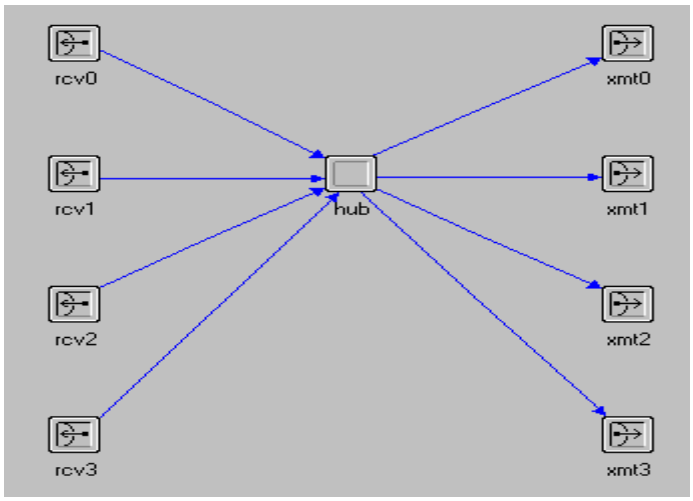


图 6-3　中心交换节点结构

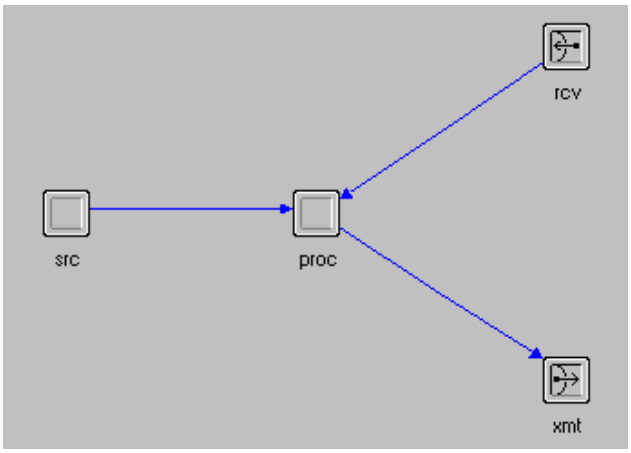


图 6-4　周边节点结构

关键概念 OPNET Modeler 的精髓之一为层次化建模的思想，在构建本网络中，采用如下的层次化的步骤:

定义包格式→定义链路模型→创建中心交换节点模型→创建周边节点模型→建立网络模型

关键概念 OPNET 的包格式编辑器可以创建包含任意数量子域的数据包，包的大小由两部分组成:（1）所有子域大小的叠加。（2）校验值（bulk size）。

6.1.3　创建新的包格式

要创建一个新的包模型：

（1）从 File 菜单中选择 New…，然后从列表中选择 Packet Format，单击 OK 按钮。这时打开包格式编辑器。

（2）单击 Create New Field 工具按钮  ，然后将光标移到编辑窗口中，单击鼠标左键，接着单击右键。

这时一个新的包域出现在编辑窗口中。

现在我们来设置包域的属性：

（3）在包域上单击鼠标右键，从弹出的菜单中选择 Edit Attribute。

（4）从弹出的属性设置对话框中，按图 6-5 设置属性值，然后单击 OK 按钮。

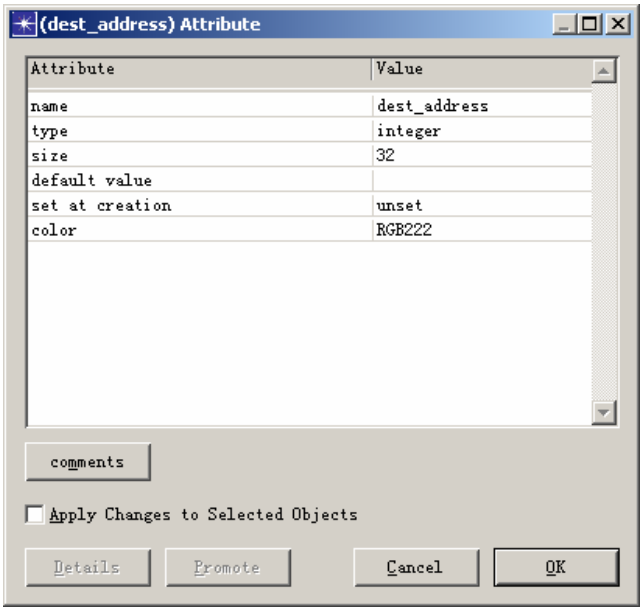


图 6-5　包域的属性

这时定义好的包域名称和大小会在编辑窗口中显示，如图 6-6 所示。

图 6-6　定义好的包域

（5）从 File 菜单中选择 Save…，将包格式命名为<initials>_pksw_format。<initials>为文件名的前缀，可以任意取一个合适的前缀名。

（6）关闭包格式编辑器。

6.1.4　创建新的链路模型

关键概念　使用链路模型编辑器创建自定义链路。

要创建连接中心和周边节点的双工链路模型，并且能支持已定义的包：

（1）从 File 菜单中选择 New…，然后从列表中选择 Link Model，单击 OK 按钮。这时打开链路模型编辑器，如图 6-7 所示。

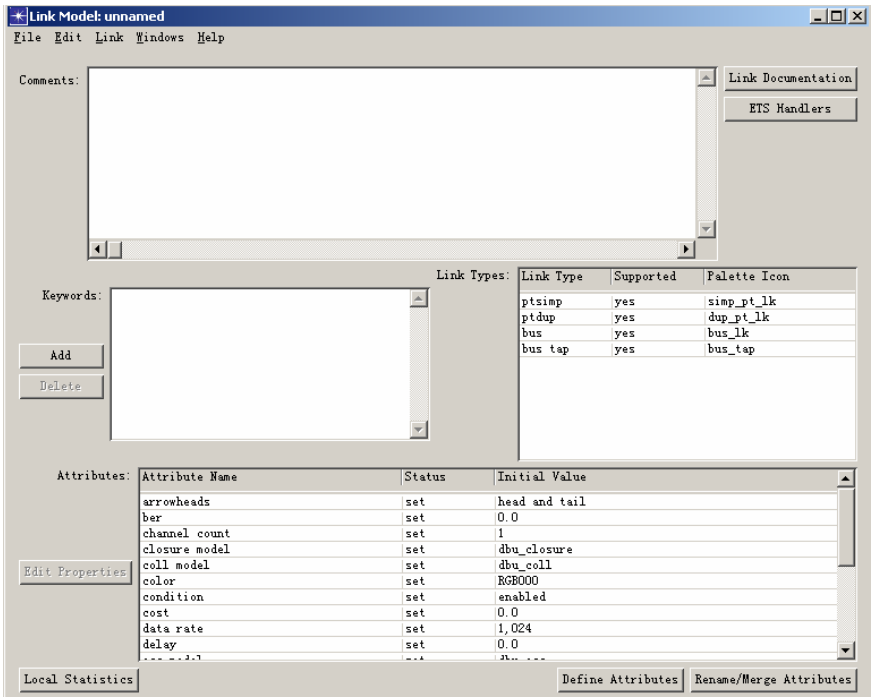


图 6-7　链路模型编辑器

接下来设置支持的包格式：

（2）找到链路类型支持属性框，如图 6-8 所示除了 ptdup 外的链路类型对应的 Supported 属性设置为 no，表明该链路只支持点对点双工连接。

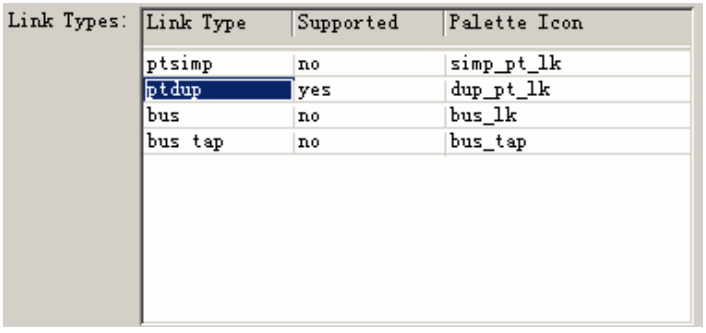


图 6-8　链路类型支持属性框

（3）在 packet formats 属性右边对应的 Initial Value 栏中单击鼠标左键。这时弹出选择包格式支持对话框。

（4）单击“Supports All Packet Formats”和“Supports Unformatted Packets”复选框，

关掉所有默认支持的格式，然后找到<initials>_pksw_format 包格式，将它属性改为 supported。

（5）单击 OK 按钮关闭此对话框。

接下来需要定义链路模型的其他属性：

（6）设置 data rate 属性值为 9600。

（7）设置 ecc model（错误纠错模式）属性值为 ecc_zero_err（取消链路的纠错功能）。

（8）设置 error model（链路干扰模式）属性值为 error_zero_err（链路无干扰）。

（9）设置 prodel model（传播延时计算模式）属性值为 dpt_prodel（计算点对点传播延时）。

（10）设置 txdel model（传输延时计算模式）属性值为 dpt_txdel（计算点对点传输延时）。

如果需要，还可以增加对该链路模型的描述。

设置完属性后，我们还需要增加 link_delay 外部函数。

注意：这一步只针对 OPNET 9.0 及其更高的版本，如果漏掉这一步编译 dpt_prodel 时会因为找不到 link_delay 函数而出现 unresolved external error 错误。在 OPNET 8.0 系列版本中由于默认 dpt_prodel 管道程序没有使用 link_delay 函数，因此不存在这个问题。

（11）从 File 菜单中选择 Declare External Files…。

这时出现申明外部函数文件对话框。

（12）找到 link_delay 并单击其左边的复选框，这时出现绿色的勾。

（13）单击 OK 按钮关闭对话框。

最后命名链路模型：

（14）File 菜单中选择 Save…，将链路模型命名为<initials>_pksw_link，然后单击 Save。

（15）关闭链路模型编辑器。

6.1.5 创建中心交换节点模型

关键概念 创建节点模型需要定义节点模型和定义进程模型两个步骤。

我们从定义节点模型开始，中心交换节点包含：四对发信机和收信机（每对收发信机对应一个周边节点），一个中心交换处理进程（用来按地址转交包）。

要创建节点模型：

（1）从 File 菜单中选择 New…，然后从列表中选择 Node Model，单击 OK 按钮。这时打开节点模型编辑器。

（2）在编辑窗口中放置一个进程模块  ，四个点对点发信机  ，和四个点对点收信机  。

（3）如图 6-9 所示给每个对象命名，并用包流将每个收信机和发信机和 hub 相连。

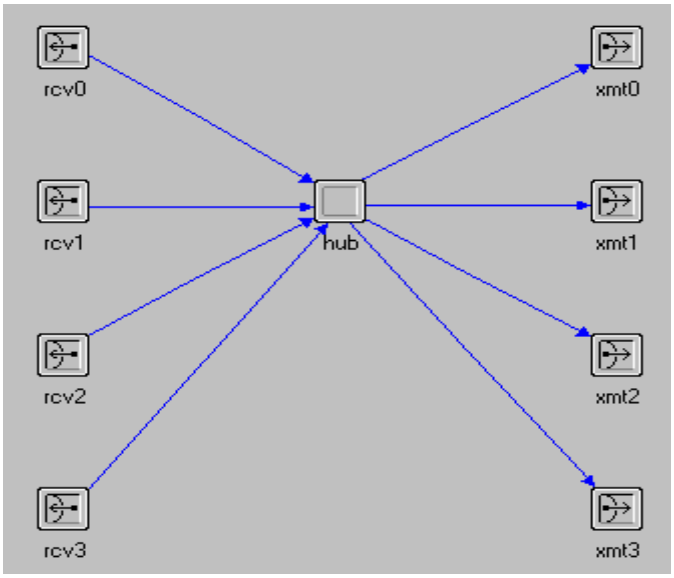


图 6-9　中心交换节点模型

接下来查看包流的连接情况：

（4）在 hub 进程模块上单击右键，从弹出的菜单中选择 Show Connectivity。

这时出现一个包流指向列表，描述包流与 hub 连接情况（格式为：hub[输出流索引号]→发信机；收信机→hub[输入流索引号]），如图 6-10 所示。

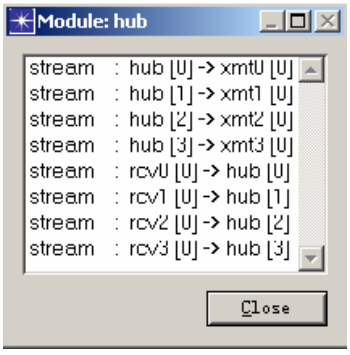


图 6-10　包流的连接关系

（5）关闭该对话框。

接下来你需要定义收发机模型属性：

（6）按住 shift 键，依次以鼠标左键单击所有的收信机和发信机。

注意不要选中包流。

（7）在其中一个收信机或收发信机模块上单击鼠标右键，从弹出的菜单中选择 Edit Attributes。

（8）单击 channel 属性右边的 value 栏，在弹出的信道属性表中将 data rate 设置为 9600。

（9）单击 packet formats 栏，在弹出的对话框中单击“Supports All Packet Formats”和“Supports Unformatted Packets”复选框，关掉所有默认支持的格式，然后找到<initials>_pksw_format 包格式，将它属性改为 supported。单击 OK 关闭对话框。

（10）确定数据率和支持的包格式正确设置，如图 6-11 所示，然后单击 OK 关闭对话框。

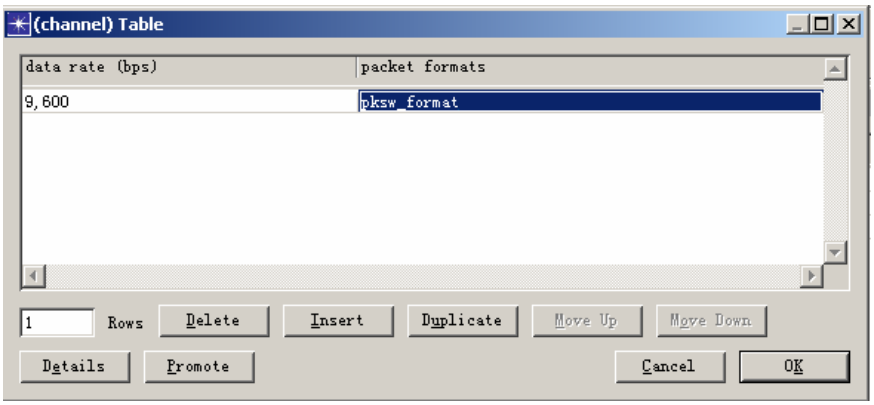


图 6-11　有线收发信机的属性框

（11）需要将以上的设置改变对所有选中的对象起作用，单击 Apply changes to selected objects 复选框，然后单击 OK 按钮。

接下来需要定义节点模型的界面属性：

（12）从 Interfaces 菜单选择 Node Interfaces。

出现节点界面对话框。

（13）找到节点类型支持属性表框，如图 6-12 所示除了 fixed 外的节点类型对应的 Supported 属性设置为 no，表明该节点只能作为固定节点。

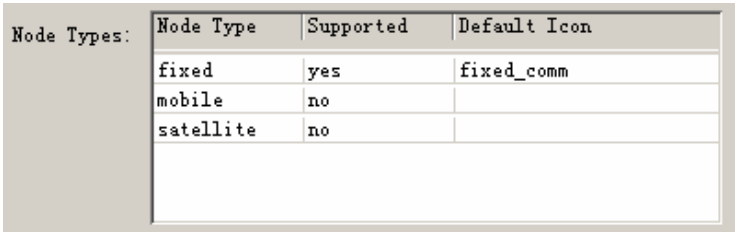


图 6-12 节点类型支持属性表框

如果需要，还可以在 Comments 文本栏中加上对该节点的描述。

现在节点模型就建立完了，将节点模型命名为<initials>_pksw_hub 并保存，但是不要关闭节点模型编辑器，接下来我们来创建 hub 进程模型。

6.1.6　创建 hub 进程模型

关键概念　hub 进程模块将接收到的包按照目的地址转交给正确的发信机，然后通过发信机将包发往目的节点。

在节点模型中，hub 进程模块通过包流与发信机和收信机相连。因为每个包的到达都触发 hub 进程的一次中断，hub 进程接收到中断后将从休眠状态（idle 非强制状态）激活执行代码处理包（绿色的强制状态）。

（1）从 File 菜单中选择 New…，然后从列表中选择 Process Model，单击 OK 按钮。这时打开进程模型编辑器。

（2）单击创建状态按钮，然后将光标移到编辑窗口中，单击鼠标左键，放置一个状态，然后单击鼠标右键，命名该状态为 idle。

关键概念　当包被收信机接收，即给进程触发一个流中断，因此状态必须能够判断出这个条件并做出正确的状态转移。

接着我们来建立状态转移：

（3）单击创建状态转移按钮，单击 idle 状态，创建一个回到该状态自身的转移。

（4）在转移线上单击鼠标右键，从弹出的菜单中选择 Edit Attributes，然后将转移的 condition 属性改为 PK_ARRVL，并且将 executive 属性改为 route_pk()。如图 6-13 所示。

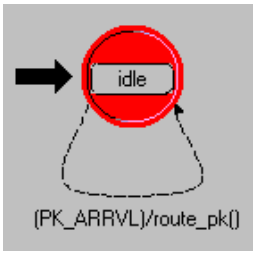


图 6-13　hub 进程模型

（5）单击 OK 关闭转移属性对话框。

接下来你需要定义 PK_ARRVL 条件的宏

（6）单击编辑头块按钮

（7）输入以下定义宏 PK_ARRVL 的代码

```
#define PK_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM)
```

（8）从 File 菜单中选择 Save。

PK_ARRVL 条件判断 hub 进程接收的中断类型是否是流中断（在 OPNET 中以常量 OPC_INTRPT_STRM 表示），如果进程异常地接收到其他类型的中断则状态找不到转移条

件从而导致出错，为以防万一还需要为 idle 状态创建一个指向自身 default（其他条件不满足则该条件满足）的转移线：

（9）为 idle 状态创建一个指向自身的转移线。

（10）在转移线上单击鼠标右键，从弹出的菜单中选择 Edit Attributes，然后将转移的 condition 属性改为 default，右击鼠标关闭对话框，如图 6-14 所示。

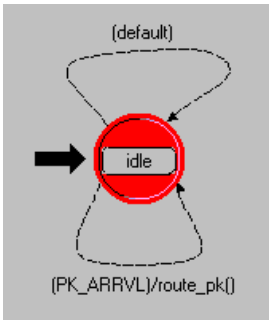


图 6-14　hub 进程模型

接下来你需要编写条件执行代码 route_pk()：

（11）单击编辑函数块按钮 

（12）输入以下代码：

```
static void route_pk(void)
    {
    int dest_address;
    Packet* pkptr;
    FIN(route_pk());
    pkptr = op_pk_get (op_intrpt_strm ());
    op_pk_nfd_get (pkptr, "dest_address", &dest_address);
    op_pk_send (pkptr, dest_address);
    }
```

真正有效的代码是在 FIN(route_pk())之后。第一句用来从合适的输入流（输入流索引通过核心函数 op_intrpt_strm 得到）中取得包（op_pk_get）。第二句代码析取包中的目的域，它含有包的目的地址。前面提过，这里的目的地址实际上是输出流索引，它对应发往目的节点的收信机，而最后一句代码将包发送给相应的收信机。

（13）从 File 菜单中选择 Save。

然后，需要更改进程的属性：

（14）从 Interfaces 菜单中选择 Process Interfaces。

（15）把 begsim intrpt 属性的初识值改为 enabled。

（16）如果需要，在 Comments 文本栏增加模块的说明。

（17）单击 OK 按钮，保存更改。

接下来，你需要编译模块  ：

（18）单击编译进程模型按钮。

（19）从 File 菜单中选择 Close，关闭进程模型编辑器。

最后，需要将编译好的进程模型指定给节点模型：

（20）从 Windows 下拉菜单中选择 Node Editors，然后找到<initials>_pksw_hub。

这时节点模型编辑器被激活。

（21）在 hub 进程上单击鼠标右键，从弹出的菜单中选择 Edit Attributes，将 process model 的属性值改为<initials>_pksw_hub_proc。

（22）单击 OK 按钮关闭属性对话框。

（23）保存节点模型。

6.1.7　创建周边节点模型

关键概念　当周边节点生成一个包时，它必须给这个包指定一个目的地址，然后将它发往中心节点。如果周边节点接收到一个包时，它必须计算该包的端对端延时。因此周边节点必须包括一个业务生成模块、一个进程模块和一对点对点收发信机来完成这些任务。

要创建周边节点模型：

（1）在刚刚保存过 hub 节点模型编辑器中的 Edit 的菜单下选择 Clear Model。

这时编辑器工作空间被清空。

（2）按图 6-15 所示放置并命名模块。

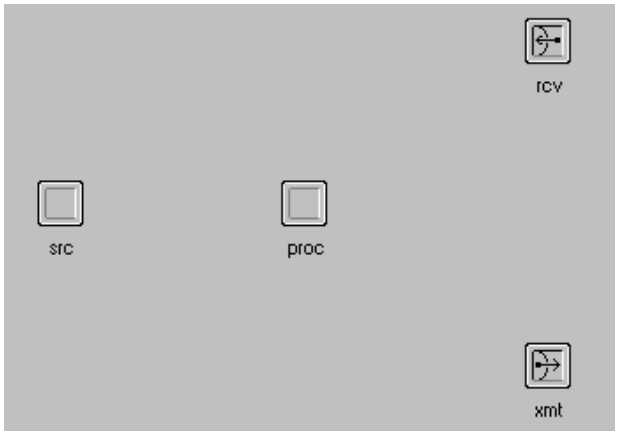


图 6-15　周边节点模型包含的模块

（3）在 src 模块上单击鼠标右键，从弹出的菜单中选择 Edit Attributes，将 process model

属性值改为 simple_source，然后单击 OK 关闭属性对话框。

(4)按下列方向建立包流：rcv→proc；proc→xmt；src→proc。

- 在 proc 进程模块上单击鼠标右键，在弹出的菜单中选择 Show Connectivity，查看包流分配表，如图 6-16 所示。

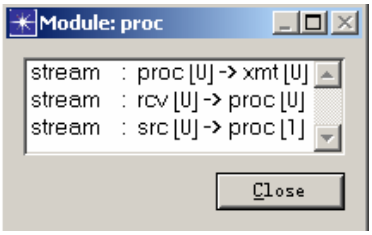


图 6-16　proc 进程模块的包流连接关系

- 如果每个设置都正确，需要删掉所有的包流重新按照步骤（4）设置一遍。

为了运行参数化仿真，需要将业务的 Packet Interarrival Time 属性提升。当提升了属性后，就可以在仿真运行时很容易地改变了。

（5）在 src 模块上单击鼠标右键，从弹出地菜单中选择 Edit Attribute。

（6）在属性表中，选中左边一栏的 Packet Interarrival Time（这时该属性变成蓝色），然后单击 Promote 按钮。

这样就提升了属性，可以在仿真属性中设置它的值。

同时希望业务生成模块能够产生前面定义的包格式：

（7）单击 Packet Format 属性对应的右边 Value 栏，将它更改为<initials>_pksw_format。

（8）参考图 6-17，确定你的设置正确，然后单击 OK 关闭属性对话框。

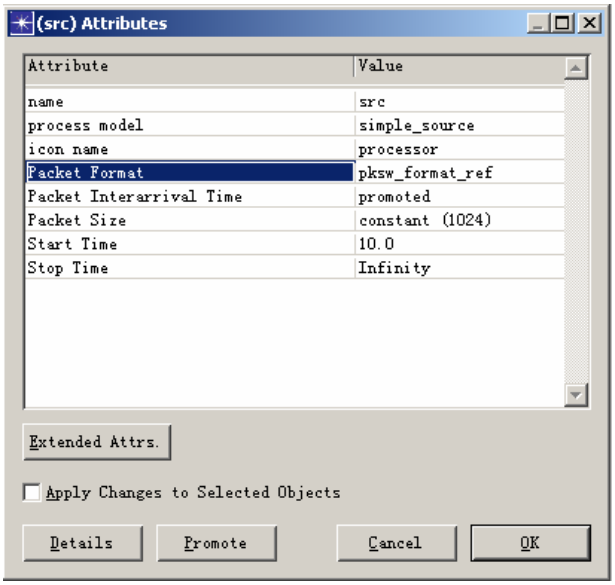


图 6-17　src 进程模块属性

接下来，需要改变收发信机的信道速率和支持的包格式，以匹配指定的链路模型。我们采取和前面类似的操作。

接下来你需要定义收发机模型属性：

（9）住 shift 键，依次以鼠标左键单击收信机和发信机。

注意不要选中包流。

（10）在其中一个收信机或收发信机模块上单击鼠标右键，从弹出的菜单中选择 Edit Attributes。

（11）单击 channel 属性右边的 value 栏，在弹出的信道属性表中将 data rate 设置为 9600.

（12）单击 packet formats 栏，在弹出的对话框中单击“Supports All Packet Formats”和“Supports Unformatted Packets”复选框，关掉所有默认支持的格式，然后找到 <initials>_pksw_format 包格式，将它属性改为 supported。单击 OK 关闭对话框。

（13）确定数据率和支持的包格式正确设置，然后单击 OK 关闭对话框。

（14）需要将以上的设置改变对所有选中的对象起作用，单击 Apply changes to selected objects 复选框，然后单击 OK 按钮。

接下来你需要定义节点模型的界面属性：

（15）Interfaces 菜单中选择 Node Interfaces

出现节点界面对话框。

（16）找到支持的节点类型属性表，除了 fixed 外的节点类型对应的 Supported 属性设置为 no，表明该节点只能作为固定节点。

关键概念　属性重命名可以简化复杂的属性名称，或者扩展过于简化的名称。

当某个属性是由底层提升得来的，它的名称就会变得很冗长而且没有意义，这时可能需要把它的名称简化。本例程将为包到达间隔属性重新命名。

（17）Node Interfaces 对话框中选择 Rename/Merge…按钮。

（18）在 Unmodified Attributes 栏中找到要更名的属性 src.Packet Interarrival Time，然后单击按钮 >> 。

（19）在 Promotion Name 文本栏中中输入新的名字 source interarrival time，如图 6-18 所示。

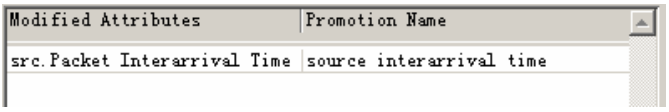


图 6-18　原属性名与重命名后的属性名

（20）单击 OK 关闭重命名对话框。

关键概念　你可以指定一系列预定值给某个属性，这样属性的设置可以通过界面来选择，这将给用户提供方便。

为属性指定预定值有下面几个好处：

- 限制属性取值的范围。
- 用户可以直观地根据预定值的名称来选择相应的参数。
- 用户不需要输入具体值，从下拉列表中选择即可。

接下来为 source interarrival time 属性指定预定值：

（21）在 Node Interfaces 对话框中，选择新命名的 source interarrival time 属性，这时左边的 Edit Properties 按钮被激活，单击它。

这时出现 Attribute:source interarrival time 对话框。

（22）在 Symbol Map 表中，将所有 Symbol 对应的 Status 变为 suppress。

（23）如图 6-19 所示增加 4 个符号与值的映射项。

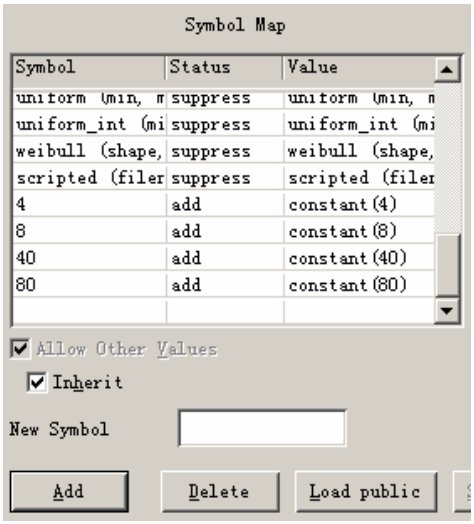


图 6-19　符号与属性真实值的映射表

关键概念　隐藏属性可以避免用户看到不需要设置参数的属性项，从而能够简化用户界面。这个操作不会影响仿真结果。

周边节点的许多属性与仿真无关。为了避免混淆，需要隐藏这些属性：

（24）如图 6-20 所示除了 source interarrival time 外的所有属性的 Status 改为 hidden。

Attribute Name	Status
altitude modeling	hidden
condition	hidden
financial cost	hidden
phase	hidden
priority	hidden
source interarrival time	promoted
user id	hidden

图 6-20　属性状态表

（25）单击 OK 按钮关闭节点界面对话框。

（26）从 File 菜单中选择 Save As…，将节点模型命名为<initials>_pksw_node，然后关闭节点模型编辑器。

接下来，需要创建周边节点的处理模块：

关键概念　周边节点的处理模块主要有两个功能：（1）为包分配目的地址并且发送出去。（2）计算包的端对端延时。

为了完成以上的任务，进程模型需要设置两个状态：一个初试化 initial 状态，一个 idle 状态。

创建进程模型：

（1）从 File 菜单中选择 New…，从弹出的菜单中选择 Process Model，单击 OK 按钮。

（2）如图 6-22 所示在编辑窗口中放置两个状态：

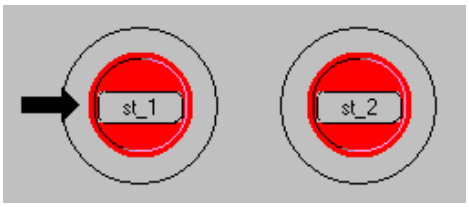


图 6-21　进程模型包含两个状态

（3）改变状态的属性：

- 在第一个状态上单击鼠标右键，在弹出的菜单中选择 Set name 将其改名为 init，并且选择 Make State Unforced 使其变为强制的（forced），这时状态颜色变为绿色。
- 将第二个状态更名为 idle。（保持它为红色的非强制 unforced 状态），如图 6-22 所示。

图 6-22　状态命名后的进程模型

在 init 状态中，进程模型将加载一个从 0~3 的均匀分布概率函数。

下一步，需要为状态创建转移线。

（1）如图 6-23 所指定状态转移以及条件满足所执行的函数。

xmt()转移执行函数产生将调用概率函数随即产生目的地址，并将其分配给来自业务生成模块的包，然后再将它发送出去。

rcv()转移执行函数作用是在接收到包是计算其端对端延时，并且将结果写入全局统计量。

（2）单击编辑头块按钮定义转移条件。

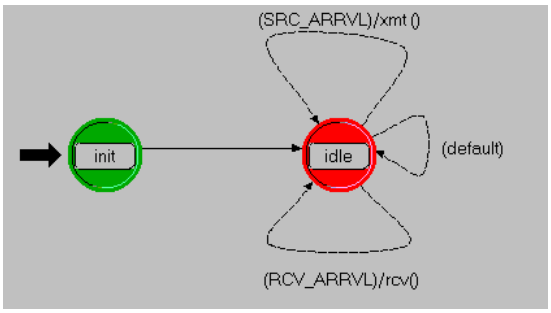


图 6-23　加入状态转移的进程模型

（3）输入以下代码：

```
/* 包流定义 */
#define RCV_IN_STRM 0
#define SRC_IN_STRM 1
#define XMT_OUT_STRM 0
/* 条件宏定义 */
#define SRC_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM && op_intrpt_strm
() == SRC_IN_STRM)
#define RCV_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM && op_intrpt_strm
() == RCV_IN_STRM)
```

RCV_IN_STRM，SRC_IN_STRM 对应数据包的输入流索引号，而 XMT_OUT_STRM 为输出流索引号，输入输出都是相对当前进程模块(proc)而言，它们对应与 proc 模块相连的某条包流，连接关系一旦确定，它们的索引号是常数。之所以要放在头文件中定义这些端口号，是为了修改方便而且避免混淆。

（4）从 File 菜单中选择 Save 保存文件。

接下来，需要定义状态变量和临时变量。

（1）单击编辑状态变量工具按钮  。

（2）在状态变量对话框中输入以下内容，如图 6-24 所示。

Type	Name
Distribution *	address_dist
Stathandle	ete_gsh

图 6-24　设置状态变量

（3）单击 OK 关闭对话框。

下一步，需要创建一个全局统计探针收集包的端对端延时结果。

（1）在进程模型的 Interfaces 菜单中选择 Declare Global Statistics（申明全局统计量）。

（2）将 Stat Name 属性命名为 ETE Delay。

（3）在探针描述文本栏中输入:

Calculates ETE delay by subtracting packet creation time from current simulation time.

（4）从 File 菜单中选择 Save 保存描述文件。

（5）检查的设置是否完成。

参考如图 6-25 所示的对话框。

Stat Name	Mode	C...	Desc.
ETE Delay	Single	N/A	Calculates ETE delay by subtr...

图 6-25　申明全局统计量

（6）单击 OK 关闭 Declare Global Statistics 对话框。

接下来，需要为进程模型中的每个状态添加入口和出口执行代码。首先为 init 状态添加入口执行代码：

（7）双击 init 状态的上半部打开其入口执行代码编辑框，输入以下代码。

```
address_dist = op_dist_load ("uniform_int", 0, 3);
ete_gsh = op_stat_reg("ETE Delay",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL）;
```

（8）从 File 菜单中选择 Save 保存代码。

xmt()转移执行函数当 SRC_ARRVL 条件满足时(即包从业务生成模块到达 proc 模块)才执行。该函数在将包发送之前要为它分配一个目的地址。

（9）在函数块  中，输入以下代码：

```
static void xmt (void)
    {
    Packet* pkptr;
    FIN (xmt());
    pkptr = op_pk_get (SRC_IN_STRM);
    op_pk_nfd_set_int32  (pkptr,  “dest_address”,  (int)op_dist_outcome
    (address_dist));
    op_pk_send (pkptr, XMT_OUT_STRM);
    FOUT;
    }
```

第一行代码从包流的输入流索引号(SRC_IN_STRM)获取数据包。第二行代码通过调用均匀概率分布函数指针(address_dist，它在 init 状态中定义)而产生一个随机值，将该值设置为包的"dest_address"域(请参考前面的包格式定义)。

最后一句从包流的输出流索引号（XMT_OUT_STRM）将包发送出去。

rcv()转移执行函数当 RCV_ARRVL 条件满足（即包从收信机到达 proc 模块）时执行。

主要目的是计算端对端延时并写入全局统计探针。

（10）在函数块中输入以下代码：

```
static void rcv (void)
    {
    Packet* pkptr;
    double ete_delay;
    FIN (rcv());
    pkptr = op_pk_get (RCV_IN_STRM);
    ete_delay = op_sim_time() - op_pk_creation_time_get (pkptr);
    op_stat_write (ete_gsh, ete_delay);
    op_pk_destroy (pkptr);
    FOUT;
    }
```

第 7 行代码获取包指针（如前所述）。第二行代码通过将当前仿真时间减去包的创建时间得到包的端对端延时。第 9 行代码将计算的延时写入矢量结果文件中，第 10 行代码最后销毁包。

（5）从 File 菜单中选择 Save 关闭函数编辑器。

还需要激活“仿真开始”中断：

（1）在 Interfaces 菜单中选择 Process Interfaces，从 Process Interfaces 对话框中，将 begsim intrpt 属性变改为 enabled。

（2）在 comment 文本框中加入进程描述。单击 OK 关闭对话框。

6.1.8 创建网络模型

现在你已经建好了底层的节点、进程和链路模型，依据层次化建模的思想，现在可以构建网络模型了。回想一下，我们开始的网络拓扑结构包括一个中心交换(hub)节点和四个周边节点。

（1）从 OPNET Modeler 主窗口中的 File 菜单中选择 New...，从下拉列表中选择 Project，然后单击 OK。

（2）Project Name 命名为<initials>_pksw_net，将 Scenario 命名为 baseline，单击 OK 按钮。

（3）这时出现网络建立向导，单击 Quit。

将自己指定网络规格，这时需要从一个对象模板中选择。

首先需要创建一个对象模板，它包含你需要用到的模块。

（1）单击打开对象模板工具按钮。

（2）在弹出的对话框中单击配置模板按钮(Configure Palette...)

（3）在 Configure Palette 对话框中，单击 Clear 按钮，然后单击 Node Models 按钮。

（4）找到<initials>_pksw_hub 和<initials>_pksw_node 节点模型并单击右边的 Status 栏使其变为 included。然后单击 OK。

（5）在 Configure Palette 对话框中，单击 Link Models 按钮。

（6）找到<initials>_pksw_link 并包括 include 该链路模型。单击 OK。

（7）在 Configure Palette 对话框中，单击 OK 按钮，将模板命名为<intitials>_pksw_palette，如图 6-26 所示。

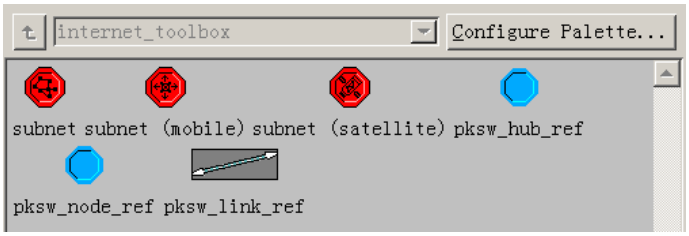


图 6-26　对象模板

现在准备构建网络了。

（1）在项目编辑窗口中放置一个 subnet 模型  并命名为 pksw1。

（2）双击这个子网模块进入它的内部。

（3）放置四个周边节点对象<initials>_pksw_node。

（4）放置一个中心节点对象<initials>_pksw_hub，并将该节点命名为 hub。

（5）单击模板中的链路对象<initials>_pksw_link，

依次（node_0，node_1，node_2，node_3）连接四个周边和 hub 节点，如图 6-27 所示。

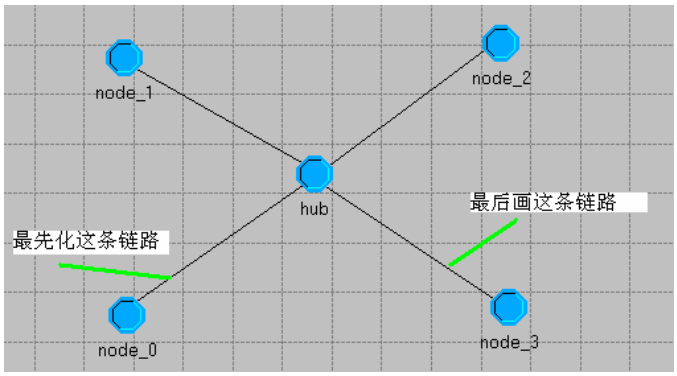


图 6-27　在网络模型中添加链路对象

在保存项目之前，最好验证链路的连接是否正确:

（1）单击验证连接工具按钮  。

（2）选中 Verify links，单击 OK 按钮，如图 6-28 所示。

（3）如果某个链路上出现红色的叉，如图 6-29 所示，则链路不通。

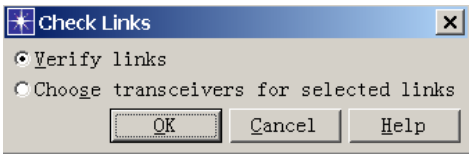


图 6-28　验证连接对话框

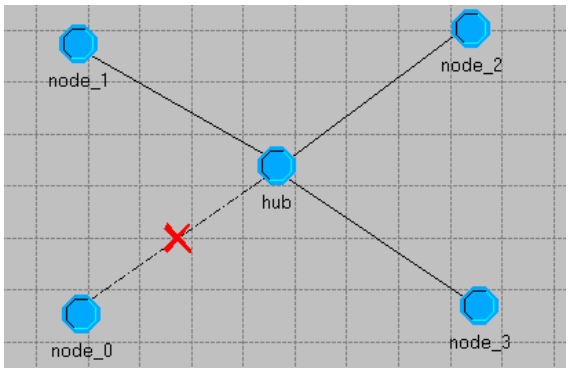


图 6-29　链路不通的情形

在验证连接对话框中单击 Choose transceivers for selected links，这时红色的叉消失，但并不代表问题已经解决，只不过把连通失败链路的连接属性置为空，如图 6-30 所示。

Attribute	Value
name	node_0 <-> hub
model	pksw_link_ref
transmitter a	node_0.NONE
receiver a	node_0.NONE
transmitter b	hub.NONE
receiver b	hub.NONE
data rate	9,600

图 6-30　链路属性

必须重新选择连接的收发信机(必要时调整收发信机属性使它们和链路匹配)直到再次单击 Verify links 验证链路连通性红叉消失。

关键概念　收发信机和链路属性(包格式、数据率等)必须和链路的相应属性匹配才能够使链路连通。

6.1.9　收集统计量并分析结果

已经建好了所有模型，现在可以开始仿真网络行为。

关键概念　对于这个例子，为了观察不同包的产生速率对网络性能的影响，需要在仿真

编辑器中为相应的仿真属性配置多个值，这时一次会运行一系列仿真，每个仿真结果对应属性的一个取值。

选择要收集的结果：

（1）在工程窗口的空白处（任意位置）单击鼠标右键，从弹出的菜单中选择 Choose Individual DES Statistics。

（2）打开 Global Statistics 列表，选中 ETE Delay，单击 OK 关闭对话框，如图 6-31 所示。

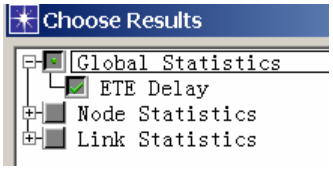


图 6-31　选择全局结果统计量

这是在周边节点中的处理模块定义过的全局统计探针。

（3）在 node_0 与 hub 间的链路上单击鼠标右键，从弹出的菜单中选择 Choose Individual DES Statistics。

（4）打开 point-to-point 列表，选中上行和下行链路利用率，如图 6-32 所示。单击 OK 关闭对话框。

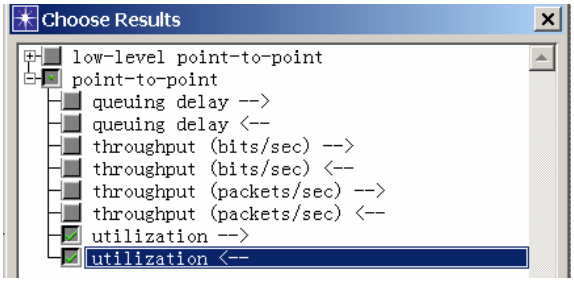


图 6-32　选择链路结果统计量

（5）保存项目文件。

6.1.10　配置仿真

对于这个例子，包的大小和收发机的速率都是恒定的，因此期望端对端延时也应该固定不变。然而，如果包的产生速率足够快，就会导致部分包在发信机队列中积压，这时包的端对端延时加大。如果包的产生速率不定，有可能造成业务突发，因此端对端延时也会受影响。为了模拟这些行为，需要配置 source interarrival time 仿真属性，将给它指定两个值。

（1）从 Simulation 菜单中选择 Configure Simulation(Advanced)

这时仿真编辑器打开。

（2）在仿真设置  上单击鼠标右键，从弹出的菜单中选择 Edit Attributes。

回想前面我们已经将业务生成模块的 interarrival time 属性提升为仿真属性了，现在可以为它指定两个不同的值（每个值运行一次仿真）。

下面将配置当包产生间隔为“4”的仿真:

（1）将仿真设置文件命名为_pksw_sim1。

（2）将随机种子 Seed 设置为 21，仿真时间设为 1000 seconds。

（3）给 source interarrival time 属性赋值：

① 单击 Add 按钮，然后选择未引用的仿真属性，单击 OK 按钮，如图 6-33 所示。

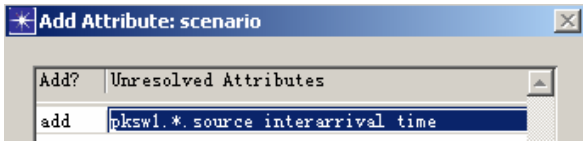


图 6-33　增加未引用的仿真属性

② 在仿真设置对话框中单击 Value 栏，并从下拉列表中选择 4，如图 6-34 所示（下拉列表的效果是因为前面给属性指定了预定值）。

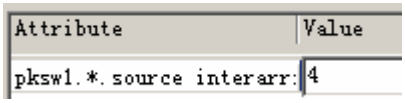


图 6-34　设置仿真属性的取值

（4）将矢量结果文件 Vector file 命名为_pksw_sim1。

（5）单击 OK 关闭仿真设置对话框。

下面将配置另一个仿真:

（6）复制并且粘贴刚刚配置的仿真_pksw_sim1。

新的仿真配置自动命名为_pksw_sim2。

（7）在仿真设置上单击鼠标右键，从弹出的菜单中选择 Edit Attributes，如图 6-35 所示。。

（8）同上，将 source interarrival time 属性赋值为 40。

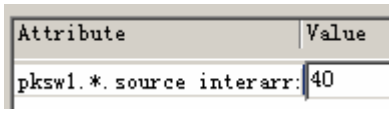


图 6-35　设置仿真属性的取值

（9）将矢量结果文件 Vector file 命名为_pksw_sim2。

（10）单击 OK 关闭仿真设置对话框。

（11）从 File 菜单中选择 Save 保存仿真配置文件。

6.1.11　运行仿真

（1）单击执行仿真按钮。

（2）仿真完毕后关闭仿真消息对话框，并且关闭仿真配置编辑器。

分析结果：

配置并运行了两个不同仿真，现在需要使用分析工具来查看仿真结果。

比较两个场景的端对端延时和链路利用率:

（1）从 File 菜单中选择 New...，从下拉列表中选择 Analysis Configuration。

（2）单击创建结果图按钮

这时出现 View Results 对话框。

（3）找到刚刚在仿真配置中设置的矢量结果文件名：_pksw_sim1 和_pksw_sim2

（4）如图 6-36 分别选中两个场景的点对点链路上行利用率结果（它们隶属于同一个节点，以便比较）。

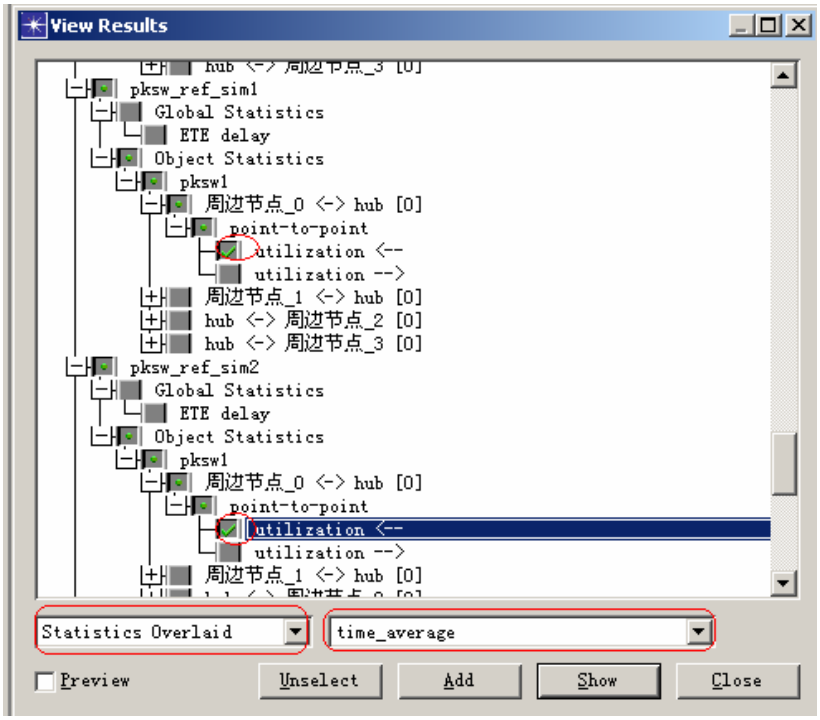


图 6-36　查看结果对话框

（7）在对话框下面选择结果显示视觉效果为 Statistics Overlaid，结果显示模式为 time_average。

（8）单击 Show 按钮。

将看到图 6-37 所示的链路利用率比较图。

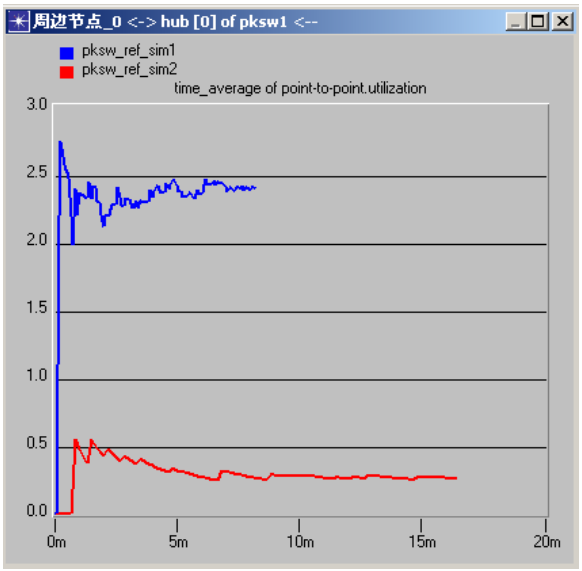


图 6-37　链路利用率比较图

可以看出包的产生速率过小导致链路利用率很低。

接下来查看包的端对端延时，确定包是否会在队列中积压。

（1）在刚刚的 View Results 对话框中单击 Unselect 按钮。

（2）改变结果过滤模式为 As Is。

（3）找到矢量结果文件_pksw_sim1，选择 Global Statistics: ETE Delay，单击 Show 按钮，出现如图 6-38 的结果。

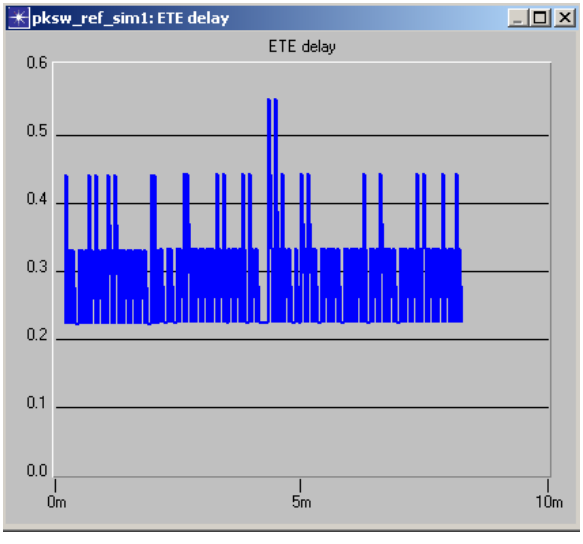


图 6-38　场景 pksw_sim1 的端对端延时图

（4）在 View Results 对话框中单击 Unselect 按钮。

（5）找到矢量结果文件_pksw_sim2，选择 Global Statistics: ETE Delay，单击 Show 按钮。出现如图 6-39 的结果。

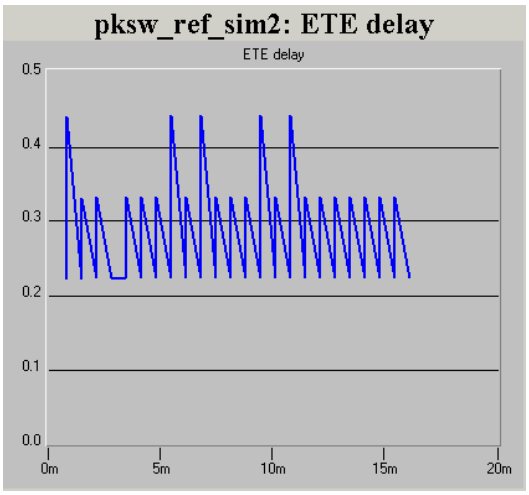


图 6-39　场景 pksw_sim2 的端对端延时图

线性的画图模式不能让我们知道每个包的确切延时。为了让每个包的延时更加清楚，我们将显示模式改为离散方式。

（6）在画图板上单击鼠标右键，在弹出的菜单中选择 Draw style→Discrete。

这时显示模式由线性变为离散，如图 6-40 所示。

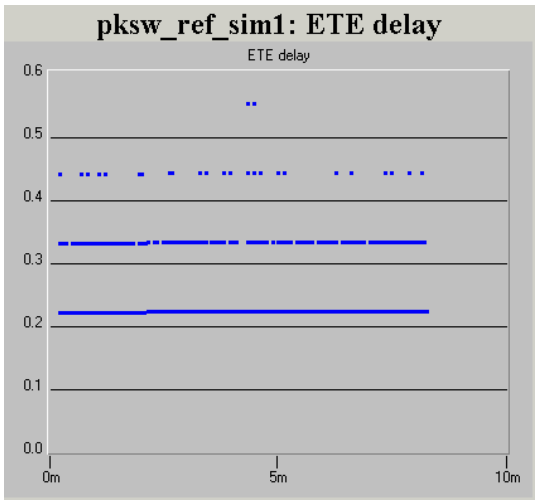


图 6-40　场景 pksw_sim1 端对端延时离散曲线

（7）同样方法改变另一副图，如图 6-41 所示。

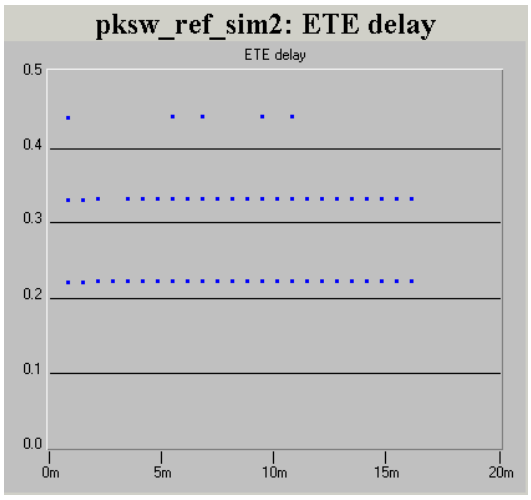


图 6-41　场景 pksw_sim2 端对端延时离散曲线

6.2　OPNET Modeler 核心函数介绍

6.2.1　动画类核心函数

动画类核心函数支持进程模型通过编写一系列图形操作命令来定义动画，这涉及到动画编程，是 OPNET 的高级应用。由于仿真并不支持直接显示动画图形，所以必须通过动画浏览程序（op_vuanim）间接地对动画请求进行解释，并显示动画。观看动画可以在项目编辑器 Results 菜单中单击 Play Animation，如图 6-42 所示。如果仿真过程有动画记录，则 OPNET 自动调用 op_vuanim 应用程序显示动画。

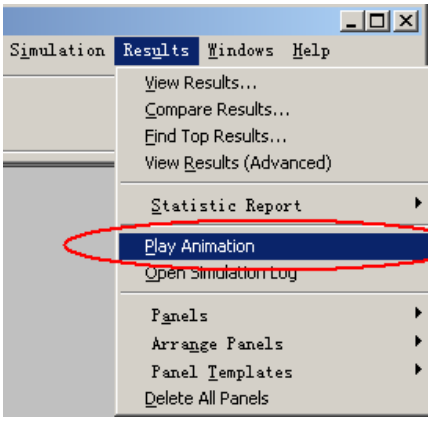


图 6-42　显示动画历史按钮

OPNET 动画技术涉及以下三个方面：

（1）动画类核心函数

（2）在探针编辑器中定义动画探针

（3）动画浏览器程序 op_vuanim

本节将对动画类核心函数进行描述，具体的函数使用可以参考第 11 章，首先我们列举与动画编程相关的一些术语：

（1）动画请求（request）

一个动画请求是构成动画的基本组成部分，例如，“画一条线”。一般来说，每个动画类核心函数发起单个动画请求，也有少数函数同时发起几个请求。

（2）绘图语法（graphical primitive)

一个绘图语法是一个动画基本操作，可以是绘制一个图标或者一个几何图形，如圆圈或矩形，或者更复杂的图形。有些动画请求直接为一个绘图语法，进行如绘制或者擦除等操作。

（3）模型对象（model element)

指节点、模块和状态（分别处在网络域、节点域和进程域中）等对象。一些动画请求对模型对象进行操作，如更新对象的图形属性或决定对象的位置等。

（4）动画流（flow）

一个动画流指仿真过程中发起的一系列动画请求。动画流在仿真开始时启动，并在以下三种情况下结束：

- 仿真终止。
- 发生致命的动画错误。
- 接收特殊的流终止请求。

如图 6-43 所示，在仿真属性中可以选择在仿真过程中启动动画浏览器程序（op_vuanim）同步播放动画，或者保存为动画历史。

（5）动画历史（history）

一个动画历史是一个动画流的完整记录，它保存为*.ah 文件，这些文件可以在仿真结束后通过 op_vuanim 动画浏览程序观看。

（6）动画浏览器（viewer）

动画浏览器是显示一系列动画效果的矩形窗口。

（7）图样（drawing）

图样是一个绘图语法操作的一个实例，如一个圆圈或线段。具体来说，它是通过实现一个绘图语法（可带某些参数，如图形位置和大小，以及填充属性）而创建的。

（8）立即模式（immediate-mode)

立即模式下的动画请求将图形直接显示到动画浏览器中。与之对应的是宏模式请求，它通过动画宏来定义将来要执行操作。

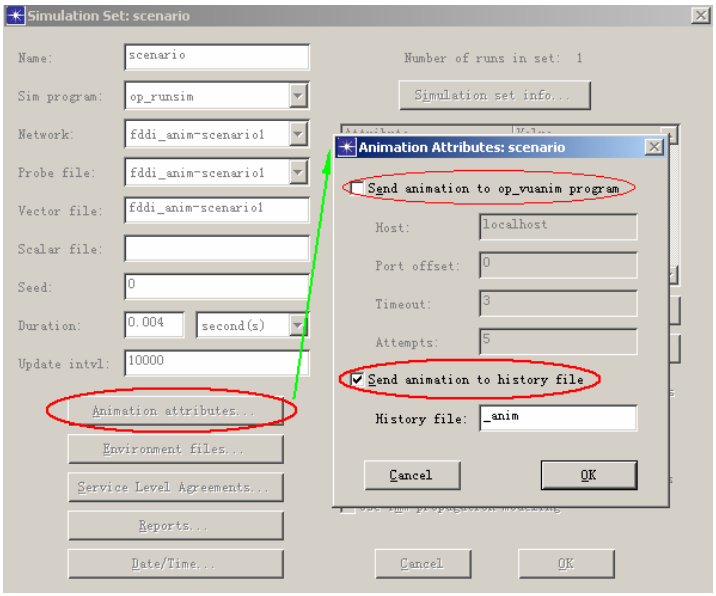


图 6-43　与动画相关的仿真属性

（9）动画宏（macro）

动画宏为几个动画请求的绑定，它代表一个特定的图形。例如一个动画宏可以绘制有两条对角线的矩形。动画宏提出的初衷是为重复绘制某个复杂图形带来方便，如果一个宏被多次使用，那么调用宏比重复发起一连串的动画请求更简洁有效。

（10）注册（register）

OPNE 为每一个宏的定制分配了一些存储空间，它们用来保存计算动画参数过程中的一些中间结果，它们本身没有任何意义，也可以把它们看作是为了计算参数而提供的“草稿纸”。但是 OPNET 对这些“草稿纸”的使用又制定了一些规则，在使用某个存储空间来保存变量时必须先注册，注册后变量被分配了一个标识符，即建立了变量与象征性标识符的映射，于是对标识符的运算或取值就等于对变量本身的操作。OPNET 为 int 型、double 型和 string 型的变量分别提供了 26 个标识符，之所以是“26”个，是因为英文字母 A~Z 正好有 26 个，数量已经足够，并且也好区分。

大部分动画核心函数如表 6-1 所示的分成以下动画子类。

表 6-1　动画子类函数简写及其扩展名与描述

简　写	扩　展　名	描　述
igp	Immediate-Mode Graphics Primitvie	针对动画浏览器的绘图语法操作
ime	Immediate-Mode Model Element	针对动画浏览器的模型对象操作
mgp	Macro-Oriented Graphics Primitive	针对动画宏的绘图语法操作
mme	Macro-Oriented Model Element	针对动画宏的模型对象操作

igp 子类函数直接对一个指定的动画浏览器发起立即模式绘图语法的动画请求。大多数

igp 函数绘制基本的图形组件，如图 6-44 所示。

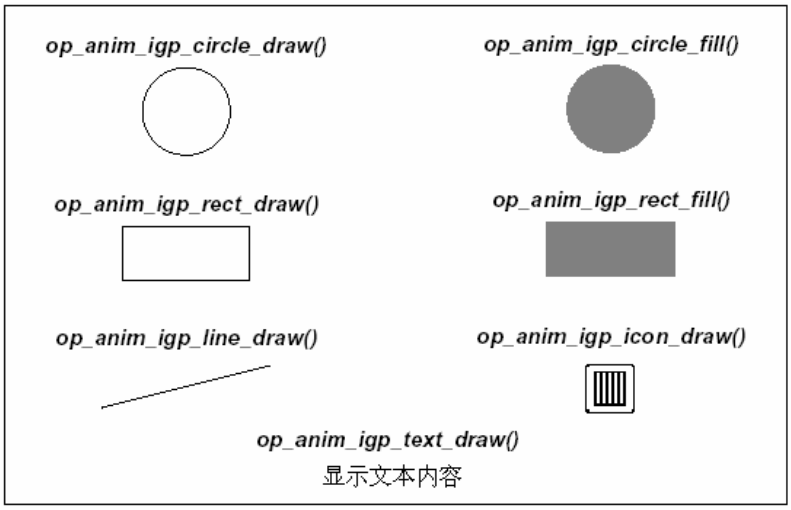


图 6-44　igp 子类函数及其绘制的基本图样

igp 函数接收两类参数，一类是浏览器中的显示参数（指定图形位置和大小），另一类是图形属性参数（包括前景颜色、背景颜色、填充模式、文字字体及对齐方式等）。指定这些参数，就可以生成图样。

igp 还有另外两个函数生成图样：op_anim_igp_bg_fill()，以指定的颜色和模式填充动画浏览器，op_anim_igp_macro_draw()调用带参数的预先定义的宏。图形更新使图形移动或改变以达到动画的效果，它可以分成以下两个步骤：擦除旧的图形；在新的位置下绘制另一个图，可能它的属性也改变了。第一步可以通过调用 op_anim_igp_drawing_erase()完成。对于宏图样，以上两个步骤可以一并由 op_anim_igp_macro_redraw()完成。

mgp 函数针对特定的宏指定一个动画请求，这个宏及其绑定的动画请求可供以后调用。

mgp 子类与 igp 的区别在于一个对宏进行操作，一个对动画浏览器直接操作，而指令是相同的，如图 6-45 所示。

Igp 子类函数	Mgp 子类函数
op_anim_igp_bg_fill()	op_anim_mgp_bg_fill()
op_anim_igp_circle_draw()	op_anim_mgp_circle_draw()
op_anim_igp_circle_fill()	op_anim_mgp_circle_fill()
op_anim_igp_icon_draw()	op_anim_mgp_icon_draw()
op_anim_igp_line_draw()	op_anim_mgp_line_draw()
op_anim_igp_rect_draw()	op_anim_mgp_rect_draw()
op_anim_igp_rect_fill()	op_anim_mgp_rect_fill()
op_anim_igp_text_draw()	op_anim_mgp_text_draw()

图 6-45　igp 于 mgp 子类函数列表

OPNET 给每一个宏分配了一些存储空间，OPNET 为 int 型、double 型和 string 型的变

量分别提供了 26 个标识符，正好为英文字母 A~Z。使用这些存储标识符是必须先注册。当一个宏注册了某个标识符后，它实际上申明了一个变量参数，举例来说，如果宏注册了存储半径的标识符，则在绘制宏时就可以任意指定该半径的大小，而生成我们想要生成的圆形图样。宏也支持数学操作，不像加减乘除运算直接用数字表示，OPNET 规定每个参与运算的数字都必须注册，即建立数字与标识符的映射，于是对标识符的运算就等于对数字本身的运算，并且结果也用某个标识符来表示。

OPNET 提供两个函数用来支持数学操作：一般先用 op_anim_mgp_reg_set()注册某个标识符，建立数字与标识符的映射，以备后用；然后使用 op_anim_mgp_arop()对两个标识符执行加减乘除运算，并将结果放入另一个结果标识符中。

一个宏的建立可以分为两个步骤：设置标识符和绘图。设置标识符只运算和存储绘图所要用到的参数。而绘图阶段利用前一阶段得到的参数绘制自定义的图形。当进程模型使用函数 op_anim_igp_macro_draw()调用宏时，建立和绘图阶段都将执行。而以刷新绘图方式调用宏 op_anim_igp_macro_redraw()时，建立标识符阶段不用再执行，只运行绘图请求代码即可，这也是将宏分成两个部分的原因。与之类似，当动画浏览器整个显示内容需要刷新（redraw)时，也只用执行宏的绘图部分的代码。在特殊情况下，我们也可以改变这个默认规则，通过指定 OPC_ANIM_SETUP 可以使首次调用宏时，只运行设置标识符部分的代码，而不调用绘图请求，而通过指定 OPC_ANIM_ERASE_MODE_XOR_RESET 可使再次调用宏时仍然执行设置标识符部分的代码。op_anim_mgp_setup_end()标识设置标识符完毕，暗含接下来的代码为绘图请求。如果没有使用 op_anim_mgp_mgp_setup_end()说明，宏只有绘图部分。

op_anim_macro_create()创建一个宏，它的输入参数为动画标签（在 Probe Model 中设置）返回一个宏 ID 号。op_anim_macro_destroy()销毁一个宏。当一个宏创建后，它使用于“开放”状态，也就是说它能够接收任何标识符的设置和动画绘制请求，但是调用函数 op_anim_macro_close()后，它就处于“关闭”状态，表示不在接受任何指令，这可以在设置多个宏时产生混淆。

除了目前讨论的与基本绘图语法相关的核心函数，ime 子类还提供更高级的操作，它直接在动画浏览器中生成显示效果。op_anim_ime_nmod_draw()在浏览器中显示一个模型，其输入参数为模型的名字 NamedMODel，可以是网络模型、节点模型和进程模型，这些模型实际上可以看成是动画的背景。与之相似的函数是 op_anim_ime_nobj_draw 和 op_anim_ime_npath_draw()，顾名思义，它们分别输入参数分别为网络对象和路径。

在显示模型图后，它的图形对象的属性可以通过 op_anim_ime_nobj_update()进行更新。图形属性包括位置（对移动节点而言），颜色（如路径）和图标（节点、模块或状态）。但是象像包和中断这样的动态对象不能在模型图中显示，因为它们不是模型的一部分，它们是在程序运行时动态生成的。然而通过调用 op_anim_ime_npath_traverse()可以显示包在包流，或者中断在状态中断线上的流动。

对基于移动的网络场景来说，移动节点和其他运动对象的动画能够产生强烈的视觉效果。op_anim_ime_gen_pos()可以将移动对象在网络模型中的坐标转化为动画浏览器中的坐

标，但是它只提供具有比例限制的坐标转换如 1∶1 或 2∶1。进程模型可以根据这些坐标计算节点的动态位置并且在相应位置绘制图样，这样视觉上能够感到物体在移动。当然这个函数也能得到区域边界或者固定节点的静态坐标，它们可以作为计算移动对象坐标的参考。

还有一些函数专门针对浏览器而设计。任何 igp 和 ime 函数的操作都必须针对某个浏览器 ID。进程模型可以使用 op_anim_viewer_open()创建一个浏览器，并获得其 ID 号，通过设置浏览器的属性还可以得到自己喜欢的大小和规格。与之类似，op_anim_viewer_open_std()创建一个标准规格的浏览器。

浏览器 ID 号的获取还有两种方法：（1）通过关联动画探针获得。OPC_ANIM_ANVID_PBSPEC 用来指定对应自定义动画探针的浏览器 ID 号；（2）通过对应动画探针标签获取，这里用到 op_anim_lprobe_anvid()，值得注意的是动画标签必须先在探针模型中定义好，否则不能得到正确的浏览器 ID 号，具体如何操作请参见 11.4 节的动画教程。

有时，我们希望动画运行到一定程序倒回从前，重新设置绘图参数从头播放，这时可用 op_anim_viewer_redraw()。当浏览器显示动画完毕，可以调用 op_anim_viewer_close()清空其所占内存。

另外几个函数用来控制动画流。op_anim_force()强制所有缓存的绘图操作全部在浏览器窗口中起作用。而一个动画流可以通过调用 op_anim_flow_end()强行终止，此后任何动画操作指令将视为无效。

6.2.2　分布类核心函数

分布类核心函数功能是按照指定的概率分布函数产生随机值。要求仿真表现出随机行为时，这些随机值作为输入参数是必不可少的，例如在计算中断的随机触发时间，随机生成包的目的地址等应用场合。

在仿真中普遍用到的随机数都服从以 0 为起点的均匀分布，op_dist_uniform()可以得到这样的值，其输入参数为一个非 0 的实数，代表取值的上限，函数等概率地返回一个从 0 到该界限之间的一个实数。一个非常值得注意的地方是 op_dist_uniform()结果不包含上限。另外一个较常用的是产生服从指数分布随机数的函数 op_dist_exponential()，它的输入参数为指数分布函数的平均值。

对于除此之外的随机分布，它们的使用就稍微复杂一些，我们可以采用两个步骤来取得这些函数的随机值：

（1）调用 op_dist_load()自定义一个带参数的随机分布函数。

返回值不是一个随机数，而是指向一个分布函数的指针（Distribution *），这个指针供以后使用，所以这个步骤一般是在进程的初始化状态完成的；

不管是对 OPNET 自带的一些分布函数，还是对 PDF Editor（分布函数编辑器）创建的

函数，都可以调用 op_dist_load()进行加载。当函数不再使用时还可以调用 op_dist_unload()将所占内存释放。

（2）函数一旦被加载，就可以在整个仿真过程中通过调用。

op_dist_outcome()用来产生一个服从自定义分布函数的随机值。这两个核心函数以分布函数指针为输入参数，返回一个随机值。表 6-2 列出 OPNET 支持的分布函数及其输入参数。

表 6-2 分布核心函数表

函数名称		输入参数#0	输入参数#1
Bernoulli	贝努利分布	均值	无
Binornial	二项式分布	样本个数	取样概率
Constant	常数分布	输出值	无
Exponetial	指数分布	均值	无
Gamma	伽码分布	尺度因子 Scale	形状因子 Shape
Geometric	几何分布	取样概率	无
Laplace	拉普拉斯分布	均值	尺度因子
Logistic	Log 分布	均值	尺度因子
Possion	泊松分布	均值	无
Raleigh	瑞利分布	尺度因子	无
triangular	三角分布	最小值	最大值
Uniform	均匀分布	最小界限	最大界限
Uniform_int	整型均匀分布	最小界限	最大界限

6.2.3 事件类核心函数

在仿真过程中，事件类核心函数为进程模型提供有关事件的信息。这些事件由仿真核心管理，按照执行时间的顺序被存储在一个事件列表中。事件列表的队首事件为当前要执行的事件，而事件类核心函数使用事件句柄（Evhandle）来对事件进行操作。

OPNET 提供三个函数访问事件列表中的事件。

（1）op_ev_current()返回当前事件的句柄。

（2）以一个有效事件为参考点，进程可以通过调用 op_ev_next()在事件列表中获得该事件的下一个事件。

（3）op_ev_seek_time()可以获得与输入的仿真时间最接近的那个事件的句柄。

当一个进程接收某个事件，也就是说，该事件作用于本地进程模块自身，则该事件被看作时本地事件。因此 op_ev_current()返回的肯定是本地事件，因为是这个事件唤醒本地进程的。而 op_ev_next_local()返回下一个本地事件。

事件类核心函数还支持管理并查找将来的事件。如果进程要遍历全部的事件，可以分为两步进行操作：

（1）调用 op_ev_count()得到事件的个数，当然也可以采用 op_ev_count_local()得到本

地事件的个数。

（2）事件个数为循环语句的上限，对每个事件进行操作。

有时当事件到达时，进程模块却认为该事件已经过期无效，例如进程发送包是为了确保它将来没有成功发送时重新再发，于是进程调度一个将来 5 分钟以后再次发送包的事件，但是 2 分钟过后，收到一个确认表明刚才的包发送成功了，于是这个“将来 5 分钟以后再次发送包的事件”就变得无效，这时可以调用 op_ev_cancel()在收到“2 分钟过后的确认”后将这个无效的将来事件删除。而 op_ev_pending()判断一个事件是否还在事件列表中等待调度。如果 op_ev_cancel()试图删除一个在事件列表中不存在的事件，则会出错，因此一般 op_ev_pending()配合 op_ev_cancel()使用，确保能够正确删除事件。

事件的两个常用属性分别是事件类型和调度时间。op_ev_type()可返回一个事件的类型，中断的类型在编程中用得较多，每种类型其实对应的是（1,2,3...）的整数索引号,OPNET 常用的事件（也可称为中断，中断是事件起作用的结果）类型如表 6-3 所示。

表 6-3　中断核心函数表

OPNET 中断类型常数	描　　述
OPC_INTRPT_FAIL	节点或链路失效中断
OPC_INTRPT_RECOVER	节点或链路恢复中断
OPC_INTRPT_PROCEDURE	程序调用中断
OPC_INTRPT_SELF	自中断
OPC_INTRPT_STRM	流中断
OPC_INTRPT_STAT	状态中断
OPC_INTRPT_REMOTE	来自外地进程的遥远中断
OPC_INTRPT_BEGSIM	仿真开始中断
OPC_INTRPT_ENDSIM	仿真结束中断
OPC_INTRPT_ACCESS	进程模块用来获取包流队列中封包的中断
OPC_INTRPT_MCAST	多播中断
OPC_INTRPT_PROCESS	进程调用中断

自中断、遥远中断或多播中断有着它们各自的用途。自中断最常用，模拟进程的各种延时（将来某个时刻）的处理。由于进程可以调度多个同种类型的事件，因此光用事件类型来区分是不够的，OPNET 为每个事件分配一个事件号（不是自动分配的，是用户调度事件时手动分配的），op_ev_code()可以得到这些事件号。流中断是由于包到达输入流端口时触发的，op_ev_strm()决定包到达的流索引号。状态中断由于特定模块参数改变而触发的。op_ev_stat()决定事件作用的状态线索引号，注意它只返回输入状态线索引号，而得不到状态线触发条件等信息。op_ev_time()可以返回事件调用的仿真时间。

大多数事件源于某个模块，例如流中断源于发送包的模块，自中断、遥远中断和多播中断源于调度它们的进程模块。为了获取产生这些事件的源模块 ID 号，可以调用 op_ev_src_id()。但是有三种事件是没有源模块的，因为它们是仿真核心自动生成的，它们是：仿真开始事件 bigsim，仿真结束事件 endsim 和 regular 事件。与 op_ev_src_id()对应，事件的目的模块 ID 号可以通过 op_ev_dst_id()获得。

op_ev_equal()提供比较两个事件句柄的支持。如果是事件句柄相同，则认为它们是同一个事件，这在进程等待一个特定事件时很有用，具体来说，首先进程得到该特定事件的句柄，然后每接收一个事件将当前事件的句柄和这个特定事件句柄相比较，一直等到特定事件的到达。

事件可以调度（scheduled），也可以强制（forced）执行。如果事件按照事件列表中按照正常顺序执行，则称为调度。而强制执行的事件不用在事件列表中按照先进先出顺序排队，直接“插队”而立即执行。

事件生成的同时也可以要求绑定一些相关信息，以便给接收该事件的目的模块以相关信息的提示。OPNET 提供两种方法：

（1）采用 op_ici_install()将界面控制信息（ICI）句柄与事件绑定，而 op_ev_ici()获得该界面控制信息的句柄。

（2）使用状态信息（State information)，它是比 ICI 机制更快速的传递信息的方式。op_ev_state_install()将用户自定义的数据结构与事件相绑定，然后通过 op_ev_state()返回该信息。

另外有时可以通过 op_ev_valid()来判断事件是否有效。它一般用在对一个事件的有效性感到怀疑的场合，如我们想知道一个事件是否仍然存在事件列表中，就可以调用 op_ev_valid()先做判断再进行其他的事件操作。

6.2.4 接口控制类核心函数

ICI 全称为 Interface Control Information，指接口控制信息。ICI 作为进程块间传递信息的载体，它是一种特殊的数据结构。ICI 可以再中断产生时与之绑定。ICI 可以用作分层协议间的协议会话工具，传递接口参数。ICI 和包一样，都是动态的仿真实体，因此它需要创建和销毁。op_ici_create()基于指定的 ICI 格式文件返回一个指向 ICI 文件的指针。

ICI 格式的文件需要通过 ICI 格式编辑器创建（请参见 5.9 节），它包括一个信息列表，每行信息由属性名称、数据类型及缺默认值组成。

当已定义过的 ICI 不再使用，则可以通过 op_ici_destroy()将它销毁。ICI 的信息一般只用一次，即进程接收到中断后，将 ICI 信息分离并读取出来，它就变得无效了。

新创建的 ICI 并不包含任何信息，它只是一个信息载体，还必须对它写入信息。op_ici_attr_set()设置一个 ICI 信息行。

虽然进程可以同时创建多个 ICI，并且给它们赋值，但是只有一个能够与输出中断相绑

定，op_ici_install()实现绑定功能。

当进程收到 ICI 后，可以通过调用 op_intrpt_ici()获取 ICI 指针，之后就可以通过 op_ici_attr_get()取得 ICI 属性的值。有时不能确保 ICI 是否包含某个属性，可以先调用 op_ici_attr_exists()来检查，再取值。

当进程期望接收某种格式的 ICI，而又不能确定接收到的 ICI 就是该格式，这时可以调用 op_ici_format()先核对格式。

在调试程序时，有时希望看到 ICI 的内容，op_ici_print()可以将 ICI 的内容打印在调试 DOS 窗口中。

6.2.5　标识类核心函数

标识类函数解决如何获取仿真对象 Objid 的问题。大量核心函数都需要以 Objid 为输入参数进行操作，所以 Objid 的获取很重要。仿真核心为每个对象分配惟一的 ID 号，即使是不同种类的对象，它们的 ID 号在任何情况下都不相同，如 Objid=5 的对象只可能是一个，不管它是进程模型还是节点模型。进程可以通过调用 op_id_self()获取自身的 ID 号。基于 OPNET 层次化建模的特点，仿真对象之间也可能有着继承关系。当一个对象包含另一个时，它被称为父对象（parent），而被包含的称为子对象（child）。例如，节点是进程模块的父对象，而子网又是节点和链路的父对象。

除了 Objid 外，OPNET 提供了另外三种区别网络域对象（如节点、链路和子网）的标识方法：

（1）对象名称，它是一个字符串。

（2）system id，对于同种类型的对象集合，它包含的每个对象都有一个单独的 system id，它是一个整数序列号。

（3）user id，它是用户设置的，可以不惟一，通常在对象的界面属性中设置，而 Objid 是系统分配的，全局惟一的，注意不要将 user id 与 Objid 混淆。

对于上述几种对象的标识方法，OPNET 还支持它们与 Objid 之间的转换，如 op_id_from_name()可将对象名称映射为 Objid，而 op_id_from_userid()将 user id 映射为 Objid。但是 OPNET 并不支持以上相反的映射，因为对象名称和 user id 为对象的属性，而对象的属性很容易调用 Ima(内部模块访问）类函数得到。

有时进程虽然知道对象 Objid，但是并不清楚它的类型，这时可以调用 op_id_to_type()查看它的类型。拓扑类核心函数还提供了一些由子对象 Objid 得到父对象 Objid，或者知道父对象 Objid 得到子对象 Objid 的操作，还有的函数通过物理上的关联得到周边对象的 Objid，感兴趣读者请参考 2.3.8 小节。

在标识类核心函数的使用中，有着一些对象继承的规则如下：

（1）每个对象可以看作是包含它的父对象的子对象，反之亦然。

（2）网络域对象有四种，分别是子网、节点、链路和总线接头（taps）。最高层（top）

全球网的 Objid 永远是 0，它是所有其他对象的祖父。节点、链路和总线接头是包含它们的子网子对象。

（3）节点域对象为模块（如进程、队列、包生成器、收信机、发信机和天线）。模块为节点的子对象。模块不能再包含其他模块，因此它不可能是其他模块的父对象。某些模块可以包含子模块对象，如队列模块是子队列的父对象，收信机发信机是信道的父对象。子模块对象也有 Objid，但是它们却没有对象名称。虽然有限状态机不是网络对象，但是它隶属的进程模块或队列模块有 Objid，所以一般来说将它们看作是同一个对象。

6.2.6 内部模型访问类核心函数

内部模块访问(Ima:Internal Model Access)类核心函数提供一套访问仿真实体的方法，仿真身体包括仿真属性、对象属性以及进程状态变量等。

仿真属性是和整个仿真密切相关的参数，进程模型可以通过 op_ima_sim_attr_get()访问这些属性，有时不能确保某个仿真属性是否存在，可以调用 op_ima_sim_attr_exists()进行判断，如果返回 OPC_TRUE 则表明属性存在。

对象属性隶属于特定的仿真对象，它可以由用户使用 OPNET 编辑器定义，或者通过 EMA（外部模块访问）方式设置，或者由底层对象属性提升（promote）得来。不管对象属性的设置方式如何，都可以通过 op_ima_obj_attr_get()取得它的值，而且也可以调用 op_ima_obj_attr_exists()判断某个对象属性是否存在。op_ima_obj_hname_get()可以获得某个对象的全称，它表明了对象的派生源（从哪些对象继承得来的）。

虽然 op_ima_obj_attr_get()提供了访问整型、双精度型和字符串型属性值的有效方式，但是它只支持一次访问一个属性，这对位置信息（涉及六个不同的属性）的获取是不够的。而且实际使用中节点的位置属性需要转换为地球坐标系坐标，op_ima_obj_pos_get()得到的六个与位置相关的属性包括两个部分：

（1）节点本身的位置属性。

（2）自动转换的地球坐标系坐标。

op_ima_obj_pos_get()得到的信息为经纬度和平面座标值两者之一，具体得到哪种信息要根据场景的网格度量单位是经纬度还是公里、米来定。

当需要用到无线建模时，移动站（移动节点或者移动子网）的位置随着时间改变。op_ima_obj_pos_get_time()在指定的时间获取移动站的位置。一般情况下，我们指定移动站沿着预先定义的轨迹文件移动，但是也可以通过 op_ima_obj_pos_query_proc_set()指定运动程序使移动站按照自定义的运动规律移动。如果进程想知道移动站何时改变过位置，可以调用 op_ima_obj_pos_notificaiton_register()，从而使移动位置发生改变时给进程触发一个中断。

除了对象的属性外，对象还可以通过绑定值或数据结构的方式包含状态信息，op_ima_obj_state_set()和 op_ima_obj_state_get()函数对提供设置和读取这些状态信息

的支持。

6.2.7　中断类核心函数

（1）仿真核心中断

在各个 process model 的 process interface 有一些 attribute：如 begsim intrpt、endsim intrpt、failure intrpt、regular intrpt 等，在设计模块些属性的设置很关键，尤其是 begsim intrpt 的设置，其影响 init 的状态的 Enter 代码何时执行，如果是 enabled，那么仿真一开始，即仿真 0 时刻，可以对 porcess 进行初始化，process 被触发后，即执行 init 的 Enter 代码。如果是 disabled 话，就不会被 kernel 触发。如果仿真结束时需要进行一些工作（如变量的收集，内存的释放等），则需要 enable endsim intrpt。regular intrpt 可用来做定时器，在 process interface 设定了 intrpt interval 之后，仿真核心每个该时间量触发一次 regular 中断。

（2）状态中断

stat_intrpt 函数可以用来提取统计信息作为反馈控制变量，将该信息反馈回模型中进行控制。

对 StatisticWire 可以这样理解：StatisticWire 将 A 进程的一个变量反映给 B 进程，该变量一般由 op_stat_write ()改变值，在 B 进程中：① 监测到 OPC_INTRPT_STAT 型中断以后，由 op_stat_local_read()读入。② 通过 op_stat_local_read()查询 stat 值，一般是在收到 OPC_INTRPT_STAT 中断时去查询。而进行 stat 触发，当 intrpt method 选择为 forced 方式，将直接进行触发；当 intrpt method 为 scheduled 方式时，有多种触发方式，比如 rising edge、falling edge trigger, repeated value trigger 等，按照触发方式在接收端引发 OPC_INTRPT_STAT。

小技巧 状态中断运用最多的地方是在信道接入层（MAC）中判断无线信道是否空闲。在节点模型中，一般收信机（receiver）到 MAC 有一条状态线，假设它的源状态（src stat）为 receiver.busy，状态目的地（dest stat）为 instat[0]，则判断无线信道是否空闲的程序如下：

```
double status;
status = op_stat_local_read(0); /*0 是目的状态的序号，即 instat 数组的下标*/
if (status==1.0)
        {表示信道忙，进行相应的处理}
else if(status==0.0)
        {表示信道空闲，可以发送包}
```

（3）流中断：

op_intrpt_strm()返回接收的流索引号（stream index）。

6.2.8 包类核心函数

包是 OPNET 中主要的数据模型，基于包的通信是 OPNET 仿真的主要通信机制，大多数网络应用都涉及到包的传输。包类核心函数介绍有关对包操作的函数集合。包的操作有三类：创建和销毁包；设置和得到包中的内容；析取包的相关信息和属性。

包是动态的仿真实体，并且有一定的生存期，在创建之后即担负承载数据的责任，一旦使命完成将会被销毁以释放所占用的内存，以供其他包使用。根据承载数据的要求，包可以分为有格式（formatted）和无格式（unformatted）两种。不管是哪种，包实质上是一种数据结构，也可以看作是一个容器，有时它包含的数据可以根据要求动态地增加或者减小，因此给它分配的内存也是动态改变的。op_pk_create()创建一个无格式的包，括号中的参数指示包的大小，这个函数可以看成是根据指定的规格（容器的容积）产生一个空的容器。而 op_pk_create_fmt()创建一个有格式的包，这个包的规格必须在包格式编辑器（Packet Format Editor）中定义好。这些包创建函数的返回值都是指向同一种包数据结构的指针，而区别只在于承载的数据不同。所有的包可以用同样的函数 op_pk_destroy()来销毁，并且可以调用函数 op_pk_type()来判断它的类型。

包的另外一种创建方式可以是复制一个已经存在的包。以某个包作为模板，op_pk_copy()函数可以复制一个包头和内容和原始包一摸一样新包，惟一不同的是包的创建时间和包的标识号（Packet ID）。事实上，包的创建时间为当前的仿真时刻，它们可能相同，但是包的标识号不会相同，仿真核心按照递增的顺序“0、1、2….”为包分配惟一的 ID 号，因此可以达到区分的目的。包的标识号可以通过两种方式得到：（1）调用 op_pk_id()；（2）在调试时打印此次跟踪操作所涉及的包的标识号，直到 Packet ID，我们并不能确定某个包是原始创建的，还是通过其他包复制得来的，这时可以调用 op_pk_tree_id()函数。当原始包创建时，它的树标识号（tree ID）就是自己的 Packet ID，随后它派生的任何包的树标识号将沿用这个 tree ID。当包在网络中重传或者泛洪时，它的树标识号就可以追踪它的源头。

现在来介绍包的时间戳的概念，它在数据通信中用得非常普遍。我们可以很形象地把包看作是一个包裹，邮局是仿真核心，邮寄包裹可以看成是包的传输，邮局将它寄出去之前必须盖上邮戳（这个邮戳可以看成这里的时间戳）标记什么时候和什么地点寄出去的，这实际上是邮局对包裹的一种管理方式，以便以后包裹丢失或者延期的查询。与之相似，通过调用 op_pk_stamp()，仿真核心标志创建时间为当前的仿真时间，地点为创建包的进程所对应的对象标识号（Objid）。当包被设置好时间戳后，当它经过任何进程时，所在进程都可以析取时间戳中的信息，得到上一次对包操作的时间（通过调用 op_pk_stamp_time_get）和地点（通过调用 op_pk_stamp_mod_get）。值得一提的是，包的时间戳可以通过调用 op_pk_creation_time_set()被进程修改，并非一定是最原始的创建时间。

除了手动调用 op_pk_stamp()函数设置时间戳，仿真核心在包创建时自动标记原始时间戳，可以调用函数 op_pk_creation_time _get()和 op_pk_creation _mod _get()来分别得到包的

原始创建时间和地点。

包的传输是 OPNET 仿真的主要行为。OPNET 规定包传输的两种方式，分别是“发送（sending）”和“传递（delivering）”。Sending 是通过连接模块与模块的包流（packet stream）来实现，而 delivering 不需要实际的物理连接。这两种传输模式针对不同的应用有各自的用途。

对于 sending，有下面 4 种方式：

（1）常用的发送方式是调用 op_pk_send()，当包沿着输出包流到达目的模块时立即向目的模块触发流中断。整个过程没有时延，所以包到达的时刻也是包发送的时刻。

（2）与第一种方式相比，如果要模拟包在包流传输过程的延时，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到达目的模块。

（3）前面两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输入队列中取出一个包，此时包到达引起的时间上不规则的中断显得无意义。

（4）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。

包的子域相关函数

op_pk_nfd_get("pkptr","dest_address",&dest_address）是从 pkptr 所指向的包中得到 dest_address 域的内容，放到地址为&dest_address 的内存中。

作者的理解：包域大小（field size）的设定和最终在这个包域中写入的内容的大小是没有多大关系的。例如，即使包域大小设置为0，也可以在该包域中写入100字节的内容，操作也是正确的。包域大小主要用来计算数据包的大小，进而计算发送时延。包的总长度可以通过op_pk_total_size_get()获得，包的长度等于所有field长度的和加上一个校正值（bulk size）。

当用 op_pk_get(pkptr,strm_index)接收包时，strm_index 为接收的流中断对应的输入流索引号，它可以通过 op_intrpt_strm()取得。在使用 op_pk_send(pkptr, strm_index)发送的时，这个 strm_index 的取值应和进程模块包流连接关系一致。

如图 6-46 所示，在进程模块上单击鼠标右键，选择 Show Connectivity，就可以观察与该进程模块相关的所有包流导向，例如 gen_0[0]→dp[0]，意思为源模块 gen_0 的端口号为 0 的输出流到达目的模块 dp 的 0 号输入流端口。

6.2.9　进程类核心函数

进程类核心函数为进程或者队列模块（本书中进程模块与队列模块统称为进程模块，不加以区分）提供创建和管理多个进程的支持。

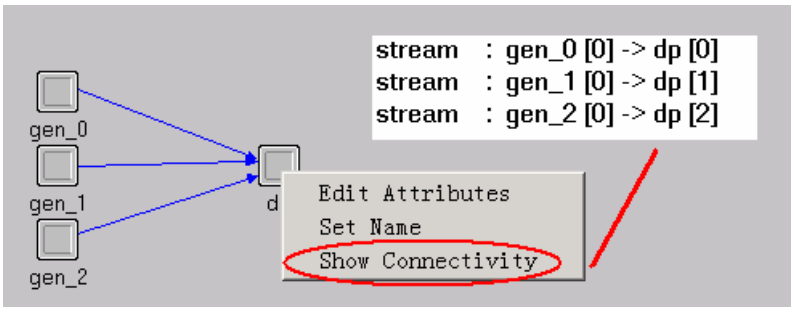


图 6-46 进程的包流连接关系

OPNET 中每个进程模块至少包含一个进程，这个进程称为根进程（root process），它在仿真开始时自动由仿真核心自动创建的进程模型实例。之后，可以在根进程中调用 op_pro_create()创建子进程。

创建子进程（child process）与根进程相比，子进程是在仿真中通过调用程序创建的。不光是根进程，任意的进程都可以通过 op_pro_create()创建子进程，它被称为该子进程的父进程。一个父进程可以有多个子进程（没有个数限制），而一个子进程只能有一个创建它的父进程。

当 op_pro_create()创建进程后，返回一个进程句柄（Prohandle），它是作为父进程调用、销毁或查询子进程属性的依据。进程也可以通过 op_pro_self()得到自己的进程句柄，与之类似，op_pro_parent()可以得到父进程的句柄，op_pro_root()得到根进程的句柄。

进程句柄实际上是一个多成员的数据结构，因此进程句柄之间不方便作比较，但是 OPNET 为每个进程分配了惟一的识别号，称为 process id，可以调用 op_pro_id()得到它。

每个进程隶属于某个进程模块，它可以使用与该进程模块相关的物理资源，如包流和状态线，具体来说，进程可以调用 op_pk_get()得到包流上的包或者通过 op_stat_local_read()读出状态线上的参数，这也暗含着另一层意思，即进程可以获得本属于进程模块的物理资源的所有权，而且随着进程的继承关系，根进程可以将该所有权传给其子进程，而子进程又能将该所有权传给更深层次的子进程。进程可以通过调用 op_pro_mod_objid()获得其隶属的进程模块的 Objid。

如果一个子进程的使命已经完成（例如某个进程管理一个逻辑信道，随着该信道的关闭，进程的使命也完成），可以通过 op_pro_destroy()将它销毁，这将释放存储进程状态变量的内存空间。

然而，仿真核心不会因为销毁某个进程而连带销毁其所有的子进程，而且仿真核心也没有子进程使用内存的记录，因此释放子进程内存的责任必须由子进程自己完成，事实上当一个父进程被销毁，它的子进程还是“活”的。

op_pro_destroy_options()执行类似的操作，但是更加具体，如从仿真核心事件列表中删除所有即将调度给被销毁进程的事件。在同一进程模块中，一个进程可以通过 op_pro_invoke()调用本模块中的其他进程（输入参数为被调用的进程的 Prohandle）。

注意这里强调“同一个进程模块”，如果被调用的进程隶属于另一个模块，则调用失败并出错。

进程一旦被调用（激活），则从调用它的进程中获取仿真核心控制权，一旦它停滞（例如它跳转到红色状态，并且执行完红色 unforced 状态的入口执行代码后，没有任何条件满足，这时进程停滞），则将控制权交会调用它的进程。

为了支持进程间的协同运作，OPNET 提供三种参数传递的接口内存，它们分别是 module memory、parent-to-child memory 和 argument memory，它们用在不同的场合。

除此之外，进程间可以直接通过状态变量传递参数。下面把四种内存共享机制列举如下：

（1）module 内存用得最普遍，op_pro_modmem_install()可以将其与进程模块进行绑定，于是所有隶属于该进程模块的进程都能够通过调用 op_pro_modmen_access()使用它。

（2）parent-to-child 内存支持进程与其父进程之间传递数据。当子进程通过 op_pro_create()被创建时，其父进程可以调用 op_pro_parmem_install()将要传递给该子进程的参数与子进程的 Prohandle 相绑定，而子进程可以通过 op_pro_parmem_access()得到这些参数。注意 parent-to-child 内存只能在进程创建时绑定一次。如果 op_pro_destroy_options()或 op_pro_destroy()将父进程销毁，其子进程还是“活”的，但是这些子进程变成了“孤儿”，所以它们调用 op_pro_parent()将出错，而父进程通过 parent-to-child 内存传给它们的参数也变得无效。

（3）与前两种共享内存机制不同，argument 内存基于进程的调用。当 op_pro_invoke()调用某个进程时，可以将绑定的 argument 内存传递给该进程，被调用的进程可以通过 op_pro_argmem_access()访问绑定的 argument 内存。

（4）为了访问其他进程的变量，调用 op_pro_svar_get()可以得到指向另一个进程中状态变量的指针。

6.2.10　队列类核心函数

队列类核心函数为队列模块提供管理队列资源的支持。值得注意的是，队列类核心函数只针对队列模块，进程模块或无线收发机管道程序不能使用。

队列由多个子队列（Subqueue)组成，换句话说，队列是由多个子队列拼贴在一起而形成的。子队列的大小由头和尾界定，它可以看作是一个包的列表，因此随着包的到达和离开，队列大小动态变化。

子队列类核心函数支持对子队列的操作，例如插入和访问包，而队列类核心函数不支持这些操作，它只针对队列（所有子队列的集合）。

一个子队列不包含任何包，则为空，一个队列的所有子队列都为空，则它为空。op_q_empty()判断队列是否为空。有时队列需要清空（或称为刷新），例如设备重新启动，这时可以调用 op_q_flush()。

队列模块自动为队列和子队列收集了大量统计量，OPNET 用一些常量表示这些状态，如表 6-4 所示，常量对子队列类核心函数同样适用队列的统计量可以通过 op_q_stat()获取。

表 6-4 队列核心函数

队列状态标识	描　　述
OPC_QSTAT_DELAY	队列中最后离开的包在队列中停留的时间
OPC_QSTAT_PKSIZE	当前队列中积压包的个数
OPC_QSTAT_FREE_PKSIZE	当前队列还可容纳包的个数
OPC_QSTAT_IN_PKSIZE	当前队列到达包的个数
OPC_QSTAT_BITSIZE	当前队列中积压的比特数
OPC_QSTAT_FREE_BITSIZE	当前队列还可容纳的比特数
OPC_QSTAT_IN_BITSIZE	当前队列到达的比特数
OPC_QSTAT_OVERFLOWS	队列溢出包的个数

包进入队列后，一些与之相关信息会被保存下来，例如什么时候进入队列的，在队列中积压（等待）多长时间等。包进入队列的时间可以通过 op_q_insert_time()获取，包的等待时间可以通过 op_q_wait_time()获取。

6.2.11 分割与组装类核心函数

分段和组装（Sar，Segmentation and Reassembly）类核心函数支持包的拆分，并且将包段(Segment，为包经过分段后形成的小片段)组装为原始包等操作。当长度不定的包要在只支持定长包（MAC 或物理层的固定帧 frame，或 ATM 网中大小为 53 字节的 cell）的信道上传输时，如果这些不定长包的长度过大，就必须将它们拆分为合适的大小。

1．术语

以下的术语涉及有关分割和组装的关键概念，它们内容互相关联并逐渐深入，建议读者按顺序阅读：

（1）分割：包的分割将生成两个新的包，它们共同代表原始包的净荷（Payload）。例如一个 100 比特的包，在第 40 比特位置被分割开，将生成一个 40 比特的包和一个 60 比特的包。

（2）合并：合并是与分割对应的操作，它将两个或更多的包进行融合从而形成一个新的包。融合的包长为所有参与合并包长的总和。

（3）分段：对一系列包进行分割与合并的操作称为分段。例如有三个 100 比特的包可以经过两次分段生成 2 个新的包，分别为 50 比特和 250 比特。首先将一个 100 比特的包分为两个 50 比特的包，然后将其中一个与另外两个 100 比特的包合并成 250 比特的新包。

（4）原始包：分段的操作对象称为原始包，一个或多个原始包成为每个分段的输入数

据。

（5）包段：经过分段操作得到的包。从传输的角度来看，包段可以看作是一个正常的 OPNET 包，换句话说，只要不涉及 Sar 类核心函数，对它们的操作和正常包是一样的。但是对于 Sar 类核心函数却能够识别出这些包段。包段并不包含生成它的原始包的真实数据，但是 Sar 类函数通过对它们的记录和管理，能够使包段看上去包含真实数据。因此不能对包段通过 op_pk_nfd_get()进行读取包域的操作，实际上包段为无格式（Unformatted)的包，只不过给它分配的长度是真实的。

（6）组装：将包段进行重构，恢复原始包的操作称为组装。重构过程中，包段中的虚拟数据转化为真实的原始包的数据。

（7）重分段（Resegmentation）：它和分段的操作类似，只不过操作对象变了，分段的操作对象为一系列的原始包，而重分段的操作对象为一系列的包段。

（8）Sar 缓存（Buffer）：对于分段、组装、重分段等操作，Sar 缓存提供管理参与这些操作的包序列（或包的列表），它存储这些“素材”以待进一步的操作，随着操作的结束 Sar 缓存的使命也完成，而被销毁。

（9）比特范围（Bit-range）：如果将缓存中的包列表看作是比特数据量的队列，那么列表中第一个包的第一个比特为队首，最后一个包的最后一个比特为队尾，队列中间的某些比特为包的边界，整个队列的比特量称为 full range，其中的任何一段连续的比特量称为一个 bit-range，它由两个参数来标识，分别是起始比特索引号和比特量的长度。队首的比特索引号为 0，队尾的比特索引号为比特量的长度减去 1。

（10）包插入操作：OPNET 提供几个 Sar 函数实现将包插入到 Sar 缓存的操作，这将在 Sar 包列表中增加一个包，并且立刻将新插入的包与已经存在的包相融合。

（11）包移除操作：OPNET 提供几个 Sar 函数实现将包从 Sar 缓存移除的操作，同时返回一个指向包的指针。

（13）刷新：一个 Sar 缓存为一个比特数量的队列，如果其中的一个 bit-range 不再有用，可以对它进行刷新的操作，刷新后 bit-range 并不转化为新的包，而是被删除。

（14）分段缓存（Segmentation buffer）：分段缓存为支持分段操作保存一个原始包的列表。原始包被插入到分段缓存中，而出来的是包段。

（15）组装缓存：组装缓存提供组装操作的支持，它包含两个包列表：

- 正在重构中的包列表，它包含一系列的包段，但是还缺一部分包段而不能构成完整的原始包。
- 已经重构的包列表，当包段收集到可以重构一个原始包时，OPNET 自动将其转化为原始包，这些原始包如果没有被移除则积压在该列表中。

（15）重分段缓存（Resegmentation buffer）：重分段缓存为支持重分段操作保存一个包段的列表。包段被插入到重分段缓存中，而出来的是新的包段。

（16）源缓存：分段和重分段缓存统称为源缓存，最为产生包段的源。

大多数 Sar 类核心核心函数可以根据操作针对的缓存类型来分类，如表 6-5 所示。

表 6-5 分割与组装类核心函数

简 写	缓存类型	相关的核心函数
buf	任何缓存	op_sar_buf_create()
		op_sar_buf_destroy()
		op_sar_buf_options_set()
		op_sar_buf_print()
		op_sar_buf_size()
segbuf	分段缓存	op_sar_segbuf_pk_insert()
rsmbuf	组装缓存	op_sar_rsmbuf_pk_count()
		op_sar_rsmbuf_pk_flush()
		op_sar_rsmbuf_pk_remove()
		op_sar_rsmbuf_seg_insert()
rsgbuf	重分段缓存	op_sar_rsgbuf_bits_flush_abs()
		op_sar_rsgbuf_lbl_pk_range_get()
		op_sar_rsgbuf_lbl_seg_range_get()
		op_sar_rsgbuf_seg_insert()
srcbuf	源缓存	op_sar_srcbuf_bits_flush ()
		op_sar_srcbuf_seg_access()
		op_sar_srcbuf_seg_remove()

除上表外，还有一些 Sar 函数不针对缓存操作，如 op_sar_pk_is_segment()判断一个包是否为包段，以及针对 bit-range 操作的函数。

2．有关分段缓存的用法:

首先，调用 op_sar_buf_create (buf_type, options)创建一个缓存，函数返回这个缓存的句柄（Sbhandle）。然后，调用 op_sar_segbuf_pk_insert (sbhandle, src_pkptr, src_pk_label)使用这个缓存，存入一个包，前两个参数很好理解，最后一个参数为指定原始包的标签（label for the specified source packet），默认值为 0，即不使用包标签功能。当然也可以使用它来标识某个特定的 packet，op_sar_rsgbuf_lbl_pk_range_get()是包标签的应用，它根据包标签在组装缓存中找出该包对应的比特范围，然后对这个比特范围所做的操作就等于对包的内容进行操作。要将一个原始包进行分段传输时，首先获得它的大小 pk_size，再调用 op_sar_segbuf_pk_insert()把它插入分段缓存中，然后就可以用 op_sar_srcbuf_seg_remove()取出比特量为 segment_size 的包段，随后将包段发送出去。接下来就用到包的大小 pk_size，因为事先并不知道原始包能够分成多少个包段，所以需要一个循环语句，每发一个包段就将 pk_size 减去 segment_size，直到 pk_size 小于 0，最后一个包段的比特量通常小于

segment_size。在接收端，收到一个包后，调用 op_sar_pk_is_segment()判断它是否是一个包段，如果是就调用 op_sar_rsmbuf_seg_insert()将其加入到组装缓存中，然后通过 op_sar_rsmbuf_pk_remove()判断原始包是否完全重构，如果是则移出一个完整的原始包，如果不行就返回 OPC_NIL，包段的重构是 OPNET 自动完成，用户不用管哪几个包段属于某个原始包。

接下来讨论原始包及其包段的 Packet id 问题。假设原始包 Packet id 为 a，在发送端放入分段缓存中再取出其包段，由于包段是系统动态创建的新包，因此它分配了一个新的 Packet id（假设为 b）。接收端收到 Packet id 为 b 的包段时，将其送入组装缓存，当所有包段都被收集并放置在组装缓存中后，原始包被重构，它的 Packet id 仍为 a。仿真类核心函数

6.2.12　统计类核心函数

统计量（Stat，Statistic）类核心函数针对用户自定义或者仿真自动创建的统计量数据，将这些数据记录到结果文件中。

OPNET 提供两种类型的结果文件：矢量文件（Vector）和标量文件（Scalar）。

矢量文件包含动态的，基于事件的十进制数据，这些数据跟踪统计量随时间变化的情况，每个数据点都是在某个时刻访问矢量文件生成的。一个矢量文件只能包含一次仿真的数据，换句话说，仿真过程中不能将新的数据加入以前创建的矢量输出文件中。每个矢量统计量在隶属的探针模型（Probe Model）文件中都对应一个探针，而标量统计量不需要在探针编辑器中定义探针。

标量输出文件可以收集由许多仿真共同产生的结果，具体来说，对于一系列仿真，每次仿真更新一次参数得出一个新的结果，我们希望将每次仿真的参数与其对应的结果画成一条曲线，这时就可以采用将结果写入标量输出文件的方法。与矢量文件相比，标量文件包含非动态的数据。标量文件以数据块的方式组织数据，每一次仿真的所有标量数据被写入一个相应的数据块中。

所有的节点中的模块（除了天线）都自动计算内嵌的统计量。通过状态线，这些统计量信息可以被传输到进程模块中，并影响进程模块的行为，同时它们也可以被探测并且状态信息的变化可以记录到矢量输出文件中。基于结果收集的范围，统计量可以分成两种类型：

（1）本地统计量（Local statistic），它只针对某个模块。只关注单个模块的行为时可以采用这种类型的统计量。

（2）全局统计量（Global statistic），它针对整个网络模型，关注整个网络的行为和性能，例如对网络包的端对端延时性能的测试，它并不关心某个包的源和目的，只关心所有包的延时性能的统计结果。

op_stat_annotate()为矢量输出文件中的一个状态统计增加一个标签。op_stat_rename()

对矢量输出文件中的一个状态统计重命名。op_stat_reg()根据进程模块中统计量的名字（在Process Model 编辑器中的Interfaces 菜单中选择Local Statistics 或Global Statistics 定义的Stat Name 属性）返回一个统计量句柄（Statistic handle)，它作为进程写入本地或全局统计量的依据。op_stat_obj_reg()与 op_stat_reg()类似，但不局限于应用在进程模块，它还可以用来访问链路、路径、子模块的本地统计量。op_stat_dim_size_get()得到进程模块中定义的统计量的维数（Dimension)，而 op_stat_obj_dim_size_get()还可以得到路径、链路等对象的统计量的维数。op_stat_write()在当前时刻将结果写给某个指定的统计量。op_stat_write_t()在某个指定的时间将结果写给某个指定的统计量。标量统计量可以在仿真过程的任何时间写入。对于每个标量统计量只对应一个值，它与仿真时间无关，只和当前整个仿真有关，op_stat_scalar_write()直接将结果写入标量输出文件中。

除了以上将统计量写入文件的操作，进程模块还支持读取输入统计量的操作（此时进程模块为状态线信息的目的地），op_stat_local_read()可以得到指定状态线的当前值，这种方法是进程模块（或与其他节点模型内对象）间的一种通信机制。

6.2.13 队列和子队列类核心函数

子队列（Subq，Subqueue）类核心函数为队列模块提供管理子队列资源的支持。op_subq_empty()判断一个子队列是否为空，如果返回值为 1，则表示队列不包含任何包，即为空，如果为 0 则不为空。队列模块自动为子队列收集了大量统计量，它们可以通过op_subq_stat()取得，OPNET 用一些常量表示这些状态，请参见 6.2.10 小节队列类核心函数。

由于队列模块包含一个队列，这个队列又包含很多子队列，所以存在子队列的识别问题，OPNET 提供两种子队列识别机制：

（1）每个子队列都被分配一个惟一的数字索引（0,1,2,3...）。

（2）根据子队列的特点来区分，例如长度最大的队列 OPNET 用常量OPC_QSEL_MAX_PKSIZE 来标识，包含比特数最小的子队列用OPC_QSEL_MIN_BITSIZE，以这些常量作为输入参数可以通过 op_subq_index_map()找到对应的子队列索引号，有关的子队列标识常量如表 6-6 所示。

表 6-6 队列类核心函数

队列特点标识	描　　述
OPC_QSEL_MAX_DELAY	到当前时间为止，包积压时间最长的子队列
OPC_QSEL_MIN_DELAY	到当前时间为止，包积压时间最小的子队列
OPC_QSEL_MAX_BITSIZE	具有最大比特量的子队列
OPC_QSEL_MIN_BITSIZE	具有最小比特量的子队列
OPC_QSEL_MAX_AVG_BITSIZE	具有平均最大比特量的子队列

续表

队列特点标识	描　　述
OPC_QSEL_MIN_AVG_BITSIZE	具有平均最小比特量的子队列
OPC_QSEL_MAX_AVGI_BITSIZE	对于当前到达的包，平均比特量最大的子队列
OPC_QSEL_MIN_AVGI_BITSIZE	对于当前到达的包，平均比特量最小的子队列
OPC_QSEL_MAX_FREE_BITSIZE	当前还能容纳比特量最多的子队列
OPC_QSEL_MAX_IN_BITSIZE	当前到达包的比特数最大的子队列
OPC_QSEL_MIN_IN_BITSIZE	当前到达包的比特数最小的子队列
OPC_QSEL_MAX_OVERFLOWS	溢出量最大的子队列
OPC_QSEL_MIN_OVERFLOWS	溢出量最小的子队列
OPC_QSEL_MAX_PKSIZE	积压包个数最多的子队列
OPC_QSEL_MIN_PKSIZE	积压个数最少的子队列
OPC_QSEL_MAX_FREE_PKSIZE	当前还能容纳包数最多的子队列
OPC_QSEL_MAX_IN_PKSIZE	当前到达包个数最多的子队列
OPC_QSEL_MIN_FREE_PKSIZE	当前还能容纳包数最少的子队列
OPC_QSEL_MIN_IN_PKSIZE	当前到达包个数最少的子队列
OPC_QSEL_INVALID	无效的子队列

调试时，希望看到子队列内容的变化情况，可以调用 op_subq_print()，会将子队列当前信息打印在调试 DOS 窗口中。对于每个进入子队列的包，可以通过 op_pk_priority_set()为其设置优先级，相应地 op_pk_priority_get()可得到某个包优先级。配置优先级后，子队列就可以调用 op_subq_sort()对这些包按照优先级排序，越靠近队首的包优先级越高。

有时子队列需要清空（或称为刷新），这时可以调用 op_subq_flush()。

除了以上函数，最基本的子队列操作为插入、访问、删除和查找包，每次操作只能针对一个包。op_subq_pk_insert()将包插入子队列的某个指定位置，相应地 op_subq_pk_remove()从某个位置取出包，并将它从子队列中删除，而 op_subq_pk_access()只得到指向包的指针，但并不在子队列中删除它。包在队列中的位置通过 0,1,2,3...的索引号标识，0 代表队首位置，依次递增，值最大的索引号为队尾。而在不知道包在子队列中的索引号时，也可以使用一些 OPNET 常量来标识，如 OPC_QPOS_HEAD 代表队首，OPC_QPOS_TAIL 代表队尾，或者通过包优先级 OPC_QPOS_PRIO 来映射索引号。有时需要将子队列中两个包互换位置，这时可以使用 op_subq_pk_swap()。

6.2.14　表格类核心函数

表（Tbl，Table）类核心函数专门解决如何读取天线模型和调制模型数据的问题。天线

模型和调制模型分别在无线收发机管道阶段中计算天线增和 BER 时使用，由于它们涉及的数据量大（参见 5.8 节的天线模式编辑器和 5.10 节的调制曲线编辑器），OPNET 通过表的形式将这些数据组织起来。以调制曲线表的名字为输入参数，op_tbl_modulation_get()得到调制曲线表的句柄（Modulation handle)，根据该句柄，op_tbl_mod_ber()可以查找某个信噪比（SNR，Eb/No）对应的 BER 值。op_tbl_pattern_get()根据天线 Objid 得到其天线模型表的句柄（Pattern handle)，然后 op_tbl_pat_gain()根据这个句柄查找指定的 phi 和 theta 对应于天线模型的增益值。

6.2.15 传输类核心函数

传输类核心函数对一个包的收信机管道数据属性（TDA，Transceiver Pipeline Data Attribute)进行操作。TDA 的引入提供了仿真核心与实现链路模型的外部函数的数据接口。这一系列外部函数被称为管道阶段（Pipeline stage，参见第 9 章无线信道建模）。从发信机到收信机，包在链路上传输经历了一系列管道阶段（传输阶段的个数视链路的类型来定），这些管道阶段的计算环环相扣，后面阶段的计算用到前面阶段运算的结果，这就需要一种数据共享的机制来满足管道阶段计算的要求，于是包的 TDA 就充当存储共享信息的载体。

根据 TDA 所需的数据类型，op_td_set_int()、op_td_set_dbl()和 op_td_set_ptr()分别将整型、双精度型和指针型的变量赋值给 TDA。与之对应，op_td_get_int()、op_td_get_dbl()和 op_td_get_ptr()分别取出整型、双精度型和指针型的 TDA 数据。最后，op_td_is_set()判断某个 TDA 属性是否设置了参数。

6.2.16 拓扑结构类核心函数

拓扑（Topo，Topology）类核心函数决定网络和节点的拓扑结构。网络的拓扑结构为节点和链路之间的连接关系，而节点的拓扑结构为进程模块和包流（或状态线）之间的连接关系。

op_topo_object_count()返回某种指定类型静态对象的个数，这些对象包括节点、链路、模块、复合属性（compound)等。op_topo_object()以对象类型和索引号为输入参数，返回 Objid。

探求网络拓扑结构可以参考对象的继承关系。子对象（Child）是包含在父对象（Parent）中更底层的对象。例如，模块是节点的子对象，而节点又是子网（Subnet）的 子对象；子队列和信道为复合属性的子对象，而复合属性又是队列、发信机和发信机的子对象。探求网络拓扑结构的核心函数有三个：op_topo_parent() 返回父对象的 Objid；op_topo_child_count()返回指定类型子对象的个数；这个函数一般和 op_topo_child()配合使用，根据子对象的个数循环调用它可以获得所有包含的子对象 Objid。

与对象的输入输出端口直接相连的对象分别称为它的输入输出关联（Association)。例

如，有线链路中，进程模块与点对点收信机相连，则进程模块的输入关联为收信机，收信机的输入关联为点对点链路，给链路的输入关联为另一个节点中的发信机。指定关联方向("in"或者"out")和对象 Objid，op_topo_assoc_count()可以返回与该对象关联的所有对象的个数，而 op_topo_assoc()返回一个指定关联的 Objid。

一个连接器（Connector）对象连接两个对象。例如，包流可能连接两个模块，而链路可能连接两个节点，所以包流和链路都是连接器对象。op_topo_connect_count()和 op_topo_connect()分别返回两个对象间连接器的个数以及连接器的 Objid。

拓扑类核心函数可以用来获取两个节点间的链路 Objid。具体来说，在需要此信息的进程模块中，使用 op_id_from_name()获得同一节点模型中所有发信机的 Objid，然后调用 op_topo_assoc()获取与进程模块相连的收信机（在另一个节点模型中）的 Objid（如果没有找到，op_topo_assoc()将返回 OPC_OBJID_INVALID），最后用 op_topo_connect()就可以得到收发信机对间的链路 ID。

6.2.17　编程类核心函数

编程（Prg，Programming）类核心函数可以分为五个部分，它们针对不同场合的需求，而且基本上互相独立，以下分别介绍这些子类函数：

1．通用数据文件（General Data File)子类

通用数据文件是以*.gdf 为后缀的 ASCII 码文件，可以用来存储用户自定义的路由表、地址映射表以及进程模型配置表等，op_prg_gdf_read()支持对通用数据文件采用列表的方式（列表数据结构参见下一部分的列表子类函数描述）进行访问。

2．列表子类

列表子类包含 9 个核心函数，它们管理数据元素的集合。列表有以下属性：

（1）列表数据元素可以是任意类型，从常用的整型或双精度型到复杂的自定义的数据结构。

（2）列表可以包含不同种类的数据元素。

（3）列表大小没有限制，可以按照需求任意添加新的元素。

（4）列表包含的异质的数据结构通过指针相连。列表的两端分别称为表首和表尾，它们是封装在列表数据结构（List*）的指针。

（5）列表指针可以保存在 List*型的变量中。List*变量可以在很多地方定义，如进程模型的临时变量块、状态变量块、头块（全局变量），以及外部 C 文件。List*变量还可以写入包域中，这样可以将它从一个进程模块捎带到另一个进程模块中。

（6）有关列表的操作有元素的插入、访问和删除等。对于排序的列表，prg_list 类函数提供查找算法将元素插入到合适的位置。

（7）列表属性可以是排序的或非排序的，因此相应地元素的插入有以下两种方法：

- op_prg_list_insert()在指定位置插入元素，并将列表属性更改为非排序的。

- op_prg_list_insert_sorted()只针对排序的列表，将新的元素插入到合适的位置并保持列表属性（排序的）不变。

一般来说，访问一个排序列表中的元素比非排序列表快，图 6-47 为列表数据结构的内存组织构架：

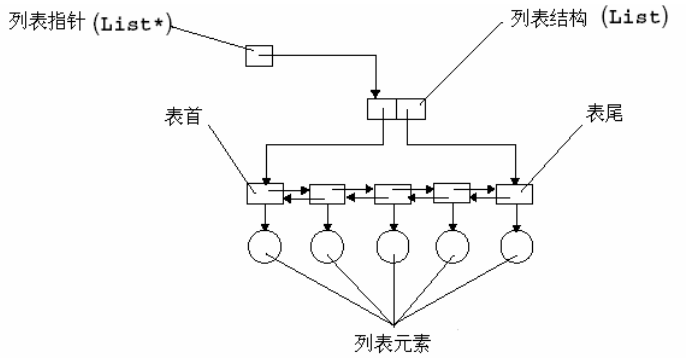


图 6-47 列表数据结构的内存组织构架

分配列表内存有两种方法，分别是动态分配和静态分配。动态分配的列表在仿真需要时通过 op_prg_list_create()创建，使用完可以通过 op_prg_mem_free()销毁，它还有可在进程模块间移植的优点。静态分配的列表必须先通过 op_prg_list_init()进行初始化。

一旦列表被创建或者初始化，就可以对它进行元素的插入、访问和删除等操作，进行这些操作之前必须知道元素的位置。元素的位置由索引号来标识，0 表示表首，中间元素依次增 1，表尾索引号最大。op_prg_list_insert()插入一个元素，op_prg_list_remove()移除一个元素并返回指向该元素的指针，op_prg_list_access()得到元素的指针但是不删除元素。op_prg_list_size()得到列表中元素的个数，它一般被用作循环调用的次数，与 op_prg_list_access()配合使用可以访问列表中所有元素，同时也可以使用 op_prg_list_remove()移除想要得到的元素。op_prg_list_sort()使用自定义的元素比较程序对列表排序，这样针对元素内容提供灵活的排序方法。op_prg_list_elems_copy()将源列表的内容复制到目的列表中。如果一个列表不再使用，可以通过 op_prg_list_free()释放内存。

3．内存和池内存（Pmo：Pooled memory）子类

在 OPNET 进程模型中，大多数数据都通过申明静态变量的方式来存储，然而变量对内存的需求量随着不同事件而变化时，就需要用到动态内存分配。除了传统的 malloc()和 free()分配和释放内存的方法，OPNET 还提供为增强调试功能而设计的内存分配和释放函数，分别是 op_prg_mem_alloc()和 op_prg_mem_free()。当某块内存被大量使用，并且重复被分配和释放，这时如果把它设置为池内存 Pmo 可以增加仿真的性能。当内存块通过 op_prg_pmo_define()被指定为池内存时，就成为一个单独的对象，称为池内存对象（pooled memory object）。op_prg_pmo_define()返回 Pmohandle 供后续操作的依据，以该 Pmohandle 为输入参数，op_prg_pmo_alloc()可以指定池内存块的大小。池内存有三个属性，分别是 name、size 和 increment size。name 作为 ODB 调试时跟踪内存使用情况的依据；size 的单

位为 byte，标识池内存的大小；increment size 指定一次能够获取 Pmo 对象的个数，这个值越大 Pmo 内存分配器的执行速度越快，但是会降低系统内存的使用效率当系统内存资源不足时该值不能设置得过大，一般在 25~100 之间取值比较合适。

4．ODB 子类

ODB 为控制和管理仿真行为提供一个交互式环境。ODB 支持断点（Breakpoint)定义，跟踪并显示仿真诊断信息。ODB 功能的实现有赖于进程模型中编写相应的程序支持，作为 ODB 命令激活调试状态（breakpoint、trace 和 action）的依据，ODB 子类函数用来编写这些 ODB 支持程序。

op_prg_odb_bkpt()在进程模型中设置一个断点，这个断点以断点标签来标识，通过 ODB 命令 lstop（针对所有断点标签）和 mlstop（针对指定模块的断点标签）激活断点标签，则程序运行到 op_prg_odb_bkpt()设置过该标签的地方就会中断。

ODB 支持几种显示跟踪信息的命令。fulltrace 显示所有事件调用过的程序，不过这样可能会因为内容太多而不容易找到想要的信息。mtrace 针对某个指定模块，显示与调用该模块所执行的程序。pktrace（针对某个包）和 pttrace(针对一个包树）显示包参与执行的所有程序。ltrace 和 mltrace（范围更小，只针对某个指定模块）显示标签界定的程序段所生成的诊断信息。

对应以上命令，分别有一些 ODB 子类函数支持它们的实现，在进程模型中判断是否激活相应调试信息的显示。op_prg_odb_trace_active() 支持 fulltrace 和 mtrace；op_prg_odb_pktrace_active()支持 pktrace；op_prg_odb_ltrace_active()支持 ltrace 和 mltrace。除了传统的 printf()可以将信息打印在 ODB 窗口外，OPNET 还提供显示标准 ODB 跟踪信息的函数：op_prg_odb_print_major 和 op_prg_odb_print_minor()。

5．字符串（Str，String）子类

Str 子类函数只有一个 op_prg_str_decomp()，它一般与 op_prg_gdf_read()配合使用，在 op_prg_gdf_read()得到 GDF（通用数据文件）句柄后，op_prg_str_decomp()将 GDF 文件中的字符串解释为列表域，这种解释是基于对文件中逗号、跳格符和冒号等符号的识别。

6.3　子　进　程

OPNET 建模层次的划分一般为网络层、节点层、进程层，实际上在进程层中还可以细分为根进程、子进程、孙进程等，它们构成更加完整的 OPNET 编程环境，如图 6-48 所示。

默认情况下，进程模块接收到中断时，通过调用根进程的方式将该中断传给进程模块。考虑到更复杂的应用场合，如果想要一个进程模块的行为更加逼近实际情况，例如执行以下操作：（1）执行多个不同的任务；（2）分多阶段地处理信息；（3）一个模块需要管理多个队列；（4）服务器同时与多个客户端通信。

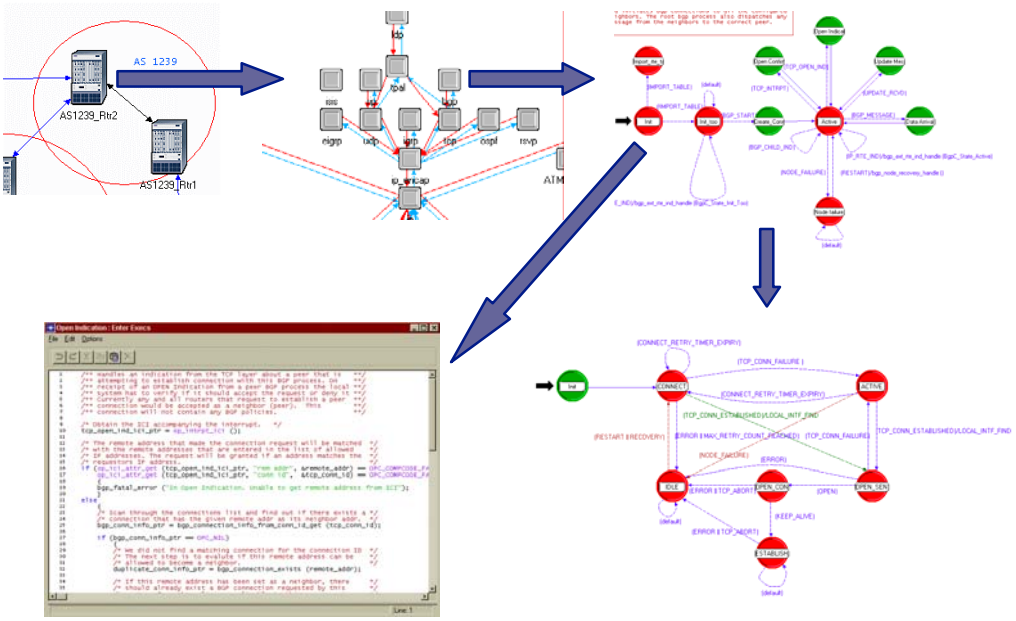
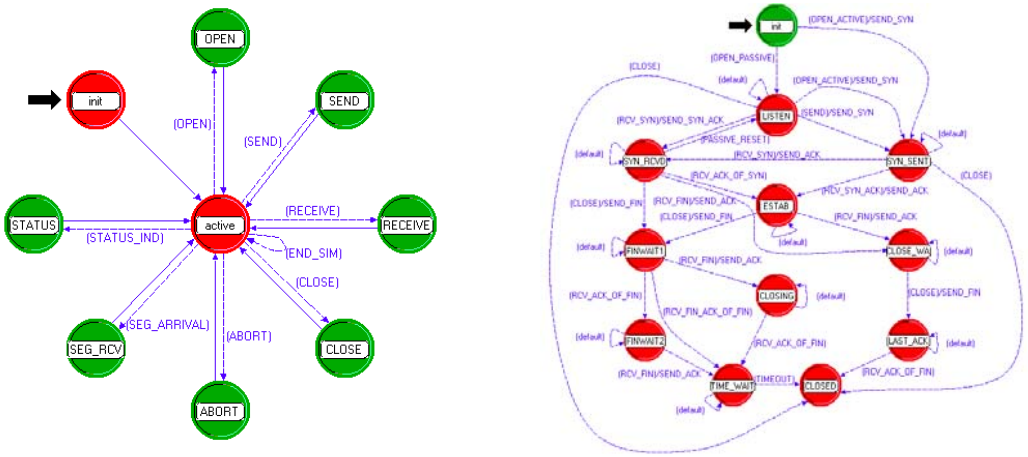


图 6-48　完整的阶层性的 OPNET 编程环境

对于这种需要并行处理多个交互式事件的情况，如果仍只使用一个根进程，则会使得进程模型变得复杂，状态转移关系不好确定。这时就需要多个进程协同运作，子进程可以看作时是处理事件的一个实例。

另外子进程也可以作为简化有限状态机结果的途径，如图 6-49 所示，TCP 跟进程采取一种花瓣式的结构，满足一个条件执行一个绿色状态之后回到 idle 状态等待中断，绿色的状态可以调用子进程，它编程结构清楚，因此作为 TCP 会话的管理进程，而为每个会话创建一个子进程，TCP 协议的主体是在该子进程实现的。



tcp_manager_v3.pr.m　　tcp_conn_v3.pr.m

图 6-49　TCP 协议的实现

子进程是 OPNET 编程的比较深入的技巧，本节主要讨论关于进程的一些关键概念和核心话题，针对子进程的操作和有关多个进程的管理请参见 6.2.9 节的进程类核心函数。

使用之进程的大致步骤如下：

（1）把子进程创建好；

（2）在根进程中包含子进程，如下图 6-50 所示；

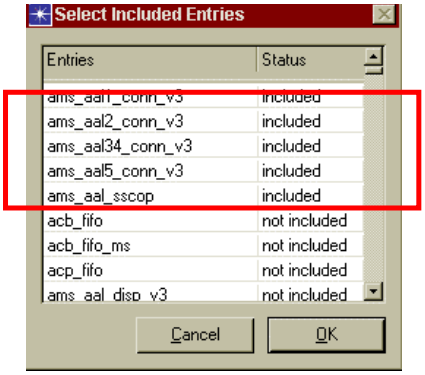


图 6-50　包含子进程

（3）在根进程中使用 op_pro_create 创建子进程；

（4）选择 6.3.3 节 4 种方法之一唤醒子进程，这 4 种方法又分为两类，一类是由根进程调用 op_pro_invoke 手动唤醒子进程，另一类是子进程通过注册机制跳过根进程直接获取仿真核心控制权。

6.3.1　有关进程的几个概念

对子进程的了解必须首先清楚关于进程的几个概念，它们分别是进程模块，进程模型，进程和有限状态机。

关键概念　进程模块（Process module，或 Processor）：它是一个静态的物理对象，具有 Objid，它可以包含多个进程。

关键概念　进程模型（Process model）：它实际上是由进程模型编辑器创建的有限状态机，表示协议逻辑上的行为，不是一个网络对象，没有 Objid，一个进程模型可以被创建为多个进程（它们被称为同种类型的进程，以 Process ID 来进行区分）。进程模型在没有被创建为进程时对仿真没有任何意义。

关键概念　进程（Process）：根进程、父进程和子进程的统称，它是进程模型的一个实例，由仿真核心创建的进程称为根进程，由进程创建的进程称为子进程，在仿真中进程被调用而表现的各种行为称为协议行为，进程对实际的网络仿真有着至关重要的意义。

关键概念 有限状态机是 OPNET 用来实现进程模型的方式。基于有限状态机机制，OPNET 才能以离散事件仿真的方式协议行为的交互。它类似数字电路中的卡诺图，可以有效直观描述复杂的一系列行为及它们之间的交互关系。OPNET 中的状态机主要用来描述网络协议，它实现的协议是整个仿真的基础。OPNET 大部分操作和仿真配置只不过是验证协议有效性和性能外部条件，实质上，要精确实现一个协议就是要做出一个好的状态转移图，这要求我们首先要认真分析好协议的本身，深入理解协议中主要做什么，分析协议的各种行为，然后将协议拆分为若干个事件（触动协议行为）及其对应的转移条件。这步是关键，因为不能期望划分的 FSM 实现的协议比我们理解的还要清晰。

6.3.2 子进程的初始化

根进程是在仿真开始由仿真核心创建的，它的初始化一般通过触发 Begin Simulation 型中断在创建的同时完成，这样看起来初始化工作是 OPNET 自动完成的，因此很容易认为子进程也该如此，其实不然。子进程被创建后即被挂起，它的初始化需要父进程对它进一步调用来完成，这时把子进程的初始化看作是一般的事件。与之类似，子进程不能接收所有来自仿真核心的中断，如 End Simulation 等，只能接收来自进程的中断。

6.3.3 仿真核心使用权的管理模式

在进程模块只有一个根进程的场合，针对该进程模块的中断毫无疑问都传给根进程，但是在仿真中进程模块包含多个处于激活状态的进程时，来自仿真核心的一个中断具体应该传给哪个进程呢？这时存在对仿真核心使用权的管理的问题。OPNET 提供四种管理模式：

（1）手动管理（Manual Steering）模式

首先根进程获取仿真核心控制权，然后经过一些处理后通过 op_pro_invoke 调用合适的子进程，这时暂时将控制权交给子进程，而将自己挂起，等待子进程将控制权交回。通常子进程运行到红色（Unforced）状态时停滞运行，自动释放仿真核心控制权，接着根进程继续执行完当前事件，如图 6-51 所示，这时相当于把子进程当作是一个函数来调用。

（2）常规管理（Normal Steering）模式

这种控制权的管理方式，通常和手动管理方式配合使用。前面提到手动管理模式下，子进程运行到红色（Unforced）状态时停滞运行，如果它为将来要做的某个任务调度自中断，当自中断触发时间到达，子进程主动获取仿真核心控制权。这时存在一个问题，当子进程独立完成了任务之后想通知父进程再分配其他任务时应该怎么办呢？OPNET 提供了一种特殊的中断，称为进程中断（OPC_INTRPT_PROCESS），可以实现任何进程间的调用，

子进程正是通过这种方式告诉父进程当前任务完成，请求下一个任务，如图 6-52 所示，当然父进程也可以完全不理会这个进程中断。

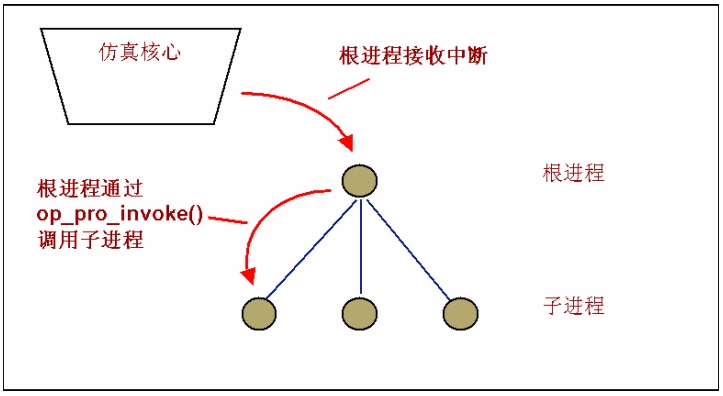


图 6-51　手动管理模式

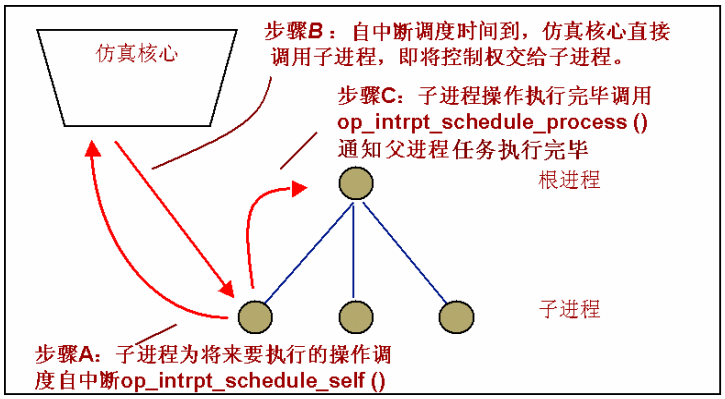


图 6-52　常规管理模式

（3）中断类型注册管理模式（Type-Based Steering）

每个进程模块可以接收任意类型的中断，有关中断的各种类型描述请参考 6.2.7 节的中断类核心函数。如果某个子进程专门用来处理某种特定类型的中断，例如，如图 6-49 所示，所有的 failure 类型的中断都由一个子进程处理，这时子进程可以先通过 op_intrpt_type_register 注册 failure 类型的中断，当中断到达进程模块时，如果是 failure 类型的中断则全部作用于相应的子进程，如图 6-53 所示。

（4）流端口注册管理模式（Type-Based Steering）

由于包成为 OPNET 最常用的通信工具，因此有必要针对因包传输而引起的流中断进行分类。由于每个流中断来自某个特定的输入流端口，这给流中断的分类带来了很大的方便，可以根据端口号进行识别。如果一个进程模块与多个输入包流相连，可能需要使用多个子进程来单独处理来自每个输入流的包，这时采用流端口注册管理模式来将流中断分发给合适的子进程，如图 6-54 所示。流端口注册除了针对流中断，对状态中断也适用，状态中断也可以看作是 1bit 的信息从一个进程模块流到另一个。

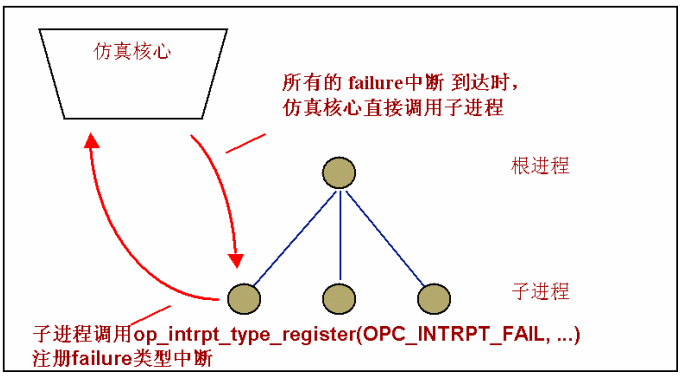


图 6-53　中断类型注册管理模式

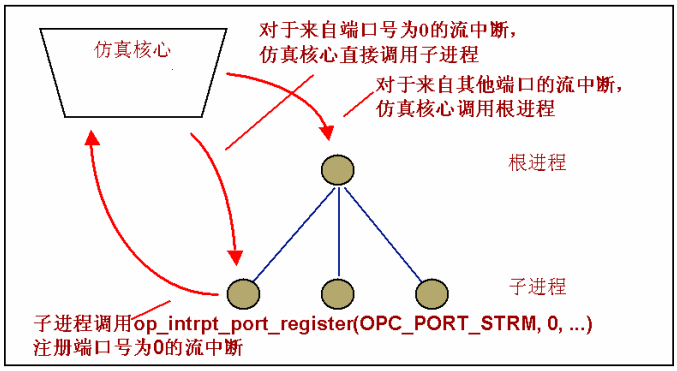


图 6-54　流端口注册管理模式

6.3.4　进程对仿真核心控制权获取方式的识别

如前所述，子进程获取仿真核心的控制权有“主动”（直接调用）和“被动”（间接调用）两种方式，例如常规管理模式、中断类型注册管理模式和流端口注册管理模式下子进程是“主动”获取控制权，如图 6-55 所示；而手动管理模式下是“被动”获取控制权，如图 6-55 所示。

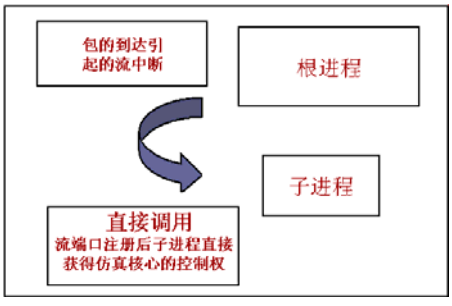


图 6-55　子进程“主动”获取控制权

在某种场合需要对两种方式进行区分，OPNET 提供两个常量来识别它们，分别是 OPC_PROINV_DIRECT 和 OPC_PROINV_INDIRECT。举例来说，子进程希望获得激活它的事件代码，这时需要根据调用模式区分对待，代码如下：

```
invoking_process=op_pro_invoker(op_pro_self(), &tnvoke_mode);
if(invoke_mode)==OPC_PROINV_DIRECT)
{
/*进程被直接调用，查看相应的事件代码*/
cur_event=(OspfT_Interface_Event)op_ubtrpt_code();
}
else
{
/*进程被进程通过 op_pro_invoke()调用，这时不能直接使用
op_intrpt_code()来获得事件代码，因为子进程不是直接被中断激活的，而是被你进程激活的，
但是可以访问 argument memory 获得*/
invoke_info_ptr=(OspfT_Interface-Invoke_Info*)op_pro_argmem_access();
cur_event=invoke_info_ptr->interface_event;
}
```

6.3.5　进程间的内存共享机制

为了支持进程间的协同运作，OPNET 提供三种参数传递的接口内存，它们分别是 module memory、parent-to-child memory 和 argument memory，它们作用范围依次减小，因此用在不同的场合。下面讨论三种进程间内存共享机制：

（1）Module 内存

Module 内存用得最普遍，所有隶属于某个进程模块的进程都能够使用，它的作用范围仅次于全局变量，如图 6-56 所示。

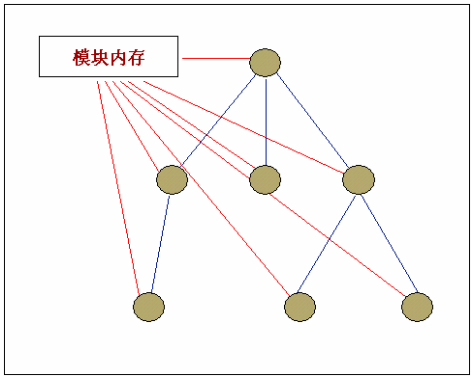


图 6-56　模块内存作用范围

（2）Parent-to-child 内存

parent-to-child 内存支持父进程传递数据给其子进程，值得注意的是，它只能在父进程创建子进程时和子进程句柄绑定一次，例如父进程要传给 someData 给子进程可以编写代码：aChildProc = op_pro_create ("childProcModel", someData);

在这以后，子进程都可以使用该内存。它的作用范围小于 module 内存，如图 6-57 所示。

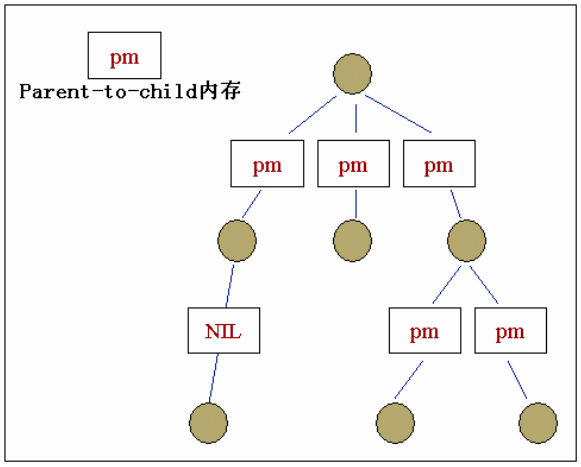


图 6-57　Parent-to-child 内存作用范围

（3）Argument 内存

Argument 内存共享基于每次中断的调用。当父进程调用子进程时，可以针对此次中断将特定的数据传给子进程。如果父进程想将 SomeData 传递给子进程可以编写代码：op_pro_invoke (aChildProc, SomeData);

它的作用范围小于 parent-to-child 内存，如图 6-58 所示。

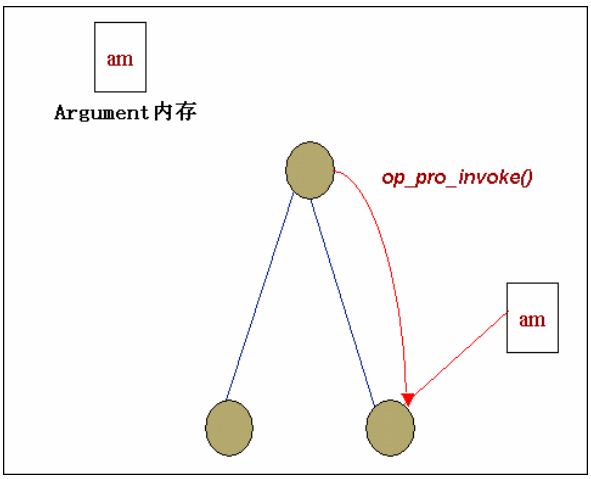


图 6-58　Argument 内存作用范围

6.3.6　使用子进程可能出现的几种错误

```
<<< Recoverable Error >>>
Process handle is for active process (31)
T (1210.5), EV (17314), MOD (top.earth_terminal.tdma), KP (op_pro_invoke)
```

出现进程回环，一个进程调用另一个正处于激活状态的进程，例如根进程调用子进程时，虽然将仿真核心的控制权暂时转交给子进程，它还是处于激活状态，这时如果子进程反过来调用根进程时就会出错。

```
<<< Recoverable Error >>>
Process handle is for remote process (44)
T (667.89), EV (19982), MOD (top.switch.switch), KP (op_pro_invoke)
```

进程调用其他进程模块的进程，例如一个模块的根进程调用其他模块的根进程。由于 OPNET 规定只能同一进程模块的进程才能够互相调用，所以以上操作将出错。

```
<<< Recoverable Error >>>
Process handle is for root process (16)
T (31.11), EV (5299), MOD (top.encryptor.encrypt), KP (op_pro_destroy)
```

子进程企图销毁根进程。子进程可以自己销毁自己，但是根进程不能够被销毁。

```
<<<Program Abort>>>
Unable to execute intrpt at process(4)
Process is already within an invocation.
T(0).EV(12).MOD(top.node_0.process)
```

某个进程 A 将仿真核心控制权转交给了另一个进程 B，而将自己暂时挂起，正常情况下是要等 B 停下来后把仿真核心控制权还给 A，但是在这之前 A 意外收到一个中断。

第 3 部分　OPNET Modeler 使用（高级篇）

第 7 章　OPNET 的调试

当我们把网络模型建立起来后，却运行出错，或者仿真结果不是所要，这些结构上或逻辑上的错误就需要用调试的方法去发现并解决问题，同时为了加强 OPNET 调试的功能，还可以与 VC 进行结合联调。

7.1　查看 OPNET 日志文件

OPNET 仿真结束后将产生两种日志文件，分别是仿真日志（DES log，Discrete Event Simulation log）和错误日志（Error log），查看它们可以得到一些有益提示从而更快地定位错误。

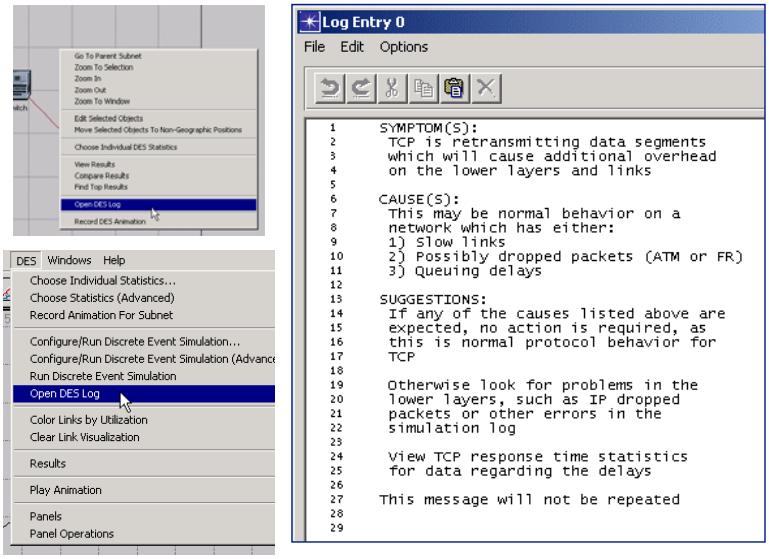


图 7-1　仿真日志文件

在工程空间上右点键可以打开仿真日志文件，如图 7-1 所示，它的内容是在仿真过程中由进程调用 OPNET 函数 op_prg_log_handle_create ()和 op_prg_log_entry_write ()写入的。OPNET 标准协议模块将协议可能发生的各种行为及其说明记录在该文件中，而不局限于出错时的提示，例如 TCP 协议仿真日志记录可能提示 TCP 重传引起额外底层的协议开销，

如果链路负载繁重或队列排队延时过大将引起大的重传延时，接着说明这种情况是正常的，不需做任何处理。

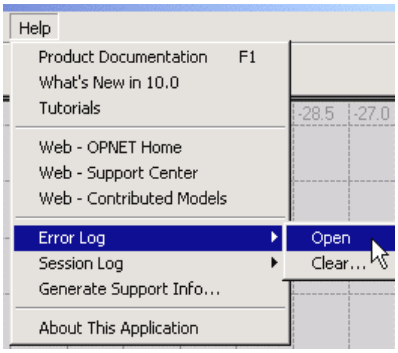


图 7-2　错误日志文件

在 Help 菜单下可以打开错误日志文件，如图 7-2 所示。错误日志文件以文本方式保存为<home>/op_admin/err_log，除了在菜单中打开也可以在 OPNET 控制台（console）窗口输入 op_vuerr 命令查看。 它包含了函数调用堆栈信息，我们可以从函数阶层性的调用关系中精确定位出错位置，

```
<<< Recoverable Error >>>          错误类型
  * Time:     20:02:54 Tue Jul 08 2003
  * Product:   modeler
  * Program:   op_runsim (Version 10.0.A PL0 Build 2269)
  * System:    Windows NT 5.0 Build 2195
  * Package: process (ip_dispatch) at module (top.switch 3 rsm.ip)
  * Function:  op_prg_mem_alloc (size)
  * Error:     Request to allocate (0) bytes failed.  出错原因
               T (0), EV (279), MOD (top. switch 3 rsm.ip),
 KP (op_prg_mem_alloc) 出错时间、事件、进程模块、出错函数
* Function call stack: (builds down)
-----------------------------------------------------
    Call  Block
   Count  Line#  Function
-----------------------------------------------------
 0)    1  1073741928  0x09254d0e [name not available]
 1)    1  1879048703  0x00004c00 [name not available]    可忽略的
 2)    1  -805306207  0x0000c400 [name not available]    内存消息

 3)    1    268  m3_main
 4)    1    914  sim_main                  仿真核心程序调用堆栈
 5)    1   2362  sim_ev_loop
```

```
 6)   280    522  sim_obj_qps_intrpt
 7)    30     15  ip_dispatch [wait exit execs]                出错进程
 8)    10    756  ip_dispatch_init_phase_2 ()                  出错状态
 9)    10    475  ip_rte_proto_intf_attr_objid_table_build
10)    12    599  ip_rte_support_intf_objid_add                出错函数堆栈
11)  3830    981  op_prg_mem_alloc (size)
---------------------------------------------------
```

以上是错误日志中的一条完整出错提示，有用的信息包括错误类型、出错原因、出错时间、错误事件、出错所在进程模块、函数调用堆栈以及最终出错函数。出错时，要获得准确的函数调用堆栈信息，在编写函数时必须使用FIN（function begin）、FOUT（function out）、FRET（function return）等界定函数范围的标识符，而且必须使它们配对。如果FIN后漏掉FOUT或FRET可能出现如下错误：

<<<Program Abort>>> Standard function stack imbalance

有时即使程序运行成功，也是个潜在的危机。尤其是使用外部文件中（external file）层层调用的子程序时，容易出现以上错误，而提示信息可能不会给出真正出错的位置（没有配对 FIN 和 FOUT 的子程序），给出的位置是由于函数堆栈不平衡不断积累最终导致内存泄漏时的程序（这个程序可能是真正有错误子程序的父亲、祖父…），造成找错困难，因此我们编写程序时切记使 FIN 和 FOUT/FRET 配对。

7.2 使用 OPNET Debugger 调试

7.2.1 ODB 调试概述

要产生 ODB 调试信息，必须将仿真核心类型设定为 development，优化的仿真核心（optimized）为了加快仿真速度不产生 ODB 调试信息。之后我们还需要在仿真属性中包含 debug 环境变量，如图 7-3 所示为 Modeler 9.0 以前版本启动 ODB 的方法，以后的版本直接在仿真属性界面上做简单的操作。

ODB 为控制和管理仿真行为提供一个交互式环境。ODB 支持断点（Breakpoint）定义，跟踪并显示仿真诊断信息。ODB 功能的实现有赖于进程模型中编写相应的程序支持，作为 ODB 指令激活调试状态（breakpoint、trace 和 action）的依据，可以在 ODB 窗口中输入 help <参数：all，basic，action，event，memeory，misc，object，packet，process，scripting，stop，trace>查看感兴趣的指令，如图 7-4 所示。

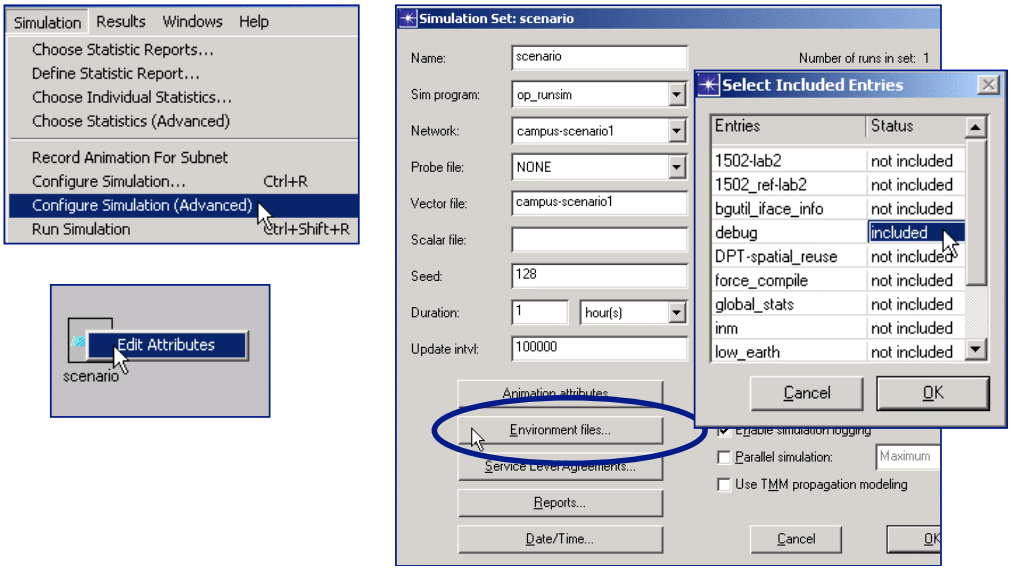


图 7-3　Modeler 9.0 以前版本中启动 ODB 调试的方法

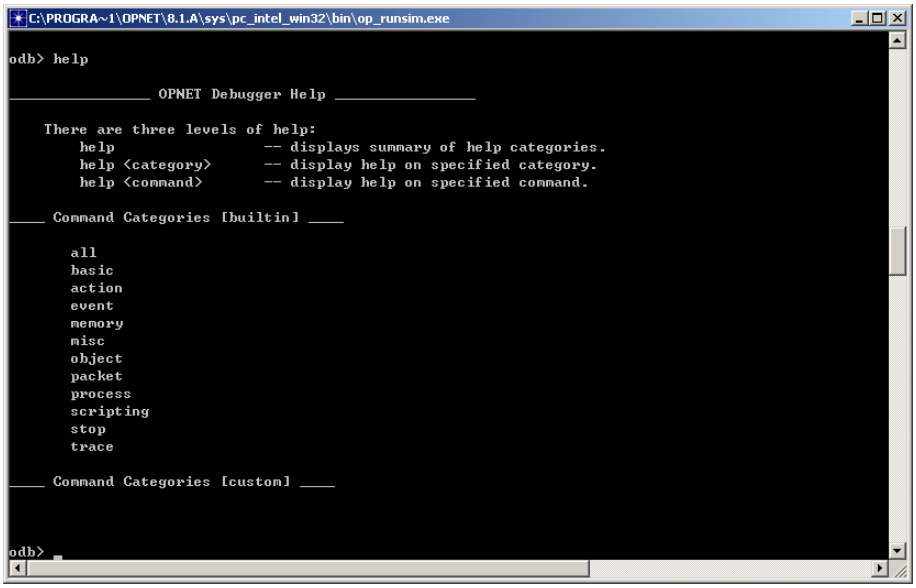


图 7-4　Help 指令带来的参数列表

ODB常用的指令分为basic，event，object，packet，stop，trace，process几类。Basic类指令主要包含了一些基本的操作；Event类指令主要针对事件进行操作；Object类指令主要针对各类对象（如节点，信道等）进行操作。Packet类指令处理所有与包相关的操作。它们的用法如表7-1所示。

表 7-1 ODB 常用指令名称及其功能描述

指令类型	指令名称	功 能 描 述
Basic 类	tstop	为与特定时间最接近的事件设置断点
	cont	继续事件运行直至下一个断点
	next	执行下面几个事件
	quit	退出程序
	status	显示用户当前所设的断点，跟踪信息。
	mstop	为特定进程模块设置断点
	delstop	取消断点设置
Event 类	evprint	打印事件信息
	evstop	在某个事件处设置断点
Object 类	attrget	获取某类的属性
	attrprint	打印目标的属性信息
	attrset	设置目标的属性值
	objassoc	打印与目标关联的信息
	objid	获取目标的 id
	objpkmap	打印由指定目标所拥有的包的列表
	objprint	打印目标的信息
	objmap	打印所指定类型的目标列表
Packet 类	iciprint_pk	打印与包关联的 ici 信息
	pkmap	打印指定的包的列表
	pkstop	为指定的包设置断点
	pttrace	跟踪所指定的包树
	pktrace	跟踪所指定的包
Trace 类	fulltrace	显示所有事件调用函数的情况
	ltrace	激活对某个标签的跟踪
	mtrace	针对某个指定模块，显示调用该模块所执行的程序
	deltrace	取消对某个标签的跟踪
Process 类	promap	打印进程模块当前包含的进程信息
	prodiag	执行隶属于某进程诊断块中的程序
	proldiag	将诊断块中的标签激活，并执行诊断块中的程序

有益提示 proldiag 带的参数有 2 个，分别是进程 ID 和标签（label）。它的效果等同于 3 条指令的叠加，首先 ltrace 激活标签；然后 prodiag 执行进程诊断块中的程序，并且打印标签被激活程序段的信息；最后，在执行完程序后 deltrace 取消对标签的跟踪。

有益提示 deltrace 取消对某个标签的跟踪，与激活标签不同的是，它带的参数为 trace_id，而不是标签本身，但是 trace_id 是系统分配的，不为我们所知，需要通过输入 status 指令查看。

7.2.2　针对结构错误（Structural Error）的 ODB 调试实例

我们编写完模块后一般直接运行仿真，有时候会出现 Program Abort 类型的错误，将使仿真中断，如图 7-5 所示，焦点放在出错事件上，我们得知是第 13 个事件出错，之后启动 ODB。

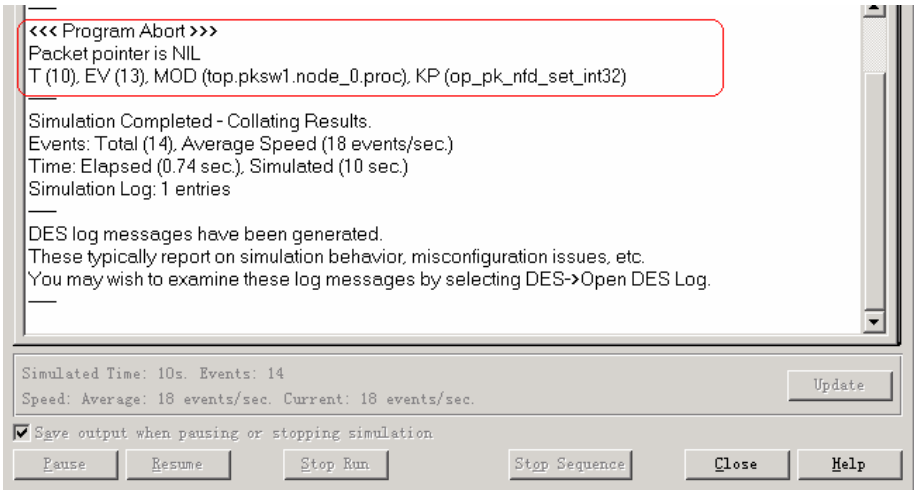


图 7-5　仿真完毕后的出错提示

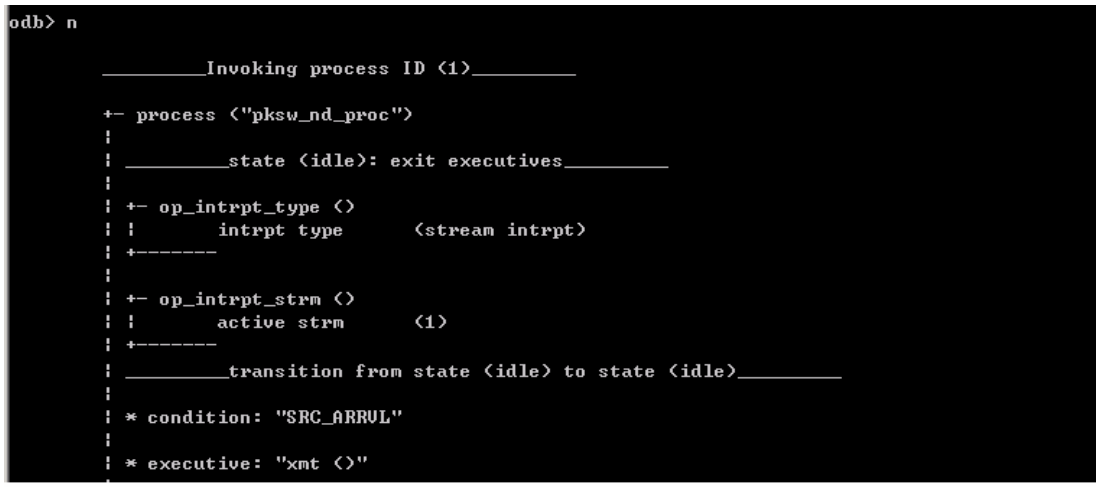
我们让仿真停止在第 13 个事件执行之前，如下所示，在事件栏中我们看到当前中断类型为流中断，另外还有两个 ID 号，其中 execution ID 为当前事件 ID，schedule ID 标明当前事件处在仿真核心事件列表中的位置，这两个值可能不相同，因为随着事件的增加和消减，仿真核心列表是不断变化的。Source 指明了当前中断源，例如这里表明中断是由全球网 top 下的 pksw1 子网下的 node_0 节点下的 src 进程模块在执行事件 9 时发出来的。Data 指明了与当前中断相关的信息，这里表明封包是从 1 号流端口接收来的，封包的 ID 为 0。Module 指明当前中断的接收方，top.pksw1.node_0.proc（processor）指明了物件的阶层关系和类型，OPNET 中最高层物件永远为 top，代表全球网。

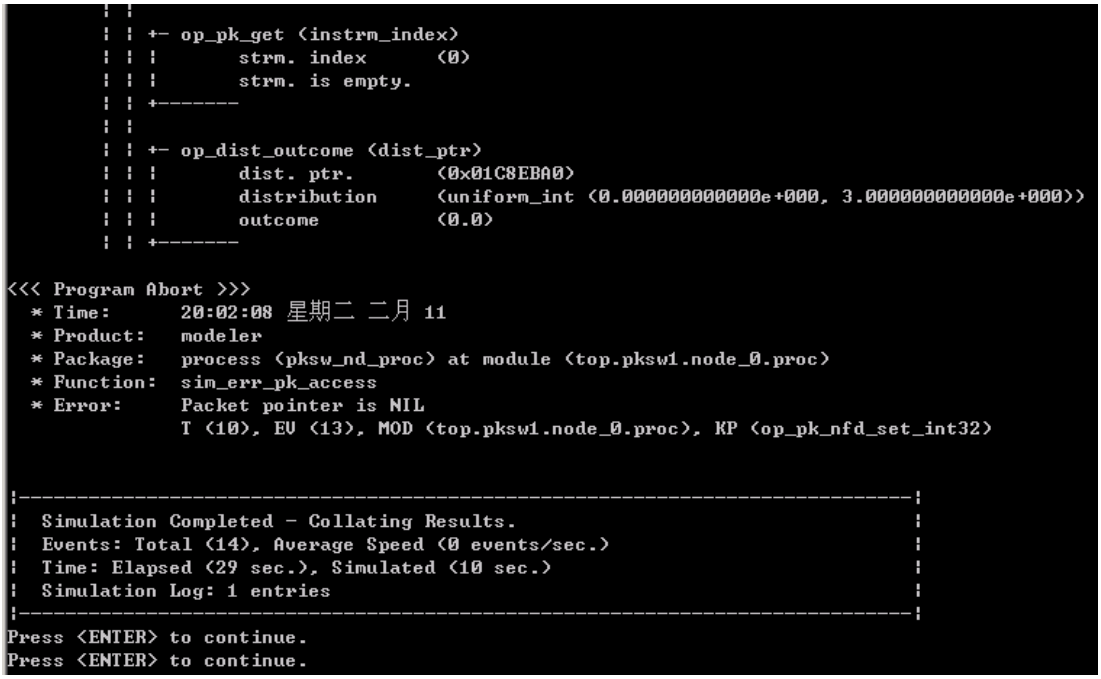
```
odb> evstop 13

odb> c


_____________________________ (ODB 10.0.A: Event) _____________________________

    * Time   :  10 sec, [00d 00h 00m 10s . 000ms 000us 000ns 000ps]
    * Event  :  execution ID (13), schedule ID (#13), type (stream intrpt)
    * Source :  execution ID (9), top.pksw1.node_0.src (processor)
    * Data   :  instrm (1), packet ID (0)
    > Module :  top.pksw1.node_0.proc (processor)

breakpoint trapped : "stop at event (13)"
```

接着将执行第 13 个事件，为了观察间中执行的代码，我们启动完全跟踪（fulltrace），之后输入 status 命令就可以查看已设定的中断和跟踪有哪些。

```
odb> fulltrace

odb> status

Breakpoints :
        None
Traces :
        Full trace is enabled

        Encapsulation trace is enabled
Actions :
        None
```

接下来输入 next 命令，让仿真执行下一个事件。从进程信息栏中我们可以看出当前进程（Invoking process）的 ID 号为 1，进程模型的名称为 pksw_nd_proc。进程收到中断后将执行红色 idle 状态的窗口执行代码（exit executives），首先判断中断的类型为流中断，接着获取流中断索引号，其值为 1。执行完之后满足条件 SRC_ARRVL，从 idle 状态再次转移到 idle 状态，同时执行条件子程序 xmt()，间中试图从 0 号流索引的包流中获取封包，这时我们已经看出一些端倪，应该是从 1 号流索引收包才对，果然指示包流中并没有数据包（strm. is empty），接着出现封包指针为空的错误提示，因此 op_pk_nfd_set_int32 代码肯定不能正常运行。到这里我们找到了错误所在，只要把 op_pk_get(0)改为 op_pk_get(1)就行了。





另外我们注意到最后两行的提示 Press (ENTER) to continue。这是在 edit->preference 中，将 console_exit_pause 设定为 TRUE 带来的结果，在 OPNET 9.0 后的版本也可以直接在仿真属性对话框中设定，如图 7-6 所示，这样就可以放心地浏览 ODB 调试历史信息，只有在两次单击回车键之后，才会退出 ODB 窗口。否则想在仿真运行结束后，听到“嘟”的一声

（提示）立即自动推出 ODB 显示界面。

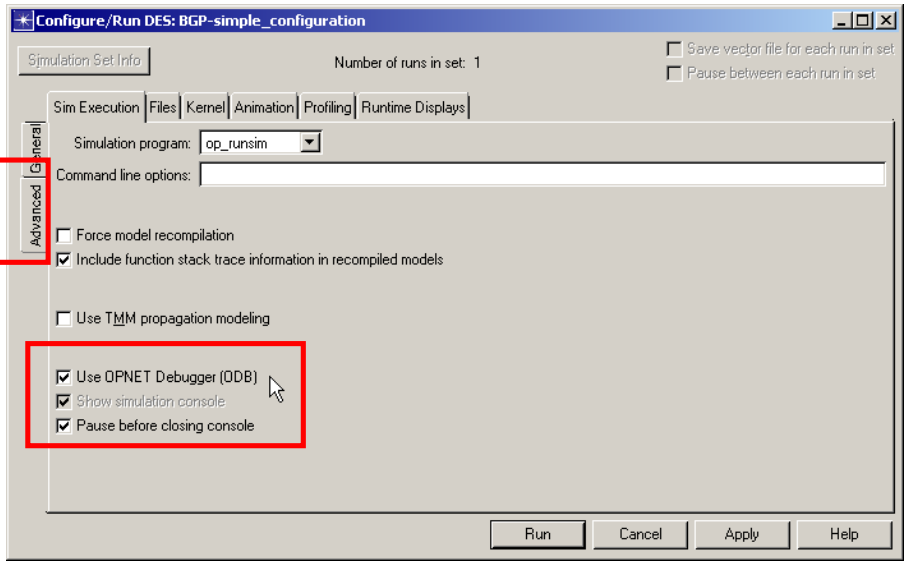


图 7-6　手动控制 ODB 窗口的退出

7.2.3　针对逻辑错误的 ODB 调试实例

有时仿真运行通过，而结果和想象的不一样，例如某个端口正常情况下应该收到数据包，吞吐量却为零。在这种没有错误提示的情况下我们应该如何发现逻辑上的错误呢。首先，启动 ODB 调试模式，我们试图能从封包从生成，传输，接收及最后销毁过程中找到原因。如图 7-7 所示，我们通过 pktrace 命令启动对 Packet ID 为 1 的封包的跟踪，pktrace 跟踪有关包的所有执行语句，直至包被销毁。在新版的 OPNET 中支持对包设置断点，任何涉及到处理封包的事件到达时仿真都会停下来，如图 xxx 所示，要推进事件必须再次输入继续仿真的命令（cont 或者 next），这样让我们清晰查看包在每个事件中的行为。

```
odb> pktrace 1

odb> pkstop 1

odb> c


_____________________________ (ODB 10.0.A: Event) _____________________________

    * Time   :  10 sec, [00d 00h 00m 10s . 000ms 000us 000ns 000ps]
    * Event  :  execution ID (10), schedule ID (#10), type (self intrpt)
    * Source :  execution ID (2), top.pksw1.node_1.src (processor)
    * Data   :  code (0)
    > Module :  top.pksw1.node_1.src (processor) [process id: 2]

breakpoint trapped : "stop at access of packet (1)"
```

图 7-7　包跟踪和设置包断点命令的组合

下面我们再次输入 cont 命令，包被 top.pksw1.node_1.src 进程模块创建，接着被 top.pksw1.node_1.proc 进程接收并将其 dest_address 域设定为 1，如下图方框所示，之后被传输到底层。

```
* Time   :  10 sec, [00d 00h 00m 10s . 000ms 000us 000ns 000ps]
* Event  :  execution ID (14), schedule ID (#15), type (stream intrpt)
* Source :  execution ID (10), top.pksw1.node_1.src (processor)
* Data   :  instrm (1), packet ID (1)
> Module :  top.pksw1.node_1.proc (processor)

   +- op_pk_get (instrm_index)
   ¦       strm. index      (1)
   ¦       packet ID        (1)
   +-------

   +- op_pk_nfd_set_int32 (pkptr, fd_name, value)
   ¦       packet ID        (1)
   ¦       field name       (dest_address)
   ¦       field type       (integer)
   ¦       field value      (1)
   ¦       field size       (32)
   +-------

   +- op_pk_send (pkptr, outstrm_index)
   ¦       packet ID        (1)
   ¦       stream index     (0)
   +-------
```

接着我们再连续输入 cont 命令，直到包被交换机节点进程（top.pksw1.hub.hub）接收，发现包的 dest_address 被修改为 0，如图 7-8 所示。我们仔细想一下交换机应该取出包的目的地址，根据目的地址再选择正确的路由，而不是去重新设置，这时我们已经找到了这个逻辑上的错误，只要将 op_pk_nfd_set_int32 改为 op_pk_nfd_get_int32 问题就解决了。

```
* Time   :  10.11 sec, [00d 00h 00m 10s . 110ms 000us 000ns 000ps]
* Event  :  execution ID (34), schedule ID (#38), type (stream intrpt)
* Source :  execution ID (28), top.pksw1.hub.rcv1 (pt-pt receiver)
* Data   :  instrm (1), packet ID (1)
> Module :  top.pksw1.hub.hub (processor)

   +- op_pk_get (instrm_index)
   ¦       strm. index      (1)
   ¦       packet ID        (1)
   +-------

   +- op_pk_nfd_set_int32 (pkptr, fd_name, value)
   ¦       packet ID        (1)
   ¦       field name       (dest_address)
   ¦       field type       (integer)
   ¦       field value      (0)
   ¦       field size       (32)
   +-------

   +- op_pk_send (pkptr, outstrm_index)
   ¦       packet ID        (1)
   ¦       stream index     (0)
   +-------
```

图 7-8　跟踪封包行为直至发现逻辑错误

7.2.4 针对进程模块的 ODB 调试

由于进程是实现整个仿真的基础，因此针对进程的调试是 ODB 调试的主要内容，主要包括四个部分，分别是定位进程、控制进程、跟踪进程及显示进程状态。

1．定位进程

（1）promap <Objid>显示指定进程模块包含的所有进程的 Process ID。

（2）promap all 显示所有仿真中存在的进程 Process ID，如图 7-9 所示。

关键概念 进程标记，用来更直观地区别同种类型进程（父进程产生多个进程，这些进程是由同一个进程模型派生而来的），使用这种方法必须在进程模块中通过 op_pro_tag_set(pro_handle,tag_string)来设置进程标记。

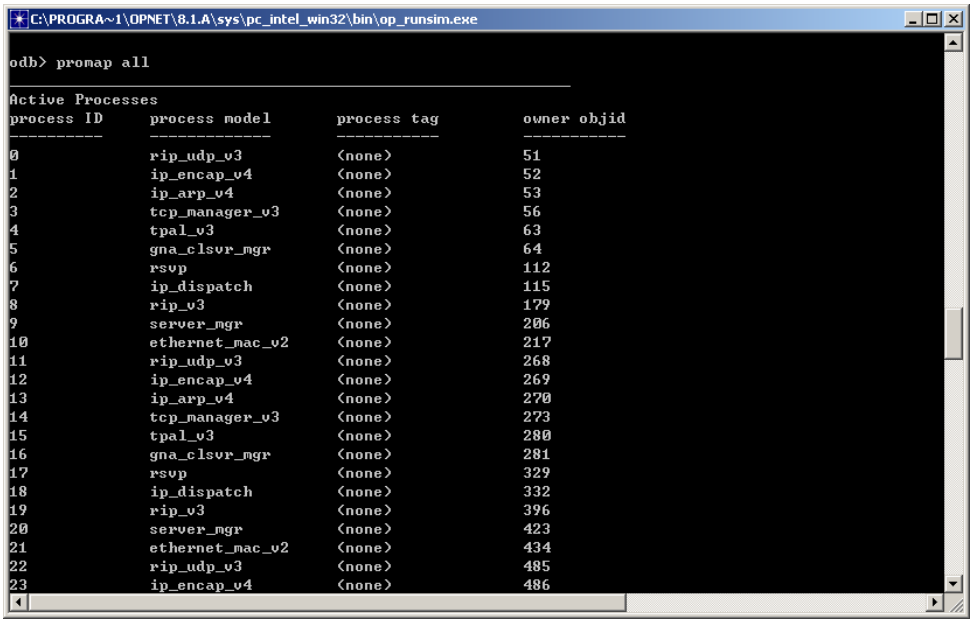


图 7-9 输入 promap all 指令的结果

2．控制进程

ODB 调试时输入指令 prostop <proc_id>为进程调用设置断点。无论什么中断，只要调用该进程，仿真都会暂时中断。

首先在进程模型中通过 op_prg_odb_bkpt(label)为指定位置设置断点，这个断点以标签（label）标识，ODB 调试过程中如果通过 prolstop <proc_id> <label>激活了该断点标签，则仿真会在被设置标签的位置中断。

3．跟踪进程

（1）protrace <proc_id>跟踪并显示调用指定进程的信息。

（2）ltrace <label>是最实用的调试指令，例如如果想找到程序中的逻辑错误，可以在可疑的程序段中加入这样一个语句。

```
if (op_prg_odb_ltrace_active("label")==OPC_TRUE){ printf(...)},
```

printf 打印需要观察的变量，在 ODB 中激活标签就可以看到仿真中这些变化的情况。

（3）proltrace <proc_id> <label>显示指定进程中设置过某个标签的位置所对应的信息。

4．显示进程状态

（1）在进程的诊断块（Diagnostic block）中编写与调试相关的程序后，ODB 调试时就能通过 prodiag <proc_id>执行诊断块对应的程序，从而显示调试信息。

（2）proldiag <proc_id> <label>指令相当于 prodiag <proc_id>与 ltrace <label>的叠加。

下面我们列举一个进程 ODB 调试的实例：

ODB>objmap proc dp

查看进程对象的信息，dp 为进程的名字，结果下图所示。

```
 Obj ID        Obj Name                                  Obj Type                Parent ID
-------------------------------------------------------------------------------------------
      2        top.Node.dp                               processor                       1
```

ODB> promap 2

查看进程模块当前被激活的进程信息，其中 2 为 Objid，如下图所示。

```
-----------------------------------------------------------
Active Processes for Module (2)
       process ID     process model       process tag
       ----------     -------------       -----------
       0              dynproc_root_lab3_ref(none)
```

ODB> prostop 0

为进程设置断点，其中 0 为 process id

ODB> protrace 0

激活进程跟踪信息显示, 其中 0 为 process id

ODB> status

显示指令状态，如下图所示。

```
Breakpoints :
         0)    stop at invocation of process (0)
Traces :
       Full trace is disabled
       Encapsulation trace is enabled
         0)    trace on process (0)
Actions :
       None
```

ODB> cont，继续执行事件直至仿真运行到断点位置，如下图所示。

```
   * Time   :  0 sec, [00d 00h 00m 00s . 000ms 000us 000ns 000ps]
   * Event  :  execution ID (0), schedule ID (#0), type (begin sim intrpt)
   * Source :  Simulation Kernel
   * Data   :  none
   > Module :  top.Node.dp (processor)

breakpoint trapped : "stop at invocation of process (0)"
```

ODB> next，执行下一个事件，如下图所示。

```
        _________Invoking process ID (0)_________

        +- process ("dynproc_root_lab3_ref")
        |
        | _________state (init): enter executives_________
        |
        | +- op_id_self ()
        | |       object ID        (2)
        | +-------
        |
        | +- op_topo_assoc_count (objid, direction, objmtype)
        | |       objid            (2)
        | |       assoc. dir.      (OPC_TOPO_ASSOC_IN)
        | |       assoc. type      (packet stream)
        | |       num assocs.      (3)
        | +-------
        |
        | +- op_prg_mem_alloc (size)
        | |       block size       (24)
        | |       mem. addr.       (0x01FA7D98)
        | +-------
```

ODB> promap 2

查看进程模块当前被激活的进程信息,其中 2 为 Objid，如下图所示。

```
------------------------------------------------------------
Active Processes for Module (2)
        process ID      process model       process tag
        ----------      -------------       -----------
        0               dynproc_root_lab3_ref(none)
        4               dynproc_child_lab3_ref(none)
        5               dynproc_child_lab3_ref(none)
        6               dynproc_child_lab3_ref(none)
```

ODB> prostop 6

ODB> cont，继续运行仿真，如下图所示。

```
    * Time   :  10 sec, [00d 00h 00m 10s . 000ms 000us 000ns 000ps]
    * Event  :  execution ID (9), schedule ID (#12), type (stream intrpt)
    * Source :  execution ID (6), top.Node.gen_2 (processor)
    * Data   :  instrm (2), packet ID (2)
    > Module :  top.Node.dp (processor)

breakpoint trapped : "stop at invocation of process (6)"
```

ODB> protrace 6

ODB> next

打印调用子进程 6 执行的语句，如下图所示。

```
        _________Invoking process ID (6)_________

        +- process ("dynproc_child_lab3_ref")
        |
        | _________state (init): enter executives_________
        |
        | +- op_id_self ()
        | |       object ID        (2)
        | +-------
        |
        | +- op_topo_assoc_count (objid, direction, objmtype)
        | |       objid            (2)
        | |       assoc. dir.      (OPC_TOPO_ASSOC_IN)
        | |       assoc. type      (packet stream)
        | |       num assocs.      (3)
        | +-------
        |
        | +- op_prg_mem_alloc (size)
        | |       block size       (12)
        | |       mem. addr.       (0x01FDC398)
        | +-------
        |
        | +- op_intrpt_strm ()
        | |       active strm      (2)
        | +-------
```

ODB> promap 2，如下图所示。

```
Active Processes for Module (2)
        process ID      process model       process tag
        ----------      -------------       -----------
        0               dynproc_root_lab3_ref(none)
        4               dynproc_child_lab3_refDynProc Child (stream 0)
        5               dynproc_child_lab3_refDynProc Child (stream 1)
        6               dynproc_child_lab3_refDynProc Child (stream 2)
```

查看进程模块当前被激活的进程信息，其中 2 为 Objid。这时出现进程标记的信息，因为子进程 dp_child 中的有设置进程标记的语句：

sprinf(tag_string,"DynProc Child (stream %d)", op_intrpt_strm());

根据包流的输入端口索引号制定进程标记 tag_string

op_pro_tag_set(op_pro_self(),tag_string);

为进程自身设置标签。

ODB> status，显示指令状态，如下图所示。

```
Breakpoints :
          0)    stop at invocation of process (0)
          1)    stop at invocation of process (6)
Traces :
        Full trace is disabled
        Encapsulation trace is enabled
          0)    trace on process (0)
          1)    trace on process (6)
Actions :
        None
```

ODB> delstop all

ODB> deltrace all

ODB> status，再次显示指令状态，发现断点已经被取消，如下图。

```
Breakpoints :
        None
Traces :
        Full trace is disabled
        Encapsulation trace is enabled
        None
Actions :
        None
```

ODB> prolstop 6 "dynproc_50"

这时激活进程 6 的断点标签，

当执行到进程中的 op_prg_odb_bkpt("dynproc_50")语句时程序中止运行。

ODB> cont，继续运行仿真，如下图所示。

```
    * Time   :  243.15348017 sec, [00d 00h 04m 03s . 153ms 480us 170ns 433ps]
    * Event  :  execution ID (459), schedule ID (#462), type (stream intrpt)
    * Source :  execution ID (458), top.Node.gen_2 (processor)
    * Data   :  instrm (2), packet ID (227)
    > Module :  top.Node.dp (processor)

breakpoint trapped : "stop at label (dynproc_50) in process (6)"
```

ODB> prodiag 6，运行进程诊断块的代码，如下图所示。

```
|-------------------------------------------------------------------------|
|  Module (2), (top.Node.dp)                                              |
|  From procedure: dynproc_child_status ()                                |
|  Packets handled: (0) 0  (1) 0  (2) 50                                  |
|-------------------------------------------------------------------------|
```

7.2.5 调整 ODB 窗口缓存大小

在进行 ODB 调试时，默认的缓存只能支持行数不多的输出 DOS 界面显示，如果要追踪大量的调试结果，则必须增加输出缓存。在出现的 ODB 窗口中的顶部激活蓝色标题栏，然后右击鼠标

这时会出现如图 7-10 所示的窗口。

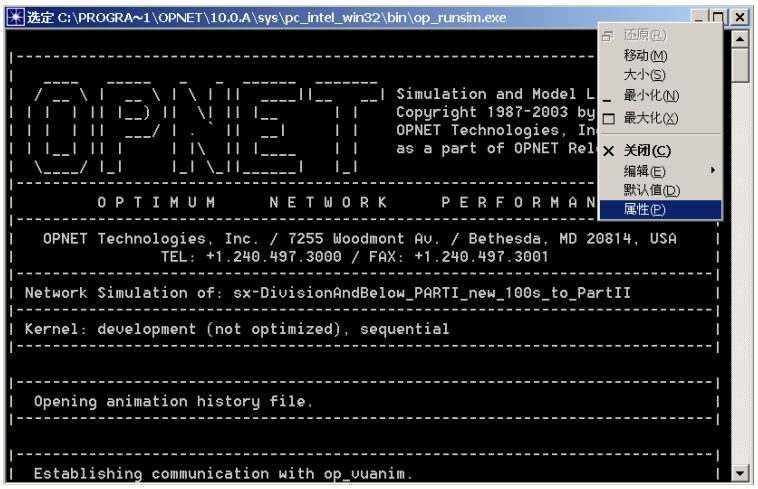


图 7-10 ODB 窗口

选择属性选项。单击布局选项卡。

这时会出现如图 7-11 所示的窗口。

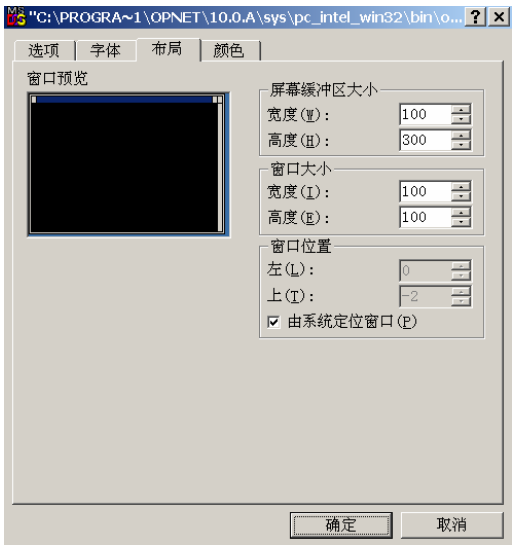


图 7-11 ODB 窗口属性对话框

设置合适的屏幕缓冲区的宽度和高度，以及自己喜欢的窗口宽度和高度。单击确定按钮。

这时会出现如图 7-12 所示的窗口。

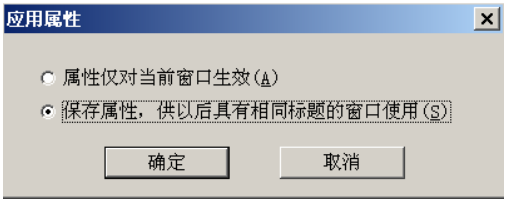


图 7-12　应用属性窗口

选择保存属性选项，单击确定按钮，以后的 ODB 窗口的规格将保持不变。

调整 ODB 窗口的大小，如果不小心调成全屏，则不能找到菜单，请按键盘上的 Windows 键退出全屏状态，再修改属性。

7.3　OPNET 与 Visual C++联合调试

ODB 调试功能很强大，它更侧重于逻辑上的调试，与 VC 调试相比，不足的是，即使是在 fulltrace 下也只能显示函数的调用情况和代码的返回值，而一些有关赋值、比较等代码却显示不出来，此外 ODB 只显示函数的原型和参数的结果。所以 ODB 调试一般用于全局错误定位，VC 一般用于局部精细地跟踪程序，查看变量的变化，或进入函数查看细节。

VC 提供一个非常直观、功能强大的调试环境，可以支持设置断点、观察变量、单步跟踪。追踪子程序等操作。本节主要介绍 OPNET 与 Visual C++联合调试需要进行一些参数配置。OPNET 与 VC 联调大致来说可以分为以下几个步骤：设定环境变量；设定 OPNET 参数；首次联调选择 OPNET 强制编译（force compile）；绑定（attach）OPNET 仿真进程；观察变量。

7.3.1　VC 的安装及环境变量的设置

在 OPNET 没有和 VC 结合之前，必须保证 VC 的安装及环境变量设置正确。

（1）正确安装 VC++，默认目录为（以下均以默认目录为例）

C:\Program Files\Microsoft Visual Studio

注意在安装过程会弹出是否需要注册环境变量（Register Environment Variable）的对话框，请“确定”按钮。

（2）在桌面“我的电脑”图标上单击鼠标右击，从弹出的菜单中选择“属性”。这时出现系统特性对话框，选择“高级”选项卡，然后单击“环境变量”。

（3）增加“用户变量”属性值，如表 7-2 所示，黑体部分为针对 OPNET 调试而需要设置的变量值，其中<opnet_dir>表示 OPNET 安装目录，<version_num>表示版本号。

表 7-2 环境变量配置表

变 量 名	变 量 值
include	C:\Program Files\Microsoft Visual Studi0\VC98\atl\include
	C:\Program Files\Microsoft Visual Studio\VC98\mfc\include
	C:\Program Files\Microsoft Visual Studio\VC98\include
	<opnet_dir>\<version_num>\sys\include
	<opnet_dir>\<version_num>\models\std\include
Lib	C:\Program Files\Microsoft Visual Studio\VC98\mfc\lib
	C:\Program Files\Microsoft Visual Studio\VC98\lib
	<opnet_dir>\<version_num>\sys\lib
	<opnet_dir>\<version_num>\sys\pc_intel_win32\lib
MSDevDir	C:\Program Files\Microsoft Visual Studio\Common\MSDev98
path	C:\Program Files\Microsoft Visual Studio\Common\Tools\WinNT
	C:\Program Files\Microsoft Visual Studio\Common\MSDev98\bin
	C:\Program Files\Microsoft Visual Studio\Common\Tools\ Program Files\Microsoft Visual Studio\VC98\bin
	<opnet_dir>\<version_num>\sys\pc_intel_win32\bin

有益提示 如果 Visual C++环境变量设置不正确，则 OPNET 将提示 comp_msvc 不能执行；如果 OPNET 环境变量设置不正确，将不能使用 Visual C++进行联合调试。

（4）单击“确定”按钮，退出设置。

7.3.2 修改 OPNET 有关与 VC 联合调试的属性

在项目编辑器中单击“Edit”选项，从弹出的菜单中选择 Preferences，如图 7-13 所示修改以下属性值。

（1）在 bind_shobj_flags（动态连接）和 bind_static_flags（静态连接）的值后面加上/DEBUG，这一步的作用是连接时将所有的目标（*.obj）文件集成为一个动态连接库（*.dll）文件，同时加入调试信息。

（2）在 comp_flags 和 comp_flags_cpp 后面加上/Zi /Od，这一步的作用是在编译时产生调试信息，并且在调试时关闭编译器的优化功能。

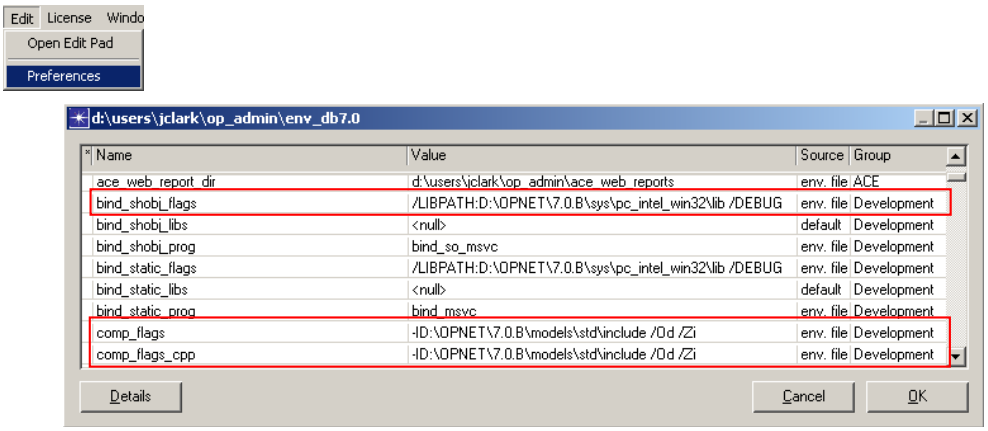


图 7-13 与调试相关属性的设置

7.3.3 OPNET 与 VC 联合调试的步骤

首先，设置仿真属性。在 Simulation 菜单中选择 Configure Simulation (Advanced)，按如图 7--21 所示设置然后选择 Envirement files，将其中 debug 的属性值变为 included，这使仿真处于编译模式，运行仿真将弹出 ODB 窗口；并且视情况将 force_compile 的属性值改变为 included，这将强制编译所有的进程模型。具体操作如图 7-14 所示。

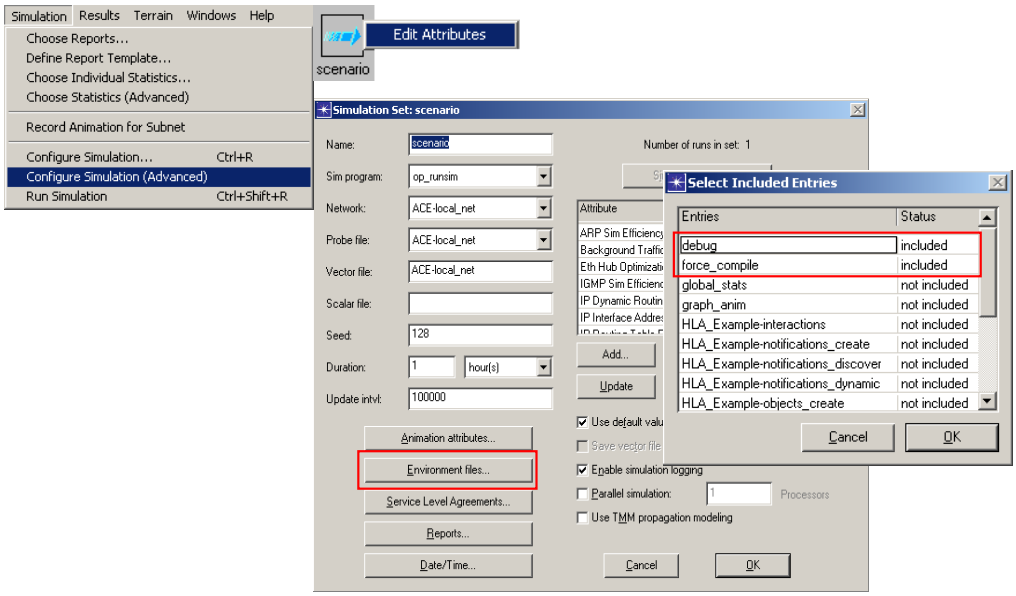


图 7-14　与调试相关的仿真属性

接下来需要选择用 VC 来调试 OPNET 进程，一般的方法是打开 VC，在 Debug 菜单下单击 attach to process，然后选择 op_runsim_dev.exe（或者 op_runsim_opt.exe），这样就可以

利用 VC 提供的调试功能进行 OPNET 程序调试。有时 attach to process 选项框为空，这可能不是 VC 和 OPNET 程序本身的问题，因为有些应用软件的进程和 op_runsim_dev.exe（或者 op_runsim_opt.exe）进程冲突，如一些杀毒软件，acrobat 等，或者是其他进程开得过多，这时可以通过以下方法解决这个问题：按住键盘的 Ctrl+Del+Alt 按键，选择任务管理器，找到 op_runsim_dev.exe 进程，在它上面单击鼠标右键，从弹出的菜单中选择“调试”，这时就会启动 VC 并且自动 attach OPNET。

用 VC 观察变量时，进程的状态变量不能直接观察到，必须通过引用指针 op_sv_ptr 来看，它指向了所有的状态变量。例如要观察状态变量 A，可以在查看变量窗口输入（*op_sv_ptr）.A。

7.4 常见错误及其说明

```
<<<Program Abort>>>
Invalid Memory Access
```

内存无效访问是调试程序中最常碰到的错误，一般是程序中的指针出了问题，可以尝试用以下办法来解决：

（1）按照 7.3.2 节所示，在 edit->preference 中给 comp_flags 加/Od /Zi 字段，给 bind_shobj_flags 加/DEBUG 字段；

（2）在 edit->preference 中找到 handle_exception，将 TRUE 改为 FALSE，这样程序中的异常就可以由 VC 来调试；

（3）运行仿真，如提示出现异常，点击 cancel，则自动打开 VC，并且 debug 会停留在发生异常的指针处。

但是有时候 VC 的 debug 有可能停留在让人看不懂的汇编语言处，此时可以观察发生错误的事件 event_id 值，在 ODB 调试的时候使用 evstop 指令设置断点，让程序中断在出错的事件前，接下来可以采取下列两种方法之一：

（1）按照 7.2.3 节所示，采用 OPNET 与 VC 联合调试，在 VC 中通过单步执行查看；

（2）用 ODB 的 next 指令一个一个时间看，往往配合 ltrace 和 fulltrace 指令查看程序运行状况。

另外，当我们成功地完成一个简单场景的仿真时，往往会有如释重负的感觉，因为虽然是简单场景但包含了所需要仿真的所有内容。如果需要扩展系统，剩下的工作看似只要直接拷贝粘贴相同类型的节点，再次运行仿真就行了。但是可能在运行中间出现无效内存访问(Invalid Memory Access)的错误，这是最难发现的内存错误，必需要有心里准备花大量

的时间来找到这些错误，因为它们是隐藏的，所以是系统稳健性不强的潜在危机。这类错误通常是由于内存读写越界引起的，因为内存的状态不断变化，所以导致出错的位置不断变化，而且当前仿真出错的位置未必是错误的症结所在，必需耐心地查看每个模块初始化时的内存分配方式。碰到这个问题我们也可以缩短仿真时间，在确保仿真能够完成前提下，在仿真属性中选择 Advanced->Profiling->Collect detailed profiling for function，之后出现每一个内存使用的统计，如图 7-15 所示，我们可能从中获得一些启示了，例如哪个函数使用最多的内存，哪个函数编写的不够好。我们也可以在 utilities 物件拼盘下找到一个称为 Memory Usage 的物件，它可以列出在一定时间内存的使用是多少，从而看出有无不正常的增加内存使用率。

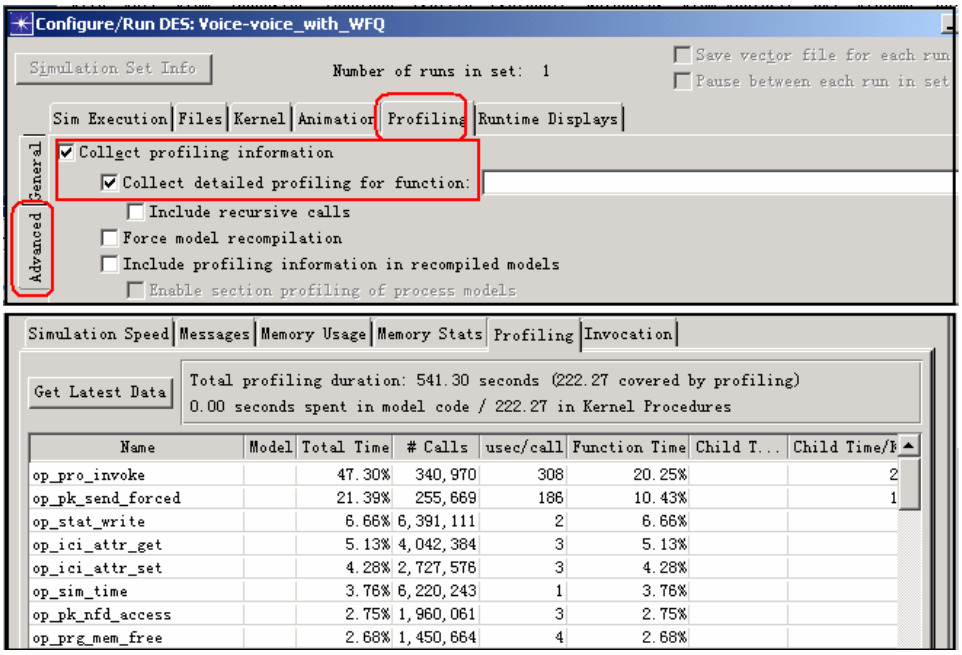


图 7-15　仿真完成后显示的函数被调用信息

```
<<< Recoverable Error >>>
Object repository construction failed
due to errors encountered by the binder program (bind_so_msvc)
-----
<<< Program Abort >>>
Error encountered rebuilding repository -- unable to proceed
```

出现这个错误一般是编译连接出错 unresolved external symbol，常见的有以下可能性：

（1）Pipeline Stage (C code)文件名与函数名不一样，这时改为同名就行了。

（2）进程模型用到一个无法定位的外部函数，这时在进程模型编辑器中选择

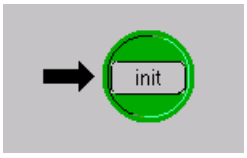
File->Declare External Files…，然后选中含有该外部函数的外部文件。

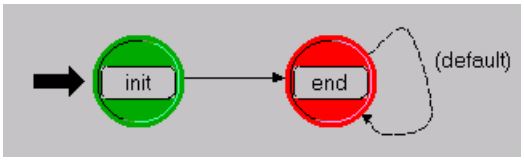
（3）外部文件用到一个无法定位的函数，这时查看是否漏掉 include 需要用到的头文件。

```
<<<Program Abort>>>
No true transitions from state ()
T(), EV(), MOD(), PROC (sim_pro_err_transit)
```

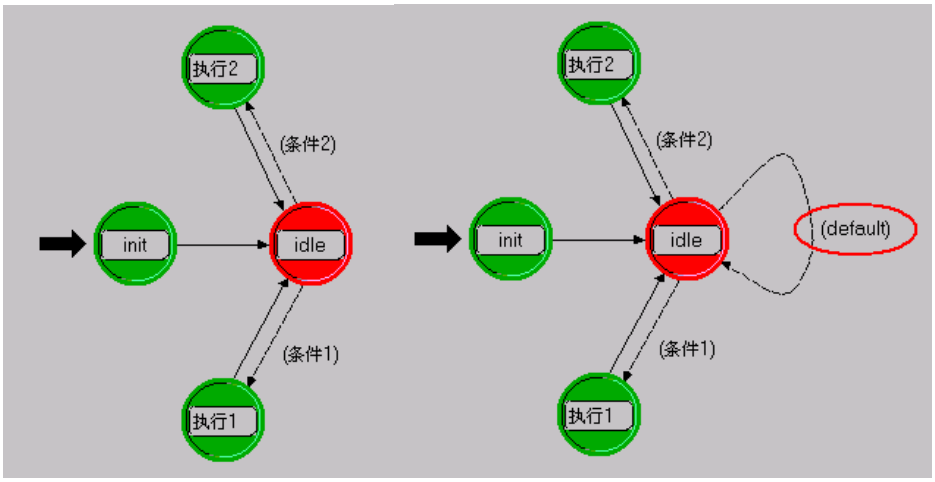
状态程序执行完毕找不到出口，有限状态机要能够正常运行要求在任意条件下，每个状态执行完毕之后都必须发生转移，出错的情况有以下两种：

（1）根本没有转移条件及相应的状态，如下图所示，这时需要增加一个红色的状态。





（2）如下图所示，当进程处于 idle 状态并被调用时，条件 1 和条件 2 都不满足，这时需要增加一个默认的转移条件 default。




```
Unable to write file (*.pr.m) compilation failed
Source code file couldn't be generated
```

查看模型文件属性是否设置为只读，编译模型文件需要刷新其内容，去掉只读属性。

第 8 章　业 务 建 模

业务建模的准确性是任何通信系统性能评估的关键所在。如果想要获得对实际网络设计有指导意义的结果，那么用于仿真的业务源必须能够正确反映实际业务的统计特性，因为 Modeler 就像是一个黑箱，如果加载不精确的业务，出来的结果也是不精确的。因此我们希望能够尽量确保业务的精确度，采用离散事件仿真机制可以达到这个要求，它使每一个封包都经历整个协议栈，对包的封装、路由都模拟出来。OPNET 精确业务建模可以在 3 个协议阶层实现，如图 8-1 所示，分别为：（1）封包的精确模拟从应用层开始；（2）封包的精确模拟从网络层开始，其中 RPG 模块为自相似业务源，它与 IP 层有良好的接口；（3）封包的精确模拟从底层开始。

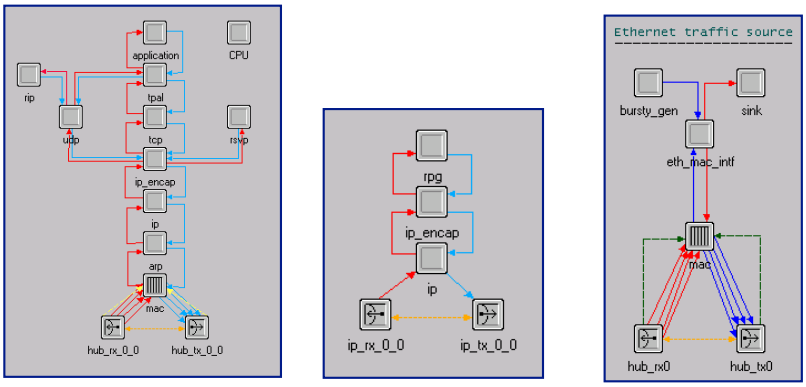


图 8-1　三种精确业务建模方式

对于从应用层开始的精确建模，OPNET 提供一些配置端对端业务和自定义多端业务的方法，它们将分别在 8.2 和 8.3 节描述。对于数据量非常大的业务，如话音和视频，如果采用精确业务模拟的方法会使整个仿真变得非常慢，如果每个封包都从应用层发送的话，往往仿真一两个话音流可能需要 30 分钟左右。因此在 8.4 节我们采取一种流分析的方式加快仿真速度，但是它的精确度有所削减。由于业务精确度（粒度，granularity，Level of Detail）和仿真速度的固有矛盾，因此在 8.5 节我们同时使用多种业务建模技巧使仿真速度和精确度尽可能可能同时达到。

8.1　ON/OFF 业务建模

针对简单的业务，OPNET 中提供了 ON/OFF 的建模机制，它难以逼近真实的网络业务，只作为一种数学上的分析模型。

图 8-2 数据包生成时间图

在 ON 期间生成数据包，如图 8-2 所示，ON 期间的黑竖线代表生成一个数据包，每个包大小可以按照某种分布函数来确定，两条黑竖线间的时间间隔代表包到达间隔。为了使在 ON 期间包到达服从泊松过程，包间隔可以由指数函数来确定。OPNET 业务生成模块的业务输入参数如表 8-1 所示。

表 8-1 业务输入参数

业务参数	取值举例
业务开始时间(s)	Constant(0)
业务结束时间(s)	Constant(1000)
ON 的持续时间 (s)	Exponential(10)
OFF 的持续时间(s)	Exponential(5)
包长度(bits)	Exponential(1024)
包到达间隔(s)	Exponential(0.1)

基于表 8-1，平均数据总负载可由以下公式计算：

$$\text{平均数据总负载} = (\text{业务结束时间} - \text{业务开始时间}) \times \frac{\text{平均ON的持续时间}}{\text{平均ON的持续时间} + \text{平均OFF的持续时间}} \times \frac{\text{包长度均值}}{\text{包到达间隔均值}}$$

8.2 配置标准端对端业务

端对端业务也可称为客户<->服务器模型下的业务，OPNET 已经为我们编写了一些常用应用的协议，并且将这些应用模块化，只要按照既定的流程设定一些参数就可以配置这类标准的业务，一般分为 4 个步骤：（1）定义应用；（2）设定业务主询；（3）配置服务器支持的应用；（4）设定客户端业务主询。本节对这 4 个步骤分别做说明。

8.2.1 设定应用参数

应用（Application）具体描述应用的动作，比如说 http 应用，规定了每次取得页面的大小和时间间隔；对于 ftp 应用，规定上传和下载的流量，文件的大小和产生的事件间隔。首先我们需要使用应用配置物件设定这些应用的参数，如图 8-3 所示，应用配置物件处在

特殊物件拼盘（utilities）中。

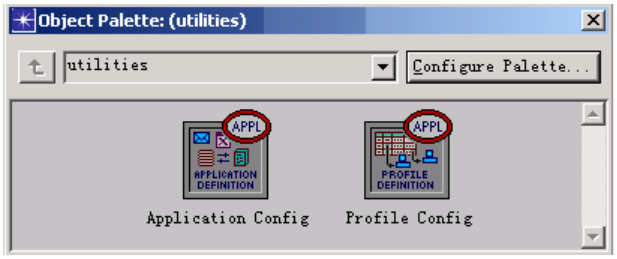


图 8-3　在 utilities 物件拼盘中找到应用配置物件

打开应用配置器物件的属性对话框，我们可以看到 OPNET 已经为我们定义了 9 种应用，如图 8-4 所示。

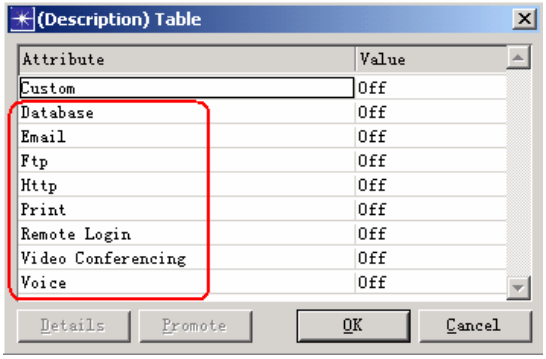


图 8-4　OPNET 自带的标准应用

一般的所需的应用都可以在这 9 种应用中找到，接下来我们需要具体配置业务参数。我们以 Database 应用为例做详细的说明，如图 8-5 所示。

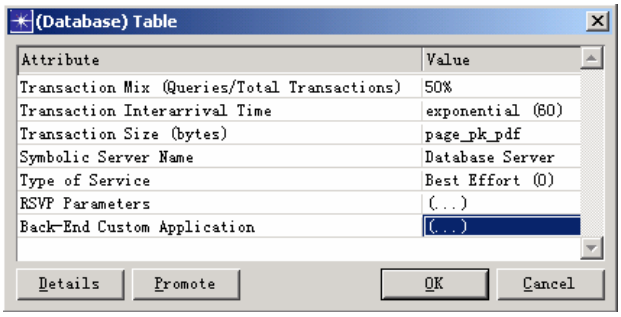


图 8-5　数据库应用参数的设定

Transaction Mix 代表查询数据量占整个输入交易量的比例，一般是查询一半，其他交易占一半；

Transaction Interarrival Time 指定间隔时间，比如多长时间进行一次交易或查询；

Transaction Size 指定交易数据包的大小；

Symbolic Server Name 代表象征性的服务器名字；

Type of Service 用来模拟 QoS，根据业务对服务质量的要求分成 0 至 7 八个不同的等级，如图 8-6 所示。其中 Best Effort (0)尽力而为业务优先等级最低，Reserved (7) 优先级最高，在网络带宽不能同时保证不同优先级业务带宽需求时，将牺牲优先级低的业务以保证高优先级业务的低丢弃率。

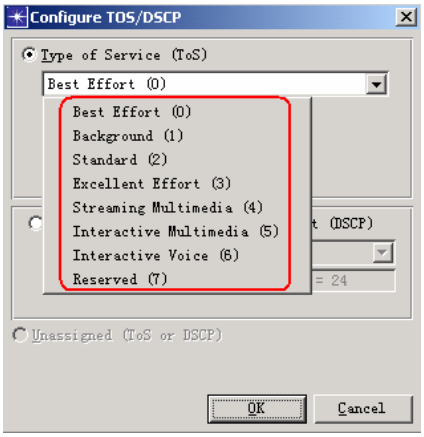


图 8-6　业务 ToS 的设定

除了以上传统方法设定 QoS 外，OPNET 还可以选择另一种具有良好扩展性的 QoS 的解决方案——区分服务（DiffServ），它基于 IP 流分类聚合，区别对待不同等级的聚合流，根据包头的 DSCP 选择提供特定质量的调度转发服务，其外特性称为逐点行为（PHB，Per-Hop-Behavior）。如图 8-7 所示，OPNET 中可供选择的的 PHB 有加速型 EF（Expedited Forwarding）、确保型 AF（Assured Forwarding）、尽力而为型 BF（Best Effort Forwarding）等。

RSVP Parameters 用来启动资源预留协议；

Back-End Custom Application 设定 CPU 背景业务，例如在处理数据库业务交易同时，CPU 可能还需要作一些其他与业务相关的处理，这部分通过加载背景业务的方式来模拟。尤其是针对服务器，比如想模拟如果 CPU 有额外负担时，是否仍然能够承受负载的业务。

8.2.2　设定业务主询

业务主询（或称为业务规格），描述一类用户群所涉及的应用，因此包含了多个应用，例如校园用户上网所涉及的业务主要是 ftp、Email、http 等，而行政单位用户可能用到数据库（database）、远程登录（remote login）、Email、ftp 和 http 等业务；同时业务主询也描述用户应用的行为，如用户什么时候开始使用某种应用，持续多久。例如校园网用户不限制 ftp 流量，而政府机关出于安全性的考虑需要限制 ftp 流量。

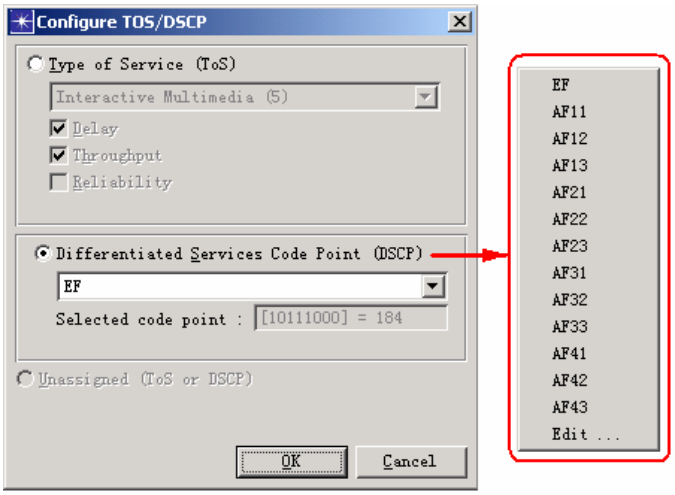


图 8-7　区分服务服务等级的设定

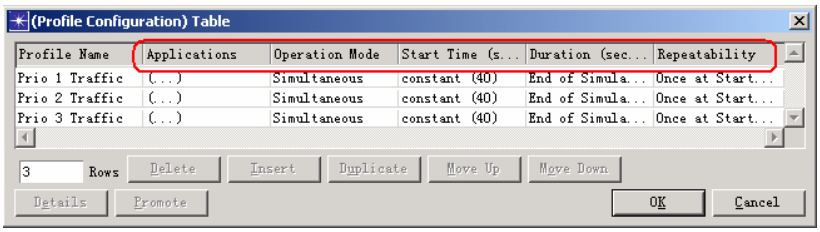


图 8-8　业务主询配置表

图 8-8 为业务主询的配置表，下面我们对每个属性分别做说明。

（1）Start Time 设定业务主询的开始时间，通常会设定成 100s，作为初始化预留时间，因为仿真开始时，所有的模块都需要一段时间初始化。例如 IP OSPF（开放最短路径优先）和 RIP（路由信息协议）可能需要上百秒时间来建立路由表，在此之前传输数据包，有可能会被路由器丢弃，因此所有业务必需在协议初始化完毕时才开始加载。另外值得注意的是，当仿真场景配置多种业务主询时，通常需要把它们间隔开来，比如开始时间设定为在 100－110s 之间随机选择一个时间点，这样不同的业务主询开始时间会稍微错开，否则如果整个业务都卡在 100s 一个时间点上，仿真可能不会运行得很顺畅；

（2）Duration 业务主询加载多长时间，通常设定为仿真结束才终止（end of simulation）；

（3）Repeatibity 指业务主询重复性设置，一般整个仿真就加载一次(once at start time)。如果设定 Number of Repetitions 大于 0，则重复多次，这时需要设置每次重复间隔时间 Inter-repetition Time，及重复模式 Repetion Pattern，如图 8-9 所示。

其中 Serial 为首尾相连模式，前一个业务主询加载完毕并跨过重复间隔时间才开始下一个，值得注意的是如果业务主询得持续时间（Duration）设定为 end of simulation，则不会重复，因为一次主询结束仿真也结束了。

另外一种重复模式称为 Concurrent，可能前一个业务主询并没有结束，只是机械性地

隔一段时间（Inter-repetition Time）又开始一个。

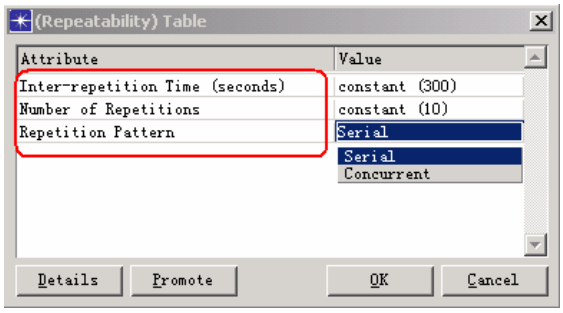


图 8-9　业务主询重复性的设定

（4）Operation Mode 指应用运行模式，如图 8-10 所示，它只针对业务主询中包含两种或两种以上应用的情况，需要根据实际情况来选择运行模式。举例来说，如果业务主询包含 4 种应用，分别是 HTTP、FTP、Telnet 和 Email，实际中，可能同时上网，下载 FTP 文件，上 BBS，收 Email，这时我们应该设置操作模式为同时加载模式（Simultaneous），当然也可以顺序执行业务（Serial），可能我们希望一个业务完成才加载后续的任务。具体选择何种方式要看如何才能模拟更真实的业务。

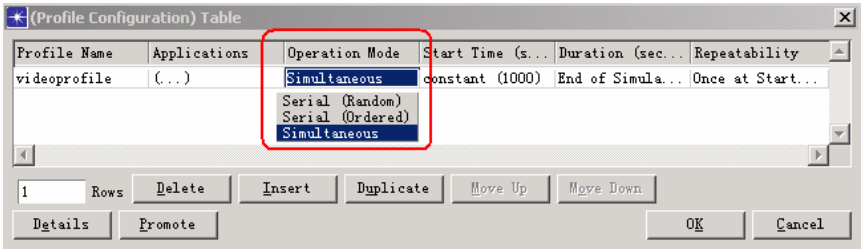


图 8-10　业务主询的运行模式

（5）Applications 用来圈定业务主询包含的应用，如图 8-11 所示。

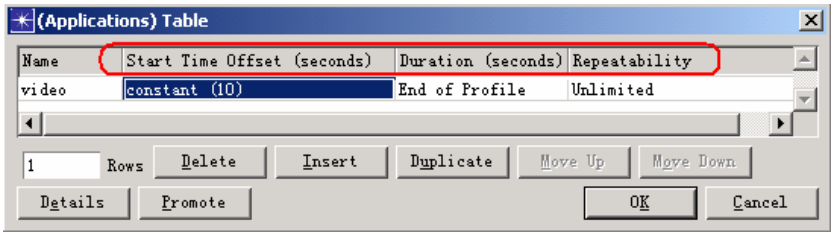


图 8-11　配置业务主询中的应用

Start Time Offset 代表应用的开始时间，注意不要和业务主询开始时间混淆，业务主询开始时间加上应用本身的开始时间才是业务真正的开始时间，例如业务主询开始时间为 100s，其中的某个应用开始时间为 5s，则该应用真正开始时间为 105s；

Druation 代表应用持续时间，可以设定业务主询时间多长应用也多长（End of Profile），

或者规定时间；

Repeatability 设定应用的重复性，如图 8-12 所示。它的机理与业务主询重复性类似，这里举例说明应用重复间隔和重复模式（Repetition Pattern）的作用。比如工程部业务主询包括 4 种业务，分别是 Database，Email，FTP，Webbrowsing，假如我们模拟从早上 9 点到下午 5 点这些业务的真实运作，可能 Database 在开机时就一直运行，Email 每隔 5 分钟收一次，FTP 可能 1 个小时下载一次，也可能同时开多个线程下载，Webbrowsing 也是隔一段时间看一次，因为上班时间不可能一直上网，我们也可以同时打开多个 IE 浏览器上网。之后一个星期业务就可以重复这一天的业务情形。一方面我们要将业务主询的操作模式设定为 Simultaneous，因为这些应用是独立并同时运行的；另一方面，针对每种业务，我们也必须根据真实情况设定它的重复模式，每次重复的间隔，例如上述的 Email 重复间隔为 5 分钟，重复模式为 Serial；FTP 重复间隔为 1 小时，重复模式设定为 Concurrent 来模拟多个 FTP 线程。

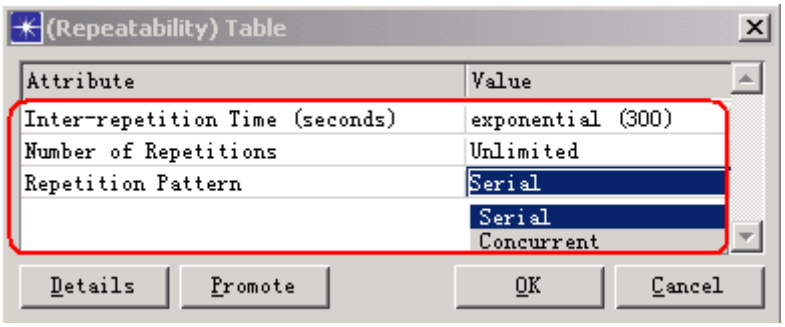


图 8-12　应用重复性的设定

8.2.3　配置服务器支持的应用

配置好应用参数和业务主询后，接下来就可以为服务器设定所支持的服务和应用有哪些，如图 8-13 所示，只要在应用配置物件中定义的业务都可以选。

一台服务器可以同时支持多种业务，例如一台服务器既可以作为 FTP server，也可以作为 HTTP server。应用个数可以直接在 Rows 编辑栏中输入，如图 8-14 所示。

当选定好业务之后，如图 8-14 所示，点击服务器说明栏（Description）配置有关服务器对应用支持的参数。

Processing speed 代表服务器处理交易的速度，具体设定多大的值要看达到的业务交易量有多大，

Overhead 为 CPU 处理开销。这两个参数用来粗略地计算服务器 CPU 的表现，因为实际中业务最终的端对端性能不光是网络本身的问题，还要看服务器的性能。

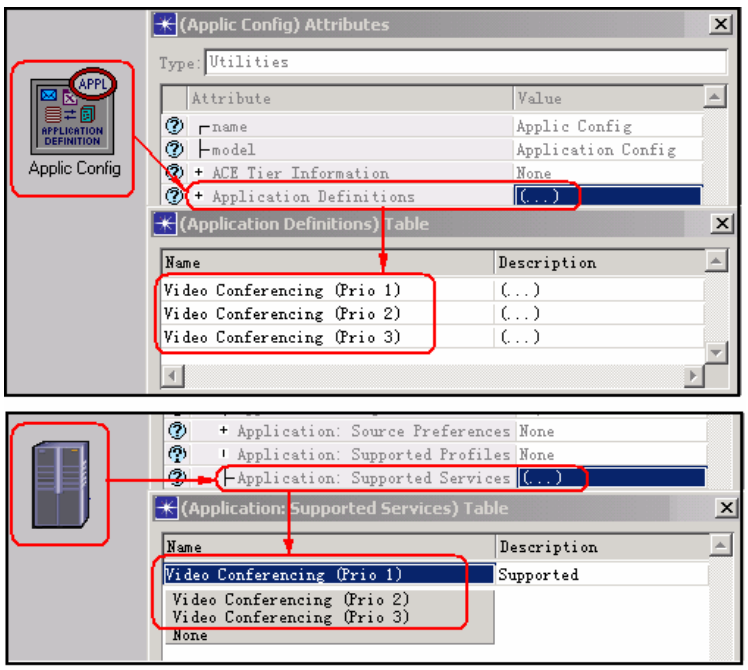


图 8-13　配置服务器支持的应用

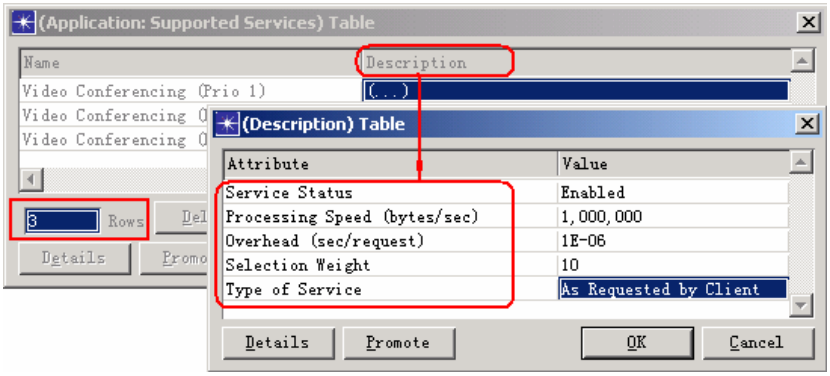


图 8-14　有关服务器对应用支持的参数

Selection Weight 代表服务器被客户选中的比重，这是个非常重要的设置，当多个服务器同时支持某种业务时，客户具体向哪台服务器索取服务就根据这个比重值随机选取。它不是绝对值，而是相对值，一般预设值为 10，如果两台服务器都设定为 10，则它们被选中的概率都是 50%，如果两台服务器都设定为 20，则结果还是一样。

Type of Service 指定服务器能够提供的 TOS，一般的设定为满足客户所需求的值（As Requested by Client）

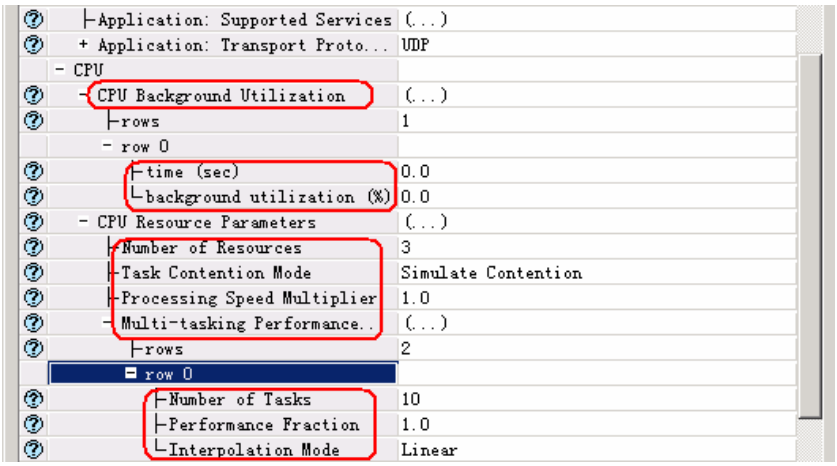


图 8-15　有关 CPU 参数的设定

除了在图 8-14 中有对 CPU 粗略模拟的两个参数外，还可以一个专门设定 CPU 参数的属性栏，如图 8-15 所示。

CPU Background Utilization 模拟 CPU 背景使用率，通常来说，我们配置的业务是网络中传输的业务，而这里模拟的背景业务是本地业务，不需要消耗网络资源，只是单机运行的业务。其中 time 为背景业务的开始时间，background untilization 为 CPU 被占用率。假如我们想模拟 80%CPU 被本地业务占用后，剩下 20%CPU 处理能力能否支持网络的应用。另外我们也可以通过这种方法快速模拟出服务器瓶颈。

CPU Resource Parameters 可以设定 CPU 的个数（Number of Resources），CPU 跳频倍率（Processing Speed Multipler），另外多 CPU 还需要多线程支持，表现方式和应用方式要匹配，这点非常重要，在 Multi-tasking Performace 必须相应设置多线程处理方式，否则多 CPU 的配置效果发挥不出来。Task Contention Mode 决定是否模拟 CPU 处理竞争的所引入的延时。

8.2.4　设定客户端业务主询

与服务器配置应用相对应，客户端则需要设定业务主询。可供客户端选择的业务主询的种类和在业务主询配置器中的设定完全吻合，如图 8-16 所示。

同时客户端可以配置多种业务主询，例如一个客户，他既可以是工程部的人，同时也可能是业务部的人，因此他需要同时配置这两个部门所对应的业务主询。

配置业务主询后，客户端可能还要设定 Application: Destination Preference，它表明从哪些服务器上取得特定业务的服务，其中 Symbolic Name 对应 Application Config 中针对某种应用所定义的象征性名称，如图 8-17 所示，客户端 Symbolic Name 如果设定为“Video Destination”，则意味着它想得到 Video Conferencing 这种应用服务，因为在 Applcation Config

中业务定义表中可以找到与“Video Destination”关联的应用为 Video Conferencing，如图 8-17 所示。

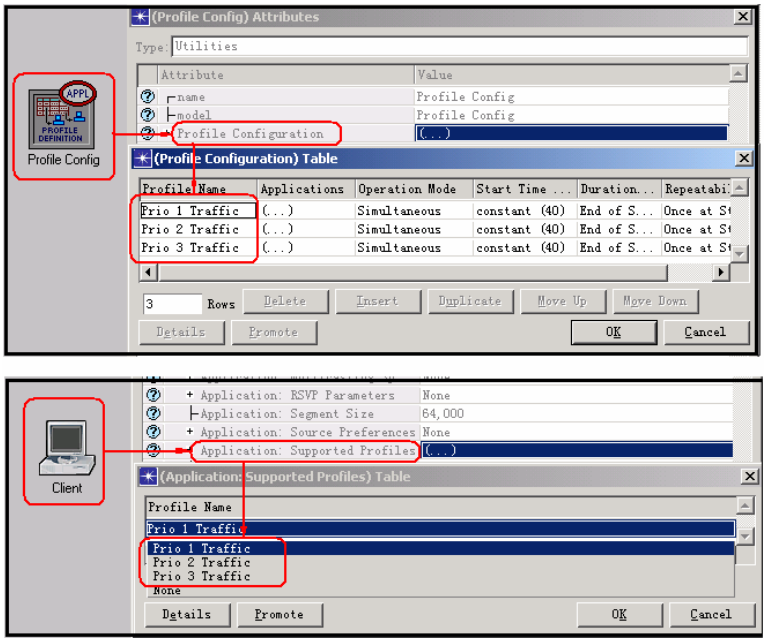


图 8-16　设定客户端业务主询

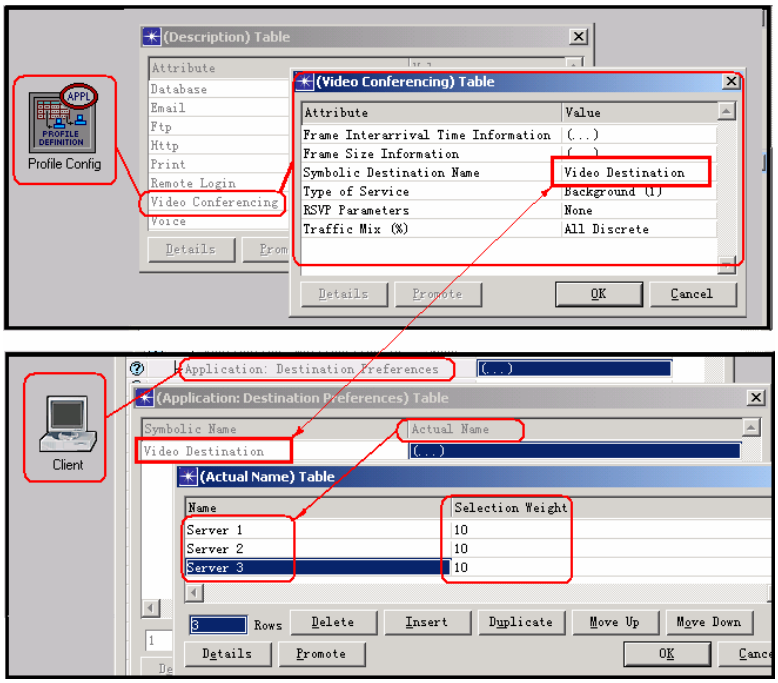


图 8-17　客户端设定业务的象征性目的地

接下来我们需要将象征性名称与实名进行映射，这种映射不只局限于一对一，而且可以一对多，如图 8-17 所示，我们可以加入 3 个服务器来支持某种应用，它们分别为 Server 1、Server 2、Server 3，这三个名称才是服务器真正的名称，我们可以在相应服务器的 Server Address 属性中找到它，如图 8-18 所示，映射了多个服务器之后就出现一个问题，既然客户都可以想这些服务器索取服务那么该选择哪一台呢？这时我们需要设定服务器被选择的比重（Selection Weight），选择比重是相对值，如图 8-17 所示的设定后，每台服务器被选中的概率是相同的。另外服务器端的 Application: Source Preference 设定原理同上。

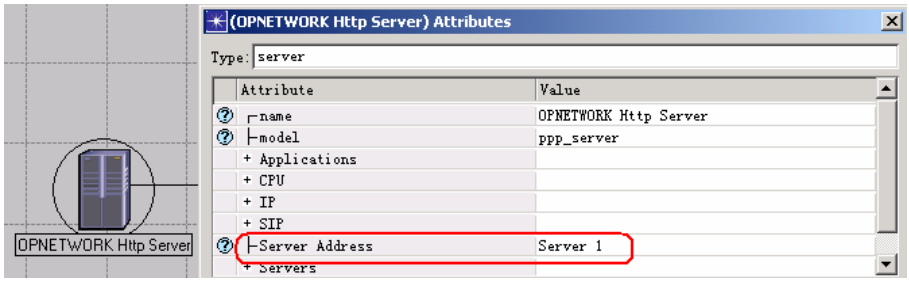


图 8-18　服务器实名的设定

如图 8-19 所示，针对不同业务需要配置的传输协议（Transport Proctol Specification）也不同，例如对于延时敏感的视频业务需要启用 UDP 协议，而 Email、FTP 等业务更侧重于业务准确性，则选择 TCP 协议。Application Segment Size 指定应用层数据包的最大值极限，如果 Application Config 中配置的封包大于这个门限就要分段。

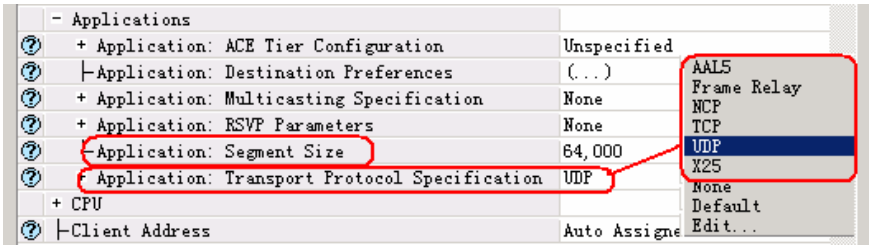


图 8-19　传输协议的设定

有益提示　在 OPNET 中，服务器和客户端的概念不是分得很清楚，例如一台服务器在提供某种应用服务的同时也可以作为另一台服务器的客户，同样道理，一个客户端同时也可以作为服务器使用。但是有一点是肯定的，一个物件设定了 Application: Supported Profiles 和 Application: Destination Preference（可设可不设）之后，它肯定是某个物件的客户，同样道理，一个物件设定了 Application: Supported Services 和 Application: Source Preference（可设可不设）之后，它肯定是某个物件的服务器。

8.3 自定义多端（Multi-Tier）业务

OPNET 提供的 9 种标准应用都是端对端业务，如果所有这些应用都不符合要求，则需要自定义业务。我们将通过任务配置器（Task）配置的业务称为多端（Multi-Tier）业务，因为它能够描述复杂的业务流程。启用 Task 设定好自定义业务后，将成为一种新的应用出现在 Application Config 业务设定选项中。

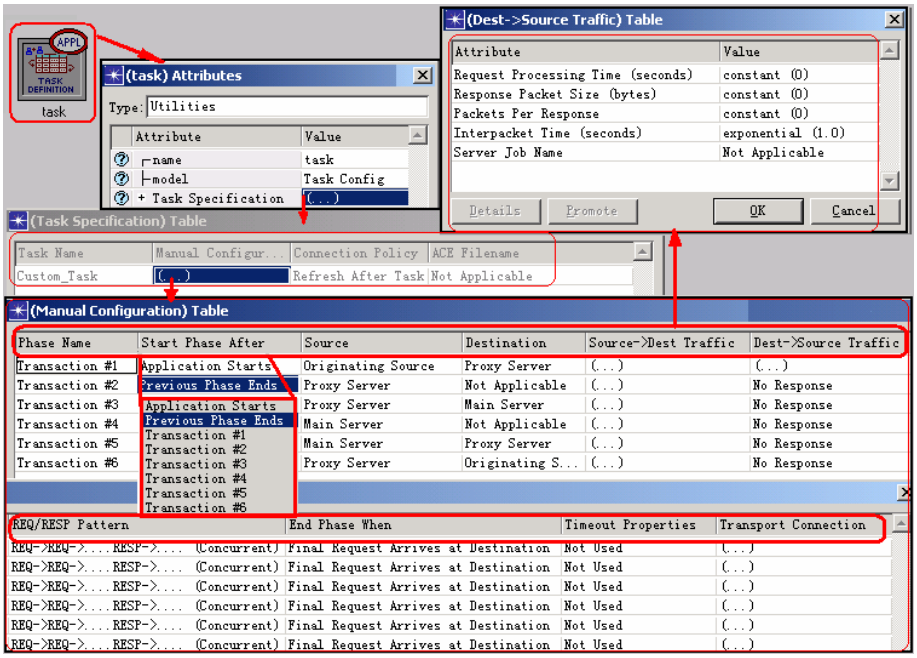


图 8-20 自定义多端业务流程

图 8-20 描述了定义任务的整个流程：打开 Task Specification 配置表；为每个任务手动配置任务包含的阶段，打开 Manual Configuration 配置表；按实际要求设定阶段的个数并且设置每个阶段的参数。其中 Phase Name 代表阶段名称；Start Phase After 代表当前阶段是接着哪个阶段开始的，如果是整个任务的开始就设定为 Application Starts，如果是紧接着前一个阶段就设定为 Previous Phase Ends，这两种设定是保留值，我们也可以从自定义的名称中选择；Source 代表当前阶段开始的起点，这个起点是场景中某个物件的象征性名称；Destination 代表当前阶段开始的终点，它也是某个物件的象征性名称；Source->Dest Traffic 描述了从源到目的节点的上行业务量大小，其中包括目的节点处理请求所需时间，请求包的大小（通常很小），业务封包的大小（代表实际有效的业务），每次请求的时间间隔；Dest Traffic 设定同上；REQ/RESP Pattern 代表响应模式，响应代表业务没有开始之前的握手信息，可以是请求一次响应一次，也可以是整个阶段响应一次，如果是前者，那么源节点在

发出新的请求之前必须等待目的节点的响应到达；End Phase When 代表阶段何时结束；Transport Connection 代表源和目的地的连接规则，其中 Policy 描述请求作用范围，可以是一次请求后始终连接，或者请求一次连接一次断开一次。

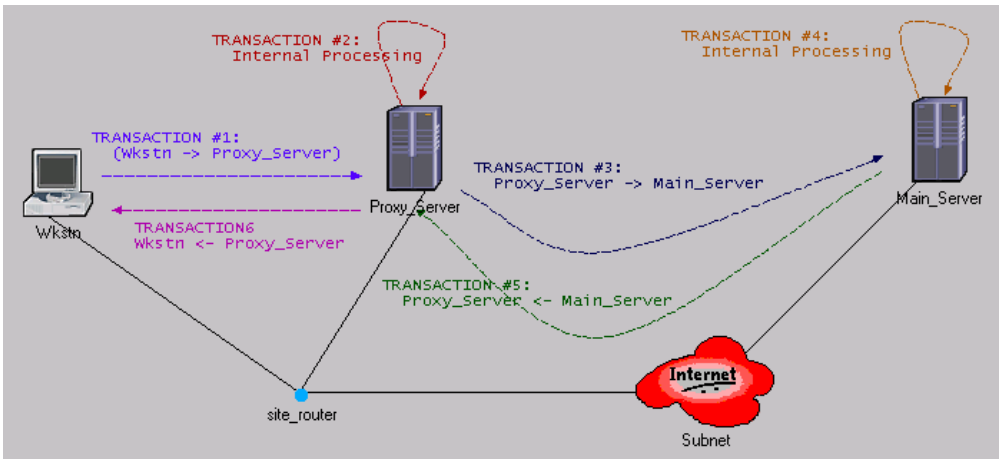


图 8-21　实际的多端业务流程

对应图 8-20 设定的 6 个阶段，实际中的情况如图 8-21 所示，下面我们列举出实际的 6 个阶段，并以物件实际名称来指定源和目的。

Transaction #1：wkstn 向 Proxy_Server 发送请求，希望它能向 Main_Server 转达业务请求

Transaction #2：Proxy_Server 收到 wkstn 请求后，处理了一段时间

Transaction #3：Proxy_Server 向 Main_Server 提交 wkstn 的业务请求

Transaction #4：Main_Server 接到 Proxy_Server 转发的请求后，处理了一段时间

Transaction #5：Main_Server 将 wkstn 想要的业务传给 Proxy_Server

Transaction #6：Proxy_Server 收到来自 Main_Server 的业务后将其转给 wkstn

我们再将这 6 个实际的阶段对照图 8-20 中 6 个定义的阶段，可以看出 Originating Source 对应 wkstn；Proxy Server 对应 Proxy_Server；Main Server 对应 Main_Server。那么它们是怎样绑定在一起的呢？

首先，图 8-20 中 Manual Configuration Table 中的 Source（象征性的源名称）栏是可以编辑的，同时也设定了 3 个保留值，Originating Source 是其中的一个，代表所有阶段的起点。我们在 wkstn 上右点键，找到 Application: Source Preferences，接着会提供几个象征性名称可供选择，我们不禁会问，其中 Proxy Server 和 Main Server 需要在哪里设定的才会在这个列表中出现呢？我们再看到图 8-20 中 Manual Configuration Table 中的 Source 栏，双击它，其实是可以编辑的，如果我们曾经输入 Proxy Server 和 Main Server 这两个名称，OPNET 将自动记录下来，并作为图 8-22 中的符号名称选项。其实我们也可以另外输入一个源名称作为任务的起点。

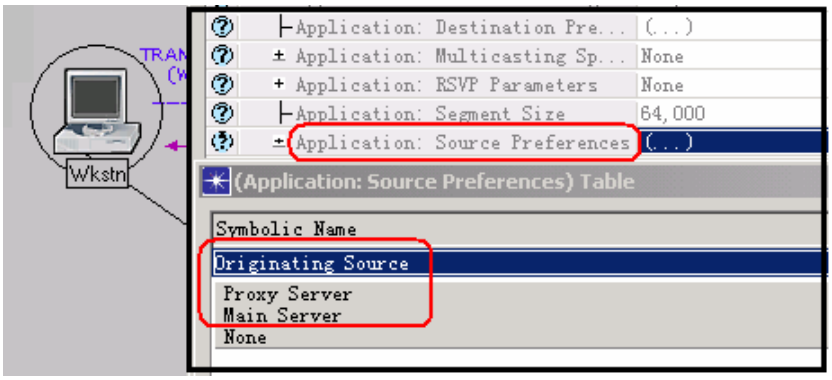


图 8-22　指定多端业务的源节点

值得注意的是，Application: Source Preferences 只要指定象征性名称就行了，而不需要设定实际名称，因为这个象征性名称就指向物件本身，相比之下 Application: Destination Preferences 的设定还需要再指定物件实际名称，如图 8-23 所示。其他的任务阶段设定类似。

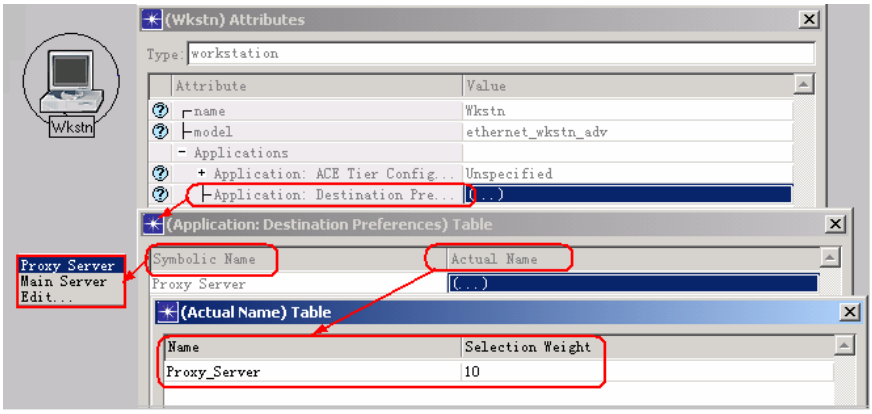


图 8-23　指定阶段的目的节点

定义完 Task 后，就需要应用配置器中对任务再作更详细的设定。在 Application Config 中肯定有一种称为 Custom 的应用类型，在任务描述表（Task Description）中设定 Task Name 为刚刚我们定义好的任务名称，我们也可以配置多个任务，同时还可以设置多个任务的比重，如图 8-24 所示。

对于每个任务可以设定其为顺序执行还是并行的，其他的设定与 8.2.1 节所述应用参数配置大致相同。值得注意的是，除了自定义业务外，也可以同时加载其他标准业务，例如 FTP 业务和 HTTP 业务。

接下来按照惯例设定业务主询，配置物件支持的应用，及设定物件业务主询，这些过程请参见 8.2.2 至 8.2.4 节。最后整个多端业务的响应时间为各个阶段响应时间之和。

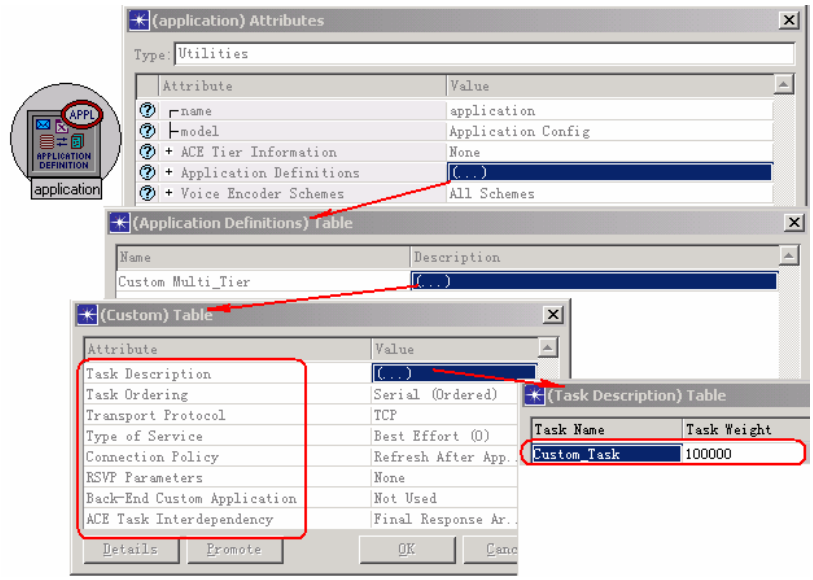


图 8-24　在应用配置器中设定自定义业务参数

8.4　流业务建模技巧

流方式的业务建模通过使用特殊的带信号的封包，告诉间中路由器背景负载量有多少，可能很长一段时间背景负载量才做一次调整然后再次通知路由器，这种方法称为分析的方法，它可以避免离散事件仿真技术所消耗的大量仿真时间，但是用这种方式得不到应用层的表现，只能模拟出 IP 层的延时，队列的性能。针对大型骨干网流量，如果需要测试加载这些业务后路由表可能出现的故障，或对协议性能的影响，则可以采用这种分析的方法。分析的方法，也称业务流方式，可以在 3 个阶层去设定，如图 8-25 所示，分别为：

（1）设定应用层业务流，采用配置 App demand 的方式来实现，其中 demand 为背景流量，在以前的 OPNET 版本中也称为 Background Router Traffic,或者 Conversation pair。

（2）设定网络层业务流，以一种称为 Tracer packet 的信息包通知路由器所需要模拟的背景总流量的多少来代替对精确封包传输的模拟，能够对 IP 层的延时和 queue 产生影响，而不能观察到应用层的表现；

（3）设定底层业务流，一般针对 ATM 层，目前并不是很关注这种方式。

接下来我们分别针对应用层和网络层业务流，说明它们如何设定。

8.4.1　针对话音和视频业务背景流的设置

由于话音和视频业务数据量特别繁重，如果采用精确业务模拟的方法会使整个仿真变

得非常慢，如果每个封包都从应用层发送的话，往往仿真一两个话音流可能需要 30 分钟左右。针对这种情况，OPNET 为话音和影像业务特别增加了一个称为 Traffic Mix (%)选项，如图 8-26 所示，如果选择 All Background 则全部作为背景流量，这将得不到应用层的统计，只能看到 IP 层的表现；也可以设置百分比，例如 50%，则有一半封包精确发送，这部分封包将对应用层的表现起作用，而另外一半当作是背景流量。

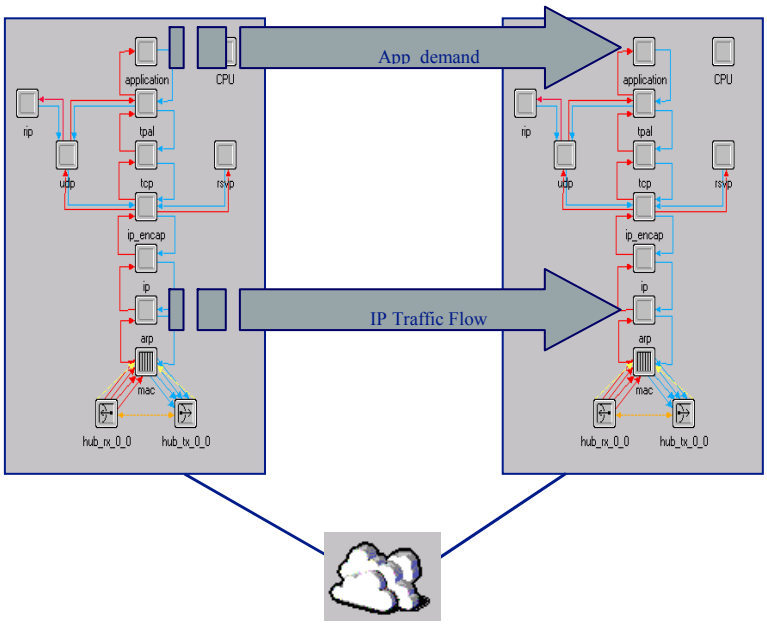


图 8-25　应用层和网络层业务流的模拟

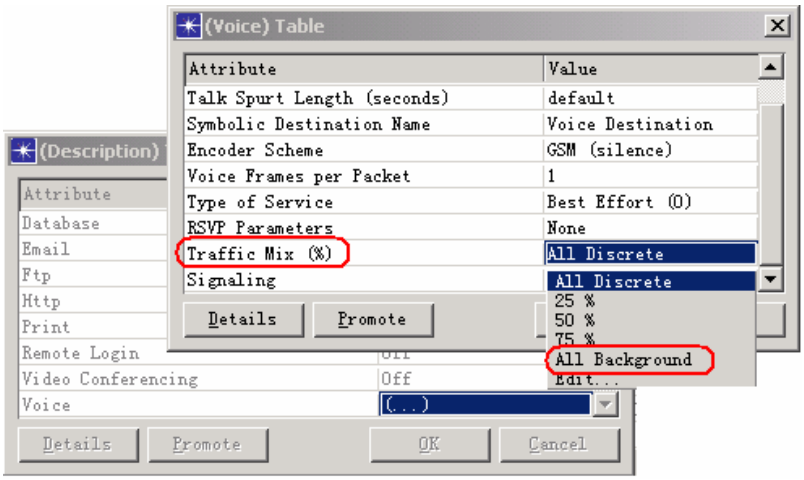


图 8-26　针对话音和视频的背景流比例的设置

一般来说要验证复杂场景所加业务的有效性，首先做流分析模拟，得到一种预期的结果，然后在我们外出的时候，例如下班回家时再进行精确模拟。针对话音和视频业务我们

可以很方便地设定业务仿真的粒度。

8.4.2　应用流背景流建模

首先设定应用流，如图 8-27 所示，打开流模型编辑器，主要将 technology 属性设定为 Application，它指明了流建模针对网络哪个阶层，其他还有如背景流在场景中表现的线条颜色、线型和粗细等设定。之后找到包含该应用流模型的物件拼盘，采用一拉一点的方式在场景中两个物件之间建立一条应用流。这里配置好的应用流为蓝色的虚线，如图 8-28 所示。

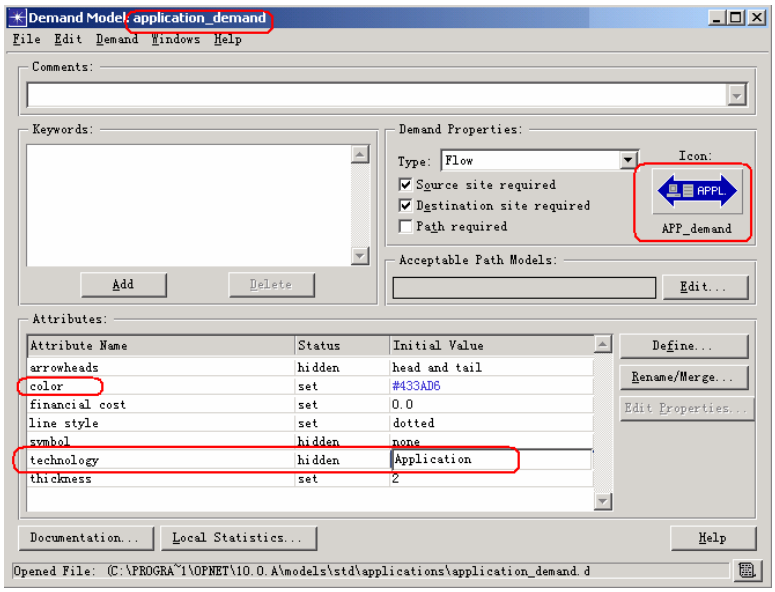


图 8-27　流模型编辑器

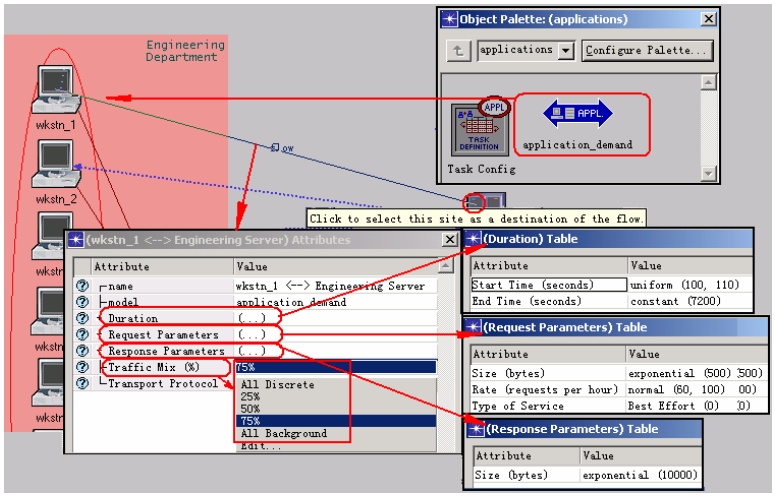


图 8-28 应用流的设定

接下来在应用流线上右点键设定其属性，其中 Duration 指定流的开始和结束时间；Request Parameters 设定源节点发送的请求包大小和请求速率；Response Parameters 设定目的节点发来的响应包的大小；Traffic Mix (%) 指明百分之多少作为背景流量，余下的还是可以得到高层的业务表现，如果该值设定为 0%则意味着全部业务采用离散仿真方式，如果设定为 100%则全部采用背景流方式。如果 Traffic Mix 小于 100%则还是能看到一些应用层对业务的表现；另外还有传输协议的设定 Transport Protocol，要根据业务的类型来定。

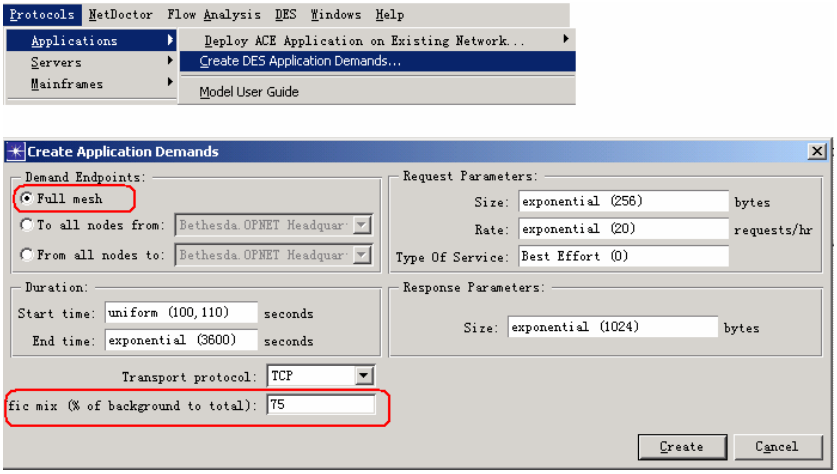


图 8-29 配置应用背景流

如果觉得上述一拉一点设置方式比较麻烦，特别是针对大型网络，我还可直接从 Protocol 菜单导入，如图 8-29 所示，关键是设定背景流的比例。另外如果选择 Full mesh 的流配置方式，所有的节点都会以矩阵形式配对，当网络场景包含的物件很多时，配置流的时间会很长，图 8-30 为一种配置后的情形。

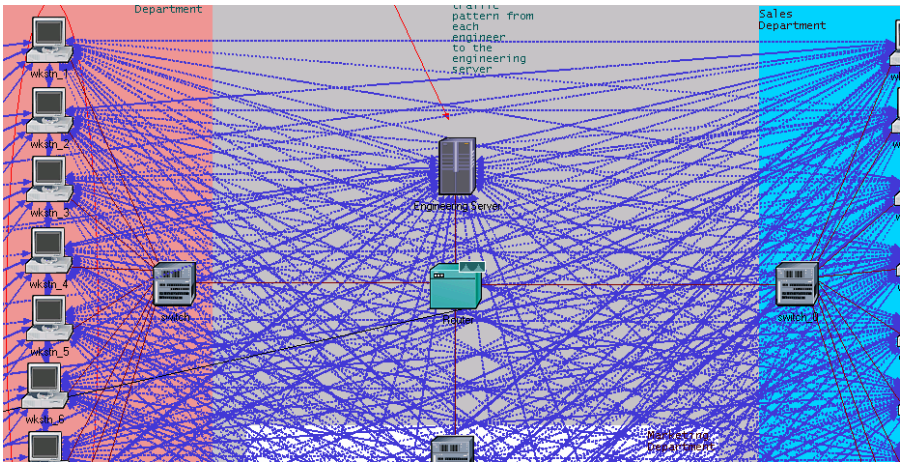


图 8-30 使用 Full mesh 的方式配置应有流后的场景

8.4.2 网络层背景流建模

对于 IP 层背景流的设定前面步骤和应用流设置相同，只是 IP 流属性设定要复杂一些，如图 8-31 所示，Traffic (packets/second)代表背景流需要模拟封包的生成速率，我们可以在不同时间段设定不同的封包速率，最后一个设定的值将一直延续到仿真结束，Traffic (bits/second)的设定与上述方法相同。将 Traffic (bits/second) 除于 Traffic (packets/second)将得到背景流所模拟的平均封包大小；Traffic Characteristics 指定业务的需求的服务质量（Type of Service），包的大小（Packet Size PDF），包间间隔（Packet Interarrival Time PDF），在每段背景流间隔内发送的跟踪包的个数（Tracer Packets Per Interval），可以设定多个，如果选择 Same As Global Setting，则将在仿真属性界面中指定，如图 8-32 所示。由于 Tracer packet 承载着 demand 流量的信息，考虑到可能在路由过程中被丢掉，Tracer Packet Redundancy 确保当传输有故障时重发跟踪包；Traffic Scaling Factor 流量尺度因子，加大它流量会成倍增加，一般越小仿真越精确。

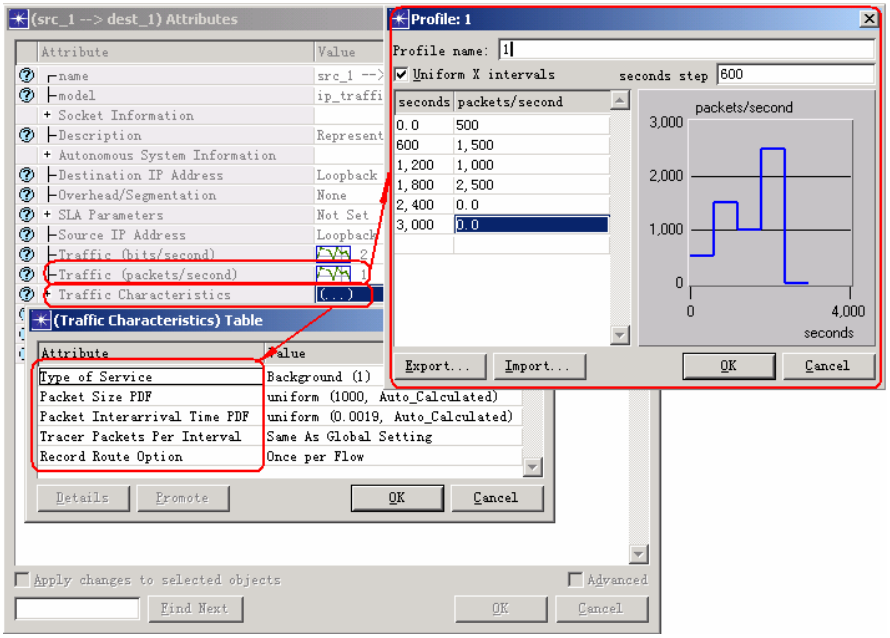


图 8-31 网络层背景流的设定

以上的业务特征的设定只有在路由器中配置了 QoS 时才有效。如图 8-33 所示，首先在场景中放置一个 QoS 配置器（在 utility 物件拼盘中），假如我们想要配置 WFQ（加权公平调度队列）调度器来支持路由器实现 QoS，那么就需要设定 WFQ 调度算法的规格（WFQ Profiles），接下来有几种方案可供选择，有基于 ToS 方案，有基于协议或者端口的方案，有基于区分服务方案（在 IP 包头中设定区分服务编码点 DSCP），或者自定义其他方案。

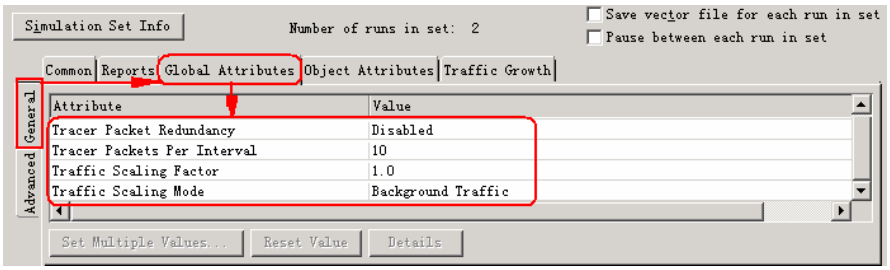


图 8-32　仿真属性中有关全局业务属性的设定

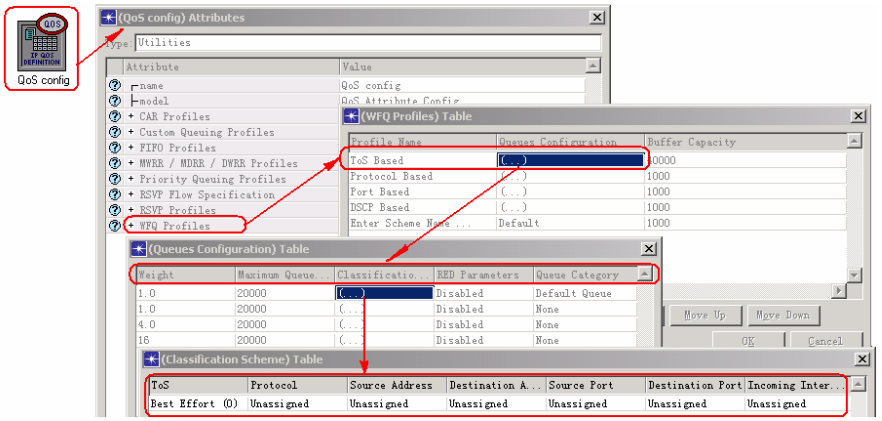


图 8-33　配置支持路由器实现 QoS 的调度算法参数

配置好调度算法参数后，就可以将 QoS 设定配置到路由器中，如图 8-34 所示，我们将在 QoS 配置器中设定的 WFQ 方案指定给该路由器。

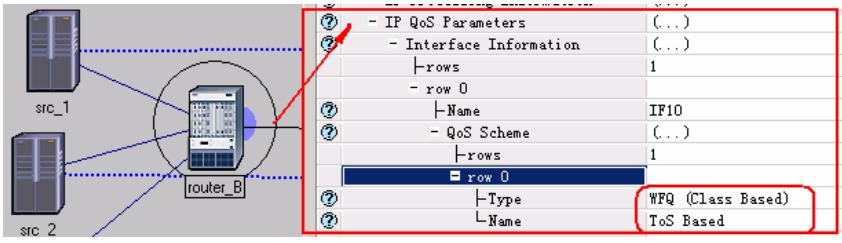


图 8-34　路由器中配置 QoS 调度方案

配置背景流需要在网络模型中一拉一点，每次只能设定一对，如果觉得这种方法麻烦，我们也可以采用一种特殊的方式，它非常简单，首先选定需要配置背景流的所有物件，如图 8-35 所示在 Traffic 菜单下选择将背景流导入到 Spreadsheet 中。

这时会出现指定输出流规格的对话框，如图 xxx 所示，其中 Traffic duration 和 Time Step 指定了背景流一共有多少段，每段持续时间多少，例如我们想配置 60 分钟业务，每 10 变化一次，那么 Traffic duration 就设定为 60，Time Step 设定为 10；Traffic units 指定背景流量单位，一般选择 Kbps；比较有用的是 Export node pairs that have no traffic flows 要选上，

这样原本在场景中没有配置背景流的物件也会出现在配置文件中，从而提供了一个设置具体流参数的模板。OPNET 以矩阵方式产生背景流，假如有 10 个物件参与背景流的导出，那么就有 10 x 9 共 90 条流配置记录，因为对于每个物件其他的 9 个物件都以它为目的地建了一个背景流，那么 10 个物件共有 90 条流。当我们点击确认后流量配置文件就自动生成了，如果安装了 Microsoft Excel 将自动启动 Excel 打开流量配置文件，如图 8-36 所示。

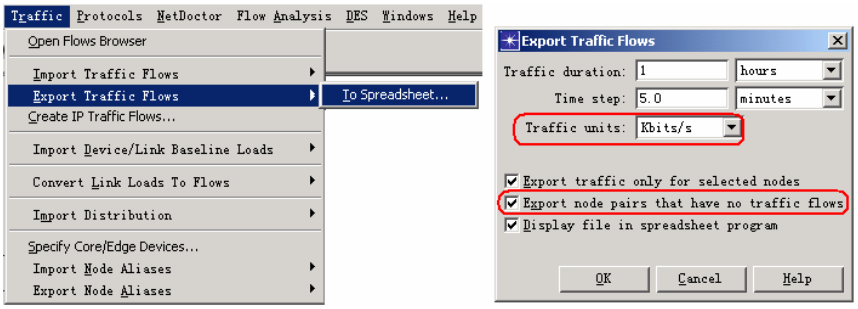
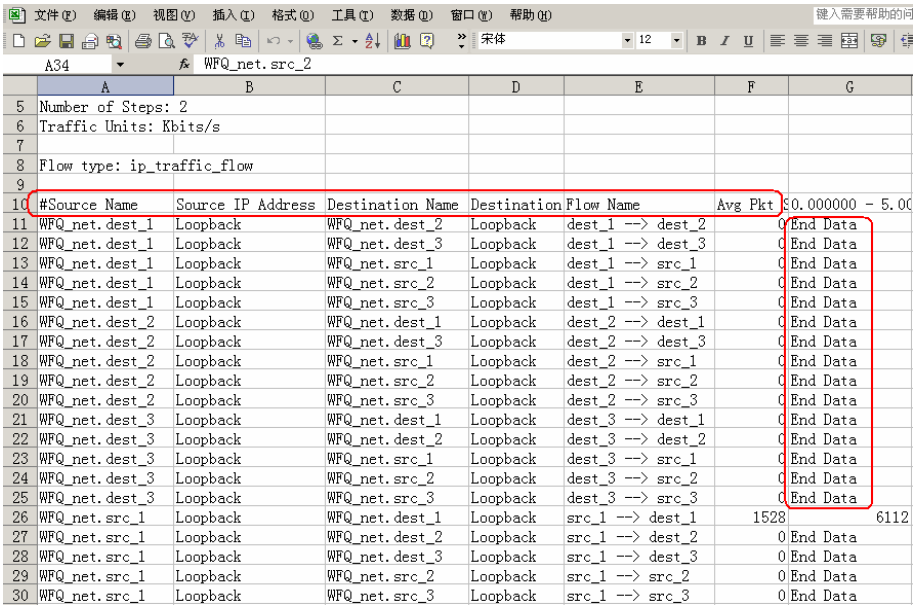


图 8-35　导出背景流的操作



	A	B	C	D	E	F	G
5	Number of Steps: 2						
6	Traffic Units: Kbits/s						
7							
8	Flow type: ip_traffic_flow						
9							
10	#Source Name	Source IP Address	Destination Name	Destination	Flow Name	Avg Pkt	30.000000 - 5.00
11	WFQ_net.dest_1	Loopback	WFQ_net.dest_2	Loopback	dest_1 --> dest_2	0	End Data
12	WFQ_net.dest_1	Loopback	WFQ_net.dest_3	Loopback	dest_1 --> dest_3	0	End Data
13	WFQ_net.dest_1	Loopback	WFQ_net.src_1	Loopback	dest_1 --> src_1	0	End Data
14	WFQ_net.dest_1	Loopback	WFQ_net.src_2	Loopback	dest_1 --> src_2	0	End Data
15	WFQ_net.dest_1	Loopback	WFQ_net.src_3	Loopback	dest_1 --> src_3	0	End Data
16	WFQ_net.dest_2	Loopback	WFQ_net.dest_1	Loopback	dest_2 --> dest_1	0	End Data
17	WFQ_net.dest_2	Loopback	WFQ_net.dest_3	Loopback	dest_2 --> dest_3	0	End Data
18	WFQ_net.dest_2	Loopback	WFQ_net.src_1	Loopback	dest_2 --> src_1	0	End Data
19	WFQ_net.dest_2	Loopback	WFQ_net.src_2	Loopback	dest_2 --> src_2	0	End Data
20	WFQ_net.dest_2	Loopback	WFQ_net.src_3	Loopback	dest_2 --> src_3	0	End Data
21	WFQ_net.dest_3	Loopback	WFQ_net.dest_1	Loopback	dest_3 --> dest_1	0	End Data
22	WFQ_net.dest_3	Loopback	WFQ_net.dest_2	Loopback	dest_3 --> dest_2	0	End Data
23	WFQ_net.dest_3	Loopback	WFQ_net.src_1	Loopback	dest_3 --> src_1	0	End Data
24	WFQ_net.dest_3	Loopback	WFQ_net.src_2	Loopback	dest_3 --> src_2	0	End Data
25	WFQ_net.dest_3	Loopback	WFQ_net.src_3	Loopback	dest_3 --> src_3	0	End Data
26	WFQ_net.src_1	Loopback	WFQ_net.dest_1	Loopback	src_1 --> dest_1	1528	6112
27	WFQ_net.src_1	Loopback	WFQ_net.dest_2	Loopback	src_1 --> dest_2	0	End Data
28	WFQ_net.src_1	Loopback	WFQ_net.dest_3	Loopback	src_1 --> dest_3	0	End Data
29	WFQ_net.src_1	Loopback	WFQ_net.src_2	Loopback	src_1 --> src_2	0	End Data
30	WFQ_net.src_1	Loopback	WFQ_net.src_3	Loopback	src_1 --> src_3	0	End Data

图 8-36　Excel 打开的流量配置文件

我们可以看到有些记录并没有指定背景流参数，它们被标识为 End Data，但是这只不过是为我们提供的创建背景流的一个模板，接下来我们我们需要删除一些不感兴趣的背景流记录，并为剩下的记录指定参数。由于数据量太大，我们可以通过以下方法设定参数：

假如我们想设定包的速率在 100kbps 上下抖动，（带宽单位是在图 xxx 中设定的），那么我们可以先选定一个参数输入栏，在 Excel 函数栏输入 =RAND()*100，按回车；接着横向拖动右下角的十字架，在列拉，如图 8-37 所示。

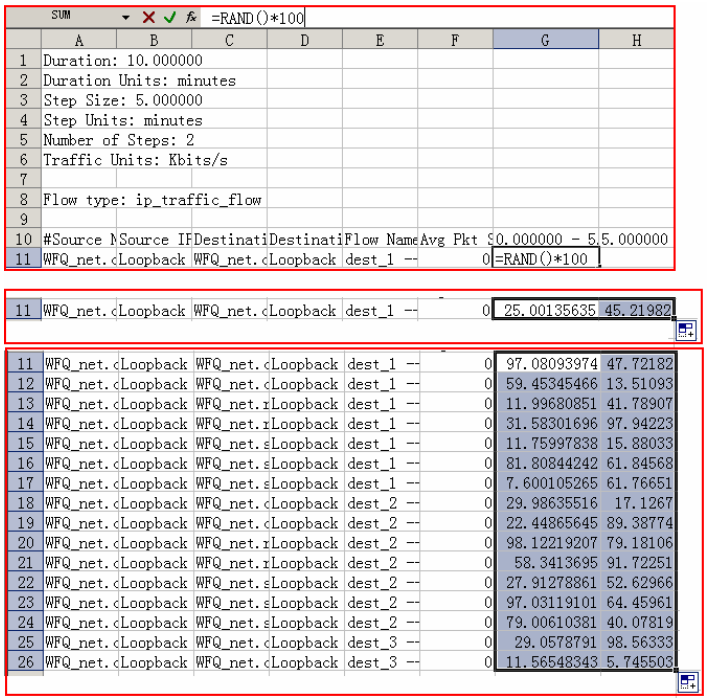


图 8-37　设定流配置文件中业务量大小

之后将改变的结果包存为新的 txt 文件，接着从 Traffic 菜单中选择导入背景流，如图 8-38 所示，一般选择 Replace all existing traffic，完全覆盖之前的设定。

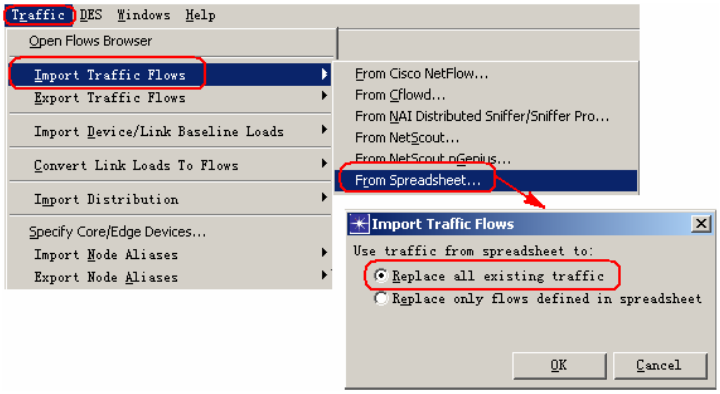


图 8-38　导入流配置文件

8.4.3　Micro-Simulation 技术

针对大型网络中的繁重业务，传统的离散事件仿真技术要得到收敛的结果必须消耗大量的仿真事件且占用大量内存。尤其是在网络层引入调度算法（如 PQ，CQ，WFQ）之后，

纯粹的离散事件仿真方法就更加缺乏其可伸缩性。而分析的方法也不足以很好地表现协议的动态行为。因此，针对涉及某些复杂算法的网络建模，OPNET 引入一种称为 Micro-Simulation 业务建模方法，它既能保证速度，又能比分析的方法提供更多的协议细节。

那么 Micro-Simulation 是怎么运作的呢，它如何体现背景业务的细节呢？我们知道，背景业务的特点是在一个大范围内业务参数是一定的，小到十分钟，也可以大到几天的时间。那么这段时间内路由器处在相对稳定的状态，如果它配置了多个队列，并且采用了某些特殊的调度算法，从而支持具备高优先级的队列能够获得更多的权值带宽，而一些延时敏感的实时业务进入高优先级的队列也将获得更多的服务机会，以保证其服务质量。我们期望的是加载的业务流能够呈现出不同的 QoS 表现。可是背景业务流在很长时间所保持的稳定性不能够产生突发，从而不能有效验证调度算法的性能，也就是说，调度算法的性能发挥不出来。

Micro-Simulation 可以解决上述问题，为了突破背景业务仅仅关心整个时段收集的总封包流量大小的局限性，OPNET 中采用一种称为 Tracer packet 的封包将包含背景业务流特征的信息传递给它所经过的路由器或交换机。当背景业务流特征发生改变时发送新的 Tracer packet，从而及时通知相关物件，使它们能够立即调整队列的状态，这就是 Micro-Simulation 的核心思想。具体来说，背景流封包在一段时间内每个包的速度、大小一样，Micro-Simulation 使它们的大小和速度产生抖动，从而引起潜在的业务突发，从而使精确封包的排队延时的计算更加精确。Micro-Simulation 并不产生真正的封包，而是一种数学模型上参数校验。

一般 Micro-Simulation 配合背景流业务能使得业务仿真即不失离散事件仿真的精确度，又使仿真时间大大加快。OPNET 使用背景业务配置器（Background Traffic Config）来界定在何种条件下采用分析方式，何种情况下启动 Micro-Simulation。如图 8-39 所示，我们从 utilities 物件拼盘中找到背景业务配置器，下面对其中的属性进行说明。

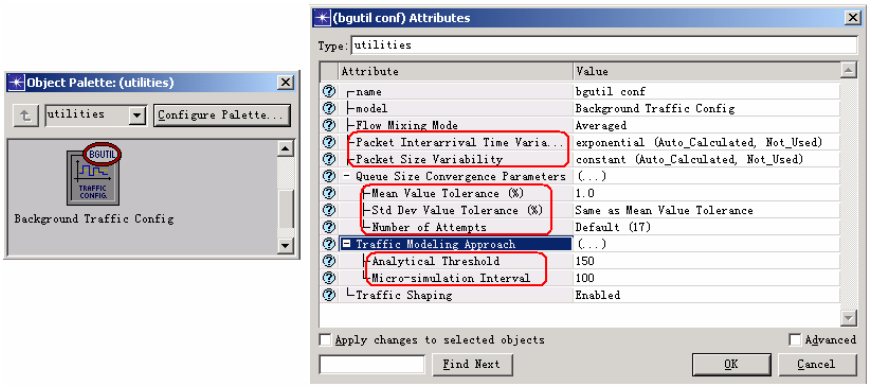


图 8-39　设定背景业务配置器

Packet Interarrival Time Variability 和 Packet Size Variablility 分别代表为背景封包在速度和大小上引入的抖动。

Mean Value Tolerance (%) 代表队列对队长抖动可容忍的程度。如果队列的平均长度的抖动小于这个值，则说明精确封包流在某种程度上达到稳定状态，队列的特殊调度算法将不能发挥出它的特性，此时 Micro Simulation 的意义已经不大，所以停止这种方式，背景流将启用分析的方式。这个值设置得越大，说明对队列得抖动越不敏感，不能够很好地表现调度算法性能，Micro Simulation 相对使用的比例要小，仿真速度也更加快。

Traffic Modeling Approach 设定选择 Micro-simulation 还是分析方式。

Analytical Threshold 设定启动分析模式的门限值，如果背景封包的平均间隔时间超过这个值则采用分析的方式，否则启动 Micro-simulation；如果设定为 Infinite 则一直启动 Micro-simulation，如果设定为 Minimum，则始终采用分析的方式。

Micro-simulation Interval 设定启用 Micro-simulation 的持续时间，如果选择 Always 则整个背景流采用 Micro-simulation 仿真，Never 则使用纯分析的方法。

8.5　链路背景业务建模

链路背景业务建模相对其他方法最简单，精确度也最差，只是通过改变当前链路的背景负载来模拟被背景业务占用的带宽。例如链路背景使用率（Background Utilization）为 60%，假设共有 1Mbps 的带宽，则实际只有 400kbps 带宽可供使用。这种方法不能象流建模那样模拟出网络层的表现和队列的性能。

在感兴趣的链路上右点键，接着就可以在链路属性对话框中配置背景负载，Average Packet Size 指定背景封包的大小，Intensity 指定在哪个时间间隔内带宽被占用多少，如图 8-40 所示。

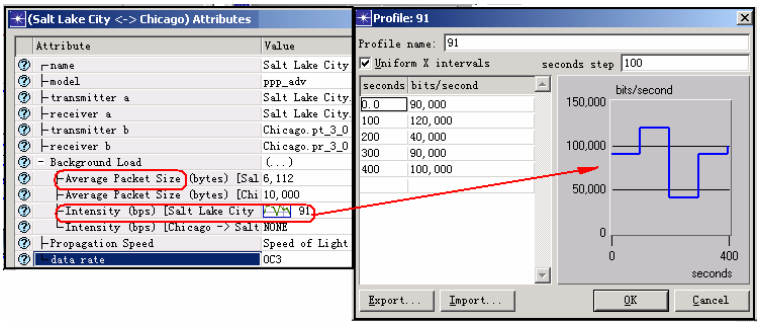


图 8-40　设定链路背景负载

与设定链路背景使用率相似，对服务器也可以设定 CPU 的背景使用率，如图 8-15 所示。

8.6　混合业务建模

业务建模方法可以分为 3 类：

（1）完全精确的方式，模拟每一个封包的行为，如果要得到应用层延时性能则要采取这种方式，但是占用大量的仿真时间和资源；

（2）采用背景流的方式模拟业务，从统计学角度对网络性能给予数学上的逼近，其中又分为分析的方法和 Micro-Simulation；如果纯粹用分析的方式，得到结果精确度还是有一定的距离，而引入 Micro-Simulation 机制能够更加精确地计算队列延时，因此这两种背景流方式通常配合使用以达到较好的效果。

（3）加载链路背景业务。

这三种业务模拟方式粒度由大到小，完全采用第 1 类方法所耗网络仿真时间非常长，在未知仿真结果就是所要的情况下，我们可以采用第 2 种或第 3 种。为了在精确度和仿真时间中取得一个折中，所以采用混合模拟方式。在仿真的不同阶段我们可以采用 3 种方法的不同程度的混合，达到一个最好的平衡点，如图 8-41 所示。在不简化总的业务量情况下，既能得到精确模拟所带来的应用层表现，又使得仿真速度能够接受。

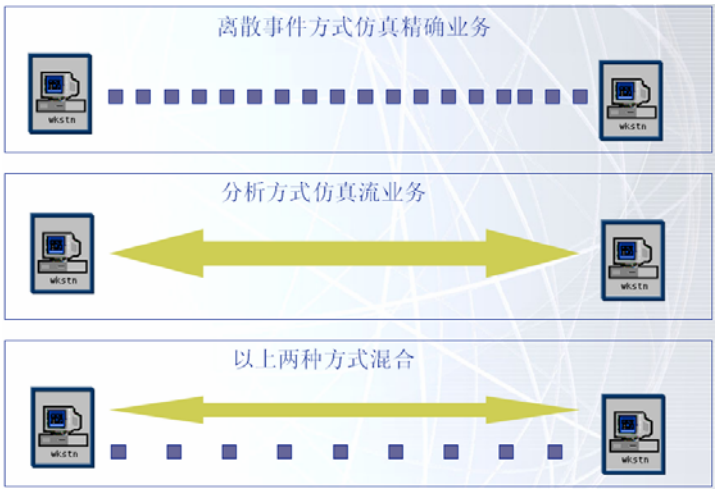


图 8-41　混合业务建模

第 9 章　无线信道建模

随着宽带无线网络的发展和多媒体技术的进步，无线网络的研究越来越引起人们的兴趣。特别是无线局域网（WLAN：Wireless Local Area Network）由于联网方便、移动性和扩展性好以及费用相对低廉等特点，应用越来越广泛。

在无线环境中，由于噪声、干扰、多径和移动终端漫游等因素影响，信道状况随着时

间变化很大。无线链路的仿真对被传输的包产生的两个方面的效果，一个是包的时延，另一个是包的误码特性。无线链路仿真最终能够得到各种无线信道误码特性、数据成功率、数据服务质量和抗干扰能力等主要性能指标。但是无线链路不作为物理对象而存在，不存在独立的链路实体，而且无线链路是一种广播媒介，每一传输都可能影响整个网络系统中的多个接收终端，所以仿真一个无线数据包的传输要考虑发射信道和所有可能接收信道的组合。无线建模的种种困难随着 OPNET 等网络仿真工具的诞生迎刃而解。

无线链路级仿真器与系统级仿真器的接口如图 9-1 所示。链路级仿真器向系统级仿真器提供了所需的输入参数，包括上行链路及下行链路的包信道传输延时和误码率特性。

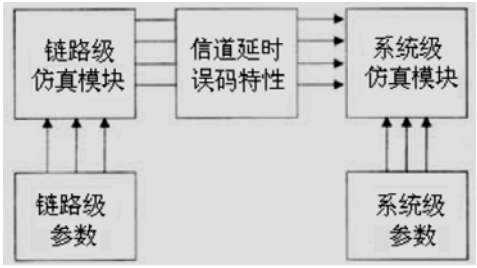


图 9-1　链路级仿真器与系统级仿真器的接口

如果需要对物理层非常精细的仿真，光靠 OPNET 的管道建模（pipeline）机制可能达不到要求的仿真精度。例如对特殊的无线物理媒质传输特性的仿真，或复杂物理层通信协议的仿真。这时可以用 Matlab、SPW 和 COSSAP 等软件仿真链路层相关协议，然后通过接口将它们和 OPNET 联系起来。

9.1　无线模拟简介

无线信道通过设定无线收发信机属性来模拟，如图 9-1 所示设定信道属性：

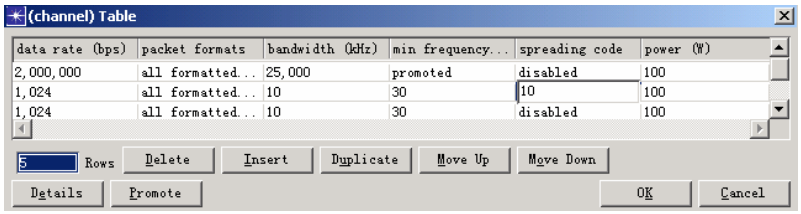


图 9-1　设定信道属性

我们可以建立多个信道，每个信道包括信道传输率、支持封包格式、基本频率、带宽、功率和扩频码（spreading code）等属性。如果启用扩频码就直接设定一个 double 值，如果两个收发信机信道对的扩频码相同就可以互相通信，否则视为噪声。在仿真过程中，data rate 可以改动，因此 10.0 版本将 802.11 MAC 物理层由原来的 4 个信道（data rate 分别为 1.1M，2M，5M，11M）简化为 1 个信道，由于任何时候只有一个信道正在使用，所以只要动态设置所需的 data rate 即可。

接下来设定管道阶段（Pipeline Stage）模型，它们将计算物理层特质，最终目的是计算差错率并判断是否丢包，无线管道阶段共有 13 个，期间的任何数据都保存在封包的 TDA 属性中，后续的管道阶段就可以共享这些数据。包编辑器定义的指示包的主体部分（Packet Body），而包的信息头记载了包的创建时间、Packet ID、Packet Tree ID 等信息，如图 9-2 所示。包的信息头和 TDA 都入封包的长度。

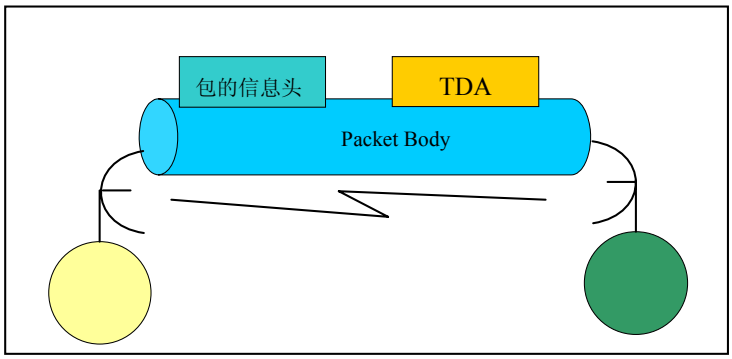
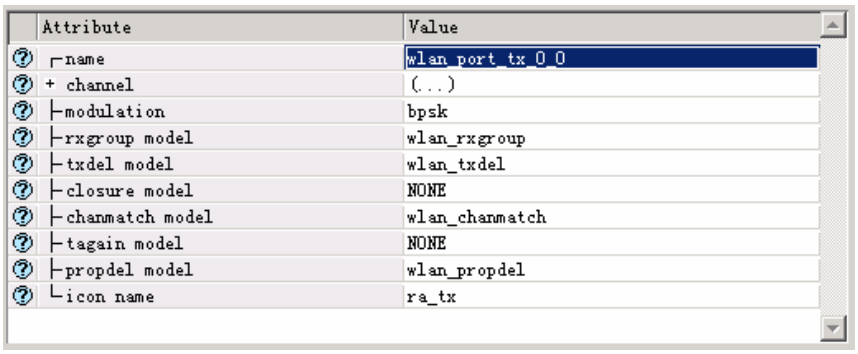


图 9-2　包完整的结构

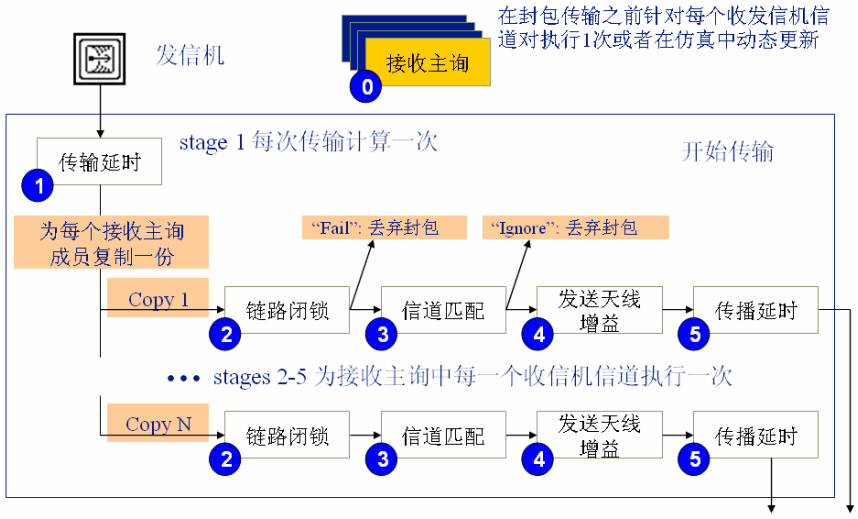
对于任何可能的收信机信道，封包都被拷贝一次经历后续的管道阶段，接下来查看管道阶段的设定。如图 9-3 所示，打开发信机属性对话框：

我们可以看到配置的几个管道阶段模型，如接收主询（rxgroup）；链路闭锁（closure）；信道匹配（chanmatch）；发送天线增益（txgain）；传播延时（propdel）。由于对每个可能的接收主询，发信机都将复制一份数据包去尝试是否能达到对方收信机信道，因此可以想象无线仿真的速度比有线慢得多。注意如果在 rxgroup 中没有将同一节点模块下的收信机隔掉，将发生自己对自己发送数据包的情况，这在逻辑上应该避免的。

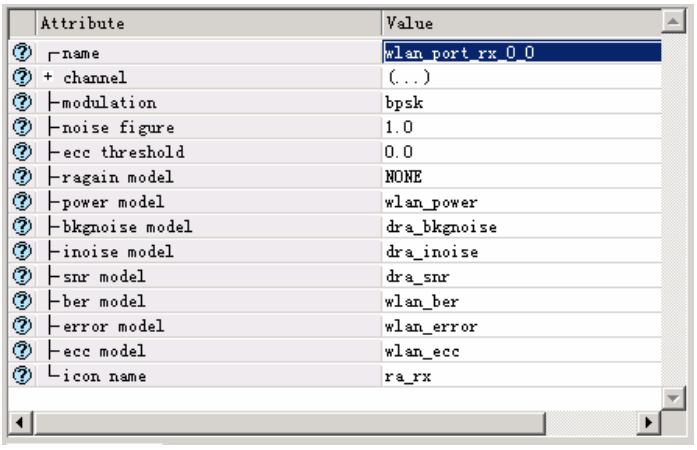


Attribute	Value
name	wlan_port_tx_0_0
+ channel	(...)
modulation	bpsk
rxgroup model	wlan_rxgroup
txdel model	wlan_txdel
closure model	NONE
chanmatch model	wlan_chanmatch
tagain model	NONE
propdel model	wlan_propdel
icon name	ra_tx

图 9-3　发信机的属性



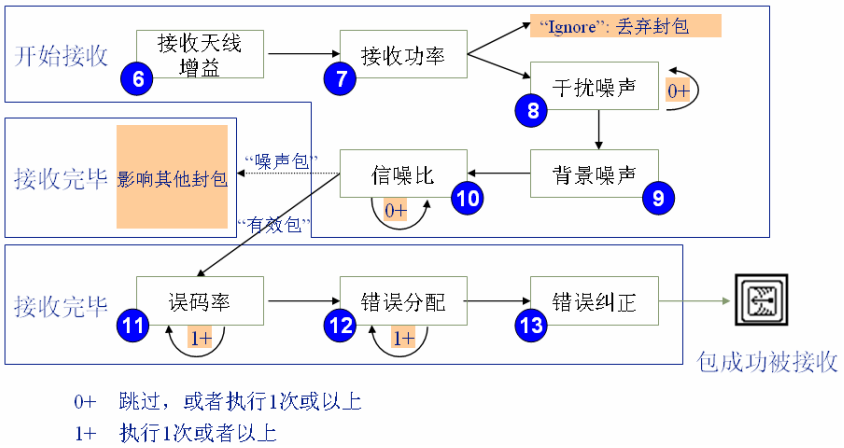
如图 9-4 所示，打开收信机属性对话框：



Attribute	Value
name	wlan_port_rx_0_0
+ channel	(...)
modulation	bpsk
noise figure	1.0
ecc threshold	0.0
ragain model	NONE
power model	wlan_power
bkgnoise model	dra_bkgnoise
inoise model	dra_inoise
snr model	dra_snr
ber model	wlan_ber
error model	wlan_error
ecc model	wlan_ecc
icon name	ra_rx

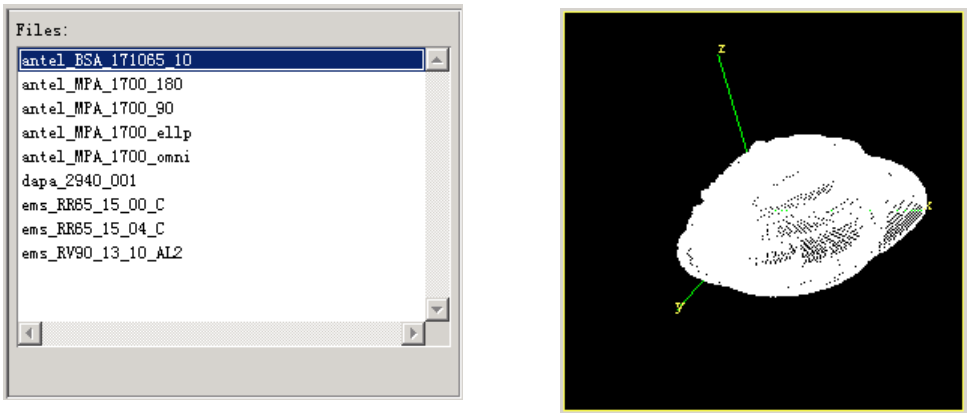
图 9-4　发信机的属性

对于收信机来说有 8 个管道阶段，如接收天线增益（ragain）；接收功率（power）；背景噪声（bkgnoise）；干扰噪声（inoise）；信噪比（snr）；误码率（ber）；差错分布（error）；错误纠正（ecc）。计算了接收功率和背景与干扰噪声的叠加功率就可以计算 snr，之后调用调制曲线找出相应误码率。注意封包在无线中的传输分为很多段，信噪比在每一段都不一样，将每段误码率加起来才能决定最后是收包还是丢包。



注意管道阶段的文件名和函数名要完全吻合，否则编译通不过。在 xxx 节将进一步描述无线管道阶段做什么，什么时候调用，返回什么值等细节。

OPNET 本身自带一个天线模型编辑器可以方便地对天线进行建模。在 9.1 版本之后出现一些工业和商业标准的天线模型，它们的样子奇特，图 9-5 所示为其中的一种。



特殊天线列表　　antel_BSA_171065_10

图 9-5　新增的天线模型

设置天线模型首先需要设置天线模型的粒度（Number of Phi planes），如果设定天线的片数为 180，每片 2 度。天线模型将被水平面切成 180 份，读者可以想象水平面与天线模型相交将得到一条闭合曲线，它与三维坐标原点构成一个锥面。将基于极坐标（ρ，θ）（其

中 ρ 为天线增益）的闭合曲线展开到二维平面坐标系就得到图 9-6 所示的曲线。

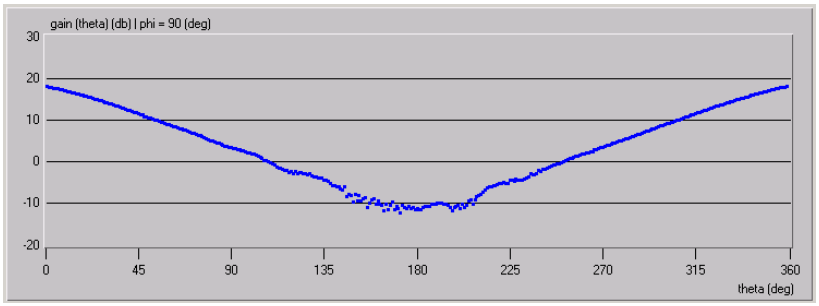


图 9-6　片展开而成的曲线

theta 的采样点数为 360 度除以每片的度数（360/2 = 180 个采样点）。如果想让增益设定得更加准确，可以定义其上限和下限值，使增益的分辨率提高。如果觉得手动设置不够精确还可以导成 EMA 文件继续修改。

当我们建好天线模型时，可以将其设定到天线模块中，同时设置天线基准点（pointing ref）角度以及天线瞄准的目标（target）坐标如图 9-7 所示。

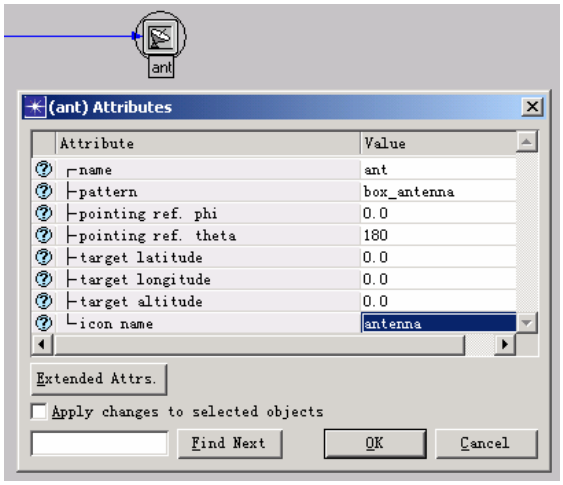


图 9-7　天线属性的设定

调制曲线（modulation curve）模型的设定和天线模型类似，它用来计算误码率。

如果有地形建模模块（TMM）还可以考虑地形在无线路径损耗计算中的影响，从而增强传播损耗、信号强度和噪声等参数计算的准确性。OPNET 仿真支持 DTED 或者 USGS DEM 格式输入的地形文件。其中 USGS 是美国专用的格式，DEM 被世界各地采用。整个地形数据大致由地形网格拼凑而成，地形网格大小表征地形建模的分辨率，一个网格代表一小块区域，它包含数据起始经度(东) 、终止经度(西)、终止纬度(南)和 起始纬度(北)以东南西北四个方向界定仿真地域在地球表面所处的位置和大小。地形参数将在无线链路建模中的 9.3 节中使用，主要用来计算特定地形中收发天线的物理可达性和它们之间的无线

链路损耗。如图 9-8 所示导入电子地图后，可以看到 OPNET 用等高线表示的地形图。

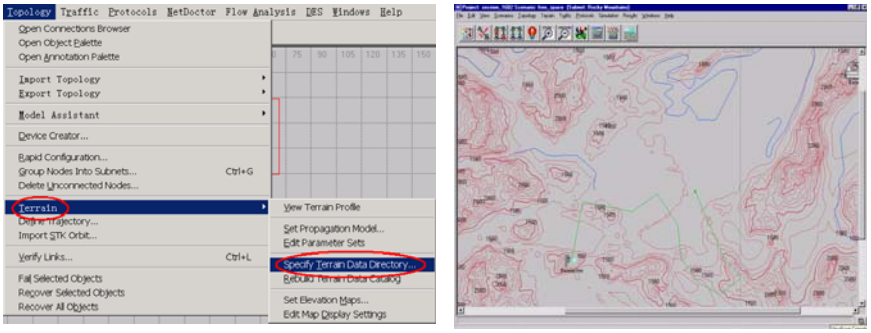


图 9-8　导入电子地形图

另外点击 Topology->Terrain->View Terrain Profile 可以查看两点之间的地形，选择 Topology->Terrain->Set Propagation Model…可以设置地形的传输模型，如图 9-9 所示。

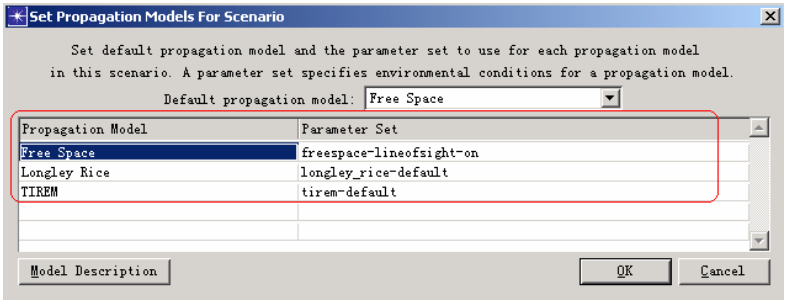


图 9-9　设置地形的传输模型

9.2　无线移动方式

OPNET 在无线中可采用的移动的方式基本上可以分成两类：定义移动站在仿真过程中运动的路线（运动轨迹）或者改变物件的位置属性。另外，如果配置了地形模型时，物件也会自动根据地形高度来移动。

9.2.1　分段移动方式

这种是最原始也是最常用的定义运动轨迹 OPNET 的方法，基于段（Segment）的运动轨迹通过预先定义好的一系列点来进行定义。运动轨迹文件存在 ASCII 码的文本文件中，文件后缀是*.trj。通过设定节点或者子网的 Trajectory 属性来为其分配运动轨迹，这时轨道线将变成绿色，如图 9-10 所示。仿真中，移动节点根据定义的运动轨迹，从一个点移动到

下一个点。如果仿真时间超过基于段的运动轨迹所设定的最大时间，则该对象仍然停留在运动轨迹的末端。

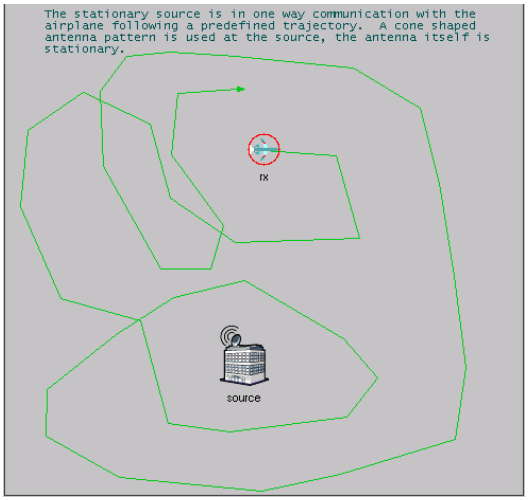


图 9-10　无线轨迹设定的效果

如果界面定义的方式不够精确，也可以直接打开文本编辑器来编辑*.trj 文件。或者觉得 OPNET 一拉一点定义轨迹效率低，也可以自己编写程序设置*.trj 文件。

根据两个位置点之间的运动时间是否固定又可再细分成两种：

1．每段间隔时间是固定的：

无论这两点的距离是多少，两点之间的运动时间是相等的。

2．每段间隔时间不等：可以设置轨道端点间的移动速度，另外还可以设置高度，停留时间（节点在向下一点运动前，在该点停留的时间）以及运动时间（前一点到当前点的运动时间）。

定义分段轨迹的步骤：

（1）打开项目编辑器，选择 Topology→Define trajectory...，出现对话框，可以选择定长时间间隔的，也可以选择不定长时间间隔的。

（2）在 Altitude 或者 Initial Altitude 字段中，输入高度或者初始高度值。

（3）（只适用于变长时间间隔运动轨迹）在 Intial Wait Time 字段中填入初始等待时间。

（4）在 Time Step 域内设置每段的运动时间（只适用定长时间间隔运动轨迹）。

（5）设置 Coordinates are relative to object's position，如果该框被选中，则 x，y 坐标是相对于初始位置的偏移。如果该框没有被选中，则该坐标被视为绝对值。

（6）单击 Define Path 按钮。

（7）定义运动轨迹上的点，如下所示：

- 定长时间间隔的轨迹：在一些点上，单击鼠标左键，完成定义整个运动轨迹后，单击鼠标右键，命名定义的轨迹文件的名字。
- 不定长时间间隔的轨迹：在第一个要定义的位置单击鼠标左键，出现运动轨迹段

信息对话框，在该对话框中输入相应的信息。

在 Traverse segment in/at 字段中输入上一点到该点的运动时间或者速度，然后输入该点的高度以及在该点的等待时间。

（8）最后，如果还想编辑定义运动轨迹，单击鼠标右键，编辑运动轨迹，出现如下对话框。

（9）选中节点，单击右键，选择 Edit Attribute，在 Trajectory 一项，选择定义好的运动轨迹。

9.2.2 设置向量（Vector）轨迹的方式

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零，物件就持续不断地运动，在仿真过程中也可以动态修改这三个参数，使物件运动状态发生改变。如图 9-11 所示，配置向量轨迹后节点左边将出现一个绿色的箭头线。

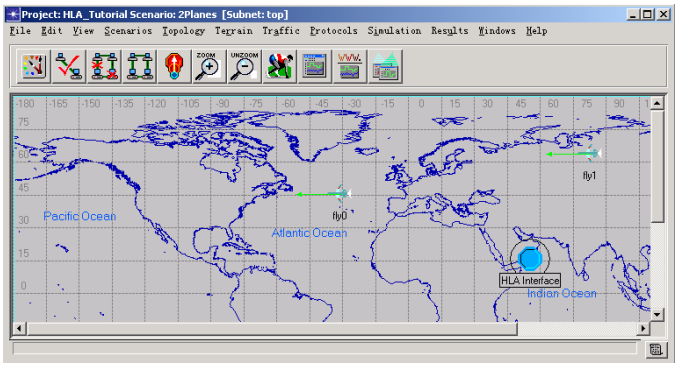


图 9-11 配置向量轨迹的效果

9.2.3 修改节点的位置属性

也可以采用直接将位置改动写入物件位置属性（x position、y position、altitude）的方式来使物件移动。

这种移动方式将使物件直接从一点跳转另一点，而不影响无线传播延时的计算，如果要考虑位置变化传播延时的影响，需要额外编写代码的支持。另一点需要注意的是，如果 altitude 设为 0，天线将不能发送数据包。

假设子网的左上角坐标为（X_MIN,Y_MIN），右下角坐标为（X_MAX,Y_MAX），移动物件在子网中平均移动距离为 x 方向 0.5*X_ MOBILE_SPAN，y 方向 0.5*Y_MOBILE_SPAN，则以下代码可以使物件按上述要求移动，注意移动距离的单位必须和子网范围度量单位相匹配。

```
static void move_node (void)
    {
    double  xpos, ypos, newx, newy;
    FIN (move_node ());
    op_ima_obj_attr_get_dbl (node_id, "x position", &xpos);
    op_ima_obj_attr_get_dbl (node_id, "y position", &ypos);
    do
newx = xpos + op_dist_exponential (X_MOBILE_SPAN) - 0.5*X_ MOBILE_SPAN;
    while ((newx < X_MIN) || (newx > X_MAX));
    do
      newy = ypos + op_dist_exponential (Y_ MOBILE_SPAN) -0.5*Y_ MOBILE_SPAN;
    while ((newy < Y_MIN) || (newy > Y_MAX));
    op_ima_obj_attr_set_dbl (node_id, "x position", newx);
    op_ima_obj_attr_set_dbl (node_id, "y position", newy);
    FOUT;
    }
```

我们可以专门设置一个进程模块定时地执行以上代码使物件等时间间隔地移动，在进程界面中激活 regular 中断（OPC_INTRPT_REGULAR），如果设置中断时间间隔为 60，如图 9-12 所示，则每分钟由仿真核心触发一次 regular 中断，使物件移动一次。

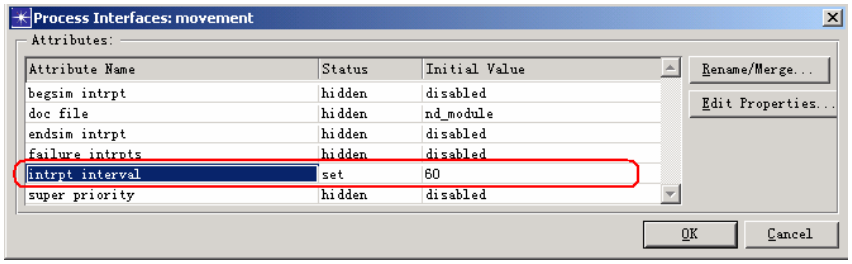


图 9-12　设置 regular 中断的时间间隔

9.2.4　使用移动配置器（Mobility Config）

该功能在 10.0.A PL2 版本首次出现。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。它能够加强物件移动的随机性，方便地设置移动速度、停留时间和范围。

图 9-13 为移动配置器的属性界面，可以自定义域的大小，在这个域中移动的设定才是有效的；接下来可以设置物件以何种随机方式移动，每移动一次暂停时间，移动速度多少。定义好移动参数后，选定感兴趣的物件，在 topology 中选择刚刚定义的移动主询（set mobility profiling...），之后在相应物件的属性栏中将增加这个属性。

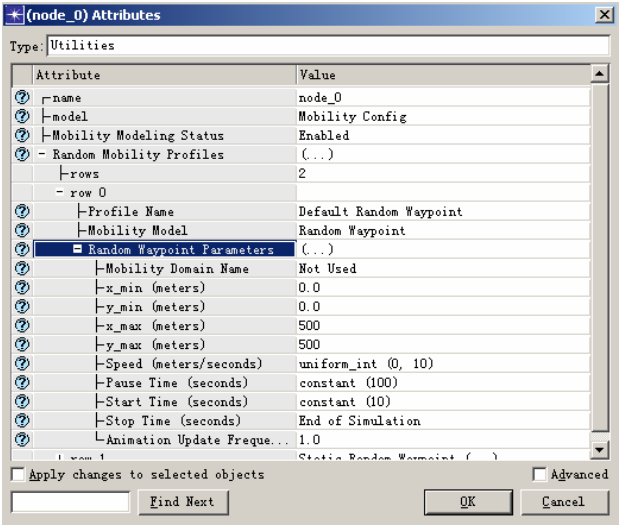


图 9-13　移动配置器属性对话框

9.3　无线收发机管道建模

对于无线系统和有线网络最大的区别是无线信道的广播和时变特性，以及结点的移动性。无线信道建模的内容涉及广泛，包括无线信道的频率、功率、视距以及干扰等。如果无线信道刻画的不准确，将直接影响到高层的性能以及仿真的精确性。

对于每个发射信道和接收信道对，它们之间的整个无线传输过程可以用一系列功能单一的子传输阶段的组合来描述，这些传输阶段是仿真无线链路所涉及到的一系列参数计算。有些无线链路的参数互为因果，时间上有先后顺序，所以传输阶段的排列顺序也应按照实际传输的先后来定。OPNET 无线仿真中用 14 个首尾相接的管道阶段（Pipeline Stage）来尽量接近真实地模仿数据帧在信道中的传输。首先在整个传输过程还没有进行之前，把肯定不能被接收的物件圈定出来；在计算传输延时后接着复制封包，对每一个接收主询中的物件都复制一份；然后计算接收闭锁，检查信道是否完全吻合，如果完全匹配当作有效信号，如果部分吻合当作噪声处理；对于接收器来说，在经历传播延时后，内部产生一个中断，对每一个可能接收的信道进行 6 至 13 阶段，由于包的每段有可能存在不同程度的干扰，因此对每一段都需要单独计算，如果是有效包则计算误码率，如果是噪声则考虑对有效包的影响；之后得到包的总误码数多少，最后决定是否丢包。

整个过程中计算数据保存在包的 TDA（Transmission Data Attribute）里，TDA 预设了一些值，如某个发信机和发信机信道的 Objid 等。这些值一共有 OPC_TDA_RA_MAX_INDEX，它是 OPNET 定义的象征性名字，代表 TDA 属性的最大索引号，如果需要自定义 TDA 属性，则将新属性定义为 OPC_TDA_RA_MAX_INDEX 加 1，

依此类推。

下面详细介绍无线链路建模的整个过程及其原理分析。

9.3.1　Stage 0：接收主询（收信机组）

该管道阶段确定候选的收信机对象，排除明显不符合的对象，把肯定不能被接收的物件圈定出来。在某些网络模型中，仿真内核可以判断发送接收信道对之间是完全不能进行通信的，如无线发信机的接收组中不应有本节点的无线收信机、点到点和总线收信机。

以上情况下，将接收信道从发信机信道的接收主询中移去可以加快仿真速度，因为这样减少包的复制和后续管道阶段的运行，而对实际仿真效果不产生影响。

具体来说，OPNET 为每对发射机和收信机都建立管道传输阶段，相对于每个收信机对，原始数据包都被复制了一次，由于包的复制是为每一对可能的收发信机之间建立管道阶段造成的，每个复制后的包需要经历 13 个管道阶段。即使对于模块内部的收信机对也不例外，如果在收信机组中没有将自己的收信机从接收组中删除，而且又能经过信道匹配（channel match）阶段，那么自己发出的数据包也会被自己接收到。这种情况显然不是我们所期望的，因此在 OPNET 中一些标准的接收主询管道阶段程序都将这种情况排除了。

仿真内核将发送和接收信道的 Objid 传给管道阶段程。随后管道程序返回一个整数值（OPC_TRUE 或 OPC_FALSE）给内核；该数值表明了收信机信道是否为一个合适的目的端，是否应当包含在收信机组中。

TD 类核心函数可以用来改变默认的收信机组属性，以及针对仿真事件动态地改变和重新计算收信机组。例如，如果在仿真过程中收信机节点被屏蔽掉，可以从接收组中移除该接收信道。使用函数 op_radio_txch_rxgroup_compute()可以在仿真中的计算给定信道收信机组，实质上，这将重新调用收信机组管道阶段。

收信机组阶段也可以完全“跳过”，只要将“rxgroup model”设置为“dra_no_rxgroup”（默认的值为“dra_rxgroup”），但是跳过并不是完全不用该管道阶段，而是动态更新收信机组，正如 9.4 节所述，这样可以加快仿真运行速度，尤其针对无线业务负载量大的网络仿真。

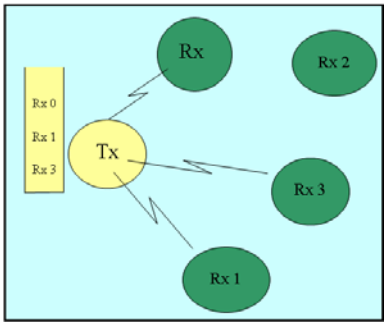


图 9-14　只针对接收主询内的节点发送封包

如图 9-14 所示，Tx 节点传输数据包之前，查看接收主询列表包含 Rx0、Rx1 和 Rx3，而 Rx2 不在之内，因此将封包只作 3 次拷贝，分别发往 Rx0、Rx1 和 Rx3，而不发给 Rx2。

9.3.2 Stage 1：传输时延

传输延时是数据包在无线链路中所经历的一部分延时，它是包按信道速率发送所需要的时间。这个时间是数据包的第一个比特开始发送时间和最后一个比特发送时间之差，也是发信机处理数据包所用的仿真时间。这个过程信道处于忙状态。当该事件发生时，媒体接入层的数据包将在队列中等候，直到信道空闲才可发送下一个数据包。如图 9-14 所示，计算传输时延可以通过下列公式：

传输时延＝数据包长度/数据传输速率。

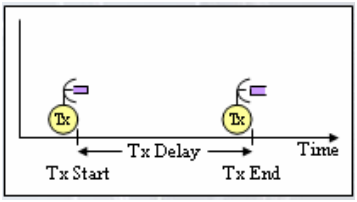


图 9-15　传输时延的计算

计算传输时延是整个管道阶段的第一个阶段，对于每个封包该阶段只计算一次，结果写入封包 TDA 的 TX_DELAY 属性中。

9.3.3 Stage 2：物理可达性（链路闭锁）

通过无线链路物理可达性（closure）的判断可以加快仿真运行速度，它的作用与收信机组管道阶段有点类似。如果是链路有任何阻碍，封包将被丢掉，后续的管道阶段不必计算，否则将 PROP_CLOSURE 设为 OPC_FALSE，表示没有阻碍。

无线链路的物理可达性计算依据视通性来决定，不用 TMM 地形模块只有三种判断，如图 9-16 所示。基于物理上的考虑，测试连接发信机与收信机之间的连线是否被障碍物遮挡。如果发信机与收信机之间物理上不可达，则数据包传输失败。仿真中可以配置特定的地形图，不同的地形地貌计算的结果也不同。如图 9-16 所示，可以分别读取发射电台和接收在地心坐标系统上的坐标，计算出连通向量，来判断是否和地球表面或障碍物相交。查看收发双方是否有视线连接的可能，有则认为可以连接，判断基于地球是完全理想的球形的假设，地球半径取 6378000 米。

（1）得到发射天线和接收天线的坐标，进而计算出天线之间的连通向量；

（2）读入地形参数，采用如图 9-16 所示方法判断天线之间的连通向量是否被地面遮挡。

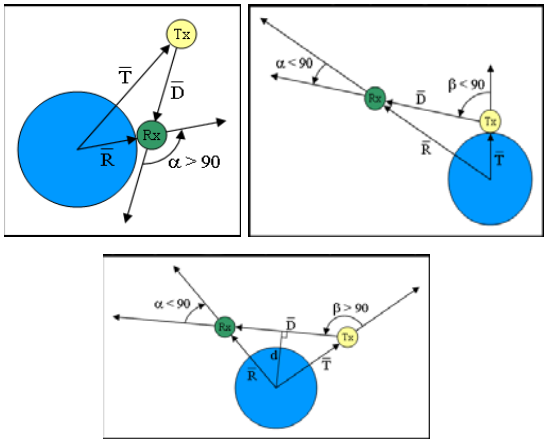


图 9-16　判断地形物理可达性的算法

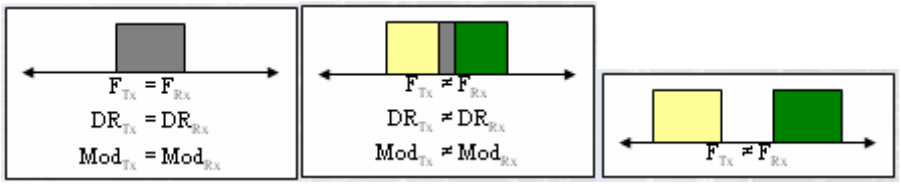
9.3.4　Stage 3：信道匹配

根据发射电台和接收电台的频率、带宽、数据速率、扩频码等 4 个属性来判断信道是否匹配，为正在传输的数据包分成三类，如图 9-17 所示。

（1）有效数据包（valid）：接收电台和发射电台属性完全匹配，接收电台能够正确接收并解码当前传输的数据包。

（2）干扰数据包（noise）：带内干扰，发射电台和接收电台的频率和带宽等属性有重叠部分，该数据包不但不能被正确解码和利用，而且对其他数据包的接收产生干扰。

（3）可忽略的数据包（ignore）：带外数据包，频带不交叉。即收信机的频率和带宽等属性和发射电台完全不一致，该数据包虽然不能被正确解码和利用，但是不会对其他数据包的接收产生干扰，会被仿真核心销毁。



完全吻合　　部分吻合　　频率完全不对应

图 9-17　信道匹配计算后的三种情形

9.3.5　Stage 4：发射机天线增益

发射天线增益刻画了发射信号能量被放大或衰减的现象。发射功率的“整形”是基于

天线结构的物理特性以及发送的方位角。由于天线在各个方向上对进行传输的数据包功率衰减程度不一样，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。无线链路仿真时将考虑天线模型的影响。例如，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。

实际仿真中，要得到天线的增益必须首先获得天线仰角（lookup_phi 和 lookup_theta）。天线仰角的计算涉及到坐标的变换，坐标的变换取决于天线基准点（boresight point），首先将天线模型坐标系的 Z 轴旋转至对准基准点，然后将发射电台与接收电台的连通向量投影到旋转后的天线模型的坐标系上，得到天线的仰角，进而查找获得该链路方向上的天线增益值。

发射电台与接收电台的连通向量（即发送端与接收端的夹角）可以根据天线的位置（天线属性含有位置信息：target latitude、target longtitude、target altitude）来计算。当节点移动时，需要人为对这三个属性进行更新。根据这三个物理位置属性，仿真核心会自动更新当前的天线坐标（phi_point，theta_point），管道程序可以通过以下语句读出这两个属性：

```
point_phi = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_PHI_POINT);
point_theta = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_THETA_POINT);
```

天线的基准点方向即为天线模型的主瓣指向（也即是天线模型增益最强的方向），基准点方向默认值为：boresight_phi＝ 0，boresight_theta＝180。基准点的值也可以在天线的属性“pointing ref. phi”和“pointing ref. theta "中设定。OPNET 也提供两个常量来存储这两个值，分别为 OPC_TDA_RA_TX_BORESIGHT_PHI 和 OPC_TDA_RA_TX_BORESIGHT_THETA。

9.3.6 Stage 5：传播延时

传播延时和传输延时对应，传播延时是数据包在无线链路中所经历的另一部分延时。在无线链路仿真中，考虑到无线电台的移动，在数据包传输过程中，发射电台和接收电台之间的传输距离可能发生变化。因此，需要计算两个时延，即传输开始时的传播时延和传输结束时的传播时延，来逼近节点的移动特性，如图 9-18 所示。

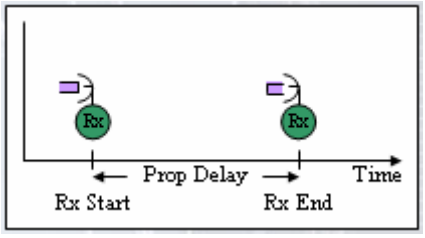


图 9-18　传播延时的计算

采用以下方法计算传播延时：

（1）分别计算发射电台和接收电台之间传输开始和传输结束的距离。

（2）传播开始时延=传输开始时收发机间距离/电磁波传播速率。

可以通过 od_td_set_dbl(pkptr，OPC_TDA_RA_START_PROPDEL,start_prop_delay)设置。

（3）传播结束时延=传输结束时收发机间距离/电磁波传播速率，可以通过 op_td_set_dbl(pkptr，OPC_TDA_RA_END_PROPDEL，end_prop_delay)设置。

（4）传播时延为传播开始时延和传播结束时延的折中。

该阶段的数据也用来计算多普勒频移。如公式（1）所示：

$$V_{relative}=\frac{d_{end}-d_{start}}{D_{tx}}$$

$$F_{shifted}=\left(\frac{V_{relative}+V_{propagation}}{V_{propagation}}\right)*F_{unshifted} \tag{1}$$

9.3.7　Stage 6：收信机天线增益

该管道阶段必需等到传播延时完毕才执行，间中仿真核心将转移去处理其他事件。收信机天线增益和发射天线增益的计算方法完全相同。结果将写入 TDA 下的 RX_GAIN 属性中。

9.3.8　Stage 7：接收功率

接收功率是有效的数据包到达接收电台的有效功率。接收功率仿真通过以下步骤：

（1）根据发射电台和接收电台的基准频率和带宽，得到收发电台互相重叠的带宽。

（2）由频率计算发送波长，再根据无线传播的距离，计算自由空间的电磁波功率传播损耗。

在无线路径损耗计算中考虑地形的影响，可以增加计算传播损耗、信号强度和噪声等参数的准确性。

基于地形仿真可以提供在给定地形条件下最佳频率预测；各种战术部署、调遣时的通信状态的预测；在给定地形条件时，给出通信节点的部署位置的可行建议；对通信偶然性计划的建议。

基于地形建模的传播损耗包括两种计算模型，分别是自由空间模型和 Longley-Rice 模型。

① 自由空间模型

自由空间模型当作天线处在真空之中，不考虑任何大气的影响，不被障碍物遮挡，以

下是自由空间传播损耗计算方法：

$$\lambda = \frac{C}{\text{Bandwidth}} \qquad \text{传输损耗} = \frac{1}{4\pi \times \text{距离}^2} \times \frac{\lambda^2}{4\pi} \tag{2}$$

② Longley-Rice 模型

Longley-Rice 模型基于 Longley 和 Rice 两个学者发表的论文。在此模型下，首先获取传输电台和接收电台天线间的地形规格（离地面的海拔），地面海拔来估计天线的实际高度。根据天线高度和地形参数计算无线电波的射、地面反射。基于上面的自由空间传播损耗公式，综合考虑、地面反射和地形衍射，计算路径损耗。

在模型中有些限制：

1．无线频率必须在 20MHz~20GHz 之间；

2．天线高度必须在 0.5m~3000m 之间；

3．天线的仰角必须小于 0.2 弧度。

（3）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。图 9-19 为接收功率计算公式的组合。

$$L_p = \left(\frac{\lambda}{4\pi D}\right)^2$$

$$\lambda = \frac{C}{f_c}$$

$$P_i = \frac{P_{tx}\left(f_{\max} - f_{\min}\right)}{B}$$

$$P_{rx} = P_i G_{tx} L_p G_{rx}$$

L_p = Pathloss
λ = Wavelength
D = Distance
C = Speed of Light
f_c = Center Frequency
P_i = Inband Frequency
P_{tx} = Transmitted Power (watts)
$f_{\max}$ = Maximum Frequency (Hz)
$f_{\min}$ = Minimum Frequency (Hz)
B = Bandwidth
P_{rx} = Received Power (watts)
G_{tx} = Transmitter Antenna Gain (watts)
G_{rx} = Receiver Antenna Gain (watts)

图 9-19　接收功率计算公式

抗干扰接收方式（信号锁和功率锁）

信号锁（signal lock）指收信机认定先到达的包是应该接收的包，而在这个包的接收期间，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。

信号锁是管道阶段引入的一个内部变量，来防止接收信道同时正确接收多个数据包。其值为一个布尔变量，表示信道的忙、闲。取真时，表示当前信道正在接收一个有效的数据包，而且该数据包可能被正确接收；当取假时，表示当前信道没有数据包到达（或者正在处理中，可以准备接收新的数据包），或者表示当前信道接收的数据包都不可能正确（在信道匹配阶段就设置过的）。

总之，设计自己的管道阶段时，在接收数据包时，先判断信号锁值，若为真，那么当前接收的包就只能是噪声了；若为假，当前接收的包送给就有可能被正确接收，交给后面

继续处理，这时再将信号锁值设置为真。

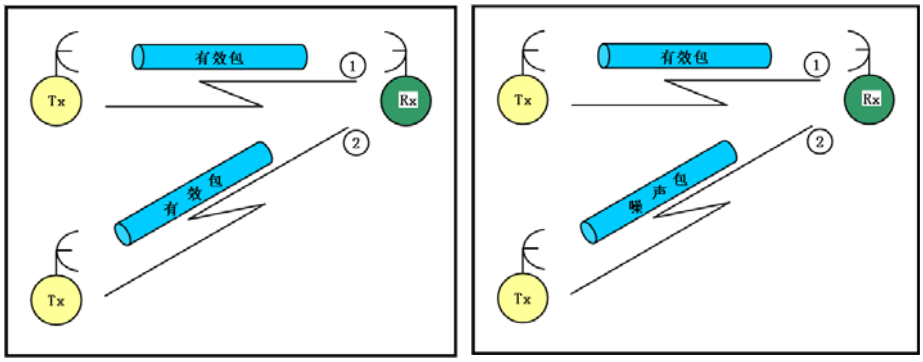


图 9-20　接收有效封包过程中另一个包到达

如图 9-20 所示，接收的时候两个封包同时抵达，根据信号如果第 1 个包先到，则第 1 个封包信号被锁定，第 2 个则被当作是噪声而不管它是有效的还是噪声的。

功率锁（power lock）指收信机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。

不管有效包还是噪声包，最后接收功率将写入 RX_POWER 属性中。

9.3.9　Stage 8：干扰噪声功率

干扰噪声功率描述了同时到达接收信道的各个数据包间的互相影响。如果有效数据包到达目的信道的同时另一个数据包正在接收，或者数据包正在被接收同时另一个数据包到达目的信道，则有干扰发生。

在大多数情况下，以上两种情形可能在一个数据包接收过程中出现多次。在接收过程中，对所有的干扰功率，结果施加至接收数据包。虽然背景噪声功率对于每个包的传输来说，只估算一次，但是干扰噪声功率却可能要计算多次。如果有多个数据包互相干扰，则干扰功率需进行累加。

计算所有碰撞对一个帧产生的干扰噪声。当两个帧发生碰撞时需要计算相互干扰。如果两个帧都是合法帧，则分别将对方的接收功率加到自己的干扰累计中，如果是噪声帧则将其接收功率加到对方的干扰累计中。如果一个帧和多个帧发生碰撞则这一过程要被触发多次，并将导致后面三个过程也被触发多次。

如图 9-21 所示，干扰噪声的产生有三种情况，接收有效包时来了另一个有效包；接收有效包时来了一个噪声包；接收噪声包时来了一个有效包。最后将总的干扰和冲突次数分别写入 TDA 属性 NOISE_ACCUM 和 MAX_COLLS 中。

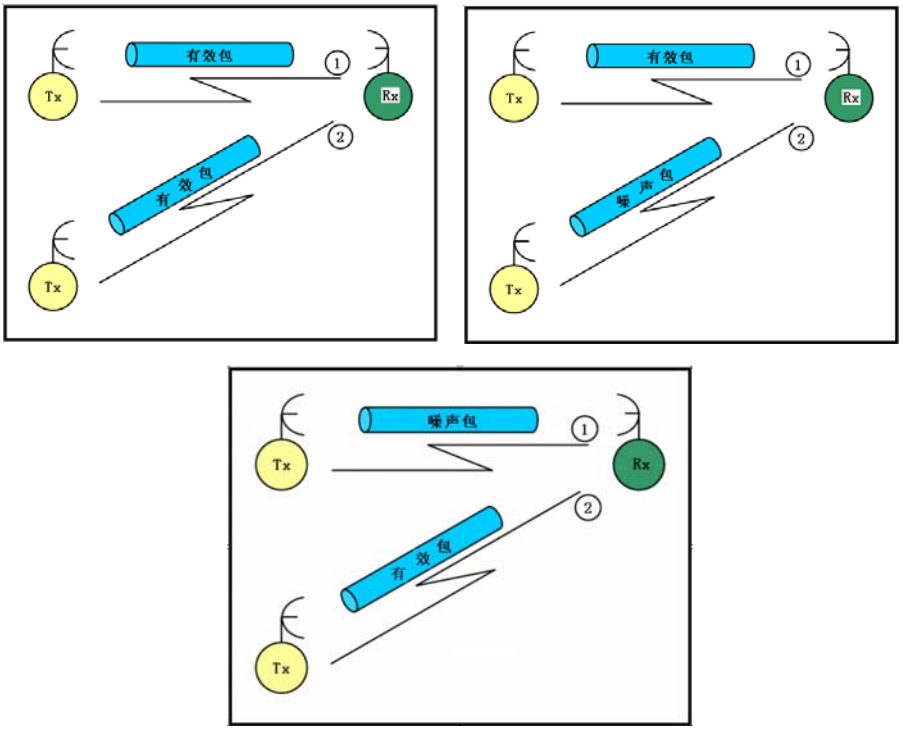


图 9-21　干扰噪声产生的三种情形

9.3.10　Stage 9：背景噪声功率

典型的背景噪声源包括了从临近电子元件或者无线电发射的热噪声或射电噪声（例如，车载无线电台、干扰电台、电视或其他器材噪音造成的影响等，下雨天或其他天气情况的影响），背景噪声功率建模如下：

（1）背景环境噪声：环境噪声功率=带宽*功率谱密度；

（2）背景热噪声：累计热噪声功率=带宽*波尔兹曼常数*（背景温度+设备温度）；

（3）背景噪声功率=环境噪声功率+背景热噪声功率。

图 9-22 为背景噪声功率计算公式描述。

背景噪声计算热噪声和环境噪声，通过计算将收信机温度转化成等效噪声，加上环境噪声，取落在带内的部分作为背景噪声。

9.3.11　Stage 10：信噪比

根据前面计算获得的接收功率、背景噪声和干扰噪声等参数来计算 SNR。数据包的 SNR 值是一个重要的性能度量来判断接收电台是否正确接收到包的内容。在一个数据包的整个

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（segment）的信噪比是相同的。如图 9-23 所示，对于第一个子图，从收信机角度来看，接收第一行封包时，第二行封包到达形成干扰，之后，第三行干扰封包到达，这时干扰噪声功率重新计算，当第二行封包接收完毕干扰功率又不一样，等到第三行封包也接收完毕才没有干扰。因此对封包的每一段需分别计算 SNR，然后计算出累计信噪比。对于其他两个子图的分析类似。

$$T_{rx} = (NF - 1.0) * 290.0$$
$$T_{bk} = 290.0$$
$$k = 1.379\mathrm{E}^{-23}$$
$$N_b = (T_{rx} + T_{bk}) B_{rx} k$$
$$N_a = B_{rx}\left(1.0\mathrm{E}^{-26}\right)$$
$$N = N_b + N_a$$

NF = Noise Figure
T_{rx} = Receiver Temperature
T_{bk} = Background Temperature
k = Boltzmann's Constant
B_{rx} = Receiver Bandwidth
N_b = Background Noise
N_a = Ambient Noise
N = *Noise*

图 9-22　背景噪声功率计算公式

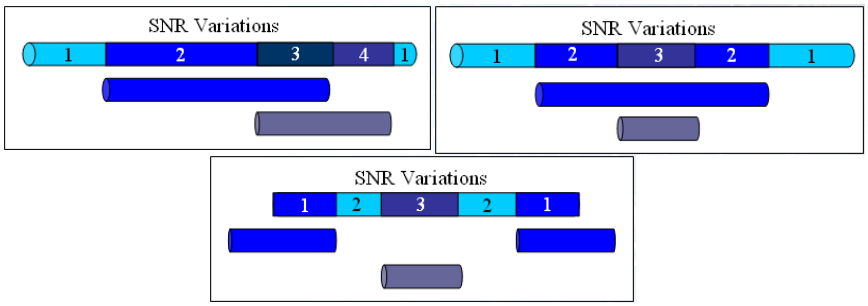


图 9-23　划分包段计算 SNR 的三种情形

数据包的信噪比计算公式如图 9-24 所示。另外还要加上处理机增益来计算有效的信噪比。

$$SNR_{actual} = 10\log_{10}\left[P_r / (P_b + P_i)\right]$$

P_r = Received Power (watts)
P_b = Backgroun Noise (watts)
P_i = Interference Noise (watts)

$$SNR_{\text{effective}} = SNR_{\text{actual}} + G_p$$

G_p = Processing　Gain

图 9-24　信噪比计算公式

9.3.12　Stage 11：误比特率

目的是根据 SNR 值得到比特错误概率。为了精确仿真无线链路的误比特率特性，BER

的计算不是基于整个数据包的，而是基于数据包中的一个一个小段来计算的。因为在数据包的传输过程中，信噪比不是固定不变的，从而导致 BER 也不是固定不变的。

根据调制曲线模型参数和前面传输阶段得到的信噪比，再加上信道处理增益（Processing gain）得到有效信噪比，进而查找调制曲线得到误码率。因为包的每段 SNR 可能不同，对每一段找出 BER 之后，最后叠加得出总的误码率。Modeler 10.0 版本支持动态调用调制模块，op_tbl_modulation_get 可得到当前需要的调制模块。

9.3.13 Stage 12：错误分布

根据前一阶段得到的数据包的每一段的误码率，即可计算出数据包每一段数据中的误码数目。然后将它们累积起来即可得到总的误码数目。

一个封包的位差错并不是每一个位模拟，只是随机选择差错位。如图 9-25 所示，根据前面得到的误比特率 BER，计算有 k 个比特数错误的概率，再和 0～1 间的随机数比较，如果大于这个随机数，则给包分配 k 个比特错误。

$$P_k = p^k (1-p)^{N-k} \binom{N}{k}$$

$$\sum_{k=0}^{N} P_k \geq r$$

P_k = Probability of k Errors
p = Probability of Error
N = Packet Length (bits)
r = (0～1间的随机数)
k = Number of Errors

图 9-25 错误分布计算公式

从上次计算信噪比、误码率到激发此过程之间传送的信号段中按误码率随机产生 i 个误码，加到误码个数累计。方法是以误码率为每一位产生错误的几率 P，在 N 个位中产生 K 个错误的可能性是

$$P(K) = P^K (1-P)^{N-K} C_N^K$$

每次激发错误分配时计算一个随机量 R，从 1 到 N 逐个判断

$$R < \sum_{K=0}^{i} P(K)$$

是否成立，成立则本次分配 i 个错误。

9.3.14 Stage 13：错误纠正

该结果描述数据包经历了碰撞和背景噪声干扰后的纠错能力，根据帧长度、收信机错误纠正门限设定和最后得到的误码个数，决定此帧是否能被接受。 如果判断能够接收当前数据包，则允许其被继续发送到高层。此阶段的判断直接影响接收信道的数据包丢弃率和吞吐量结果。

基于仿真中考虑到无线电台设备可能被关闭或摧毁，数据包是否可以接收的判断标准有两个，一是源端是否完整发送数据包；另一是比较误比特数是否小于收信机的纠错门限值。如果设备被关闭或摧毁则数据包接收失败；进一步根据误码数目和纠错门限来判断，如果误码数目小于纠错门限则数据包可以被接收，将 PK_ACCEPT 属性置为 OPC_TRUE，否则销毁包。到此包接收完毕，之后把信号锁（signal lock）解开，准备收下一个包。

9.4　加快无线仿真的速度

由于无线的广播传输方式，封包大量被复制，仿真时间急剧增加，因此如何加快无线仿真的速度是个非常重要的话题。

9.4.1 采用优化的仿真核心

首先我们运行一个无线场景来比较两种仿真核心 Development 和 Optimized 速度上的差别，如表 9-1 所示。

表 9-1　两种仿真核心速度比较

仿真核心	仿真时间	事件速率	事件总数
Development	58s	1819	105429
Optimized	44s	2386	105429

我们可以看到事件总数不变，而事件速率提高。由于 Development 仿真核心产生许多调试信息，当开发的模块工作稳定时就可以直接采用 Optimized 仿真核心，这样调试信息就会省略掉，从而提高仿真速度。

9.4.2　在仿真中动态删减接收主询成员

如我们可以在默认的信道匹配管道阶段程序 dra_chanmatch 中加入如下代码，将完全不匹配的信道从接收主询中去除。再次运行上述场景，仿真时间减少到 29s，总的事件个数为 105424，少了 5 个事件。

```
if ((tx_freq > rx_freq + rx_bw) || (tx_freq + tx_bw < rx_freq))
    {
    Objid tx_ch_objid, rx_ch_objid;
    op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS, OPC_
```

```
TDA_RA_MATCH_IGNORE);
    tx_ch_objid = op_td_get_int (pkptr, OPC_TDA_RA_TX_CH_OBJID);
    rx_ch_objid = op_td_get_int (pkptr, OPC_TDA_RA_RX_CH_OBJID);
    op_radio_txch_rxch_remove (tx_ch_objid, rx_ch_objid);
    FOUT
    }
```

9.4.3 简化无线封包的复制

由于无线管道阶段只对封包的包头设置参数，而对包的主体（body）不产生任何影响，因此可以将整个封包的复制简化为只复制包头。

在 Edit-> preference 下找到 sim_packet_sharing 属性，如果该值是 disabled，则复制整个包，如果设定为 conservative，则只复制包头。

9.4.4 动态更新接收主询

在特殊工具（utility）的物件拼盘中，有一个称为接收主询配置（receiver_group_config）的物件，如图 9-26 所示。通过设置其信道匹配（Channel Match）、距离门限（Distance Threshold，多少米以外不计入接收主询）和路径损耗门限（Pathloss Threshold，低于某个功率不计入接收主询）属性可以使可能的接收主询范围更加明确，从而有效过滤了一些无关的接收主询。考虑到物件的移动性对接收主询可能带来的改变，例如当物件移动后超出范围，可能需要将其从其他移动台接收主询中删除，通过设置 Refresh interval 来指定更新接收主询信息的时间间隔。

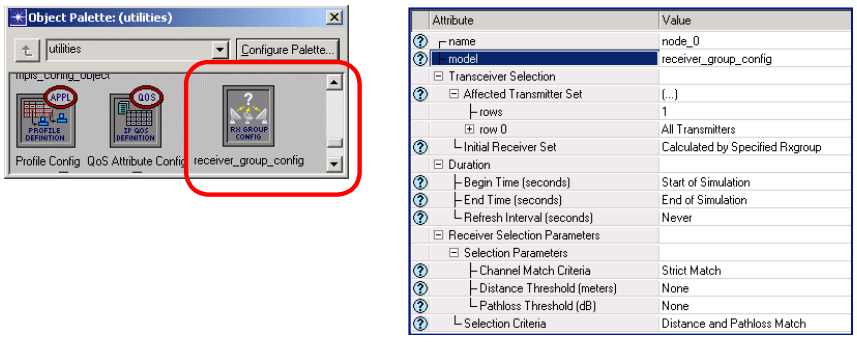


图 9-26 接收主询配置器及其属性

9.4.5 通过无线区域（wireless domain）划分接收主询

OPNET 提供了一种编辑无线区域模型（Wireless Domain Model）的新编辑器，它提供

了一些特殊的编程块，同时在物件拼盘配置对话框中也增加了相应的配置按钮，如图 9-27 所示。

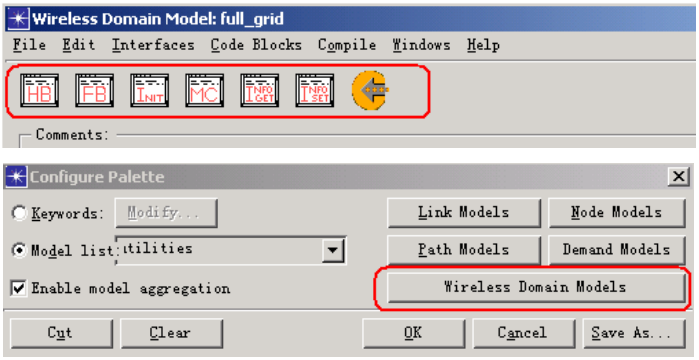


图 9-27　无线区域模型及其在物件拼盘中的配置按钮

下面举例说明如何通过无线区域划分接收主询，如图 9-28 所示的区域中，均等地分成 6 个小方块。假设最开始 T1 与 R1 传输数据，将计算一些如链路闭锁、信道匹配、传播时延和功率等信息暂时储存起来，在第 2 次 T1 与 R1 传输数据时就不需要再计算这些信息。进一步，在同一对方块中的节点通信也可以调用暂存的信息，如 T1 与 R2 之间通信，因为 R1 与 R2 都同处在第 1 个方块内，因此可以用储存的结果计算。同理，假如 R3 移到方块 1 也可以用储存的结果。

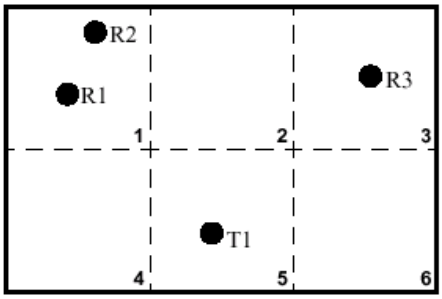


图 9-28　以空间划分接收主询

以上是根据空间来划分主询的方法，还可以根据同种性质来划分，例如把所有同样频率的接收机化为相同主询。OPNET 并没有提供一拉一点就能实现上述功能的设置，需要在无线区域模型的程序块中编写相应的代码

9.4.6　过滤无关的管道阶段

在不影响仿真精确度的前提下，有时我们并不想让无线传输经历所有的管道阶段，则可以将某些管道阶段设置为 NONE，如图 9-29 所示，则相应的管道阶段将跳过不执行从而

减少仿真时间。

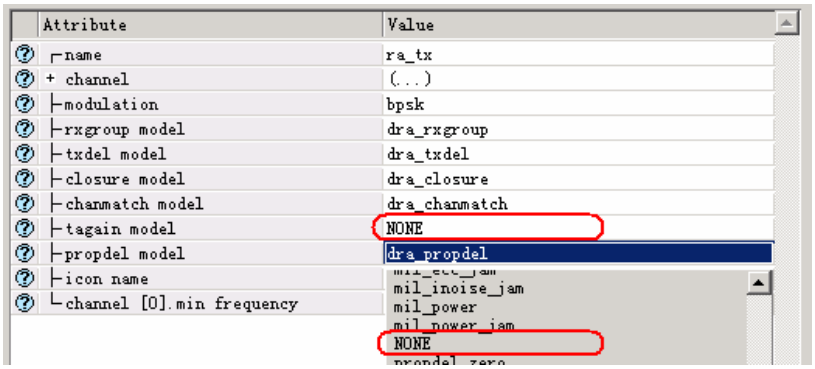


图 9-29　将管道阶段设置为 NONE

9.4.7　采用并行仿真

并行仿真允许使用使用多台计算机的多个处理器同时运行。为了加快无线传输的仿真速度，可以在管道阶段程序名称后面加上后缀_mt，并且程序开始以 FIN_MT 标识，如图 9-30 所示，将调用并行仿真机制，处理器自动分配处理带宽给每个线程，同时验证多进程的安全性，不会造成冲突。另外进程模块也有类似的选项。

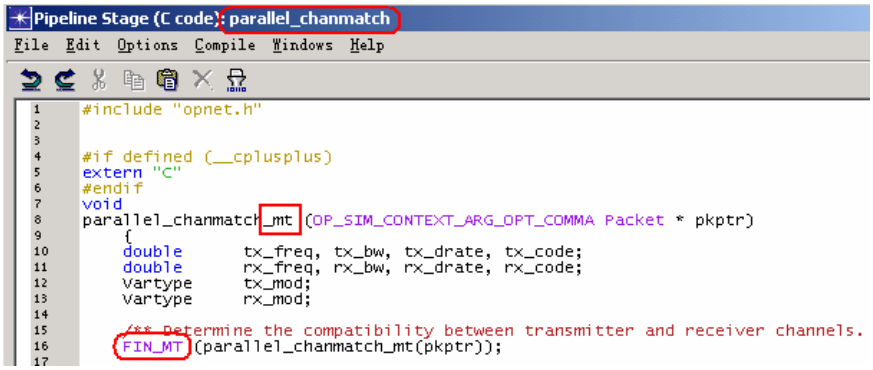


图 9-30　激活并行仿真必须在管道阶段程序中的设定

9.5　创建一个移动无线网络

OPNET 无线模块支持地面和卫星无线系统的构建。在此例程中我们将构建一个简单的无线网络，它包含一个移动干扰节点和两个固定基站。通过配置运动轨迹，干扰节点可以移动，从而使网络拓扑结构动态地改变，可以观察这种改变对接收信号质量的影响。

同时，移动干扰节点产生的无线噪声干扰使信噪比（SNR）降低，为了改善网络性能，将采用有向天线来增强网络的抗干扰能力。因此，在该例程中还将用到天线模型编辑器创建一个有向天线模型。最后，通过实验将看到，当基站采用有向天线时，网络的 SNR 比采用全向天线有明显提高。

9.5.1　概述

关键概念　OPNET Modeler/Radio 具有为陆地和卫星无线系统建模的能力。本例程需要创建一个带有移动干扰节点的无线网络，通过本例程学习，你将学会:

- 使用 OPNET Modeler/Radio 创建一个无线网络并在一个动态网络拓扑结构的接收节点处观察由无线噪声引起的接收信号质量的变化。
- 使用一个新型的链路——无线链路和一个新型的节点——移动节点。
- 使用天线模型编辑器创建一个定向天线模型。
- 定义移动节点轨道。
- 使用参数化仿真。

在一个基于无线的网络中，干扰将对信噪比产生巨大影响。不同类型的天线，比如定向天线，可以改进一个特定网络的 SNR。本例程你的工作是构建一个简单的无线网络，该网络具有一个移动干扰节点和两个固定通信节点（发射节点和接收节点），然后显示当固定节点使用全向天线和定向天线时 SNR 的不同。

9.5.2　开始建立

本节讨论组成网络拓扑结构的各个独立模型。拓扑结构包括三个节点，如图 9-31 所示。

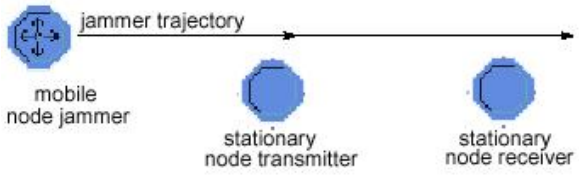


图 9-31　网络拓扑结构

- 发送节点——在各个方向以相同的强度发送数据。该节点包含一个包生成模块，一个无线发射机模块和一个天线模块。
- 接收节点——测量由固定发射机节点发送的信号质量。该节点包括一个天线模块，一个无线收信机模块，一个收信机处理器模块和一个与定向天线协同工作的附加处理器模块。
- 移动干扰节点——产生无线噪声。干扰器的轨道使得它可以在收信机的无线范围

内进进出出，从而增加和减小收信机收到的干扰。

关键概念 要创建每一个网络对象的节点模型，你需要使用节点模型编辑器中的几个新的对象，包括一个天线模块用于为定向增益建模；无线发射机和收信机模块，和一个处理器模块。

天线模块通过引用其模式属性为一个物理天线的定向增益建模。天线使用两种不同的模式：全向模式（在各个方向上具有均匀增益）和定向模式。

发射机模块以 1024bits/second 的速率向天线发送数据包，它将使用其 100%的信道带宽。

对于每一个到达的数据包，收信机模块参考几个属性来决定包平均误比特率（BER）是否小于指定的门限。如果 BER 足够低，包数据将被发送到收信机然后被销毁。

处理器模块（本课称为指向处理器）计算天线达到某一指定目标所需的纬度、经度和高度坐标（三维坐标）。指向处理器通过使用一个核心程序完成该计算任务，核心程序将一个子网中的节点位置（以 x 位置和 y 位置属性表示）转化为天线所需的球面坐标。

关键概念 无线链路动态存在于无线收发信道对之间，并在仿真时动态建立。（因此它们在任何一个编辑器中都看不到）

本例程中，信息将从一个固定发射机对象转移到一个固定收信机对象。这些对象通过一个无线信道连接。该链路由系统中各组成部分的许多不同的物理特性决定，包括频带、调制类型、发射机功率、距离和天线方向。

9.5.3 创建天线模型

关键概念 OPNET 的天线模型编辑器使用球面角 phi 和 theta 图形化地创建 3 维天线模型。

一个天线模型的 OPNET 表示可分为球面角 phi 和 theta 两部分值。常量值 phi 代表大致的二维圆锥形表面，该表面被映射到笛卡儿坐标中，并用一个称为片（Slice）或层（Plane）的二维函数来描述。对于每一个二维片（2D Slice），函数的横坐标为 theta，纵坐标为相应的增益值。这样，三维天线模型函数就被表示成一个二维层的集合，如图 9-32 所示。

每一层都用一个图形面板表示，该图形面板中，抽样点指定了与每一个 theta 的变动度数相对应的增益值。你可以使用 phi 平面操作菜单选择哪一个二维函数层（或 phi 值）将被用来作为编辑对象进行显示。

本例程将创建一个新的天线模型，该天线在一个方向的增益是 200dB，在其他任何方向的增益均为零（这是一个理想的选择性收信机）。

（1）从 File 菜单中选择 New…，然后从下拉菜单中选择 Antenna Pattern。单击 OK 按钮。

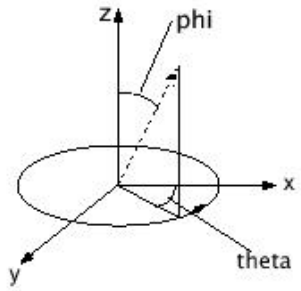


图 9-32　天线模型所用的坐标系

这时打开如图 9-33 所示的天线模型编辑器。

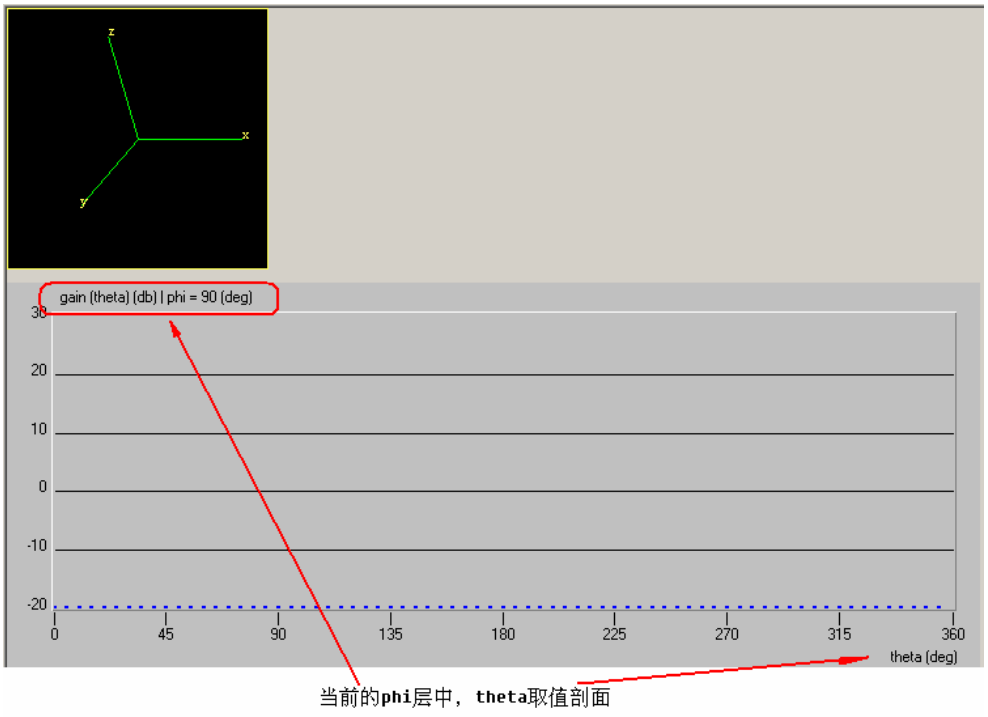


图 9-33　天线模型编辑器

本例程可以使用默认的 theta 的分割数（72，即每隔 5 度一个点），因此，能够用抽样点表示的最大的 theta 值为 355 度。对从零到 355 度的所有的 theta 值，可以指定增益大约等于 200dB 的抽样点。指定了图形面板中的任何两个抽样点，系统都会使用线性内插的增益值设置这两点之间的所有抽样点。因此，在该层中只需要设置两个抽样点：0 度点和 355 度点。就可以设置所有的 72 个点。

下面将当前层的设置调整为 5 度（360/72）：

（2）在项目工作空间中单击右键并从弹出的菜单中选择 Set Phi Plane。

这时弹出如图 9-34 所示的选项表。

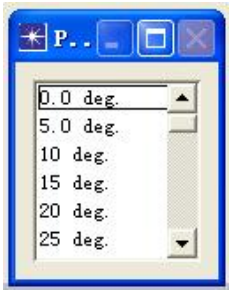


图 9-34　每个 phi 值对应一个层

（3）从该表中选择 5.0 Deg.。

这时选项表自动关闭，图形面板显示了层参数被设为 5 度时的二维层曲线。位于面板顶端的功能标签显示了当前 Phi 的设定（为 5 度），如图 9-35 所示。

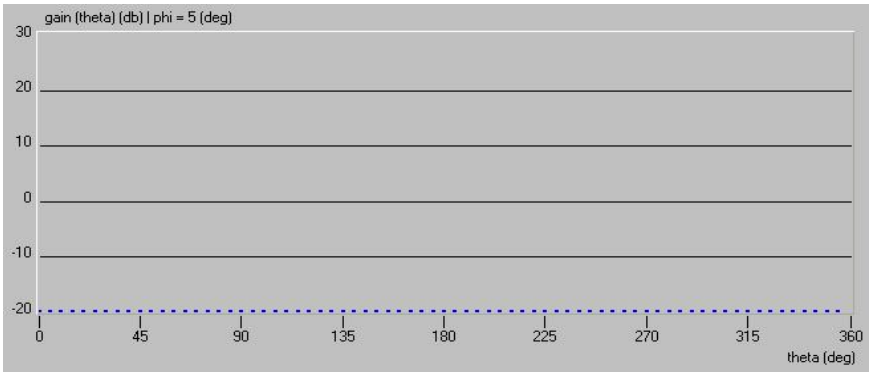


图 9-35　Phi=5(deg)层中 theta 的取值对应的增益值

下面设置纵坐标范围：

（4）单击 Set Ordinate Upper Bound 动作按钮。

（5）在对话框中输入 201 作为纵坐标上界，单击 OK 按钮。

（6）单击 Set Ordinate Lower Bound 动作按钮。

（7）话框中输入 199 作为纵坐标下界，单击 OK 按钮。

这时图形面板显示新的坐标范围。该范围允许我们更加精确地输入增益值。

既然已经正确设置了图形面板，就可以指定 phi=5 度时的抽样点。

（8）将鼠标移动到 200dB 线的最左端的点上，然后单击确定第一个抽样点（0 度）。将鼠标移到 200dB 线的最右端然后单击确定第二个抽样点（355 度）。

这时所有的介于这两个指定点之间的抽样点都使用线性内插增益值自动设置。然后出现一个点线，所示抽样点的范围，如图 9-36 所示。

当你在图形面板中定义点时，3 维投影视图区域将会显示一个锥形的外壳，来表示 phi 从 5 度到 10 度，theta 从 0 度到 360 度的增益，如图 9-37 所示。

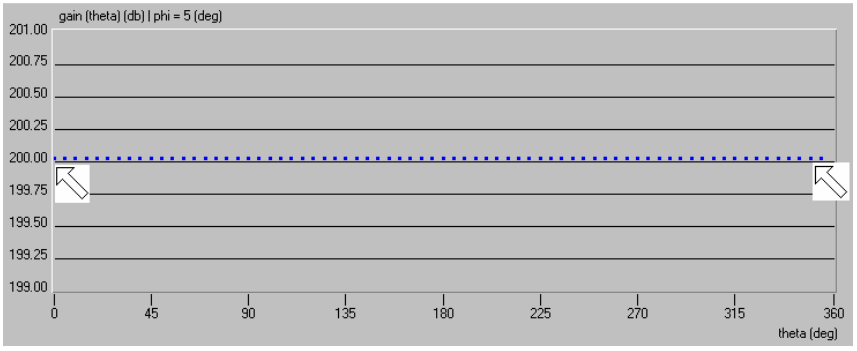


图 9-36　设置抽样点后的图形

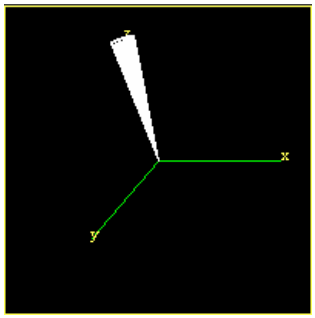


图 9-37　天线模型图

既然已经指定了 phi=>5 度的增益值，需要将层设置改为 0 度，然后设置该层的增益和抽样点。将 phi=>0 度~5 度和 theta=>0 度~360 度的增益值设为 200dB。该 dB 值将填充到由圆锥形外壳指定的 phi=>5 度的平面中。

（9）在工作空间中单击右键并从菜单中选择 Decrease Phi Plane。这时当前的 phi 平面设置从 5 度变为 0 度。

（10）将纵坐标的上边界设为 201，下边界设为 199。

（11）将鼠标移到 200dB 线附近（越近越好），然后在最左端（0 度点）处单击确定第一个抽样点，再将鼠标移到 200dB 线的最右端（355 度点）单击，确定第二个抽样点。

最后，在整个模式上归一化增益函数。

（12）单击 Normalize Function 动作按钮，在整个模式上归一化增益函数。这时，3D 投影视图将会刷新，显示归一化后的结果，如图 9-38 所示。模式中小的球面部分描述了旁瓣增益采样点，在归一化时用到了它们，以使旁瓣增益驱近于零。

注意：归一化使得图形面板中的点上移了，所以此时它们不可见。

（1）从 File 菜单中选择 Save…，将天线模型命名为<initials>_mrt_cone。

（2）关闭天线模型编辑器。

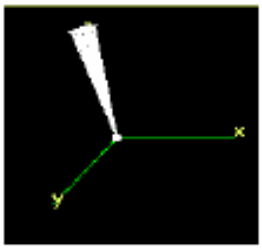


图 9-38　归一化后的天线模型图

9.5.4　创建指向处理器

关键概念　天线指向处理器计算发射机模块的位置，然后设置天线模块的目标属性。它不接收中断，因此，它可以被设置为一个独立的非强制的状态（Unforced State）。

（1）从 File 菜单中选择 New…，然后从列表中选择 Process Model，单击 OK 按钮。这时打开过程模型编辑器。

（2）使用 Create State 动作按钮，在工具窗口中放置一个状态。

（3）在该状态上单击鼠标右键，从弹出菜单中选择 Set Name。

（4）将状态命名为 point。

（5）然后，创建一个回到该状态自身的转移，为该状态输入代码。创建的转移如图 9-39 所示。

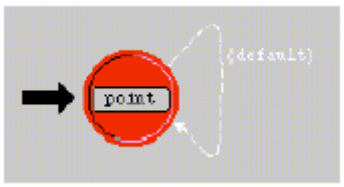


图 9-39　进程状态图

（6）在转移上单击鼠标右键，从弹出菜单中选择 Edit Attributes，然后将转移的 condition 属性改成 default。

（7）单击 OK 按钮关闭对话框。

关键概念　入口执行中的代码需要变量声明，但由于过程只被调用一次，仿真开始时，变量不需要保留它们的值。因此，变量可以存储在临时变量块中。

过程模型决定对象的身份，然后重新得到并修改对象的属性值。辨识（Identification）和拓扑工具包（Topology Packages）（前缀是 **op_id** 和 **op_topo**）中的核心程序执行第一项

任务。Ima 工具包（前缀为 **op_ima**）中的核心程序执行第二个任务。已经编写了两个不同的代码块（一个完成第一项任务，一个完成第二项任务）。仅需要把代码块导入（或者“读入”）模型中即可。

（8）单击 temporary variable block 动作按钮，从 File 菜单中选择 Import…。选择下面的文件，然后单击 OK 按钮导入此文件。

<reldir>/models/std/tutorial_req/modeler/mrt_tv

导入的文件出现在 temporary variables 编辑器窗口中。

（9）在编辑器窗口中，在文件结束的地方（最后一个空行）加上一个空格，然后保存文件。

然后，导入入口执行代码块的代码：

（10）双击 **point** 状态的上部，打开入口执行代码块，然后从 File 菜单中选择 Import…。选择下面的文件，然后单击 OK 按钮导入此文件。

<reldir>/models/std/tutorial_req/modeler/mrt_ex

导入的文件出现在 **point：Enter Execs** 编辑器窗口中。

（11）在编辑器窗口中，在文件结束的地方（最后一个空行）加上一个空格，然后保存文件。

继续之前可以看一下每一个块中的代码。

接下来，需要更改过程的属性：

（12）从 Interfaces 菜单中选择 Process Interfaces，出现 Process Interfaces 对话框。

（13）把 begsim intrpt 属性的初始值改为 enabled。

（14）将所有属性的 Status 值改为 hidden。

（15）单击 OK，保存更改。最后，编译过程模型：

（16）单击 Compile Process  动作按钮。当提示你保存模型时，命名为 <initials>_mrt_rx_point，单击 OK。

注意：如果模型没有编译，请参考建模概念手册中的解决问题章节。

（17）过程模型编译完后，关闭过程编辑器。编译过程中自动保存了过程模型，所以这里就不需要保存了。

9.5.5　创建节点模型

建立无线网络模型需要三个节点模型：一个发射机，一个收信机和一个干扰发射机节点。

1．发射机节点

发射机节点包括一个包产生模块，一个无线发射机模块和一个天线模块。包产生器产生大小为 1024bit 的包，包间隔时间为常数，平均速度为 1.0 packet/second（缺省值）。

产生后，包通过包流送到无线发射机模块，它将包发送到速率为 1024 bit/second、使用 100%信道带宽的信道上。包就经过发射机通过另一条包流到达天线模块。

天线模块使用全向天线模型（缺省值），表示在空间各个方向上增益相同。

要创建发射机节点模型：

（1）从 File 菜单中选择 New…，然后从列表中选择 Node Model，单击 OK 按钮。这时打开节点模型编辑器。

（2）按图 9-40 创建模块和包流，设置相应的模块名称。

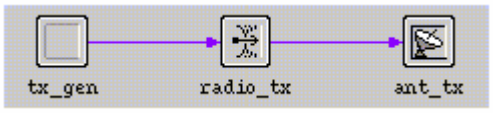


图 9-40 设置模块名称

（3）将 tx_gen 处理器的 process model 属性设置为 simple_source。

为了运行参数化仿真，需要将所用的信道的 power 属性提升。当提升了属性后，就可以在仿真运行时很容易地改变了。

（4）在 radio_tx 节点上单击鼠标右键，从弹出菜单中选择 Edit Attribute。然后单击 channel 属性的 value 字段。

出现一个对话框，显示了 channel 的复合属性表，如图 9-41 所示。

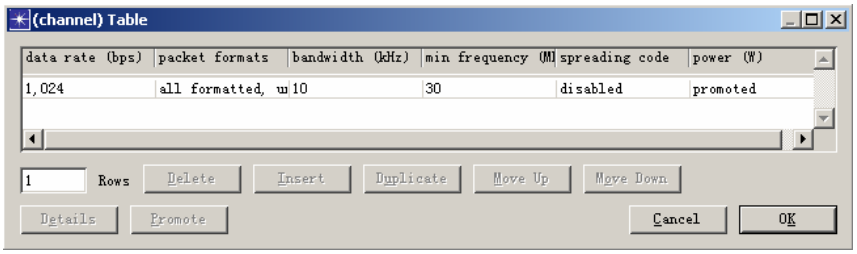


图 9-41 channel 复合属性

（5）在信道的复合属性表中，选中 power 属性，然后单击 Promote 按钮。这样就提升了 power 属性，如图 9-42 所示。

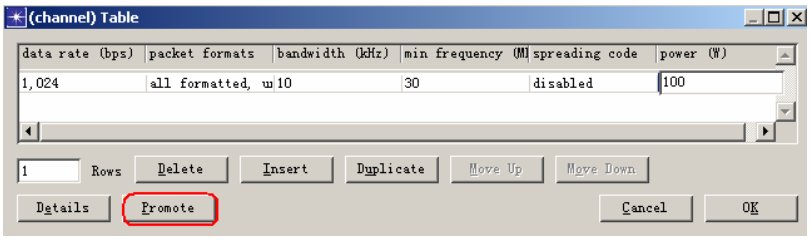


图 9-42 Channel 表

power 的属性显示为 promoted。

（6）关闭所有对话框。

接下来需要定义节点模型界面属性。

（7）从 Interfaces 菜单中选择 Node Interfaces

出现节点接口对话框，如图 9-43 所示。

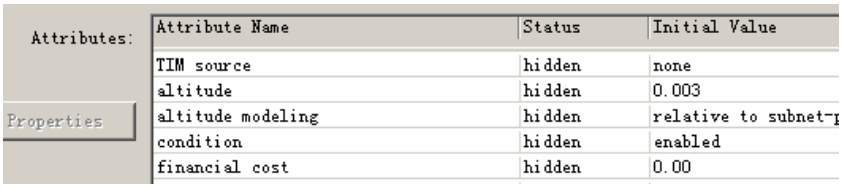


图 9-43　节点接口属性

（8）在 Node Type 表中，将 satellite type 的 Supported 值设为 no。

（9）在 Attribute 表中，将 altitude（高度）初始值改为 0.003。

（10）除了 radio_txchannel[0].power 属性之外，将所有其他属性的 Status 值设置为 hidden。

（11）在 Keywords（关键字）表中，加上 mrt。

（12）为便于参考，添加描述该节点的注释。

节点界面对话框设置完成后应该和图 9-44 中一致。

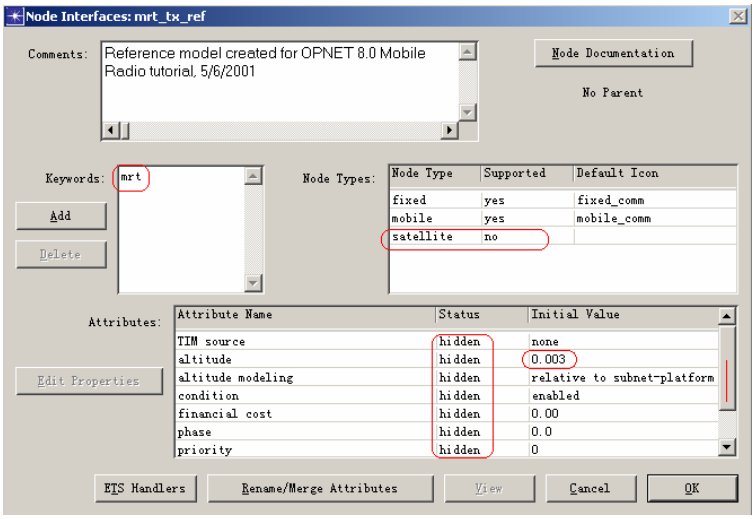


图 9-44　节点接口属性

（13）单击 OK，保存更改。

（14）保存节点模型。从 File 菜单选择 Save。命名为<initials>_mrt_tx。

2．干扰发射机节点

干扰发射机节点向网络中引入了无线电噪声和静止的发射机节点一样，它包含一个包产生模块，一个无线发射机模块和一个天线模块。它的行为和静止发射机的一致，但是信

道功率和信号调制方式不同。这些差别使得干扰发射机节点发送的包在收信机看来像噪声。干扰发射机节点模型可以从发射机节点模型（<initials>_mrt_tx）生成。

（1）打开<initianls>_mrt_tx 节点模型。

（2）在 radio_tx 节点上单击鼠标右键，从弹出菜单中选择 Edit Attribute。将 modulation（调制类型）属性改为 jammod。

（3）关闭 radio_tx 属性对话框。

（4）从 Node Interfaces 菜单下选择 Interfaces 菜单，修改描述干扰发射机节点的注释，然后保存改动。

（5）从 File 菜单选择 Save As…,将文件另存为<initials>_mrt_jam。

3．收信机节点

收信机节点包含一个天线模块，一个无线收信机模块，一个 sink 处理器模块以及一个指向处理模块，它的作用是让定向天线指向发射机。

（1）从 Edit 菜单选择 Clear Model。

（2）按图 9-45 创建模块和包流。设置相应的模块名称。

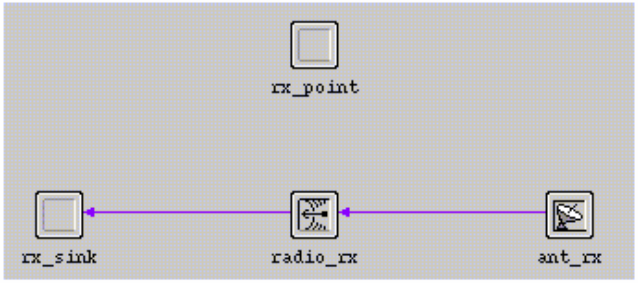


图 9-45　模块和包流图

注意：确定天线模块的名称为 ant_rx。<initials>_mrt_rx_point 处理模型用到了这个名称。

更改下列属性值：

（3）右击 rx_point 模块，打开属性对话框。将 process model 属性的值设成<initials>_mrt_rx_point。

（4）右击 ant_rx 模块，打开属性对话框。单击 pattern 属性的左边一栏，然后单击 Promote 按钮，将 pattern 属性提升，如图 9-46 所示。

（5）关闭所有对话框。

接下来定义节点模型的界面属性：

（6）从 Interfaces 菜单中选择 Node Interfaces。

（7）在 Node Type 表中，将 satellite type 的 Supported 值设为 no。

（8）在 Attribute 表中，将 altitude（高度）初始值改为 0.003。

（9）除了 ant_rx.pattern 属性之外，将所有其他属性的 Status 值设置为 hidden。

（10）在 Keywords（关键字）表中，加上 mrt。

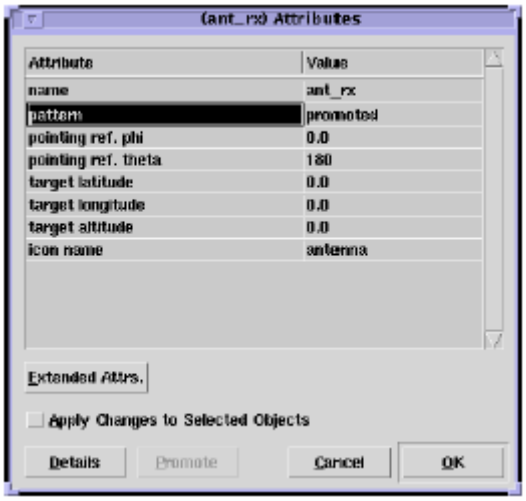


图 9-46　Ant_rx 属性

（11）单击 OK 按钮，保存更改。

（12）从 File 菜单选择 Save As…,将文件另存为<initianls>_mrt_rx。关闭节点模型编辑器。

9.5.6　创建网络模型

所需的节点和过程模型创建好之后，就可以创建网络模型了。

创建网络模型步骤如下：

（1）从 File 菜单中选择 New…，然后从列表中选择 Project，单击 OK 按钮。

（2）将新项目命名为<initials>_mrt_net，场景名命为 antenna_test。

（3）在启动向导中，使用以下设定值，如图 9-47 所示。

Dialog Box Name	Value
Initial Topology	Default value: **Create Empty Scenario**
Choose Network Scale	**Enterprise** ("Use Metric Units" enabled)
Enterprise Sizing Method	**Specify Size**
Specify Size	**10 km x 10 km**
Select Technologies	None
Review	Check values, then click **OK**

图 9-47　向导设置值

（4）在对象模板中，单击 Configure Palette…，然后清空面板并添加<initials>_mrt_jam，<initials>_mrt_tx，<initials>_mrt_rx 节点模型。将面板存为<initials>_mrt_palette。

请注意每个节点都有两个对象，一个固定的，一个移动的。

（5）关闭 Configure Palette 对话框，创建图 9-48 中的网络。确保使用干扰发射机的移动版，发射机和收信机的固定版。

（6）对于每一个节点：

- 打开属性对话框。
- 按图 9-48 所示设置 name 属性。

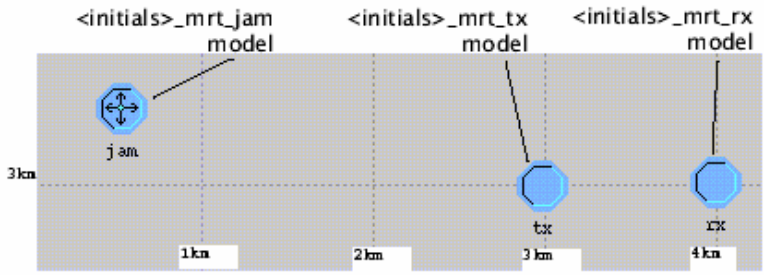


图 9-48　网络拓扑图

- 单击 Advanced 检查框。
- 编辑 x position 和 y position 属性来放置各个节点。

将 tx 节点放置在网格(3,3)位置，rx 节点放在 tx 节点右边 1 公里外的网格(4,3)位置。将 jam 节点放在网格中的(0.5,2.5)处。

注意： 节点的相对位置对于移动无线通信的行为有重要影响。为了得到期望的仿真结果，按上面所说的来定位节点显得非常重要。

（7）关闭对象模板。

关键概念

要使用移动平台来指定节点的运动，网络模型使用了一个称为 trajectory（轨迹）的属性。这个属性的值是一个 ACSII 文本文件的文件名，该文件是在项目编辑器中。该文件包含了指定仿真过程中移动节点将要经过的时间和位置的数据。

定义完网络模型之后，必须指定一条移动干扰发射机节点行进的轨迹。

（8）从 Topology 菜单中，选择 Define Trajectory。

（9）在 Define Trajectory 对话框中，按图 9-49 设置属性值，然后单击 Define Path 按钮。

当你在 Define Trajectory 对话框中按下 Define Path 按钮后，对话框会消失，光标在项目编辑器中变成一条线。现在可以画出移动节点的轨迹：

（10）在 jam 节点的左边缘单击鼠标左键，开始描绘轨迹。

（11）在网格的(7.5,2.5)位置单击左键（7.5 为水平位置，2.5 为垂直位置）。

（13）单击右键结束轨迹。轨迹会从屏幕消失，因为此时它还没有被移动节点引用。

最后，将刚刚创建的轨迹应用给 jam 节点。在 jam 节点上单击右键，选择 Edit Attributes。

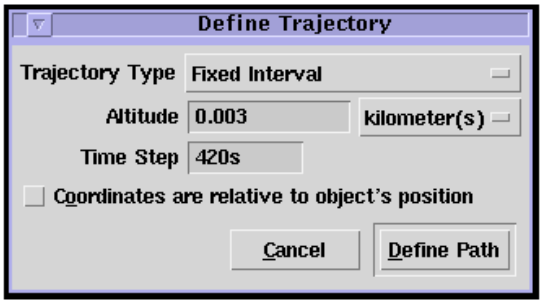


图 9-49　定义轨迹

（14）将 trajectory（轨迹）属性改为 mrt。

（15）单击 OK 关闭对话框。轨迹在项目编辑器显示为绿色的箭头。

（16）在绿色的线上单击右键，选择 Edit Trajectory。Edit Trajectory Information 对话框出现。

（17）编辑每行中的 X 和 Y 的值，如表 9-2 所示。

表 9-2　x、y 的值

#	X Position	Y Position
1	0.5	2.5
2	7.5	2.5

确保 Coordinate are relative to object's position 检查框没有选中。

（18）单击 OK 关闭对话框，然后保存项目。

9.5.7　收集统计量并运行仿真

关键概念　对于这个例子，我们感兴趣的是不同的天线模型对于网络中接收节点的影响。可以通过配置 Simulation Tool（仿真工具）自动改变天线模型属性进行参数化仿真研究，而不需要每次仿真时都在节点域中改变天线模型属性值。

可以在项目编辑器中收集仿真后的接收信道统计量。这些统计量包括误码率（bit error rate）（BER）和吞吐量（throughput）（packets/sec）。包吞吐量统计值代表了接收信道每秒正确接收到的包的平均值。这个属性值采集的样值仅是那些包 BER 值小于收信机 ECC 门限的包，该门限在节点模型中无线收信机模块的 ecc threshold 属性中指定。由于本例中无线收信机该属性的值为 0.0errors/bit，只有没有比特误码的包才会被接收。

关键概念　对于不同的统计量，可以改变其 collection mode（采集模式）属性值。这些模

式指定了统计量的捕获方式（all values，bucket，sample，glitch removal），以及它们的采集模式。

要采集误码率和吞吐量统计值：

（1）在 rx 节点对象上右击，从弹出菜单中选择 Choose Individual Statistics。

（2）选择下列统计值：

Module Statistics: radio_rx.channel [0]: radio receiver: bit error rate

（3）在 bit error rate 统计量上单击鼠标右键，从弹出中菜单选择 Change Collection Mode。

（4）在 Capture Mode 对话框中选中 Advanced 选项。

（5）将 Capture Mode 改为 glitch removal。完成后单击 OK 按钮。

要设置吞吐量统计值的采集模式：

（6）在 throughput (packets/sec)统计量上单击鼠标右键，从弹出菜单中选择 Change Collection Mode。

（7）在 Capture Mode 对话框中选中 Advanced 选项。

（8）确信 Capture Mode 设置的是 bucket，然后将 Bucket Mode 改为 sum/time。

（9）单击 Every…second 按钮，编辑它的值，将 Sample Frequency（采样频率）设为 10 seconds。确信 Reset 框没有被选中。

（10）单击 OK 关闭 Capture Mode 对话框，然后在单击 OK 关闭 Choose Results 对话框。

指定完要收集的统计量后，就可以对仿真进行配置了。可以使用Simulation Tool进行带参数的研究：某个属性的值是变化的，该值决定了对网络行为的影响。

（11）从 Simulation 菜单中选择 Configure Simulation(Advanced)，如图 9-50 所示。

Simulation Tool 打开。

图 9-50 仿真图标

（12）在仿真设置上单击右键，从弹出菜单中选择 Edit Attributes。

配置仿真如下：

（13）单击 Add 按钮，然后选择所有的未引用属性。选择完后单击 OK 按钮，如图 9-51 所示。

这些是在节点模型编辑器中提升过的属性。因为在节点模型编辑器中没有分配值，现在必须给它们赋值。需要注意的是，虽然这时它们出现在属性列表里，但并没有值。

给ant_rx.pattern属性赋值：

（14）单击左键选中 ant_rx.pattern 属性。

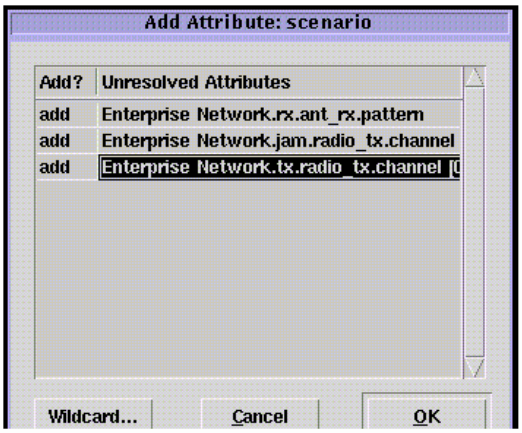


图 9-51　属性窗口

（15）单击 Values…按钮。

（16）在属性对话框的 Value 部分，将第一个值设为 isotropic，第二个设为 <initials>_mrt_cone。单击 OK 按钮。

然后设置 jam.radio_tx.channel[0].power 和 tx.radio_tx.channel[0].power 属性的值：

（17）对于 jam.radio_tx.channel[0].power 属性，单击 Value 字段，输入 20。完成后按敲回车键。

（18）对于 tx.radio_tx.channel[0].power 属性，单击 Value 字段，输入 1。完成后按回车键，如图 9-52 所示。

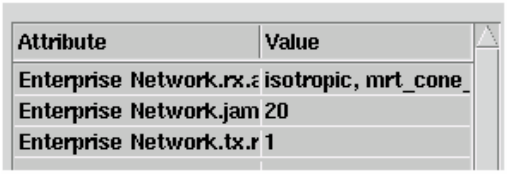


图 9-52　属性

注意这时 Number of runs in set 是 2。这是因为 ant_rx.pattern 属性现在有两个可能的值，所以会进行两次独立的仿真，每次使用该属性的一个不同值。当运行多个仿真时，必须指定是否为每次仿真保存结果。

（19）选中 Save vector file for each run in set 单选按钮，如图 9-53 所示。

图 9-53　设置属性

（20）同时还需要改变仿真的 Seed，Duration 和 Update Interval 设置。

（21）将 Seed 改为 50。

（22）将 Duration 改为 420 seconds。

（23）将 Update Interval 改为 10。

（24）完成上述设置任务后，单击 OK 按钮。

注意仿真设置图标已经变化，表明仿真设置中有多个要运行的仿真，如图 9-54 所示。

图 9-54 多个仿真图

（25）保存仿真设置。

（26）单击 Excute simulation sequence 动作按钮运行仿真。

（27）两个仿真均完成后，关闭仿真工具。

9.5.8 查看并分析结果

仿真运行结束后，就可以检查误码率和包吞吐量结果了。

要查看结果：

（1）从 File…菜单中选择 New…，然后选择 Analysis Configuration，单击 OK 按钮。

（2）单击 Create a graph of a statistic 动作按钮。

注意有两个结果集：<initials>_mrt_net-antenna_test_1 和<initials>_mrt_net-antenna_test_2。这两个结果和两次仿真对应，一次是全向天线模型；一次是锥形天线模型。

（3）双击<initials>_mrt_net-antenna_test-1 左边的箭头，展开可获得的统计量的层次结构。

（4）单击 bit error rate 左边的框，然后单击 Show 按钮。把图形移到一边。

（5）取消对<initials>_mrt_net-antenna_test_1 的 bit error rate 的选中。

（6）双击<initials>_mrt_net-antenna_test-2 左边的箭头，单击 bit error rate 左边的框，然后单击 Show 按钮。

全向天线的误码率图显示如图 9-55 所示。

和料想的一样，全向天线模型下收信机的误码率随着干扰发射机节点和收信机节点距离的减少而逐渐增加。当干扰发射机和收信机距离最小时，误码率最大，约为 0.32。全向接收天线在整个仿真过程中都接收到干扰发射机的干扰信号。

注意：定向天线的仿真结果在很大程度上取决于天线增益。如果你的结果和这里的不一样，很可能是因为定义的增益有小的差异。定向天线模型下的误码率曲线如图 9-56 所示。

定向天线的图也表明开始时随着干扰发射机节点和收信机节点距离减少接收方误码率增加。但大约 1 分钟之后，连接干扰发射机天线和收信机天线的方向矢量就不在收信机天线的最大增益方向内了。于是收信机节点不再接收来自干扰发射机的干扰，误码率降为 0。这就使得成功接收静止发射机节点的包的概率急剧增加。

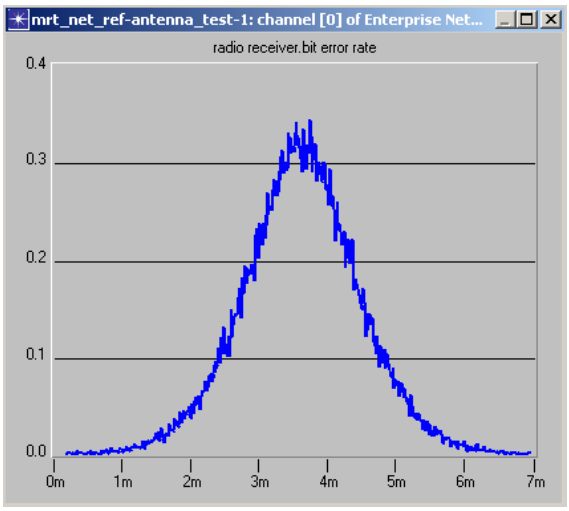


图 9-55　全向天线误码率图

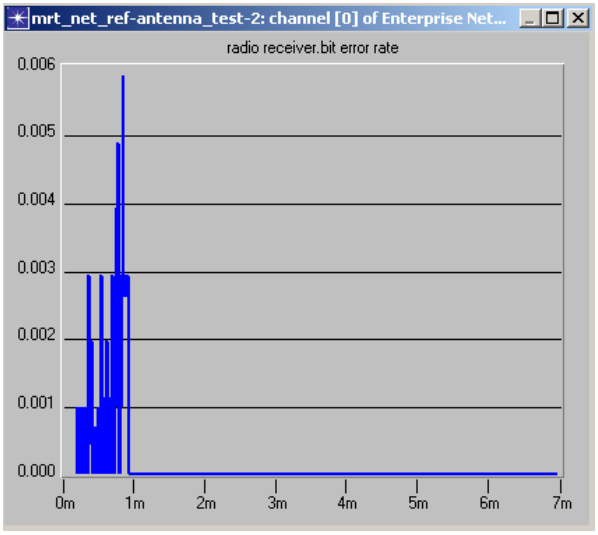


图 9-56　误码率曲线

要查看包吞吐量的结果：

（7）关闭误码率曲线图，在 View Results 对话框中取消对误码率统计量的选中状态。

（8）选中 <initials>_mrt_net-antenna_test_1 和 <initials>_mrt_net-antenna_test_2 的 throughput（packet/sec）统计量。如图 9-57 所示

（9）选择 Statistics Overlaid，然后单击 Show 按钮，使两张图显示在一个面板中。

吞吐量（packets/second）曲线图如图 9-58 所示，你得出的图可能不完全相同，因为描绘的轨迹图可能会稍有出入。

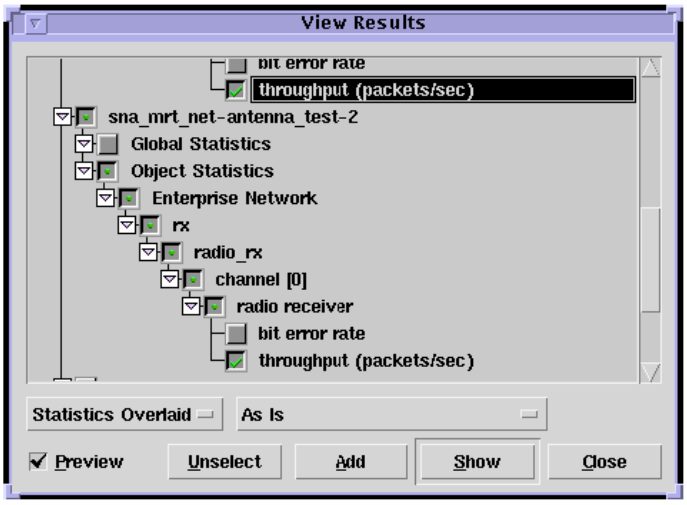


图 9-57 统计结果图

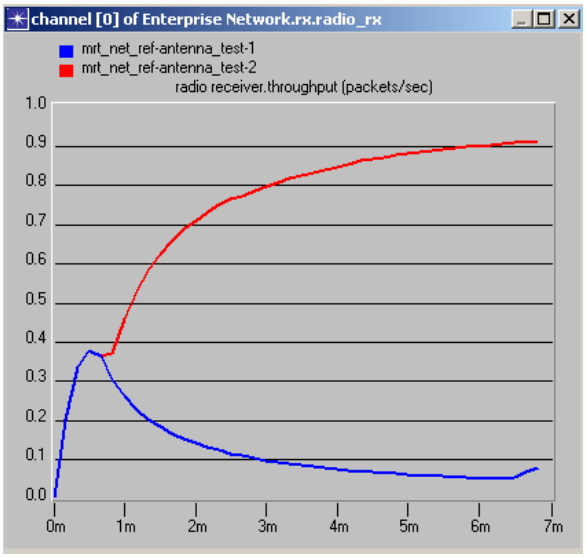


图 9-58 吞吐量结果图

注意对全向天线模型来说，仿真过程中接收到的包的平均值是下降的（在仿真快要结束时，由于干扰发射机和收信机的距离增加，这种趋势某种程度上有所逆转。）对于定向天线，当连接干扰发射机和收信机天线的方向矢量与收信机天线的最大增益的方向一致时，包吞吐量很低。但是，大约 1 分钟之后（干扰发射机不在收信机天线的最大增益方向上），接收到的包开始增加。

第 10 章　OPNET 标准模块介绍

10.1　IEEE 802.11 模块内部结构及仿真

10.1.1　IEEE 802.11 无线局域网概述

无线局域网协议是以 IEEE 802.11 标准为基础[10]。该标准定义了一个信道接入控制（MAC）子层和三个物理（PHY）层。IEEE 802.11 协议的目标是构建一个能够提供与有线网络类似服务的无线网络。

IEEE 802.11 无线局域网的构架是用来支持一种移动站主要以分布式的方式进行协议会话的网络。组成 IEEE 802.11 网络可能有以下几种等级成分：

1．移动站（Station）：移动站是直接与无线信道连接的组件。它可以是移动的、便携式的或是固定的。每个移动站支持包括授权、认证、密码保护和交换数据（MAC 服务数据单元）等服务。

2．基本服务子集（Basic Service Set，BSS）：一个 IEEE 802.11 无线局域网至少包含一个 BSS。BSS 是由一系列可以互相通信的移动站组成。如果基本服务子集中的所有移动站可以直接互相通信而不与有线网络相连，称该 BSS 为独立基本服务子集（Independent BSS）。IBSS 代表一种典型的自组织网络，它构成简单，规模小，而且源和目的节点之间的路由只有一跳。

如果 BSS 包含一个接入点（Access Point：AP），则称该 BSS 为“架构 BSS（Infrastructure BSS）”，意味着它可以作为更大网络的一个组成部分。在一个架构 BSS 中，所有移动站和 AP 进行通信。AP 既可作为无线子网通向有线网络的入口设备，又可作为本地无线子网络由交换设备。

3．扩展服务子集（Extended Service Set，ESS）：一个 ESS 由多个“架构 BSS”组成，而每个“架构 BSS”都含有一个 AP，这些 AP 成为数据从一个 BSS 通向另一个 BSS 的桥梁。同时 AP 也可将数据转交给作为无线局域网的主干分布式网络系统（Distribution System，DS），它的布网方式一般是有线的。

10.1.2　无线局域网的协议行为建模

由于 IEEE 802.11 协议本身的复杂性，使得对其建模非常困难。我们根据协议标准将其拆分为多个相对独立的部分，本文为“协议行为”，首先列举无线局域网的如下各种行为：

（1）MAC 协议会话：

MAC 协议会话至少涉及两种帧的交互参与，分别是从源到目的节点的数据帧和从目的到源节点的 ACK 帧。如果源节点没有接收到确认帧，则它会等待合适的退避时间并且次数有限地重传数据帧。数据帧和确认帧的交互可以提供数据传输一定的可靠性保证。

如果要进一步增大数据传输的可靠性，MAC 协议会话额外要求请求发送帧 RTS 和确认发送帧 CTS 的参与，它们用来预留信道带宽。首次源节点向目的节点发送 RTS 请求预留信道，如果成功，目的节点会响应一个 CTS 帧，这个过程可以看作是传输数据之前的握手。RTS/CTS 帧交互是协议非强制的方案，图 10-1 描述了启动 Rts/Cts 协议会话成功地传输数据帧的流程图。

（2）接入机制：

标准中定义基本的接入机制采用基于二进制指数退避的载波监听冲突避免协议（CSMA/CA）的 DCF 接入方案。在传输开始之前，移动站先监听信道。如果监听到信道被占用，则移动站不会传输包。如果有两个或两个以上的移动站同时传输包，则产生冲突，而导致一个或多个包受损。这是基本的 CSMA 机制。为了进一步避免冲突，移动站在开始传输包之前如果监测到信道被占用，则自动进入基于二进制指数退避算法的退避阶段。退避算法随机选择一定的时间量，规定移动站必须等待这段时间才能尝试发送包。比起传统的冲突监测机制，IEEE 802.11 MAC 更倾向于使用冲突避免，从而使得一对传输和接收可以尽可能连续以减少端对端延时。为了达到这个目的，IEEE 802.11 MAC 引入一个称为网络分配矢量（NAV，Network Allocation Vector）的变量。如图 10-1 所示，NAV 指示某个移动站认为信道再次开放之前必须等待的时间。所有的帧都带有它们所要求占用信道的持续时间值，移动站通过查看这个值可以使其 NAV 始终保持最新并且准确。通过联合虚拟载波监听机制（使用 NAV）和物理载波监听机制，MAC 实现了 CSMA/CA 接入机制中的冲突避免的功能，这个基本的接入机制也是分布式协调功能（DCF）的主要内容。如果物理和虚拟的载波监听机制同时监测出信道空闲 DIFS（Distributed Interframe Space，分布式帧间间隔）长的时间间隔，那么移动站就可以开始传输数据帧。然而，如果信道忙则必须启动退避算法。如果没有从目的站点收到确认帧，则此次传输视为失败，而可能导致该帧的重传。

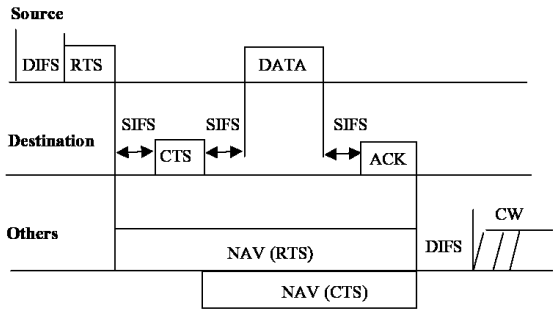


图 10-1　通过启动 Rts/Cts 协议会话成功传输数据帧的流程图

IEEE 802.11 标准定义有两种信道接入控制方式，分别是分布式协调方式（Distributed

Coordinated Function，DCF）和中心协调方式（Point Coordination Function，PCF），它们的区别在于“distribution（分布式）”和“Point，（中心）”。对于 DCF 的“distribution（分布式）”，众所周知分布式网络也就是由许多同等实体组成的一个网络系统，这里是很多同等的移动站组成的一个系统，它们依托分布式协调方式公平地竞争资源，采用有竞争的信道共享方式（CSMA/CA），其具体的机制参考文献[3]。而 PCF 为依托中心节点协调无线资源分配的一种方式，必须要有一个节点担任“中央控制节点”的角色，也就是接入点 AP，即基站（如果把用户称为移动站的话）。PCF 采用轮询的方式来分配信道的使用，因此它是没有冲突的；这种方式可以使用在一些有延时要求的场合中，比方说语音，多媒体等。

大多数情况 WLAN 的默认配置采用 DCF 而不是 PCF 方式。因为 DCF 基本上能够满足传送数据业务的服务要求，而 PCF 由于采用轮循（Round Robin）的方式，增加了开销，因此带宽利用率较 DCF 低。具体来说，在数据业务负载比较低时，PCF 如果需要在发送站点和接收站点之间进行对话，AP 先通过发送“POLL 包”询问是否有移动站要求发送数据，移动站应答“NULL 包”或“DATA Packet”后，AP 再发送“ACK 包”确认。显然，POLL 包的引入使开销增大。其次，从网络的鲁棒性来说，集中控制方式下，一旦 AP 站点损坏，则整个网络将无法工作；而在 DCF 情况下，即使 AP 损坏，网络还可以以 ad hoc 的方式运作。

（3）帧间强制等待

在两帧之间的传输必须经历一个强制等待时间：DCF 机制包含 DIFS，SIFS，和 EIFS。这些帧间间隔的具体取值是根据物理信道特征（跳频、红外、直接序列）来定。

（4）监测信道忙后的退避

按照二进制指数分布规律增长退避函数的上限，并且退避函数的上限和下限之间随机选取一个时间值，作为再次访问信道所必须经历的时间量。

（5）支持多种数据传输速率

WLAN 协议支持的数据率有：1Mbps，2Mbps，5.5Mbps，和 11Mbps。这些数据率模拟发信机和收信机处理包的速率。而为了支持不同的数据率，物理层需配置相应的信道参数，并且通过不同的信道流和 MAC 进程相连。

（6）数据恢复机制

确认帧接收失败将触发重传机制。并且根据数据帧的大小对重传次数有不同的限制长包重试最大次数（long retry limit）和短包重试最大次数（short retry limit）。

（7）拆分和集成

根据高层数据包的大小确定是否启动数据拆分功能。超过门限则拆成多个帧（PSDU，协议数据单元）。目的站会将这些帧集成为原始数据包送往高层。

（8）包接收重复监测

包接收后，相关信息被存储。任何重复接收的包将被 MAC 层丢弃。

（9）接入点功能的支持

在一个架构 BSS 网络中任何移动站配置该功能则可以充当一个接入点。为了能够连接 BSS 和分布式核心网，AP 还必须实现 IEEE 802.11 协议和分布式核心网接入协议的转换，

它实际上充当 WLAN 路由器的角色。进一步要搭建一个 ESS（扩展分子集），为了能够跨网通信，其属下 BSS 中的所有移动站都必须具备支持 IP 协议站的功能。一个 BSS 被看成是一个 OPNET 子网（subnet）。

（10）数据缓冲存储

数据从高层流向 WLAN MAC 层将被存储在一个缓存中，它是一个 FIFO（先进先出）队列，在移动站竞争到信道之前高层数据包在该队列中排队等待。

（11）物理层建模：

详细的无线物理层行为建模参见文献[10]。但是物理层建模对 IEEE 802.11 无线局域网的性能几乎没有多大影响。所以不需要仿真实际 IEEE 802.11 指定的物理协议，只是通过设置协议所对应的相关参数（如时隙值和退避时隙个数）来粗略逼近实际效果。

以上所属分析了无线局域网的各种行为，分别对这些行为单独建模后，通过有限状态机将它们集成为一个系统而形成最终的 IEEE 802.11 协议支持模块。IEEE 802.11 无线局域网 MAC 有限状态机结构如图 10-2 所示，它基本实现了 IEEE 802.11 协议。

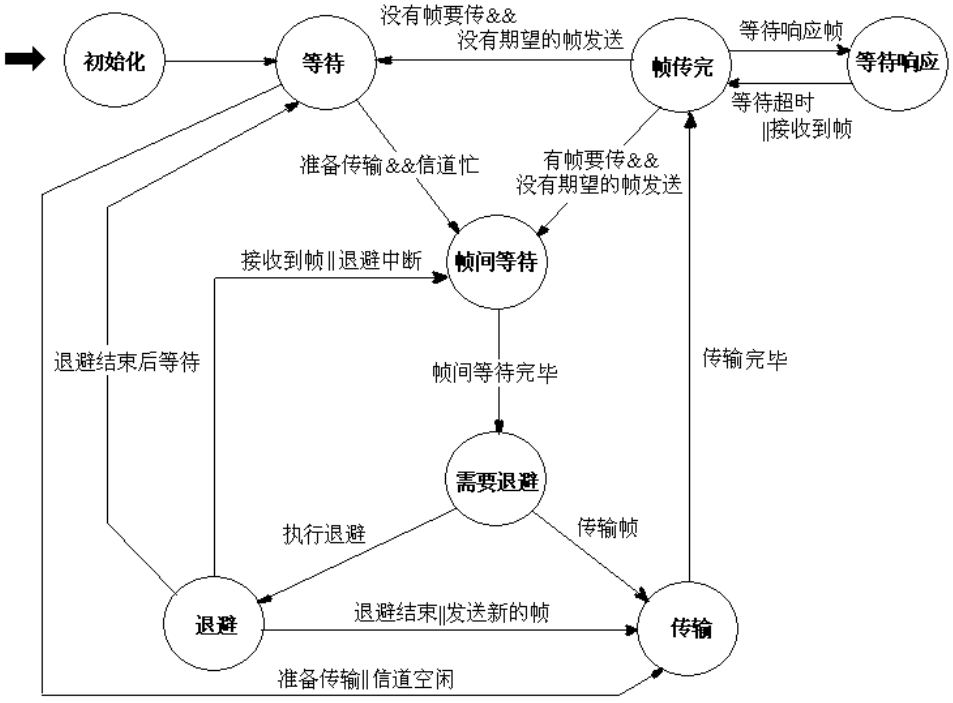


图 10-2　IEEE 802.11 无线局域网 MAC 有限状态机结构图

10.1.3　IEEE 802.11 无线局域网 MAC 的输入接口

模型的输入接口界面如图 10-3 所示。按照图中顺序，输入接口参数描述如下。

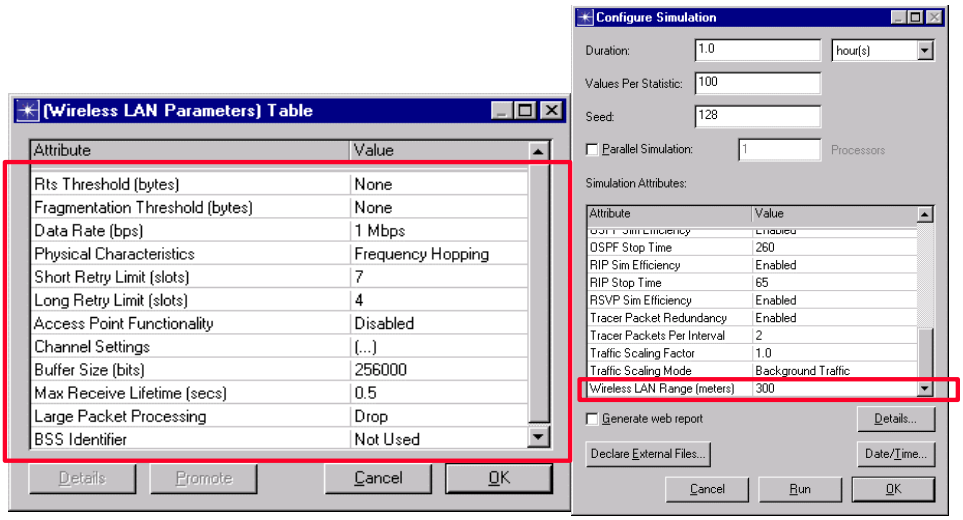


图 10-3　IEEE 802.11 无线局域网 MAC 的输入接口

（1）Rts 门限

该门限决定某个数据帧的传输是否要启动 Rts/Cts 协议会话。如果从高层接收到的包（也称为 MAC 服务数据单元 MSDU）大于 Rts 门限，为了增加传输效率（对于大包额外花销资源预留带宽而增加这次发送成功的概率是值得的），则启动 Rts/Cts 协议会话。由于 Rts/Cts 协议会话是协议非强制的功能，所以该值默认为 None，意味着不管 MSDU 多大也不启用该功能。

当 Rts/Cts 协议会话功能启用则意味着对于每次成功的数据帧的传输，为了提高数据传输的可靠性，都必须为预留信道消耗额外带宽。

（2）拆分门限

该门限决定高层数据包（MSDU）是否需要拆分。拆分后帧的数量是由 MSDU 的大小和拆分门限决定的。目的站点会将当前接收的帧放入隶属于某个 MSDU 的集成缓存，直到收到所有的帧才集成还原为 MSDU 并释放集成缓存。

OPNET 在实现拆分和集成操作时用到了 SAR 核心函数库。该值默认为 NONE，意味着任何时候都不启用该功能。

对大包采用拆分传输提高了数据传输的可靠性。但是由于对于每一个数据拆分帧都需要目的站点恢复一个确认帧从而使协议开销增大。拆分门限和 Rts 门限存在一定的关联性，如果拆分门限小于 Rts 门限则 Rts/Cts 协议会话功能不可能启用。为了提高信道预留效率，这两个门限值的设置尽可能匹配。

（3）数据率

WLAN 模型支持 1Mbps、2Mbps、5.5Mbps 和 11Mbps 四种数据率。移动站可以根据界面所选的数据率参数来发送数据，但是可以以任意速率接收包。另外，根据协议指定，所有的信令包都以 1Mbps 的速率传输。

（4）物理特征的选择

IEEE 802.11 标准指定三种物理层配置方案：跳频（Frequency Hopping），红外（Infra Red）和直接序列（Direct Sequence）。虽然 WLAN 模型没有实现这些物理层的建模，但是提供了 MAC 层所需的物理层的参数。这些参数是根据物理特征来设置：

- 信令帧间间隔（SIFS）。
- DCF 帧间间隔（DIFS）。
- 最小和最大的竞争窗口大小（退避时隙的个数）。模型默认的设置是跳频。

（5）短包重试限制

该参数为数据帧传输可允许的最大的重传次数，如果超过次数则被丢弃。短重试限制只针对 MSDU 小于 Rts 门限的数据帧，即只针对不需要 Rts/Cts 协议会话的数据帧。默认值为 7。

（6）长包重试限制

该参数为数据帧传输可允许的最大的重传次数，如果超过次数则被丢弃。长重试限制只针对 MSDU 大小超过或者等于 Rts 门限的数据帧，即只针对需要 Rts/Cts 协议会话的数据帧。默认值为 4。

（7）信道设置

WLAN 模型由四对传输接收信道对。分别提供 1Mbps、2Mbps、5.5Mbps 和 11Mbps 的数据率。用户可以设置这些信道的最小频率和带宽。

（8）缓存大小

指定高层缓存的最大容量。如果接收当前高层包会导致该缓存溢出则被丢弃，直到有包移出缓存。

（9）最大接收生存时间

该参数为目的站点集成陆续到达的数据拆分帧所能等待的最大时间，如果在生存时限之前成功地集成隶属于某个高层数据包（MSDU）的所有拆分帧，则此次传输成功，并将集成的包送往高层。

（10）无线 LAN 通信范围

该参数指定移动站能够互相通信的最大距离。根据 IEEE 802.11 标准指定的内容，移动站之间的空中传播时间为 1μs，从而计算出最大允许的通信范围为 300m。

10.1.4 IEEE 802.11 无线局域网 MAC 的输出接口

仿真结束后通过观察模型提供的统计量可以对无线局域网的性能进行分析。模型的输出接口界面如图 10-4 所示。按照图中顺序，输出接口参数描述如下：

（1）退避时隙个数。

（2）信道预留（NAV 计数器）。

（3）发送的信令业务（包括 Ack，Rts 和 Cts）。

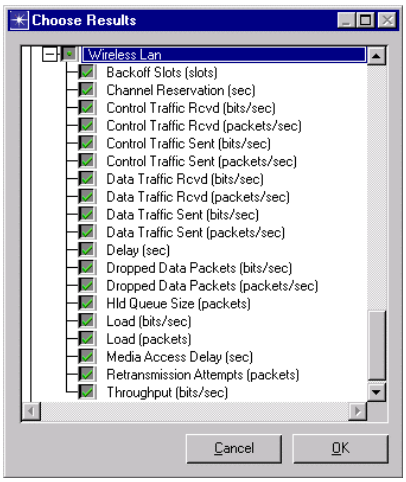


图 10-4　IEEE 802.11 无线局域网 MAC 的输出接口

（4）接收的信令业务（包括 Ack，Rts 和 Cts）。

（5）发送的数据业务。

（6）接收的数据业务。

（7）丢弃的数据包（由于高层缓存的溢出）。

（8）高层数据包队列大小。

（9）负载：从高层接收的总比特数据量（从高层到达的包将被存储在高层队列中）。

（10）信道接入延时：包在高层队列停留的时间，也即是从高层包到达队列的时刻和移出队列被传输时刻之间的时间间隔。

（11）重传尝试次数。

（12）吞吐量：作为包的目的节点，移动站将接收的数据从 MAC 层送往高层的所累积的总比特数。

10.1.5　仿真和实验

图 10-5 为 WLAN 的拓扑结构。图中随机放置了 10 个移动站，这是一种基本无中心型结构（Ad hoc 模式）。所有节点的传输速率均设置为 1Mbps，业务产生采用 ON-OFF 模式。高层包在 MAC 层中不拆分。根据表 10-1 所示的业务的配置，可以通过公式（1）计算业务总负载：

$$\text{每个用户的平均业务负载}=\frac{1}{\text{包平均到达间隔}}\times\text{包平均大小}$$

$$=\frac{1}{0.15}\times(1700\times8)=90666.7(\text{bps}) \qquad \cdots\cdots\cdots\cdots(1)$$

业务总负载＝每个用户的平均业务负载×移动站个数＝906667(bps)

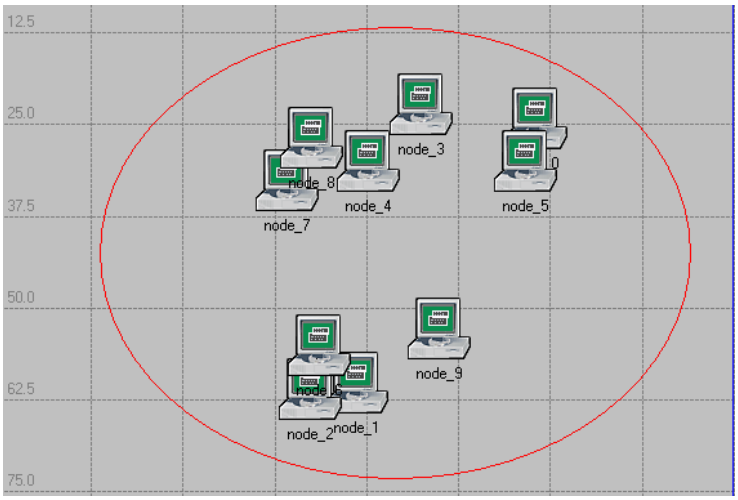


图 10-5　WLAN 的拓扑结构图

表 10-1　业务的配置

移动站个数	10
ON 的平均持续时间/s	100
OFF 的平均持续时间/s	0
包平均到达间隔/s	0.15
包平均大小/bytes	1700
数据率/Mbps	1
业务总负载/Mbps	0.906667

在配置表 1 所示的业务负载情况下，我们运行仿真 3 分钟，得到如图 10-6 所示的业务总负载和吞吐量曲线，在 3 分钟以后曲线基本收敛，意味着网络性能趋于稳定。可以看出，平均的业务总负载为 876.34Kbps，虽然传输速率有 1Mbps，但是平均吞吐量只有 766.67Kbps，并且有 109.76Kbps 的平均数据丢失率，如图 10-7 所示。平均的信道接入延时也较大，高达 1.521s，如图 10-8 所示。从以上结果可以推测出，该业务配置可能没有达到 WLAN 最优的性能。因此我们继续通过调节“包平均大小”，从而改变业务负载，做了一系列的实验，并记录在不同业务总负载情况下平均吞吐量、平均数据丢失率和平均端对端延时等网络性能指标的值，如表 10-2 所示。值得注意的是，网络的端对端延时基本上是由竞争信道引起的，所以把信道接入延时直接看作是端对端延时。

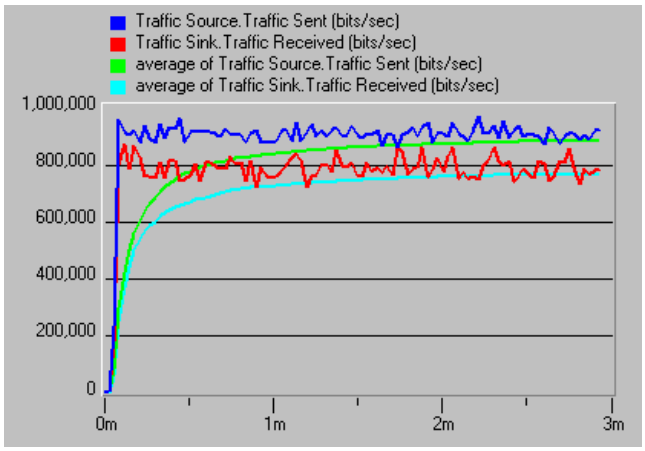


图 10-7　配置表 1 业务时网络负载和吞吐量曲线比较

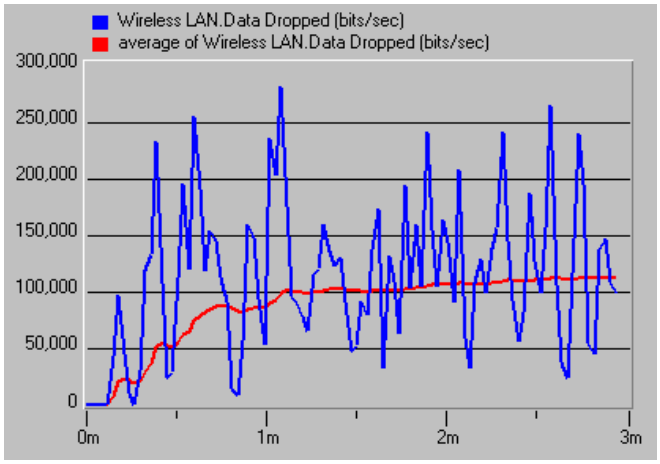


图 10-8　数据丢失率曲线

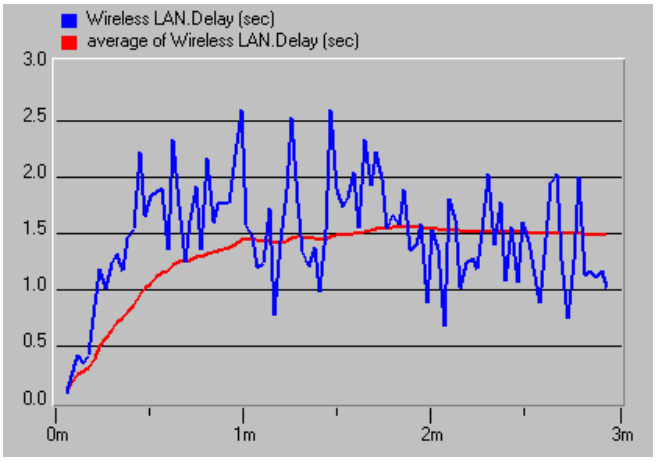


图 10-9　信道接入延时曲线

表 10-2 在不同业务负载下平均吞吐量、平均数据丢失率和平均端对端延时表

业务总负载/Kpbs		574.00	752.71	770.66	807.00	843.00	876.34
网络性能指标	平均吞吐量/Kpbs	574.00	752.71	770.66	771.00	770.00	766.67
	平均数据丢失率/Kpbs	0	0	0	36.00	73.00	109.76
	平均端对端延时/s	0.015	0.027	0.032	0.852	1.163	1.521

通过对表 10-2 进行分析比较，可以得出以下结论：

（1）1Mbps 带宽配置下，网络最大吞吐量为 771Kpbs。

（2）最着业务总负载的加大，网络吞吐量也逐渐加大。当业务总负载达到 770.66kpbs，网络各种性能指标基本达到最优，随着业务总负载的继续增加，网络丢失率和端对端延时性能开始下降。

（3）无线局域网的吞吐量性能当达到最优时开始趋于稳定（有微弱下降），但是丢失率和端对端延时等性能开始急剧下降。当负载过大，而移动站不能及时竞争到有限的信道资源，从而造成高层数据包队列积压过多包，而导致数据丢失；同时由于包的积压使包在队列中等待服务的时间增大，而造成包的端对端延时增大。

（4）1Mbps 带宽配置下，当负载满载时大约有 770Kbps 左右的吞吐量，由此我们可以算出满载时“协议开销”为 230Kbps。

通过引入协议开销的概念，我们可以进一步将结论简化为公式（2）：

$$\begin{cases} \text{业务总负载} < \text{数据传输率} - \text{协议开销}，\text{则吞吐量性能未达到最优} \\ \text{业务总负载} = \text{数据传输率} - \text{协议开销}，\text{则网络各种性能达到最优} \\ \text{业务总负载} > \text{数据传输率} - \text{协议开销}，\text{则数据丢失率和端对端延时性能下降} \end{cases} \quad (2)$$

10.2 X.25 模块介绍

10.2.1 引言

X.25 协议是 CCITT（ITU）建议的一种协议，它定义终端和计算机到包交换网络的连接。在通信前一方首先通过请求通信进程呼叫另一方，被呼叫方接受或拒绝该呼叫。如果呼叫建立，两个系统可以开始进行全双工数据传输，任何一方都可以在任何时候中断连接。OPNET 自带的 X.25 协议模块有：网络层模块（x25_dte_root、x25_dte_chan、x25_dce_root 和 x25_dce_chan）和物理层模块（Lapb）。基于 X.25 协议的 OPNET 仿真流程包括两个部分，（1）基于 X.25 传输控制协议的应用会话建立流程；（2）基于 X.25 数据链路的建立和包交换流程。其中第二部分是对第一部分的底层支持，它们互相独立但是缺一不可，为实现基于 X.25 协议传输系统的互联互通都起着重要的作用。下面将对它们分别进行介绍。

10.2.2　基于 X.25 传输控制协议的应用会话建立流程

（1）传输适应层（tpal,连接应用层和 X.25 网络层的接口）登记 tpal 地址和本地设备（DTE, Data Terminal）地址，连接 X.25 传输协议和应用层协议。

（2）本地设备（支持该应用的客户或者服务器）应用层注册支持的应用（该应用是在业务规格配置模块 profile configure 中配置的）。

（3）传输适应层为本地设备注册应用服务于 X.25 端口。

（4）本地设备在 X.25 端口上开启新的服务（刚刚在传输适应层注册过的，并且可以指定服务等级，如 besteffort:0,background:1,standard:2,Excellent Effort:3, Streaming Multimedia:4,Interactive Multimedia:5,Interactive Voice:6 等）

（5）传输适应层确定会话地址（格式为：tpal 地址.应用端口号）

（6）应用服务器（在前面步骤已经注册过，可支持某个应用）创建业务规格，包括以下几个参数：

- 业务的其实时间。
- 业务终止时间。
- 首次应用（许多不同应用和应用的不同阶段组成一种业务，业务规格配置模块对该业务进行规格描述）的起始时间。
- 应用的调用方式（同时调用或者先进先出调用）。

（7）应用服务器发起某个应用，包括以下几个参数：a、应用的起始时间；b、应用的持续间隔。

（8）应用服务器 tpal 层查找目的地的 X.25 DTE 地址，并通过配置以下几个参数并激活 X.25 会话：

- 本地 X.25 DTE 地址。
- 目的 X.25 DTE 地址。
- 本地应用标识（格式为 tpal 地址＋应用端口号）。
- 目的应用标识。

（9）客户设备 X.25 传输控制层接到来自 X.25 端口的会话，参数类似步骤 8，但是取值互补，相对服务器的目的地址，客户设备则为本地地址，反之亦然。

（10）经过步骤（8）和步骤（9），X.25 会话建立完成，应用服务器 tpal 层接受来自应用层的数据包，并且根据以下参数（Local Address、Remote Address、Local Application、Remote Application）将包发送至目的地 Remote Address 的某个应用端口 Remote Application.

（11）客户设备从 X.25 传输控制层接收来自某应用的 X.25 端口 Remote Applicatio 的数据包并且转交给应用层。

（12）应用服务器提供某应用服务完毕，请求关闭会话（REQ_CLOSE：Local Address、Remote Address、Local Application、Remote Application）。

（13）客户设备响应关闭会话请求。

注意：X.25 DTE 和 Tpal 地址是唯一的，但是服务器提供某应用可以同时开始多个端口或轮循（Round Robin）使用不同端口提供服务。

10.2.3 基于 X.25 数据链路的建立和包交换流程

（1）1 DTE 设备的 x25_dte 层创建并且初始化永久虚电路信道，该信道作为一种专用信道用来建立物理数据链路连接，格式：

channel xx :Local xx Local Application xx

channel xx :Remote xx Remote Application xx

并请求 lapb 子层在本地和终端节点间建立数据链路。

（2）本地 DTE 设备 lapb 层接收到高层的 SETUP 命令，开始向物理链路发送 SABM 信令帧。

（3）DTE 设备 lapb 层从物理链路上收到 LINK_ALIVE 帧，证实物理链路已经建立，进入 SETUP 状态，发信机准备发送数据。

（4）DCE 设备 lapb 层处在 DISCONND 状态，从物理链路接受到 SABM 帧 [P=1]后，发送 UA 响应帧。

（5）DCE 设备的 x25_dce 层接收到来自 lapb 层的连接确认后，进入状态 r1。

（6）DCE 设备的 lapb 层进入 INFO_XFER 状态，发信机准备发送数据。

（7）DTE 设备的 lapb 层处在 SETUP 状态，接收来自数据链路的 UA 帧 [F=1]。

（8）DTE 设备的 x25_dte 层处在 startup 状态，建立 X.25 高层的数据链路。

（9）DTE 设备 1 的 x25_dte 层处在 r1 状态，并创建和初始化数据信道，与步骤（1）建立链路用的专用信道不同的是该信道是临时信道，用来传输数据，并且可以有多个（信道的个数可以通过节点属性 X.25 Highest Two-way Channel Number 和 X.25 Lowest Two-way Channel Number 相减得到）。随后 DTE 设备 1 的 x25_dte 层处在 p1 状态，准备接力高层包。从高层接到连接请求，创建包格式：format=5 channel=1 type=013 给 DCE 设备发送呼叫请求。

（10）DTE 设备的 lapb 层处在 INFO_XFER 状态，接收来自 x25_dte 层的包并向数据链路发送 I frame #0。

（11）DCE 设备的 lapb 层处在 INFO_XFER 状态，接收来自数据链路的 I frame [P=0] 通过#7 确认信息帧。

（12）DCE 设备的 x25_dce 层处在 r1 状态，解码接收来的包：format=5 channel=1 type=013 （即为步骤（9）DTE 发送的包）。建立并初始化信道：

channel 1: Local 1 Local Application d1.0

channel 1:Remote 2 Remote Application d2.25

将该包转交给信道 1。

信道进程进入状态 p1，等待并接收来自 DTE 地址为 2 的 DTE 设备的会话请求。接到

该请求后，包界面进程跳转到 r1。

（13）DCE 设备的 x25_dce 层处在 r1 状态，建立并初始化信道：

channel 2: Local 2 Local Application d2.25

channel 2: Remote 1 Remote Application d1.0

进入状态 p1，创建本地包：format=5, channel=2, type=013，将该会话包转交给 DTE 2。

（14）DCE 设备的 lapb 层处在 INFO_XFER 状态，将步骤 11 的高层数据包传输至物理链路。（Sending I frame #0 to data link）

（15）DTE 设备 2 的 lapb 层处在 INFO_XFER 状态，接收包 [P=0] ，并通过#7 确认信息帧。

（16）DTE 设备 2 的 x25_dce 层处在 r1 状态，解码接收的包：format=5, channel=2, type=013。并创建和初始化信道 2 ：

channel 2: Local 2 Local Application d2.25

channel 2: Remote 1 Remote Application d1.0

信道进程进入 p1 状态，接收来自 DCE 的呼叫，确认 X.25 连接是由 DTE 设备 1 呼叫 DTE 设备 2（X.25 CONN: Calling Addr 1 Called Addr 2），并通知高层。

注意：以上提到的 p1、r1、INFO_XFER 等状态请参照 X.25 相关进程模型。

10.3　干扰机模型

干扰电台分为三种，分别是连续包干扰、脉冲干扰和扫频干扰，它们的结构如图 10-9 所示。

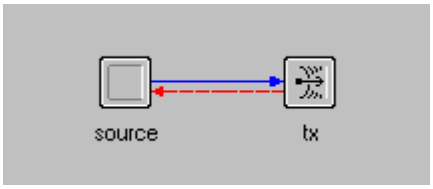


图 10-9　干扰无线电台结构

（1）连续包干扰电台：它基于固定频率带宽，以每秒一个包的速率连续发送干扰数据包。

（2）脉冲干扰：这种干扰方式下频率固定，并且仿真时动态可调，产生的干扰包流量服从 ON－OFF 模型，在 ON 期间连续生成干扰包直到 ON 时间结束，而 OFF 期间不产生干扰包。可以调节 ON 和 OFF 的比例来调整干扰业务的突发程度，如果 ON 和 OFF 的时间比例越小，干扰突发越严重。

干扰包大小=脉冲持续时间×干扰业务速率

（3）扫频干扰：将所用频带分成诺干个等间隔的频段间隙，每传输一个数据包则发射频率增加一个频段间隙，即以扫频干扰方式发送干扰数据包。

当发射频率增加到可使用的最大频率，则发射频率跳到最小频率，即采用轮循（Round Robin）方式改变发射频率。扫频持续时间=扫频周期/频段个数

干扰包大小 = 扫频持续时间 × 干扰业务速率

10.4 OPNET IPv6 模块介绍与仿真

IPv6 是网络 IP 协议的新版本，是在 IPv4 基础上设计的。IPv6 和 IPv4 的主要区别：

（1）扩展了寻址能力：IPv6 将地址位数从 32 为增加到 128 位；

（2）包头格式更加简化：IPv6 略去了 IPv4 包头的某些域，或作为可选域，从而加快了处理包的速度；

（3）流标签：IPv6 允许源节点为每个包分配一个流 ID。这个信息可以用来 QoS 分类；

（4）授权和保密功能：IPv6 的扩展域提供对授权，保密，数据完整性检测等功能的支持。

以下列出 OPNET IPv6 模型的一些特点：

（1）节点支持 dual stack：所有的节点都用该功能，它们可以同时支持 IPv4 和 IPv6，也可以只支持 IPv4 或者 IPv6；

（2）地址配置：允许在不同接口配置 Link-local 和全局地址；

（3）IPv6 隧道机制：支持三种机制，分别是手动配置，自动配置和自动 IPv6 转 IPv4(6to4)；

（4）IPv6 静态路由:可在网关节点中配置到 IPv6 目的节点的静态路由；

（5）RIPng: 在 IPv6 网络中，RIPng 可作为动态路由协议；

（6）ICMPv6 ping: 支持向 IPv6 目的节点发送 Ping 数据包。也可以选择 Ping 数据包是否记录路由；

（7）对应用协议的支持：IPv6 网络支持所有标准的网络应用协议以及自定义的应用协议的运作。

本节介绍 OPNET IPv6 标准模块中的 2 个场景，分别是 ICMPv6 Route Print 和 Manual Tunnel。

10.4.1 ICMPv6 Route Print 场景

（1）实验目的。

验证 ICMPv6 ping 数据包如何跟踪并记录路由。

（2）实验步骤与网络架构

在 IPv6 工程中打开场景：icmpv6_route_print。这时出现如图 10-10 所示的网络拓扑图。

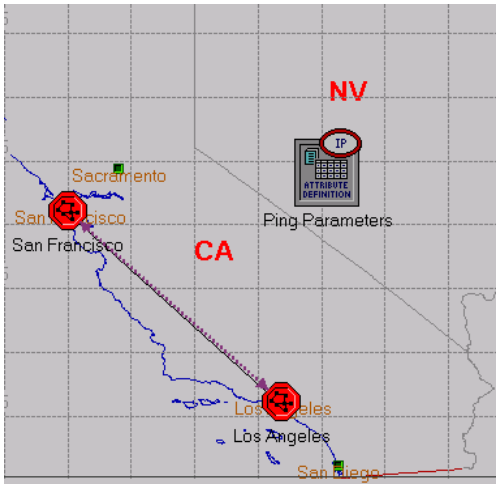


图 10-10　场景 icmpv6_route_print 的网络拓拓扑结构

可以观察到场景中包含两个子网，分别代表一个总公司的两个分公司，它们分别位于两地：San Francisco 与 Los Angle。两地的局域网都包含一个路由器，它们通过链路模型（PPP_DS3）相连，从而使两个局域网能够互连互通，图 10-11 为 San Francisco 与 Los Angle 间链路的属性。

(San Francisco <-> Los Angeles) Attributes

Attribute	Value
name	San Francisco <-> Los Angeles
model	PPP_DS3
transmitter a	San Francisco.gateway.pt_10_0
receiver a	San Francisco.gateway.pr_10_0
transmitter b	Los Angeles.gateway.pt_10_0
receiver b	Los Angeles.gateway.pr_10_0
Background Load	None

图 10-11　San Francisco 与 Los Angle 间链路的属性

位于 San Francisco 的局域网内部结构如图 10-12 所示。

10.4.1.1　San Francisco 局域网构架

位于 Los Angles 的局域网内部结构如图 10-13 所示。

从图 10-13 中可以看出两地内部网络的基本架构是相同的，包含有一个网关（gateway），连接两台交换机（switch），而 switch 再分别连接两个工作站（wkstn, work station）。两地网络不同之处只在于 IPv6 地址的前缀（prefix）不同。此外，内部网络中的每个节点都是 dual stack 的，也就是说同时支持 IPv4 和 IPv6 协议。并且 gateway 也同时支持 RIP 和 RIPng 路由协议。

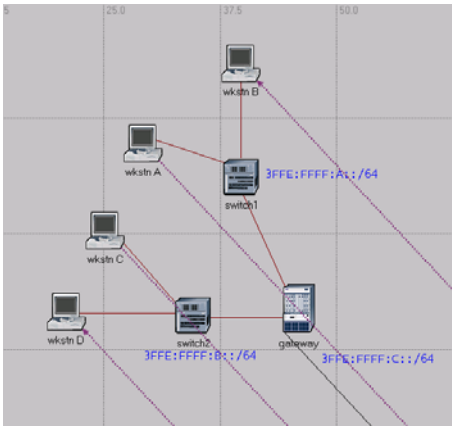


图 10-12　San Francisco 的局域网内部结构

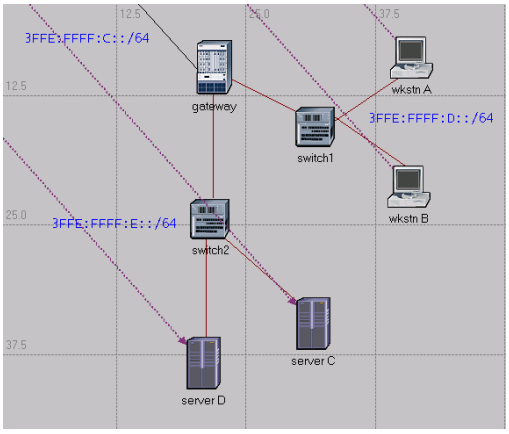


图 10-13　Los Angels 局域网构架

对于 gateway，有关 IPv6 的设置如下：

（1）在 gateway 上单击鼠标右键，从弹出的对话框中选择 Edit Attributes。

（2）在属性对话框中选择 IP→IPv6 Parameters →Interface Information。

（3）在接口信息配置行（row）中进行如下操作：

- 为接口命名。
- 手动指定一个有效的 Link Local 地址，或者保留其默认值“Default EUI-64”。
- 在 Global Address 属性栏中配置至少一个 global 地址。

简单地说，所有的 IPv6 信息设定都需要在 IPv6 Parameter 属性中来配置，包含在 Interface information 中来增加 rows，用以设定包括 Name, Link-Local Address 与 Global Address 等，而这些值同时也要和 IP Routing Parameter 中相对应的值匹配，如图 10-14 所示为 San Francisco 中 gateway 的 IPv6 参数设置。

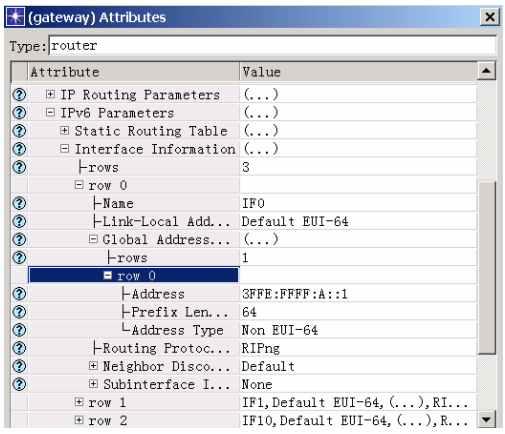


图 10-14　San Francisco 中 gateway 的 IPv6 参数设置

10.4.1.2　San Francisco 中 gateway 的 IPv6 参数设置

Interface Information 属性栏，总共增加了三个 rows 来设定三个 与 IPv6 相关的 Interface 信息，例如：Name (i.e. IF0), Global Address (i.e. 3FFE:FFFF:A::1), Routing Protocol (i.e. RIPng) 等，而这些参数必须和 IP Routing parameter →Interface information 的配置相匹配，如图 10-15 所示。

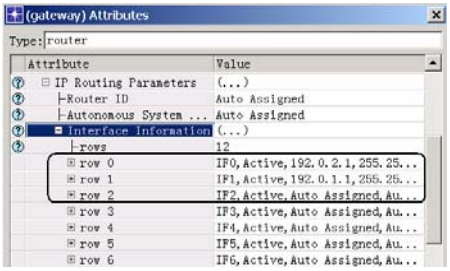


图 10-15　gateway 属性设置

10.4.1.3　IP Routing parameter 与 IPv6 Parameter 接口信息必须相匹配

另外，由于本场景不涉及到隧道封装的操作，因此将 IP Routing parameter →Tunnel Interface 设为 None。

对于其余的 wkstn 和 server 节点，在 IP Host Parameter 属性栏中作类似上面的 IPv6 信息设置，如图 10-16 所示。

10.4.1.4　San Francisco→wkstn C 的 IPv6 Parameter 设置

从上图可以看出，位于 San Francisco 的 wkstn C 的 IPv6 Parameter 中，IPv6 Global Address 设为 3FFE:FFFF:B::2，这符合 gateway 中的 IF1 界面信息中的 IPv6 地址前缀设置（3FFE:FFFF:B::/64），如图 10-17 所示。

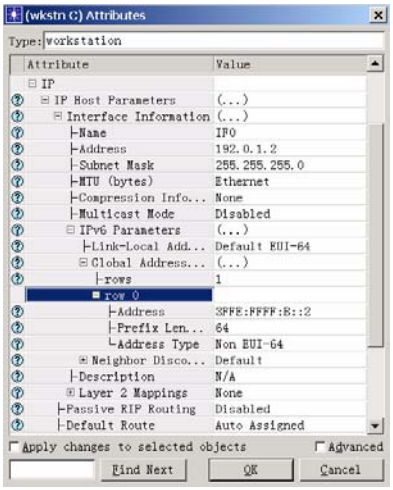


图 10-16　IP Host Parameter 属性

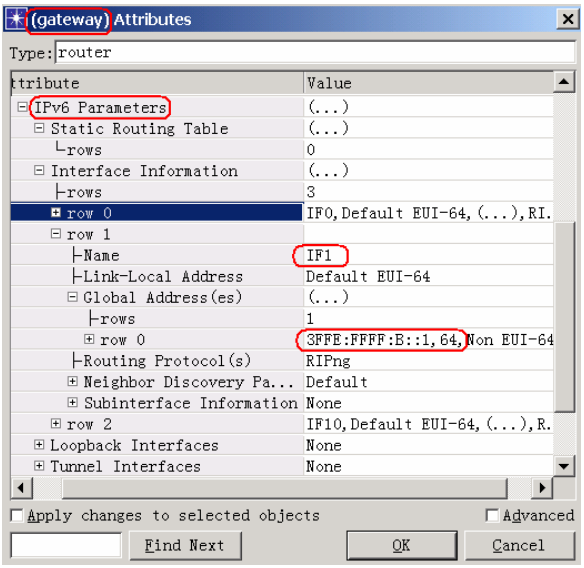


图 10-17　IPv6 地址前缀设置

10.4.1.5　San Francisco→中的 IF1 界面信息设置

将 Tunnel Interface 设为 None，这点和 gateway 设置相同。局域网中其他节点的 IPv6 参数设置也大致相同。

接下来，要配置 Ping Parameters，在网络模型中找到 IP Attribute Configure 模块，它是用来储存与收集仿真所需的相关数据，和前面设置类似，也要通过增加 rows 来配置一些参数，如图 10-18 所示。

10.4.1.6　Ping Parameters 参数的设置

从图 10-18 中可以看到增加了四个 rows（row0 至 row3），针对每个 row 在 Detail 中设置了一些 Ping 信息格式所包含的参数，如 IP Version、Interval、Packet Size、Timeout、Record Route 等，其中最重要的参数为 Pattern（设为 IPv4 或 v6）与 Record Route（设为 Disabled 或 Enabled），所以组合起来共有四个不同的 rows 配置。

在 San Francisco 内部的网络中 wkstn A,B,C,D 分别有一条 ip_ping_traffic model 关联到 Los Angles 的 wkstn A,B 与 Server C,D，在网络模型中以有向的紫色虚线标识，来代表 ping traffic 传送于两地网络之间的源节点和目的节点：

① San Francisco.wkstn A→Los Angeles.server C　　Record Route (IPv6)
② San Francisco.wkstn C→Los Angeles.server D　　Record Route
③ Los Angeles.wkstn A→San Francisco.wkstn B　　Record Route (IPv6)
④ Los Angeles.wkstn B→San Francisco.wkstn D　　Record Route

其中②和④运行 IPv4，而另外两条①和③运行 IPv6 的，其中③的属性配置如图 10-19 所示。

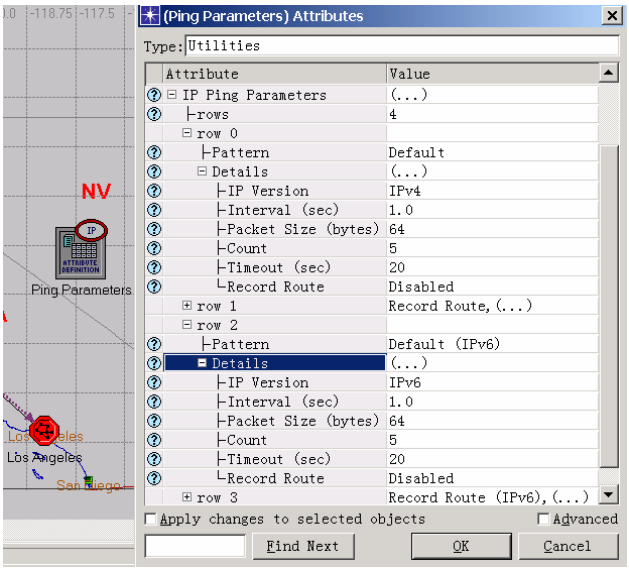


图 10-18　属性设置

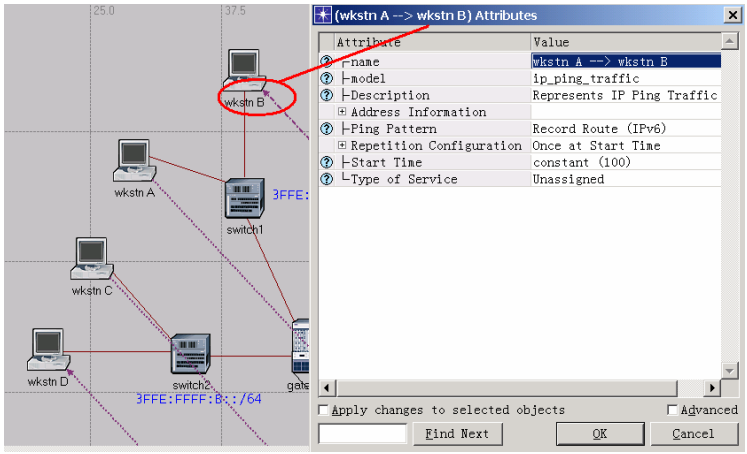


图 10-19　（WkstnA→WKstnB）链路属性设置

10.4.1.7　Ping 关联线的属性

（3）实验模拟结果

依照上述的布置网络拓扑和配置节点属性后，就可以运行仿真来观测实验结果与数据图。

如图 10-20 选择统计量 IP Traffic Sent / Received、IPv6 Traffic Sent / Received、IP Ping Request Send、IP Reply Received。将仿真时间设为 20 分钟并运行仿真。

San Francisco 中 wkstn A 的流量仿真数据结果如图 10-21 所示，。

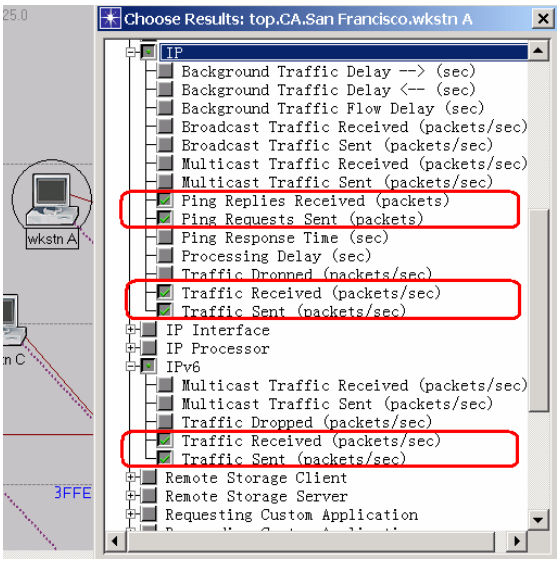


图 10-20　选择统计量

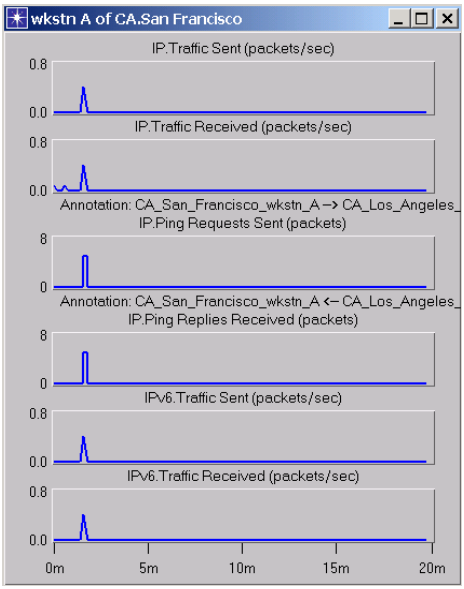


图 10-21　仿真结果

10.4.1.8　San Francisco→wkstn A 的流量仿真数据结果

从图 10-21 中可以看到仿真的数据大约介于仿真时间 1~2 分种内，该结果与 IP Attribute Configure 模块中的 IP Ping Parameters 设置有关，因为属性 Interval 设为 1.0，Packet Size 设为 64，Count 设为 5，如果将这些值增加，则仿真中数据包变多，从而使模拟时间拉长。

另外，对于 Traffic Sent，无论是 IPv4 或 IPv6，其值大约为 0.42 packet/sec 左右，所以

每个节点在 dual stack 支持下并没有多大的效果。当然这也有可能是因为仿真的数据与时间较小所致。

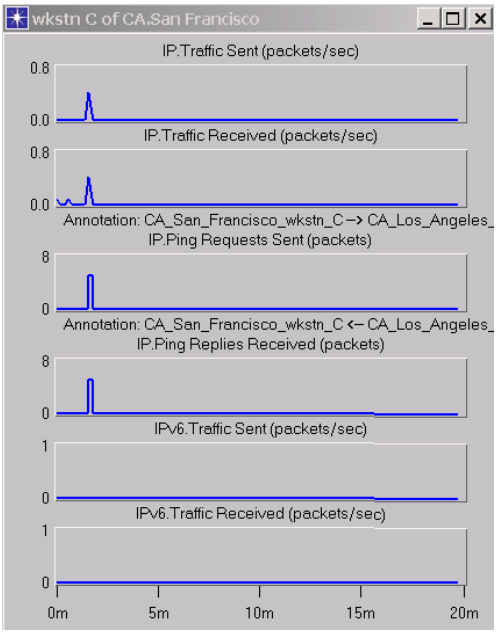


图 10-22　仿真结果

10.4.1.9　San Francisco→wkstn C 的流量仿真数据结果

图 10-22 所示为 San Francisco 中 wkstn C 的仿真数据结果，其中 IPv4 下的结果与 wkstn A 相近（Traffic Sent 都近似为 0.4 packet/sec），但是在 IPv6 下，数据为零，这是因为从在 San Francisco 的 wkstn C 到 Lan Angles server D 间设置了 Record Route 的 ip_ping_traffic，如图 10-23 所示，所以并没有 IPv6 的仿真数据产生。

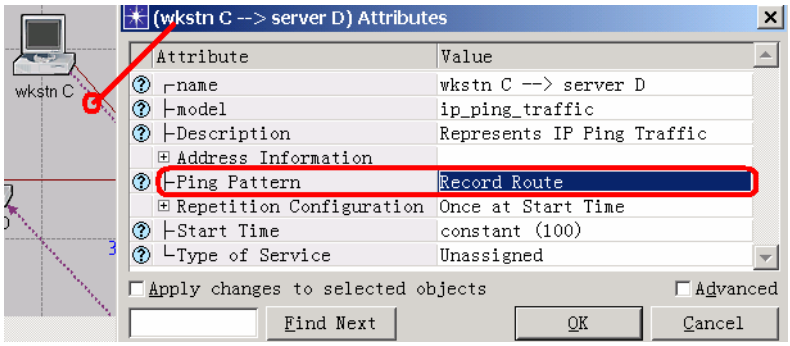


图 10-23　属性设置

10.4.1.10 Ping 关联线属性的 Ping Pattern 设置

若相反地，将 wkstn A 中的 ip_ping_traffic 改为 Record Route，而 wkstn C 改为 Record Route (v6)，则仿真数据就会变成 wkstn A 的结果中没有 IPv6 的数据 ，而 wkstn C 则有 IPv6 数据值，这个结论可以通过再做一次实验从如图 10-24 所示的比较来证实。

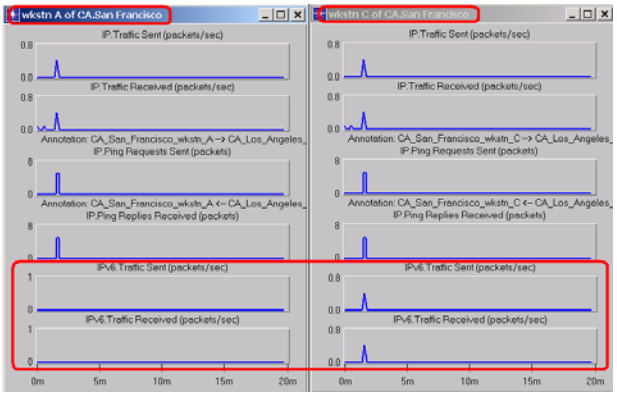


图 10-24 （wkstn C→server）链路吐量比较图

基于仿真结果数据，也可以计算网络的总吞吐量（Total Traffic）。例如要计算 IPv6 的 Traffic，先从网络模型中观察哪些节点传送或接收 IPv6 Traffic:

- Los Angles→gateway 的 IPv6 Traffic Sent 减掉 Los Angles→Server C 的 IPv6 Traffic Received 值后，剩余的即为 Los Angles→gateway 传送给 San Francisco →gateway 的 IPv6 Traffic 值。
- San Francisco→wkstn A 传送 IPv6 Traffic 到 San Francisco→gateway
- 所以 San Francisco →gateway 的 IPv6 Received 为 San Francisco→wkstn A 的 IPv6 Traffic Sent 加上 Los Angles →gateway 的 IPv6 Traffic Sent，再减去 Los Angles →Server C 的 IPv6 Traffic Received 值，约为 1.666 = 0.415+1.666 - 0.413 (packet/sec)，可由图 10-25 证实。

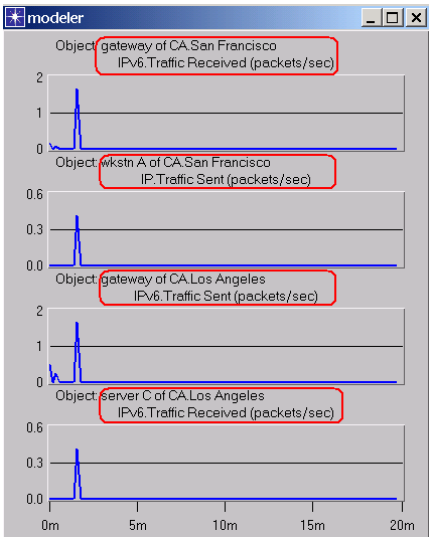


图 10-25　IPv6 业务吞吐量

10.4.2 Manual Tunnel

（1）实验介绍

目前 IPv6 主干网（backbone）还处在发展阶段，因此还是以 ipv4 backbone 为主，本实验主要是模拟 San Francisco 及 Los Angeles 中的两个 IPv6 网络，通过由 San Francisco 与 Los Angeles 间 IPv4 backbone 形成的隧道（tunnel），而互连互通。网络场景如图 10-26 所示。

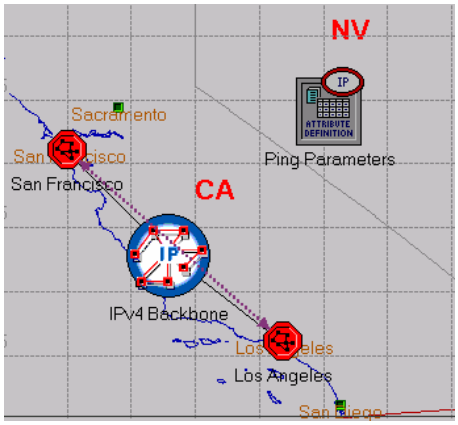


图 10-26 场景 manual_tunnel 的拓扑结构

（2）实验方法与架构

在 San Frahcisco 中，有四个客户端（wkstn A，B，C，D）通过两个交换机（switch） 连接到 gateway，再将局域网连接到 ipv4 backbone。而在 los angeles 分别有两个客户端及两个服务器（server）通过两个 switch 连接到 gateway 再连接到 ipv4 backbone。通过在 IPv4 骨干网中使用隧道而形成 IPv6 扩展网络。

本实验场景类似于前面介绍的 icmpv6_route_print 场景，不同之处在于 San Francisco →gateway 与 Los Angles→gateway 中需要配置隧道参数，如图 10-27 所示。

（3）结论

图 10-28 所示为两地网络与 IPv4 backbone 之间地吞吐量结果。它们对应场景的默认 Ping 参数设置：interval（间隔时间）为 1，Packet Size（封包大小）为 64 byte，count（次数）为 5。由图可以看出，IPv4 Backbone <–>San Francisco 间的吞吐量与 Los Angeles <–>IPv4 Backbone 吞吐量结果相近，都在 2 分钟内完成所有封包的传送。而将 packet size 调大至 1024 byte、count 调大至 50000，仿真结果仍与以上结果类似，这是因为要传送的封包数量并不多，所以都能在很短的时间内就将封包传送完毕。

（4）Ping 参数的设置

而将 interval（间隔时间）调整至 20 时，如图 10-30 所示。传送完所有封包用的时间才明显加大，接近 3 分钟，如图 10-29 所示。

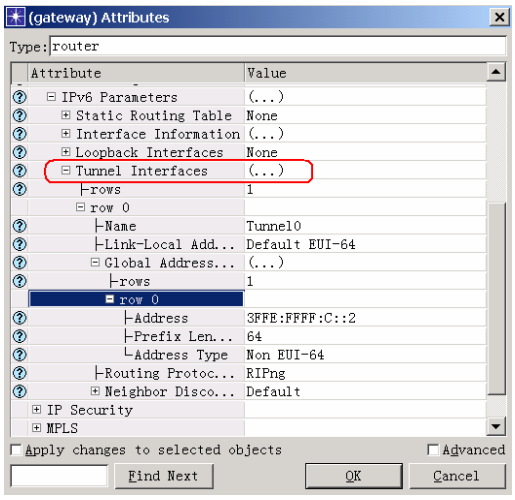


图 10-27　路由器属性

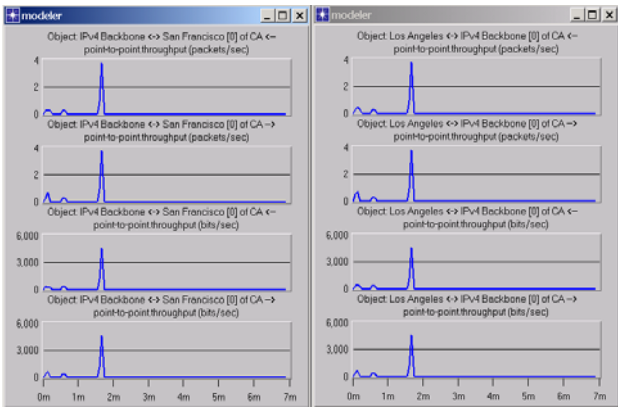


图 10-28　局域网与 IPv4 骨干网间的吞吐量结果

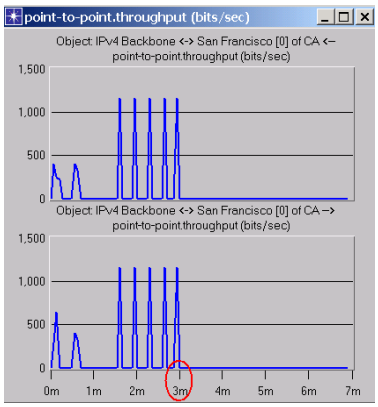


图 10-29　局域网与 IPv4 骨干网间的吞吐量

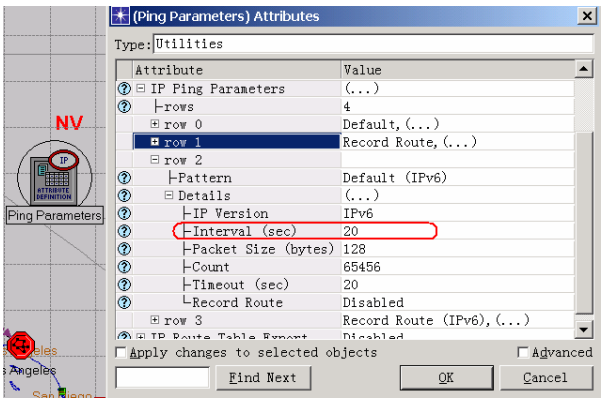


图 10-30　属性设置

10.5 小区系统模型

小区系统模型（Cellular System Example）是OPNET自带的无线网络仿真系统的一个实例，它对加深无线建模的理解有较好的帮助。

10.5.1 模型的导入

（1）新建一个 project，将 Project Name 和 Scenario Name 命名为 cellphone，点击 OK 按钮。

（2）在弹出的初始化拓扑向导对话框中点击 Quit 按钮。

（3）从 Scenarios 菜单中选择 Scenario Components->Import… 如图 10-31 所示。

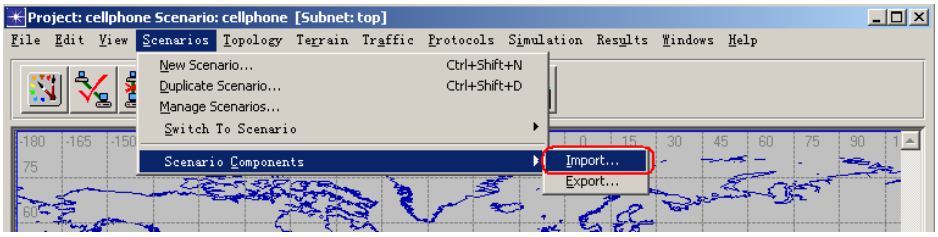


图 10-31 从菜单中导入网络模型

（4）从弹出的网络模型选择对话框中选中 cellphone_net，导出的网络场景如图 11-32 所示：

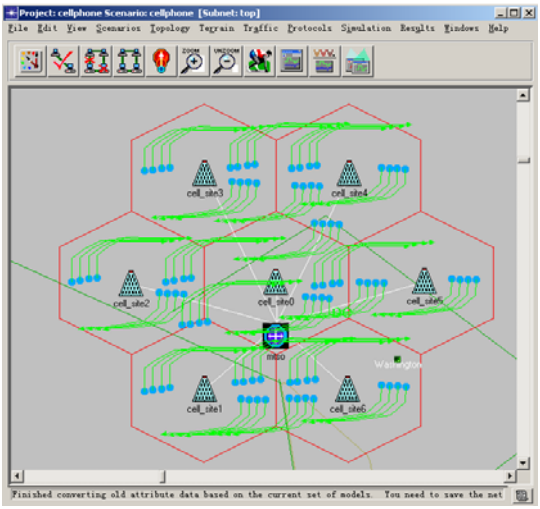


图 11-32 导入小区模型后的场景

这是一个 7 个手机小区及一个主终端交换机（MTSO，main terminal switching office）构成的小区系统模型。7 个小区分别通过 10kbps 带宽接入 MTSO，实现话音传输，小区中的移动手机可以切换至其他小区。

10.5.2　模型的适用范围和限制

（1）小区模型中讲述了大量 AMPS 手机的功能（BELL 系统技术杂志），下面来描述该模型的一些特性：

- 系统模型中的小区数量可以是任意的
- 每个小区中的手机数量也可以是任意的
- 模型中的通话的建立/拆除是明确无误的
- 小区之间的切换过程是自动的基于接收信号质量来决定的
- 手机的自我定位也是基于接收到的小区信号强度来决定的
- 物理层的影响可以自己定义相应的管道模型
- 每一个手机单元的轨迹也是任意的
- 导入 EMA 模型库来确定小区边界
- 统计量输出包括：通话拥塞率，信道利用率，通话建立延迟，信噪比

（2）网络模型建立在小区节点模型和进程模型之上。现在外部接口程序能够允许用户自由的修改小区网络模型，在示例模型中需要配置如下参量：

- 通话产生的分布和平均内部到达时间
- 通话长度的分布和平均通话长度
- 系统中的小区数目
- 小区切换阀值
- 路径损耗计算中的区域类型（城区，郊区，农村）
- 因为树木而带来的损耗值
- 语音的特征
- 为每一个小区分配控制信道的频率
- 移动电话交换局的延迟特征

（3）为了降低仿真运行时间和保持模型的稳定性，必须要做如下假设：

- 就算含有主要状态的信息和通话建立/拆除的信息的包仅仅被发送出去，控制信道的位流也不能描述的太准确，这样的假定有助于提高仿真运行效率。
- 移动用户尽量不使用特殊的语音信道。通常移动台使用一个普通的语音信道来传送语音包，以便于检测信号质量。例如在 AMPS 系统中，实际的语音数据并不是非常的符合要求的，因此并不需要产生几百条信道，而仅仅用一条共享信道就可以完成需求。
- 自定义的管道类型要建立在李建业的“移动通讯系统工程”的分析基础上。

- 通话仅取决于移动用户。

10.5.3 模型包含的文件

表 10-3 列出小区模型所包含的模型文件：

表 10-3 小区模型所包含的文件及其描述

模型所包含的文件	描　　述
cell_multi_7.nt.m	七个小区的网络模型
cell_mob_node_m.nd.m	手机单元节点模型
cell_site_nd_m.nd.m	小区位置节点模型
cell_mtso_m_7.nd.m	七个小区的 MSTO 节点模型
cell_call_gen.pr.m	手机用户通话产生器模型
cell_call_svr_m.nr.m	手机用户协议管理进程模型
cell_resp_fil.pr.m	手机用户响应滤波器进程模型
cell_hoff_mgr.pr.m	小区切换管理进程模型
cell_locator.pr.m	小区定位包管理进程
cell_bis_mgr.pr.m	小区忙/闲状态（BIS）管理进程
cell_ant_point.pr.m	小区天线和定义的动画进程模型
cell_mtso_mgr.pr.m	MTSO 信息操作和切换管理进程
cell_ch_mgr.pr.m	MSTO 独立小区信道管理进程
cell_bkgnoise.ps.c	自定义背景噪声模型管道代码
cell_power.ps.c	自定义接收功率模型管道代码
cell_snr.ps.c	自定义 snr 模型管道代码
cell_call_init.pf.m	通话初始化包格式
cell_call_term.pf.m	通话终结包格式
cell_handoff_pkt.pf.m	通话切换包格式
cell_hoff_msg_pf.m	特殊通话切换消息包格式
celllocator.pf.m	特殊小区定位信息包格式
cell_request.pf.m	手机用户通话建立所需的包格式
cell_tx.pa.m	自定义的三维小区天线模式
cell_urb_def_m.ef	七个小区栓经文件（城区环境）

续表

模型所包含的文件	描　　述
cell_wash.em.c	通过外部接口提供的华盛顿地区的七小区地图
cell_wash.cds	通过外部接口提供的地理背景

10.5.4　模型的属性

对于示例小区模型而言的众多用户特殊属性都是 OPNET 仿真属性，剩下的就是目标属性，目标属性在他们的响应模型中有着详细的说明，表 10-4 中列出了仿真属性：

表 10-4　小区模型的仿真属性

目标	单位	数据类型	值	说明
avg_call_rate	calls/hour	double	5.0	每小时每个用户单元发起的平均通话数量
avg_call_length	minutes	double	3.0	平均通话长度（以分钟计算）
call_rate_dist	N/A	string	exponential	通话数率的 PDF
call_length_dist	N/A	string	exponential	通话长度的 PDF
setting_type	N/A	integer	2	不同环境的路径损耗 0 农村；1 郊区；2 城区
call_update_interval	seconds	double	60.0	在手机语音包到通话建立之间所需的时间
locator_update_interval	seconds	double	60.0	从小区到手机包的定位所需的时间
foliage_density	N/A	integer	0	区域内植被密度所影响的路径损耗 0 没有；1 中等程度；2 严重程度

10.5.5　模型的接口

这节主要描述基本的模型接口。首先用独立的节点模型接口来描述了网络模型接口。

（1）网络模型接口

在网络模型中有三个重要的组成部分：

- 至少有一个用户
- 至少有一个小区
- 仅仅只有一个 MTSO

每一个手机节点都要利用 cell_mob_node_m 节点模型并且和它有轨迹的联系，每一个小区节点都要利用 cell_site_nd_m 节点模型。每一个小区都和 MTSO 建立有双向的点到点的通讯方式。MTSO 利用 cell_mtso_m7 节点模型。同样 MSTO 都要和每一个小区建立一个

双向的点到点的通讯方式。

（2）节点模型接口

主要包括 2 个基本的节点模型，分别是手机模型 cell_mob_node_m 和基站设备模型 cell_site_nd_m_adv。

❑ 手机节点模型（cell_mob_node_m）如图 10-33 所示，共有五个基本功能：

1．通话产生器可以放置多个通话单元。

2．通话服务器能够管理通话请求，通话切换和通话建立/拆除等手机节点功能。

3．响应滤波器能够滤除对于临时手机无用的信息

4．一对接收机/发射机

5．一组天线

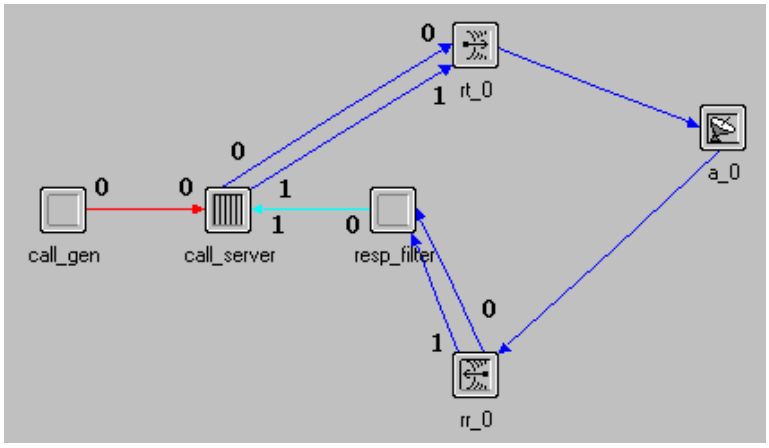


图 10-33 手机设备节点模型（cell_mob_node_m）

call_gen 模块利用进程模型 cell_call_gen，并产生一个输出流，0 输出流用来联系 call_server 模块，call_server 利用 cell_call_svr_m 进程模型，指定 2 个输出流流向发射机（0 是数据流，1 是语音流），还有 2 和输入流，0 来自于 call_gen，1 来自 resp_filter。Resp_filter 有 2 个来自于接收机的输入流，0 是数据流，1 是语音流，还有一个输出流流向 call_server。无论接收机还是发射机都需要连接天线

❑ 基站设备（cell_site_nd_m_adv）节点模型如图 10-34 所示。这个模型实现了移动系统中的小区模型功能，由下面几个基础部分组成：

1. 天线模块

2. 两对接收机和发射机，分别用来进行语音和数据传送

3. 切换模块，它用来监视接收信号强度，并用来进行通话切换的判断

4. bis_m 模块，它用来监视数据接收机来确定忙/闲状态位的正确状态

5. 定位器模块，它周期性的送出手机数据包，并获得信号场强的测量报告，返回后进行包的“自我定位”

6. 一对点对点的发射机/接收机用来提供和 MSTO 的通讯

7. 一个天线点模块，它用来进行自定义小区天线模式的校正和正确控制。它同样可以建立一个自定义的宏

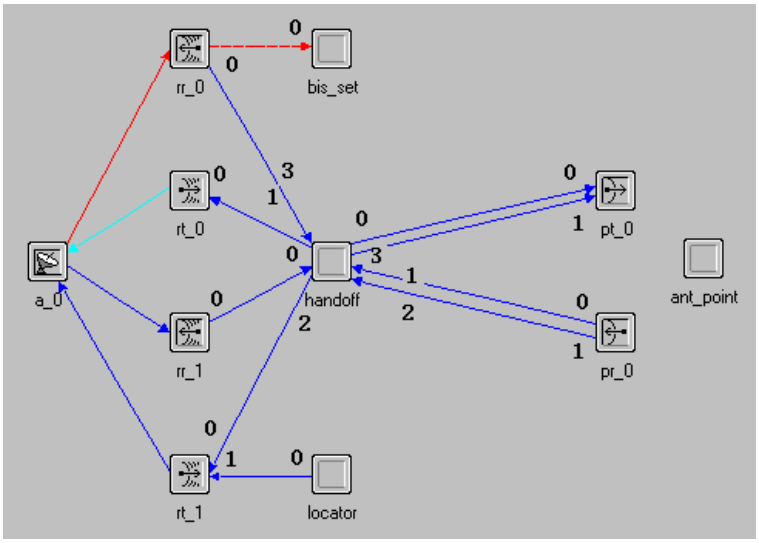


图 10-34　基站设备节点模型（cell_site_nd_m_adv）

切换模型利用 cell_hoff_mgr 进程模型，它用输入流 3 接收数据流，输入流 0 接收语音流（都来自于无线接收机）。它用输入流 1 接收数据流，输入流 2 接收语音流（都来自于 MSTO）。它还通过输出流 1 传送数据流，输出流 2 传送语音流分别送到无线发射机。它并能通过输出流 0 传送数据流，输出流 3 传送语音流分别送到 MSTO 重要的点对点的发射机上。

定位器（locator）模块利用 cell_locator 进程模型，并通过输出流 0 来向 rt_1 传输包数据流。

Bis_set 模块利用 cell_bis_mgr 进程模型，它利用一个统计链路连接 rr_0。这个统计链路使用输入统计量 0，并把“rising edge trigger”和“falling edge trigger”设置成使能状态。

Ant_point 模块利用 cell_ant_point 进程模块，但是它不与任何模块相连接。

第 4 部分　OPNET Modeler 的高级应用

第 11 章　自定义动画编程的运用

11.1 动态队列计量器

有一个 FDDI 令牌环网络，它由 f0~f9 十个节点组成，如图 11-1 所示。通过编写动画程序，可以达到这样的显示效果：每个节点右上角显示一个队列计量器，它能够动态变化（反映队列大小的改变），同时计量器上方以文本方式显示当前队列包的个数和队列总容量。

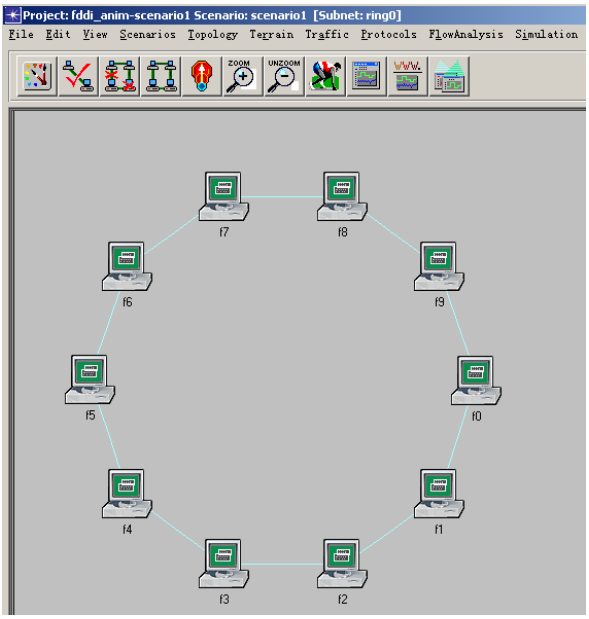


图 11-1 FDDI 令牌环网络模型

11.1.1 设置探针属性

首先要设置自定义动画的探针属性。

有益提示　本课基于 FDDI 令牌环网络模型创建自定义动态队列动画，你也可以按照类似本课步骤将此动画加入感兴趣的网络模型中。首先，你要确认已经切换到想要加动画的网络场景中，然后模仿以下步骤操作。

（1）从 File 菜单中选择 Open…选项。

（2）从下拉列表中选择 Probe Model，找到与当前场景对应的探针模型（默认文件名为：项目名—场景名）。

- 这时出现探针编辑器窗口。
- 你也可以通过在打开的场景窗口中选择菜单“Simulation”→“Choose Statistics(Advanced)”来打开。

（3）单击添加自定义动画按钮。

这时在 Custom Animation Probes 列表中出现新的一行，并且自动命名为 pb3（表明前面已经创建过两个统计量），如图 11-2 所示。

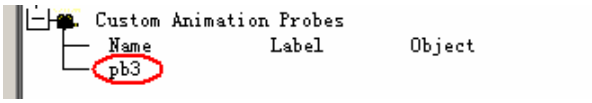


图 11-2　自定义动画探针

有益提示　一个探针模型包含多种类型统计量的收集，自定义动画是其中的一种。

有益提示　每创建一个新的统计量，探针编辑器会自动生成其配置行，并且以 pb*命名，*为按照创建先后顺序（1，2，3…）递增的名称索引号。

（4）在 pb3 上单击鼠标右键，从弹出的菜单中选择 Choose Probed Object

这时出现选择需要探求的网络对象的对话框，，如图 11-3，左边一栏为网络对象列表（为树状结构，表明层次关系）。

（5）如图 11-3 所示选择 top→ring0，单击 OK 按钮关闭对话框。

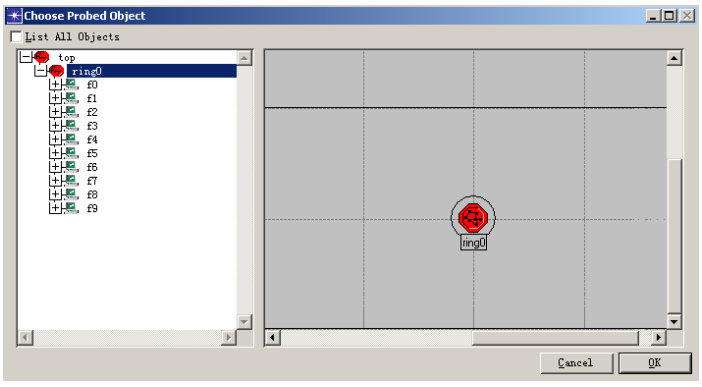


图 11-3　选择探针对象对话框

有益提示 这里 ring0 为 subnet 的名称，在项目编辑其中单击扩展网络按钮  既可以看到需要加动画的网络场景（图 11-1）。之所以选择子网作为动画探求对象是根据提出的问题来确定的，因为要在节点右上角显示一个队列计量器，显然在网络模型中才能看到这一效果。

（6）在 pb3 上单击鼠标右键，从弹出的菜单中选择 Edit Attributes，这时出现动画浏览器属性对话框。

（7）如图 11-4 所示设置动画探针属性。

- 动画探针名称（Network View）。
- 动画标签（Fddi Queue）。
- 动画浏览器名称（Fddi Anim Network View）。
- 其他的属性不大重要（如动画显示起始时间、动画显示终止时间，浏览器窗口的位置和大小等）。

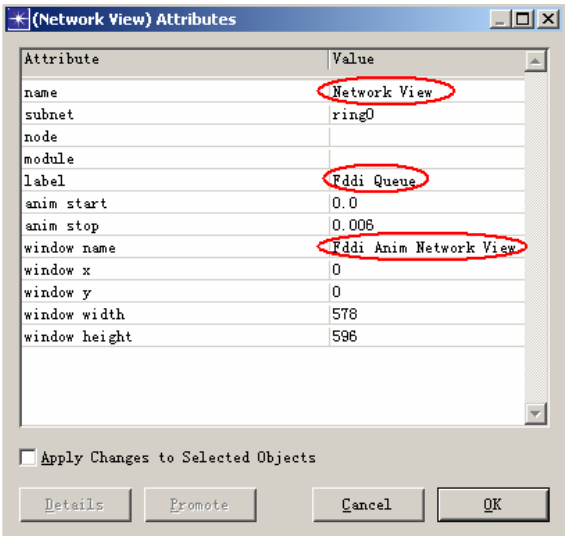


图 11-4　自定义动画探针属性

有益提示 注意，subnet 名称自动变为 ring0，这是步骤（5）选择的结果。

有益提示 动画标签属性很重要，需要通过核心函数 op_anim_lprobe_anvid()与之关联，返回与之对应的动画浏览器 ID 号。动画浏览器名称将在图中红圈里出现，如图 11-5 所示。

11.1.2　动态队列计量器动画程序讲解

接下来就可以在进程模型中编写动画程序了，首先在初始化状态（init 状态）中做一些

前期工作，下面我们对每一句代码进行解释。

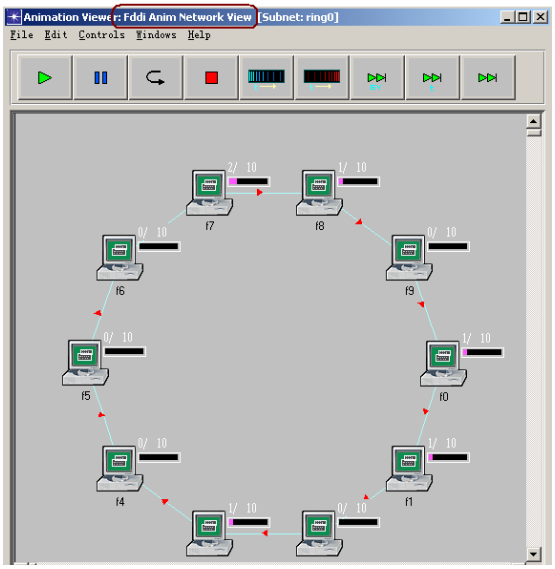


图 11-5　正在播放中的动画浏览器

（1）fddinet_vid = op_anim_lprobe_anvid ("Fddi Queue");

- 通过 op_anim_lprobe_anvid()将动画标签 Fddi Queue 映射为动画浏览器 ID 号 fddinet_vid

（2）op_anim_ime_nmod_draw (fddinet_vid, OPC_ANIM_MODTYPE_NETWORK, "fddi_anim_net","ring0",OPC_ANIM_MOD_OPTION_ICONSUBS, 0);

- 动画浏览中刻画网络模型中的子网“ring0”，这里“fddi_anim_net”是与当前仿真场景相对应的网络模型。

小技巧　当场景拓扑结构进行变动时，运行仿真观察动画会发现显示出的拓扑结构并没有同步更新。因为以上刻画网络模型的语句只认网络模型（*.nt.m）文件，需要重新将改变的拓扑导出为网络模型，并覆盖以前的文件名：
从 Scenarios 菜单中选择 Scenario Component，从弹出的菜单中选择 Export…。

（3）op_anim_ime_nobj_update (fddinet_vid, OPC_ANIM_OBJTYPE_NODE, fddi_sta_name, OPC_ANIM_OBJ_ATTR_ICON, "terminal", OPC_EOL);

- 更新节点图标，不管实际场景中节点图标是什么，这个语句将使动画浏览器中的图标一律变为名称为“terminal”的图标，图标名称与其对应的图形可以查看图标面板，如图 11-6 所示。

（4）fddiqueue_mid = op_anim_macro_create("Queue Anim");

- 通过 op_anim_macro_create()创建名称为“Queue Anim”的动画宏，返回宏 ID 号 fddiqueue_mid。下面要做的动态队列主要通过这个宏来实现，通过这个宏 ID 对它

所指定宏增加动画请求。

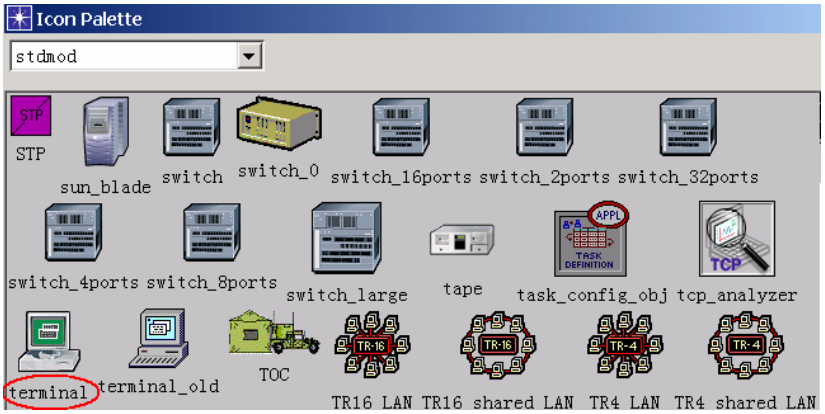


图 11-6　图标面板

（5）op_anim_mme_nobj_pos (fddiqueue_mid, OPC_ANIM_OBJTYPE_NODE, OPC_ANIM_REG_A_STR, OPC_ANIM_REG_X_INT, OPC_ANIM_REG_Y_INT, OPC_ANIM_VERTEX_ICON_NE);

- 从这句代码开始，将要制定动画宏“Queue Anim”。
- 首先这句代码关注名称为 OPC_ANIM_REG_A_STR 的节点模型（OPC_ANIM_OBJTYPE_NODE 表示对象类型为节点），基于这个节点图标在动画浏览器中的位置，取得它东北角的坐标（OPC_ANIM_VERTEX_ICON_NE 代表图标的东北 NorthEast 角，或者称为顶点 VERTEX，相关常量还有很多，都可以通过这种方法来理解其意义），把 X 轴坐标放在 OPC_ANIM_REG_X_INT 中，Y 轴坐标放在 OPC_ANIM_REG_Y_INT 中。

有益提示

注意，这句代码用到三个存储标识符：OPC_ANIM_REG_A_STR，OPC_ANIM_REG_X_INT 和 OPC_ANIM_REG_Y_INT。但是 OPC_ANIM_REG_A_STR 和后两者有着本质的区别，它是代表输入参数，由于尚未指定，所以它代表是一个虚拟的变量，在宏编写完成后调用时要给它赋值。而 OPC_ANIM_REG_X_INT 和 OPC_ANIM_REG_Y_INT 代表结果参数，它是有值的，根据 OPC_ANIM_REG_A_STR 得来，这两个结果参数可供下面的运算使用。

关键概念

OPNET 给每一个宏分配了一些存储空间，这些内存用来保存计算动画参数过程中的一些中间结果，它们本身没有任何意义，也可以把它们看作是为了计算参数而提供的“草稿纸”。即，OPNET 为方便用户制定宏提供了存储中间结果的“草稿纸”，但是 OPNET 对这些“草稿纸”的使用又制定了一些规则，不像我们做加减乘除运算那样直接用数字表示，OPNET 规定每个参与运算的

数字都必须注册，即建立数字与标识符的映射，于是对标识符的运算就等于对数字本身的运算，并且结果也用某个标识符来表示。OPNET 为 int 型、double 型的数字分别提供了 26 个标识符，之所以是“26”个，是因为英文字母 A~Z 正好有 26 个，数量已经足够，并且也便于区分。

（6）op_anim_mgp_reg_set (fddiqueue_mid, OPC_ANIM_REG_C_INT, 5);

- 这句将“5”赋值给 OPC_ANIM_REG_C_INT

（7）op_anim_mgp_arop (fddiqueue_mid, OPC_ANIM_REG_X_INT, OPC_ANIM_AROP_ADD, OPC_ANIM_REG_C_INT, OPC_ANIM_REG_X_INT);

- 这句代码实际上会进行如下运算：

  OPC_ANIM_REG_X_INT = OPC_ANIM_REG_X_INT + OPC_ANIM_REG_C_INT

- 因为 OPC_ANIM_AROP_ADD 为加法的意思，如果我们把存储标识符简化为简单的变量和数字，以上的计算可以写为 X = X + 5，意思为将节点东北角那个点的横坐标加 5，这其实是绘制队列计量器边框的起始点横坐标，之所以加 5 是想与节点图标隔开一定的距离。

（8）op_anim_mgp_reg_set (fddiqueue_mid, OPC_ANIM_REG_C_INT, 2); op_anim_mgp_arop(fddiqueue_mid, OPC_ANIM_REG_X_INT, OPC_ANIM_AROP_ADD, OPC_ANIM_REG_C_INT, OPC_ANIM_REG_A_INT);

- 同上分析，这两句代码做了如下运算：

  A = X + 2，运算的结果暂时存储在 OPC_ANIM_REG_A_INT，这其实是队列计量器的起始点横坐标（X 为计量器边框的起始点横坐标）

（9）op_anim_mgp_arop (fddiqueue_mid, OPC_ANIM_REG_Y_INT, OPC_ANIM_AROP_ADD, OPC_ANIM_REG_C_INT, OPC_ANIM_REG_B_INT);

- 这句代码做了如下运算：

B = Y + 2，运算的结果暂时存储在 OPC_ANIM_REG_B_INT，这其实是队列计量器的起始点纵坐标。

（10）op_anim_mgp_reg_set (fddiqueue_mid, OPC_ANIM_REG_C_INT, 2); op_anim_mgp_arop (fddiqueue_mid, OPC_ANIM_REG_Y_INT, OPC_ANIM_AROP_SUBTRACT, OPC_ANIM_REG_C_INT, OPC_ANIM_REG_D_INT);

- 这句代码做了如下运算：

D = Y - 2，运算的结果暂时存储在 OPC_ANIM_REG_D_INT 中，这其实是文本显示栏的起始点纵坐标（根据我们提出的动画要求：计量器上方以文本方式显示当前队列包的个数和队列总容量）。

（11）op_anim_mgp_setup_end (fddiqueue_mid);

- 这句代码表明动画宏需要用到的参数都已经计算完了，参数设置完毕(_setup_end)。
- 接下来我们来为宏增加绘图请求：

（12）op_anim_mgp_rect_draw (fddiqueue_mid, OPC_ANIM_COLOR_WHITE, OPC_ANIM_REG_X_INT, OPC_ANIM_REG_Y_INT, 44, 10);

- 以 X 和 Y 为起始点画一个长为 44，高为 10 的矩形(_rect_draw)，并且矩形的线条颜色为白色（OPC_ANIM_COLOR_WHITE）。
- X 和 Y 为队列计量器边框的起始点。
- 想象以上代码可以绘制如图 11-7 所示的图形。

图 11-7　绘制的图形

（13）op_anim_mgp_rect_fill (fddiqueue_mid, OPC_ANIM_COLOR_BLACK, " ",OPC_ANIM_REG_A_INT, OPC_ANIM_REG_B_INT, 40, 6);

- 以 A 和 B 为起始点画一个长为 40，高为 6 的矩形，并用黑色（OPC_ANIM_COLOR_BLACK）填充（_rect_fill）。
- A 和 B 为队列计量器的起始点（参考前面的运算结果）。
- 想象以上代码可以绘制如图 11-8 所示图形。

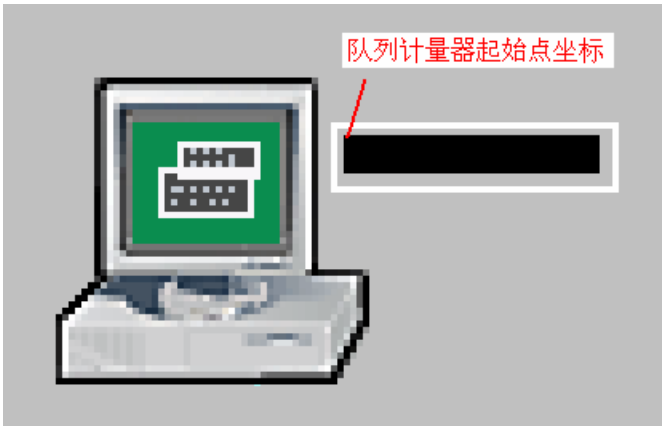


图 11-8　绘制结果图

（14）op_anim_mgp_rect_fill (fddiqueue_mid, OPC_ANIM_COLOR_PINK, " ",

OPC_ANIM_REG_A_INT, OPC_ANIM_REG_B_INT, OPC_ANIM_REG_Q_INT, 6);

- 以 A 和 B 为起始点画一个长为 OPC_ANIM_REG_Q_INT，高为 6 的矩形，并用粉红色（OPC_ANIM_COLOR_PINK）填充。
- A 和 B 为队列计量器的起始点。
- 这里再次用到虚拟参数 OPC_ANIM_REG_Q_INT，这其实是动态队列的长度，真实的取值要根据程序中队列大小来定，这个参数是显示动态队列效果的关键。
- 想象以上代码可以绘制如图 11-9 所示图形。

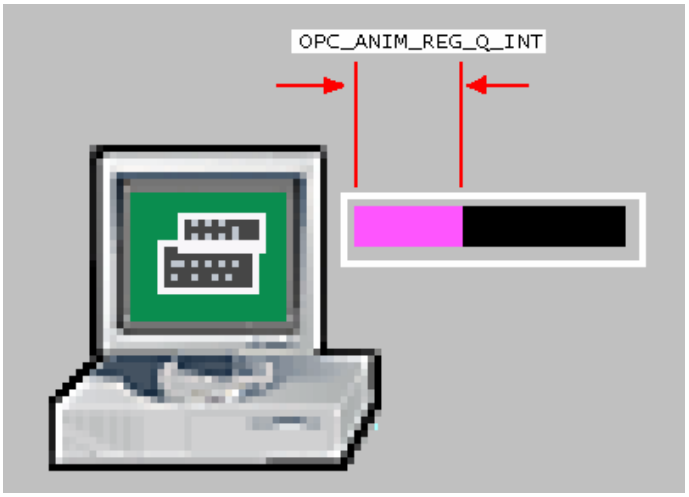


图 11-9　绘制结果图

（15）op_anim_mgp_text_draw (fddiqueue_mid, OPC_ANIM_COLOR_WHITE | OPC_ANIM_ALIGNV_BOTTOM | OPC_ANIM_FONT_OPEN_10, OPC_ANIM_REG_X_INT, OPC_ANIM_REG_D_INT, OPC_ANIM_REG_E_STR);

- 以 X 和 D 为起始点
- 显示字符串为 OPC_ANIM_REG_E_STR 的文本
- 文本为白颜色（OPC_ANIM_COLOR_WHITE）
- 底部对齐（OPC_ANIM_ALIGNV_ BOTTOM：意思为文本框起始点为其左下角）
- 字体为 OPEN 10 号字（OPC_ANIM_FONT_OPEN_10）
- 这里用到虚拟文本参数 OPC_ANIM_REG_E_STR，假设 OPC_ANIM_REG_E_STR = "0/　10"，则可以想象可以绘制如图 11-10 所示图形。

（16）op_anim_macro_close (fddiqueue_mid);

- 至此，这个动画宏就编写完了。下面我们利用编写好的宏绘制动画初始画像。

（17）op_ima_obj_attr_get (op25_topo_child(my_objid, OPC_OBJTYPE_SUBQ, 0), "pk capacity", &queue_depth); sprintf(fddiqueue_str,"0/%5.0f　　", queue_depth); op_anim_igp_macro_draw (fddinet_vid, OPC_ANIM_RETAIN, fddiqueue_mid, OPC_ANIM_REG_A_STR, fddi_sta_name, OPC_ANIM_REG_Q_INT, 0, OPC_ANIM_REG_E_STR, fddiqueue_str,

OPC_EOL);

图 11-10　绘制结果图

- 这段代码首先取出队列的深度（即队列的最大容量）queue_depth（假设它的值为 10），然后根据"0/%5.0f　　"的格式将 queue_depth 合成为字符串"0/　　10"，并将其赋值给 fddiqueue_str，最后调用宏画出队列计量器的雏形。
- 回想之前提到的三个虚拟参数 OPC_ANIM_REG_A_STR（节点模型名称），OPC_ANIM_REG_Q_INT（动态队列长度），OPC_ANIM_REG_E_STR（文本框内容），现在对它们进行赋值：
- OPC_ANIM_REG_A_STR = fddi_sta_name（进程模型对应的节点模型名称）。
- OPC_ANIM_REG_Q_INT = 0　（初试队列长度为 0，没有包到达）。
- OPC_ANIM_REG_E_STR = 0/　　10　（队列长度为 0，队列最大容量为 10）。
- 此时绘制的图形如图 11-10 所示。
- 以上工作是初始化状态中所作的一些前期工作，在仿真运行过程中，可以对这个宏指定特殊的参数，使它不断产生更新的图样，这样在动画浏览器中就能感觉图形动起来。

（18）queuepk_size = op_subq_stat(0,OPC_QSTAT_PKSIZE);

- 取得队列的当前长度。

（19）queue_fill = queuepk_size * 40 / queue_depth;

- 计算应该填充的队列长度。
- 回想之前队列计量器总长度为 40，这里将当前队列大小换算成需填充的长度。

（20）sprintf(fddiqueue_str,"%d/%5.0f　　",queuepk_size,queue_depth);

- 组合要显示的文本，格式为"当前队列大小/　　队列总长度"

（21）op_anim_igp_macro_draw (fddinet_vid, OPC_ANIM_RETAIN, fddiqueue_mid, OPC_ANIM_REG_A_STR, fddi_sta_name, OPC_ANIM_REG_Q_INT, queue_fill, OPC_ANIM_REG_E_STR, fddiqueue_str, OPC_EOL);

- 对虚拟参数进行赋值：

OPC_ANIM_REG_A_STR = fddi_sta_name （进程模型对应的节点模型名称）。
OPC_ANIM_REG_Q_INT = queue_fill（需填充的计量器长度）。
OPC_ANIM_REG_E_STR = 当前队列大小/ 10。

- 并绘制更新参数后的宏，从而生成新的图样。
- 动画浏览器观看到的动画会有如图 11-11 所示的效果。

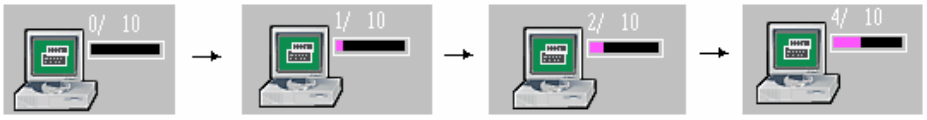


图 11-11 最后动画示意图

11.2 无线包传输

由于 OPNET 自动收集的动画不支持无线数据包的发送的过程显示，也即是不能表现出数据包在空中传播的动态过程，因此有必要实现自定义的无线动画。本节无线动画的显示效果如下：

当节点 B 收到来自节点 A（节点 A 和 B 之间通过无线信道连接）发送的数据包时，节点 B 将判断是否成功接收到该数据包，如果接受成功，B 的图标变为"sat_dish_rcv"  （这是一个黄颜色的小图标），并且在动画浏览器中绘制一条连接 A 和 B 的线段，表示此刻 A 和 B 正在通信，否则表示数据包接收失败，这时不需要绘制连接 A 和 B 的线段，同时 B 的图标变为"sat_dish_jam"  。

考虑到无线传输过程时间跨度很短，标明 A 和 B 成功通信的线段必需在通信完毕后擦除，并且将 B 的图标变为初始状态"sat_dish”  ，表明 B 再次回到空闲状态。

11.2.1 设置探针属性

按照 11.1.1 节类似的步骤设置自定义动画探针属性，关键是设置 label 和 subnet 属性，如图 11-12 所示：

11.2.2 无线包传输动画初始化程序

一般来说，复杂的自定义动画都必须创建动画宏，这部分前期工作在进程模型的初始化状态中完成。下面我们对初始化程序中的每句代码进行解释：

（1）Anvid vid;

```
Anmid   mid;
```

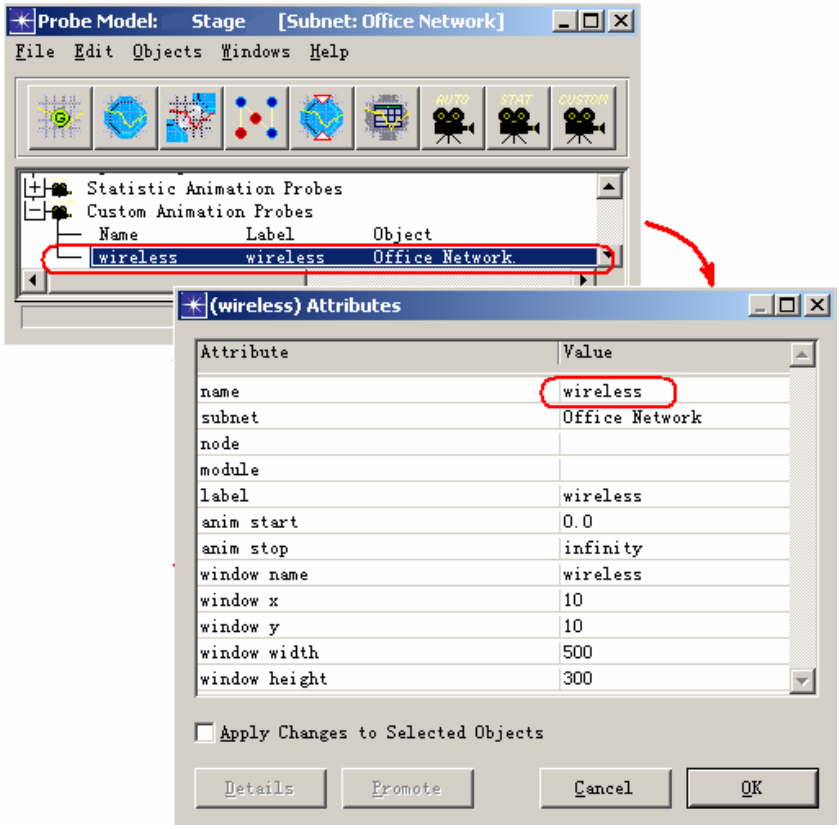


图 11-12 设置自定义动画探针属性

- 在进程模型头块 HB 中定义全局的动画浏览器 ID 号 vid 和动画宏 ID 号 mid，。

（2）vid = op_anim_lprobe_anvid ("wireless");

- 打开动画浏览器，将动画标签 wireless 映射为动画浏览器 ID 号 vid。

（3）op_anim_ime_nmod_draw (vid, OPC_ANIM_MODTYPE_NETWORK, network_scenario_name", "Office Network", OPC_ANIM_MOD_OPTION_NONE, OPC_ANIM_DEFPROPS);

- 在动画浏览器中绘制场景"network_scenario_name"。

（4）mid = op_anim_macro_create ("line_draw");

- 创建名称为"line_draw"的动画宏，其 ID 号为 mid

（5）op_anim_mme_nobj_pos (mid, OPC_ANIM_OBJTYPE_NODE, OPC_ANIM_REG_A_STR, OPC_ANIM_REG_B_INT, OPC_ANIM_REG_C_INT, OPC_ANIM_VERTEX_ICON_CTR);

- 从这句代码开始，我们将创建绘制线段所用的动画宏 mid。
- 假设输入参数为 OPC_ANIM_REG_A_STR 和 OPC_ANIM_REG_D_STR，它们分别用来存储传输和接收节点的名称

- 将传输节点的中心点（常量 OPC_ANIM_VERTEX_ICON_CTR 标识）的 X 和 Y 坐标分别写入 OPC_ANIM_REG_B_INT 和 OPC_ANIM_REG_C_INT 中。

（6）op_anim_mme_nobj_pos (mid, OPC_ANIM_OBJTYPE_NODE, OPC_ANIM_REG_D_STR, OPC_ANIM_REG_E_INT, OPC_ANIM_REG_F_INT, OPC_ANIM_VERTEX_ICON_CTR);

- 将接收节点的中心点 X 和 Y 坐标分别写入 OPC_ANIM_REG_B_INT 和 OPC_ANIM_REG_C_INT 中

（7）op_anim_mgp_setup_end (mid);

- 宏 mid 需要用到的参数设置完毕

（8）op_anim_mgp_line_draw (mid, OPC_ANIM_COLOR_RED | OPC_ANIM_PIXOP_XOR, OPC_ANIM_REG_B_INT, OPC_ANIM_REG_C_INT, OPC_ANIM_REG_E_INT, OPC_ANIM_REG_F_INT);

- 分别以传输和接收节点中心点的坐标为线段的两个端点，采用背景前景颜色异或操作的方式绘制一条线段。

有益提示　"OPC_ANIM_PIXOP_" 为图形颜色填充操作的前缀，默认值为 OPC_ANIM_PIXOP_SRC，表示用前景色填充或绘制图形，而 OPC_ANIM_PIXOP_XOR 用在需要擦除图形的场合，OPNET 中绘图的颜色实际上为 6 比特长度的二进制表示(共有 64 种颜色)，而颜色异或 XOR 操作将背景和前景颜色对应的二进制进行异或，背景颜色 A 和前景颜色 B 异或会得到新的颜色 C，颜色 C 再和前景颜色 B 异或又得到背景颜色 A，两次异或操作的效果将使某个图形的颜色由 C 再次变为背景颜色 A，而动画的显示效果为图形从动画浏览器中擦除了，感兴趣的读者可以自己验证一下。

（9）op_anim_macro_close (mid);

- 宏 mid 制作完毕

11.2.3　在接收功率阶段加入动画程序

动画制作完毕后，接下来要在无线收信机的三个管道阶段中加入自定义动画程序，它们是在仿真运行过程中起作用。以默认管道阶段为例,这三个管道阶段分别是：接收功率阶段（dra_power）、干扰噪声功率计算阶段（dra_inoise）和错误纠正阶段（dra_ecc）。

首先在接收功率阶段加入动画程序，下面对要加入的每句代码进行解释：

（1）extern Anvid vid;

extern Anmid mid;

- 获得已经在初始化状态中设置好的动画浏览器 ID 号和动画宏 ID 号，由于在 11.2.2 节已经将这两个变量设为全局变量，这里只要以外部方式（extern）就可以直接访

问。

（2）Andid* line_ptr;

- 定义图样句柄

（3）char tx_nodename [256], rx_nodename [256];

- 定义传输节点和接收节点名称，作为绘制线段的两个端点。

（4）tx_nodeid = op25_topo_parent (op_td_get_int (pkptr, OPC_TDA_RA_TX_OBJID));
rx_nodeid = op25_topo_parent (op_td_get_int (pkptr, OPC_TDA_RA_RX_OBJID));
op_ima_obj_attr_get (tx_nodeid, "name", tx_nodename);
op_ima_obj_attr_get (rx_nodeid, "name", rx_nodename);

- 分别得到传输和接收数据包的节点名称

（5）line_ptr = (Andid *) op_prg_mem_alloc (sizeof (Andid));

- 为图样句柄 line_ptr 分配内存

（6）*line_ptr = op_anim_igp_macro_draw (vid, OPC_ANIM_RETAIN, mid, OPC_ANIM_REG_A_STR, tx_nodename, OPC_ANIM_REG_D_STR, rx_nodename, OPC_EOL);

- 对虚拟参数 OPC_ANIM_REG_A_STR 和 OPC_ANIM_REG_D_STR 分别赋值为传输节点和接收节点名称，在浏览器中绘制宏，从而生成新的图样，并将该图样的句柄保存为 line_ptr。

有益提示 MAC 层数据包格式需要增加一个包域"line_andid"，如图 11-13 所示

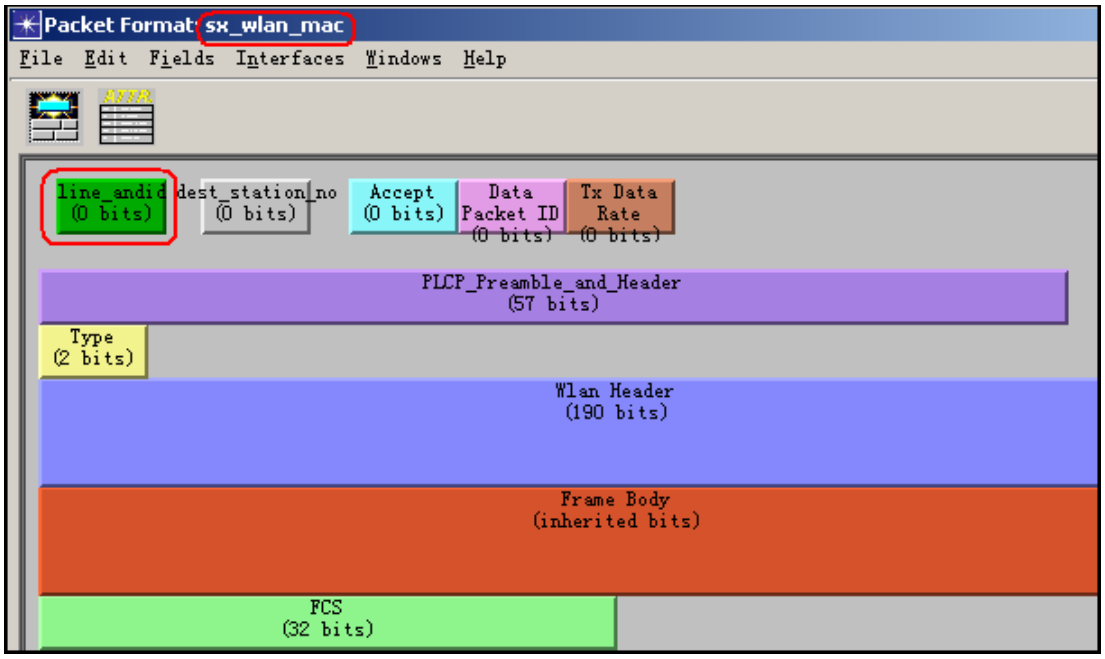


图 11-13 增加包域“line_andid”

（7）if (op_td_get_int (pkptr, OPC_TDA_RA_MATCH_STATUS) ==

```
                                                    OPC_TDA_RA_MATCH_VALID)
        {
        op_anim_ime_nobj_update (vid, OPC_ANIM_OBJTYPE_NODE, rx_nodename,
        OPC_ANIM_OBJ_ATTR_ICON, "sat_dish_rcv", OPC_EOL);
        op_pk_nfd_set (pkptr, "line_andid", line_ptr, op_prg_mem_alloc,
                                                    op_prg_mem_free, sizeof (Andid));
        /* 将图样 ID 号 line_ptr 写入数据包中，
                                        这是在 dra_ecc 阶段中擦除该图样的依据*/
        op_prg_mem_free, sizeof (Andid));
    else
        {
        op_anim_ime_nobj_update (vid, OPC_ANIM_OBJTYPE_NODE, rx_nodename,
            OPC_ANIM_OBJ_ATTR_ICON, "sat_dish_jam", OPC_EOL);
        }
```

- 当节点 B 收到来自节点 A(节点 A 和 B 之间通过无线信道连接)发送的数据包时，节点 B 将判断是否成功接收到该数据包，如果接受成功，B 的图标变为"sta_dish_rcv"，并且在动画浏览器中绘制一条连接 A 和 B 的线段，表示此刻 A 和 B 正在通信，否则表示数据包接收失败，这时不需要绘制连接 A 和 B 的线段，同时 B 的图标变为"sta_dish_jam".

11.2.4　在干扰噪声功率计算阶段加入动画程序

如果数据包正在被接收，无线链路处于正常通信状态，11.2.3 节加入的动画程序已经再现了这一过程（在两个通信节点之间画一条连线，并改变接收节点的图标），但是由于干扰噪声的影响，可能使本次接收无效，而在干扰功率计算阶段（dra_inoise）加入的动画程序将反映出这种情形，下面对要加入的每句代码进行解释：

（1）extern Anvid vid;

```
    int  vid;
    Objid    rx_nodeid;
    char rx_nodename [256];
```

- 定义所需的变量

（2）rx_nodeid = op25_topo_parent (op_td_get_int (pkptr_prev, OPC_TDA_RA_RX_OBJID));

```
op_ima_obj_attr_get (rx_nodeid, "name", rx_nodename);
```

- 得到数据包接收节点的名称

（3）if (prev_match == OPC_TDA_RA_MATCH_VALID)

```
{
op_anim_ime_nobj_update (vid, OPC_ANIM_OBJTYPE_NODE, rx_nodename,
OPC_ANIM_OBJ_ATTR_ICON, "sat_dish_jam", OPC_EOL);
}
```

- 如果前一个数据包仍处在接收状态（OPC_TDA_RA_MATCH_VALID），则当前接收的数据包成为干扰包，这时将接收节点的图标变为"sta_dish_jam"，表示数据包接收失败。

11.2.5 在错误纠正阶段加入动画程序

在经历 13 个管道阶段后，无线数据传输过程就结束了，接收节点又回到空闲状态，因此需要擦除 11.2.3 节中所画的线段，并且将接收节点的图标变为初始状态"sta_dish”  。选择在错误纠正阶段（dra_ecc）加入这段动画程序是因为它是无线管道阶段的最后阶段，下面对要加入的每句代码进行解释：

（1）Andid* line_ptr;
Objid rx_nodeid;
char rx_nodename [256];

- 定义所需的变量

（2）line_ptr = (Andid *) op_prg_mem_alloc (sizeof (Andid));
op_pk_nfd_get (pkptr, "line_andid", &line_ptr);

- 从数据包中取得图样 ID 号 line_ptr，这是下面擦除该图样的依据

（3）op_anim_igp_drawing_erase (vid, *line_ptr, OPC_ANIM_ERASE_MODE_XOR);

- 擦除传输节点至接收节点间的线段

（4）rx_nodeid = op25_topo_parent (op_td_get_int (pkptr, OPC_TDA_RA_RX_OBJID));
op_ima_obj_attr_get (rx_nodeid, "name", rx_nodename);

- 得到数据包接收节点的名称

（5）op_anim_ime_nobj_update (vid, OPC_ANIM_OBJTYPE_NODE, rx_nodename,
OPC_ANIM_OBJ_ATTR_ICON, "sat_dish", OPC_EOL);

- 将接收节点图标改为"sat_dish"，表明此时节点回到空闲状态。

第 12 章 自定义流媒体协议的实现

12.1　OPNET 应用层建模构架

图 12-1 所示为 OPNET 应用层建模构架，根据建模的层次从高到低排列依次为：业务规格建模、应用建模、任务建模和任务阶段建模。

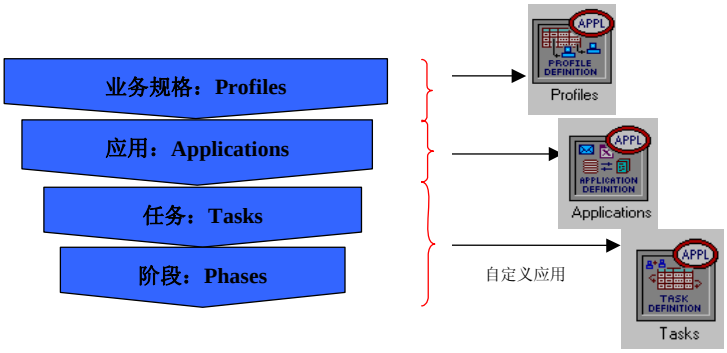


图 12-1　OPNET 应用建模构架

业务规格（或称为业务主询）用来描述用户的行为，用户使用什么类型的应用，什么时候使用这样的应用，使用多久？应用（Applications）具体描述应用的动作，比如说 http 应用，就规定每次取得页面的大小和时间间隔。一个 Profile 可以包含多个 Application。

OPNET 应用层标准模块已经提供多种应用协议（包括 Database、Email、Ftp、Http、Print、Remote Login、Video Conferencing、Voice 等），一般通过自定义配置图 12-2 所示的 Profiles 和 Applications 模块都能达到业务建模的要求。如果仿真的业务不在 OPNET 提供的标准应用协议之内，这时可以采用两种方式自定义业务：（1）将业务分成若干个任务，每个任务由任务阶段组成，通过精细描述应用的每个动作来逼近实际的应用行为。（2）将自定义的应用协议加入 OPNET 标准应用层模块，这涉及到修改 OPNET 多个标准的应用层进程模块和头文件，难度很大，接下来的章节会详细介绍如何自定义地将流媒体传输协议加入到标准应用层。

12.2　自定义的应用协议

图 12-2 为客户与服务器业务会话流程，客户端向服务器发送业务请求，服务器收到请求后，响应客户的要求，给客户发送定量的应用数据。这是一种常用的客户服务器会话模型，因为绝大部分数据是从服务器流向客户（提供服务），所以只在服务器端存在资源竞争。

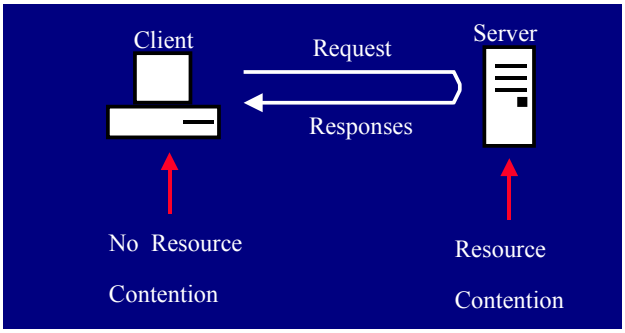


图 12-2 客户－服务器业务会话流程

图 12-3 为客户端建模架构，可以看出它包括四个等级进程模型，实际上它们从上到下互为父进程和子进程的关系。其中 gna_clsvr_mgr 为标准应用层模块的根进程（Root Process）。Network Application Manager 为应用层协议管理模块，它首先向目的地发送业务连接请求包，如果业务会话成功建立，则生成与该业务对相应业务规格的流量，当业务发送完毕时关闭会话。Profile Managers 为业务规格管理模块，由于一种业务规格可能包含多种应用，该模块将为所有的应用分别创建一个子进程，从而并行地生成隶属于不同应用类型的业务。Application Managers 为应用管理模块，一个应用服务器有可能同时为多个客户端提供服务，因此它将为不同的客户端分别创建一个客户端子进程。最后，Application Clients 为客户协议模块，前面三个模块构建了应用层建模的框架，而应用协议的实现主要在该模块中，各种应用（如：Email、HTTP、FTP、Voice、Video 等）的主要区别也体现在客户协议模块的不同。

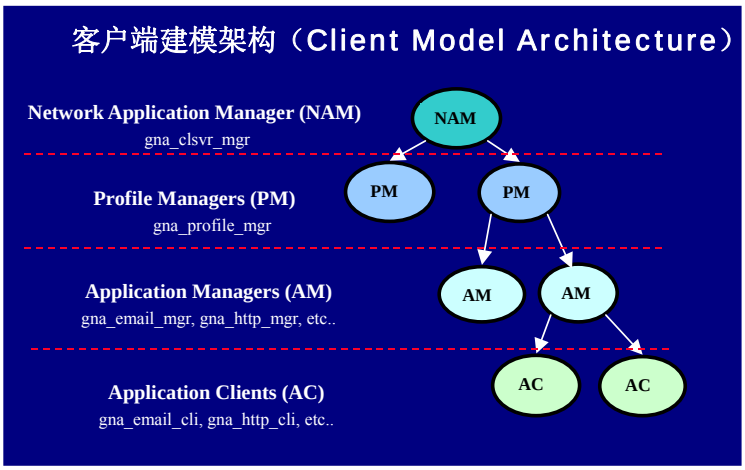


图 12-3 客户端建模架构

众所周知，目前最常用的流媒体播放软件是 Real Player，如图 12-5 所示，它是客户端的应用软件。有了以上的预备知识，下一步就可以开始自定义它的应用协议，用 OPNET 模拟 Real Player 从网上观看视频这一行为。

图 12-5　流媒体播放软件 Real Player

首先必须了解流媒体传输协议，因为最后用 OPNET 实现的协议不可能比我们所理解的还要精确。如图 12-6 所示，客户向服务器发送一个或多个视频播放请求，服务器根据客户的请求和当前带宽资源的情况为客户发送定量的视频包，客户将接收到的视频包放入视频缓存，积累到一定的数据量时即开始连续播放视频（这个延时称为视频启动延时，它是由于首帧采用帧内编码造成的，因为帧内编码 Intra-coding 产生大量数据包）。

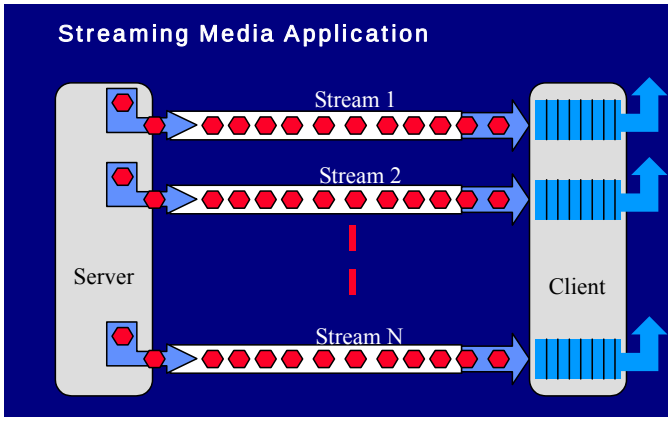


图 12-6　流媒体传输示意图

如图 12-7 所示，流媒体包含如下几个属性：视频文件的大小（Media File Size）、视频流的个数（Number of Streams）、客户端视频回放速度（Playback Speed）、视频包的大小（Pakcet Size）、视频缓存大小（Buffer Size）、视频包使用的传输协议（Transport protocol）。

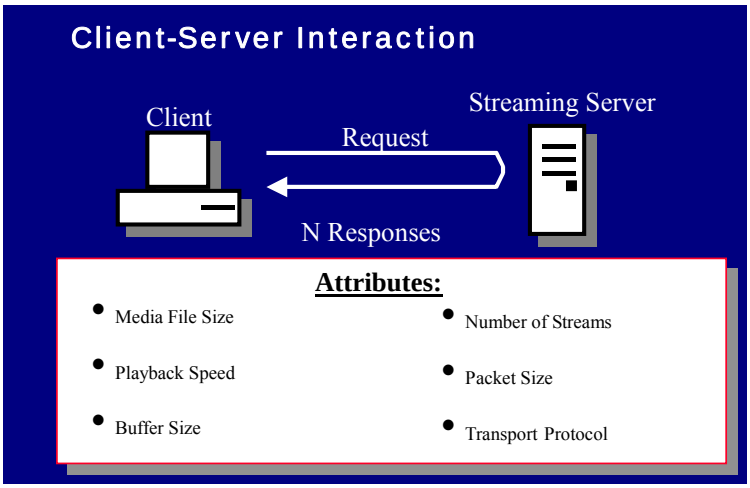


图 12-7　流媒体应用包含的属性

根据以上属性，我们可以通过公式计算客户端期望接收视频包的速率，它等于回放速率（bps）与视频包大小相除。通过公式可以计算服务器端响应客户的视频包的个数，它等于客户请求播放的视频文件大小除以每个视频包的大小。

$$Packet\ Rate = \frac{Playback\ Speed}{Packet\ Size}$$

$$Number\ of\ \mathrm{Re}sponses = \frac{Media\ File\ Size}{Packet\ Size}$$

如图 12-8 所示，从左至右对应业务定义模块的层次从低到高，分别为：任务定义和任务阶段定义、应用定义、业务规格定义。它们的主要功能是对界面的配置参数进行分析，并写入内存，供应用层进程模块进一步使用。

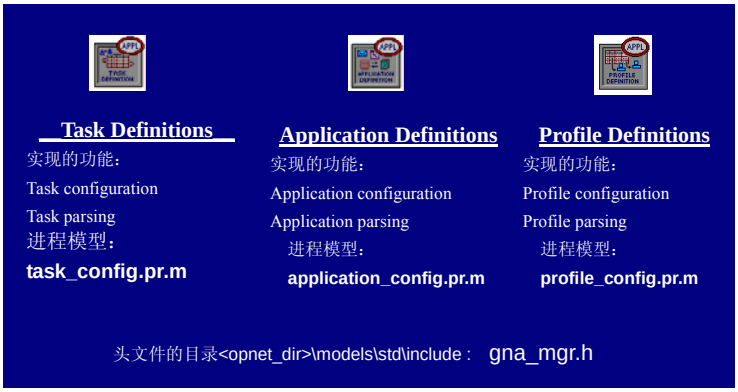


图 12-8　业务定义模块的功能对应的文件

我们可以在目录<opnet_dir>\models\std\applications 找到这三个业务定义模块文件。

图 12-9 为业务属性的层次结构，它描述了属性的等级关系，通过定义模块的处理，图

中所示的直观的多级属性结构将被转化为如图 12-10 所示的抽象复杂多级的结构体。这些结构体参数将提供应用层进程模块实现自定义的应用，因此熟悉它们的定义是成功添加应用协议的关键之一。接着我们描述这些结构体。

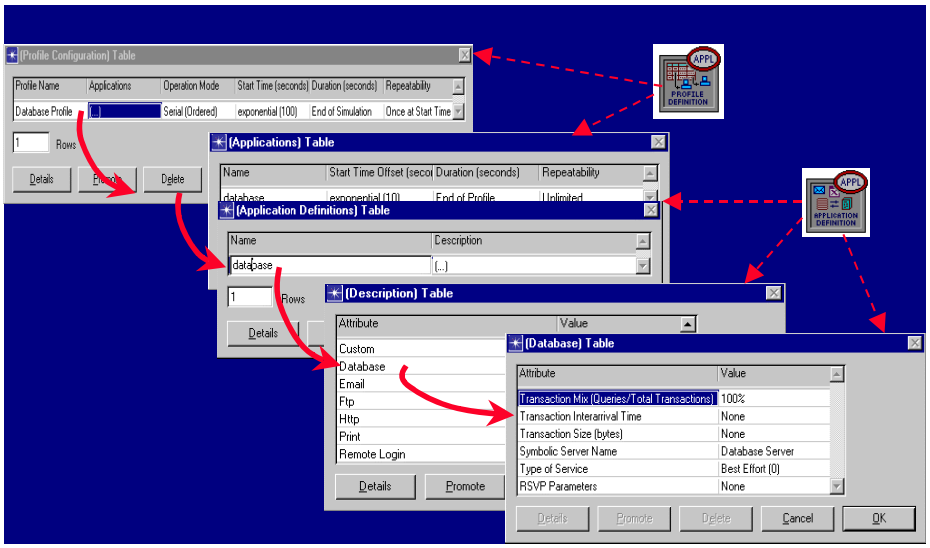


图 12-9　业务属性的层次结构

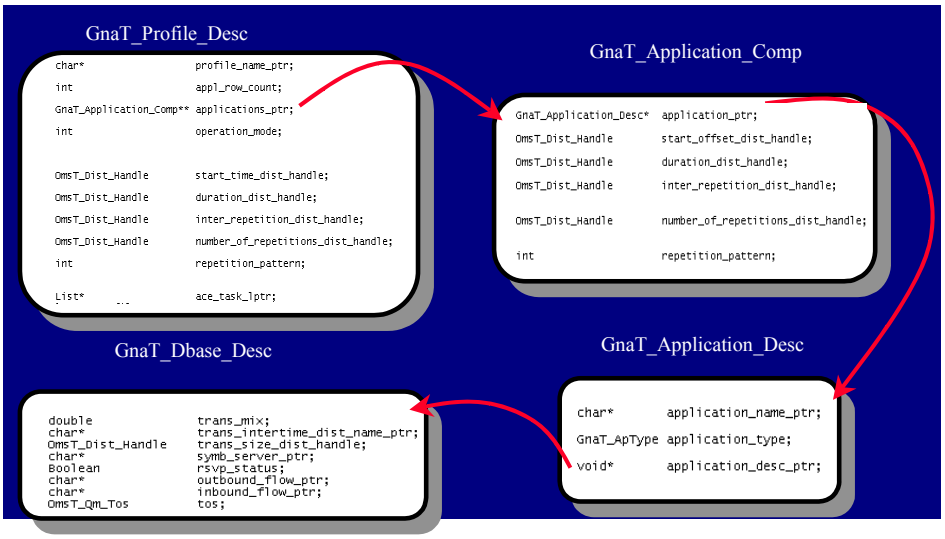


图 12-10　业务建模涉及的数据结构及其继承关系

图 12-10 所示为业务建模涉及的四种数据结构，它们互为父结构与子结构的关系，这种关系和图 12-4 所示的进程继承关系是一致的，换句话说它们分别用在图 12-4 所示的四个进程模型中。这四个结构体的定义在头文件“gna_mgr.h”，我们可以在目录<opnet_dir>\models\std\include 下找到它。

12.3.3 修改头文件“gna_mgr.h”

（1）从 Edit 菜单中选择 Open Edit Pad，这时会弹出一个编辑器，从 File 菜单中选择 Open…，找到<opnet_dir>\models\std\include 目录下的“gna_mgr.h”文件。

❑ 修改该头文件之前请先备份。

（2）在文件的第 103 行，Gna_App_Type 结构体定义中增加新的应用类型，代码如下所示：

```
typedef enum
 {
    GnaT_ApType_Dbase = 0,
    GnaT_ApType_Email,
    GnaT_ApType_Ftp,
    GnaT_ApType_Http,
    GnaT_ApType_Rlogin,
    GnaT_ApType_Print,
    GnaT_ApType_Video,
    GnaT_ApType_Voice,
    GnaT_ApType_Custom,
/*此为 gna_mgr.h 文件的第 103 行*/
    GnaT_ApType_Stream  /*增加新的应用类型*/
    } GnaT_ApType;
```

（3）在文件的第 315 行，GnaT_Stream_Info 结构体定义中增加流媒体应用描述结构体 Gna_Stream_Desc，如下所示。

```
/*此为 gna_mgr.h 文件的第 315 行*/
typedef struct
    {
    OmsT_Dist_Handle    packet_size_dist_handle;
                                                / *包大小分布函数句柄*/
    OmsT_Dist_Handle    file_size_dist_handle;
                                               /*流媒体文件大小分部函数句柄*/
double              buffer_duration;        /*回放延时*/
    int             stream_speed;           /*流媒体传输速率*/
    } GnaT_Stream_Info;     //指定流媒体应用参数的结构体

typedef struct
    {
```

```
int             stream_row_count;
                                /*复合对象的行数,表明同时播放流的个数*/
GnaT_Stream_Info**   stream_info_ptr;
                                /*流媒体应用的相关参数,参考上面的结构定义*/
char*               transport_protocol;     /*所用的传输协议*/
char*              symb_server_ptr;
                                /*服务器的符号名称,此服务器为数据源*/
OmsT_Qm_Tos     tos;            /*服务类型,为服务质量 QoS 的参数*/
Boolean         rsvp_status;    /*是否启用资源预留协议 RSVP 的标志*/
char*              outbound_flow_ptr;
char*              inbound_flow_ptr;
} GnaT_Stream_Desc;     //流媒体业务描述结构体
```

（4）保存头文件 gna_mgr.h。

12.3.4　在应用配置进程模型中增加应用属性

打开进程模型编辑器，在<opnet_dir>\models\std\applications 目录下找到进程模型文件 application_config.pr.m。

- 修改该文件之前请先备份。

从 Interfaces 菜单中选择 Model Attributes。

找到 Applciation Definitions 属性，单击右边的 Default Value 栏。

- 这时打开如图 12-13 所示的应用定义表，在左下角 Rows 栏中输入“1”。

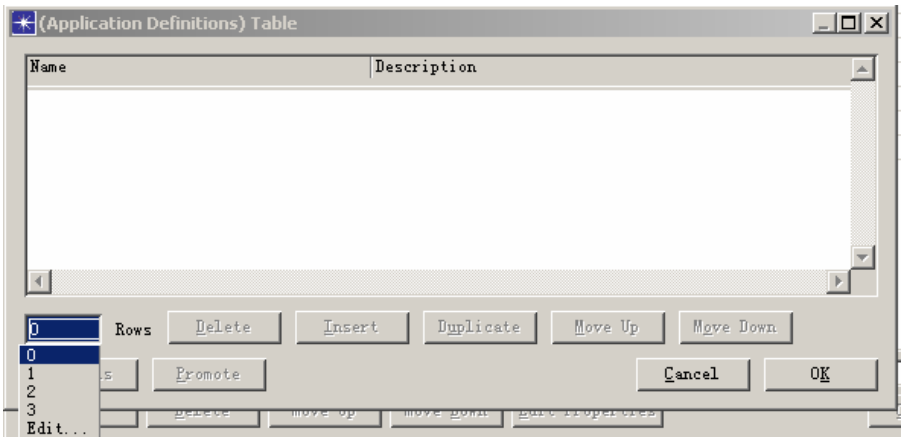


图 12-13　应用定义属性表

输入流媒体应用名称“Streaming Medias”，单击右边的 Description 栏。

这时打开如图 12-14 所示的应用描述表，注意到在蓝色箭头所示位置输入新的应用属性“Streaming Applcation”。然后按照图中提示设置流媒体应用属性表及其默认参数。

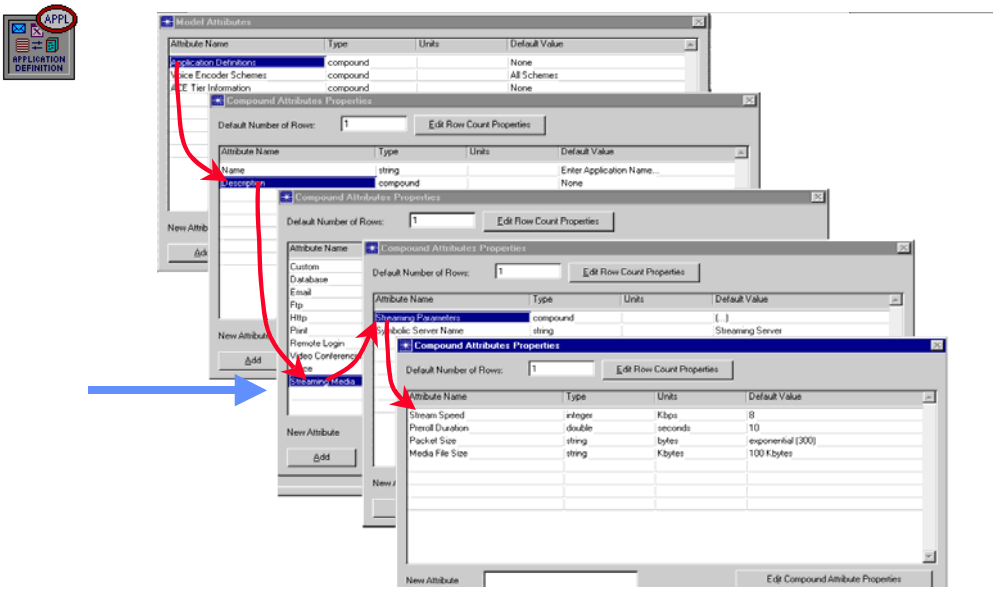


图 12-14　配置应用属性

1．在应用配置进程模块中增加流媒体应用分析程序

（1）打开进程模型编辑器，在<opnet_dir>\models\std\applications 目录下找到进程模型文件 application_config.pr.m，修改该文件之前请先备份。

（2）单击  按钮打开 Function block 编辑器，在第 504 行输入如下代码，该段代码作用是析取属性值并给应用描述结构体赋值。

```
/*此为 application_config.pr.m 文件 Function block 的第 504 行*/
static void*
gna_stream_desc_parser (Objid application_row_objid, Boolean application_
already_parsed)
    {
    Objid                   temp_row_objid, temp_comp_objid;
    Objid                   temp2_row_objid, temp2_comp_objid;
    GnaT_Stream_Desc*       stream_desc_ptr;
    char                    dist_args [256];
    char                    temp_name [256];
    int                 stream_speed;
    int                 stream_index;
    int                 number_of_streams;

    /** 从应用配置模块属性中获取与流媒体应用相关的参数,写入流媒体应用描述结构体中**/
    FIN   (gna_stream_desc_parser   (application_row_objid,   application_
already_parsed));
```

```
    /*获取属性对象"Streaming Media"的 Objid*/
    op_ima_obj_attr_get   (application_row_objid,   "Streaming   Media",
    &temp_comp_objid);
    temp_row_objid = op_topo_child (temp_comp_objid, OPC_OBJTYPE_GENERIC,
    0);
    /*获取复合属性对象"Streaming Parameters"的 Objid*/
    op_ima_obj_attr_get    (temp_row_objid,    "Streaming    Parameters",
    &temp2_comp_objid);
    /*得到复合属性的行数,表明同时播放流的个数*/
 number_of_streams     =     op_topo_child_count     (temp2_comp_objid,
OPC_OBJTYPE_GENERIC);

    /*如果包含一个或以上的流,则分别解析其属性值*/
    if (number_of_streams > 0)
        {
        /*该子程序只能够执行一次,如果再次执行则出现逻辑错误*/
        if (application_already_parsed)
            {
            op_sim_end ("More than one application is defined", "", "", "");
            }
        /*分配存储流媒体参数的内存空间*/
        stream_desc_ptr = (GnaT_Stream_Desc*) op_prg_mem_alloc (sizeof
        (GnaT_Stream_Desc));
        stream_desc_ptr->stream_row_count = number_of_streams;
        stream_desc_ptr->stream_info_ptr = (GnaT_Stream_Info**)
        op_prg_mem_alloc (sizeof (GnaT_Stream_Info*) * number_of_streams);
        for  (stream_index  =  0;  stream_index  <  number_of_streams;
        stream_index++)
            {
            /*得到复合属性中当前行的 Objid,这是获取当前行对应应用参数的依据*/
          temp2_row_objid     =     op_topo_child     (temp2_comp_objid,
          OPC_OBJTYPE_GENERIC, stream_index);
          stream_desc_ptr->stream_info_ptr       [stream_index]       =
          (GnaT_Stream_Info*)
             op_prg_mem_alloc (sizeof (GnaT_Stream_Info));
          /*将属性值写入结构体中*/
          op_ima_obj_attr_get (temp2_row_objid, "Stream Speed",
                               &stream_desc_ptr->stream_info_ptr
                               [stream_index]->stream_speed);
          op_ima_obj_attr_get (temp2_row_objid, "Preroll Duration",
                               &stream_desc_ptr->stream_info_ptr
```

```
                                [stream_index]->buffer_duration);
            op_ima_obj_attr_get    (temp2_row_objid,    "Packet    Size",
            dist_args);
            stream_desc_ptr->stream_info_ptr [stream_index]->packet_size_
            dist_handle =
oms_dist_load_from_string (dist_args);
            op_ima_obj_attr_get  (temp2_row_objid,  "Media  File  Size",
            dist_args);
            stream_desc_ptr->stream_info_ptr  [stream_index]->file_size_
            dist_handle =
oms_dist_load_from_string (dist_args);
            }
        op_ima_obj_attr_get  (temp_row_objid,  "Symbolic  Server  Name",
        temp_name);
        stream_desc_ptr->symb_server_ptr  =  (char  *)  op_prg_mem_alloc
        (strlen (temp_name) + 1);
        strcpy (stream_desc_ptr->symb_server_ptr, temp_name);
        FRET (stream_desc_ptr);
        }
    else
        {
        /* 如果不包含任何流应用,则返回空指针*/
        FRET (OPC_NIL);
        }
    }
```

（3）在第 245 行输入如下代码，增加一个新的应用。

```
/*此为 application_config.pr.m 文件 Function block 的第 245 行*/
    temp_application_ptr = gna_stream_desc_parser (application_row_objid,
    application_already_parsed);
    if (temp_application_ptr != OPC_NIL)
        {
        /*写入解析好的流应用描述参数*/
        application_ptr->application_desc_ptr = temp_application_ptr;
        /*标记与业务描述参数结构体对应的业务类型*/
        application_ptr->application_type = GnaT_ApType_Stream;
        /*标记该种应用已经被解析过*/
        application_already_parsed = OPC_TRUE;
        }
```

（4）编译模块。

2．在业务规格管理进程模型中增加流媒体应用启动程序

（1）打开进程模型编辑器，在<opnet_dir>\models\std\applications 目录下找到进程模型文件 gna_profile_mgr.pr.m。该进程模型及其功能如图 12-17 所示。

修改该文件之前请先备份。

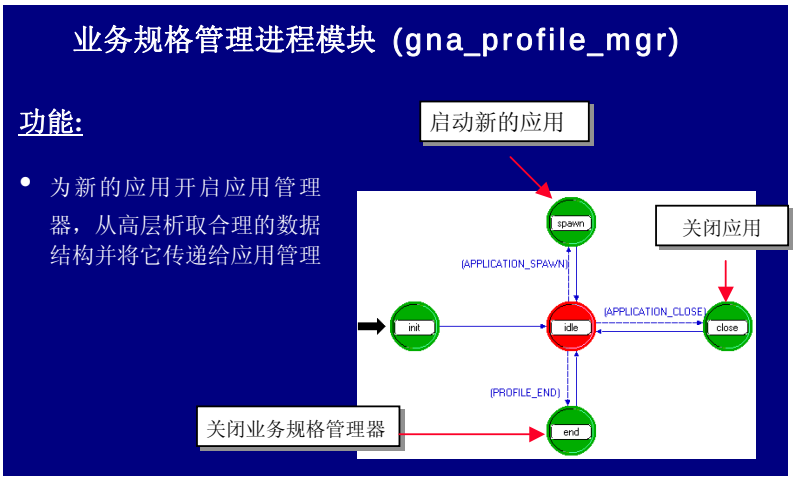


图 12-17　业务规格管理进程模型

（2）在业务规格管理进程中的 SPAWN 状态中第 168 行输入如下代码，创建应用管理进程。

```
/*此为 gna_profile_mgr 进程模型 spawn 状态入口执行代码的第 168 行*/
/*业务规格管理进程根据 application_ptr->application_type 指示的业务类型来创建相应
  应用的管理进程*/
    case GnaT_ApType_Stream:
        {
        /*分配进程句柄内存*/
        application_mgr_prohandle_ptr  =  (Prohandle*)  op_prg_mem_alloc
        (sizeof (Prohandle) * int_traffic_growth_factor);
        /*int_traffic_growth_factor 是一个仿真属性的值,
          表示业务的增长因子,它的值为同种应用业务生成个数,默认值为 1*/
        for (ith_app = 0; ith_app < int_traffic_growth_factor; ith_app++)
            {
/*创建一个流媒体应用管理进程*/
            application_mgr_prohandle_ptr   [ith_app]   =   op_pro_create
            ("gna_stream_mgr", OPC_NIL);
            }
        break;
        }
```

注意变量 int_traffic_grouth_factor 对应仿真属性“Traffic Scaling Factor”，它主要用来设置一系列比例因子来成比例地使业务增长或减小，从而得到不同业务负载下的一系列仿真结果。默认值为“1”，意味着只运行一次仿真，业务按原始定义分配，此时该变量不起作用。

（3）编译模块。

3．创建流媒体应用管理进程模型（gna_stream_mgr）

（1）从 Edit 菜单中选择 Open Edit Pad，这时会弹出一个编辑器，从 File 菜单中选择 Open…，找到<opnet_dir>\models\std\include 目录下的“gna_mgr.h”文件。

（2）在第 607 行加入如下代码，创建流媒体应用参数结构体，并保存文件。

```
/*此为 gna_mgr.h 文件的第 607 行*/
/*以下为结构体 GnaT_Cli_Stream_Params_Info 的定义,它给出实现流媒体协议的主要参数,
  这些参数在业务会话的开始由流媒体管理进程传给流媒体客户进程*/
typedef struct
    {
   int             number_of_responses;   /* 客户要求流媒体服务器提供包(响应
   包)的总个数*/
    int             strm_index;         /* 流媒体索引号,区分不同的流*/
    int             preroll_buffer_size;    /* 回放缓存的大小*/
    double          req_packet_size;        /* 客户端请求包的大小*/
    double          resp_packet_size;       /* 响应包的大小*/
    double          resp_service_time;      /* 处理响应包所需的服务时间*/
    double          stream_speed;       /* 流媒体传输速率*/
    double          estimated_plaback_time;/* 在没有网络延时情况下的估计回放
    时间*/
    double          end_time;           /* 流媒体播放结束时间*/
    char*               server_name;        /* 服务器名称*/
    GnaT_Nam_Appl*      app_info_ptr;       /* 与流应用相关的信息*/
    OmsT_Qm_Tos     tos;                /* 服务质量类型*/
    GnaT_Cli_Mgr_Session*     session_ptr;      /* 与当前业务会话相关的信息*/
    } GnaT_Cli_Stream_Params_Info;
```

根据流媒体应用协议和如图 12-20 所示的非持续连接协议，创建如图 12-21 所示的流媒体应用管理进程模块（gna_stream_mgr.pr.m）。

4．创建流媒体客户进程模型

（1）创建如图 12-22 所示的流媒体客户进程模块（gna_stream_cli.pr.m），因为该进程涉及与传输适应层（TPAL）的接口，熟悉客户端如何向传输适应层发起连接会话请求。

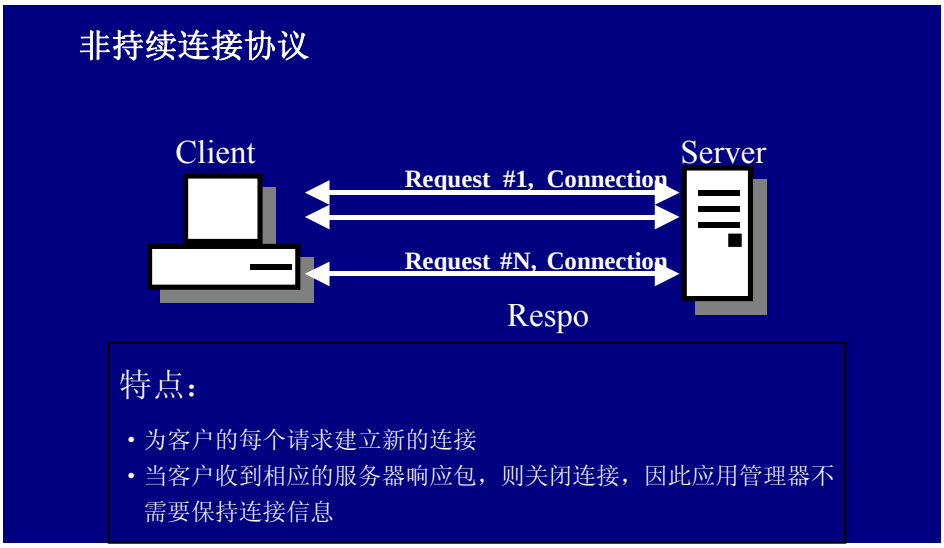


图 12-20　非持续连接传输协议

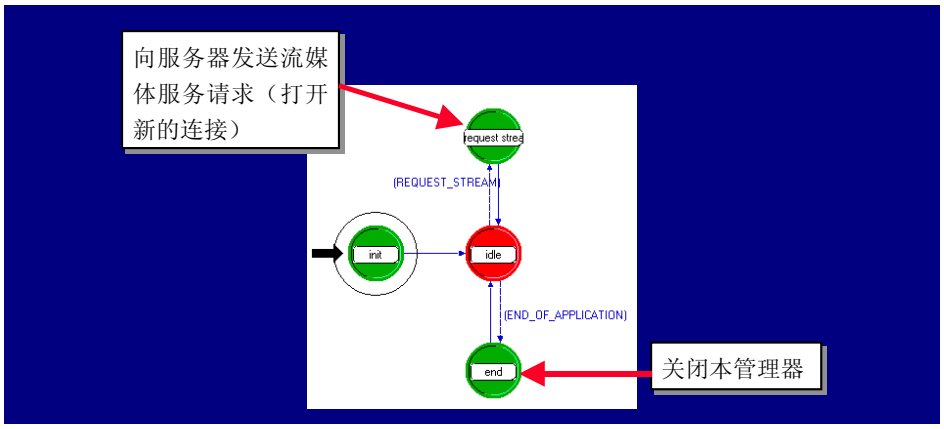


图 12-21　流媒体应用管理进程模型

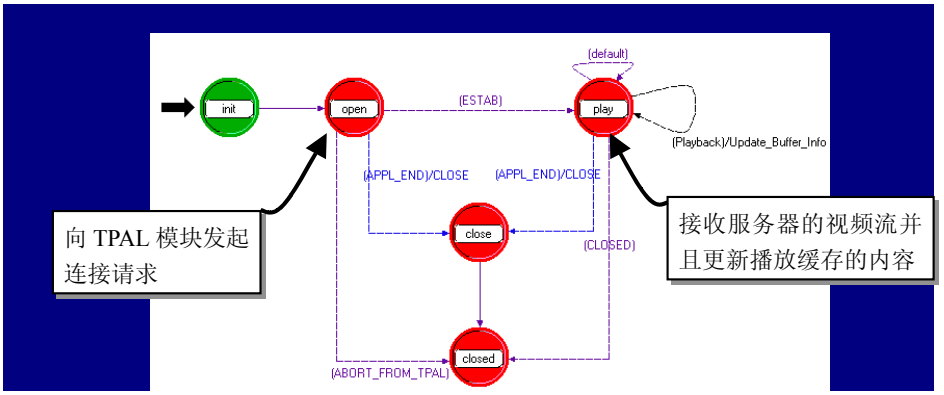


图 12-22　流媒体客户进程模型

5．修改网络应用支持头文件

（1）从 Edit 菜单中选择 Open Edit Pad，这时会弹出一个编辑器，从 File 菜单中选择 Open…，找到<opnet_dir>\models\std\include 目录下的“gna_support.h”文件。修改该头文件之前请先备份。

（2）在第 100 行的位置增加流媒体应用名字，代码如下。

```
typedef enum GnaT_Application_Name
    {
    GnaC_App_Custom_Application,
    GnaC_App_Database_Entry,
    GnaC_App_Database_Query,
    GnaC_App_Email,
    GnaC_App_Ftp,
    GnaC_App_Http,
    GnaC_App_Print,
    GnaC_App_Remote_Login,
    GnaC_App_Video_Conferencing,
    GnaC_App_Voice,
 /*此为 gna_support.h 文件的第 100 行*/
    GnaC_App_Stream         /*增加流媒体应用名称标识*/
    } GnaT_Application_Name;
```

（3）在第 118 行增加流媒体应用类型，代码如下。

```
typedef enum GnaT_Application_Type
    {
    GnaC_App_Type_Custom_Application,
    GnaC_App_Type_Database,
    GnaC_App_Type_Email_Send,
    GnaC_App_Type_Email_Recv,
    GnaC_App_Type_Ftp_Get,
    GnaC_App_Type_Ftp_Put,
    GnaC_App_Type_Http,
    GnaC_App_Type_Print,
    GnaC_App_Type_Remote_Login,
    GnaC_App_Type_Video_Conferencing,
    GnaC_App_Type_Voice,
    GnaC_App_Type_Ace,
 /*此为 gna_support.h 文件的第 118 行*/
    GnaC_App_Type_Stream            /*增加流媒体应用类型标识*/
    } GnaT_Application_Type;
```

（4）在第 153 行的位置增加流媒体应用端口，代码如下所示。每个应用会话的连接和应用数据的传输都是基于传输适应层 TPAL 响应的端口进行的。

```
typedef enum
    {
    Ftp = 20,
    Rlogin = 23,
    Email = 25,
    Http = 80,
    Video = GNAC_PORT_BASE,
    Database,
    Print,
    Cust_App,
    Voice,
 /*此为 gna_support.h 文件的第 153 行*/
    Stream      /*增加流媒体应用端口*/
    } GnaT_App;
```

6．修改网络应用管理进程模块

（1）打开进程模型编辑器，在<opnet_dir>\models\std\applications 目录下找到进程模型文件 gna_clsvr_mgr.pr.m。

修改该文件之前请先备份。

（2）申明流媒体相关的本地和全局统计结果收集变量。

（3）从 Interfaces 菜单中选择 Model Attributes。

（4）找到 Transport Protocol 属性，单击右边的 Default Value 栏。

（5）增加属性 streaming application，并设置默认值为 UDP。

（6）单击　按钮打开 Function block 编辑器，在第 1682 行输入如图如下程序。

```
/*此为 gna_clsvr_mgr 进程模型 Function block 的的 1682 行*/
case GnaT_ApType_Stream:
        {
        serv_index = Print;
/* 获取流媒体应用所用的传输协议:UDP 或者 TCP*/
       op_ima_obj_attr_get   (prtcl_cattr_id,   "Streaming   Transport",
       protocol);
/* 检查是否需要收集结果统计量*/
        op_ima_obj_attr_get (comp_attr_objid, "Service Status", &status);
        /* 如果不需要搜集结果,则下面代码跳过*/
        if (status == OPC_FALSE)  continue;
/* 注册所要收集的统计量*/
```

```
        gna_clsvr_mgr_stat_initialize
("Streaming",  application_name_ptr,  stat_indexes_table  [GnaT_ApType_
Stream], p_speed, overhead);
stat_indexes_table [GnaT_ApType_Stream]++;
        }
        break;
```

7．建立新项目，设置应用定义参数和业务规格定义参数并运行仿真

（1）新建一个项目，命名为 steaming，配置如图 12-27 所示的网络模型。

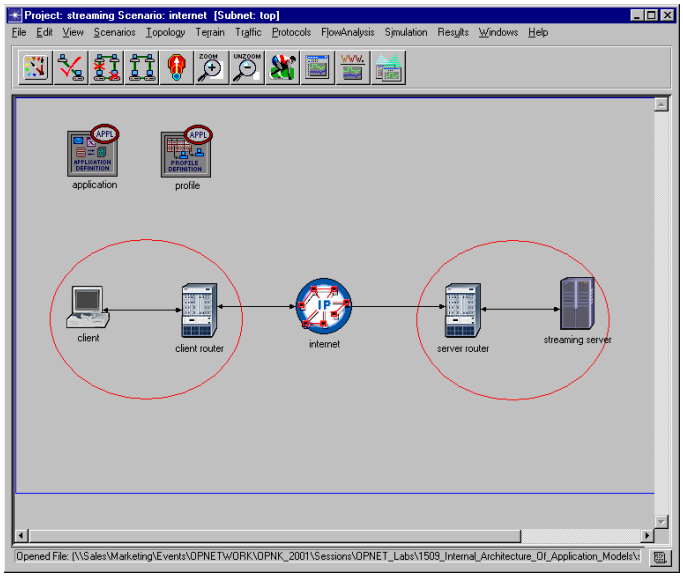


图 12-27　网络模型

（2）右击应用定义节点（application），在弹出的菜单中选择 Edit Attributes。

（3）如图 12-28 所示设置流媒体应用参数。

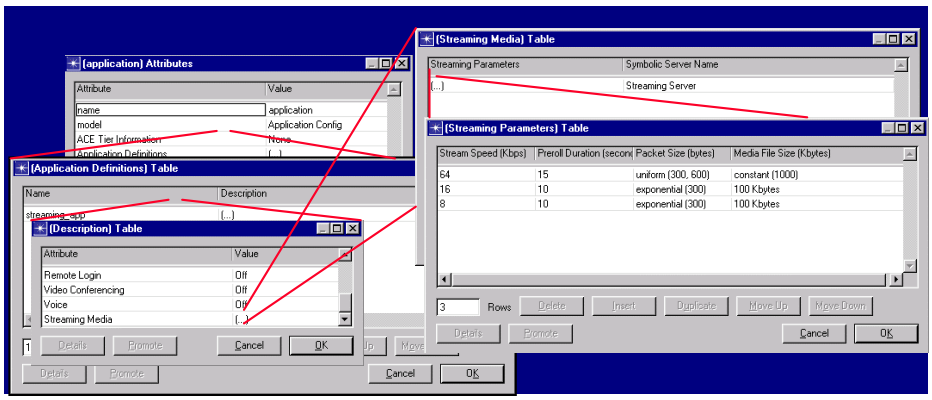


图 12-28　设置流媒体应用参数

（4）右击业务规格定义节点（profile），在弹出的菜单中选择 Edit Attributes。

（5）如图 12-29 所示设置流媒体业务规格参数。

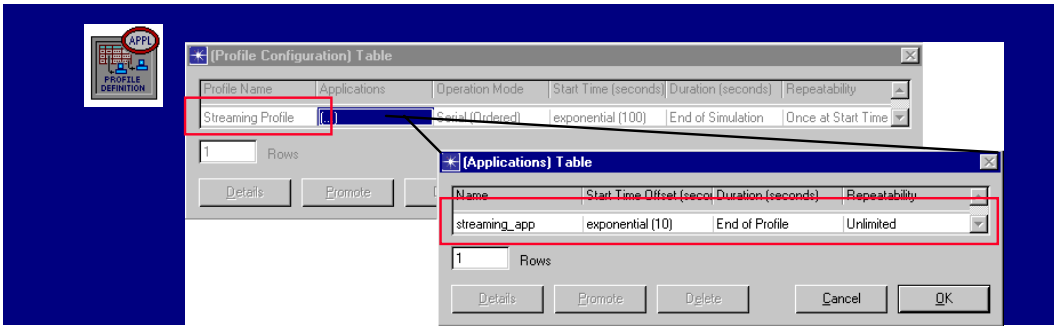


图 12-29　设置流媒体业务规格参数

（6）运行仿真。

第 13 章　自定义 IP 协议的实现

IP 协议已经是一个非常成熟而庞大的协议体系，一般很少对 IP 协议本身进行研究，我们只要清楚它的接口，能够在 OPNET 中使用它就可以了。IP 协议的接口主要是 IP 包和 IP 地址。OPNET 自带的许多设备模型都采用 OPNET 标准协议栈模块，对于这些设备构成的网络场景，可以在项目编辑器中的 protocol 菜单中选择 IP->auto assigned address 自动分配设备的 IP 地址

OPNET 自带的 IP 协议接口并不复杂，关键问题是它和 OPNET 其他标准的协议模块相关联，如传输层模块（tcp）、传输适应层模块（tpal）以及应用层模块（application）协议。如果采用的业务是自定义的（例如采用读取文件的方式导入业务），这些业务要融入到 OPNET 标准应用层中，可以采用类似第 11 章的操作或者编写 EMA 程序将业务加载到 OPNET 标准应用层中，这个工作量很大。相比之下，自定义 IP 协议对应用层没有任何特殊的要求，但是不如 OPNET 标准 IP 协议真实，它涉及 IP 协议的浅显功能，如封装高层数据包，解封底层数据包和 IP 路由等。

13.1 自定义 IP 协议接口

首先建立自定义包格式，如图 13-1 所示，将其命名为"Custom_IP"。

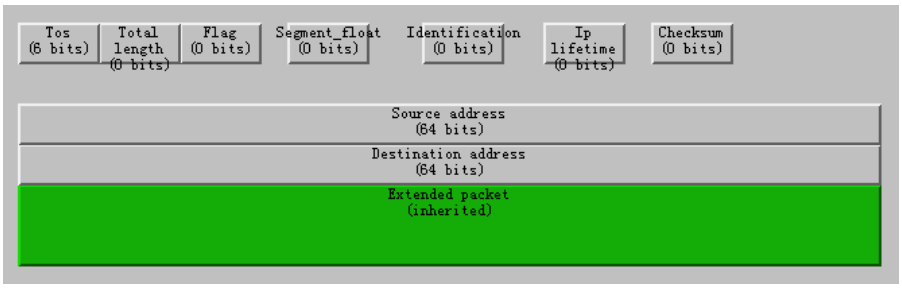


图 13-1 自定义 IP 包的格式

在节点编辑器的 Interfaces 菜单中选择 Model Attribute，添加节点属性“Addr_IP”，它是 string 型的属性，标记节点的 IP 地址，格式为"xxx.xxx.xxx.xxx"，如图 13-2 所示。

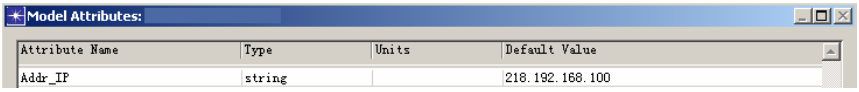


图 13-2　添加 IP 地址属性

在项目编辑器中选中该节点，单击鼠标右键。从弹出的菜单中选择 Edit Attributes 则可以看到可编辑的 Addr_IP 属性栏，作为 IP 地址信息的输入界面，如图 13-3 所示。

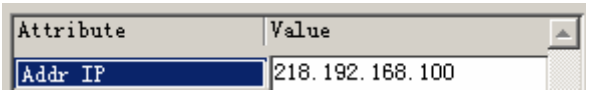


图 13-3　IP 地址的输入界面

13.2　IP 包的创建和高层数据包的封装

有了 13.1 节定义的 IP 包格式和 IP 地址属性，我们就可以实现 IP 包的创建，IP 地址的读取以及源 IP 地址写入 IP 包等操作，这部分的程序可以在进程的初始化状态中完成，因为源 IP 地址始终为节点本身的 IP 地址。初始化状态中创建好一个 IP 空包并且在其中填写了源 IP 地址后就制作好了一个 IP 样本包。当高层数据包到达时，首先对 IP 样本包进行拷贝（这样可以保留样本包），再将高层包封装进 IP 拷贝包，剩下的工作只需填写目的 IP 地址。下面我们就来简单分析这个过程的代码。

```
Packet* ip_pkptr, ip_pkptr_tmp ;
char addr_ip[32];
unsigned long int src_ip_addr,dest_ip_addr;

//创建自定义格式的 IP 包
ip_pkptr = op_pk_create_fmt("Custom_IP");
//得到本节点的 IP 地址，格式为“xxx.xxx.xxx.xxx”，它是一个字符串
op_ima_obj_attr_get(op_id_self(), "Addr_IP" , addr_ip);
//将字符串型的 IP 地址通过 prg_ip_address_string_to_value 函数转成整型 IP 地址，这样符合标准 IP 协议的 IP 地址读写规范
src_ip_addr = prg_ip_address_string_to_value(addr_ip);
//设置 IP 源地址，IP 样本包制作完成
op_pk_nfd_set(ip_pkptr,"Source address",src_ip_addr);
//得到高层数据包
app_layer_pkptr = op_pk_get(op_intrpt_strm());
//拷贝样本包
ip_pkptr_tmp = op_pk_copy(ip_pkptr);
//设置 IP 目的地址
```

```
op_pk_nfd_set(ip_pkptr_tmp,"Destination address",dest_ip_addr);
//封装高层数据包
op_pk_nfd_set(ip_pkptr_tmp, "Extended packet", app_layer_pkptr);
//最后将创建好的 IP 包发往低层
op_pk_send(ip_pkptr,IP_LAYER_OUT_STRM);
```

小技巧 加入以下代码后，在 ODB 调试窗口中输入命令 ltrace ip_pk 可以打印 IP 包的相关信息，这样能够看 IP 包的创建是否正确:

```
if (op_prg_odb_ltrace_active("ip_pk")==OPC_TRUE)
    {
    //取得 IP 包的源和目的 IP 地址
    op_pk_nfd_get(ip_pkptr,"Source address",&src_ip_addr);
    op_pk_nfd_get(ip_pkptr,"Destination address",&dest_ip_addr);
    //采用函数 prg_ip_address_value_to_string
     将无符号长整型的 IP 地址转换为字符串型的地址以便显示
    prg_ip_address_value_to_string(src_ip_addr,addr_ip);
    printf("\n");
    printf("|--------------------------------------------------------------------------|\n");
    printf("|          Source address:%s \n",addr_ip);
    prg_ip_address_value_to_string(dest_ip_addr,addr_ip);
    printf("|       Destination address:%s |\n",addr_ip);
    printf("|--------------------------------------------------------------------------|\n");
    }
```

13.3 IP 路由表初始化

路由器通过自学习方式得到与之有线相连节点的 IP 地址前缀，并将这些 IP 地址前缀写入路由器的路由表中。下面我们来分析这个过程的代码，这些程序放在初始化状态中使用。

我们采用数据链表的方式构造路由表，

首先定义链表元素结构体：

```
struct route {
  unsigned long int netprefix;
  int MAC_addr;
  struct route *nextunit;
  };
```

//这里 netprefix 存储 IP 地址前缀，ObjId 存储与连接对应的流索引号，nextunit 指向路由表的下一项内容

//定义状态变量 route_table_ptr

struct route *route_table_ptr;

（2）编写初始化程序所需变量

//路由表初始化所必需的变量

```
struct route *table_item1 = route_table_ptr ;
struct route *table_item2 ;
```

//路由发现所必需的变量

```
Objid self_if;
Objid IP_rte_id;
    Objid xmt_id;
    Objid remote_rcv_id;
    Objid remote_node_id;
```

//IP 地址读取所必需的变量

```
char addr_ip[32];
unsigned long int addr;
```

（3）初始化路由表，将 tmp_ptr 再连接一个链表元素 temp_ptr1

```
table_item1->nextunit = table_item1 + 1 ;
table_item2 = table_item1->nextunit ;
table_item2->nextunit = NULL ;
```

有益提示　这里我们假设与路由器有线相连的节点为 2 个，感兴趣的读者可以自己增加个数。

（4）接下来我们要根据每个有线发射机的连接关系自适应地查找到与之相连节点的 IP 地址前缀，如图 13-3 所示。

首先查找与名称为"xmt0"有线发信机相连节点 Objid。

```
//得到进程模块 Objid
```

```
 self_id = op_id_self ();
//得到路由器节点 Objid
 IP_rte_id = op_topo_parent (self_id);
//得到名称为"xmt0"有线发信机的 Objid
xmt_id = op_id_from_name(IP_rte_id,OPC_OBJTYPE_PTTX, "xmt0");
//得到与"xmt0"相连的有线收信机的 Objid
remote_rcv_id = op_topo_assoc
              (xmt_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_RECV, 0);
```

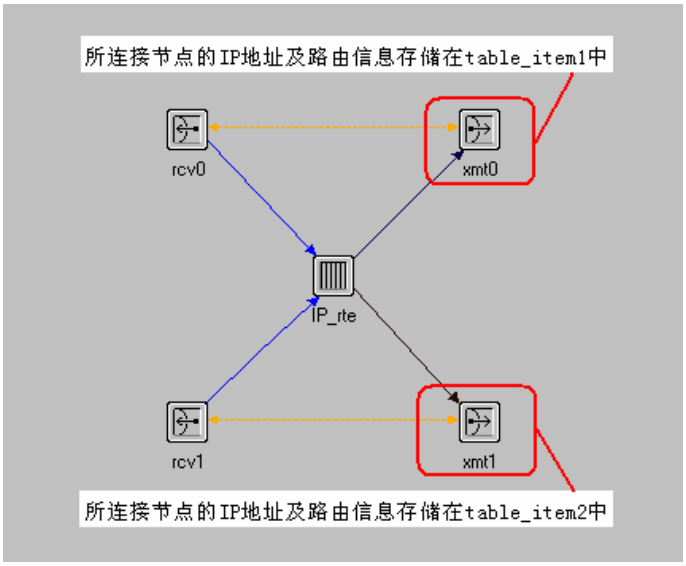


图 13-3　简单的路由器节点模型

有益提示 该收信机隶属与另一个节点，它与"xmt0"通过有线链路连接。

```
//得到与"xmt0"相连节点的 Objid
remote_node_id = op_topo_parent(remote_rcv_id);
```

接下来就可以根据这个 remote_node_id 得到其 IP 地址信息，我们假设节点 IP 信息已经存在节点的"Addr_IP"属性中。

```
//得到节点的 IP 地址
op_ima_obj_attr_get(remote_node_id, "Addr_IP" , addr_ip);
//得到节点的名称
op_ima_obj_attr_get(remote_node_id,"name",node_name);
//将字符串型的 IP 地址转换成长整型以便存储
addr=prg_ip_address_string_to_value(addr_ip);
```

小技巧　加入以下代码后，在 ODB 调试窗口中输入命令 ltrace ip_discover 可以打印路由器查找到与之相连节点的名称及 IP 地址，这样能够查看路由发现是否成功。

```
if (op_prg_odb_ltrace_active("ip_discover")==OPC_TRUE)
        {
        printf("IP address of Node %s is %s\n",name,addr_ip);
        }
```

得到 IP 地址前缀

```
//将 addr 以二进制的方式向右偏移 8 位，取得 IP 地址前缀，它由 IP 地址前面三个字段组成
addr=(addr)>>8;
```

最后将 IP 地址前缀以及与"xmt0"相对应的流索引号写入链表元素 table_item1 中

```
table_item1->netprefix = addr;
table_item1->MAC_addr = XMT0_STRM_INDEX;
```

有益提示　将 IP 地址信息与流索引号绑定可以看作建立了 IP 地址与物理地址的映射。

采用以上方法可以查找与"xmt1"相连节点的 IP 地址信息，并将其写入链表元素 table_item2 中，至此，整个路由表就建立完成了。

这里为了将路由表项的建立过程表现得更加直观，这部分代码没有采用子程序的方式编写。感兴趣的读者实际使用时可以将该部分代码独立为一个子程序。

13.4　路由表的查找

假设路由表已经按照 13.3 节所示建立完毕，当路由器收到 IP 包时，必须根据其目的 IP 地址前缀查找路由表，从而将其路由至相应的子网。下面我们来分析这个过程的代码，这些程序放在包到达状态中使用。

```
Packet* pkptr;
unsigned long int dest_ip_addr;
int MAC_addr;

//得到 IP 包
pkptr = op_pk_get ( op_intrpt_strm() );
//获取 IP 包的目的地址
```

```
op_pk_nfd_get (pkptr,"Destination address",&dest_ip_addr);

/*for debug*/
addr_ip = str;
prg_ip_address_value_to_string(dst_ip_addr,addr_ip);
//得到 IP 地址前缀
net_prefix = ( dest_ip_addr ) >> 8 ;

//假设路由表的大小 RTE_TABLE_SIZE，搜索路由表找到与 IP 地址前缀对应的物理地址
for ( i = 0 ; i <= RTE_TABLE_SIZE - 1 ; i++ )
{
  if ( route_table_ptr->netprefix == net_prefix )
     {
    MAC_addr = route_table_ptr->MAC_addr ;
     break ;
     }
     route_table_ptr = route_table_ptr->nextunit ;
}
//将 IP 包发送至下一跳节点
 op_pk_send (pkptr, MAC_addr) ;
```

第 14 章　图形化建模和文本方式建模 EMA

OPNET Modeler 的一大优点是提供了方便的图形化建模，但是有时候图形化建模方式并不能满足我们的需求，比如需要刻画几千个节点，或者需要刻画出很精密的调制曲线或天线模型。针对这个需求就提出了外部模块访问（EMA，External Model Access）的概念，它采用文本方式建模，可以用循环语句来刻画多个特定规格的节点。在建模中，图形化建模和文本方式建模各有优缺点，应该灵活使用。

对于 OPNET 大多数的编辑器，如网络、进程、调制曲线、天线等编辑器都具有文本建模的功能，EMA 就是用类似 C 语言的方式来描述模型。

14.1　EMA 配置网络模型

首先在图形界面下做一个简单的网络模型（可以只包含一个网络对象，例如一个固定节点或一个移动节点），选择 Topology→Export Topology→To EMA 就可以生成 EMA 文件（*.em.c），仔细观察这个文件，就可以发现文件中包含设置节点属性的函数，通常是 ema_obj_attr_set()等函数，手动把这个函数放置到一个 for 循环语句里，再把函数设置的属性值（如坐标、名称）改为变量，这样就可以通过循环设置大量的节点。

举例来说，要在一个网络模型中仿真一千个节点，如果把一千个节点一个个放进去，似乎不大合理，用 EMA 可以解决这个问题。一般操作流程是在图形方式下做出一个框架，生成 EMA 代码，然后再用 for 循环重复创建对象并设置对象的属性。

改完后需要在 OPNET 控制台（console）下进行编译，具体步骤如下：

（1）打开 OPNET 控制台窗口，进入修改过的*.em.c 所在目录

（2）输入命令 op_mkema -m 文件名<不加后缀>。

- 这时界面提示 Ema executable program <文件名.i0.em.x> produced.。
- 如果出现如图 14-1 所示的错误提示，需要启动 OPNET，并将存放*.em.c 文件的目录加入到 OPNET 模型目录中。

（3）执行刚创建的可执行文件<文件名.i0.em.x>。整个操作过程如图 14-2 所示。

- 执行以后将生成新的模型文件（*.m）
- 如果出现如图 14-3 所示的错误提示，启动 OPNET，在 Edit->Preferences 中找到 license_port 属性，并将其设置为 port_a

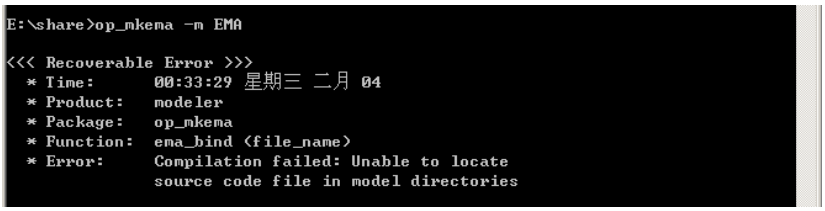
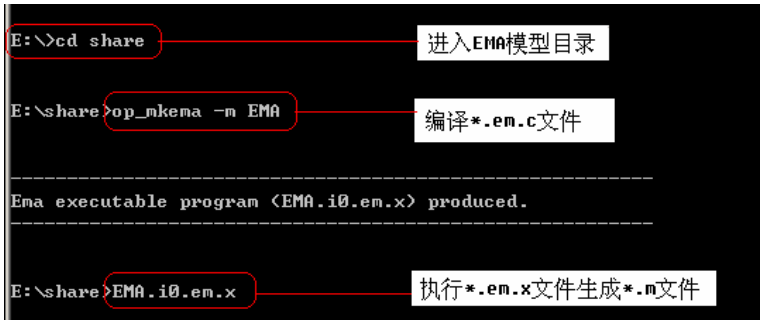


图 14-1　编译未放入 OPNET 模型目录的*.em.c 文件时出现的错误提示



14-2　生成模型文件的过程

图 14-3　未设置 license_port 属性时的错误提示

（4）在 OPNET 菜单中点击 File->Model Files->Refresh Model Directories 刷新模型目录，否则刚刚新创建的模型文件在已启动的 OPNET 中是不可见的。

- 对于生成的网络模型需要导入到项目场景中才能看到，在项目编辑器中选择 Scenarios→Scenario Components→import...

有益提示　没有 op 前缀开头的函数可以被 EMA、进程模型以及管道阶段程序使用，而以 op 前缀开始的函数不能被 EMA 调用。有些以 op_前缀开始的核心函数，去了前缀就可以在 ema 中直接使用，例如 op_prg_list_elem_find 与 prg_list_elem_find，它们除了函数命名上的区别外，功能完全相同，而其他的函数就不行，例如 op_dist_load()，如果需要随机设置节点的位置只能使用只能用 C 自带的随机值产生函数 rand()，否则编译会出错。

14.2　EMA 与外部数据的接口

EMA 除了用在如 14.1 节所述的精细扩展网络规模的场合，还有一个重要的用途是导

入外部测量的数据，这些数据一般以文本方式存储，可以是业务信息（可以包括业务的种类、生成规律以及加载在哪个节点上），节点信息（可以包括节点所采用模型名称、节点名称、节点图标及其位置），有线链路的连接关系（可以包括链路使用的模型名称和链路连接的两个端节点的名称）。不管何种文本信息，最终都是将相关数据设置在节点的属性中。

要将描述模型的复杂的纯文本信息通过 EMA 编程而转换成想要的模型文件是件非常细致的工作，这将涉及 EMA 的一些核心编程。下面我们列举一些值得注意的一些属性。

14.2.1　EMA 设置对象的固有属性

固有属性指的是 OPNET 为每个对象所固定配置的属性项，它们的设置相对要简单，因为导出的 EMA 程序中已经为它们创建了 EMA 对象 ID 号（EMA Objid），我们只要弄清这些对象的含义和它们之间的关系，剩下的工作就是修改它们的值了。表 14-1 中列出了各种网络对象常用的固有属性。

表 14-1　常用的固有属性及其用法

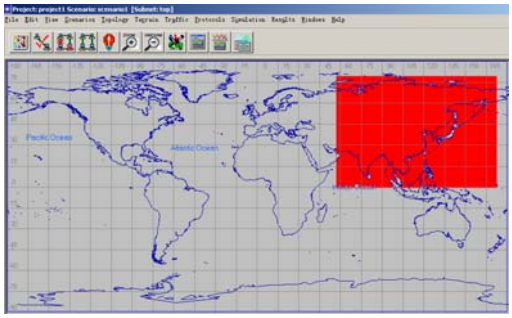
属性名称	隶属对象	值的类型	描　　述
View stack	子网	复合属性	子网的缩放视角
Map	子网	String	子网所用地图
X span	子网	度数或公里	子网的长度或经度的跨度
Y span	子网	度数或公里	子网的宽度或纬度的跨度
subnet	子网、节点、链路	EMA Objid	对象所属的子网Objid
name	子网、节点、链路	String	对象名称
model	节点、链路	String	对象模型名称
x position	节点	double	X轴坐标
y position	节点	double	Y轴坐标
icon name	子网、节点	string	对象图标名称
trajectory	节点	string	路径名称
Transmitter a	链路	String	节点a中收信机名称
Receiver a	链路	String	节点a中收信机名称
Transmitter b	链路	String	节点b中发信机名称
Receiver b	链路	String	节点b中发信机名称
Data rate	链路	double	有线链路速率
Packet formats	链路	String	链路传输支持的包格式
Site a	链路	String	链路连接的a端节点
Site b	链路	String	链路连接的b端节点

续表

属性名称	隶属对象	值的类型	描　　述
Line style	链路	String	表示链路线条的风格
thickness	链路	Int	表示链路线条的粗细
path	链路	复合属性	包含两个子对象 pos_start和pos_end
X和y	pos_start、pos_end	Double	链路两个端点的坐标

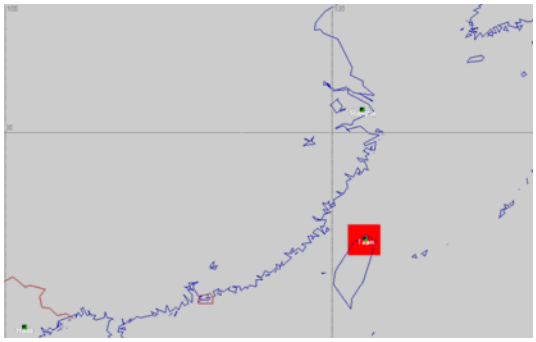
接下来我们对其中一些属性进行说明：

子网的 View stack 属性可能包含多个值，它们记录场景缩放历史，例如当我们创建一个 world 级场景时，点击放大镜按钮，选中东亚区域，如图 14-4（a）所示，这时场景布满选中的区域，如图 14-4（b）所示，再选中中国东南角，如图 14-4（c）所示，最后找到我们关心的区域，如图 14-4（d）所示。通过这样 4 级定位后，OPNET 自动将这 4 个缩放范围记录了下来，当我们将该场景导成 EMA 文件，就可以看到 view stack 属性有 4 个类型为 EMA Objid 的值，它们记录了每步缩放操作的具体属性。

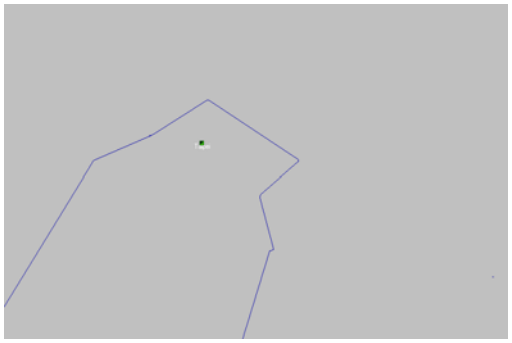


（a）world 级场景

（b）东亚区域



（c）中国东南角



（d）台湾地区

图 14-4　4 级缩放历史记录

子网的 map 属性所能设置的地图只局限于 OPNET 自带的地图，它们的名称可以在新

建一个场景过程中选择 Choose From Maps 看得到，如图 14-5 所示。遗憾的是不能设置外部导入的 BMP 或 TIFF 格式的图片作为背景地图。

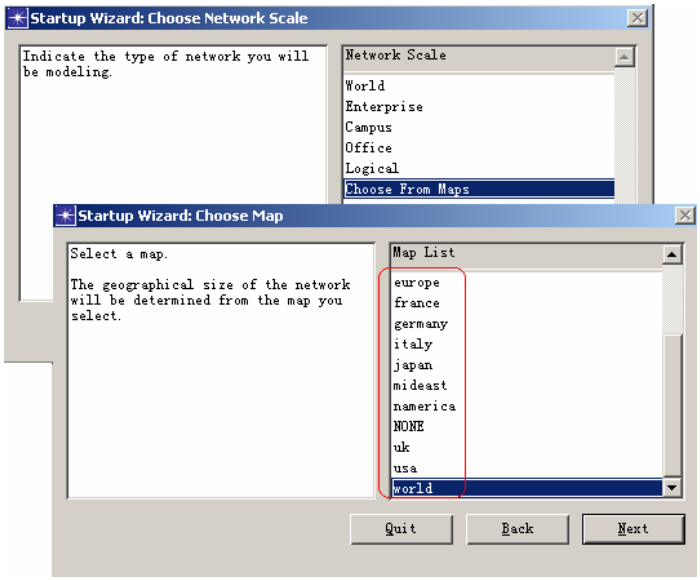


图 14-5　OPNET 自带地图选择框

子网的 x span 和 y span 确定了子网的大小，这两个属性的取值单位常用的是经纬度和公里，具体采取哪种，可以通过点击 View->Set View Properties 打开场景视觉效果对话框查看，如图 14-6 所示。另外最高级网络 top 全球网一定是以经纬度来衡量的。

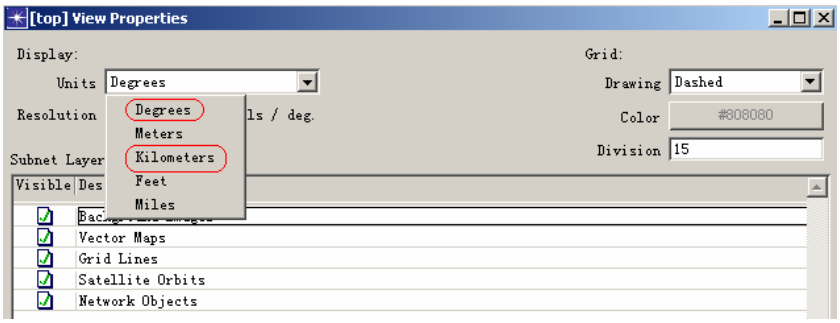


图 14-6　场景视觉效果属性对话框

设置节点的 icon name 属性往往能收到良好的视觉效果，我们可以自己绘制图标，然后将这些图标通过 EMA 加载。当我们创建好新的 icon 文件后，不要忘了在 Edit->Preferences 下查找 icon_dbs 属性，添加自定义图标库文件的名称，否则 EMA 对自定义的图标名称视而不见。

链路的 site a 和 site b 属性确定了链路的两个端节点；Transmitter a、receiver a、transmitter b 和 receiver b 属性描述了有线双工链路连接的两对收发信机，其值的格式为“节点名称.收发信机名称”如 node_name.xmt 或 node_name.rcv；pos_start 和 pos_end 设置链路两个端

点的位置，这 8 个属性共同确定一条链路。实际上两个端节点已经包含了链路的端点位置信息，pos_start 和 pos_end 没有多大意义，一般分别设置为两个端节点的坐标值。

14.2.2 EMA 设置对象的自定义属性

除了 14.2.1 节所属的固有属性，我们还可以设置自定义属性，这些属性主要作为将自定义外部数据传给进程模型的中介。下面我们列举导入外部数据并写入自定义属性的操作步骤：

（1） 读取数据文件，这里给出一个样本程序：

```
double double_data;
string node_name;
printf ("开始读入%s 文件.\n", FILE);
fp = fopen(FILE, "r");
if (fp == NULL)  printf ("打不开文件%s.\n", FILE);
while (fscanf (fp, "%f;%s", &double_data,node_name) != EOF)
    {
     //找到 node_name 对应的节点 Objid
        //根据这个 Objid 给节点设置属性值 double_data
    }
    fclose(fp);
    printf ("成功读入文件%s.\n", FILE);
```

（2）定义所需变量

- EmaT_Model_Id model_id;
  这时整个 EMA 模型的 ID 号，导出的 EMA 文件已经自动定义好
- EmaT_Object_Id node_objid, self_attr
  定义需要设置属性的节点 Objid（node_objid）和自定义属性对象的 Objid（self_attr）

（3）Self_attr = Ema_Object_Create (model_id, OBJ_ATTR_PROPS);

- 创建自定义属性对象

（4）Ema_Object_Attr_Set (model_id, self_attr,

```
        "units",              COMP_CONTENTS, "",
        "default value",       COMP_CONTENTS_TYPE, EMAC_DOUBLE,
        "default value",       COMP_CONTENTS, (double) 0,
        "high limit",          COMP_CONTENTS, (double) 0,
        "low limit",           COMP_CONTENTS, (double) 0,
        "symbol map list",      COMP_INTENDED, EMAC_DISABLED,
        "flags",              COMP_CONTENTS, 0,
```

```
"data type",          COMP_CONTENTS, 1,
"count properties",    COMP_INTENDED, EMAC_DISABLED,
"list attribute definitions",COMP_INTENDED, EMAC_DISABLED,
EMAC_EOL);
```

- 自定义属性本身也是一个对象，它同样有属性值，以上语句设置自定义属性对象 self_attr 的属性

（5）Ema_Object_Prom_Attr_Set (model_id, node_objid, "attr name", self_attr);

- 绑定自定义属性的名称（attr name）及其 Objid（self_attr）
- 最终在节点属性界面上看到的是自定义属性的名称

（6）Ema_Object_Attr_Set (model_id, node_objid, "attr name",COMP_PROMOTE, EMAC_DISABLED, "attr name",COMP_CONTENTS, (double) double_data, EMAC_EOL);

- 将从步骤（1）读到的数据 double_data 设置为自定义属性的值。

小技巧 使用 OPNET 导出的 EMA 文件为模型中每一个对象都分配了一个 EMA 对象 ID 号，如 EmaT_Object_Id　obj [721]，它包含了 721 个对象。这似乎表明 EMA 需要为每一对象分配一个唯一的 EMA Objid，这样给编程造成一些麻烦，因为要确保循环语句中操作的 EMA Objid 不重复多少要动些脑筋。而实际上 EMA Objid 是可以重复使用，并不需要保持其唯一性，我们只要定义几个 EMA Objid 就行了，往往一个定义代表一类对象。

小技巧 EMA 还可以用来导出矢量文件。标量收集方式的一般用法是，针对标量统计量，可以将一系列的仿真结果收集到一个文件中；而特殊用法可以针对矢量统计量采用标量收集方式，可以把一次仿真所有的矢量统计量结果数据集成到一个文件中，但是这样会导致该结果文件不能被 OPNET 直接显示为图形，因为 OPNET 矢量结果显示只能是一个结果图对应一个矢量统计量，因此需要将其中不同的矢量统计量数据分开，并存为不同的文件。这个工作非常繁重，似乎只能求助于 EMA。通过一个 EMA 程序能够做到将所有数据按统计量类别自动导出，一次就可以把矢量文件中的所有的数据全部转换成文本形式。目前已经有学者解决了这个问题，只要设置合适的输入参数就能达到以上目的，这个 EMA 程序为 ov_export.em.c，可在 OPNET 官方网站（www.opnet.com）上 contribute models 页面下载。

有益提示 EMA 虽然功能强大，但是对于有些情况却是无能为力，这里列举两种情况：通过 View->Background->Add Image…，可以在场景中导入 BMP 地图，如图 14-7 所示，想象一下在 EMA 程序中可能在设置地图属性相关语句中，有一个设置 BMP 文件路径的地方，但是却没有这种设置。

还有一种情况，一个场景原来的设置为 network scale：enterprise；size：1500*1000km。现在想将 size 属性改为 5000*2500km，本来在 EMA 中是很轻松的

事情，只要修改两个值就行了，但是 EMA 中却没有这样的属性，只好再新建一个场景，重新设置好需要的大小，然后从原来的场景中把东西全都拷贝到新的场景中去。

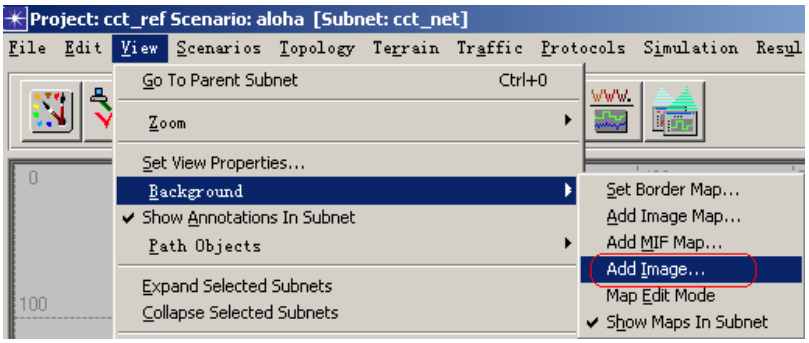


图 14-7　设置背景 BMP 地图

第 15 章　高级体系架构（HLA）

高级体系架构模块（HLA，High-Level Architecture）已经发展成为一个支持分布式建模与仿真（M&S）的通用体系结构。为支持 HLA 仿真系统的开发，建立了联邦开发与执行过程（FEDEP，Federation Development and Execution Process）模型。联邦测试过程是该模型中的一个重要环节，通常分为两步：联邦成员测试和联邦测试。联邦成员测试又称一致性测试，是联邦测试过程的基础和重要组成部分，以检测联邦成员是否符合联邦规则、接口规范和对象模型样板规范。

采用以 HLA 为核心的技术框架，以基础资源库为中心，建立能支撑仿真应用系统的开发、运行、演示、管理、控制、评估和研讨的仿真环境，在此通用仿真环境的支撑下，通过重用来实现“快、好、省”地开发各种仿真应用系统，即联邦，它的目的是为形成更大仿真而一起运行的一组仿真及有关应用系统（图形显示、数据记录等）。以下列举了有关 HLA 的一些重要概念：

HLA 是一个灵活可重用的软件体系结构，是创建基于组件的分布式仿真的框架性技术的结果。

成员：参与联邦的所有仿真应用，如仿真平台、聚合模型、图形显示等。

联邦执行：联邦的一次运行。

对象：联邦所仿真的领域中的一个实体，它为多个成员的关联。

交互：由一成员生成的非持续、有时间标注的事件，并经 RTI 由其他成员接收。

属性：在联邦对象模型中定义的有名数据，它与一对象类的每一实例相连接。

参数：与交互类每一实例相连接的有名数据。

互操作：联邦成员能向其他成员提供服务，或能接收其他成员提供的服务的特性。

OPNET 中高级体系架构模块支持多个仿真器的联合创建和运行，每个仿真器模拟一个复杂系统的某个方面。OPNET 的 HLA 模块使得 OPNET 的仿真 HLA 联合模型通信方面的部分或者全部。OPNET-HLA 接口为多个仿真器提供了必要的机制来共享公共的对象，交互消息以及维护时间同步。该模块支持很多 HLA 接口，包括：时间管理使得 OPNET 仿真可以和整个 HLA 时间同步；使用 OPNET 包来生成和接收运行时间支撑结构（RTI, Runtime Infrastructure）交互信息；在仿真中创建，销毁和发现 RTI 对象；收到 RTI 对象属性更新后自动更新 OPNET 属性；用户自定义 RTI 和 OPNET 对象之间的映射关系；特定的属性集。

在联邦执行中，通过 OPNET-HLA 接口，OPNET 将成为联邦中的一个成员。联邦成员之间的信息交换是通过对象和交互的形式进行的，如图 15-1 所示。

联邦执行细节（FED，Federation Execution Detail）定义了联邦执行过程中经由 RTI 通信的数据，即对象类和交互类，以及它们的属性和参数。我们必须为联邦设计联邦对象模型（FOM，Federation Object Model），为每个联邦成员设计仿真对象模型（SOM， Simulation Object Model）。

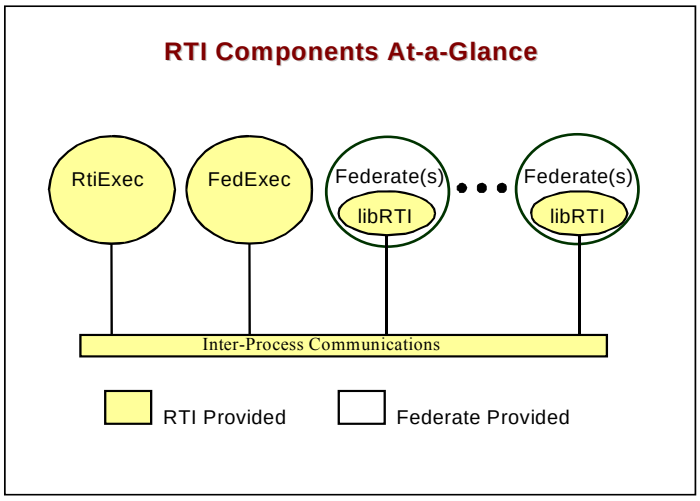


图 15-1　联邦成员与 RTI 的关系

HLA 中，各联邦成员及 RTI 共同组成的一个有名的集合称为联邦。联邦成员可以是仿真应用，各类辅助工具（数据纪录器，显示器，管理工具等等）或任何其它软件应用。联邦成员通过“联邦成员接口规范”（*.fed 文件指定）和 RTI 接口，调用 RTI 提供的服务。各联邦成员遵循共同的 FOM，FOM 符合对象模型模板（OMT，Object Model Template）规范。FOM 处于联邦互操作的核心。因此成员软件框架表示对象模型的正确和直观是极为重要的。另外，框架有效性的评估也部分依赖于它的应用范围，成员软件框架必须能适应并表示任意的 FOM，即支持成员开发的框架必须独立于任意的特定对象模型。如图 15-1 所

示。

HLA 联邦开发和运行过程最基本的步骤如下：联邦发起人和联邦开发小组必须确定联邦开发的一系列的目标并记录为达到这些目标必须完成哪些工作，表示联邦所要表达的真实世界的兴趣范围（实体和任务），并基于一系列所需的对象和交互对它们进行描述，决定联邦参与者（如果没有事先指定的话）；开发 FOM 以明确地记录联邦中各联邦成员之间信息交换的需求和各自的责任完成所有必需的“联邦实现”活动，并对联邦进行测试以保证互操作要求得到满足运行联邦，分析输出结果并把结果反馈给联邦发起人 。

15.1 RTI 的安装及其环境变量的设置

作为 RTI 底层支撑环境的 RTI 软件有几种，目前较新的是瑞典 Pitch 公司的 pRTI，而 OPNET HLA 运行要求使用的 RTI 软件为 HLA RTI 1.3NG release 2 以上的版本，它们可以从 DMSO 官方网站 http://hla.dmso.mil 上下载。

（1）本节的实例将基于 RTI 1.3NG release 4，它安装时的界面如图 15-2 所示，操作很简单，只是将压缩包的内容解出来。

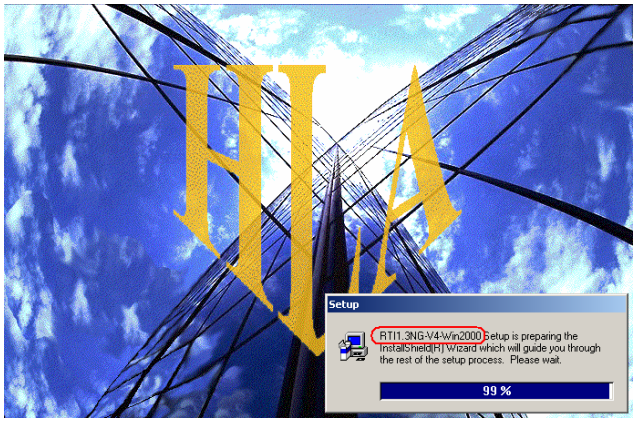


图 15-2 RTI 1.3NG release 4 安装时的界面

（2）在桌面“我的电脑”图标上单击鼠标右击，从弹出的菜单中选择“属性”。这时出现系统特性对话框，选择“高级”选项卡，然后单击“环境变量”。

接下来需要设置 RTI 的环境变量，它包括两个部分，分别是用户变量和系统变量。

（3）编辑三个户变量（include、lib、path），依次添加如下的变量值：

C:\Program Files\DMSO\RTI1.3NG-V4\Win2000-VC6\include

C:\Program Files\DMSO\RTI1.3NG-V4\Win2000-VC6\lib

C:\Program Files\DMSO\RTI1.3NG-V4\Win2000-VC6\bin

（4）如图 15-3 所示设置两个系统变量：RTI_BUILD_TYPE 和 RTI_HOME

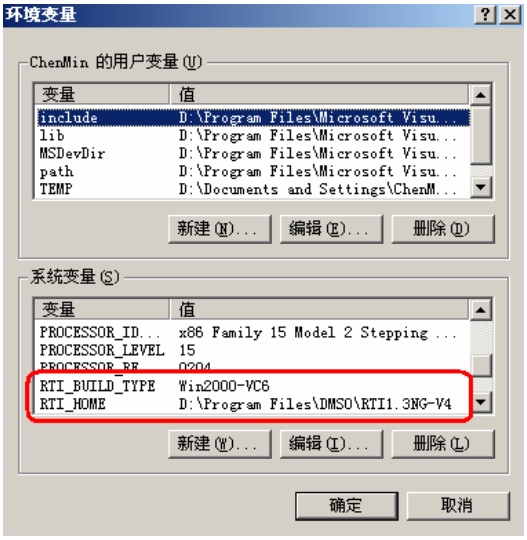


图 15-3　设置 RTI 所需的环境变量

如果 RTI 所在程序目录为 C:\Program Files\DMSO\RTI1.3NG-V4\Win2000-VC6 则 RTI_HOME 值为目录的前段 C:\Program Files\DMSO\RTI1.3NG-V4 而 RTI_BUILD_TYPE 为后段 Win2000-VC6

（5）环境变量设置完毕要重新启动机器才生效。

15.2　建立控制联邦成员

控制联邦成员（controller_tutorial）的代码是公开的，它只和 RTI 关联，和 OPNET 本身没有任何关系，是从 HLA 基类继承来的 C++结构，使用时还必须将其编译成可执行文件，下面我们来建立这个可执行文件。

（1）在<opnet_dir>\<version_num>\models\std\tutorial_req\hla 目录下找到以下四个文件（<opnet_dir>表示 OPNET 安装目录，<version_num>表示版本号）：

controller_tutorial.cpp	controller_tutorial.hpp	s_hla_stream.cpp	s_hla_stream.hpp

（2）将它们拷贝到一个新目录下，如 E:\controller_tutorial

（3）在 Windows2000 操作系统下用 VC 打开 controller_tutorial.cpp，然后按快捷键 F7。这时会弹出如图 15-4 所示的对话框，点击确定按钮，会自动生成一个工程空间。

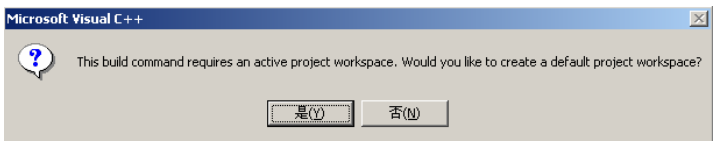


图 15-4　自动生成工程空间提示对话框

（4）编译时会出现错误 fatal error C1083:

Cannot open include file: 'strstream.h': No such file or directory

解决方法：将 s_hla_stream.hpp 中的#include <strstream.h>改为#include <strstrea.h>，存盘。

（4）再次编译出现错误 error C2632: 'long' followed by 'long' is illegal

解决方法：将 s_hla_stream.hpp 中的 long long 全部替换为 long int，存盘，再次编译

（5）这时编译通过，但是绑定目标文件时出错，提示出现不可识别的外部符号

controller_tutorial.obj : error LNK2001: unresolved external symbol

解决方法如下：

在工作空间管理窗口中点击 FileView 选项卡

在 controller_tutorial_files 上点击鼠标右键，在弹出的对话框中选择 Add Files to Project…，如图 15-5 所示。

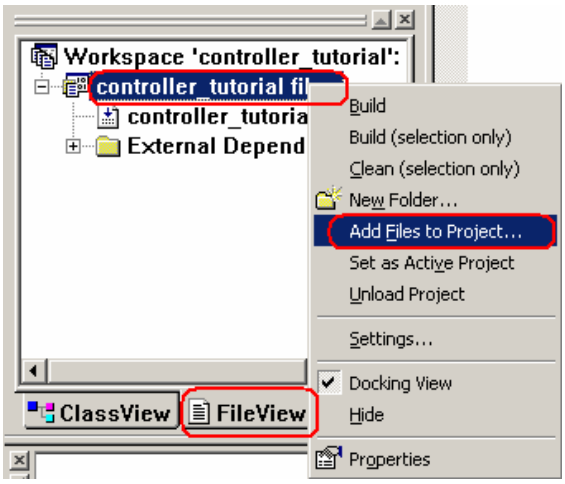


图 15-5　工作空间管理窗口

加入 C:\Program Files\DMSO\RTI1.3NG-V4\Win2000-VC6\lib 目录下的两个库文件：
libFedTime.lib　libRTI-NG.lib

加入<opnet_dir>\<version_num>\sys\pc_intel_win32\lib 目录下的库文件 ophla.lib（8.1.A 以上的版本没有该文件，8.0.C 版本有）

加入 C:\Program Files\Microsoft Visual Studio\VC98\Lib 目录下的库文件 WS2_32.LIB

图 15-6 为以上 4 个文件添加完毕后的工作空间管理窗口

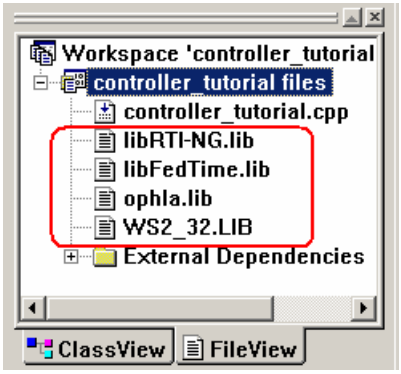


图 15-6　增加库文件后的工作空间管理窗口

（6）点击 Build 按钮，这时绑定通过，可执行文件已生成

从 E:\controller_tutorial\Debug 目录下找到已生成的 control_tutorial.exe 文件，拷贝至 OPNET 工作目录 E:\op_models 下。

15.3 OPNET HLA 仿真实例

通过 OPNET HLA 模块的支持，Modeler 可以在 HLA 联邦中作为其中一个联邦成员来仿真，而联邦中的其他成员影响或者受 Modeler 仿真事件的影响，它们同时运行，扩展成为更大的仿真系统，彼此提供实时的接口参数，使得整个仿真更加逼近现实并能使得每个独立仿真的准确性得以提高。

举例来说，当 Modeler 中的节点以另一个联邦所描述的方式移动时，Modeler 的仿真结果（如丢包率，响应时间等）则反映出节点受这种移动方式的影响，如果将这些结果实时地公布给其他联邦，就有可能影响它们的仿真行为。本节讲解的仿真实例将反映这一过程。

本实例包含两个联邦成员：一个是读者要创建的 Modeler 联邦成员（OPNET），另一个是已存控制台联邦成员（controller_tutorial)。

OPNET 中包含两个飞机节点（图 15-7 中分别为 fly0 和 fly1）和一个 HLA 接口节点（图 15-7 中为 HLA Interface）。飞机移动的方位和速度将受到 controller_tutotial 的控制。HLA 接口节点作为 RTI 与 OPNET 进程之间交换信息的通信中介。

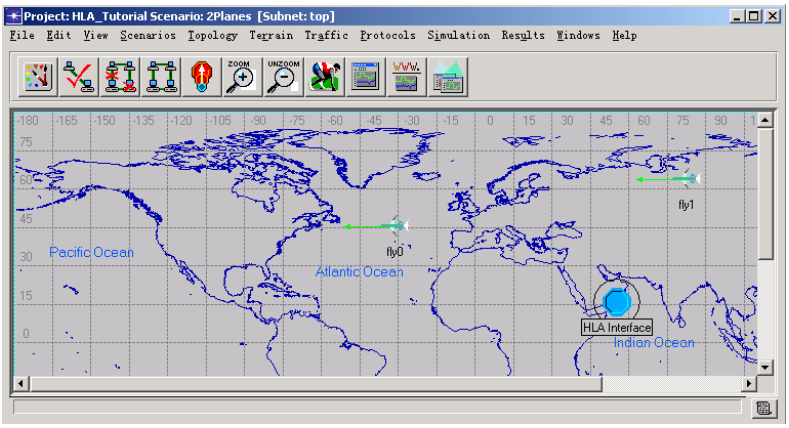


图 15-7 HLA 仿真场景

Controller_tutotial 主要有 3 个功能：

推进联邦的时间

向 OPNET 发送含有方位和速度信息的交互，控制 OPNET 中飞机节点的移动方式

接收来自 OPNET 的交互（包含飞机节点当前的位置信息），并显示交互信息。

15.3.1 准备所需的文件

OPNET 模型目录<opnet_dir>\<version_num>\models\tutorial_ref\hla 下已经包含了一套

可运行的 OPNET HLA 相关文件，如表 15-1 所示，将这些文件复制至工作目录 C:\op_models 下（或其它自创建的目录）。

表 15-1　OPNET HLA 仿真所需文件及其描述

文件名称	描　　述
HLA_Tutorial.prj	项目文件
HLA_Tutorial-2Planes.nt.m	网络模型文件
HLA_Tutorial-2Planes.pb.m	结果探针文件
HLA_Interface.nd.m	OPNET HLA接口节点模型
Sim_Hla_Interface.pr.m	OPNET HLA接口进程模型
Plane.nd.m	飞机节点模型
Plane.pr.m	飞机今进程模型
Plane_Order.pk.m	与Order交互类相映射的数据包格式文件
Plane_Ack.pk.m	与Oder_Reply交互类相映射的数据包格式文件
HLA_Tutorial.fed	HLA_Tutorial联邦执行细节文件
HLA_Tutorial.map	HLA_Tutorial交互类与OPNET接口的映射文件
HLA_Tutorial.ef	联邦环境设置文件
RTI.rid	RTI运行初始化信息文件

表 15-1 最重要的文件是 HLA_Tutorial.fed 和 HLA_Tutorial.map，它们分别定义了仿真所用到的交互类，以及交互类与 OPNET 数据包的映射。

有益提示　HLA 可以发送两种格式的信息，交互和对象，这里只涉及交互的定义。

HLA_Tutotirl.fed 文件中定义的两个交互类（Order 和 Order_Reply）：

```
;;  user interaction classes here
    (class Order reliable timestamp
       (parameter bearing)
    (parameter speed)
    )
    (class Order_Reply  reliable timestamp
       (parameter id )
       (parameter change_lat)
       (parameter change_long )
    )
```

对应 HLA_Tutotirl.map 文件中的交互类与包的映射：

```
 (interactions
      (class Order Plane_Order
         (parameter bearing bearing double)
         (parameter speed "ground speed" double)
      )
      (class Order_Reply Plane_Ack
       (parameter id id double)
       (parameter change_lat change_lat double)
       (parameter change_long change_long double)
      )
)
```

15.3.2 运行 HLA 仿真环境

（1）运行 C:\Program Files\DMSO\RTI1.3NG-V4\Win2000-VC6\bin 目录下的 RTI 的主支撑程序 rtiexec.exe 和控制台程序 rticonsole.exe

确保连接好网线，否则出现如图 15-8 所示的提示错误：

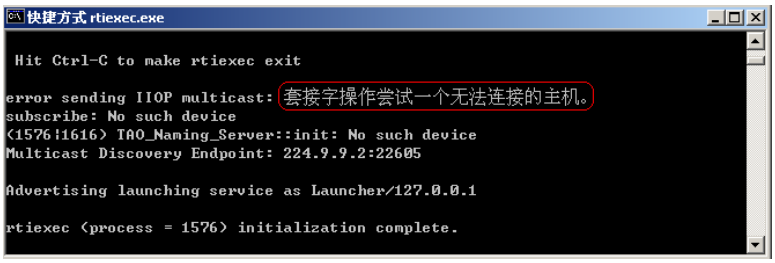


图 15-8 RTI 未成功启动

（2）运行控制联邦成员 controller_tutotiral 程序：

打开一个 DOS 命令提示符窗口，并切换到 controller_tutotiral.exe 与 HLA_Tutorial.fed 文件所在目录（C:\op_models）。

输入“controller_tutorial HLA_Tutorial”就可以启动联邦 HLA_Tutorial 和控制联邦成员 controller_tutorial。正常启动后应有如图 15-9 所示的信息。

（3）运行 OPNET 仿真

将 C:\op_models 下的 RTI.rid 拷贝一份至 OPNET bin 目录下：

<opnet_dir>\<version_num>\sys\pc_intel_win32\bin

启动 OPNET 软件，从 File 菜单中选择 Open…->Project->HLA_Tutorial

将仿真场景切换到 2Planes

将仿真属性设置为调试模式（Use ODB Debugger）

如表 15-2 所示设置仿真属性（HLA 环境变量），并运行仿真

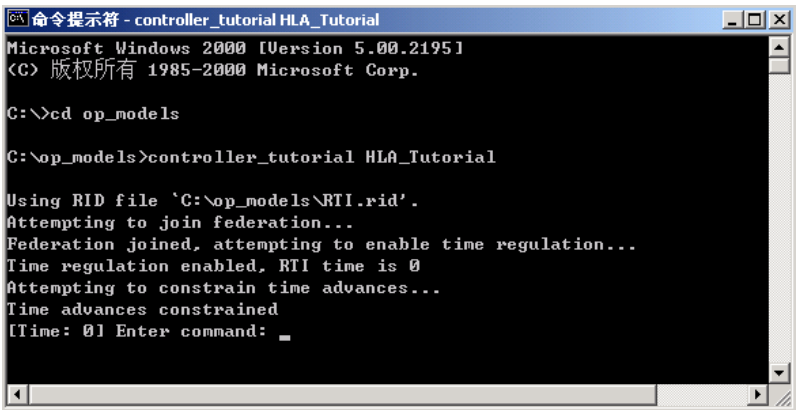


图 15-9　运行控制联邦成员

表 15-2　HLA 仿真属性及其描述

属性名称	默认属性值	描　　述
HLA Create Federation	enable	如果联邦未启动，OPNET是否自动创建联邦
HLA Destroy Federation	enable	如果OPNET退出联邦，是否销毁联邦
HLA Federation Name	HLA_Tutorial	OPNET所要加入联邦的名称
HLA Federate Name	OPNET	OPNET作为联邦成员的名称
HLA Time Lookahead	0.1	前瞻量
HLA Use Lowest Priority	disable	是否使用最低的事件优先级
HLA Update Check Interval	-1.0	查看对象属性值是否更新的时间间隔
HLA Class Mapping File	C:\op_models\ HLA_Tutorial.map	交互类映射文件的目录
HLA Abort On Failure	disable	当HLA接口发生错误时仿真是否退出
HLA Verbose	enable	

有益提示

HLA Time Lookahead 为 HLA 时间前瞻量，它是当每个联邦成员仿真到同一时间点后，单个成员被允许继续仿真的时间量

HLA Use Lowest Priority 用来控制某个时间点上 OPNET 事件的优先级，假设在时间 T 时联邦成员暂停仿真，而在 T 时刻 OPNET 仍有一些事件尚未执行，但是它们并不使时间推进，这时如果使用最低优先级则不执行，否则执行

HLA Update Check Interval 为复制代表 Sim_Hla_Interface 进程不需要周期性地检查对象属性是否更新，因为本例程只用到交互类传递信息

HLA Class Mapping File 属性非常重要，确保设置成 HLA_Tutorial.map 文件所在目录，如图 15-10 所示。

小技巧 除了在仿真属性设置对话框中设置属性，也可以直接在 HLA_Tutorial.ef 文件中设置。

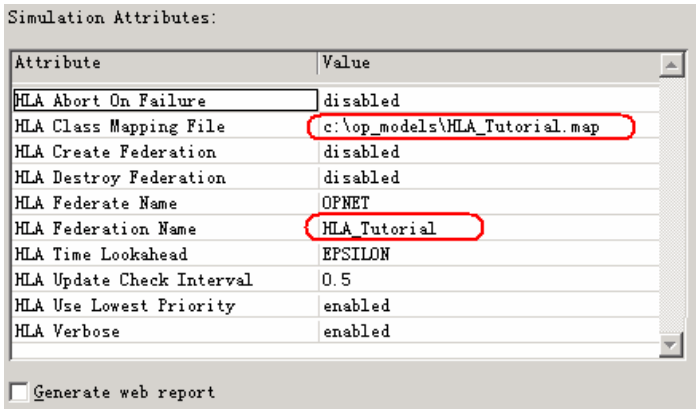


图 15-10 仿真属性的设置

15.3.3 实现 HLA 交互

按照 15.3.2 所示步骤操作后，将进入 HLA_Tutorial 项目场景 2Planes 仿真的 ODB 窗口，下面来观察 OPNET 是如何加入联邦，如何与 controller_tutorial 成员交互的。

（1）输入 next 命令，如图 15-11 所示

图 15-11 输入 next 命令

（2）再输入一次 next 命令，如图 15-12 所示

收到 beg sim 中断后，Sim_Hla_Interafce 进程将运行初始化代码，其中 special_kp_prim_hla_interface 是主要的 HLA 接口程序，它控制 OPNET 加入联邦 HLA_Tutorial，并输出提示信息，如图 xxx 所示。

在出现异常情况下 special_kp_prim_hla_interface 将启动 end sim intrpt 自动中断退出 OPNET 仿真，下面我们列举可能出现的错误：

错误 1：未将 RTI.rid 文件拷贝至 <opnet_dir>\<version_num>\sys\pc_intel_win32\bin 目录下，出现如图 15-13 所示的错误提示

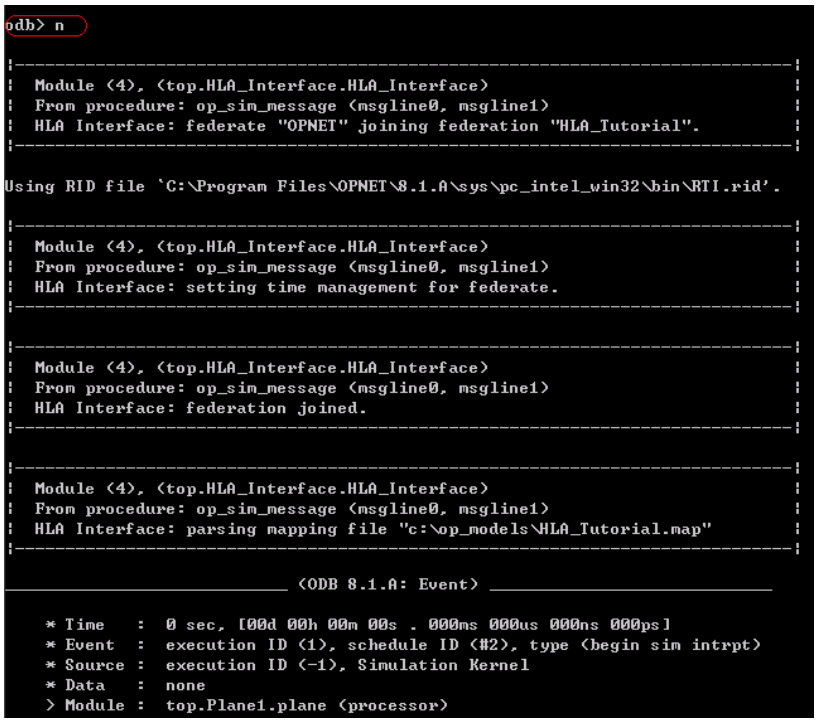


图 15-12　OPNET HLA 接口成功初始化时的信息

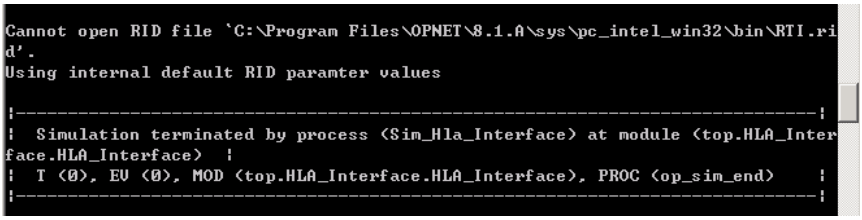


图 15-13　OPNET HLA 接口程序找不到 RTI 初始化信息文件

错误 2：fed 文件和 map 文件不匹配，map 文件所用到的交互类在 fed 文件中没有定义，提示信息为“federate resigning from federation”，这说明映射出错，需再次核实 HLA_Tutorial.fed 文件，看 Order 和 Order_Reply 交互类是否定义正确。同时在 RTI 控制台（rticonsole）窗口中看到 OPNET 这个联邦成员加入又离开 HLA_Tutorial 联邦，如图 15-14 所示。

错误 3：未创建所要定购的包格式，或者包域与 HLA_Tutorial.map 文件不匹配。这时首先查看仿真属性 HLA Class Mapping File，确保映射文件的目录和文件名正确，再仔细检查 map 文件内容，看看是否映射了 OPNET 包格式 Plane_Order 和 Plane_Ack。

关键概念　map 文件定义了数据包和交互的关联，一个交互类对应一种格式的数据包，而交互类的成员变量在包格式中变为相关的包域。

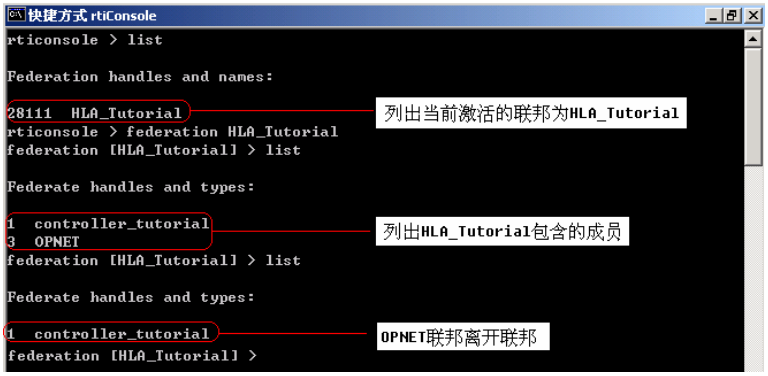


图 15-14　在 RTI 控制台窗口中查看联邦信息

（3）如果步骤（2）成功，则 OPNET 将加入 HLA_Tutorial 联邦，事件信息栏表明 HLA 接口进程模型初始化完毕。这时可以查看映射是否成功建立：

输入 odb> proldiag 0 hla_interactions，按回车键，将出现如图 15-15 所示信息，这些信息和你在 HLA_Tutorial.map 文件中定义的一致。

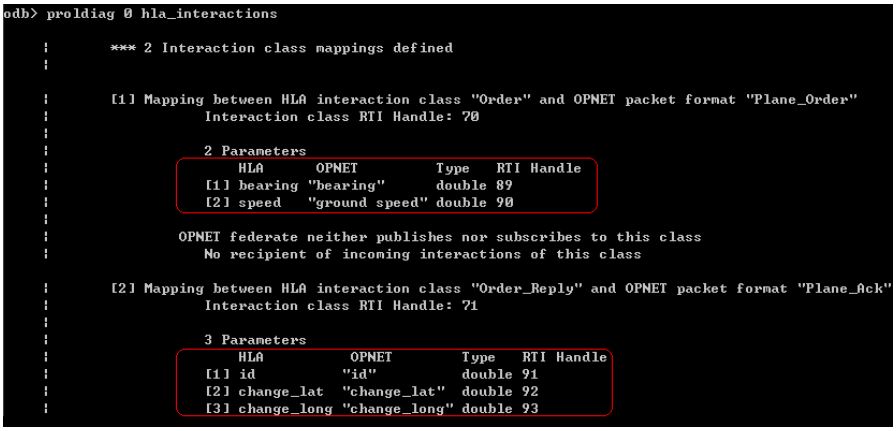


图 15-15　通过 proldiag 命令验证映射是否建立

输入 odb> proldiag 0 hla_pending 查看当前积压的订购交互类请求。开始这个请求列表为空。

（3）接下来即将对第一个 Plane 进程初始化，在 ODB 窗口中敲击 n 两次，直到出现一个 regular 中断，事件栏出现如图 15-16 所示信息：

图 15-16　Plane 节点收到 regular 中断

这时表明 plane 进程已完成初始化，主要执行以下语句

```
op_hla_interaction_pk_register (op_id_self (), "Plane_Order",
                                PLANE_ORDER_STREAM_INDEX);
```

这句代码订购流索引为 PLANE_ORDER_STREAM_INDEX 的数据包，数据包的格式为 Plane_Order，由于特定格式的数据包和某类交互相关联，实际上为当前节点订购了某类交互，当 controller_tutorial 联邦成员发出交互指令时，本节点将收到一个格式为 Plane_Order 的包，它包含了交互指令的信息。

（4）输入命令 odb> proldiag 0 hla_pending 查看积压的请求，如图 15-17 所示

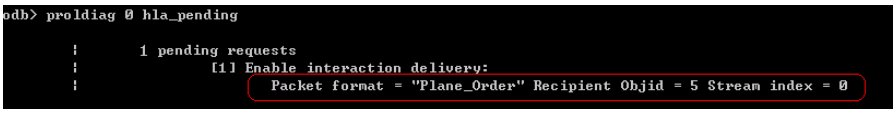


图 15-17　通过 proldiag 命令查看积压的订购交互类的请求

这时增加一个请求订购一类交互信息，通过格式为 Plane_Order 的包传递，其中 Recipient Objid，该交互信息的接收方 Objid，它是执行初始化程序对应节点 Objid，随着节点的不同，其值也不同，而 Stream index 始终为 0，因为在头文件中已经设定为固定值：#define PLANE_ORDER_STREAM_INDEX　0

（5）连续敲击两次 n，使另外一个 Plane 节点也完成初始化，

（6）再次输入命令 odb> proldiag 0 hla_pending 再次查看积压的请求，如图 15-18 所示

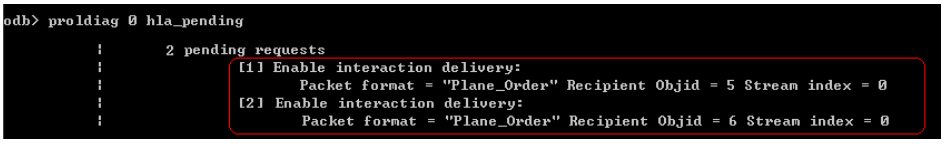


图 15-18　通过 proldiag 命令查看积压的订购交互类的请求

这时多出另一个 Plane 节点的订购交互类的请求

（7）再敲击两次 n，OPNET 仿真停滞，仿真核心不再发出事件，否则将推进 OPNET 仿真时间从而超过联邦时间。这里 OPNET 联邦成为时间受限联邦成员（time-constrained federate），而 controller_tutorial 联邦称为时间调度联邦成员（time-regulated federate），它控制着联邦时间，进而控制 OPNET 的仿真进度。这样 OPNET 像是受到 controller_tutorial 的调度而分段地仿真，controller_tutorial 也可以为 OPNET 某段仿真指定一些参数，这个行为也称为给 OPNET 联邦传递一个交互，结果是 OPNET 接收到含有该交互信息的数据包，从而根据数据包中的信息来调整以后的仿真行为。

有益提示　当 OPNET 联邦成员被再次激活时，OPNET 中的每个进程将收到一个 regular 型的中断（OPC_REGULAR_INTRPT），它由仿真核心周期性地产生，本例程隔 1 秒种产生一个，主要用来定期收集节点的状态，这些状态可以被记录下

来成为仿真结果数据或者动画。

（8）切换至运行 controller_tutorial 联邦成员的窗口，并输入命令 time 1

这将使得下一次 OPNET 在 1 秒钟时停止仿真

（9）切换至 ODB 窗口，可以观察到刚才的停滞状态被解除，时间被推进。

（10）再次输入 odb> proldiag 0 hla_interactions

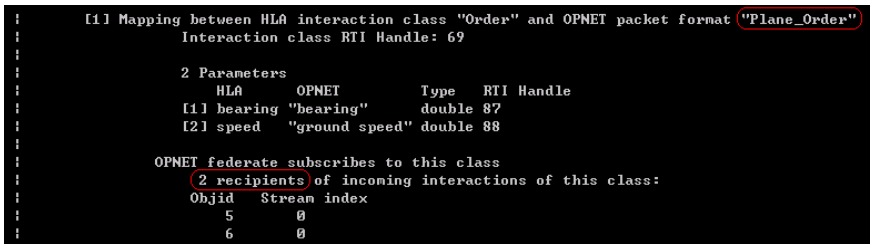


图 15-19　通过 proldiag 命令查看订购交互类节点的信息

由于 OPNET 联邦成员订购了 HLA 交互类 Order，OPNET 中的两个接收方（两个 fly 节点）将收到格式为 Plane_Order 的数据包，如图 15-19 所示。

（11）在 ODB 窗口中输入 c 让仿真继续。

在 OPNET 仿真时间 1 秒种时，仿真再次停滞

接下来我们要控制 OPNET 中 fly 节点的移动。

（12）在 controller_tutorial 联邦成员命令窗口中输入 fly 2 90 200

这将产生一个 Order 类型的交互类，这将使得 RTI 在第 2 秒的时候通知 OPNET 中的 HLA_Interface 节点生成一个格式为 Plane_Order 的数据包，并发给订购该类型数据包的节点（fly0 和 fly1）

（13）在 controller_tutorial 联邦成员命令窗口中输入 time 2

这将使联邦仿真时间推进至 2 秒

（14）切换至 ODB 窗口，可以观察到刚才的停滞状态被解除，时间被推进。

在 2 秒钟的时候 OPNET HLA 接口收到交互类信息，Sim_Hla_Interface 进程将创建数据包 Plane_Order，并将其发送给 fly0 和 fly1 节点，如图 15-20 示。

（15）2 秒种的时候 fly0 和 fly1 节点收到 HLA 节点发来的数据包后，执行如下程序段，我们对这些代码进行解释：

pk = op_pk_get (PLANE_ORDER_STREAM_INDEX);

从流端口 PLANE_ORDER_STREAM_INDEX 中获取 HLA 节点传来的数据包

```
op_pk_nfd_get (pk, "bearing", &bearing);
 op_pk_nfd_get (pk, "ground speed", &speed);
 op_pk_destroy (pk);
```

得到 bearing 和 speed 值

```
sprintf (gs, "%f kilometer/sec", speed);
 op_ima_obj_attr_set (parent_id, "bearing", bearing);
 op_ima_obj_attr_set (parent_id, "ground speed", gs);
```

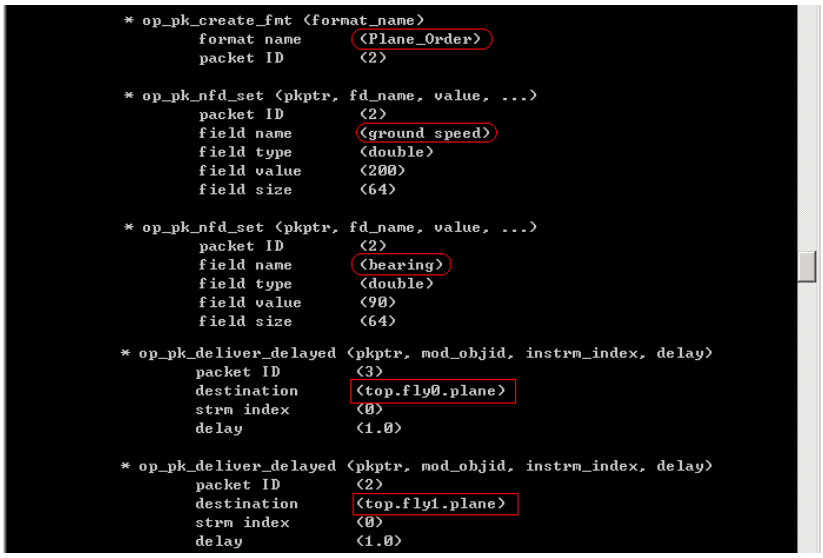


图 15-20　Sim_Hla_Interface 进程向 fly 节点发送交互包

设置 Fly 节点的 bearing 和 ground speed 属性：bearing = 90 度；ground speed = 200 公里/秒

op_ima_obj_pos_get (parent_id, &latitude, &longitude, &dummy, &dummy, &dummy, &dummy);

得到 Fly 节点当前坐标（受到方位角和相对地面移动速度的影响，坐标值也会不断变化）

```
pk = op_pk_create_fmt ("Plane_Ack");
    op_pk_nfd_set (pk, "id", parent_id);
    op_pk_nfd_set (pk, "change_lat", latitude);
    op_pk_nfd_set (pk, "change_long", longitude);
```

创建 Plane_Ack 格式的响应包，并且写入 Fly 节点 Objid，更新后的坐标值

op_hla_interaction_pk_send (pk, OPC_HLA_DELIVER_NOW);

立刻向 HLA_Interface 节点发送该响应包

有益提示　由于前瞻量设置为 0.1 秒，因此 fly0 和 fly1 节点有足够的时间（2.1 秒钟之前）将 Plane_Ack 交互包发送给 HLA_Interface 节点， OPNET 联邦成员将此信息发布给订购 Order_Reply 交互类的其他联邦成员（controller_tutorial）。

（16）在 controller_tutorial 联邦成员命令窗口中输入 time 3，并按回车键

这时 controller_tutorial 联邦成员收到 Plane_Ack 交互包含有的信息并在界面上显示出来，整个过程的命令及信息显示如下图 15-21 所示：

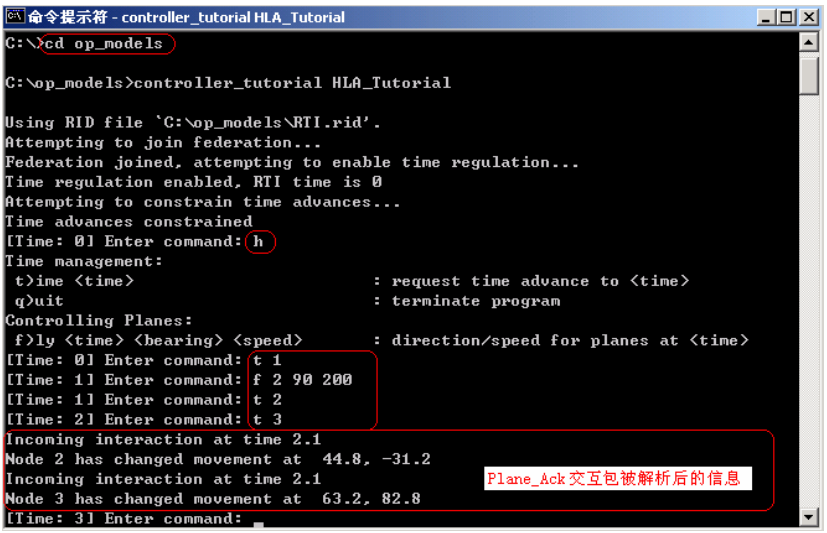


图 15-21　整个调试过程在 controller_tutorial 联邦成员命令窗口中输入的命令

15.3.4　多个 OPNET 联邦成员联机仿真

RTI 服务器之间通过 TCP/IP 互连，构成更大的仿真管理域。在单机联邦基础上，如果另一台机器再跑一个 OPNET，如果使用同一个 fed 文件和 map 文件，它将自动加入现有联邦。机器 A 运行 RTI 主支撑程序，时间调度联邦成员 controller_tutorial，和一个 OPNET 联邦成员，另一台机器 B 通过局域网（同一个工作组下）和 A 相连，运行另一个 OPNET 联邦成员。

在 RTI 控制台界面中输入命令 list，将列出当前创建的联邦，如图 15-22 所示：

图 15-22　在 RTI 控制台界面中查看建立的联邦

输入命令 federation HLA_Tutorial 查看联邦 HLA_Tutorial 的成员信息，如图 15-23 所示：

图 15-23　在 RTI 控制台界面上输入命令 federation HLA_Tutorial

再次输入 list 命令可以查看隶属于当前联邦的所有联邦成员，可以看到有两个 OPNET 联邦成员，如图 15-24 所示：

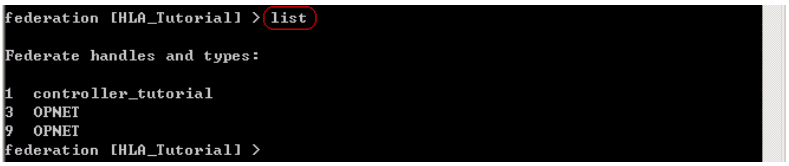


图 15-24　在 RTI 控制台界面中查看 HLA_Tutorial 联邦包含的成员

附录 A　本书中英文术语对照表

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

续表

英　　文	中　　文
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	离散事件驱动
dpt，default point ts point	点对点管道阶段文件的缺省前缀
dra，default radio	无线管段阶段文件的缺省前缀
Elapsed time	逝去时间
EMA，External Model Access	外部模型访问
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

续表

英　文	中　文
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，计算空间传播损耗模型的一种
MAC	信道接入控制层
model	模型
module	模块
Multi-tier	多端，指业务发送须经过多台服务器
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

续表

英　　文	中　　文
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，用来标识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	传播延时
queue module	队列模块，也可称为进程模块
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，结果收集模式的一种
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

续表

英　　文	中　　文
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 Vriables	状态变量
swap	交换，一般指队列中两个包的位置互换
TD,Tranmission Data	传输数据
TDA,Tranmission 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
transimission delay	传输延时
transimitter	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
Vector	矢量
Wireless domain	无线区域，用来划分接收主询
WLAN,Wireless Local Area Network	无线局域网

附录 B　参考文献

[1] Chen Min, Wei Gang．Multi-Stages Hybrid ARQ with Conditional Frame Skipping and Reference Frame Selecting Scheme for Real-Time Video Transport Over Wireless LAN．IEEE Transaction on Consumer Electronics, Vol.50, Issue 1, Feb. 2004.

[2] Chen Min, Wei Gang．Scheduling Algorithm for Real-time VBR Video Streams Using Weighted Switch Deficit Round Robin．28th IEEE Conference on LCN (Local Computer Networks), Bonn, Germany, Oct. 2003

[3] Chen Min, Wei Gang．A Novel Hybrid Algorithm for Real-Time Video Transport Over Wireless LAN．14th IEEE Conference on PIMRC (Personal, Indoor, Mobile Radio Communication), Bei Jing, China, Sep. 2003

[4] Chen Min, Zhu Xiaosong, Wei Gang．An IP DiffServ Framework for Real-time Video Transmission．ICCT (International Conference on Communication Technology), Bei Jing, China, April, 2003

[5] 陈敏、韦岗．一种适合实时变比特率视频传输的 IP 区分服务调度算法，计算机研究与发展，2004.5

[6] 陈敏、韦岗．IEEE 802.11 无线局域网 OPNET 建模与性能测试，计算机工程，2004.11

[7] 陈敏、张金文、韦岗．OPNET 无线信道建模，计算机工程与应用，2003.9

[8] 王文博、张金文．OPNET Modeler 与仿真建模，人民邮电出版社

[9] Brain P. Crow, Indra Widjaja, Jeong Geun Kim and Prescott T.Sakai．IEEE 802.11 wireless local area network．IEEE Communication magazine，September 1997

附录 C　本书中常见问题（摘自网络论坛中的帖子）

问　　题	解 决 方 案
安装了 10.0，计算机属性的环境变量设置按照陈敏书中的方法进行了设置，但与 VC++联合调试是不是还需要在 opnet 的 Edit->preference 中设置 bind_shobj_flags，comp_flags 和 comp_flags_cpp 中进行设置呢？我在书上是看到要设置的，可我 reference 目录下没有找到上面的三项，不知道该如何设置	（1）书中介绍的是 8.0 的设置，不适用于 10.0。 （2）VC 需要打补丁：sp5 （3）10.0 版本不需要在 Edit->preference 中设置，直接装上 OPNET 后就可以和 VC 联合调试的。注意记得启用 odb 模式
（1）书中 6.1 那个例子,我按书中做,但是最后无结果，为什么? （2）我在做书 P106 页 tutorials 里第二部分第三个例子 Packet Switching 1 仿真时出现： <<< Program Abort >>> Tda index (15) is out-of-range for packet (0) T (10), EV (17), MOD (top.pksw1.node_0.xmt), KP (op_td_get_int) 这样一段话，仿真结果是一小段直线，显然有错误，但是我是按照书上一步步做的，为什么做不出结果呢？	检查包流设置，在建立节点模型的时候，采用一拉一点的方式链接包流线的顺序不同，导致书中的代码（书上 p121 包流定义）所对应的包流顺序和实际的不同，需要核对。如果采用数在代码，需要确保在 node 节点模型中的建立包流的时一定要按照书上写的顺序(rcv->proc,proc->xmt,src->proc)建立。
书中一些笔误	（1）：正如楼上说的，我用的 10。0，所以不知道是不是因为你的版本问题。 （2）：p115 （12）中的代码 FOOT,应改为 FOUT. （3）:同页的（13）中省略了该进程模型保存的名字：initials_pksw_hub_proc（注：由于 10。0 中的命名不支持<>，所以我的命名把<>都舍去了）

	（4）：p120-123 周边节点的建立过程当中，没有提到 proc 进程保存的名字，我自己将其名定为 initials_pksw_node_proc，并且需要将 proc 进程的 process model 属性值改为 initials_pksw_node_proc （5）：需要注意节点间链路，进程间包流的建立顺序（p112,p117,p124）。由于我也是新手，所以具体原因我暂时也不是很清楚：） （6）：最后就是关于书中的一些图和文字说明不是很符合的情况了，例如 p117 图 6-17，p128 图 6-36 等，图中的有些属性或者实体的名称大概是由于某些原因和我们做的过程中的不太符合，不过都是些起名字的区别，应该可以识别出来，对整个的过程没有影响。
我在看应用层建模方面的我照陈敏书上 12 章自定义流媒体协议的实现 304-305 页 的步骤修改了头文件 gna_mgr.h 但是在接下来的步骤中打开 Application Definition 属性为什么找不到新配置的应用属性 streaming application" ? 里面的还是只有配置前的那九种应用属性比如 FTP HTTP DATABASE……	书中是基于 8.0.C PL 16 版本，版本不同，标准代码也不同，而 12 章涉及修改标准代码。
在 10.5 中的小区系统模型在 opnet 10.0 中小弟按书中说的模型导入，在 network model 中找不到 cellphone_net。谁能告诉我怎么找到这个模型阿。	这个模型 10.0 版本没有，在 8.0 版本中有
我在 opnet10.5 上建立一个 IP 网络模型，仿真故障情况下，流量在各路由器的丢失情况。我在每台路由器	QoS 仿真确实很慢，要想获得非常精确的方针结果，就要牺牲方针时间，也就是说精度和速度是矛盾的，可以

上都配置了 QoS，以便得到各端口的丢包率。但发现网络模型配置了 QoS 后，运行得非常非常慢，另外出来得丢包结果也不太准确。请教各位高手，如何才能加快 QoS 得仿真速度，或者有没有简单得方法获得准确的丢包信息。	用分析的方法在两者间寻求平衡，具体可以参看书业务建模那一章
在陈敏编著的 opnet 网络仿真书里面的 4.4.9 节中，在选择统计量时，我怎么怎么找不到 node packet count 选项呢？看过那本书的朋友们请帮帮我。如何把这个统计量添加上去	自己定义统计量需要好几步，一步也不能少 1.在 SV 里面定义统计量的全局变量和统计量句柄 2.注册统计量 r 3.写入统计量 4.在进程 interface 中的 local statistic 中还要定义此统计量的名称，收集方式等 这样在 choose results 中的节点中的 moduler 中才会出现统计量
出现 repositories 调用错误，系统要求你检查所有的 repositories 参数	清空那两个 bind_shob_flags 和 bind_static_flags 就可以了
本人在做 EDCA 的时候,简单地建立了四个无线节点,建立四个 process model 都是根据 10.5 版本里自带的而进行小段程序修改之后,重新命名的. 编译的时候没有问题,但仿真的时候,出现了下面的错误: <<< Recoverable Error >>> Object repository construction failed due to errors encountered by the binder program (bind_so_msvc) 进程 1.dev32.i0.pr.obj : error LNK2005: 某函数 already defined in 进程 2.dev32.i0.pr.obj	如果一个进程的函数块（FB）中定义了函数，该函数的名字在整个工程中应该是唯一的。否则，虽然单个编译都能通过，但是运行工程就会出现绑定错误。检查你的多个进程模型中是否有重名的函数，你必须将重名的函数名一一修改。或者将所有功能集成进一个进程。或者将 FB 中的函数做成 external_function 文件，然后再在进程中 declare。