OMNeT++ 学习笔记

前言

回馈开源

0.1 为什么写这本书

omnetppp-zh.pdf 记录了我在设计无人机蜂群网络仿真过程中，从初学 OMNeT++ 软件到能灵活使用各种接口所遇到的各种问题，苦于当初无处找到详细的 OMNeT++ 工程开发资料，尤其是针对实际功能实现的代码说明资料基本没有。我在阅读大量的网络仿真程序后，慢慢的对这个软件的各种接口和配置才熟悉，同时也从官方提供的手册中提取出较为常用的接口进行说明，最后将我熟悉的套路总结成文档回馈开源。由于我水平有限，难免会存在理解错误的地方，欢迎读者发邮件指出，如果您有其他宝贵的建议，也欢迎发邮件交流，希望这个文档能帮助更多的开发者。

0.2 本书结构

本书主要作为 OMNeT++ 仿真平台快速入门指导书，书中大量来源于 OMNeT++ 自带参考。从第一章到第五章，主要编写在 OMNeT++ 上设计仿真程序，第七章主要编写 OMNeT++ 仿真数据统计方法，后续可增添我在 OMNeT++ 平台下编写的相关辅助工具。各章内容如下：

第一章：OMNeT++ 简介

第二章：OMNeT++ 安装以及相关仿真库的安装

第三章：

第四章：

第五章：

第六章：

第七章：

0.3 适合读者

本书适合所有正在看的人，否则你也不会读到这里。作为作者来说，我认为此书尤其适合以下读者：

0.4 如何使用本书

0.5 如何写作本书的

本书采用排版简单的 markdown 编写的，中途遇到转换成 pdf 的问题，几度停下。本书记录的一些我对 OMNeT++ 使用的心得，总结了部分相关的接口，但都是只是一些皮毛。

0.5.1 封面

封面设置改了好几版，最开始打算使用卡通角色，最后感觉不好看，还是采用了一个简单的封面。封面中篆刻乃我近期仿刻的藏书印-“微风闲坐古松”，后期若有必要再对封面进行修改。

版本变化

下面是每次的版本变化清单，你也可以自己在 Github 上查看。

0.6 2018 年 2 月 22 日-2 月 25 日

一个比较完备的发行版，80 页左右。

加强了书的格式，解决了中文字体显示的问题，加上了色彩，加上了图书推荐页。

加强了书的格式，前言和致谢挪到了目录前，附录区别于正常章节。

补全了 Git。

致谢

感谢 Kmtalexwang，Stephenhua 对 omnetppp_zh.pdf 的 LaTeX 排版的维护，以及 Ericsyoung 对 chapter_6.md 仿真结果分析的添加。

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、OSPF、AODV、DSDV、DSR 等；

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：

802.11a

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/

其他更多的 OMNeT++ 开源仿真模型可浏览以下网站：
https://omnetpp.org/download/models-and-tools

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，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
```

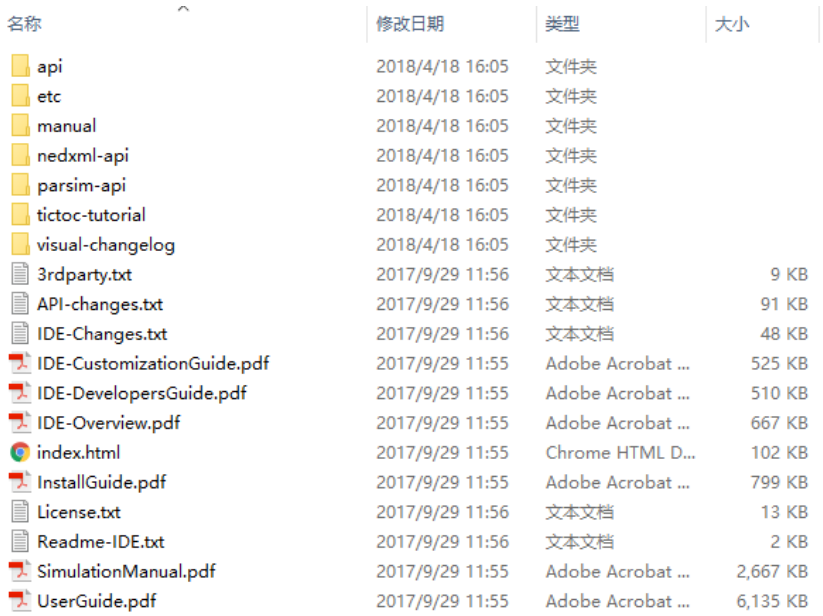
在 PC 机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

```
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）。

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：



名称	修改日期	类型	大小
api	2018/4/18 16:05	文件夹	
etc	2018/4/18 16:05	文件夹	
manual	2018/4/18 16:05	文件夹	
nedxml-api	2018/4/18 16:05	文件夹	
parsim-api	2018/4/18 16:05	文件夹	
tictoc-tutorial	2018/4/18 16:05	文件夹	
visual-changelog	2018/4/18 16:05	文件夹	
3rdparty.txt	2017/9/29 11:56	文本文档	9 KB
API-changes.txt	2017/9/29 11:56	文本文档	91 KB
IDE-Changes.txt	2017/9/29 11:56	文本文档	48 KB
IDE-CustomizationGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	525 KB
IDE-DevelopersGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	510 KB
IDE-Overview.pdf	2017/9/29 11:55	Adobe Acrobat ...	667 KB
index.html	2017/9/29 11:55	Chrome HTML D...	102 KB
InstallGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	799 KB
License.txt	2017/9/29 11:56	文本文档	13 KB
Readme-IDE.txt	2017/9/29 11:56	文本文档	2 KB
SimulationManual.pdf	2017/9/29 11:55	Adobe Acrobat ...	2,667 KB
UserGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	6,135 KB

图　2.1: 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：**/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，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 INET 库，需要到 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**，内容如下：

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

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

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

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

morekeywords

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。

```
1
2  If you are using the IDE:
3  ----------------------------
4  3. Open the OMNeT++ IDE and choose the workspace where you have extracted the
       inet directory.
5     The extracted directory must be a subdirectory of the workspace dir.
6
7  4. Import the project using: File | Import | General | Existing projects into
       Workspace.
8     Then select the workspace dir as the root directory, and be sure NOT to check
       the
9     "Copy projects into workspace" box. Click Finish.
10
11 5. Open the project (if already not open) and wait until the indexer finishes.
12    Now you can build the project by pressing CTRL-B (Project | Build all)
13
14 6. To run an example from the IDE open the example's directory in the Project
       Explorer view,
15    find the corresponding omnetpp.ini file. Right click on it and select Run As /
        Simulation.
16    This should create a Launch Configuration for this example.
17
18 If the build was successful, you may try running the demo simulations.
19 Change into examples/ and type "./rundemo".
```

morekeywords

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 INETMANET

2.4.1 INETMANET 简介

与 INET 库相似，但其之上加入了 MANET 相关仿真模型，若对自组织网进行仿真，推荐使用 INETMANET 库。

2.4.2 INETMANET 安装

在安装 INETMANET 库前，也许你可以在 **OMNeT++ 中，将 INET** 工程关掉，接下来，你可以按照以下步骤进行：

```
#!bin/bash
git clone git@github.com:aarizaq/inetmanet-3.x.git
git submodule update --init
```

morekeywords

以上得到完整 INETMANET 库文件并非唯一的方法，作者尝试过直接下载 inetmanet 包，然后单独下载 showcases 和 tutorials 两个文件与 inetmanet 包进行合并，也可以得到完整的 inetmanet 源文件。到此，INETMANET 安装包已经准备就绪，后续你可以参照 INET 安装步骤进行安装。

2.5 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。

2.6 常规使用

2.6.1 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 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：

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。

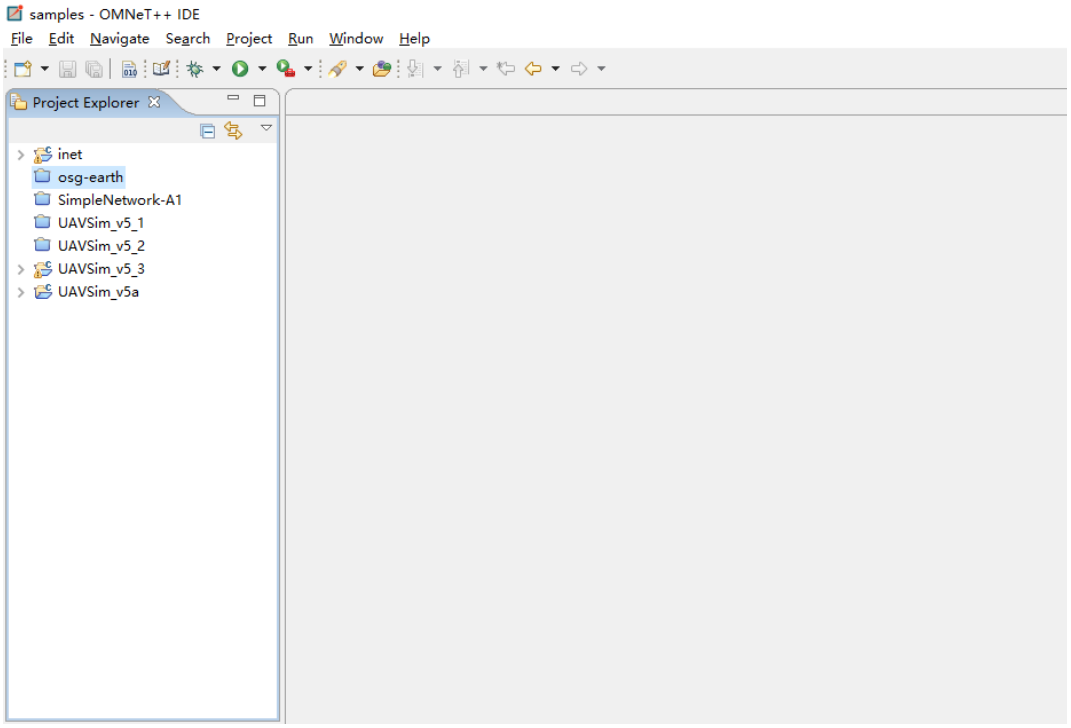


图　2.2: IDE 视图

2.6.2　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：

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

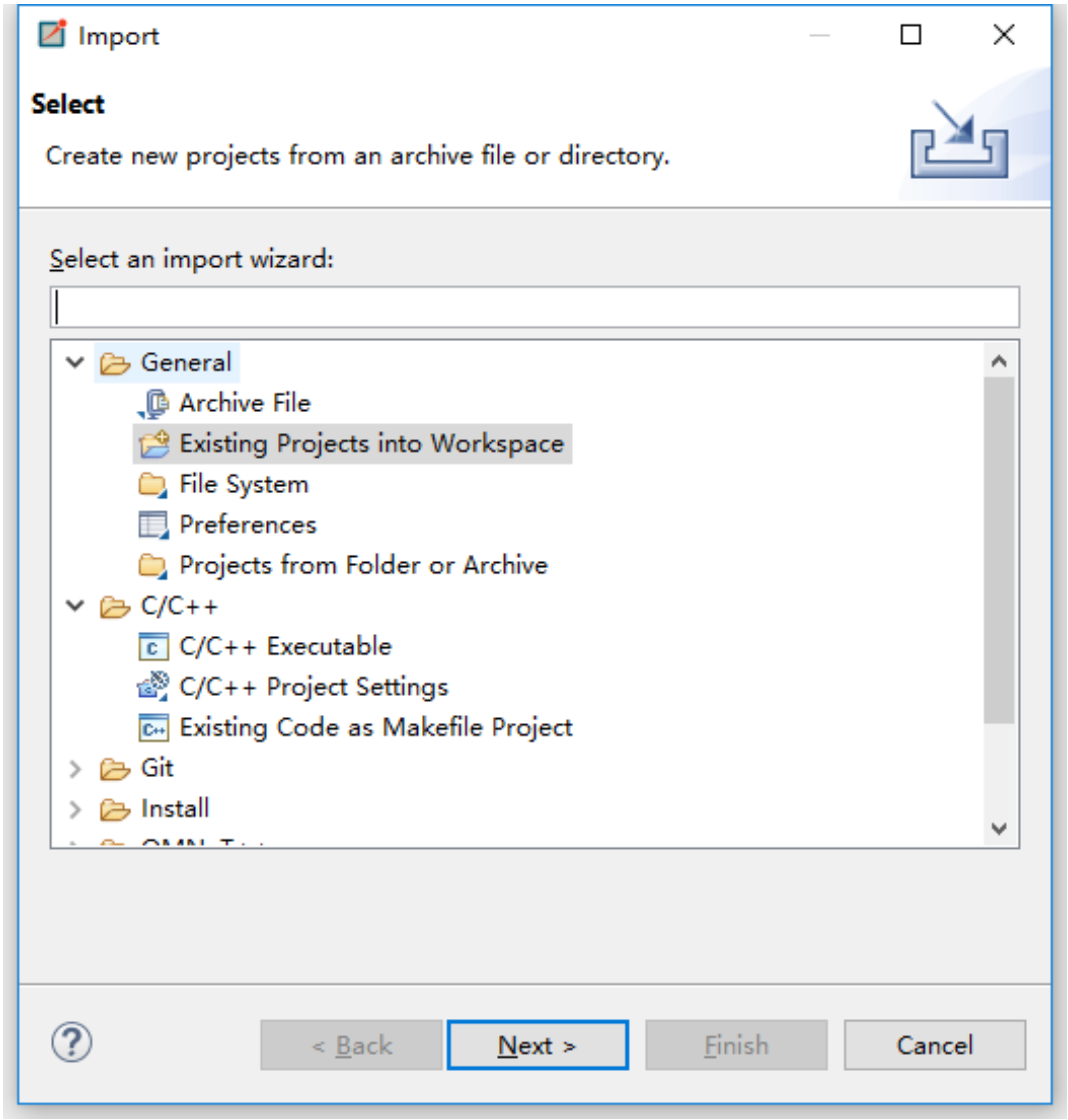


图 2.3: 点击 Import 后视图

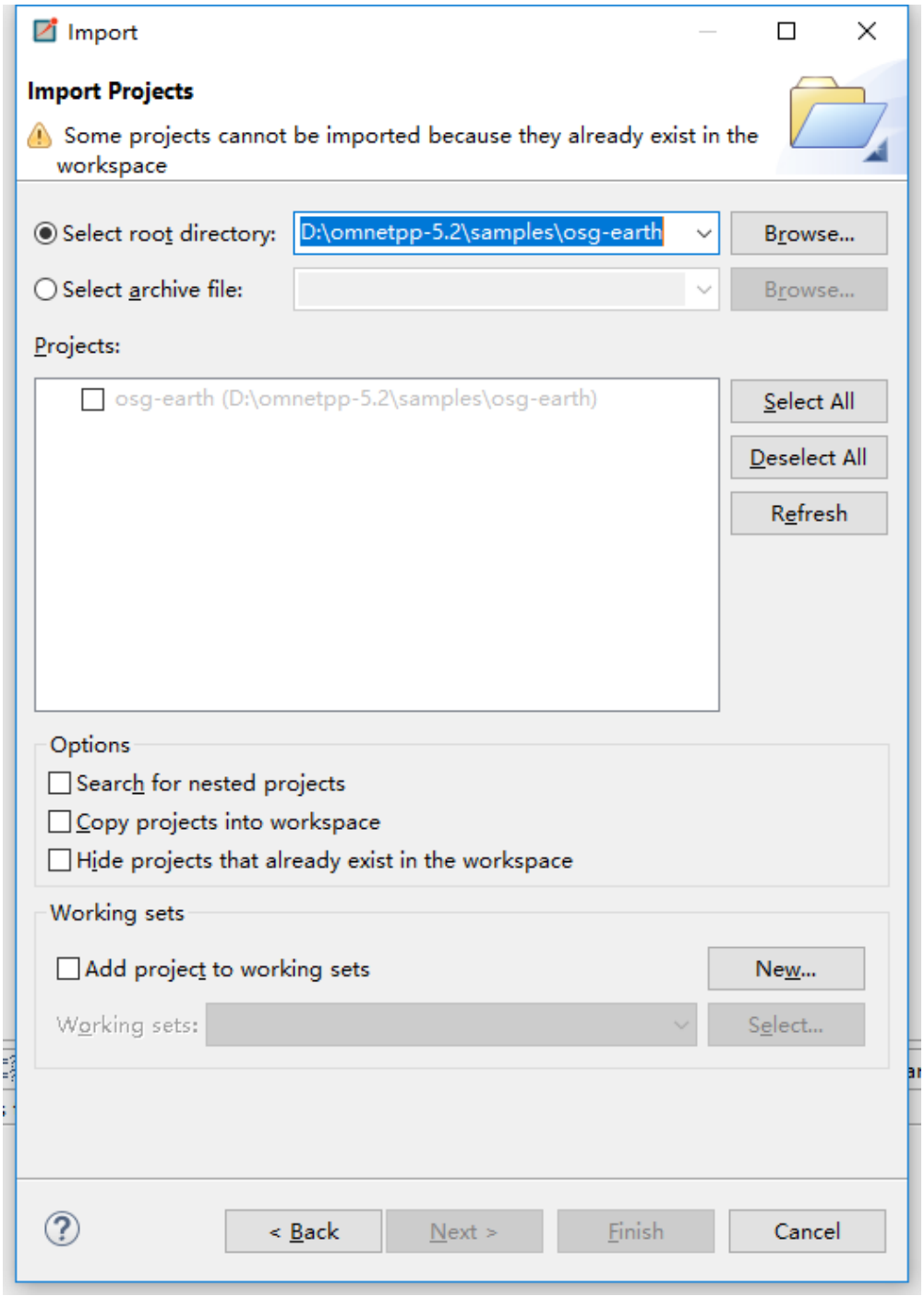


图　2.4: 点击 Existing Projects into Workspace 后视图

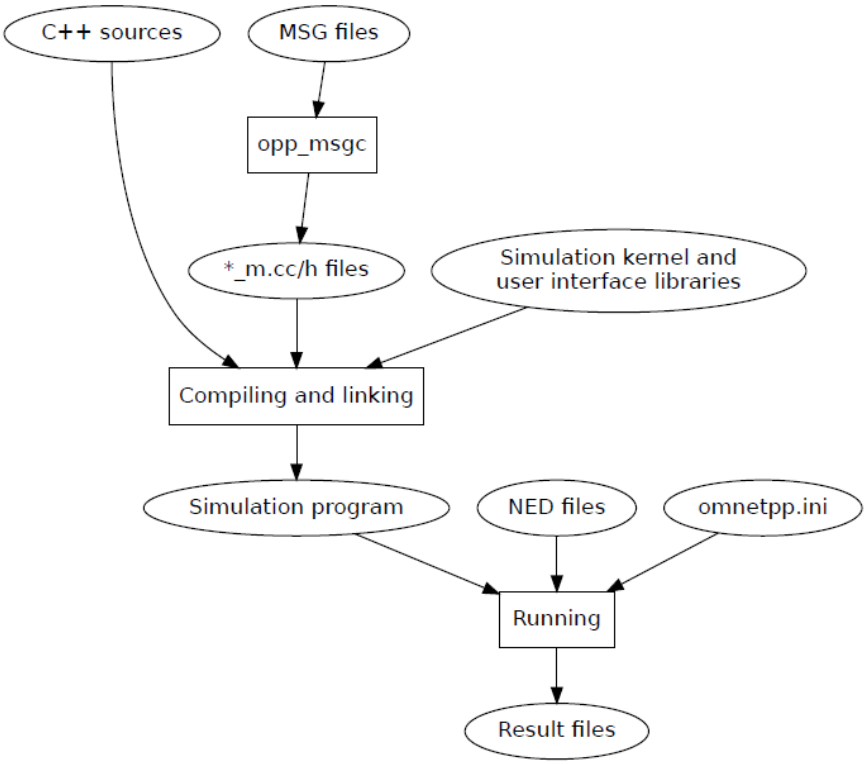


图 2.5: 编译与执行仿真流程

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

OMNeT++ 指南

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》

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：

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）

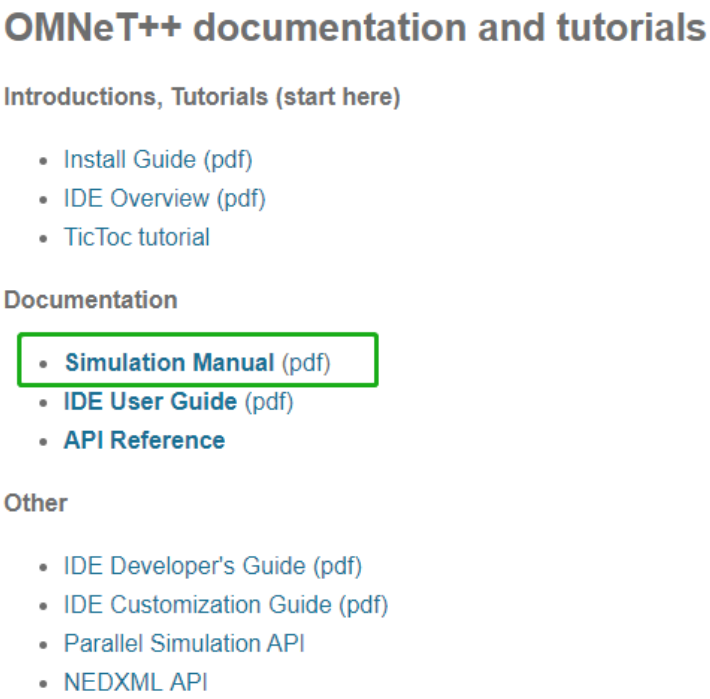


图 3.1: 手册

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：

从上图也可以看出，我们需要选择 c/c++》》Editor》》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：

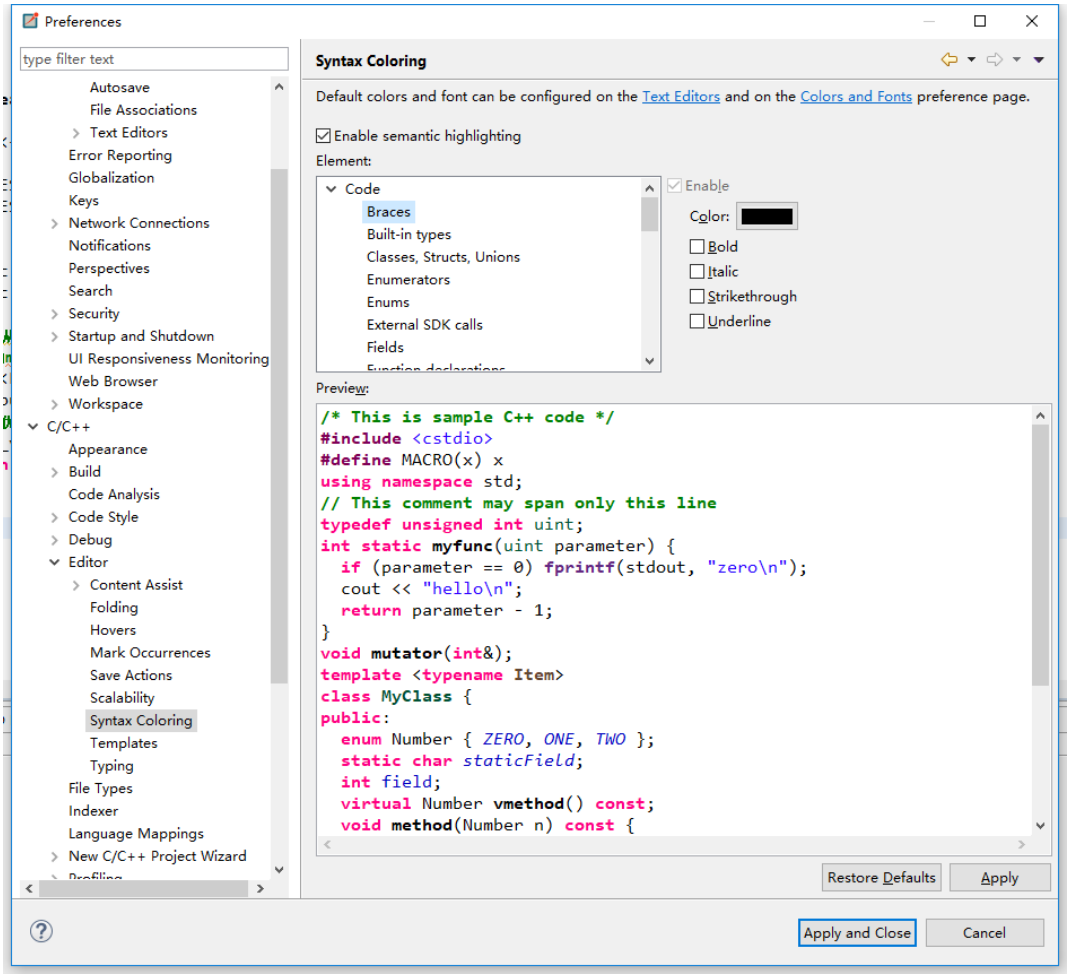

Preferences
type filter text
Autosave
File Associations
Text Editors
Error Reporting
Globalization
Keys
Network Connections
Notifications
Perspectives
Search
Security
Startup and Shutdown
UI Responsiveness Monitoring
Web Browser
Workspace
C/C++
Appearance
Build
Code Analysis
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Save Actions
Scalability
Syntax Coloring
Templates
Typing
File Types
Indexer
Language Mappings
New C/C++ Project Wizard
Syntax Coloring
Default colors and font can be configured on the Text Editors and on the Colors and Fonts preference page.
Enable semantic highlighting
Element:
Code
Braces
Built-in types
Classes, Structs, Unions
Enumerators
Enums
External SDK calls
Fields
Enable
Color:
Bold
Italic
Strikethrough
Underline
Preview:
/* This is sample C++ code */
#include <cstdio>
#define MACRO(x) x
using namespace std;
// This comment may span only this line
typedef unsigned int uint;
int static myfunc(uint parameter) {
if (parameter == 0) fprintf(stdout, "zero\n");
cout << "hello\n";
return parameter - 1;
}
void mutator(int&);
template <typename Item>
class MyClass {
public:
enum Number { ZERO, ONE, TWO };
static char staticField;
int field;
virtual Number vmethod() const;
void method(Number n) const {
Restore Defaults
Apply
Apply and Close
Cancel

图 3.2: cpp 高亮设置

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：

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。

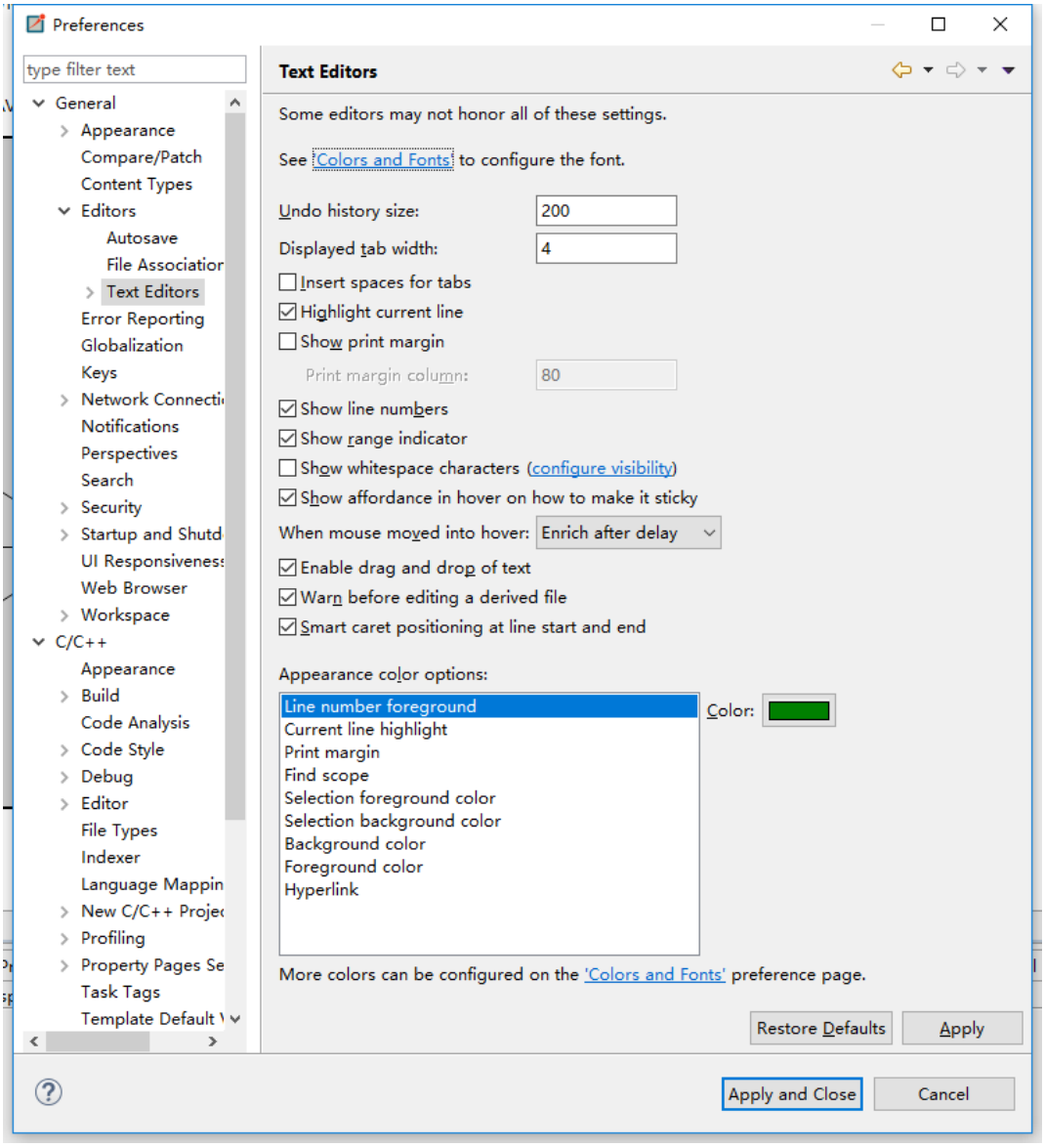

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Autosave
File Association
Text Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Search
Security
Startup and Shutd
UI Responsiveness
Web Browser
Workspace
C/C++
Appearance
Build
Code Analysis
Code Style
Debug
Editor
File Types
Indexer
Language Mappin
New C/C++ Proje
Profiling
Property Pages Se
Task Tags
Template Default
Text Editors
Some editors may not honor all of these settings.
See Colors and Fonts to configure the font.
Undo history size: 200
Displayed tab width: 4
Insert spaces for tabs
Highlight current line
Show print margin
Print margin column: 80
Show line numbers
Show range indicator
Show whitespace characters (configure visibility)
Show affordance in hover on how to make it sticky
When mouse moved into hover: Enrich after delay
Enable drag and drop of text
Warn before editing a derived file
Smart caret positioning at line start and end
Appearance color options:
Line number foreground
Current line highlight
Print margin
Find scope
Selection foreground color
Selection background color
Background color
Foreground color
Hyperlink
Color:
More colors can be configured on the 'Colors and Fonts' preference page.
Restore Defaults
Apply
Apply and Close
Cancel

图 3.3: 其他设置界面

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
{
    friend class cGate;
    friend class cSimulation;
    friend class cModuleType;
    friend class cChannelType;

    public:
        /* 迭代器 */
        GateIterator; /* 门迭代器 */
        SubmoduleIterator; /* 复合模块的子模块迭代器 */
        ChannelIterator; /* 模块信道迭代器 */

    public:
        virtual void callRefreshDisplay();
        virtual const char *getFullName(); /* 获取模块全名（绝对名） */
        virtual std::string getFullPath(); /* 获取模块路径（绝对路径） */
        virtual bool isSimple();
```

```
20
21          /* 返回模块的父模块，对于系统模块，返回nullptr */
22          virtual cModule *getParentModule();
23          bool isVector(); /* 如何模块是使用向量的形式定义的，返回true */
24          int getIndex(); /*返回模块在向量中的索引 */
25          /*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true */
26          int getVectorSize();
27
28          _OPPDEPRECATED int size(); /* 与getVectorSize()功能相似 */
29          virtual bool hasSubmodules(); /* 检测该模块是否有子模块 */
30
31          // 寻找子模块name，找到返回模块ID，否则返回-1
32          // 如何模块采用向量形式定义，那么需要指明index
33          virtual int findSubmodule(const char *name, int index=-1);
34
35          // 直接得到子模块name的指针，没有这个子模块返回nullptr
36          // 如何模块采用向量形式定义，那么需要指明index
37          virtual cModule *getSubmodule(const char *name, int index=-1);
38
39          /* 一个更强大的获取模块指针的接口，通过路径获取 */
40          virtual cModule *getModuleByPath(const char *path);
41
42          /* 门的相关函数 */
43          virtual bool hasGate(const char *gatename, int index=-1); /* 检测是否有门
        */
44          /* 寻找门，如果没有返回-1，找到返回门ID */
45          virtual int findGate(const char *gatename, int index=-1);
46          const cGate *gate(int id); /* 通过ID得到门地址，目前我还没有用到过 */
47          virtual void deleteGate(const char *gatename); /*删除一个门（很少用） */
48          /* 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"/"
        $o"） */
49          virtual std::vector<const char *> getGateNames();
50          /* 检测门（向量门）类型，可以标明"$i","$o" */
51          virtual cGate::Type gateType(const char *gatename);
52          /* 检测是否是向量门，可以标明"$i","$o" */
53          virtual bool isGateVector(const char *gatename);
54          /* 得到门的大小，可以指明"$i","$o" */
55          virtual int gateSize(const char *gatename);
56
57          /* 在父模块中寻找某个参数，没找到抛出cRuntimeError */
58          virtual cPar& getAncestorPar(const char *parname);
59          /* 设置是否在此模块的图形检查器上请求内置动画 */
60          virtual void setBuiltinAnimationsAllowed(bool enabled);
61          virtual void deleteModule(); /* 删除自己 */
62          /* 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 */
63          virtual void changeParentTo(cModule *mod);
64  };
```

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

```
usage:
for (cModule::GateIterator it(module); !it.end(); ++it) {
        cGate *gate = *it;
        ...
}
```

morekeywords

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

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

morekeywords

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

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

morekeywords

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。

```
class SIM_API cPar : public cObject
{
    friend class cComponent;
```

```
4     public:
5         virtual const char *getName(); /* 返回参数的名字 */
6         virtual std::string str(); /* 以字符串的形式返回参数 */
7
8         virtual cObject *getOwner();
9         Type getType();
10
11        static const char *getTypeName(Type t);
12
13        bool isNumeric();
14
15        bool isVolatile();
16
17        bool isExpression();
18
19        bool isShared();
20
21        bool isSet();
22
23        cPar& setBoolValue(bool b);
24
25        cPar& setLongValue(long l);
26
27        cPar& setDoubleValue(double d);
28
29        cPar& setStringValue(const char *s);
30
31        cPar& setStringValue(const std::string& s);
32
33        cPar& setXMLValue(cXMLElement *node);
34
35        bool boolValue();
36
37        long longValue();
38
39        double doubleValue();
40
41        const char *getUnit();
42
43        const char *stringValue();
44
45        std::string stdstringValue();
46
47        cXMLElement *xmlValue();
48
49        void parse(const char *text);
50
51        /* 与stdstringValue()功能一样 */
52        operator std::string() const  {return stdstringValue();}
53
```

```
54          /* 与xmlVlaue()等同。注意：返回对象树的生命周期被限制了，具体看xmlValue说
    明。 */
55          operator cXMLElement *() const  {return xmlValue();}
56  };
```

morekeywords

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。

```
1   class SIM_API cGate : public cObject, noncopyable
2   {
3       friend class cModule;
4       friend class cModuleGates;
5       friend class cPlaceholderModule;
6
7       public:
8           cGate *prevGate;
9           cGate *nextGate;
10
11          static int lastConnectionId;
12
13          static void clearFullnamePool();
14
15          // internal
16          void installChannel(cChannel *chan);
17
18          // internal
19          void checkChannels() const;
20
21          /* 例如返回门out */
22          virtual const char *getName() const override;
23
24          /* 与getName()不同，需要返回门索引，例如out[4] */
25          virtual const char *getFullName() const override;
26
27          virtual bool deliver(cMessage *msg, simtime_t at);
28
29          cChannel *connectTo(cGate *gate, cChannel *channel=nullptr, bool
        leaveUninitialized=false);
30
31          void disconnect();
32
33          cChannel *reconnectWith(cChannel *channel, bool leaveUninitialized=false)
        ;
34
35          const char *getBaseName();
36
```

```
37          const char *getNameSuffix();
38
39          cProperties *getProperties();
40
41          Type getType();
42
43          static const char *getTypeName(Type t);
44
45          cModule *getOwnerModule();
46
47          int getId();
48
49          bool isVector();
50
51          int getIndex();
52
53          int getVectorSize();
54
55          cChannel *getChannel();
56
57          cChannel *getTransmissionChannel();
58
59          cChannel *findTransmissionChannel();
60
61          cChannel *getIncomingTransmissionChannel();
62
63          cChannel *findIncomingTransmissionChannel();
64
65          cGate *getPreviousGate();
66
67          cGate *getNextGate();
68
69          cDisplayString& getDisplayString();
70
71          void setDisplayString(const char *dispstr);
72  };
```

morekeywords

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  class SIM_API cLog
2  {
3      public:
```

```
4         static LogLevel logLevel;
5
6         static const char *getLogLevelName(LogLevel logLevel);
7
8         static LogLevel resolveLogLevel(const char *name);
9 };
10
11 #define EV_LOG(logLevel, category) OPP_LOGPROXY(getThisPtr(), logLevel, category)
      .getStream()
12
13 #define EV        EV_INFO
14
15 #define EV_FATAL  EV_LOG(omnetpp::LOGLEVEL_FATAL, nullptr)
16
17 #define EV_ERROR  EV_LOG(omnetpp::LOGLEVEL_ERROR, nullptr)
18
19 #define EV_WARN   EV_LOG(omnetpp::LOGLEVEL_WARN, nullptr)
20
21 #define EV_INFO   EV_LOG(omnetpp::LOGLEVEL_INFO, nullptr)
22
23 #define EV_DETAIL EV_LOG(omnetpp::LOGLEVEL_DETAIL, nullptr)
24
25 #define EV_DEBUG  EV_LOG(omnetpp::LOGLEVEL_DEBUG, nullptr)
26
27 #define EV_TRACE  EV_LOG(omnetpp::LOGLEVEL_TRACE, nullptr)
28
29 #define EV_C(category)        EV_INFO_C(category)
30
31 #define EV_FATAL_C(category)  EV_LOG(omnetpp::LOGLEVEL_FATAL, category)
32
33 #define EV_ERROR_C(category)  EV_LOG(omnetpp::LOGLEVEL_ERROR, category)
34
35 #define EV_WARN_C(category)   EV_LOG(omnetpp::LOGLEVEL_WARN, category)
36
37 #define EV_INFO_C(category)   EV_LOG(omnetpp::LOGLEVEL_INFO, category)
38
39 #define EV_DETAIL_C(category) EV_LOG(omnetpp::LOGLEVEL_DETAIL, category)
40
41 #define EV_DEBUG_C(category)  EV_LOG(omnetpp::LOGLEVEL_DEBUG, category)
42
43 #define EV_TRACE_C(category)  EV_LOG(omnetpp::LOGLEVEL_TRACE, category)
44 }
```

morekeywords

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：

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

```
    datarate = default(1000Mbps);
}
```

morekeywords

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

```
void Node::handleMessage(cMessage* msg)
{
    if(msg->isSelfMessage()){
        if(msg->getKind()==SMSG_INIT){
                ...
                ...
            cMessage* cloudMsg = new cMessage("hello");
            cloudMsg->setKind(SMSG_INIT);//设置节点类型
            scheduleAt(simTime()+0.01,cloudMsg); //调度一个事件，发送消息给自己
        }
    }
}
```

morekeywords

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：

```
time1 = simTime();
func();
time2 = simTime();
```

morekeywords

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：$time1==time2$，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：

```
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);
}
```

morekeywords

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！

```
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)
```

morekeywords

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：

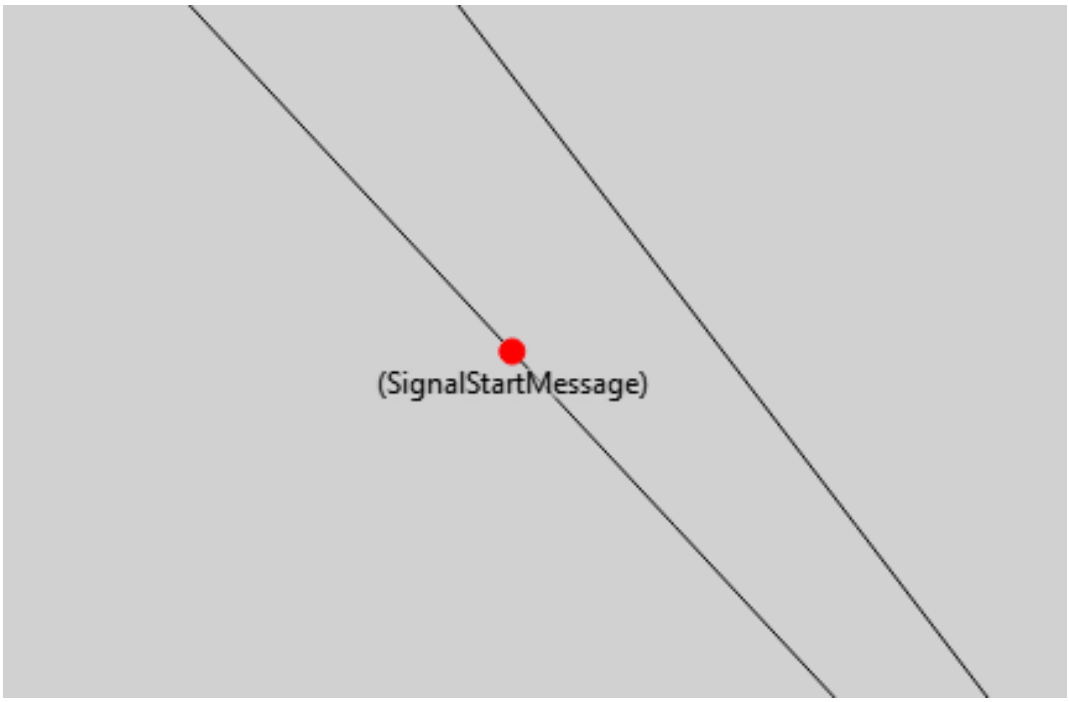


图　5.1: 普通传输效果图

如果在设计网络时，需要将包传输效果设置成图 5　2 所示，对于有线连接和无线连接的两个节点方法不同，对于有线连接的 send 函数无法在函数的参数上设置。需要在网络拓扑连接时设置好信道，如代码段 5　1 所示：

```
channel Channel extends DatarateChannel
{
    delay = default(uniform(20ns, 100ns));
    datarate = default(2000Mbps);
}
```

morekeywords

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。

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：

```
cModule *parent = getParentModule();

// 取出父模块下的beBuffer模块
cModule *psubmodBE = parent->getSubmodule("beBuffer");
BEBuffer *pBEBuffer = check_and_cast<BEBuffer *>(psubmodBE);

```

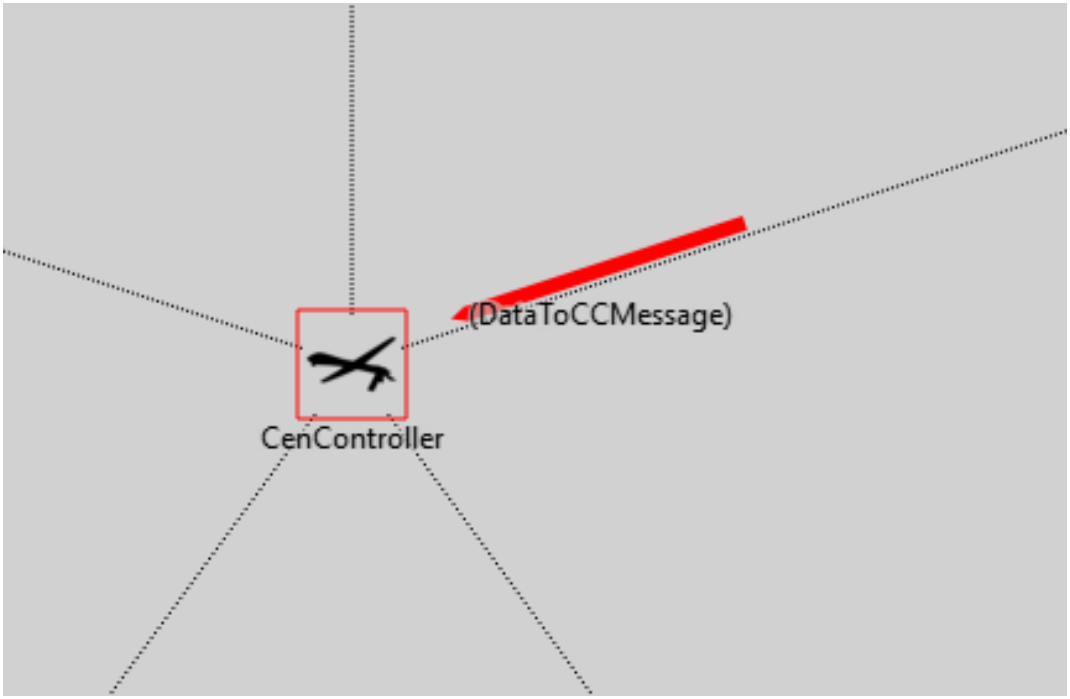


图 5.2: 设置传输延迟和持续时间

```
cModule *psubmodRC = NULL;
RCBuffer *pRCBuffer = NULL;
// 取出父模块下的rcBuffer模块
psubmodRC = parent->getSubmodule("rcBuffer");
pRCBuffer = check_and_cast<RCBuffer *>(psubmodRC);
```

morekeywords

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：

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

morekeywords

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:
  ...
}
```

morekeywords

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ES节点
        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;
        }
}
```

morekeywords

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headId号的模块
 */
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++){
```

```
11          int number_of_Bees = cSimulation::getActiveSimulation()->
    getLastComponentId();
12          cSimulation *simobj = cSimulation::getActiveSimulation();
13          /* TODO优化 */
14          mod = cSimulation::getActiveSimulation()->getModule(i);
15          if(strcmp(mod->getName(),"CenController") == 0){
16              /* 如果遍历到一个模块名为CenController的节点 */
17              continue;
18          }
19          else{
20              int j=0;
21              while(1){
22                  string modname = cSimulation::getActiveSimulation()->getModule(i)
    ->getName();
23                  midmod=cSimulation::getActiveSimulation()->getModule(i);
24                  Head=midmod->getSubmodule(this->clustername.c_str(),j)->
    getSubmodule("Wireless");
25                  if(((Node*)Head)->myId == this->headId){
26                      /* 找到簇头节点,退出while循环 */
27                      break;
28                  }
29                  j++;
30              }
31              // 在到满足条件的Head节点，开始执行相关操作
32              ...
33              ...
34              ...
35              break;
36          }
37      }
38  }
```

morekeywords

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：

```
1  network simplenet
2  {
3      parameters:
4          ...
5          ...
6      submodules:
7
```

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

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

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

    connections allowunconnected:
        ...
        ...
}
```

morekeywords

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：

```
cModule *parent = getParentModule();
const char *name = parent->getNedTypeName();

if (strcmp(name, "SimpleNetwork.Node.SimpleNode") == 0){
```

```
    cGate *outgate = gate("line$o");
    cChannel *chan = outgate->findTransmissionChannel();
    linkspeed = chan->getNominalDatarate();

}
else if (strcmp(name, "SimpleNetwork.Switch.SwitchPort") == 0){
    //int id = parent->findGate("line$o");
    cGate *outgate = parent->gate("line$o");
    cChannel *chan = outgate->findTransmissionChannel();
    linkspeed = chan->getNominalDatarate();
}
```

morekeywords

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
    SimpleNetwork.Switch.SimpleSwitch").asVector());

    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i的最短距离
        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();
```

```
26          int gateIndex = parentModuleGate->getIndex();
27          int address = topo->getNode(i)->getModule()->par("address");
28          rtable[address] = gateIndex;
29          EV << "  towards address " << address << " gateIndex is " << gateIndex
        << endl;
30      }
31      delete topo;
32  }
```

morekeywords

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：

```
1  /** @name Algorithms to find shortest paths. */
2  /*
3  * To be implemented:
4  *    -  void unweightedMultiShortestPathsTo(Node *target);
5  *    -  void weightedMultiShortestPathsTo(Node *target);
6  */
7
8  //@{
9
10 /**
11 * Apply the Dijkstra algorithm to find all shortest paths to the given
12 * graph node. The paths found can be extracted via Node's methods.
13 */
14 virtual void calculateUnweightedSingleShortestPathsTo(Node *target);
15
16 /**
17 * Apply the Dijkstra algorithm to find all shortest paths to the given
18 * graph node. The paths found can be extracted via Node's methods.
19 * Uses weights in nodes and links.
20 */
21 virtual void calculateWeightedSingleShortestPathsTo(Node *target);
```

morekeywords

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：

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

morekeywords

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5.2.9-1
2 // 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
3 auto parentdispStr = parents->getDisplayString();
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xpos改变
```

```
8 coord_Y.setDoubleValue(this->ypos); //将仿真界面上的ypos改变
```

morekeywords

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。

对于一个复合模块中的一个简单节点想获符合节点中的坐标以及该节点的移动速度时，可以利用 inet 中自带的 iMobility 模块即可以实现完成。具体的实现程序如下所示：

```
1 程序5.2.9-1
2 // 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
3 cModule *temp_NodeModule=this->getParentModule();
4 inet::IMobility *node_Mobility=check_and_cast<IMobility *>(temp_NodeModule->
      getSubmodule("mobility"));
5 Coord coord_Node=node_Mobility->getCurrentPosition();//获取节点的位置
6 double NodeX=coord_Node.x;
7 double NodeY=coord_Node.y;
8 Coord speed_Node=node_Mobility->getCurrentVelocity();//获取节点的速度
9 double  NodeVv=speed_Node.x;
10 double NodeVy=speed_Node.y;
```

morekeywords

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)
5 #else
6 #  define OPP_DLLIMPORT
7 #  define OPP_DLLEXPORT
8 #endif
```

morekeywords

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：

如果以上关系皆满足，再出现在链接工程的错误可能其其他导致的。

5.2 可视化接口

5.2.1 设置消息传输颜色

5.2.2 设置节点

在设置节点前，先阅读以下代码：

```
this->getDisplayString().setTagArg("i",1,"red");
```

morekeywords

由以上代码函数名基本可以猜想到该行代码可能是设置显示相关标记，但具体设置哪一种标记可能需要知道 setTagArg() 函数的参数意义。其中 i 代表所设置 Tags 类型，在 **OMNeT++ 中 Tags** 类型包括：

p positioning and layout

p positioning and layout

b shape (box, oval, etc.)

i icon

is icon size

i2 auxiliary or status icon

r range indicator

q queue information text

t text

tt tooltip

所以，根据以上列表得知 prog-5.x 将设置图标相关属性，接下来我们需要知道紧接着的两个参数含义：

```
@display("i=block/source,red,20")
```

morekeywords

从 ned 文件中节点显示设置可以看出，属性 i 后第 1 个参数为 red（从 0 开始数），那么 prog-5.x 便是将 i 后第 i 个参数设置为 red，到这里我们对 **setTagArg()** 函数的使用便掌握了。

5.3 调试技巧

5.3.1 gdb 调试

OMNeT++ 支持 gdb 调试，与其他 IDE 调试方式相似，不同之处在于，若在网络仿真原型中设有统计参数，需在配置文件 omnetpp.ini 中设置：

```
check-signals = false
```

morekeywords

若仿真过程中，未关闭 check-signals，调试过程将会发生统计参数内存分配问题。

5.3.2 log 日志类

由于 OMNeT++ 所提供 EV 显示信息在关闭仿真程序后，无法查看，可采用日志类将仿真信息打印到文本文件（亦可将 cout 重定向到文件）。在这里分享一个作者在调试 OMNeT++ 仿真程序时编写的日志类 logging。使用方法如下：

```
/* 声明 */
using namespace logging

logger variablename("path to file","filename");

/* 使用 */
logger<<"current simulation time = "<<simTime()<<endl;
```

morekeywords

下载地址：logging-class[1]

5.3.3 技巧十二：如何实现跨模块进行调用函数或参数

在进行仿真的过程中，难免用到跨模块的函数调用或者参数调用，本部分主要对这部分进行简单的介绍：其大致思路如下：1）从复合模块中的一个简单模块退到该简单模块的上一层，也就是其父模块；

```
cModule *temp_Module=this->getParentModule();
```

morekeywords

2）从父模块中找到你所要找到包含你需要的函数的子模块，也就是简单模块；

```
App *temp_mobility=check_and_cast<App *>(temp_Module->getSubmodule("app"));
/* App表示子模块的类名称 ，app表示你所需要的模块*/
```

morekeywords

3）找到你所需要的模块之后，然后就可以获得你所需要的函数或参数变量；

[1] https://github.com/wangrongwei/lazytools

```
if(this->mClusterHead==app->myAddress){
    if(!app->mIsClusterHead){//这就是获得其他模块中的参数
        app->mIsClusterHead=true;
    }
}
```

morekeywords

5.3.4 技巧十三：如何实现节点消息的同时显示

当一个节点对多个节点进行发送消息时，为了在视觉上看到消息同时从一个端口发出，只需要利用一个函数就可以解决：

```
simtime_t txFinishTime = gate("line$o")->getTransmissionChannel()->
    getTransmissionFinishTime();
if((txFinishTime == -1) || (txFinishTime < simTime())){
    //通过修改延迟可以使节点能够同时发送消息
    sendDirect(pk,0.001,0,allUAV[(int)pk->getDestAddr()],"port$i",address);
}
else{
        sendDirect(pk,txFinishTime-simTime(),0.5,allUAV[(int)pk->getDestAddr()],"
    port$i",address);
    }
```

morekeywords

需要注意该函数一定放在与外界相连接的简单模块，不然是看不到的；

最终的显示效果入下图所示：

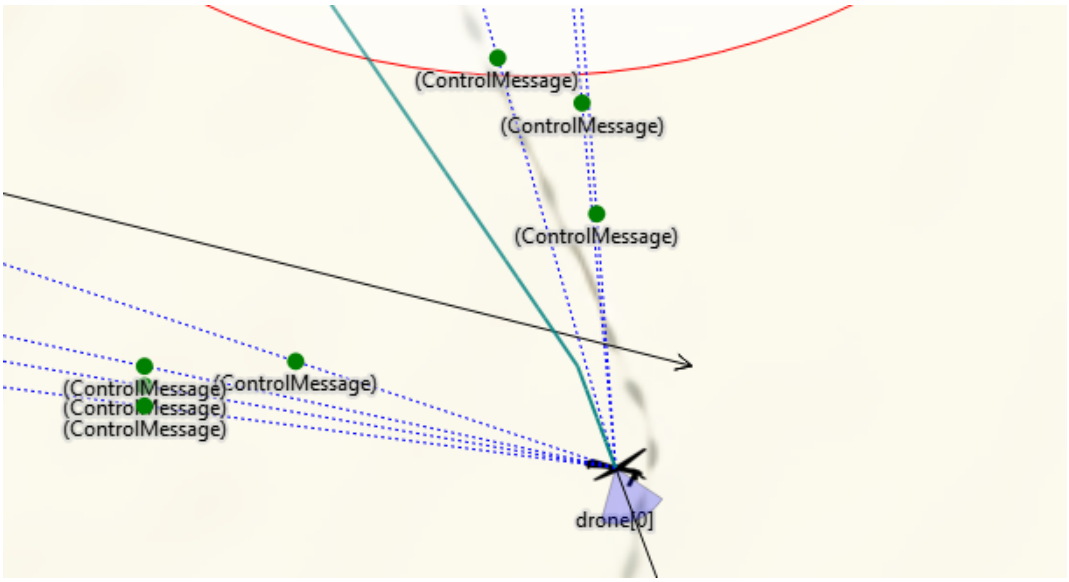


图 5.3: avatar

5.4　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

OMNeT++ 数据统计与仿真分析

OMNeT++ 提供标量和矢量的形式对数据进行统计。

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：

[1] 如何设置需要统计的标量和矢量？

[2] 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 统计结果文件

先备注一下，每次运行完仿真后，将会产生三个文件：

sca 标量统计结果，仅为一个数据，例如平均值或最大值；

vci 待补充

vec 矢量统计结果，一组数据，且每一个数据存在一个时间戳，因此可以看做(t,data) 的形式；

通过点击 vec 文件，**OMNeT++ 将提醒保存 anf 文件，新建的 anf 文件中以包括了 sca 和 vec 两个文件，此时可通过 anf 文件同时查看到标量数据和矢量数据。此外，anf 文件在仿真程序运行过程中将根据主动更新，因此下一次运行相同的配置时，anf** 中数据自动更新。

6.2 仿真结果统计

6.2.1 标量

在 OMNeT++ 中，操作标量统计接口主要包括两个步骤：在模块的 ned 文件中添加需统计标量，在模块 cc 文件中对标量进行注册。ned 文件添加如下：

```
1    @signal[TaskThroughput_Network](type=long);
2    @statistic[TaskThroughput_Network](title="Task throughput of network";unit=
     long;record=mean,max;interpolationmode=none);
```

```
morekeywords
```

在模块的 initialize 函数中，利用 registerSignal 函数对信号进行注册。代码示例如下：

```
1     ThroughputSignal = registerSignal("TaskThroughput_Network");
```

```
morekeywords
```

完成以上操作，在需要统计该变量的地方可使用 emit 函数进行记录。

6.2.2 矢量

与标量的操作方式相似，都需要在 ned 文件中添加相应变量和注册。唯一不同，主要在 ned 文件中 record 的值存在差异，详细如下：

```
1     @signal[endToEndDelay_Packet](type="simtime_t");
2     @statistic[endToEndDelay_Packet](title="end-to-end delay of arrived packets";
      unit=s;record=vector,mean,max;interpolationmode=none);
```

```
morekeywords
```

```
1     end2endSignal = registerSignal("endToEndDelay_Packet");
```

```
morekeywords
```

6.2.3 直方图

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

```
1 cLongHistogram hopCountStats;
2 hopCountStats.setName("hopcountStats"); /* 设置名称 */
3 hopCountStats.setRangeAutoUpper("0,10,1.5"); /* 设置上限值 */
4 hopCountStats.collect(hopcount); /* 记录数据 */
```

```
morekeywords
```

以上为使用 cLongHistogram 类的常用成员函数，此外，getMin()、getMax()、getMean() 以及 getStddev() 等其他成员函数可通过查看头文件了解。

cOutVector：获取输出向量

```
1 cOutVector hopCountVector;
2 hopCountVector.setName("Hopcount"); /* 设置名称 */
3 hopCountVector.record(hopcount); /* 记录数据 */
```

```
morekeywords
```

recordScalar

```
recordScalar(“string 输出名称”, 输出变量名)
```

morekeywords

输出程序中某个标量的值，直接调用即可。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。一般在 Finish() 函数中调用 recordScalar 函数。

6.3 仿真结果分析

仿真的结果存储在 project 下 result 文件夹中。例如：

```
cOutVector: \result\**.vec
cLongHistogram: \result\**.sca
```

morekeywords

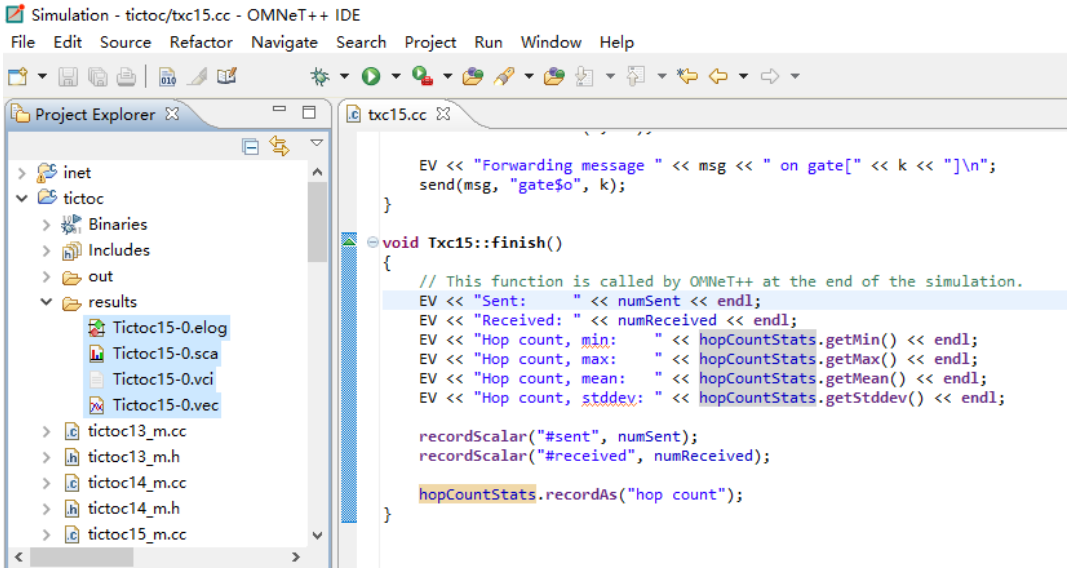


图 6.1: doc 目录

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。

然后会让我们建立新的分析文件。点击 finish 即可

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

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。

双击打开想查看的一行，如下所示：

接下来我们打开 sca 文件查看直方图：

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：

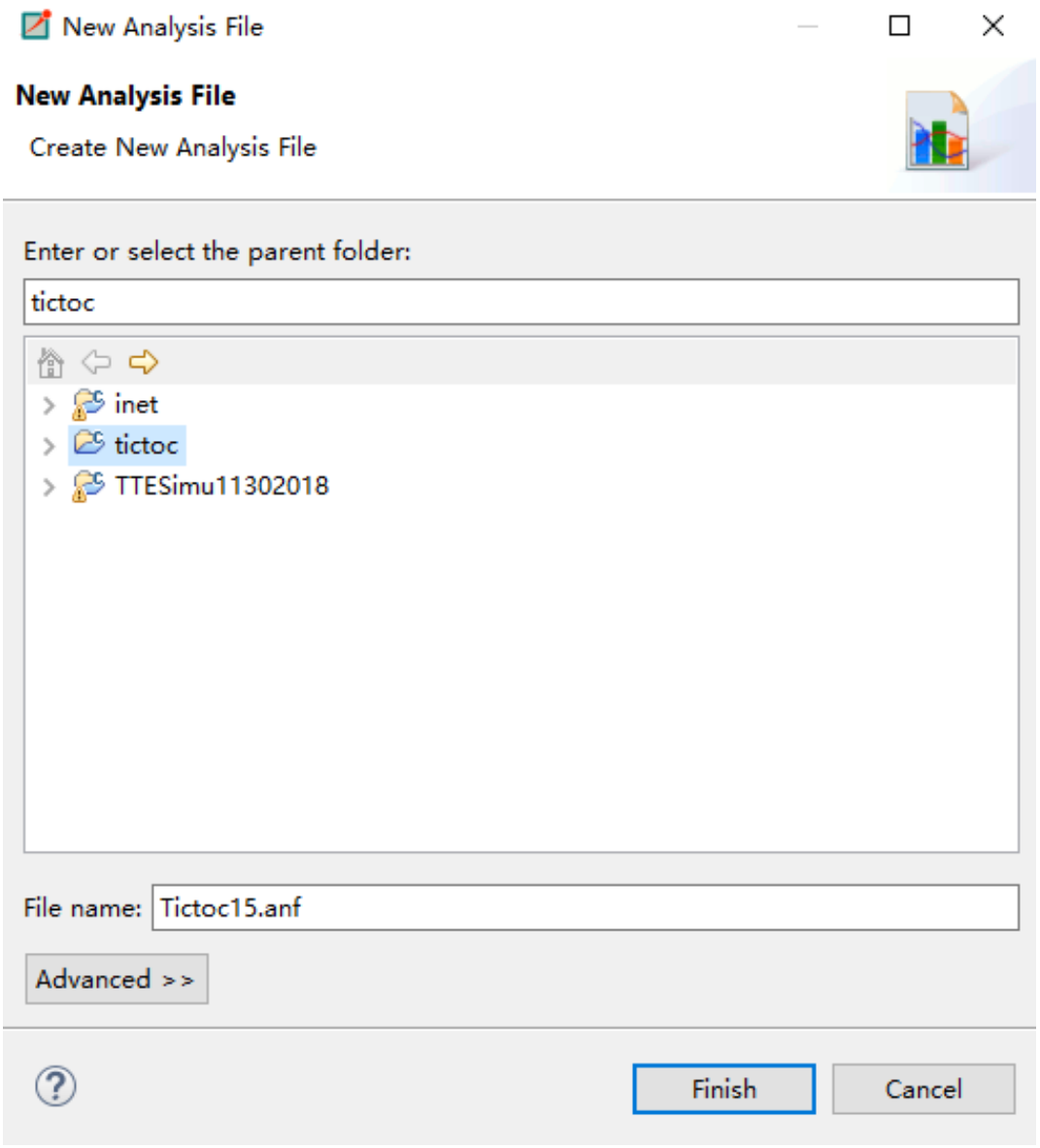

New Analysis File
New Analysis File
Create New Analysis File
Enter or select the parent folder:
tictoc
inet
tictoc
TTESimu11302018
File name:
Tictoc15.anf
Advanced >>
Finish
Cancel

图 6.2: doc 目录

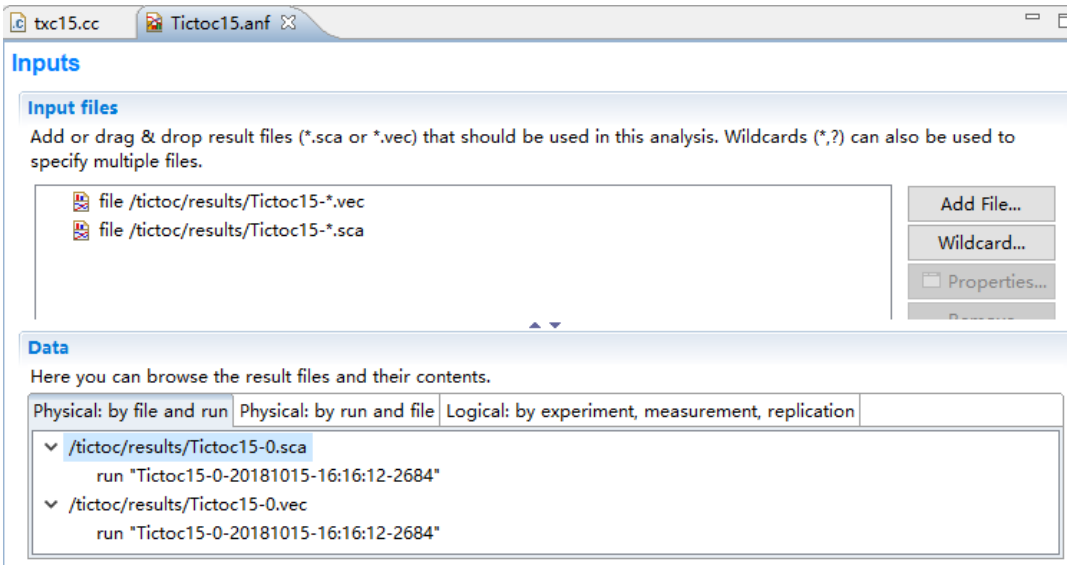


图 6.3: doc 目录

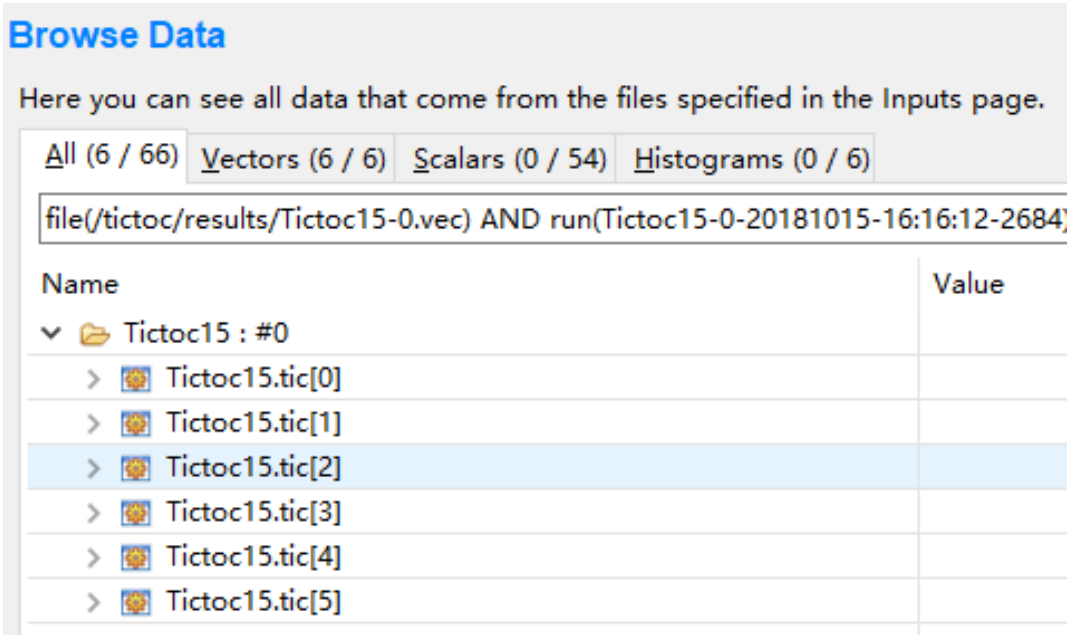


图 6.4: doc 目录

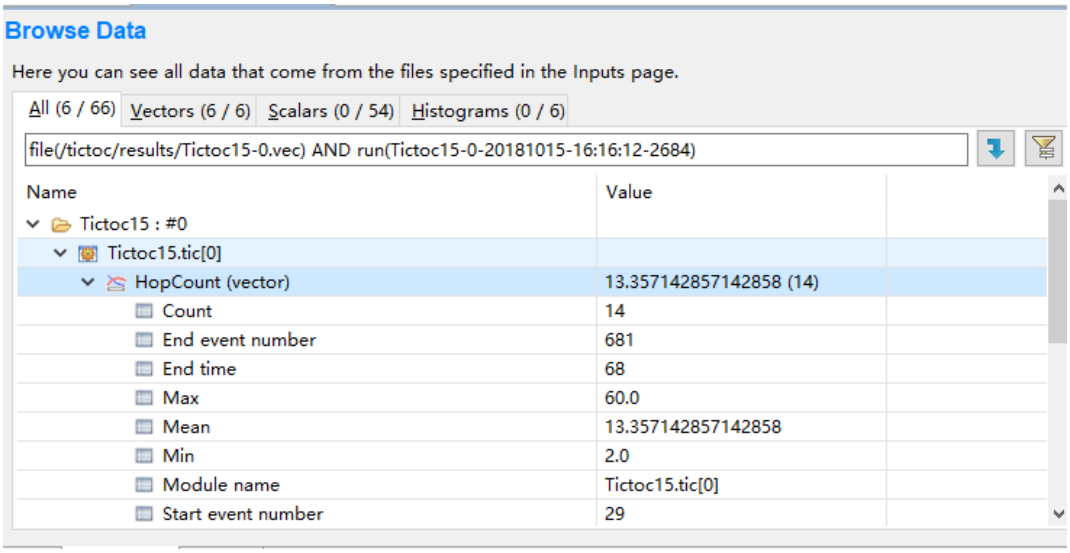


图 6.5: doc 目录

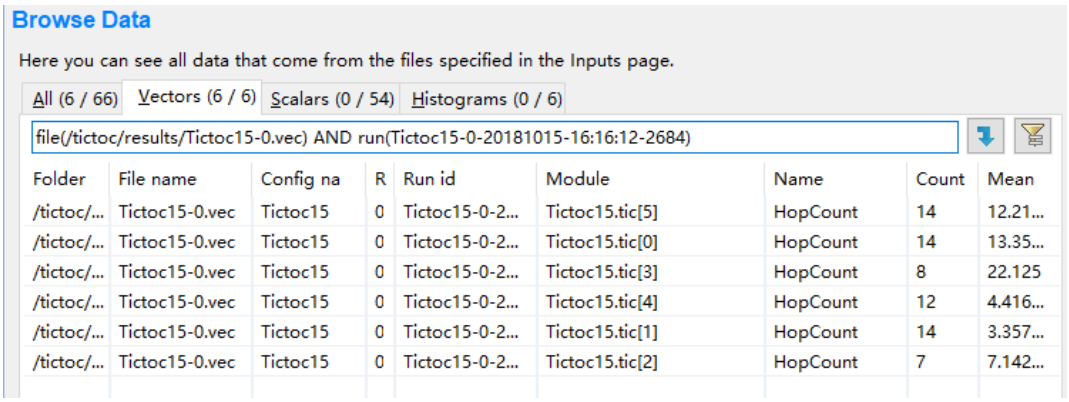


图 6.6: doc 目录

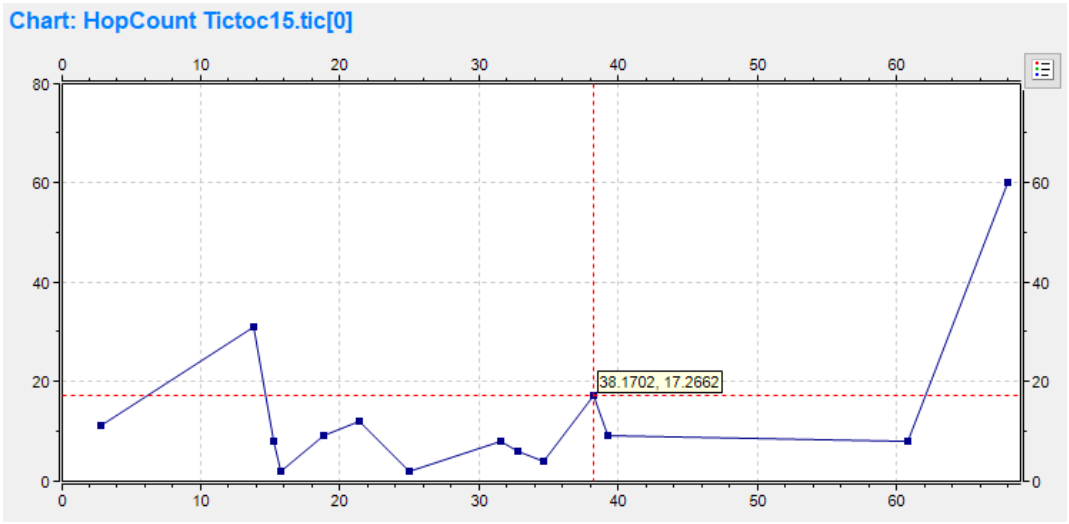


图 6.7: doc 目录

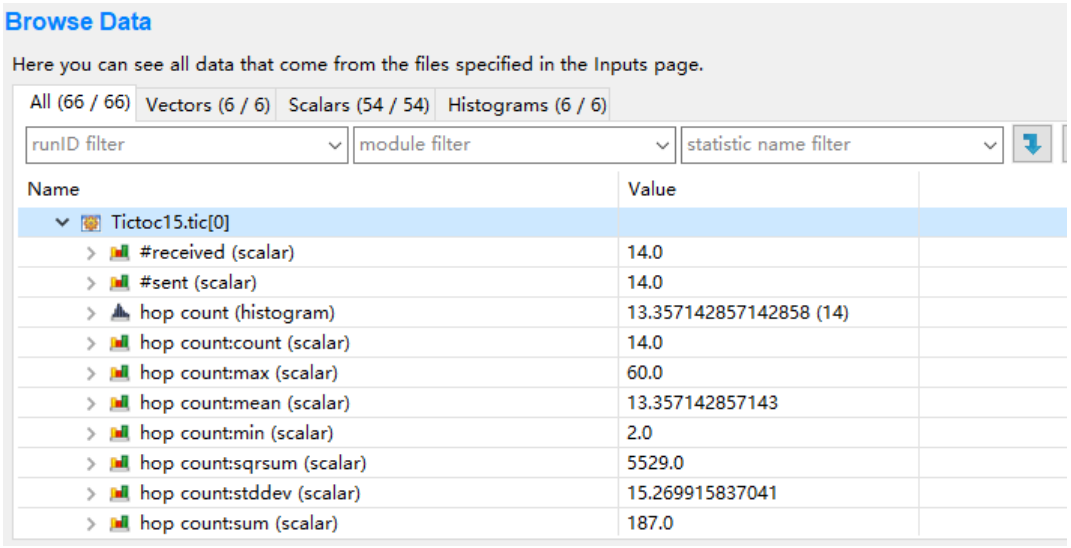


图 6.8: doc 目录

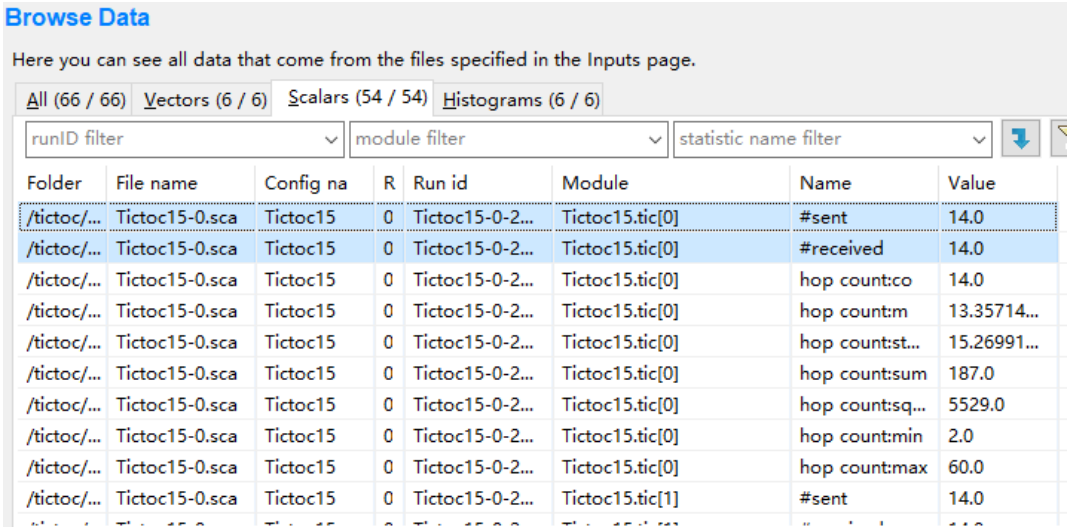


图 6.9: doc 目录

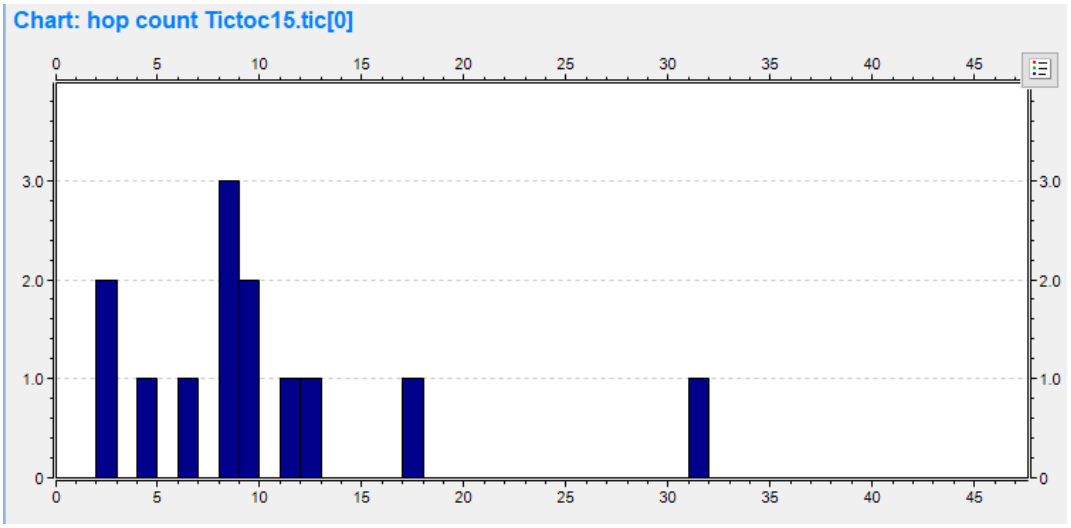


图 6.10: doc 目录

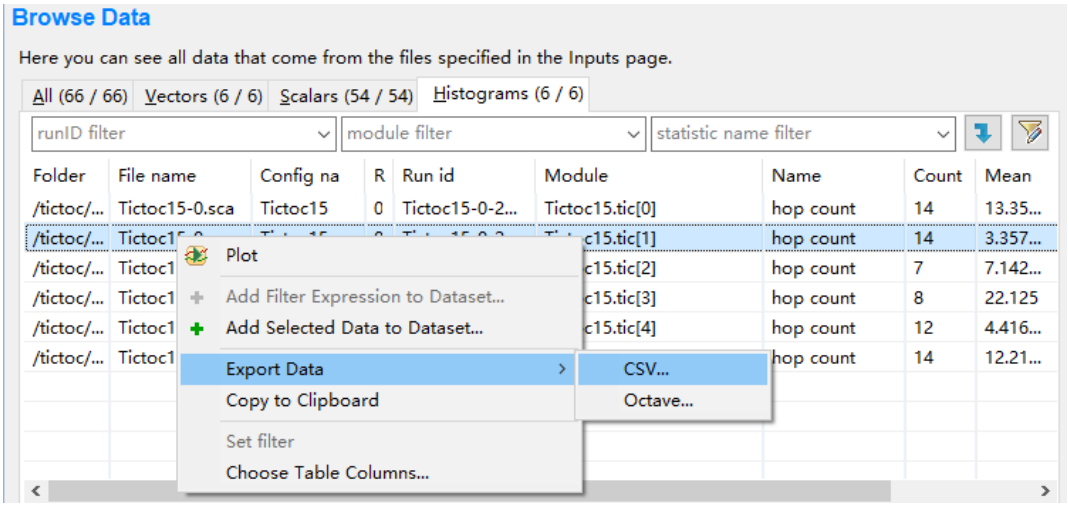


图　6.11：doc 目录

另外，我们也可以直接取出数据，方法如下：

之后可以自己进行数据处理和作图。

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。

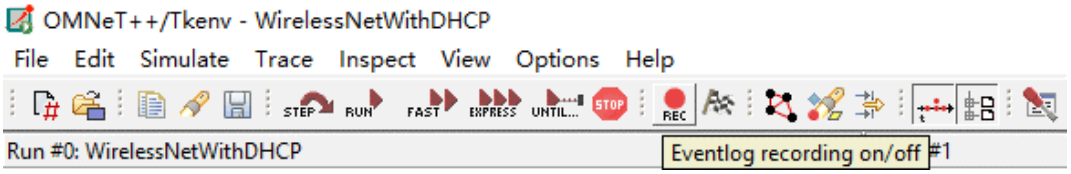


图　6.12：avatar

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：

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：

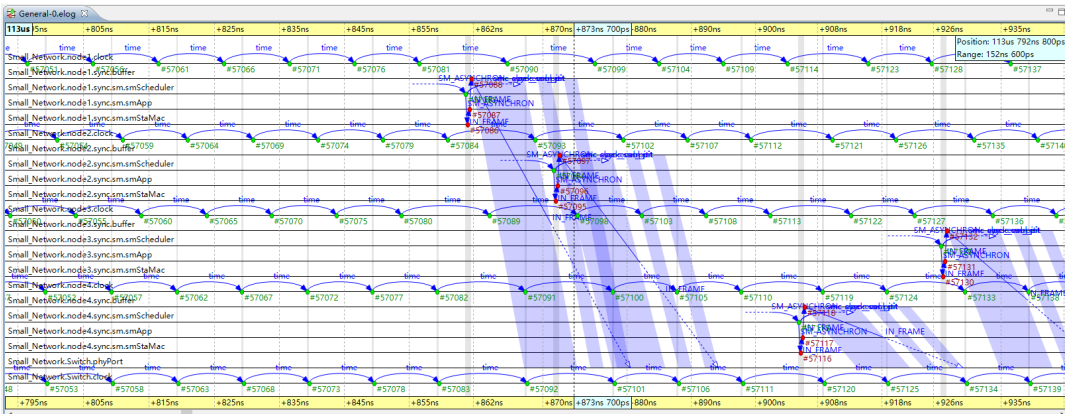


图 6.13: 常用符号

Net10.rte[7].queue[3]	以模块的全路径作为轴的标签
	空心圆，初始化事件
	绿色实心圆，自消息处理事件
	红色实心圆，消息处理事件
#5078	事件编号
	弯曲蓝色箭头，自消息
	蓝色箭头，消息发送
	绿箭头，消息重用
	棕色虚箭头，函数调用
	消息发送距离比较远
	虚拟消息发送
	传输持续时间；传输的开始移交所有权
	传输持续时间；传输的结束移交所有权
	分割传输时间；传输的开始移交所有权
	分割传输时间；传输的结束移交所有权
pk-9-to-1-#58	消息名称
requestPacket()	函数名称
	灰色背景，零仿真时间区域
	灰色虚线，仿真时间开始线

图 6.14: 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。

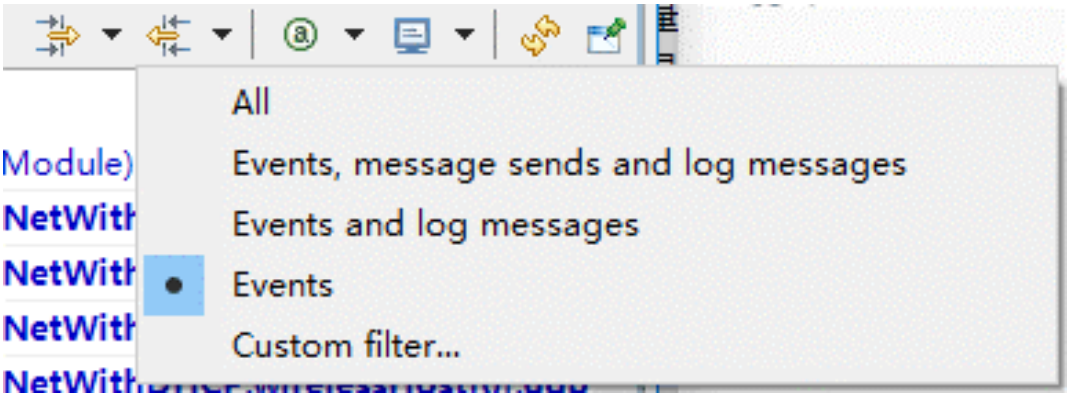


图　6.15: avatar

此外，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。

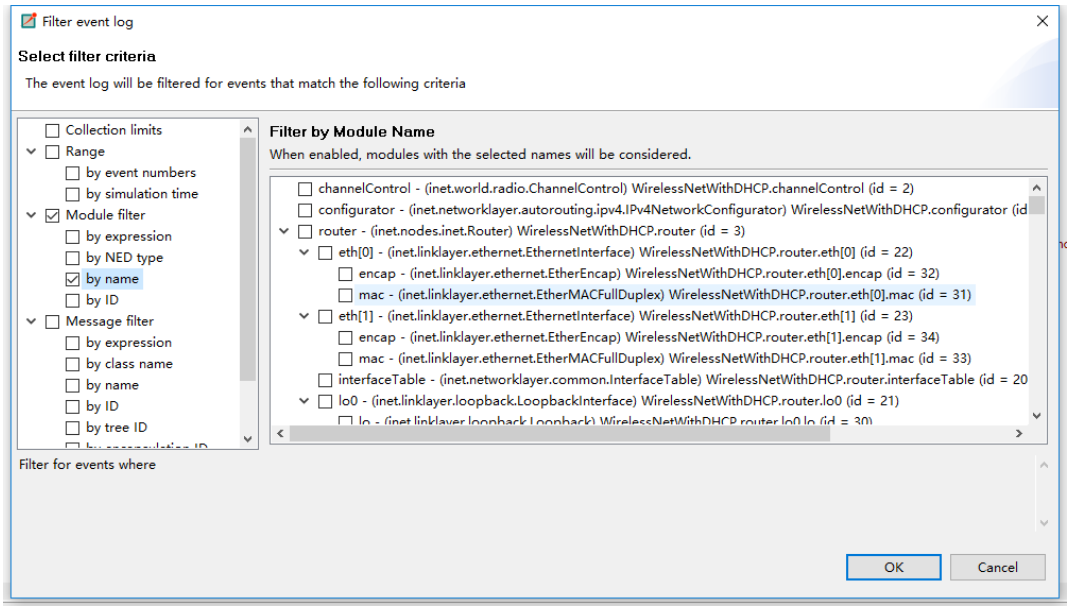


图　6.16: avatar

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。

6.5 小结

OMNeT++ 提供的普通的标量数据形式统计接口和带时间戳的矢量形式统计方式，两种方式已经基本满足需求。若需要复杂的数据对比图，可在

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
```

morekeywords

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
```

morekeywords

7.2 高亮显示 cModule 等类

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。

7.3 调用 INET 类

7.3.1 问题描述

在对节点的坐标进行实时的显示的过程中，编写如下的函数：

```
1 void node::initialize()
2 {
3     cModule *host = getContainingNode(this);
4     IMobility *mobility = check_and_cast<IMobility *>(host->getSubmodule("
      mobility"));
5     Coord selfPosition = mobility->getCurrentPosition();
6 }
```

morekeywords

同时以 “inet/mobility/contract/IMobility.h” 的形式引用头文件，但是在编译的过程中，会报出如下的错误：

```
1 ../out/clang-release/src/node.o:(.text[
      _ZN7omnetpp14check_and_castIPN4inet9IMobilityENS_7cModuleEEET_PT0_]+0x18):
      undefined reference to `__imp__ZTIN4inet9IMobilityE'
2 ../out/clang-release/src/node.o:(.rdata[_ZTIPN4inet9IMobilityE]+0x18): undefined
      reference to `typeinfo for inet::IMobility'
```

morekeywords

7.3.2 解决办法

方法一：在 inet 中将 IMobility.h 中第一行将 “INET_API” 删除后，重新对 inet 进行编译，然后在对所建立的工程进行编译，编译既可以通过。

方法二：打开当初安装 OMNET++ 的文件夹，找到 configure.user 的文本，打开后，找到 CC==gcc，将前面的“#”注释符号去掉；然后打开 mingwenv.cmd ，按顺序先后执行 “./configure” and “make” 命令，编译完成后，重新对工程进行编译即可。

7.4 节点移动轨迹

7.4.1 问题描述

当节点进行移动的时候，想观察节点的移动轨迹，那么如何进行操作？

7.4.2 解决办法

完成这个问题，需要进行两步操作，分别在.ned .ini 文件中：

1）在整个网络拓扑 ned 文件中，需要添加必要的路径和模块：

```
/*调用inet中的库函数*/
import inet.visualizer.integrated.IntegratedCanvasVisualizer;
import inet.visualizer.contract.IIntegratedVisualizer;

/* 添加visualizer模块 */
visualizer: <default("IntegratedCanvasVisualizer")> like IIntegratedVisualizer if
     hasVisualizer() {
    parameters:
        @display("p=50103.34,27751.85");
}
```

morekeywords

2）然后在.ini 文件中进行配置

```
*.visualizertype = "IntegratedOsgVisualizer"
*.hasVisualizer = true
*.visualizer.mobilityVisualizer.moduleFilter = "**"
*.visualizer.mobilityVisualizer.displayMovementTrails = true
*.visualizer.mobilityVisualizer.movementTrailLineColor = "dark"
*.visualizer.mobilityVisualizer.movementTrailLineStyle = "solid"
*.visualizer.mobilityVisualizer.movementTrailLineWidth =  2
*.visualizer.mobilityVisualizer.trailLength = 100000
*.visualizer.mobilityVisualizer.displayOrientations = true
*.visualizer.mobilityVisualizer.displayVelocities = true
```

morekeywords

通过以上两步就可以实现完成轨迹的显示

7.5 在新构建模块，函数调用

7.5.1 问题描述

当构建新的 Ned 模块时，如何调用 inet 中的指定模块？

7.5.2 解决办法

完成这个问题，需要完成下面的操作：1）在调用 inet 模块时，首先利用 import 函数从 inet 中导入指定模块即可，而对应的.cc 文件和.h 文件并不需要进行调用，直接利用既可以完成；

7.6 文件引用

7.6.1 问题描述

在新建.h 和.cc 文件时，如果引用调用同一个文件的其他.cc 文件，该如何解决该问题

7.6.2 解决办法

完成这个问题，需要完成下面的操作：1）工程文件的 perpetual 的设置中，将文件的路径添加进去即可实现相邻文件的调用；

7.7 msg 文件调用外部函数

7.7.1 问题描述

当在.msg 文件中调用其他消息时，

```
1 import inet.common.INETDefs;
```

morekeywords

将会提示下面的错误：

```
1 Error: syntax error, unexpected NAME, expecting $end
```

morekeywords

7.7.2 解决办法

完成这个问题，需要完成下面的操作：1）工程文件的 properties 中，选择 oment++/Makemake/src floder /options/Custom，2）添加下面的程序：

```
1 MSGC:=$(MSGC) --msg6
```

morekeywords

7.8 本章小结

本章主要对 OMNeT++ 中常见错误进行配置，后续尝试将解决办法进行补充。

第 8 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 = $20 \times 10 \times 10^3 / 1000 = 2 \times 10^5$ 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。