

目录

第 1 章

OMNeT++仿真平台

1.1 OMNeT++简介

       OMNeT++，一个基于Eclipse开发套件的开源网络仿真工具，目前主要在高校实验室进行一些网络仿真测试，对一些算法进行对比，它可以供使用者进行完成以下开发：

C/C++开发

网络仿真程序设计

       毫无疑问，基于Eclipse的开发工具肯定能支持普通的C/C++工程。 另外，在OMNeT++上网络仿真设计领域的优势在于，它是一个开源的项目，对大量的网络模型都提供代码支持。但是问题在于国内的确没有什么社区支持，出现问题只能自己解决，其实对于开源的项目大多存在这种问题，往往开源的项目，使用起来难度较大，开源项目往往比那些商业的软件开发难度较大，支持也较少，开源可不代表简单。   OMNeT++对初学者能力要求高，它假定使用者对编程有一定了解的，对eclipse开发环境也是特别熟悉的，另外这是一个网络仿真的软件，需要你对计算机网络有足够的认识，它提供了大量现有各种网络的仿真例子，如果你对网络认识足够强，那么这个软件你用起来会感到特别顺手。   目前有大量的开源仿真库用于OMNeT++环境，拥有丰富的外文资料，官方将其分为两类，包括Supported Models和Contributed Models：

Supported Models 模型库的开发处于激活状态，有开发者在维护，定期会推出新的版本。

Contributed Models 完成后只推出过一次或几次版本，目前没有人在维护。

1.2 OMNeT++开源库

       下面简单介绍一下几种常见的开源库。

1.2.1 INET

       由Simucraft公司主持开发，用于仿真有线及无线网络。

应用层协议：HTTP、FTP、Telnet、不同优先级的 Video、Ping；

传输层协议：TCP、UDP、RTP（RealtimeTransport Protocol）；

网络层协议：IPv4、IPv6、ICMP、ARP、MPLS、LDP、RSVP、OSPF、Mobile IPV6、AODV、DSDV、DSR；

数据链路层协议：Ethernet、PPP、IEEE 802.11、FDDI、Token Ring；

官网：http://inet.omnetpp.org

1.2.2 INETMANET

       由 Simucraft 公司主持开发，用于仿真无线、有线网络，在INET 的基础上增加了大量的 MANET 协议，INETMANET= INET+MANET，在INET的基础上增加：

[1] 802.11a,g:Ieee80211aMac，Ieee80211gMac，Ieee80211aRadioModel，Ieee80211gRadioModel

Ieee80211Mesh,Ieee80211MeshMgmt

radiomodels：TwoRayModel，ShadowingModel，qamMode

Ns2MotionMobility

ARP:global ARP cache

AODV,DSDV，DSR，DYMO，OLSR

官网：http://inet.omnetpp.org

1.2.3 Mobility Framework

       由 Simucraft 公司主持开发,是一个无线传感器仿真模型库.绝大多数协议已经被 INET 吸收

官网：http://mobility-fw.sourceforge.net/hp/index.html

1.2.4 SensorSimulator

       美国路易斯安娜州立大学开发,用于仿真无线传感器网络

官网：http://csc.lsu.edu/sensor_web/

1.2.5 Castalia

       澳大利亚国家信息技术中心（NICTA）开发,是一个基于OMNeT++ 的侧重于无线网络的仿真器。基于实测数据的高级 channel/radio 模型，Radio 详细的状态转移，允许多传输功率电平。

高度灵活的 physical process model

感应设备的噪声、偏差（bias）和功耗

节点时钟漂移，CPU 功耗

资源监控，如超出功率限制（如 CPU 或内存）

拥有大量可调参数的mac协议

用于设计优化和扩展

官网：https://github.com/boulis/Castalia

1.2.6 OverSim

       德国卡尔斯鲁厄大学开发

用于仿真点对点（p-to-p）协议，如chard，GIA等

官网：http://www.oversim.org

1.2.7 TTE4INET

       由Communication over Real-Time Ethernet Group开发的时间触发以太网仿真模型，包括对AS6802的仿真实现。现仿真模型已改名为CoRE4INET。

开发组主页：https://core.informatik.haw-hamburg.de/

第 2 章

初入OMNeT++

2.1 OMNeT++下载

       OMNeT++可以直接从网上下载，网站地址是：https://www.omnetpp.org[1],但是国内直接从该网站下载，下载较慢，同时时常在安装下载过程中出现下载中断的情况，导致前功尽弃，下载成功较难。

2.2 OMNeT++安装

2.2.1 安装准备

       由于OMNeT++支持多个操作系统环境的安装，包括MacOS、linux和Windows，在这里只描述Windows环境下的安装。 软件的安装说明肯定在软件的安装文件有说明，我们没有必要每次安装一个软件的时候都去百度一下软件安装的过程，作者的观点是对于一些破解较难，安装复杂的软件安装可以写写blog，记录记录。我们可以在OMNeT++的安装包下发现readme文件和doc目录下的installguide，去看看吧，总会发现我们的安装执行步骤，掌握这种办法，断网了也能安装、无论过多久还能记得安装过程。好了，废话不多说了。下面是几个你在安装过程中可能会用到的命令：

./configure

       在PC机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

make

       在PC机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

make clean

       清除前面安装过程中产生的中间二进制文件，这个命令主要用于重新安装软件的过程中，如果遇到make出错的问题，可以选择这个命令清除到二进制文件，然后在使用make命令编译安装（因为有些时候下载的安装包不是原始文件）。

[1]

2.2.2 图文并茂

       其实这一部分没有说明必要，姑且就当作者无聊，还是想写写，我的原则就是坚持把故事讲得透彻明白，有些时候，在阅读别人博客的时候，老是会有很多疑问，其实博主以为读者懂，但读者的专业背景不一样，导致可能很简单的问题，还得下边留个言……好了，我们还是回到本节的话题上。   以下三张图：

图2　4　1 doc目录

       上图文件是OMNeT++团队提供给开发者的基本帮助文档，我在写这个文档的时候，自我觉得还没有把这些文档都翻开看一遍，查阅这些文档久了，就会慢慢觉得这些资料本身已经够用了……我会在后续的文档中，描述一下OMNeT++提供给我们的地图。

2.3 INET库

2.3.1 INET库的介绍

        从一个初学者的角度，当安装OMNeT++后，大多数的情况下是需要安装INET库的，这个集成库包含了丰富的仿真模型，多数时候，读者如果设计一个网络仿真程序，有不想重新编写代码，这时候，可以在INET下寻找是否有满足要求的example，包括的网络有：

adhoc

aodv

ethernet

ipv6

等等，上面列举出的只是其中经常用到的一小部分，但是这也存在读者的不同研究背景，可能其中涉及的还不算很全。目前，我对于INET的使用较浅薄，水平还停留在调用INET库中的ned文件中的节点类型，或者其他诸如移动模型的水平上。在该小节，作者先为读者描述一下如何在OMNeT++下快速的使用INET库和目前我经常使用的技巧。

2.3.2 INET库的安装

       通常有两种方法安装INET，在安装之前，首先需要到:https://inet.omnetpp.org[1]，下载合适的版本，由于前面的OMNeT++使用的5.2的版本，这里我们可以选择inet-3.6.2，下载结束以后，将inet解压到omnetpp的安装路径下的samples文件下，此时inet文件的路径可能是： xxx/omnetpp-5.2/samples/inet（解压INET-3.6.2文件后只有一个inet文件）。

    接下来，开始安装INET库，有以下两种方法安装：

方法一：命令窗口安装INET

       其实，如果需要为omnetpp安装新的插件或者库，都可以通过命令行的形式进行安装，甚至，你可以在命令行的环境下对编写好的网络进行编译

[1]

和运行。   我编写这个学习手册的原则，就是为读者提供一个学习OMNeT++的地图，而不是特别详细的字典，可能我的水平还远远没有达到写一本学习OMNeT++的大全。首先，安装这个INET库，我们到inet文件下看看有什么有用的文件没有，当然是先看看README.md了，这个文件提示我们安装请看：INSTALL，下面是这个INSTALL的英文：

```
If you are building from command line:
--------------------------------------
3. Change to the INET directory.

4. Type "make makefiles". This should generate the makefiles for you automatically.

5. Type "make" to build the inet executable (debug version). Use "make MODE=release"
   to build release version.

6. You can run specific examples by changing into the example's directory and executing "./
run"
```

当然，你可以选择mingwenv.cmd命令窗口，输入以上指令进行编译安装。上面的英文安装较为简洁，下面是我使用命令窗口安装的过程：

[1] 在安装INET库之前，应先确保OMNeT++已经安装成功。进入到OMNeT++安装路径，找到mingwenv.cmd文件，双击执行，进入下图：

[2] 接下来，使用命令：cd samples/inet，进入到samples下的inet，另可使用ls命令查看当前inet文件下各子文件。

[3] 然后，执行make makefiles命令生成编译整个inet库的makefile文件，结束以后输入命令make。到这来，使用命令窗口编译inet库就结束了。

[4] 最后需要在OMNeT++ IDE中Project Explore窗口空白处右击，如图，使用Import功能导入已经编译好的inet库。过程如下图：

方法二：OMNeT++窗口安装

       一样的，在INSTALL下命令行安装方式下面就是使用IDE的安装方式，这个IDE的使用方式就是将INET库使用OMNeT++打开，当然此时库文件inet已经在samples文件下，我们需要做的就是打开OMNeT++ IDE ，然后导入整个inet工程。

```
If you are using the IDE:
-------------------------
3. Open the OMNeT++ IDE and choose the workspace where you have extracted the inet directory.
   The extracted directory must be a subdirectory of the workspace dir.

4. Import the project using: File | Import | General | Existing projects into Workspace.
   Then select the workspace dir as the root directory, and be sure NOT to check the
   "Copy projects into workspace" box. Click Finish.
```

```
5. Open the project (if already not open) and wait until the indexer finishes.
   Now you can build the project by pressing CTRL-B (Project | Build all)

6. To run an example from the IDE open the example's directory in the Project Explorer view,
   find the corresponding omnetpp.ini file. Right click on it and select Run As / Simulation.
   This should create a Launch Configuration for this example.

If the build was successful, you may try running the demo simulations.
Change into examples/ and type "./rundemo".
```

       :根据上面的步骤，需要点击：File | Import | General | Existing projects into Workspace ，导入inet整个工程文件，对整个工程进行编译即可。

2.4　TTE4INET

       :TTE4INET仿真库由Communication over Real-Time Ethernet Group开发的用于仿真时间触发以太网依赖于INET的仿真库，目前已改名为CoRE4INET，新版本对AS6802支持尚存在版本问题，对于需要仿真AS6802中同步功能需安装旧版本TTE4INET，可在官网下载：

http://sim.core-rg.de/trac/wiki/CoRE4INET_Background

本节对该库安装进行简单说明。   :TTE4INET依赖于omnetpp-4.4.1和inet-2.1.0，旧版本的omnetpp存在许多编译或者依赖设置等各种问题，例如：每次修改代码后，需要重新编译再执行，否则会直接直接原可执行文件，不会对修改的源代码进行重新编译；链接过程不智能，需在CONFIGNAME（gcc-release或gcc-debug）中寻找依赖库，若工程为release，而依赖库使用debug，那么将会存在找不到依赖库的链接错误，这是目前在使用OMNeT++旧版本中发现的问题。

2.5　常规使用

2.5.1　导入工程

       :其实我觉得还是有必要把这一小节的内容加入其中，考虑了一下，这个软件的有些操作还是不太一样，可能初学者自己去找需要花大量的时间。   :在学习如何导入工程前，先观察一张图：

图2　4　1 IDE视图

       :图2　4　1中，左边窗口为**Project Explorer，在软件安装首次打开IDE时，**Project Explorer中已经默认有相关的samples目录下的工程，如果开发者想要打开已有的工程，需要在Project Explorer窗口空白处右击鼠标，进行Import | General | Existing Projects into Workspace，最后选择工程文件即可，不需要任何设置直接finish**即可。其中相关的中间过程如下：

图2　4　2 点击Import后视图

图2　4　3 点击Existing Projects into Workspace后视图

       前面已经描述了相关过程，需要注意的是保证工程文件**不要放在有中文名的路径**下，如果包括的中文路径，在后期编译工程时，可能在ned文件下出现大量错误，无法识别ned文件路径。

2.5.2 程序执行与调试

       导入了工程，如何执行程序和调试还是很重要的，尤其是对于OMNeT++工程，在omnetpp工程下有三种文件：**ned 、cpp和ini**，下面是这三种文件的介绍：

ned ： 网络拓扑描述文件、简单节点模型和复合节点模型；

cpp ： cpp文件为描述简单节点编程，定义简单节点各种行为；

ini ： ned文件中相关参数的配置，在ned文件中一般会设置诸如节点数量的变量，一般有默认值，但是为了修改方便，可以在ini文件里边直接配置修改；

msg ： 消息描述文件，会被opp_msgc转化成*_m.cc/h文件。

       为了更好的说明以上三种文件在一个工作里边的关系，下面展示一张图：

图2 4 4 编译与执行仿真流程

       这张图还是比较重要的，尤其是当你发现你的ned或者ini文件不启作用的时候，可以根据上面的仿真程序流程思考一下找到bug所在。再这张图中，可以看出Simulation program**就是我们工程生成的可执行文件**exe，也许你会发现在我们执行程序的时候有两种选择：

Local C/C++ Application

OMNeT++ Simulation

       这两种方式执行仿真程序有何不同了，结合**图2-4-4，选择第一种执行方式其实就是执行的Simulation program，但是这种执行方式运行的仿真程序没有加入ned**文件和ini配置文件，因此就是模型节点参数没有配置好或者就没有配置。第二种执行方式就比较完整了，其模型加入了ned文件和ini配置文件。其他类似问题读者可自行揣测。   前面介绍这么多，还是直接进入主题，我们执行OMNeT++工程大致有三种方式：

[1] 直接右击工程文件–>RUN as–>Local C/C++ Application 或者OMNeT++ Simulation；

[2] 选中描述网络的ned文件，右击执行上面一样的操作；

[3] 选择配置网络参数的omnetppp.ini文件，右击执行上面一样的操作。

       对于上面执行工程的后两种方式一般都是可以的，但是对于第一种方式来说，需要执行exe文件直接在工程目录下，不能在工程目录的子文件中，否则就只能选择后面两种执行方式。   关于OMNeT++工程如何调试不再说明，其调试的方式与程序执行的方式相似，同时与其他程序的调试一样使用gdb调试，其中设置断点、单步调试或者进入函数内部等基本一样，以及添加观察变量。

第 3 章

初入OMNeT++

       欢迎来到第三章，本章主要介绍OMNeT++官方已经提供的学习资料有哪些，并以OMNeT++内一系类tictoc作为实例进行简单的设计说明，通过本章你可以快速的了解到如何学习OMNeT++、掌握官方的学习资料和利用OMNeT++可以做哪些事情。

3.1 学习map

       就目前学习OMNeT++的资料来说，网上的资料有：

[1] 《omnet++中文使用手册》

[2] 《OMNeT++与网络仿真》

[3] 《OMNeT++网络仿真》（紫色）

       目前较全的资料就上面三种，其中前两种参考价值比较好一些，其中第一本就是OMNeT++官方提供的资料的翻译版，主要介绍范范的仿真程序设计，不能称其为学习教程，应该叫参考资料。第二本《OMNeT++与网络仿真》与第一本相比，在仿真程序的设计时更有价值一些，对部分函数接口有介绍，但是没有给出使用场景。其实到目前，作者认为还是官方提供的入门手册对初学者较友好一些，但是问题在于初学的时候我们不知道它的存在，包括我在初学的时候也是恍恍惚惚的，为了使读者在初学的时候就更好的利用这些资料，我在这里总结出官方到底提供了哪些资料。

3.1.1 OMNeT++文档与指导书

       在OMNeT++安装路径下，官方提供了较多的使用指南，大多数以网页的形式给出。第一个要介绍的就是包括安装手册在内的多个文档入口：

路径：omnetpp-5.2/doc/index.html

       其内容包括从软件安装、初学Tictoc多个仿真例子、API参考到提升篇：IDE自定义指南和并行仿真指南等，详细如下：

介绍、指导手册

安装指导

IDE 浏览

TicToc指导手册

文档

仿真手册

IDE用户指南

API参考书

其他

IDE开发者指导

IDE自定义指南

并行仿真指南

NEDXML接口函数

       这里都是以中文的形式展现出官方提供的资料目录，而原目录都是以英文的形式给出。

3.1.2 tictoc指导手册

       tictoc相当于程序中的**hello world级别的例子，初学OMNeT++一般通过仿真修改tictoc例子，其路径在软件的安装路径下，点击该路径下的index.html**：

路径：omnetpp-5.2/doc/tictoc-tutorial/index.html

       包括的内容如下：

开始：一个简单的仿真模型（tictoc1.ned txc1.cc omnetpp.ini）

仿真程序的执行和仿真

改进两个节点仿真模型（tictoc9.ned txc9.cc omnetpp.ini）

一个复杂的网络（tictoc13.ned, tictoc13.msg, txc13.cc, omnetpp.ini）

如何添加统计量（tictoc17.ned, tictoc17.msg, txc17.cc, omnetpp.ini）

如何可视化观察仿真结果

如何添加参数 （在omnetpp.ini中配置.ned文件需要的参数）

       对于以上资料是目前入学OMNeT++较全系统的资料，从工程搭建、调试到添加统计量这些都是实际的网络仿真程序中一般会用到的，比如统计量，一般在网络中包括端到端延迟、入队排队时间、丢包数等。最后的仿真结果可视化观察，OMNeT++仿真程序结束后，在out文件下会生成仿真结果文件，OMNeT++提供可视化工具观察程序中统计的变量，可以转换成直方图和折线图，在后续会详细说明如何使用OMNeT++提供的观察和分析仿真结果工具。

3.1.3 仿真手册

       官方也提供了一个较详细的仿真手册，这里还是介绍一下这个手册。

路径：omnetpp-5.2/doc/manual/index.html

       入口如下（绿色部分标出）

       Simulation Manual仿真手册提供了设计一个OMNeT++工程各方面详细的介绍，从某种意义上来说，本手册也许就是Simulation Manual手册的一个子集，但是为了使本手册的意义更大，我将结合自己的几个月使用OMNeT++的经验，提供一些在设计网络时可能会出现的问题，以及解决办法。

3.2 个性化IDE

       添加这一节，只是个人乐趣而已，我曾经修改代码高亮花了一个下午的时间，每次总是很难找到相关的设置地方，只能说OMNeT++代码高亮设置放置的太隐秘了，到目前为止，我还是不能找到修改.ned文件的高亮设置（似乎没有这个功能）。相关.cc文件的设置地方下面会简单指明，其他更为详细的内容需要读者自行发挥。

3.2.1 CPP高亮设置

       首先，进入到IDE设置界面：Window 》》 Preferences，如图：

cpp高亮设置

       从上图也可以看出，我们需要选择c/c++》》Syntax Coloring，根据图中的窗口选择我们需要修改的高亮块设置。其中包括以下五种：

Code:代码块

Assembly: 汇编块

Comments: 注释块

Preprocessor: 预处理器块

Doxygen: 其他

       你可以根据自己的爱好选择不同的块设置，可选择颜色、加粗、斜体或者下划线等。

3.2.2 其他设置

显示行号

       在OMNeT++下默认没有显示行号的，但是毕竟有这个毛病，不显示行号就写不下去程序。   如下图所示：

其他设置界面

       在这一界面中，我们Tab键宽度、显示行号或者当前行背景,其他设置读者可自行编辑看看。

3.3 本章小结

 ； ； ；OMNeT++仿真平台基于Eclipse，相关编译、工程设置与其他软件不同，且难找，本章列出仿真时读者可能需要用到的相关设置。

第 4 章

OMNeT++仿真类

       在完成第五章后，考虑需要在之前加一章节关于OMNeT++类说明，在这个仿真软件中，主要使用的语言是C++，因此大多数数据类型是类或者结构，本章还是走其他技术书一样的老路线，注释这些数据类型，对类成员函数进行说明，可能与第五章有些重复的地方，但是其五章更多的偏向于实际应用，可能读者看过这里后，会发现OMNeT++接口是真好用。

4.1 类说明

4.1.1 cModule

       为了能更好的解释这个的库的使用，程序清单4.1为类cModule原型，cModule类在OMNeT++中表示一个节点的对象，这个节点可以是复合节点或者简单节点，通过这个类，程序员可以访问描述这个节点的.ned文件中设置的参数，或者是由omnetpp.ini传入的参数。简而言之，我们最后就是面向这些类进行网络设计。

程序清单 4.1

```
class SIM_API cModule : public cComponent //implies noncopyable
{
    friend class cGate;
    friend class cSimulation;
    friend class cModuleType;
    friend class cChannelType;

  public:
    /*
     * 模块门的迭代器
     * Usage:
     * for (cModule::GateIterator it(module); !it.end(); ++it) {
     *     cGate *gate = *it;
     *     ...
     * }
     */
    class SIM_API GateIterator
    {
```

```
21        ...
22      };
23
24      /*
25       * 复合模块的子模块迭代器
26       * Usage:
27       * for (cModule::SubmoduleIterator it(module); !it.end(); ++it) {
28       *     cModule *submodule = *it;
29       *     ...
30       * }
31       */
32      class SIM_API SubmoduleIterator
33      {
34        ...
35      };
36
37      /*
38       * 模块信道迭代器
39       * Usage:
40       * for (cModule::ChannelIterator it(module); !it.end(); ++it) {
41       *     cChannel *channel = *it;
42       *     ...
43       * }
44       */
45      class SIM_API ChannelIterator
46      {
47        ...
48      };
49
50    public:
51      // internal: currently used by init
52      void setRecordEvents(bool e)  {setFlag(FL_RECORD_EVENTS,e);}
53      bool isRecordEvents() const  {return flags&FL_RECORD_EVENTS;}
54
55    public:
56  #ifdef USE_OMNETPP4x_FINGERPRINTS
57      // internal: returns OMNeT++ V4.x compatible module ID
58      int getVersion4ModuleId() const { return version4ModuleId; }
59  #endif
60
61      // internal: may only be called between simulations, when no modules exist
62      static void clearNamePools();
63
64      // internal utility function. Takes O(n) time as it iterates on the gates
65      int gateCount() const;
66
67      // internal utility function. Takes O(n) time as it iterates on the gates
68      cGate *gateByOrdinal(int k) const;
69
```

```
70    // internal: calls refreshDisplay() recursively
71    virtual void callRefreshDisplay() override;
72
73    // internal: return the canvas if exists, or nullptr if not (i.e. no create-on-demanc
74    cCanvas *getCanvasIfExists() {return canvas;}
75
76    // internal: return the 3D canvas if exists, or nullptr if not (i.e. no create-on-dem
77    cOsgCanvas *getOsgCanvasIfExists() {return osgCanvas;}
78
79  public:
80
81    /** @name Redefined cObject member functions. */
82    //@{
83
84    /**
85     * Calls v->visit(this) for each contained object.
86     * See cObject for more details.
87     */
88    virtual void forEachChild(cVisitor *v) override;
89
90    /**
91     * Sets object's name. Redefined to update the stored fullName string.
92     */
93    virtual void setName(const char *s) override;
94
95    /**
96     * Returns the full name of the module, which is getName() plus the
97     * index in square brackets (e.g. "module[4]"). Redefined to add the
98     * index.
99     */
100   virtual const char *getFullName() const override;
101
102   /**
103    * Returns the full path name of the module. Example: <tt>"net.node[12].gen"</tt>.
104    * The original getFullPath() was redefined in order to hide the global cSimulation
105    * instance from the path name.
106    */
107   virtual std::string getFullPath() const override;
108
109   /**
110    * Overridden to add the module ID.
111    */
112   virtual std::string str() const override;
113   //@}
114
115   /** @name Setting up the module. */
116   //@{
117
118   /**
```

```
119      * Adds a gate or gate vector to the module. Gate vectors are created with
120      * zero size. When the creation of a (non-vector) gate of type cGate::INOUT
121      * is requested, actually two gate objects will be created, "gatename$i"
122      * and "gatename$o". The specified gatename must not contain a "$i" or "$o"
123      * suffix itself.
124      *
125      * CAUTION: The return value is only valid when a non-vector INPUT or OUTPUT
126      * gate was requested. nullptr gets returned for INOUT gates and gate vectors.
127      */
128     virtual cGate *addGate(const char *gatename, cGate::Type type, bool isvector=false);
129
130     /**
131      * Sets gate vector size. The specified gatename must not contain
132      * a "$i" or "$o" suffix: it is not possible to set different vector size
133      * for the "$i" or "$o" parts of an inout gate. Changing gate vector size
134      * is guaranteed NOT to change any gate IDs.
135      */
136     virtual void setGateSize(const char *gatename, int size);
137
138     /*
139      * 下面的接口是关于模块自己的信息
140      */
141     // 复合模块还是简单模块
142     virtual bool isSimple() const;
143
144     /**
145      * Redefined from cComponent to return KIND_MODULE.
146      */
147     virtual ComponentKind getComponentKind() const override  {return KIND_MODULE;}
148
149     /**
150      * Returns true if this module is a placeholder module, i.e.
151      * represents a remote module in a parallel simulation run.
152      */
153     virtual bool isPlaceholder() const  {return false;}
154
155     // 返回模块的父模块，对于系统模块，返回nullptr
156     virtual cModule *getParentModule() const override;
157
158     /**
159      * Convenience method: casts the return value of getComponentType() to cModuleType.
160      */
161     cModuleType *getModuleType() const  {return (cModuleType *)getComponentType();}
162
163     // 返回模块属性，属性在运行时不能修改
164     virtual cProperties *getProperties() const override;
165
166     // 如何模块是使用向量的形式定义的，返回true
167     bool isVector() const  {return vectorSize>=0;}
```

```
168
169     // 返回模块在向量中的索引
170     int getIndex() const  {return vectorIndex;}
171
172     // 返回这个模块向量的大小，如何该模块不是使用向量的方式定义的，返回1
173     int getVectorSize() const  {return vectorSize<0 ? 1 : vectorSize;}
174
175     // 与getVectorSize功能相似()
176     _OPPDEPRECATED int size() const  {return getVectorSize();}
177
178
179     /*
180      * 子模块相关功能
181      */
182
183     // 检测该模块是否有子模块
184     virtual bool hasSubmodules() const {return firstSubmodule!=nullptr;}
185
186     // 寻找子模块，找到返回模块，否则返回nameID-1
187     // 如何模块采用向量形式定义，那么需要指明index
188     virtual int findSubmodule(const char *name, int index=-1) const;
189
190     // 直接得到子模块的指针，没有这个子模块返回namenullptr
191     // 如何模块采用向量形式定义，那么需要指明index
192     virtual cModule *getSubmodule(const char *name, int index=-1) const;
193
194     /*
195      * 一个更强大的获取模块指针的接口，通过路径获取
196      *
197      * Examples:
198      *   "" means nullptr.
199      *   "." means this module;
200      *   "<root>" means the toplevel module;
201      *   ".sink" means the sink submodule of this module;
202      *   ".queue[2].srv" means the srv submodule of the queue[2] submodule;
203      *   "^.host2" or ".^.host2" means the host2 sibling module;
204      *   "src" or "<root>.src" means the src submodule of the toplevel module;
205      *   "Net.src" also means the src submodule of the toplevel module, provided
206      *   it is called Net.
207      *
208      *  @see cSimulation::getModuleByPath()
209      */
210     virtual cModule *getModuleByPath(const char *path) const;
211
212     /*
213      * 门的相关操作
214      */
215
216     /**
```

```
217      * Looks up a gate by its name and index. Gate names with the "$i" or "$o"
218      * suffix are also accepted. Throws an error if the gate does not exist.
219      * The presence of the index parameter decides whether a vector or a scalar
220      * gate will be looked for.
221      */
222     virtual cGate *gate(const char *gatename, int index=-1);
223
224     /**
225      * Looks up a gate by its name and index. Gate names with the "$i" or "$o"
226      * suffix are also accepted. Throws an error if the gate does not exist.
227      * The presence of the index parameter decides whether a vector or a scalar
228      * gate will be looked for.
229      */
230     const cGate *gate(const char *gatename, int index=-1) const {
231         return const_cast<cModule *>(this)->gate(gatename, index);
232     }
233
234
235     /**
236      * Returns the "$i" or "$o" part of an inout gate, depending on the type
237      * parameter. That is, gateHalf("port", cGate::OUTPUT, 3) would return
238      * gate "port$o[3]". Throws an error if the gate does not exist.
239      * The presence of the index parameter decides whether a vector or a scalar
240      * gate will be looked for.
241      */
242     const cGate *gateHalf(const char *gatename, cGate::Type type, int index=-1) const {
243         return const_cast<cModule *>(this)->gateHalf(gatename, type, index);
244     }
245
246     // 检测是否有门
247     virtual bool hasGate(const char *gatename, int index=-1) const;
248
249     // 寻找门，如果没有返回，找到返回门-1ID
250     virtual int findGate(const char *gatename, int index=-1) const;
251
252     // 通过得到门地址，目前我还没有用到过ID
253     const cGate *gate(int id) const {return const_cast<cModule *>(this)->gate(id);}
254
255     // 删除一个门（很少用）
256     virtual void deleteGate(const char *gatename);
257
258
259     //返回模块门的名字，只是基本名字不包括向量门的索引(, "[]" or the "$i"/"$o) "
260     virtual std::vector<const char *> getGateNames() const;
261
262     // 检测门（向量门）类型，可以标明"$i","$o"
263     virtual cGate::Type gateType(const char *gatename) const;
264
265     // 检测是否是向量门，可以标明"$i","$o"
```

```
266     virtual bool isGateVector(const char *gatename) const;
267
268     // 得到门的大小，可以指明"$i","$o"
269     virtual int gateSize(const char *gatename) const;
270
271     // 对于向量门，返回的号gate0ID
272     // 对于标量，返回IDID
273     // 一个公式：ID = gateBaseId + index
274     // 如果没有该门，抛出一个错误
275     virtual int gateBaseId(const char *gatename) const;
276
277     /**
278      * For compound modules, it checks if all gates are connected inside
279      * the module (it returns <tt>true</tt> if they are OK); for simple
280      * modules, it returns <tt>true</tt>. This function is called during
281      * network setup.
282      */
283     virtual bool checkInternalConnections() const;
284
285     /**
286      * This method is invoked as part of a send() call in another module.
287      * It is called when the message arrives at a gates in this module which
288      * is not further connected, that is, the gate's getNextGate() method
289      * returns nullptr. The default, cModule implementation reports an error
290      * ("message arrived at a compound module"), and the cSimpleModule
291      * implementation inserts the message into the FES after some processing.
292      */
293     virtual void arrived(cMessage *msg, cGate *ongate, simtime_t t);
294     //@}
295
296     /*
297      * 公用的
298      */
299     // 在父模块中寻找某个参数，没找到抛出cRuntimeError
300     virtual cPar& getAncestorPar(const char *parname);
301
302     /**
303      * Returns the default canvas for this module, creating it if it hasn't
304      * existed before.
305      */
306     virtual cCanvas *getCanvas() const;
307
308     /**
309      * Returns the default 3D (OpenSceneGraph) canvas for this module, creating
310      * it if it hasn't existed before.
311      */
312     virtual cOsgCanvas *getOsgCanvas() const;
313
314     // 设置是否在此模块的图形检查器上请求内置动画。
```

```
315     virtual void setBuiltinAnimationsAllowed(bool enabled) {setFlag(FL_BUILTIN_ANIMATIONS
316
317     /**
318      * Returns true if built-in animations are requested on this module's
319      * graphical inspector, and false otherwise.
320      */
321     virtual bool getBuiltinAnimationsAllowed() const {return flags & FL_BUILTIN_ANIMATION
322     //@}
323
324     /** @name Public methods for invoking initialize()/finish(), redefined from cComponer
325      * initialize(), numInitStages(), and finish() are themselves also declared in
326      * cComponent, and can be redefined in simple modules by the user to perform
327      * initialization and finalization (result recording, etc) tasks.
328      */
329     //@{
330     /**
331      * Interface for calling initialize() from outside.
332      */
333     virtual void callInitialize() override;
334
335     /**
336      * Interface for calling initialize() from outside. It does a single stage
337      * of initialization, and returns <tt>true</tt> if more stages are required.
338      */
339     virtual bool callInitialize(int stage) override;
340
341     /**
342      * Interface for calling finish() from outside.
343      */
344     virtual void callFinish() override;
345
346
347     /*
348      * 动态模块创建
349      */
350
351     /**
352      * Creates a starting message for modules that need it (and recursively
353      * for its submodules).
354      */
355     virtual void scheduleStart(simtime_t t);
356
357     // 删除自己
358     virtual void deleteModule();
359
360     // 移动该模块到另一个父模块下，一般用于移动场景。规则较复杂，可到原头文件查看使
    用说明
361     virtual void changeParentTo(cModule *mod);
362 };
```

       cModule是OMNeT++中用于代表一个模块的对象实体，如果你在编写网络仿真代码时，这个模块可以是简单模块或者复合模块，当需要得到这个模块相关属性时可以考虑到这个cModule类里边找找，说不定有意外的惊喜，也许有现成的函数实现你需要的功能。下面将这个类原型解剖看看：

迭代器：GateIterator

```
 1 :
2 usage
3 for (cModule::GateIterator it(module); !it.end(); ++it) {
4         cGate *gate = *it;
5         ...
6 }
```

该迭代器可用于遍历模块module的门向量，得到该门可用于其他作用。

迭代器：SubmoduleIterator

```
1 usage:
2 for (cModule::SubmoduleIterator it(module); !it.end(); ++it) {
3         cModule *submodule = *it;
4         ...
5 }
```

对于一个复合模块，包括多个简单模块或者复合模块，可使用该迭代器进行遍历操作，在第五章涉及到这个迭代器的使用。

迭代器：ChannelIterator

```
1 usage:
2 for (cModule::ChannelIterator it(module); !it.end(); ++it) {
3         cChannel *channel = *it;
4         ...
5 }
```

可用于遍历该模块的所有的信道。

4.1.2 cPar

       cPar同样是我们设置网络时不可避免的类，通过cPar得到节点在网络拓扑文件和配置文件中设置的参数，浏览完cPar所有成员函数，可以看出cPar基本提供了网络设计者想要的所有数据转换接口。

```
1 class SIM_API cPar : public cObject
2 {
3     friend class cComponent;
4   public:
5     enum Type {
6         BOOL = 'B',
7         DOUBLE = 'D',
8         LONG = 'L',
```

```
9         STRING = 'S',
10        XML = 'X'
11    };
12
13  private:
14    cComponent *ownerComponent;
15    cParImpl *p;
16    cComponent *evalContext;
17
18  private:
19    // private constructor and destructor -- only cComponent is allowed to create paramet
20    cPar() {ownerComponent = evalContext = nullptr; p = nullptr;}
21    virtual ~cPar();
22    // internal, called from cComponent
23    void init(cComponent *ownercomponent, cParImpl *p);
24    // internal
25    void moveto(cPar& other);
26    // internal: called each time before the value of this object changes.
27    void beforeChange();
28    // internal: called each time after the value of this object changes.
29    void afterChange();
30
31  public:
32    // internal, used by cComponent::finalizeParameters()
33    void read();
34    // internal, used by cComponent::finalizeParameters()
35    void finalize();
36    // internal: applies the default value if there is one
37    void acceptDefault();
38    // internal
39    void setImpl(cParImpl *p);
40    // internal
41    cParImpl *impl() const {return p;}
42    // internal
43    cParImpl *copyIfShared();
44
45 #ifdef SIMFRONTEND_SUPPORT
46    // internal
47    virtual bool hasChangedSince(int64_t lastRefreshSerial);
48 #endif
49
50  public:
51    /** @name Redefined cObject methods */
52    //@{
53    /**
54     * Assignment operator.
55     */
56    void operator=(const cPar& other);
57
```

```
58      // 返回参数的名字
59      virtual const char *getName() const override;
60
61      // 以字符串的形式返回参数
62      virtual std::string str() const override;
63
64      /**
65       * Returns the component (module/channel) this parameter belongs to.
66       * Note: return type is cObject only for technical reasons, it can be
67       * safely cast to cComponent.
68       */
69      virtual cObject *getOwner() const override; // note: cannot return cComponent* (covar
70
71      /**
72       * Calls v->visit(this) for contained objects.
73       * See cObject for more details.
74       */
75      virtual void forEachChild(cVisitor *v) override;
76      //@}
77
78      /** @name Type, flags. */
79      //@{
80      /**
81       * Returns the parameter type
82       */
83      Type getType() const;
84
85      /**
86       * Returns the given type as a string.
87       */
88      static const char *getTypeName(Type t);
89
90      /**
91       * Returns true if the stored value is of a numeric type.
92       */
93      bool isNumeric() const;
94
95      /**
96       * Returns true if this parameter is marked in the NED file as "volatile".
97       * This flag affects the operation of setExpression().
98       */
99      bool isVolatile() const;
100
101     /**
102      * Returns false if the stored value is a constant, and true if it is
103      * an expression. (It is not examined whether the expression yields
104      * a constant value.)
105      */
106     bool isExpression() const;
```

```
107
108     /**
109      * Returns true if the parameter value expression is shared among several
110      * modules to save memory. This flag is purely informational, and whether
111      * a parameter is shared or not does not affect operation at all.
112      */
113     bool isShared() const;
114
115     /**
116      * Returns true if the parameter is assigned a value, and false otherwise.
117      * Parameters of an already initialized module or channel are guaranteed to
118      * assigned, so this method will return true for them.
119      */
120     bool isSet() const;
121
122     /**
123      * Returns true if the parameter is set (see isSet()) or contains a default
124      * value, and false otherwise. Parameters of an already initialized module or
125      * channel are guaranteed to be assigned, so this method will return true for them.
126      */
127     bool containsValue() const;
128
129     /**
130      * Return the properties for this parameter. Properties cannot be changed
131      * at runtime.
132      */
133     cProperties *getProperties() const;
134     //@}
135
136     /** @name Setter functions. Note that overloaded assignment operators also exist. */
137     //@{
138
139     /**
140      * Sets the value to the given bool value.
141      */
142     cPar& setBoolValue(bool b);
143
144     /**
145      * Sets the value to the given long value.
146      */
147     cPar& setLongValue(long l);
148
149     /**
150      * Sets the value to the given double value.
151      */
152     cPar& setDoubleValue(double d);
153
154     /**
155      * Sets the value to the given string value.
```

```
156      * The cPar will make its own copy of the string. nullptr is also accepted
157      * and treated as an empty string.
158      */
159     cPar& setStringValue(const char *s);
160
161     /**
162      * Sets the value to the given string value.
163      */
164     cPar& setStringValue(const std::string& s)  {setStringValue(s.c_str()); return *this;
165
166     /**
167      * Sets the value to the given cXMLElement.
168      */
169     cPar& setXMLValue(cXMLElement *node);
170
171     /**
172      * Sets the value to the given expression. This object will assume
173      * the responsibility to delete the expression object.
174      *
175      * The evalcontext parameter determines the module or channel in the
176      * context of which the expression will be evaluated. If evalcontext
177      * is nullptr, the owner of this parameter will be used.
178      *
179      * Note: if the parameter is marked as non-volatile (isVolatile()==false),
180      * one should not set an expression as value. This is not enforced
181      * by cPar though.
182      *
183      * @see getOwner(), getEvaluationContext(), setEvaluationContext()
184      */
185     cPar& setExpression(cExpression *e, cComponent *evalcontext=nullptr);
186
187     /**
188      * If the parameter contains an expression (see isExpression()), this method
189      * sets the evaluation context for the expression.
190      *
191      * @see getEvaluationContext(), isExpression(), setExpression()
192      */
193     void setEvaluationContext(cComponent *ctx)  {evalContext = ctx;}
194     //@}
195
196     /** @name Getter functions. Note that overloaded conversion operators also exist. */
197     //@{
198
199     /**
200      * Returns value as a boolean. The cPar type must be BOOL.
201      */
202     bool boolValue() const;
203
204     /**
```

```
205      * Returns value as long. The cPar type must be LONG or DOUBLE.
206      */
207     long longValue() const;
208
209     /**
210      * Returns value as double. The cPar type must be LONG or DOUBLE.
211      */
212     double doubleValue() const;
213
214     /**
215      * Returns the parameter's unit ("s", "mW", "Hz", "bps", etc),
216      * as declared with the @unit property of the parameter in NED,
217      * or nullptr if no unit was specified. Unit is only valid for LONG and DOUBLE
218      * types.
219      */
220     const char *getUnit() const;
221
222     /**
223      * Returns value as const char *. The cPar type must be STRING.
224      * This method may only be invoked when the parameter's value is a
225      * string constant and not the result of expression evaluation, otherwise
226      * an error is thrown. This practically means this method cannot be used
227      * on parameters declared as "volatile string" in NED; they can only be
228      * accessed using stdstringValue().
229      */
230     const char *stringValue() const;
231
232     /**
233      * Returns value as string. The cPar type must be STRING.
234      */
235     std::string stdstringValue() const;
236
237     /**
238      * Returns value as pointer to cXMLElement. The cPar type must be XML.
239      *
240      * The lifetime of the returned object tree is undefined, but it is
241      * valid at least until the end of the current simulation event or
242      * initialize() call. Modules are expected to process their XML
243      * configurations at once (within one event or within initialize()),
244      * and not hang on to pointers returned from this method. The reason
245      * for the limited lifetime is that this method may return pointers to
246      * objects stored in an internal XML document cache, and the simulation
247      * kernel reserves the right to discard cached XML documents at any time
248      * to free up memory, and re-load them on demand (i.e. when xmlValue() is
249      * called again).
250      */
251     cXMLElement *xmlValue() const;
252
253     /**
```

```
254      * Returns pointer to the expression stored by the object, or nullptr.
255      */
256     cExpression *getExpression() const;
257
258     /**
259      * If the parameter contains an expression, this method returns the
260      * module or channel in the context of which the expression will be
261      * evaluated. (The context affects the resolution of parameter
262      * references, and NED operators like <tt>index</tt> or <tt>sizeof()</tt>.)
263      * If the parameter does not contain an expression, the return value is
264      * undefined.
265      *
266      * @see isExpression(), setEvaluationContext()
267      */
268     cComponent *getEvaluationContext() const  {return evalContext;}
269     //@}
270
271     /** @name Miscellaneous utility functions. */
272     //@{
273     /**
274      * For non-const values, replaces the stored expression with its
275      * evaluation.
276      */
277     void convertToConst();
278
279     /**
280      * Converts the value from string, and stores the result.
281      * If the text cannot be parsed, an exception is thrown, which
282      * can be caught as std::runtime_error& if necessary.
283      *
284      * Note: this method understands expressions too, but does NOT handle
285      * the special values "default" and "ask".
286      */
287     void parse(const char *text);
288     //@}
289
290     /** @name Overloaded assignment and conversion operators. */
291     //@{
292
293     /**
294      * Equivalent to setBoolValue().
295      */
296     cPar& operator=(bool b)  {return setBoolValue(b);}
297
298     /**
299      * Converts the argument to long, and calls setLongValue().
300      */
301     cPar& operator=(char c)  {return setLongValue((long)c);}
302
```

```
303     /**
304      * Converts the argument to long, and calls setLongValue().
305      */
306     cPar& operator=(unsigned char c)  {return setLongValue((long)c);}
307
308     /**
309      * Converts the argument to long, and calls setLongValue().
310      */
311     cPar& operator=(int i)  {return setLongValue((long)i);}
312
313     /**
314      * Converts the argument to long, and calls setLongValue().
315      */
316     cPar& operator=(unsigned int i)  {return setLongValue((long)i);}
317
318     /**
319      * Converts the argument to long, and calls setLongValue().
320      */
321     cPar& operator=(short i)  {return setLongValue((long)i);}
322
323     /**
324      * Converts the argument to long, and calls setLongValue().
325      */
326     cPar& operator=(unsigned short i)  {return setLongValue((long)i);}
327
328     /**
329      * Equivalent to setLongValue().
330      */
331     cPar& operator=(long l)  {return setLongValue(l);}
332
333     /**
334      * Converts the argument to long, and calls setLongValue().
335      */
336     cPar& operator=(unsigned long l) {return setLongValue((long)l);}
337
338     /**
339      * Equivalent to setDoubleValue().
340      */
341     cPar& operator=(double d)  {return setDoubleValue(d);}
342
343     /**
344      * Converts the argument to double, and calls setDoubleValue().
345      */
346     cPar& operator=(long double d)  {return setDoubleValue((double)d);}
347
348     // 等同于setStringValue函数()
349     cPar& operator=(const char *s)  {return setStringValue(s);}
350
351     // 等同于setStringValue函数()
```

```
352     cPar& operator=(const std::string& s)  {return setStringValue(s);}
353
354      // 等同于setXMLValue函数()
355     cPar& operator=(cXMLElement *node)  {return setXMLValue(node);}
356
357     operator bool() const  {return boolValue();}
358
359     operator char() const  {return (char)longValue();}
360
361     operator unsigned char() const  {return (unsigned char)longValue();}
362
363     operator int() const  {return (int)longValue();}
364
365     operator unsigned int() const  {return (unsigned int)longValue();}
366
367     operator short() const  {return (short)longValue();}
368
369     operator unsigned short() const  {return (unsigned short)longValue();}
370
371     // 返回值，与longlongValue相同()
372     operator long() const  {return longValue();}
373
374     /**
375     // 调用longValue，转换结果为()unsigned 类型long
376      */
377     operator unsigned long() const  {return longValue();}
378
379     // 返回值，与doubledoubleValue相同()
380     operator double() const  {return doubleValue();}
381
382     /**
383     // 调用doubleValue，将结果转换成()long 类型返回double
384      */
385     operator long double() const  {return doubleValue();}
386
387     // 与stringValue()
388     operator const char *() const  {return stringValue();}
389
390     // 与stdstringValue功能一样()
391     operator std::string() const  {return stdstringValue();}
392
393     // 与xmlVlaue等同。注意：返回对象树的生命周期被限制了，具体看()说明。xmlValue
394     operator cXMLElement *() const  {return xmlValue();}
395 };
```

4.1.3 cGate

       如果你需要在网络仿真运行时，动态实现两个节点之间的连接或者断开，那么你就需要在程序中用到这个类。

```
class SIM_API cGate : public cObject, noncopyable
{
    friend class cModule;
    friend class cModuleGates;
    friend class cPlaceholderModule;

  public:
    /**
     * Gate type
     */
    enum Type {
        NONE = 0,
        INPUT = 'I',
        OUTPUT = 'O',
        INOUT = 'B'
    };

  protected:
    // internal
    struct SIM_API Name
    {
        opp_string name;  // "foo"
        opp_string namei; // "foo$i"
        opp_string nameo; // "foo$o"
        Type type;
        Name(const char *name, Type type);
        bool operator<(const Name& other) const;
    };

  public:
    // Internal data structure, only public for technical reasons (GateIterator).
    // One instance per module and per gate vector/gate pair/gate.
    // Note: gate name and type are factored out to a global pool.
    // Note2: to reduce sizeof(Desc), "size" might be stored in input.gatev[0],
    // although it might not be worthwhile the extra complication and CPU cycles.
    //
    struct Desc
    {
        cModule *owner;
        Name *name;  // pooled (points into cModule::namePool)
        int vectorSize; // gate vector size, or -1 if scalar gate; actually allocated siz
        union Gates { cGate *gate; cGate **gatev; };
        Gates input;
        Gates output;

        Desc() {owner=nullptr; vectorSize=-1; name=nullptr; input.gate=output.gate=nullpt
        bool inUse() const {return name!=nullptr;}
        Type getType() const {return name->type;}
```

```
49          bool isVector() const {return vectorSize>=0;}
50          const char *nameFor(Type t) const {return (t==INOUT||name->type!=INOUT) ? name->n
51          int indexOf(const cGate *g) const {return (g->pos>>2)==-1 ? 0 : g->pos>>2;}
52          bool deliverOnReceptionStart(const cGate *g) const {return g->pos&2;}
53          Type getTypeOf(const cGate *g) const {return (g->pos&1)==0 ? INPUT : OUTPUT;}
54          bool isInput(const cGate *g) const {return (g->pos&1)==0;}
55          bool isOutput(const cGate *g) const {return (g->pos&1)==1;}
56          int gateSize() const {return vectorSize>=0 ? vectorSize : 1;}
57          void setInputGate(cGate *g) {ASSERT(getType()!=OUTPUT && !isVector()); input.gate
58          void setOutputGate(cGate *g) {ASSERT(getType()!=INPUT && !isVector()); output.gat
59          void setInputGate(cGate *g, int index) {ASSERT(getType()!=OUTPUT && isVector());
60          void setOutputGate(cGate *g, int index) {ASSERT(getType()!=INPUT && isVector());
61          static int capacityFor(int size) {return size<8 ? (size+1)&~1 : size<32 ? (size+3
62      };
63
64    protected:
65      Desc *desc; // descriptor of the gate or gate vector, stored in cModule
66      int pos;    // b0: input(0) or output(1); b1: deliverOnReceptionStart bit;
67                  // rest (pos>>2): array index, or -1 if scalar gate
68
69      int connectionId;   // uniquely identifies the connection between *this and *nextgate
70      cChannel *channel;  // channel object (if exists)
71      cGate *prevGate;    // previous and next gate in the path
72      cGate *nextGate;
73
74      static int lastConnectionId;
75
76    protected:
77      // internal: constructor is protected because only cModule is allowed to create insta
78      explicit cGate();
79
80      // also protected: only cModule is allowed to delete gates
81      virtual ~cGate();
82
83      // internal
84      static void clearFullnamePool();
85
86      // internal
87      void installChannel(cChannel *chan);
88
89      // internal
90      void checkChannels() const;
91
92  #ifdef SIMFRONTEND_SUPPORT
93      // internal
94      virtual bool hasChangedSince(int64_t lastRefreshSerial);
95  #endif
96
97    public:
```

```
98      /** @name Redefined cObject member functions */
99      //@{
100     /*
101      * 例如返回门out
102      */
103     virtual const char *getName() const override;
104
105     /*
106      * 与getName不同，需要返回门索引，例如()out[4]
107      */
108     virtual const char *getFullName() const override;
109
110     /**
111      * Calls v->visit(this) for each contained object.
112      * See cObject for more details.
113      */
114     virtual void forEachChild(cVisitor *v) override;
115
116     /**
117      * Produces a one-line description of the object's contents.
118      * See cObject for more details.
119      */
120     virtual std::string str() const override;
121
122     /**
123      * Returns the owner module of this gate.
124      */
125     virtual cObject *getOwner() const override; // note: cannot return cModule* (covariar
126     //@}
127
128     /**
129      * This function is called internally by the send() functions and
130      * channel classes' deliver() to deliver the message to its destination.
131      * A false return value means that the message object should be deleted
132      * by the caller. (This is used e.g. with parallel simulation, for
133      * messages leaving the partition.)
134      */
135     virtual bool deliver(cMessage *msg, simtime_t at);
136
137     /** @name Connecting the gate. */
138     //@{
139     /**
140      * Connects the gate to another gate, using the given channel object
141      * (if one is specified). This method can be used to manually create
142      * connections for dynamically created modules.
143      *
144      * This method invokes callInitialize() on the channel object, unless the
145      * compound module containing this connection is not yet initialized
146      * (then it assumes that this channel will be initialized as part of the
```

```
147      * compound module initialization process.) To leave the channel
148      * uninitialized, specify true for the leaveUninitialized parameter.
149      *
150      * If the gate is already connected, an error will occur. The gate
151      * argument cannot be nullptr, that is, you cannot use this function
152      * to disconnect a gate; use disconnect() for that.
153      *
154      * Note: When you set channel parameters after channel initialization,
155      * make sure the channel class is implemented so that the changes take
156      * effect; i.e. the channel should either override and properly handle
157      * handleParameterChange(), or should not cache any values from parameters.
158      */
159     cChannel *connectTo(cGate *gate, cChannel *channel=nullptr, bool leaveUninitialized=f
160
161     /**
162      * Disconnects the gate, and also deletes the associated channel object
163      * if one has been set. disconnect() must be invoked on the source gate
164      * ("from" side) of the connection.
165      *
166      * The method has no effect if the gate is not connected.
167      */
168     void disconnect();
169
170     /**
171      * Disconnects the gate, then connects it again to the same gate, with the
172      * given channel object (if not nullptr). The gate must be connected.
173      *
174      * @see connectTo()
175      */
176     cChannel *reconnectWith(cChannel *channel, bool leaveUninitialized=false);
177     //@}
178
179     /** @name Information about the gate. */
180     //@{
181     /**
182      * Returns the gate name without index and potential "$i"/"$o" suffix.
183      */
184     const char *getBaseName() const;
185
186     /**
187      * Returns the suffix part of the gate name ("$i", "$o" or "").
188      */
189     const char *getNameSuffix() const;
190
191     /**
192      * Returns the properties for this gate. Properties cannot be changed
193      * at runtime.
194      */
195     cProperties *getProperties() const;
```

```
196
197     /**
198      * Returns the gate's type, cGate::INPUT or cGate::OUTPUT. (It never returns
199      * cGate::INOUT, because a cGate object is always either the input or
200      * the output half of an inout gate ("name$i" or "name$o").
201      */
202     Type getType() const  {return desc->getTypeOf(this);}
203
204     /**
205      * Returns the given type as a string.
206      */
207     static const char *getTypeName(Type t);
208
209     /**
210      * Returns a pointer to the owner module of the gate.
211      */
212     cModule *getOwnerModule() const;
213
214     /**
215      * Returns the gate ID, which uniquely identifies the gate within the
216      * module. IDs are guaranteed to be contiguous within a gate vector:
217      * <tt>module->gate(id+index) == module->gate(id)+index</tt>.
218      *
219      * Gate IDs are stable: they are guaranteed not to change during
220      * simulation. (This is a new feature of \opp 4.0. In earlier releases,
221      * gate IDs could change when the containing gate vector was resized.)
222      *
223      * Note: As of \opp 4.0, gate IDs are no longer small integers, and
224      * cannot be used for iterating over the gates of a module.
225      * Use cModule::GateIterator for iteration.
226      */
227     int getId() const;
228
229     /**
230      * Returns true if the gate is part of a gate vector.
231      */
232     bool isVector() const  {return desc->isVector();}
233
234     /**
235      * If the gate is part of a gate vector, returns the ID of the first
236      * element in the gate vector. Otherwise, it returns the gate's ID.
237      */
238     int getBaseId() const;
239
240     /**
241      * If the gate is part of a gate vector, returns the gate's index in the vector.
242      * Otherwise, it returns 0.
243      */
244     int getIndex() const  {return desc->indexOf(this);}
```

```
245
246     /**
247      * If the gate is part of a gate vector, returns the size of the vector.
248      * For non-vector gates it returns 1.
249      *
250      * The gate vector size can also be obtained by calling the cModule::gateSize().
251      */
252     int getVectorSize() const  {return desc->gateSize();}
253
254     /**
255      * Alias for getVectorSize().
256      */
257     int size() const  {return getVectorSize();}
258
259     /**
260      * Returns the channel object attached to this gate, or nullptr if there is
261      * no channel. This is the channel between this gate and this->getNextGate(),
262      * that is, channels are stored on the "from" side of the connections.
263      */
264     cChannel *getChannel() const  {return channel;}
265
266     /**
267      * This method may only be invoked on input gates of simple modules.
268      * Messages with nonzero length then have a nonzero
269      * transmission duration (and thus, reception duration on the other
270      * side of the connection). By default, the delivery of the message
271      * to the module marks the end of the reception. Setting this bit will cause
272      * the channel to deliver the message to the module at the start of the
273      * reception. The duration that the reception will take can be extracted
274      * from the message object, by its getDuration() method.
275      */
276     void setDeliverOnReceptionStart(bool d);
277
278     /**
279      * Returns whether messages delivered through this gate will mark the
280      * start or the end of the reception process (assuming nonzero message length
281      * and data rate on the channel.)
282      *
283      * @see setDeliverOnReceptionStart()
284      */
285     bool getDeliverOnReceptionStart() const  {return pos&2;}
286     //@}
287
288     /** @name Transmission state. */
289     //@{
290     /**
291      * Typically invoked on an output gate, this method returns <i>the</i>
292      * channel in the connection path that supports datarate (as determined
293      * by cChannel::isTransmissionChannel(); it is guaranteed that there can be
```

```
294      * at most one such channel per path). If there is no such channel,
295      * an error is thrown.
296      *
297      * This method only checks the segment of the connection path that
298      * <i>starts</i> at this gate, so, for example, it is an error to invoke
299      * it on a simple module input gate.
300      *
301      * Note: this method searches the connection path linearly, so at
302      * performance-critical places it may be better to cache its return
303      * value (provided that connections are not removed or created dynamically
304      * during simulation.)
305      *
306      * @see cChannel::isTransmissionChannel()
307      */
308     cChannel *getTransmissionChannel() const;
309
310     /**
311      * Like getTransmissionChannel(), but returns nullptr instead of throwing
312      * an error if there is no transmission channel in the path.
313      */
314     cChannel *findTransmissionChannel() const;
315
316     /**
317      * Typically invoked on an input gate, this method searches the reverse
318      * path (i.e. calls getPreviousGate() repeatedly) for the transmission
319      * channel. It is guaranteed that there can be at most one such channel
320      * per path. If no transmission channel is found, the method throws an error.
321      *
322      * @see getTransmissionChannel(), cChannel::isTransmissionChannel()
323      */
324     cChannel *getIncomingTransmissionChannel() const;
325
326     /**
327      * Like getIncomingTransmissionChannel(), but returns nullptr instead of
328      * throwing an error if there is no transmission channel in the reverse
329      * path.
330      */
331     cChannel *findIncomingTransmissionChannel() const;
332     //@}
333
334     /** @name Gate connectivity. */
335     //@{
336
337     /**
338      * Returns the previous gate in the series of connections (the path) that
339      * contains this gate, or nullptr if this gate is the first one in the path.
340      * (E.g. for a simple module output gate, this function will return nullptr.)
341      */
342     cGate *getPreviousGate() const {return prevGate;}
```

```
343
344     /**
345      * Returns the next gate in the series of connections (the path) that
346      * contains this gate, or nullptr if this gate is the last one in the path.
347      * (E.g. for a simple module input gate, this function will return nullptr.)
348      */
349     cGate *getNextGate() const   {return nextGate;}
350
351     /**
352      * Returns an ID that uniquely identifies the connection between this gate
353      * and the next gate in the path (see getNextGate()) during the lifetime of
354      * the simulation. (Disconnecting and then reconnecting the gate results
355      * in a new connection ID being assigned.) The method returns -1 if the gate
356      * is unconnected.
357      */
358     int getConnectionId() const  {return connectionId;}
359
360     /**
361      * Return the ultimate source of the series of connections
362      * (the path) that contains this gate.
363      */
364     cGate *getPathStartGate() const;
365
366     /**
367      * Return the ultimate destination of the series of connections
368      * (the path) that contains this gate.
369      */
370     cGate *getPathEndGate() const;
371
372     /**
373      * Determines if a given module is in the path containing this gate.
374      */
375     bool pathContains(cModule *module, int gateId=-1);
376
377     /**
378      * Returns true if the gate is connected outside (i.e. to one of its
379      * sibling modules or to the parent module).
380      *
381      * This means that for an input gate, getPreviousGate() must be non-nullptr; for an o
382      * gate, getNextGate() must be non-nullptr.
383      */
384     bool isConnectedOutside() const;
385
386     /**
387      * Returns true if the gate (of a compound module) is connected inside
388      * (i.e. to one of its submodules).
389      *
390      * This means that for an input gate, getNextGate() must be non-nullptr; for an outpu
391      * gate, getPreviousGate() must be non-nullptr.
```

```
392      */
393     bool isConnectedInside() const;
394
395     /**
396      * Returns true if the gate fully connected. For a compound module gate
397      * this means both isConnectedInside() and isConnectedOutside() are true;
398      * for a simple module, only isConnectedOutside() is checked.
399      */
400     bool isConnected() const;
401
402     /**
403      * Returns true if the path (chain of connections) containing this gate
404      * starts and ends at a simple module.
405      */
406     bool isPathOK() const;
407     //@}
408
409     /** @name Display string. */
410     //@{
411     /**
412      * Returns the display string for the gate, which controls the appearance
413      * of the connection arrow starting from gate. The display string is stored
414      * in the channel associated with the connection. If there is no channel,
415      * this call creates an installs a cIdealChannel to hold the display string.
416      */
417     cDisplayString& getDisplayString();
418
419     /**
420      * Shortcut to <tt>getDisplayString().set(dispstr)</tt>.
421      */
422     void setDisplayString(const char *dispstr);
423     //@}
424 };
```

4.1.4 cTopology

4.1.5 cExpression

4.1.6 EV类

       一个对调试程序有帮助的类。

```
1 //==========================================================================
2 //   CLOG.H  -  header for
3 //                     OMNeT++/OMNEST
4 //            Discrete System Simulation in C++
```

```
5 //
6 //==========================================================================
7
8 /*--------------------------------------------------------------*
9   Copyright (C) 1992-2017 Andras Varga
10  Copyright (C) 2006-2017 OpenSim Ltd.
11
12  This file is distributed WITHOUT ANY WARRANTY. See the file
13  `license' for details on this and other legal matters.
14 *--------------------------------------------------------------*/
15
16 #ifndef __OMNETPP_CLOG_H
17 #define __OMNETPP_CLOG_H
18
19 #include <ctime>
20 #include <sstream>
21 #include "simkerneldefs.h"
22
23 namespace omnetpp {
24
25 class cObject;
26 class cComponent;
27
28 /**
29  * @brief Classifies log messages based on detail and importance.
30  *
31  * @ingroup Logging
32  */
33 enum LogLevel
34 {
35     /**
36      * The lowest log level; it should be used for low-level implementation-specific
37      * technical details that are mostly useful for the developers/maintainers of the
38      * component. For example, a MAC layer protocol component could log control flow
39      * in loops and if statements, entering/leaving methods and code blocks using this
40      * log level.
41      */
42     LOGLEVEL_TRACE,
43
44     /**
45      * This log level should be used for high-level implementation-specific technical
46      * details that are most likely important for the developers/maintainers of the
47      * component. These messages may help to debug various issues when one is looking
48      * at the code. For example, a MAC layer protocol component could log updates to
49      * internal state variables, updates to complex data structures using this log level.
50      */
51     LOGLEVEL_DEBUG,
52
53     /**
```

```
54      * This log level should be used for low-level protocol-specific details that
55      * may be useful and understandable by the users of the component. These messages
56      * may help to track down various protocol-specific issues without actually looking
57      * too deep into the code. For example, a MAC layer protocol component could log
58      * state machine updates, acknowledge timeouts and selected back-off periods using
59      * this level.
60      */
61     LOGLEVEL_DETAIL,
62
63     /**
64      * This log level should be used for high-level protocol specific details that
65      * are most likely important for the users of the component. For example, a MAC
66      * layer protocol component could log successful packet receptions and successful
67      * packet transmissions using this level.
68      */
69     LOGLEVEL_INFO,
70
71     /**
72      * This log level should be used for exceptional (non-error) situations that
73      * may be important for users and rarely occur in the component. For example,
74      * a MAC layer protocol component could log detected bit errors using this level.
75      */
76     LOGLEVEL_WARN,
77
78     /**
79      * This log level should be used for recoverable (non-fatal) errors that allow
80      * the component to continue normal operation. For example, a MAC layer protocol
81      * component could log unsuccessful packet receptions and unsuccessful packet
82      * transmissions using this level.
83      */
84     LOGLEVEL_ERROR,
85
86     /**
87      * The highest log level; it should be used for fatal (unrecoverable) errors
88      * that prevent the component from further operation. It doesn't mean that
89      * the simulation must stop immediately (because in such cases the code should
90      * throw a cRuntimeError), but rather that the a component is unable to continue
91      * normal operation. For example, a special purpose recording component may be
92      * unable to continue recording due to the disk being full.
93      */
94     LOGLEVEL_FATAL,
95
96     /**
97      * Not a real log level, it completely disables logging when set.
98      */
99     LOGLEVEL_OFF,
100 };
101
102 /**
```

```
103  * @brief For compile-time filtering of logs.
104  *
105  * One is free to define this macro before including <omnetpp.h>, or redefine
106  * it any time. The change will affect subsequent log statements.
107  * Log statements that use lower log levels than the one specified
108  * by this macro will not be compiled into the executable.
109  *
110  * @ingroup Logging
111  */
112 #ifndef COMPILETIME_LOGLEVEL
113 #ifdef NDEBUG
114 #define COMPILETIME_LOGLEVEL omnetpp::LOGLEVEL_DETAIL
115 #else
116 #define COMPILETIME_LOGLEVEL omnetpp::LOGLEVEL_TRACE
117 #endif
118 #endif
119
120 /**
121  * @brief This predicate determines if a log statement gets compiled into the
122  * executable.
123  *
124  * One is free to define this macro before including <omnetpp.h>, or redefine
125  * it any time. The change will affect subsequent log statements.
126  *
127  * @ingroup Logging
128  */
129 #ifndef COMPILETIME_LOG_PREDICATE
130 #define COMPILETIME_LOG_PREDICATE(object, logLevel, category) (logLevel >= COMPILETIME_LO
131 #endif
132
133 /**
134  * @brief This class groups logging related functionality.
135  *
136  * @see LogLevel
137  * @ingroup Logging
138  */
139 class SIM_API cLog
140 {
141   public:
142     typedef bool (*NoncomponentLogPredicate)(const void *object, LogLevel logLevel, const
143     typedef bool (*ComponentLogPredicate)(const cComponent *object, LogLevel logLevel, co
144
145   public:
146     /**
147      * This log level specifies a globally applied runtime modifiable filter. This is
148      * the fastest runtime filter, it works with a simple integer comparison at the call
149      * site.
150      */
151     static LogLevel logLevel;
```

```
152
153     /**
154      * This predicate determines if a log statement is executed for log statements
155      * that occur outside module or channel member functions. This is a customization
156      * point for logging.
157      */
158     static NoncomponentLogPredicate noncomponentLogPredicate;
159
160     /**
161      * This predicate determines if a log statement is executed for log statements
162      * that occur in module or channel member functions. This is a customization
163      * point for logging.
164      */
165     static ComponentLogPredicate componentLogPredicate;
166
167   public:
168     /**
169      * Returns a human-readable string representing the provided log level.
170      */
171     static const char *getLogLevelName(LogLevel logLevel);
172
173     /**
174      * Returns the associated log level for the provided human-readable string.
175      */
176     static LogLevel resolveLogLevel(const char *name);
177
178     static inline bool runtimeLogPredicate(const void *object, LogLevel logLevel, const c
179     { return noncomponentLogPredicate(object, logLevel, category); }
180
181     static inline bool runtimeLogPredicate(const cComponent *object, LogLevel logLevel, c
182     { return componentLogPredicate(object, logLevel, category); }
183
184     static bool defaultNoncomponentLogPredicate(const void *object, LogLevel logLevel, co
185     static bool defaultComponentLogPredicate(const cComponent *object, LogLevel logLevel,
186 };
187
188 // Creates a log proxy object that captures the provided context.
189 // This macro is internal to the logging infrastructure.
190 //
191 // NOTE: the (void)0 trick prevents GCC producing statement has no effect warnings
192 // for compile time disabled log statements.
193 //
194 #define OPP_LOGPROXY(object, logLevel, category) \
195     ((void)0, !(COMPILETIME_LOG_PREDICATE(object, logLevel, category) && \
196     omnetpp::cLog::runtimeLogPredicate(object, logLevel, category))) ? \
197     omnetpp::cLogProxy::dummyStream : omnetpp::cLogProxy(object, logLevel, category, __FI
198
199
200 // Returns nullptr. Helper function for the logging macros.
```

```
201 inline void *getThisPtr() {return nullptr;}
202
203 /**
204  * @brief Use this macro when logging from static member functions.
205  *
206  * Background: EV_LOG and derived macros (EV_INFO, EV_DETAIL, etc) will fail
207  * to compile when placed into static member functions of cObject-derived classes
208  * ("cannot call member function 'cObject::getThisPtr()' without object" in GNU C++,
209  * and "C2352: illegal call of non-static member function" in Visual C++).
210  * To fix it, add this macro at the top of the function; it contains local declarations
211  * to make the code compile.
212  *
213  * @ingroup Logging
214  * @hideinitializer
215  */
216 #define EV_STATICCONTEXT  void *(*getThisPtr)() = omnetpp::getThisPtr;
217
218 /**
219  * @brief This is the macro underlying EV_INFO, EV_DETAIL, EV_INFO_C, and
220  * similar log macros.
221  *
222  * This macro should not be used directly, but via the logging macros
223  * EV, EV_FATAL, EV_ERROR, EV_WARN, EV_INFO, EV_DETAIL, EV_DEBUG, EV_TRACE,
224  * and their "category" versions EV_C, EV_FATAL_C, EV_ERROR_C, EV_WARN_C,
225  * EV_INFO_C, EV_DETAIL_C, EV_DEBUG_C, EV_TRACE_C.
226  *
227  * Those macros act as C++ streams: one can write on them using the
228  * left-shift (<<) operator. Their names refer to the log level they
229  * represent (see LogLevel). The "category" (_C) versions accept a category
230  * string. Each category acts like a separate log channel; for example,
231  * one can use the "test" category to log text intended for consumption
232  * by an automated test suite.
233  *
234  * Log statements are wrapped with compile-time and runtime guards at the
235  * call site to efficiently prevent unnecessary computation of parameters
236  * and log content. Compile-time guards are COMPILETIME_LOGLEVEL and
237  * COMPILETIME_LOG_PREDICATE. Runtime guards (runtime log level) can be
238  * set up via omnetpp.ini.
239  *
240  * Under certain circumstances, compiling log statements may result in errors.
241  * When that happens, it is possible that the EV_STATICCONTEXT macro needs to
242  * be added to the code; please review its documentation for more info.
243  *
244  * Examples:
245  *
246  * \code
247  * EV_INFO << "Connection setup complete" << endl;
248  * EV_INFO_C("test") << "ESTAB" << endl;
249  * \endcode
```

```
250  *
251  * @see LogLevel, EV_STATICCONTEXT, EV_INFO, EV_INFO_C, COMPILETIME_LOGLEVEL, COMPILETIME
252  * @ingroup Logging
253  * @hideinitializer
254  */
255 #define EV_LOG(logLevel, category) OPP_LOGPROXY(getThisPtr(), logLevel, category).getStre
256
257 /**
258  * @brief Short for EV_INFO.
259  * @see EV_INFO @ingroup Logging
260  */
261 #define EV        EV_INFO
262
263 /**
264  * @brief Pseudo-stream for logging local fatal errors. See EV_LOG for details.
265  * @see EV_LOG @hideinitializer @ingroup Logging
266  */
267 #define EV_FATAL  EV_LOG(omnetpp::LOGLEVEL_FATAL, nullptr)
268
269 /**
270  * @brief Pseudo-stream for logging local recoverable errors. See EV_LOG for details.
271  * @see EV_LOG @hideinitializer @ingroup Logging
272  */
273 #define EV_ERROR  EV_LOG(omnetpp::LOGLEVEL_ERROR, nullptr)
274
275 /**
276  * @brief Pseudo-stream for logging warnings. See EV_LOG for details.
277  * @see EV_LOG @hideinitializer @ingroup Logging
278  */
279 #define EV_WARN   EV_LOG(omnetpp::LOGLEVEL_WARN, nullptr)
280
281 /**
282  * @brief Pseudo-stream for logging information with the default log level. See EV_LOG fo
283  * @see EV_LOG @hideinitializer @ingroup Logging
284  */
285 #define EV_INFO   EV_LOG(omnetpp::LOGLEVEL_INFO, nullptr)
286
287 /**
288  * @brief Pseudo-stream for logging low-level protocol-specific details. See EV_LOG for d
289  * @see EV_LOG @hideinitializer @ingroup Logging
290  */
291 #define EV_DETAIL EV_LOG(omnetpp::LOGLEVEL_DETAIL, nullptr)
292
293 /**
294  * @brief Pseudo-stream for logging state variables and other low-level information. See
295  * @see EV_LOG @hideinitializer @ingroup Logging
296  */
297 #define EV_DEBUG  EV_LOG(omnetpp::LOGLEVEL_DEBUG, nullptr)
298
```

```
299 /**
300  * @brief Pseudo-stream for logging control flow information (entering/exiting functions,
301  * @see EV_LOG @hideinitializer @ingroup Logging
302  */
303 #define EV_TRACE  EV_LOG(omnetpp::LOGLEVEL_TRACE, nullptr)
304
305 /**
306  * @brief Short for EV_INFO_C.
307  * @see EV_INFO_C @ingroup Logging
308  */
309 #define EV_C(category)        EV_INFO_C(category)
310
311 /**
312  * @brief Pseudo-stream for logging local fatal errors of a specific category. See EV_LOG
313  * @see EV_LOG @hideinitializer @ingroup Logging
314  */
315 #define EV_FATAL_C(category)  EV_LOG(omnetpp::LOGLEVEL_FATAL, category)
316
317 /**
318  * @brief Pseudo-stream for logging local recoverable errors of a specific category. See
319  * @see EV_LOG @hideinitializer @ingroup Logging
320  */
321 #define EV_ERROR_C(category)  EV_LOG(omnetpp::LOGLEVEL_ERROR, category)
322
323 /**
324  * @brief Pseudo-stream for logging warnings of a specific category. See EV_LOG for detai
325  * @see EV_LOG @hideinitializer @ingroup Logging
326  */
327 #define EV_WARN_C(category)   EV_LOG(omnetpp::LOGLEVEL_WARN, category)
328
329 /**
330  * @brief Pseudo-stream for logging information with the default log level of a specific
331  * @see EV_LOG @hideinitializer @ingroup Logging
332  */
333 #define EV_INFO_C(category)   EV_LOG(omnetpp::LOGLEVEL_INFO, category)
334
335 /**
336  * @brief Pseudo-stream for logging low-level protocol-specific details of a specific cat
337  * @see EV_LOG @hideinitializer @ingroup Logging
338  */
339 #define EV_DETAIL_C(category) EV_LOG(omnetpp::LOGLEVEL_DETAIL, category)
340
341 /**
342  * @brief Pseudo-stream for logging state variables and other low-level information of a
343  * @see EV_LOG @hideinitializer @ingroup Logging
344  */
345 #define EV_DEBUG_C(category)  EV_LOG(omnetpp::LOGLEVEL_DEBUG, category)
346
347 /**
```

```
348  * @brief Pseudo-stream for logging control flow information (entering/exiting functions,
349  * @see EV_LOG @hideinitializer @ingroup Logging
350  */
351 #define EV_TRACE_C(category)  EV_LOG(omnetpp::LOGLEVEL_TRACE, category)
352
353 /**
354  * @brief This class holds various data that is captured when a particular
355  * log statement executes. It also contains the text written to the log stream.
356  *
357  * @see cEnvir::log(cLogEntry*)
358  * @ingroup Internals
359  */
360 class SIM_API cLogEntry
361 {
362   public:
363     // log statement related
364     LogLevel logLevel;
365     const char *category;
366
367     // C++ source related (where the log statement appears)
368     const void *sourcePointer;
369     const cObject *sourceObject;
370     const cComponent *sourceComponent;
371     const char *sourceFile;
372     int sourceLine;
373     const char *sourceFunction;
374
375     // operating system related
376     clock_t userTime;
377
378     // the actual text of the log statement
379     const char *text;
380     int textLength;
381 };
382
383
384 //
385 // This class captures the context where the log statement appears.
386 // NOTE: This class is internal to the logging infrastructure.
387 //
388 class SIM_API cLogProxy
389 {
390   private:
391     // This class is used for buffering the text content to be able to send whole
392     // lines one by one to the active environment.
393     class LogBuffer : public std::basic_stringbuf<char> {
394       public:
395         LogBuffer() { }
396         bool isEmpty() { return pptr() == pbase(); }
```

```
397       protected:
398         virtual int sync() override;  // invokes getEnvir()->log() for each log line
399     };
400
401     // act likes /dev/null
402     class nullstream : public std::ostream {
403       public:
404         nullstream() : std::ostream(nullptr) {}  // results in rdbuf==0 and badbit==true
405     };
406
407   public:
408     static nullstream dummyStream; // EV evaluates to this when in express mode (getEnvir
409
410   private:
411     static LogBuffer buffer;  // underlying buffer that contains the text that has been w
412     static std::ostream stream;  // this singleton is used to avoid allocating a new stre
413     static cLogEntry currentEntry; // context of the current (last) log statement that ha
414     static LogLevel previousLogLevel; // log level of the previous log statement
415     static const char *previousCategory; // category of the previous log statement
416
417   private:
418     void fillEntry(LogLevel logLevel, const char *category, const char *sourceFile, int s
419
420   public:
421     cLogProxy(const void *sourcePointer, LogLevel logLevel, const char *category, const c
422     cLogProxy(const cObject *sourceObject, LogLevel logLevel, const char *category, const
423     cLogProxy(const cComponent *sourceComponent, LogLevel logLevel, const char *category,
424     ~cLogProxy();
425
426     std::ostream& getStream() { return stream; }
427     static void flushLastLine();
428 };
429
430 }  // namespace omnetpp
431
432 #endif
```

4.2 虚函数

4.2.1 initialize函数

4.2.2 handleMessage函数

4.2.3 refreshDisplay函数

4.2.4 finish函数

4.3 本章小结

 ； ；本章对OMNeT++中提供的相关类进行了描述和说明，阅读相关头文件是掌握OMNeT++仿真有效方法。

第 5 章

OMNeT++仿真技巧

       欢迎读者来到第五章的学习，本章打算从工程应用的角度，结合现有的仿真经验分享一些技巧，用套路二字来形容也不为过。   本章涉及的内容包括信道模型应用、节点分布相关、节点之间如何建立通信以及门向量的相关设置，同时也会涉及以上代码相关的说明，简而言之，本章采用情景分析的方法进行说明。也许你会发现本章好多内容可以在OMNeT++社区提供的Simulation Manual手册中发现，所以推荐读者后续再阅读Simulation Manual手册进行深度研究。

5.1 设计技巧

5.1.1 技巧一：信道模型很重要

       据说理想的运放可以摧毁整个地球，那么是不是理想的充电宝是不是充不满电，偏题了，那么理想的信道呢？当初初次使用OMNeT++时，遇到一个问题：

> 在节点之间传输消息的时候，如何加快消息的传输速度？当节点数量较大的时候，需要较快的实现消息传送的效果。

       有此疑问是在运行社区提供的相关工程时发现在他们的的仿真场景中，两个节点似乎可以同时发送消息出去，给人一种并行运行的感觉，让我不得不怀疑是不是需要调用并行接口才能达到这种效果，并且也发现他们的仿真程序运行时间特别小，换句话说就是接近现实的时间级，而我的仿真程序中两个节点传输一个消息都到秒级了，问题很大。   最后发现这个问题与信道模型有关，也与下一小节的send函数相关。在OMNeT++中仿真的时候，如果没有添加信道模型，消息在两个节点之间传输线就是理想的信道模型，这个仿真信道会影响什么呢？

仿真结果

仿真现象

       影响仿真结果好理解，仿真现象呢，那我们来看看仿真模型：

Listing 5.1: My

```
1 channel Channel extends DatarateChannel
```

```
2 {
3     delay = default(uniform(20ns, 100ns));
4     datarate = default(1000Mbps);
5 }
```

以上代码是一个简单的信道模型，将这个信道加入到传输线上将会有意想不到的效果。

5.1.2 技巧二：send函数有套路

       不知道读者有时候有没有感觉到send函数很麻烦，send函数用于两个模块之间的消息传输，但是当我们需要发送多条消息的时候，我们不能使用for循环直接就上，其主要原因就上我们使用send函数发送的消息还没有到达目的节点，此时我们不能使用send函数发送下一条消息，那么怎么办呢？这里有两种方案：

利用scheduleAt函数

Listing 5.2: My

```
1 void Node::handleMessage(cMessage* msg)
2 {
3     if(msg->isSelfMessage()){
4         if(msg->getKind()==SMSG_INIT){
5                 ...
6                 ...
7                 cMessage* cloudMsg = new cMessage("hello");
8                 cloudMsg->setKind(SMSG_INIT);//设置节点类型
9                 scheduleAt(simTime()+0.01,cloudMsg); //调度一个事件，发送
  消息给自己
10         }
11     }
12 }
```

       通过使用scheduleAt函数使仿真时间走动，完成上一个消息的完成，这里补充一点，如果读者想使用延时来等待消息传输完成是不可行的，因为使用这种方法仿真时间是不会走动的。例如下面一段代码：

Listing 5.3: My

```
1 time1 = simTime();
2 func();
3 time2 = simTime();
```

       在上面这段代码中我们的使用func函数想使时间走动，但是实验结果告诉我们： Stime1==time2S， 经过多次多个地方验证，发现在OMNeT++中如果不调用与仿真时间相关的函数，仿真时间是不会走动的，与上面的实验现象是一致的。因此为了实现仿真时间的走动我们可以采用上面scheduleAt函数自我调度一个时间然后再发送下一个消息。

一定要采用send函数呢？

       上述采用scheduleAt的方法太麻烦，需要new一个消息，然后还需要定义一个SMSG_INIT，另外无端增多handleMessage函数内容，这种方法的确不是特别简洁。这里再分享另一种方法：

Listing 5.4: My

```
cPacket *pkt = ...; // packet to be transmitted
cChannel *txChannel = gate("out")->getTransmissionChannel();
simtime_t txFinishTime = txChannel->getTransmissionFinishTime();
if (txFinishTime <= simTime())
{
    // channel free; send out packet immediately
    send(pkt, "out");
}
else
{
    // store packet and schedule timer; when the timer expires,
    // the packet should be removed from the queue and sent out
    txQueue.insert(pkt);
    scheduleAt(txFinishTime, endTxMsg);
}
```

       上面的代码用于通过out门发送一个pkt包，但是在传输前需要得到该门上传输的消息的完成时间，需要注意的是当txFinishTime为-1时，说明该门没有消息传输，可以直接发送，如果txFinishTime为一个大于0的值，说明有消息正在传输，需要等待。所以在判断时我们采用$txFinishTime <= simTime()$。   通过这种方式，我们可以在for循环中发送多个消息。但是对于有些需求不得不使用scheduleAt函数完成。

提一提sendDirect函数！

Listing 5.5: My

```
sendDirect(cMessage *msg, cModule *mod, int gateId)
sendDirect(cMessage *msg, cModule *mod, const char *gateName, int index=-1)
sendDirect(cMessage *msg, cGate *gate)

sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration,
           cModule *mod, int gateId)
sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration,
           cModule *mod, const char *gateName, int index=-1)
sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration,
           cGate *gate)
```

       对于其他send类似的函数都是有线的传输方式，需要我们将节点连接才能发送消息，那么如何实现无线的发送方式呢？这个也正是OMNeT++中wireless仿真程序中使用的函数，该函数的参数与其他send函数不同，它需要指定目的节点，以及目的节点的门。相关详细可以阅读INET库代码。   这里有一个问题，当采用前三个函数进行消息传输时，传输的效果为一个圆点，如图5-1所示：

图5 1 普通传输效果图

```
channel Channel extends DatarateChannel
{
    delay = default(uniform(20ns, 100ns));
```

```
4     datarate = default(2000Mbps);
5 }
```

       对于无线连接的sendDirect函数，要想达到相同的效果，就没有设置channel一说了，在使用sendDirect函数时，有三个重载函数包括有两个参数simtime_t propagationDelay/simtime_t duration，一个是传播延时时间和持续时间，通过设置这两个参数可以达到图5　2的效果。

图5　2 设置传输延迟和持续时间

5.1.3　技巧三：如何访问同一级的其他模块

       在设计网络拓扑时，我们有时需要在一个模块中直接访问同一级其他模块的相关参数，不再经过消息之间传输进行传输。这种接口在OMNeT++下也被提供了，如下一个代码示例：

```
1 cModule *parent = getParentModule();
2
3 // 取出父模块下的模块beBuffer
4 cModule *psubmodBE = parent->getSubmodule("beBuffer");
5 BEBuffer *pBEBuffer = check_and_cast<BEBuffer *>(psubmodBE);
6
7 cModule *psubmodRC = NULL;
8 RCBuffer *pRCBuffer = NULL;
9 // 取出父模块下的模块rcBuffer
10 psubmodRC = parent->getSubmodule("rcBuffer");
11 pRCBuffer = check_and_cast<RCBuffer *>(psubmodRC);
```

       上面的代码片段主要通过getParentModule和getSubmodule两个接口得到指向目的模块的指针，得到指针相当于我们拿到了这个目的模块的所有，需要注意的是这种方式的前提是目的模块是一个简单模块，需要与复合模块区分开，在OMNeT++中复合模块只有对应的.ned文件，其描述方式如下：

```
1 module Node{
2         parameters:
3         ...
4         gates:
5         ...
6 }
```

而简单模块有三个文件：.nde、.cc、.h，其.ned文件中描述方式如下：

```
simple Node{
parameters:
     ...
gates:
...
}
```

       因此对于没有.cc/.h文件的复合模块，在编写代码时就没有对应的C++类，因此使用上述方法就出现问题，无法事先知道指针类型，那么对于复合模块的访问，我们可以通过下面的代码实现：

```
// 得到当前父模块下的所以模块
for(cModule::SubmoduleIterator iter(getParentModule()); !iter.end(); iter++){
        string ES = string("ES");
        cModule *submodule = *iter;
        string ESnode = string(submodule->getFullName(),0,2);
        // 判断是否是节点ES
        if(ESnode == ES){
                // 访问父模块参数
                string realname = string(submodule->getFullName());
                EScpu += submodule->par("cpu").longValue();
                ESmem += submodule->par("mem").longValue();

                cout<<EScpu<<"␣"<<ESmem<<endl;
        }
        else{
                continue;
        }
}
```

       上面的代码段涉及到了OMNeT++下的SubmoduleIterator迭代器，该迭代器在较多的库中都有使用，比如：INET，当然这种方式可以经过简单的修改也可以对简单的模块进行访问。在上面的代码段中，getParentModule()指明是当前模块的父模块，该代码目的就是在当前简单模块中，得到同一父模块下的ES复合模块的cpu/mem两个参数。

5.1.4 技巧四：遍历所有模块

       在有些场景下，我们需要遍历所有节点，甚至是复合节点内部的模块，代码示例如下：

```
/*
 * 在所有节点中寻找一个ID 等于当前模块的号的模块headId
 */
void Node::doNext()
{
    cModule *parent = getParentModule();
    cModule *mod,*Head,*midmod;

    //网络中的所有节点都遍历一次，包括复合模块下的子模块
    for(int i=1;i<=cSimulation::getActiveSimulation()->getLastComponentId();i++){
        int number_of_Bees = cSimulation::getActiveSimulation()->getLastComponentId();
        cSimulation *simobj = cSimulation::getActiveSimulation();
        //这里需要优化
        mod = cSimulation::getActiveSimulation()->getModule(i);
        if(strcmp(mod->getName(),"CenController") == 0){
            //如果遍历到一个模块名为的节点CenController
            continue;
        }
        else{
```

```
            int j=0;
            while(1){
                string modname = cSimulation::getActiveSimulation()->getModule(i)->getNam
                midmod=cSimulation::getActiveSimulation()->getModule(i);
                Head=midmod->getSubmodule(this->clustername.c_str(),j)->getSubmodule("Wir
                if(((Node*)Head)->myId == this->headId){
                    //找到簇头节点退出,循环while
                    break;
                }
                j++;
            }
            // 在到满足条件的节点，开始执行相关操作Head
            ...
            ...
            ...
            break;
        }
    }
}
```

       在上面的代码段中，可能有些诸如“Wireless”相关的过程与我实验源代码本身功能相关，本例只提供一种可参考的代码，具体运用于读者自己的项目中还需要做部分修改。为了让读者更快的掌握这种方法，下面就代码段中的重要接口做一个简单的分析：

```
for(int i=1;i<=cSimulation::getActiveSimulation()->getLastComponentId();i++)
```

       这一句for循环遍历当前网络场景中的模块，只遍历仿真场景中的节点，不包括节点内部的模块，下面结合一个网络拓扑文件说明：

```
network simplenet
{
    parameters:
        ...
        ...
    submodules:

        node1[x]: typeA {
            parameters:
                ...
        }

        node2[y]: typeB{
            parameters:
                ...
        }

        node3[z]: typeD {
            parameters:
                ...
        }
```

```
22
23      connections allowunconnected:
24                  ...
25                  ...
26  }
```

对于上述网络拓扑，使用上面的for循环只能遍历node1/node2/node3，对于它们内部的子模块不在其内。当最终需要寻找的模块是其中一个的子模块，需要先遍历父模块，然后使用getSubmodule函数遍历子模块。

```
midmod = cSimulation::getActiveSimulation()->getModule(i)
```

       紧接着上面的for循环，得到第i个模块的地址，如果该模块在网络中描述是用向量的方式需要使用：

$$getSubmodule(\ "node_name"\ ,j)$$

即可得到node_name[j]所代表的模块。

```
getSubmodule( "modname" )
```

       该接口似乎使用频率较高，如何得到一个复合模块的指针，即可通过该接口得到内部子模块的指针，然后访问相关数据。

5.1.5 技巧五：如何得到某一个模块引用的ned路径

       为什么需要在一个程序中得到该“.ned”引用的路径呢？因为在OMNeT++中，我们在设计一个复合模块的内部结构时，可以直接采用图形的方式编辑，相当于我们可以直接拖动设计好的简单模块到复合模块中，而有些简单模块在不同的复合模块中其功能还有所不同，因此在为该简单模块编写.cc文件时，我们需要检测一下当前本模块在什么模块下使用的，比如是在端系统还是交换机。得到一个模块的引用路径，其实就是一个接口函数的事，如下代码段：

```
1  cModule *parent = getParentModule();
2  const char *name = parent->getNedTypeName();
3
4  if (strcmp(name, "SimpleNetwork.Node.SimpleNode") == 0){
5      cGate *outgate = gate("line$o");
6      cChannel *chan = outgate->findTransmissionChannel();
7      linkspeed = chan->getNominalDatarate();
8
9  }
10 else if (strcmp(name, "SimpleNetwork.Switch.SwitchPort") == 0){
11     //int id = parent->findGate("line$o");
12     cGate *outgate = parent->gate("line$o");
13     cChannel *chan = outgate->findTransmissionChannel();
14     linkspeed = chan->getNominalDatarate();
15 }
```

       该接口函数便是getNedTypeName，得到完整的路径后，使用c库函数strcmp进行判断即可。

5.1.6 技巧六：使用cTopology类遍历拓扑初始化路由表

       这是个好东西，其实在OMNeT++中其实提供的大量的接口函数，只是在不知道的前提下写相似的功能函数比较麻烦，这个接口函数完美解决我们寻找路由的门问题，在使用send函数传输消息的时候只要知道我们传输的目的节点便可，直接利用一个路由表即可，代码示例如下：

```
/*
 * 探测交换机网络的拓扑
 */
void Router::TopoFind()
{
    cTopology *topo = new cTopology("topo");

    topo->extractByNedTypeName(cStringTokenizer("SimpleNetwork.Node.SimpleNode␣SimpleNetw

    EV << "cTopology␣found␣" << topo->getNumNodes() << "␣nodes\n";

    //得到表示本节点的对象
    cTopology::Node *thisNode = topo->getNodeFor(getParentModule());

    // find and store next hops
    for (int i = 0; i < topo->getNumNodes(); i++){
        if (topo->getNode(i) == thisNode)
            continue; // skip ourselves
        //采用迪杰斯特拉算法计算到节点的最短距离i
        topo->calculateUnweightedSingleShortestPathsTo(topo->getNode(i));
        //本节点与外界连接的通道
        if (thisNode->getNumPaths() == 0)
            continue; // not connected

        cGate *parentModuleGate = thisNode->getPath(0)->getLocalGate();
        int gateIndex = parentModuleGate->getIndex();
        int address = topo->getNode(i)->getModule()->par("address");
        rtable[address] = gateIndex;
        EV << "␣␣towards␣address␣" << address << "␣gateIndex␣is␣" << gateIndex
<< endl;
    }
    delete topo;
}
```

该函数有三个比较重要的步骤：

[1] extractByNedTypeName

       为了得到一个路由表，我们需要指明需要遍历的节点类型。该函数便是指明遍历哪些节点。

[2] calculateUnweightedSingleShortestPathsTo

       得到路由表也涉及到路由算法的选择，在ctopology.h文件中有以下两个路由算法可供选择：

```
/** @name Algorithms to find shortest paths. */
/*
* To be implemented:
*    -  void unweightedMultiShortestPathsTo(Node *target);
*    -  void weightedMultiShortestPathsTo(Node *target);
*/

//@{

/**
* Apply the Dijkstra algorithm to find all shortest paths to the given
* graph node. The paths found can be extracted via Node's methods.
*/
virtual void calculateUnweightedSingleShortestPathsTo(Node *target);

/**
* Apply the Dijkstra algorithm to find all shortest paths to the given
* graph node. The paths found can be extracted via Node's methods.
* Uses weights in nodes and links.
*/
virtual void calculateWeightedSingleShortestPathsTo(Node *target);
```

代入参数就是目的节点地址，其他内容读者可自行探索。

[3] topo->getNode(i)->getModule()->par(“address”);

       这里比较重要的便是“address”形参，在以太网中相当于IP地址，最终得到的rtable[]表其索引就是目的地址的address，索引对应的值就是该节点的门，从该门出去到目的节点路径最短。

5.1.7 技巧七：如何使用OpenSceneGraph

       其实在OMNeT++中是可以直接使用OpenSceneGraph的，可怜的我尝试了安装了一下午，才知道OMNet++已经支持OpenSceneGraph了，以后补充这一点可以看：

omnetpp-5.2/doc/manual/index.html#sec:graphics:opp-api-for-osg

        samples里已经有支持三维显示的仿真程序了，读者可自行运行看。

5.1.8 技巧八：如何多次利用同一个msg

       在OMNeT++中，凡是使用scheduleAt调度的消息属于Self-Messages，其作用是用在模块本身调度事件使用的。有时需要利用同一个msg，但是中间必须使用cancelEvent函数取消掉上次，如下片段：

```
//cMessage *msg
if (msg->isScheduled())
    cancelEvent(msg);
scheduleAt(simTime() + delay, msg);
```

       该代码段没有什么特别大的功能，主要是重复利用已经定义好的msg变量。

5.1.9　技巧九：initialize函数的不同

       在每一个简单模块对应的.cc/.h文件中会有一个initialize函数，其功能是在仿真程序开始执行前将会执行的函数，与类的构造函数不同，引出的问题就是：

> 如果在其他成员函数中给一个指针数组成员赋值，当离开这个函数后，该指针数组值将会回到原来的值，该函数赋的值没有任何作用，但是如果在initialize函数中初始化这个指针数组，将会达到我们想要的结果。

5.1.10　技巧十：如何从仿真场景读取节点坐标

       也许作者的用词不明，这里的仿真场景指的是运行仿真后出现的仿真界面。必须提到的是这个OMNeT++的仿真场景，节点在该场景上的位置，不一定是它的属性里边的地址，它们可以不同，感觉似乎是OMNeT++开发者提供的缺口，不知这个是好还是坏，但是好消息就是这些开发者提供了读取场景上节点属性的坐标和在程序中设置该坐标（目的就是让这个显示坐标更新），简而言之，你的节点坐标更新需要你自己在程序中完成，OMNeT++不会自动帮你完成。程序5.2.9　1是关于读取坐标和更新场景坐标的显示的代码段：

```
    1 程序
2 5.2.9-1
3 // 按照最开始的网络拓扑（按圆形分布），得到每一个节点的坐标
4 this->xpos = atof(parentdispStr.getTagArg("p", 0));
5 this->ypos = atof(parentdispStr.getTagArg("p", 1));
6
7 coord_X.setDoubleValue(this->xpos); //将仿真界面上的改变xpos
8 coord_Y.setDoubleValue(this->ypos); //将仿真界面上的改变ypos
```

需要再次提示的是这个坐标读取的是显示的节点的坐标，与节点在仿真场景上显示的位置可能没有关系。

5.1.11　技巧十一：如何调用INET中的类

       有时候在仿真程序中有种需求：

> 需要在一个仿真程序中调用其他库中的函数，例如需要使用INET中相关类，那这时候的逻辑是什么？

       与在某一个工程下需要import INET中的NED模型，我们需要在工程的属性中Project References中勾上我们需要import的库，然后在工程的ned文件中添加ned模型路径。同时当我们设置了工程Project References，当编译该工程时，将会链接Project References中勾上的工程编译生成的库文件，其中涉及以下编译设置：

```
1 // macros needed for building Windows DLLs
2 #if defined(_WIN32)
3 #  define OPP_DLLEXPORT  __declspec(dllexport)
4 #  define OPP_DLLIMPORT  __declspec(dllimport)
```

```
#else
#  define OPP_DLLIMPORT
#  define OPP_DLLEXPORT
#endif
```

       以上摘取自INET开源库中platdefs.h文件，其中比较重要的是当编译INET库时，编译默认选项会使用__declspec(dllexport)，当另一个仿真工程（使用了INET库中的类）编译时，将会以__declspec(dllimport)，因此工程不需要设置其他编译选项，但是需要将诸如INET编译生成的.dll或者.a拷贝一份到该工程目录下。

注意：

       如果以上关系都满足了，再出现在链接工程的错误可能其其他导致的。

5.2 本章小结

      OMNeT++仿真内核提供的丰富的仿真接口，使用OMNeT++进行仿真，在掌握一定的C++编程方法以后，阅读   OMNeT++相关类的描述可能有意外的收获，找到合适的接口进行仿真。

第 6 章

数据统计与仿真分析

       OMNeT++有诸多工具对网络代码中统计的标量和矢量进行数据分析。   本章以一个AFDX网络的RC和BE消息的仿真结果为前提，依赖OMNeT++自带的工具集对这些结果进行分析，重点主要放在如何设置需要统计的标量和矢量？如何对最后的仿真结果进行操作得到我们想要的散点图、直方图等其他便于分析的数据图形。

6.1 仿真结果有哪些

       先备注一下，每次运行完仿真后，将会产生三个文件：.sca .vci .vec，点击.vec文件将会生成.anf文件，这个.anf文件当我们下一次重新运行仿真程序的时候，会更新，不需要删除后执行仿真程序。

6.2 仿真结果的获取

       在平常各种各样的仿真实验中，首先我们需要去获取所需的结果信息。在OMNeT++中有以下几种常用的获取仿真结果的方式，这里同时简单描述一下它们的用法。

cLongHistogram：记录数据然后实现等距直方图 定义：cLongHistogram hopCountStats；我们可以对名称进行设置，如hopCountStats.setName（“hopcountStats”）；设置上限值：hopCountStats.setRangeAutoUpper（“0,10,1.5”）；记录数据：hopCountStats.collect(hopcount)；一些其他的属性如getMin()、getMax()、getMean()以及getStddev()，不做赘述。

cOutVector：获取输出向量 定义：cOutVector hopCountVector；同样可以人为地对名称进行设置，比如hopCountVector.setName（“Hopcount”）；记录数据：hopCountVector.record(hopcount)；其中，record表示记录数据。缺少这一语句的话，不会有任何的数据输出。

recordScalar 输出程序中某个标量的值，直接调用即可。即 recordScalar（“string 输出名称”，输出变量名)；仿真之后的result文件中会有以“string 输出名称”命名的文件。“输出变量名”为我们要输出查看的变量。recordScalar较为简单，一般在Finish()函数中使用。

6.3　仿真结果分析

       每次仿真的结果都会存储在project下面的result文件夹中。 cOutVector: \result\xxx.vec cLongHistogram: \result\xxx.sca

图6　1 doc目录

       如图所示，Project Explorer中选中的文件就是我们的仿真结果。之后双击打开就可以查看里面的内容，这里我选择打开了vec文件。

图6　2 doc目录

然后会让我们建立新的分析文件。点击finish即可

图6　3 doc目录

点击打开Data栏中vec下属的记录。Tictoc15网络中有6个节点，可以看到仿真对它们全部进行了记录

图6　4 doc目录

图6　5 doc目录

其实，观察选项卡就可以发现，这里我们就可以查看所有的结果了。由于笔者打开的是vec文件，所以只有输出向量。

图6　6 doc目录

双击打开想查看的一行，如下所示：

图6　7 doc目录

接下来我们打开sca文件查看直方图：

图6　8 doc目录

图6　9 doc目录

可以看到，如前文所述，文件名是匹配的

在Histogram栏中选中一条，并打开：

图6　10 doc目录

另外，我们也可以直接取出数据，方法如下：

图5　11 doc目录

6.4　事件日志文件的使用

       事件日志文件（EventLog）所记录的内容包括用户仿真过程中各个模块发送的消息细节以及提示发送和消息接收的细节。在Tkenv界面进行仿真前，点击“Enable recording on/off”按钮，即可对仿真过程中的事件进行记录。

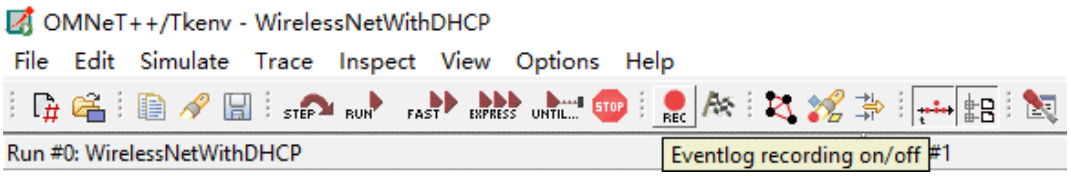

图 6.1: avatar

       默认情况下，相应工程的result文件夹中会出现一个后缀为“.elog”的文件，即我们本次仿真记录所得的时间日志文件。这里需要特别注意的是，记录的数据数量会直接决定elog文件的大小，不仅会影响仿真的速度，还可能在仿真结束后，omnetpp无法打开过大的日志文件，导致闪退，严重时甚至出现过黑屏等情况。因此建议在使用时不要记录过长的时间或过多无用的内容。

6.4.1 序列图

       打开elog文件后，里面的内容会以序列图的形式来展现，如下图所示：

图6 16 常用符号

       序列图可分为三个部分：上沿、主区域和下沿。其中，上下沿显示的是仿真时间轴。主区域则是显示各个模块名称和周线、时间与消息的发送。下面是常用符号的图例：

图6 16 常用符号

6.4.2 事件日志表

       事件日志表的事件记录分为三栏，依次是事件编号、仿真时间和事件的具体细节。 善用过滤器来减少无用内容的显示对提高工作效率很有帮助，行过滤器可以过滤特定类型的显示行。同时，事件日志表支持导航历史纪录，每个用户停留超过三秒的位置都会被记录下来作为临时数据。

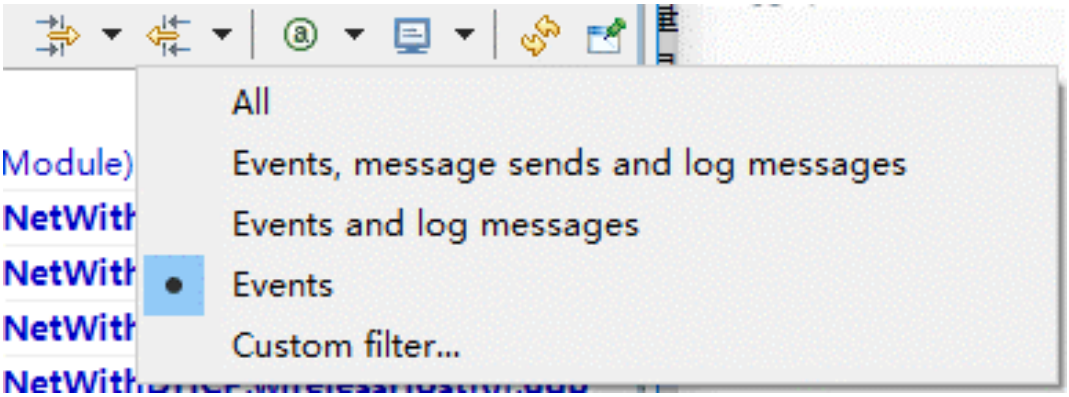

图 6.2: avatar

6.4.3 个人体会

       首先第一点，一定不要让elog文件的体积过大，因为这很可能导致处理过程中的闪退。 然后，过滤器是我最常用的功能。除了行过滤器外，序列图和事件日志表都支持同一个Filter。

如图所示，一般包括： 范围过滤器：过滤掉elog中的起始和结束事件，有助于减少计算时间。 模块过滤器：用户可以指定特定的模块，非指定模块的事件会被全部过滤。当我们倾向于研究一个或几个特定模块时，这非常有用。 消息过滤器：最复杂的一个过滤器。需要根据消息的C++名称、消息名称、消息id以及匹配表达式等进行选择。 因果过滤器：通过指定特定的事件并对其愿意和结果进行过滤。

       同时，omnet++对filter结果的计算和显示会耗费大量的时间，一定要指定适当的范围。具体根据什么来设置范围，就要以各位各自的使用情况作为标准了。

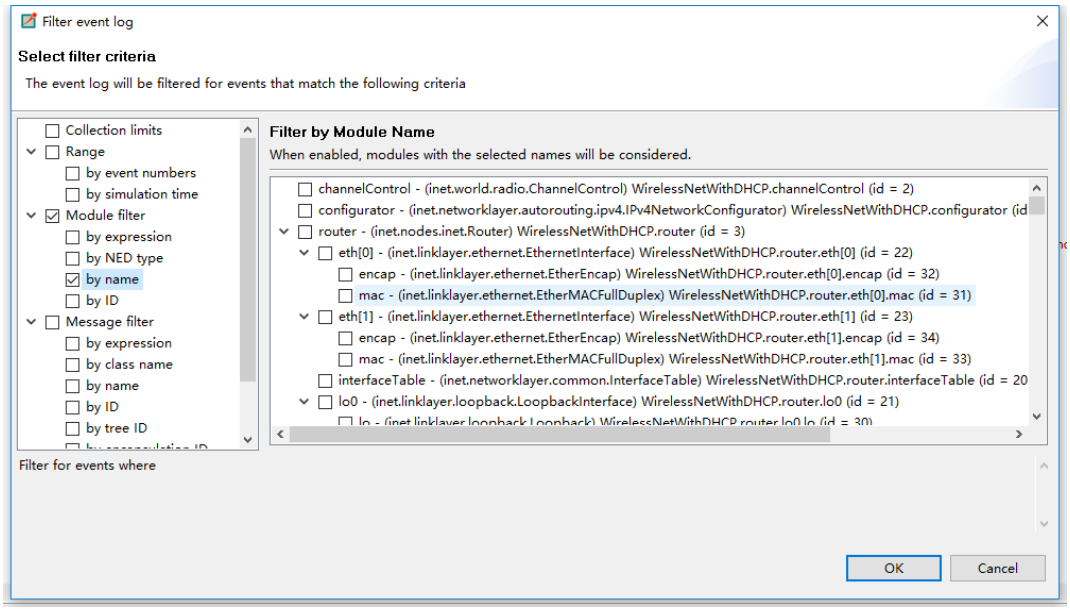
Filter event log
Select filter criteria
The event log will be filtered for events that match the following criteria
Collection limits
Range
by event numbers
by simulation time
Module filter
by expression
by NED type
by name
by ID
Message filter
by expression
by class name
by name
by ID
by tree ID
Filter by Module Name
When enabled, modules with the selected names will be considered.
channelControl - (inet.world.radio.ChannelControl) WirelessNetWithDHCP.channelControl (id = 2)
configurator - (inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator) WirelessNetWithDHCP.configurator (id
router - (inet.nodes.inet.Router) WirelessNetWithDHCP.router (id = 3)
eth[0] - (inet.linklayer.ethernet.EthernetInterface) WirelessNetWithDHCP.router.eth[0] (id = 22)
encap - (inet.linklayer.ethernet.EtherEncap) WirelessNetWithDHCP.router.eth[0].encap (id = 32)
mac - (inet.linklayer.ethernet.EtherMACFullDuplex) WirelessNetWithDHCP.router.eth[0].mac (id = 31)
eth[1] - (inet.linklayer.ethernet.EthernetInterface) WirelessNetWithDHCP.router.eth[1] (id = 23)
encap - (inet.linklayer.ethernet.EtherEncap) WirelessNetWithDHCP.router.eth[1].encap (id = 34)
mac - (inet.linklayer.ethernet.EtherMACFullDuplex) WirelessNetWithDHCP.router.eth[1].mac (id = 33)
interfaceTable - (inet.networklayer.common.InterfaceTable) WirelessNetWithDHCP.router.interfaceTable (id = 20
lo0 - (inet.linklayer.loopback.LoopbackInterface) WirelessNetWithDHCP.router.lo0 (id = 21)
Filter for events where
OK
Cancel

图 6.3: avatar

第 7 章

错误记录

       在我刚开始使用OMNeT++进行实验的时候，总是会出现一些稀奇古怪的错误，每一个错误都花了我大量的时间，所以在最后一章中总结出来，也许你正好用上。

7.1 在模块加入移动模块之后，仿真出现nan错误

7.1.1 问题描述：

对于某些工程加入移动模块以后，编译过程没有出错，但是当执行仿真程序的时候，出现一些nan错误提示。 需要在仿真配置文件加入：

```
**.constraintAreaMinX = 0m
**.constraintAreaMinY = 0m
**.constraintAreaMinZ = 0m
**.constraintAreaMaxX = 5000m
**.constraintAreaMaxY = 5000m
**.constraintAreaMaxZ = 0m
```

移动模块配置：

```
**.UAV[*].mobilityType = "MassMobility"
**.UAV[*].mobility.initFromDisplayString = true
**.UAV[*].mobility.changeInterval = truncnormal(2s, 0.5s)
**.UAV[*].mobility.changeAngleBy = normal(0deg, 30deg)
**.UAV[*].mobility.speed = truncnormal(250mps, 20mps)
**.UAV[*].mobility.updateInterval = 100ms
```

7.2 工程的例如cModule之类的类不能高亮显示？

7.2.1 问题描述：

原工程是可以高亮显示的，但是由于我在备份这个程序的时候可能方式不对，我采用的是在文件资源管理器的窗口复制原工程文件夹，没有在软件的窗口进行rename，可能是这个原因造成的。

7.2.2 解决办法：

在软件的窗口，对工程进行rename就行，编译一次，cMdoule等等关键词就可以高亮了。

第 8 章

移动自组网中的路由协议

  移动自组网是由一群可以移动的、兼具终端及路由功能的移动节点，通过无线链路形成的无中心、多跳、临时性自治系统。[1] 本章主要介绍移动自组网中的常见路由协议。

8.1 DSDV

8.2 DSR

8.3 ADOV

8.4 参考文献

[1] 潘宇航. 基于 QoS 感知的移动自组织网络路由协议设计与仿真研究. MS thesis. 西南交通大学，2014.

第 9 章

TODO 待完善

[1] 如何加快节点间消息的传输?

[2] 在一个复合模块下，如何访问同一级的其他模块?

[3] 如何得到某一个模块引用的ned路径?

[4] 如何使用cTopology类遍历网络的拓扑来初始化路由表?

[5] 如何在omnet上使用OpenSceneGraph

[6] 如何从仿真场景读取节点的坐标

[7] 使用sendDirect()函数

[8] 复合模块初始化时，先初始化节点的顺序

[9] 在initialize()中初始化类成员数组与在其他函数中的不同

附录 A

网络性能指标

       在前面几章中总是会涉及到相关网络中的术语，可能不利于读者理解，因此在最后加上附录，完善相关内容。

A.1 速率

       主要是指主机在数字通信上传送数据的速率，称为额定速率或者标称速率。 定义：单位时间（秒）传输信息量（比特）通常情况下用bit符号表示。 常用的单位：b/s，kb/s，Mb/s，Gb/s，他们之间的换算关系为：

$$1Gb/s = 10^6 Mb/s = 10^9 b/s$$

A.2 吞吐率

       网络通信中发送端与接收端之间传送数据的速率。吞吐量受到网络的带宽或网络的额定速率的限制。

即时吞吐率：给定时刻的速率；

平均吞吐率：一段时间的平均速率；

瓶颈链路：端到端路劲上，带宽最小的链路。

       吞吐率决定瓶颈链路的带宽，吞吐率为发送端的发送速率， 发送端发送数据的速率大于瓶颈链路，则吞吐率为瓶颈链路的带宽。

A.3 延迟

       数据在传输过程中所消耗的时间即称为延迟，在分组交换网络中，延迟总共包含有四种：即节点处理延迟、排队延迟、传输延迟、传播延迟。

[1] 节点处理延迟 节点（路由器）在处理数据时进行差错检测、确定链路输出等活动小号的时间，通常时间延迟很小，一般为毫秒级甚至更低。

[2] 排队延迟 需要传输的数据在节点中等待输出链路可用所花的时间，往往取决于节点（路由器）的拥塞程度。

[3] 传输延迟 节点将正在传输的分组数据发送到输出链路所用的时间，取决于L:分组长度（bit）和R：链路带宽，延迟d=L/R；

[4] 传播延迟 信号在信道中传播所用的时间，取决于信道长度d和电磁波信号在信道上的传播速度，延迟Pd=d/s

[5] 总时延 将上述的四个时延时间进行相加，既可以得到。

A.4　丢包率

 ； ； ； 丢包数/已发分组的总数，分组交换网络丢包的原因主要是节点的队列缓存容量有限，当分组到达时如果节点中的分组队列已满，则该分组会被丢弃，即称为丢包。

A.5　带宽

信号的带宽 信号的带宽是指信号所包含的各种不同频率成分所占据的频率范围。

计算机网络的带宽 在单位时间内从网络中的某一点到另一点所能通过的最高数据率，即网络可通过的最高数据率，即每秒传输多少比特，单位是“比特每秒”，或b/s(bit/s)。

A.6　时延带宽积

 ； ； ； 将两个网络性能的两个度量，传播时延和宽带相乘，就等到另外一个度量：传播时延带宽积(单位：bit；)。

$$\text{时延带宽积} = \text{传播时延} * \text{带宽}$$

 ； ； ； 下面是一个简单的例子：传播时延为20ms，带宽为10Mb/s，则时延带宽积 = 20 × 10 × 10^3 /1000 = 2 × 10^5 bit。这就表示，若发送端连续发送数据，则在发送的第一个比特即将达到终点时，发送端就已经发送了20万个比特，而这20万个bit都在链路上向前移动。

A.7　信道利用率

 ； ； ； 信道利用率是指信道有百分之几的时间是被占用的比率，其计算公式如下：

$$S = \text{发送时间}T/（\text{传播时延}t + \text{碰撞等待时间}2n * t + \text{发送时间}）$$

 ； ； ； 上述提到的发送时间T是指：T=帧长数/发送速率。 这里有一个简单例子加以说明，假设信道的长度为10km，往返传输时延为10ms,传输数据长度为2048bit，发送端的发送速率为1Mbps，在其他时延（碰撞等待时间）忽略的情况下，求信道利用率。

发送时间：2048/(1x1000x1000)=2.048ms

信道利用率：2.048/(2.048+10)=17%

A.8 网络利用率

 ； ； ； 网络利用率是指全网络的信道利用率的加权平均值。

A.9 往返时间RTT

 ； ； ； 表示从发送方发送数据开始，到发送方收到来自接收方的确认，总共经历的时间。 其与所发送的分组长度有关。发送很长的数据块的往返时间，应当比发送很短的数据块往返时间要多。