〇、NS-3简介

◆ 什么是NS-3?

- NS-3是一款用于模拟计算机网络的软件。
- NS-3是用C++编写的开源项目。
- NS-3没有图形用户界面(GUI)，只能通过脚本的方式使用它。
- NS-3的脚本支持C++和Python两种语言。（后面的教程只介绍C++语言）
- NS-3更像是一个程序库，它为模拟网络提供了丰富的API接口。
- NS-3的主要运行平台是Linux系统，虽然Windows用户也可以使用NS-3，但是部分功能无法使用。

一、NS-3的安装

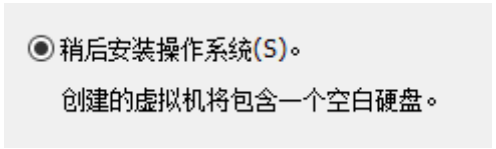
- **安装虚拟机vmware**

  下载地址：https://www.vmware.com/cn/products/workstation-pro/workstation-pro-evaluation.html

一、NS-3的安装

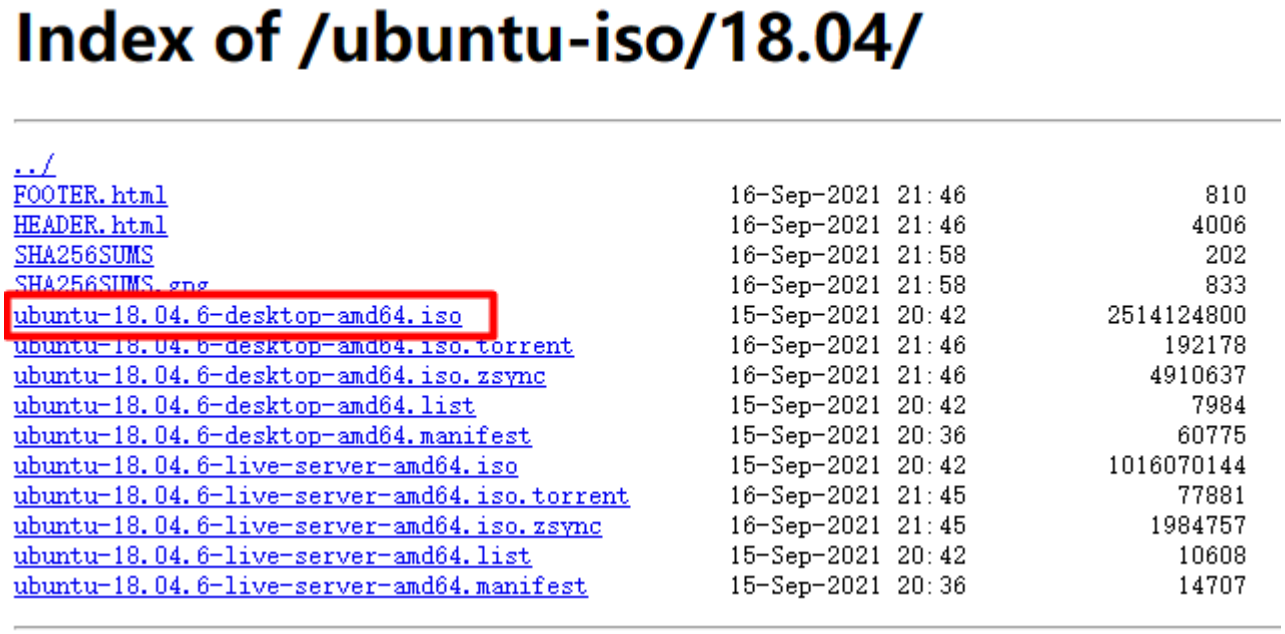
- **创建虚拟机**

  选择“稍后安装操作系统”，其余步骤选择默认选项即可。



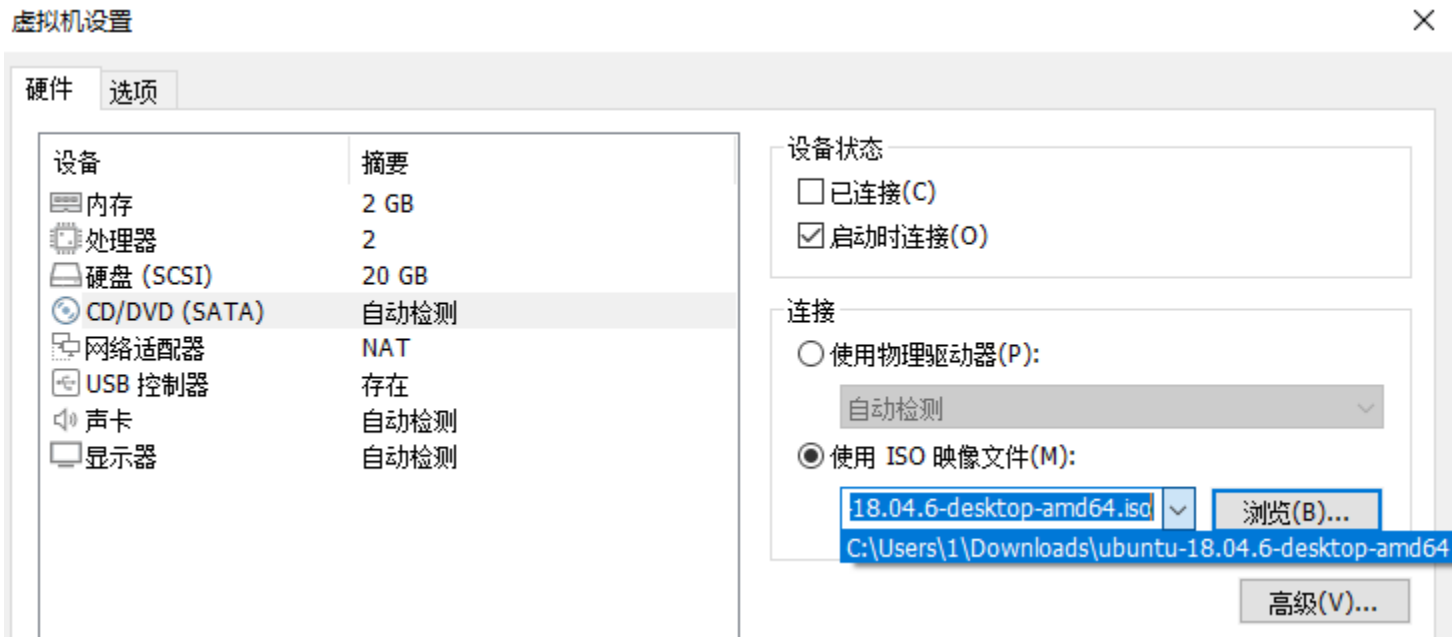
- **下载Ubuntu操作系统**（推荐使用18.04版本）

  下载地址：https://mirror.umd.edu/ubuntu-iso/18.04/



一、NS-3的安装

- **设置虚拟机的CD/DVD，设为Ubuntu的镜像文件**



- **打开虚拟机，根据提示完成Ubuntu系统的安装**

  安装完成后，在设置中将虚拟机的CD/DVD选项重新设为“**使用物理驱动器**”

一、NS-3的安装

◆ 参考步骤
- 安装open-vmtools，使屏幕可以自适应放缩和与主机之间复制粘贴：
  `sudo apt-get install open-vm-tools-desktop`
- 重启虚拟机
  `reboot`

◆ 其它问题
- 安装依赖包时有未能满足的依赖关系：

`sudo vim /etc/apt/sources.list`

在sources.list文件中添加以下内容（阿里云源）：

```
deb https://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb-src https://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse

deb https://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse
deb-src https://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse

deb https://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse
deb-src https://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse

# deb https://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse
# deb-src https://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse

deb https://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse
deb-src https://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse
```

最后更新源：`sudo apt-get update`

一、NS-3的安装

后续步骤参考：https://blog.csdn.net/yangzhenyu2/article/details/116205406

◆ **安装依赖库**

打开终端，输入以下命令：

- sudo apt-get install gcc g++ python python3
- sudo apt-get install gcc g++ python python3 python3-dev
- sudo apt-get install python3-setuptools git mercurial
- sudo apt-get install qt5-default mercurial
- sudo apt-get install gir1.2-goocanvas-2.0 python-gi python-gi-cairo python-pygraphviz python3-gi python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0 ipython ipython3
- sudo apt-get install openmpi-bin openmpi-common openmpi-doc libopenmpi-dev
- sudo apt-get install autoconf cvs bzr unrar
- sudo apt-get install gdb valgrind
- sudo apt-get install uncrustify
- sudo apt-get install doxygen graphviz imagemagick

一、NS-3的安装

◆ **安装依赖库**

打开终端，输入以下命令：

（接上页）

- sudo apt-get install texlive texlive-extra-utils texlive-latex-extra texlive-font-utils dvipng latexmk
- sudo apt-get install python3-sphinx dia
- sudo apt-get install gsl-bin libgsl-dev libgsl23 libgslcblas0
- sudo apt-get install tcpdump
- sudo apt-get install sqlite sqlite3 libsqlite3-dev
- sudo apt-get install libxml2 libxml2-dev
- sudo apt-get install cmake libc6-dev libc6-dev-i386 libclang-6.0-dev llvm-6.0-dev automake
- sudo apt-get install libgtk2.0-0 libgtk2.0-dev
- sudo apt-get install vtun lxc uml-utilities
- sudo apt-get install libboost-signals-dev libboost-filesystem-dev

一、NS-3的安装

- **在官网下载NS-3**（推荐3.27版本）

  下载的文件为一个压缩包，解压后用终端打开文件夹ns-allinone-3.27，

  执行编译命令：

  - sudo ./build.py

- **编译完成后，执行如下命令测试：**

  - sudo ./waf --run hello-simulator

  如果正确输出“Hello Simulator”则完成安装。否则继续执行下列命令，

  执行完毕后再重新测试hello-simulator：

  - cd ns-3.27
  - sudo ./waf clean
  - sudo ./waf -d debug --enable-example --enable-tests configure
  - sudo ./waf

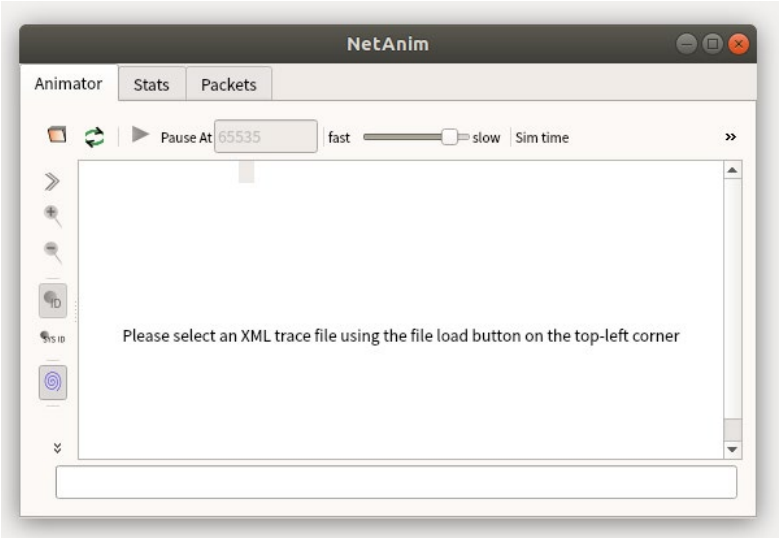
一、NS-3的安装

- **编译NetAnim可视化部分**

  进入netanim-3.108目录，执行下列命令：

  - cd ../
  - cd netanim-3.108
  - sudo make clean
  - sudo qmake NetAnim.pro
  - sudo make

- **执行“sudo ./NetAnim”打开NetAnim，至此安装完成**



二、NS-3的代码编译与运行

◆ 在安装NS-3时，我们执行了“sudo ./build.py”命令完成了NS-3的编译。

此时可以使用waf工具执行ns-3.27目录下的所有项目，例如：

- sudo ./waf --run hello-simulator
- sudo ./waf --run first

◆ 上面的两个项目hello-simulator、first的项目源代码都位于example/tutorial目录下。

NS-3的目录结构：

- example目录包含了大量的NS-3例程。
- src目录包含了NS-3各个模块的源代码。
- build目录是编译后的目标文件和可执行文件。

◆ 执行脚本的命令格式为：sudo ./waf --run <项目名>

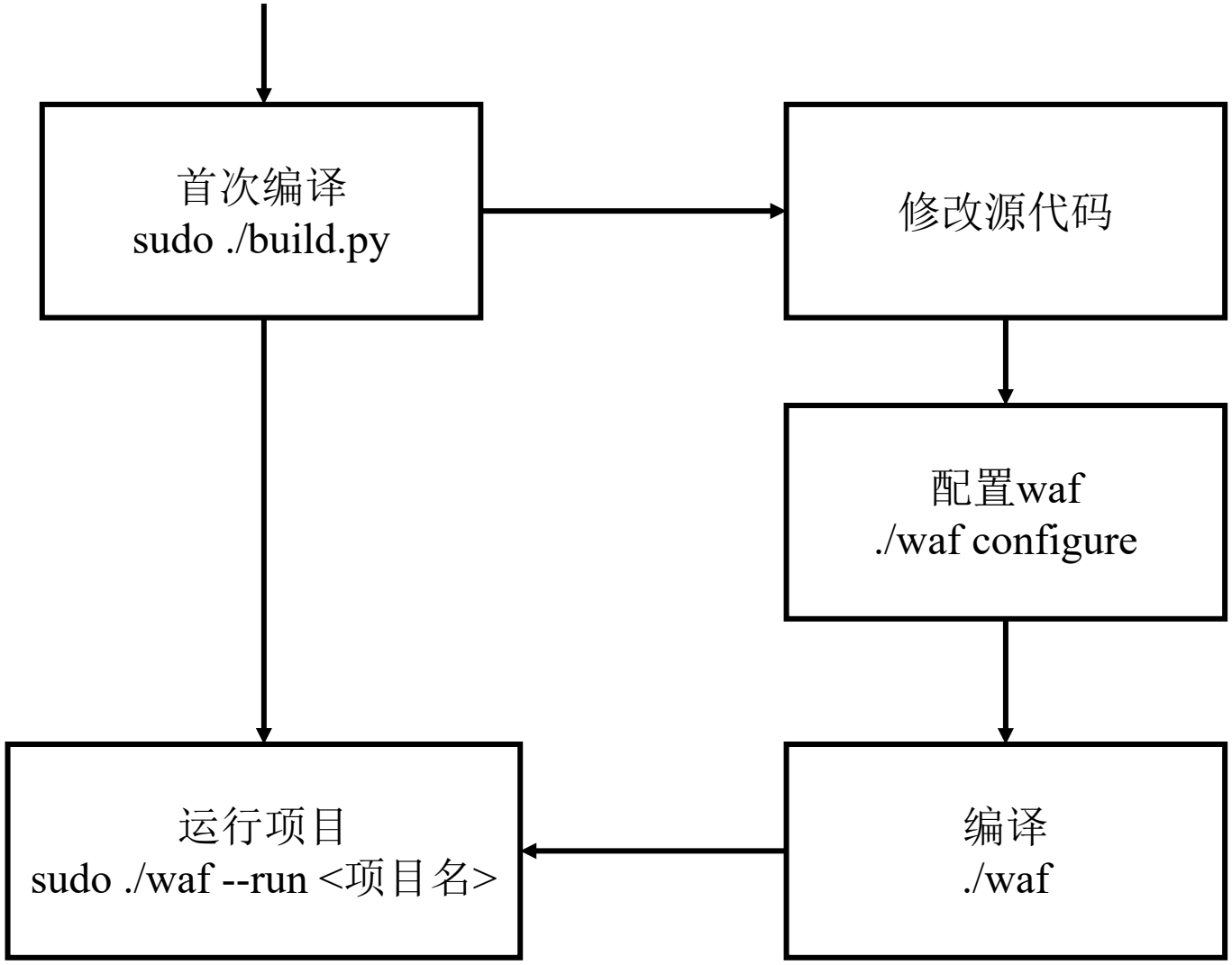
需要注意：

- 使用waf执行项目时终端必须位于**ns-3.27目录**下。
- 执行脚本时没有指定项目的路径，这是因为可执行文件在build目录下，waf命令知道各个可执行文件的路径。
- sudo ./waf --run **examples/tutorial/**hello-simulator 的执行与不加路径的命令等价。

二、NS-3的代码编译与运行

- 如果要执行自己编写的项目，则需要对项目进行编译后才能执行。
  编译命令为：sudo ./waf
- 编译命令是可以配置的，使用命令./waf configure进行配置，例如在安装NS-3时使用了如下的命令：
  - sudo ./waf -d debug --enable-example --enable-tests configure
  - 选项“--enable-example”表示编译时包含所有的example项目
  - 选项“--enable-tests”表示编译时包含所有的test项目(在src目录下)
- 默认情况下，waf命令是不编译example和test项目的，意味着如果修改了这些项目的源代码，则必须添加上述的选项。
- 用户自己编写的项目通常会放在**scratch目录**下，这个目录默认包含在./waf的编译范围内，并且不需要浪费时间来编译example和test项目。

二、NS-3的代码编译与运行



二、NS-3的代码编译与运行

◆ 在scratch文件夹下创建一个新文件myhello.cc，输入如下代码：

```
#include "ns3/core-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("HelloSimulator");
int main (int argc, char *argv[])
{
    NS_LOG_UNCOND ("My Hello");
}
```

◆ 输入命令：sudo ./waf 进行编译。

◆ 输入命令：sudo ./waf --run myhello 运行程序。

◆ 注意：如果只是**修改源代码**的情况下，可以直接执行./waf --run 命令同时完成编译和运行。

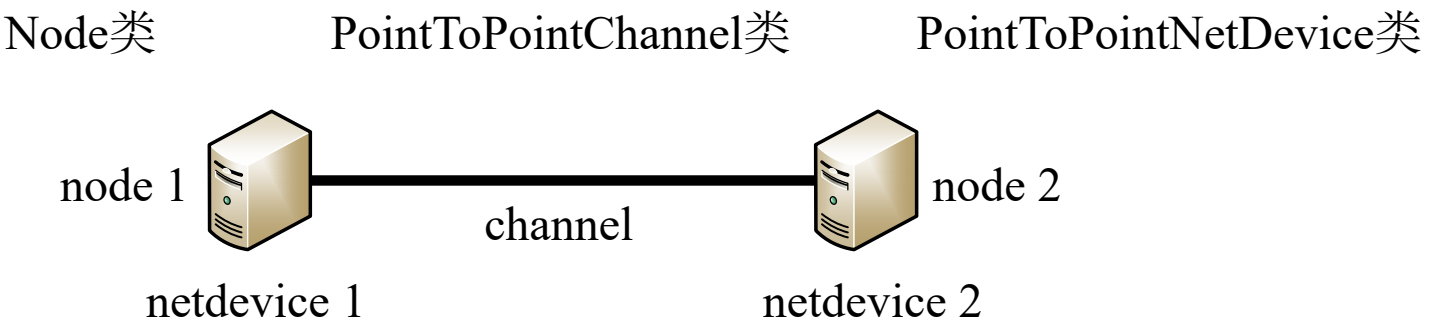
- 例如：将上述main函数的代码修改为：

```
NS_LOG_UNCOND ("My Hello!!!");
```

直接执行sudo ./waf --run myhello 也可以完成编译并正确输出结果。

三、一个NS-3的例程：third.cc

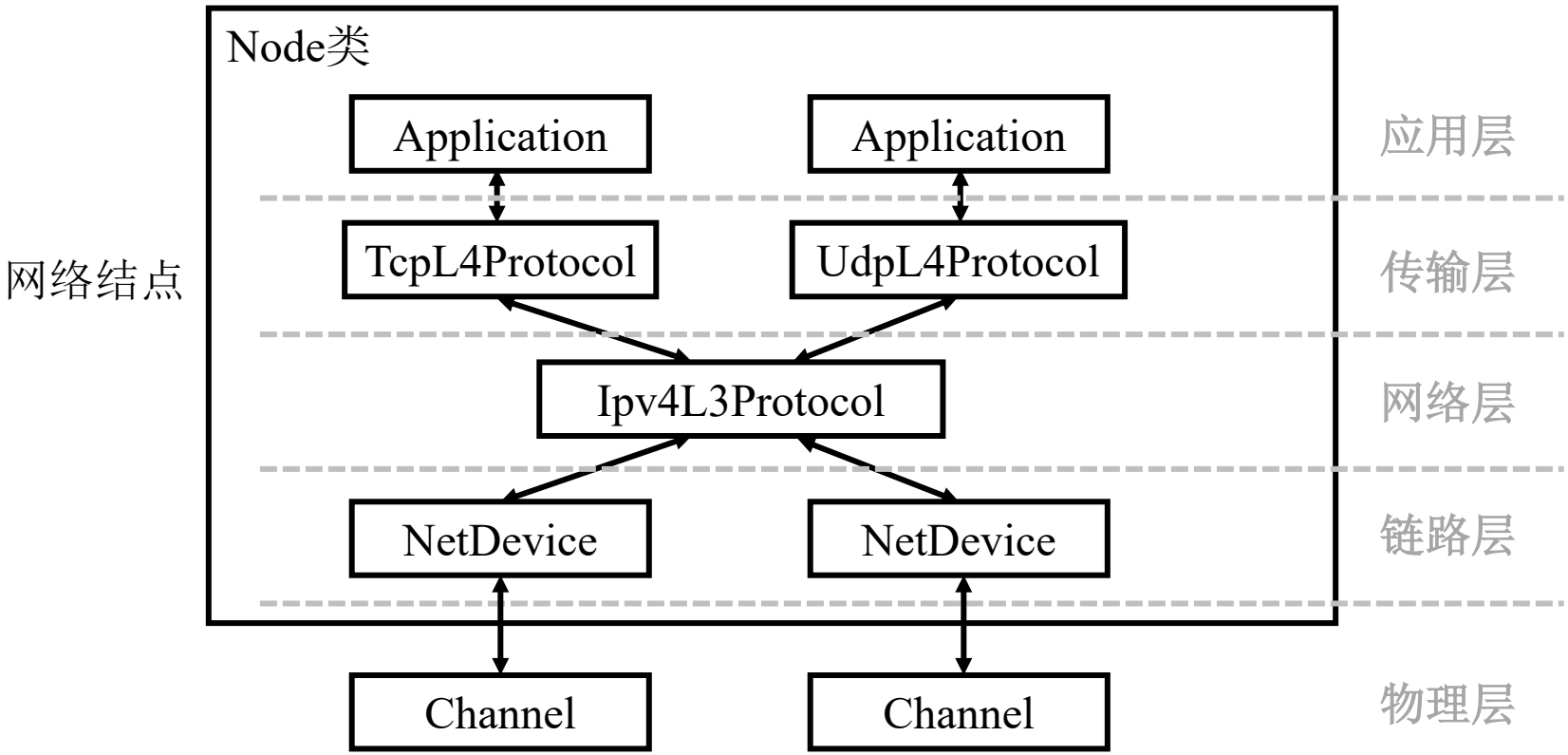
- 我们从一个NS-3的例程third.cc开始介绍NS-3网络模拟的流程。该文件位于example/tutorial目录下。
- 在理解NS-3代码时，通常从NS-3的**类**入手。一个网络模拟场景是由结点、协议、信道等等网络元素组成，在NS-3中，每一个网络元素都对应于一个**类**。这些类定义了网络元素的行为，而在模拟中真正运行的是类的**对象**。
  - 例如下图中的网络拓扑，图中的两个结点通过PPP协议进行通信。图中的网络元素有2个结点、2个网络设备和一个PPP信道。涉及的类分别是Node类、PointToPointNetDevice类和PointToPointChannel类。



注意node1和netdevice1的区别？

三、一个NS-3的例程：third.cc

◆ **Node类**：一个网络结点涉及的类非常多，见下图。



- Node类采用的是**TCP/IP参考模型**。
- 优点：几乎所有的类都与Node类相关。如果想获取某一个协议层的信息，从Node类出发很容易得到。

三、一个NS-3的例程：third.cc

- third.cc要模拟的网络拓扑如图所示：

```
// Default Network Topology
//
//   Wifi 10.1.3.0
//                  AP
//  *     *     *     *
//  |     |     |     |    10.1.1.0
// n5    n6    n7    n0 -------------- n1   n2   n3   n4
//                    point-to-point  |    |    |    |
//                                    ================
//                                      LAN 10.1.2.0
```

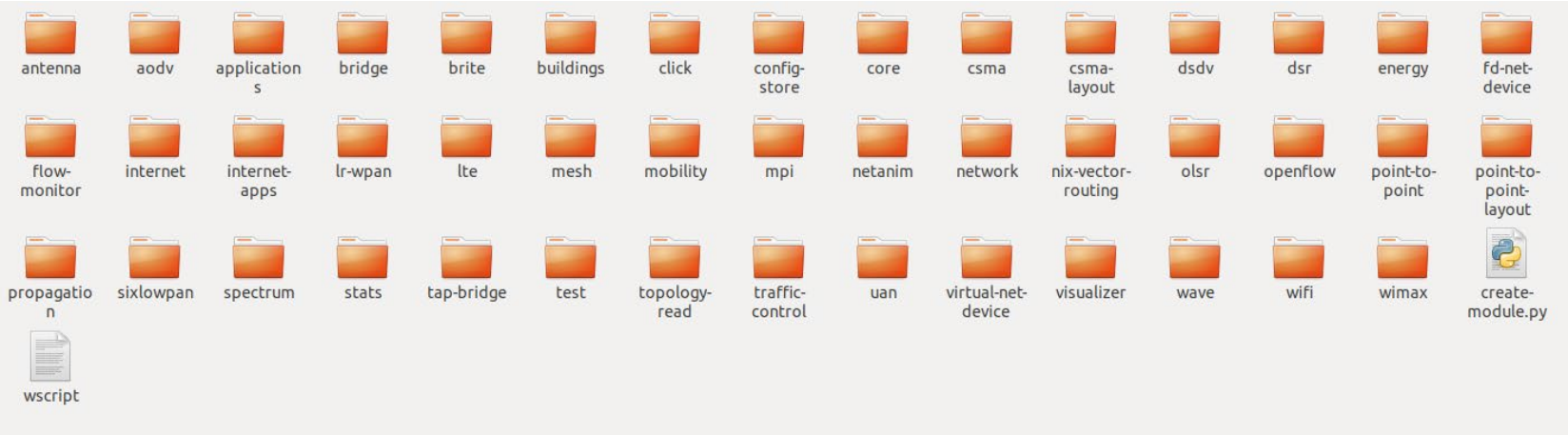
- third.cc要模拟的是一个无线网络、PPP和CSMA有线网络混合的场景。（n0是AP结点）
- 再根据要模拟的场景确定插入的头文件。

三、一个NS-3的例程：third.cc

◆ 头文件

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/csma-module.h"
#include "ns3/mobility-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
```

◆ 每一个头文件名字的后缀都有“module”，这表示每个头文件都代表一个模块，这些模块的源代码在src文件夹下面（每一个文件夹都代表一个模块）。



三、一个NS-3的例程：third.cc

◆ 命令行参数

```
int main (int argc, char *argv[])
{
  //设置参数的默认值
  bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  bool tracing = false;

  //添加命令行参数
  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);  //读取命令行参数
```

◆ 上面的代码添加了4个命令行参数，分别是nCsma、nWifi、verbose和tracing。

◆ 在waf命令中使用命令行参数，如：./waf--run "third --nWifi=18"

三、一个NS-3的例程：third.cc

◆ Log系统

```
NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

.......
if (verbose)
  {
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }
```

- 以上的代码配置了**Log系统**，使用Log系统可以输出数据。在后面会进一步介绍Log系统并且解释上面的代码。
- 其中，NS_LOG_COMPONENT_DEFINE函数定义了一个日志组件。LogComponentEnable函数允许组件输出Log信息。
- 在myhello.cc文件中的NS_LOG_UNCOND函数的功能就是将内容输出到屏幕上：

```
NS_LOG_UNCOND ("My Hello");
```

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Node容器

```
//   Wifi 10.1.3.0
//                  AP
//  *    *    *    *
//  |    |    |    |    10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                   point-to-point  |    |    |    |
//                                   ================
//                                     LAN 10.1.2.0
```

```
NodeContainer p2pNodes;
p2pNodes.Create (2);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);
```

◆ NodeContainer类即为**Node容器类**，可以方便管理Node类的对象。这里创建了4个Node容器，用于管理不同功能的node结点。

- Create函数用于批量创建结点。
- Get函数用于获取Node容器内的结点。p2pNodes.Get(1)表示获取p2pNodes容器的第二个结点。
- Add函数用于向容器内添加已创建的结点。（一个结点可以同时处于多个容器内）

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--PPP网络

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue
("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

◆ PointToPointHelper类属于**助手类**，助手类的类名都以Helper结尾。助手类可以屏蔽实现细节从而降低代码编写的复杂度。

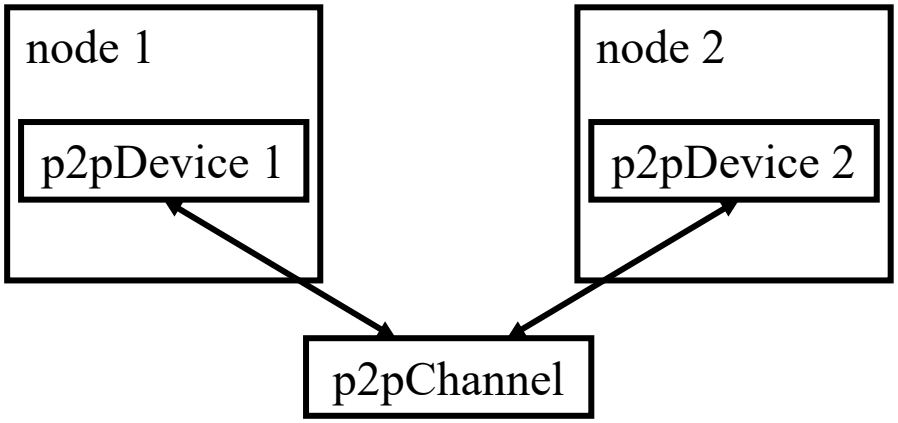
- 每一个模块都有相应的助手类，其源代码在src/<模块名>/helper目录下。
- 上面的代码中，先使用pointToPoint设置了DataRate、Delay的值，再调用**Install函数**在p2pDevices容器中创建网络设备和信道，并将网络设备和信道连接。创建的网络设备的个数与p2pNodes容器内node结点的个数对应。
- 这两个容器中**相同索引号**的netDevice和node是关联的（**与同一个结点容器相关联的所有容器中，相同索引号的对象属于同一个结点**）。

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--PPP网络

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue
("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

◆ 执行完上述的代码后构建的拓扑为（只考虑p2pNodes这一个Node容器）：



三、一个NS-3的例程：third.cc

◆ **属性**：简单的讲，属性就是用户可配置的**参数**。例如下面的代码中，DataRate和Delay就是属性，DataRate代表PPP网络设备的数据传输速率，Delay表示PPP信道的传播延迟。

```
pointToPoint.SetDeviceAttribute ("DataRate", StringValue
("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

- 显然，属性的值会影响到网络的性能和模拟的结果。

◆ **配置属性**：上面的代码中，通过调用了两个函数来配置助手类的属性值，然后通过该助手类创建的对象会被自动配置为助手类的属性值。

- 也就是说，上面的代码配置的是助手类的属性值DataRate=5Mbps，Delay=2ms，则使用该助手类创建的两个p2pDevice的属性值会被自动配置为DataRate=5Mbps,Delay=2ms，自动配置属性的操作是在Install函数内完成的。
- 用助手类可以批量配置属性，但是必须**在对象创建之前配置**。可以将助手类看作一个模版，创建的两个对象是按照同一个模版来的。

三、一个NS-3的例程：third.cc

- 如何创建两个属性值不一样的p2pDevice？

  例如，要创建2个p2pDevice，但是其中一个的DataRate为5Mbps，另一个为10Mbps。

  代码如下：

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue
("5Mbps"));

NetDeviceContainer p2pDevices_1;
p2pDevices_1 = pointToPoint.Install (p2pNodes);

pointToPoint.SetDeviceAttribute ("DataRate", StringValue
("10Mbps"));

NetDeviceContainer p2pDevices_2;
p2pDevices_2 = pointToPoint.Install (p2pNodes);
```

- 使用助手类配置属性的适用情况：对象尚未创建、批量创建对象。

三、一个NS-3的例程：third.cc

```
//   Wifi 10.1.3.0
//                  AP
//  *     *     *     *
//  |     |     |     |     10.1.1.0
// n5    n6    n7    n0 -------------- n1    n2    n3    n4
//                   point-to-point  |     |     |     |
//                                   ================
//                                     LAN 10.1.2.0
```

◆ 创建网络拓扑--CSMA网络

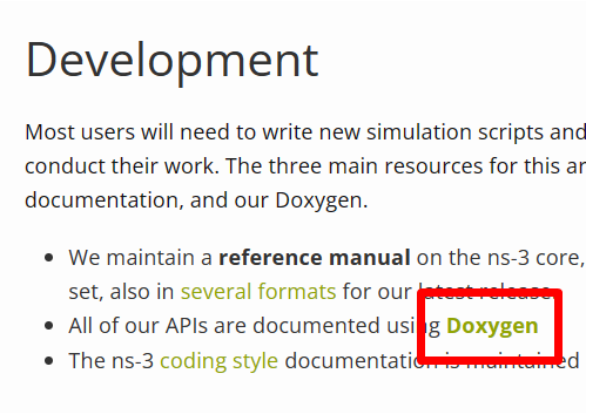
```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds
(6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

- 注意在设置DataRate属性值的时候使用的函数是

  SetChannelAttribute而不是SetDeviceAttribute。因为

  CsmaNetDevice类没有DataRate属性，而CsmaChannel有该属性。

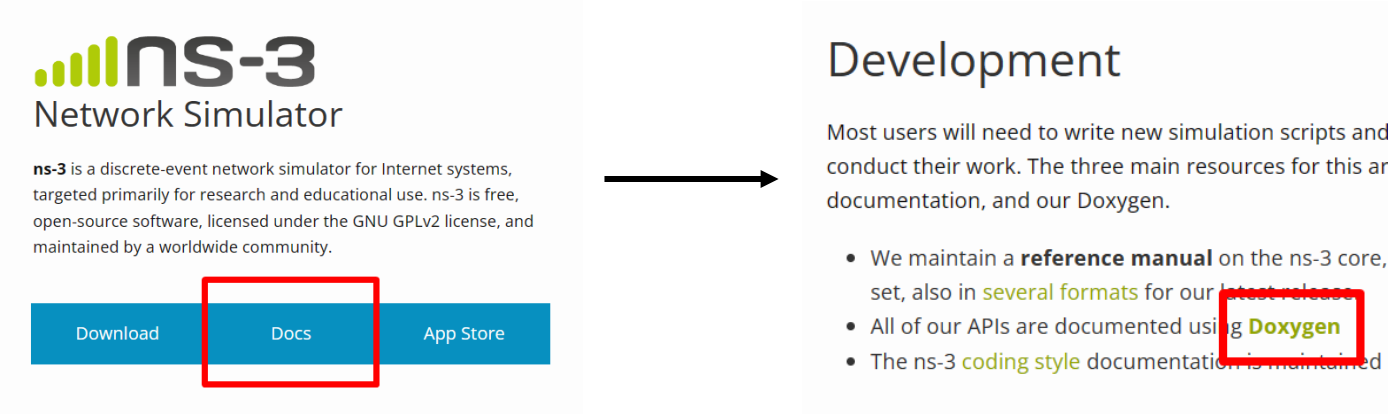
◆ 在**Doxygen**中查询NS-3的API文档：



三、一个NS-3的例程：third.cc

◆ 在**Doxygen**中查询NS-3的API：

https://www.nsnam.org/docs/release/3.27/doxygen/index.html



◆ 第二种方式：

- 在虚拟机中执行命令：./waf --doxygen 编译API文档。（编译时间较长）
- 编译完成后，打开ns-3.27/doc/html文件夹，打开**annotated.html**文件即可查询API（这个文件夹下有约十万个文件，打开文件夹后可能需要加载一段时间）。

三、一个NS-3的例程：third.cc

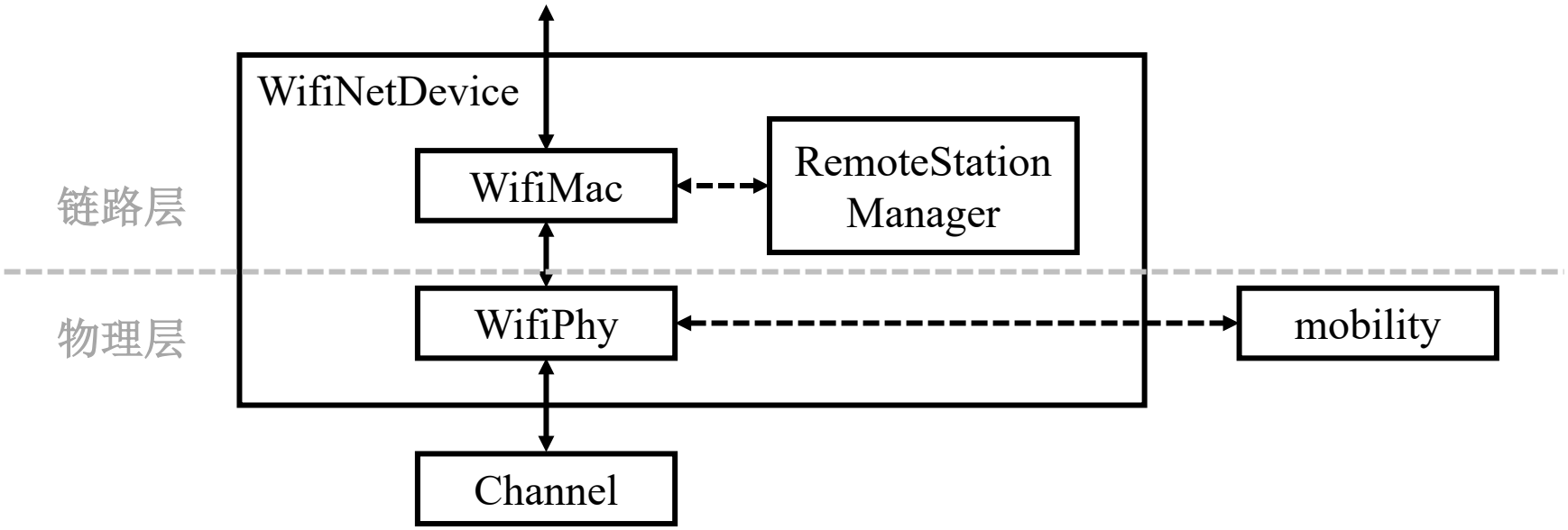
◆ 创建网络拓扑--Wifi网络

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

Wifi模块的架构：



- 可以将上图的架构划分为链路层和物理层。
  - WifiNetdevice类：可以看作网络层与Wifi通信的接口。
  - RemoteStationManager类：控制数据速率。

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

根据Wifi模块的架构，我们可以将代码划分为**物理层**和**链路层**两个部分。根据代码的顺序，先进行的是物理层（即Channel类和WifiPhy类）的配置：

```
YansWifiChannelHelper channel =
    YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
```

- YansWifiChannelHelper类和YansWifiPhyHelper类分别用于Channel类和WifiPhy类的配置。
- Channel类需要配置**传播延迟模型**和**损耗模型**，这里调用Default函数表示使用默认的模型。
- WifiPhy类需要配置**误码率模型**，本例中也是使用了默认模型。
- **Create**函数创建了一个信道，**SetChannel**函数则将这个信道与phy(助手类的对象)进行关联，那么之后用phy创建的Wifi结点都会与这个信道连接。

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

下一步进行的是链路层（即WifiMac类和RemoteStationManager类）的配置：

```
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
```

- WifiHelper类和WifiMacHelper类分别用于RemoteStationManager类和WifiMac类的配置。
- RemoteStationManager类用于动态控制WiFi的最佳发送速率，WifiHelper助手类使用**SetRemoteStationManager**函数设置速率控制。上述代码使用的是Auto Rate Fallback（ARF）算法。

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

下一步进行的是链路层（即WifiMac类和RemoteStationManager类）的配置：

```
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
```

- WifiMacHelper助手类的一个重要参数是**SSID(服务集标识符)**，SSID决定了结点所属的服务集，AP与移动结点必须属于同一服务集才能通信。
- SetType函数用于配置WifiMacHelper助手类，“ns3::StaWifiMac”是一个类，表示移动结点，ActiveProbing是StaWifiMac类的属性值。

- **ActiveProbing**: If true, we send probe requests. If false, we don't.NOTE: if more th
  STA, otherwise all the STAs will start sending probes at the same time resulting in co
  - Set with class: **BooleanValue**
  - Underlying type: bool
  - Initial value: false

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

下一步进行的是网络设备的安装：

```
WifiHelper wifi;
WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

- wifi(WiFiHelper助手类的对象)调用**Install函数**创建staDevice。

  （注意前面的Wifi模块架构，必须先用助手类配置好WifiPhy、WifiMac和RemoteStationManager后才能创建WifiNetDevice）。

- 第二个Install安装的是AP结点的网络设备，至此Wifi网络设备安装完成。

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

补充：Wifi协议标准的设置

- 不同的Wifi协议直接决定了网络的频率和数据发送速率。
- 由WifiHelper::**SetStandard()**完成设置。SetStandard()函数的定义在Doxygen的查询如下：

void
ns3::WifiHelper::SetStandard (enum WifiPhyStandard standard) virtual

Parameters

standard the phy standard to configure during installation

This method sets standards-compliant defaults for **WifiMac** parameters such as sifs time, slot time, timeout values, etc., based on the standard selected. It results in WifiMac::ConfigureStandard(standard) being called on each installed mac object.

The default standard of 802.11a will be applied if **SetStandard()** is not called.

enum ns3::WifiPhyStandard

Identifies the PHY specification that a Wifi device is configured to use.

Enumerator	
WIFI_PHY_STANDARD_80211a	OFDM PHY for the 5 GHz band (Clause 17)
WIFI_PHY_STANDARD_80211b	DSSS PHY (Clause 15) and HR/DSSS PHY (Clause 18)
WIFI_PHY_STANDARD_80211g	ERP-OFDM PHY (Clause 19, Section 19.5)
WIFI_PHY_STANDARD_80211_10MHZ	OFDM PHY for the 5 GHz band (Clause 17 with 10 MHz channe bandwidth)
WIFI_PHY_STANDARD_80211_5MHZ	OFDM PHY for the 5 GHz band (Clause 17 with 5 MHz channel bandwidth)
WIFI_PHY_STANDARD_holland	This is intended to be the configuration used in this pape

三、一个NS-3的例程：third.cc

- 这里要强调的是，WifiPhyStandard给定的参数中没有**802.11p**协议。若要配置该协议，需要使用**Wifi80211pHelper**助手类代替WifiHelper助手类。
  - Wifi模块的配置分为物理层和链路层，物理层配置的代码无需变动，而链路层配置需要使用新的助手类：**NqosWaveMacHelper**或**QosWaveMacHelper**，其区别为前者不支持QoS，后者支持QoS。
  - NqosWaveMacHelper和QosWaveMacHelper是WifiMacHelper的子类，如果只使用WifiMacHelper配置链路层，则创建的对象的参数QosSupported会默认设置为false。
  - 如果仍然使用WifiMacHelper会报错：创建新文件mythird.cc，只将WifiHelper替换为Wifi80211pHelper，尝试编译运行。注意添加头文件：#include "ns3/wave-module.h"。
  - 注意，目前NS-3的最新版本3.40可以直接使用WifiHelper配置802.11p协议，具体内容参阅3.40版本的API手册。

三、一个NS-3的例程：third.cc

◆ 配置802.11p协议：以下的代码基于third.cc例程修改：

- 原代码：

```
WifiHelper wifi;
WifiMacHelper mac;
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid));
```

- 修改后：

```
Wifi80211pHelper wifi;
NqosWaveMacHelper mac;
//mac.SetType ("ns3::StaWifiMac",
//             "Ssid", SsidValue (ssid),
//             "ActiveProbing", BooleanValue (false));
//mac.SetType ("ns3::ApWifiMac",
//             "Ssid", SsidValue (ssid));
mac.SetType ("ns3::OcbWifiMac",
             "QosSupported", BooleanValue (false));
```

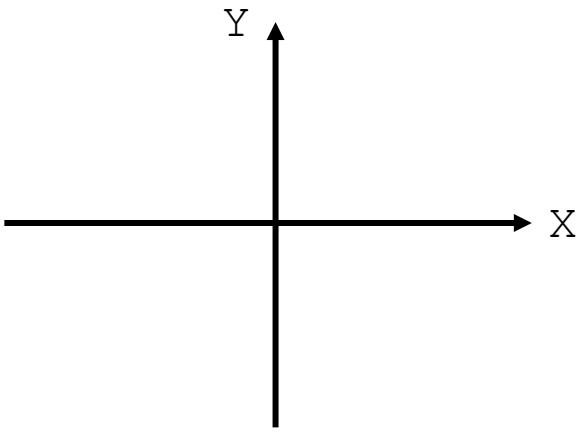
使用802.11p协议必须使用SetType函数配置且只能配置**OcbWifiMac**（或它的子类）。

三、一个NS-3的例程：third.cc

◆ **创建网络拓扑--Wifi网络**

在网络设备创建完成后，需要设置Wifi网络的**移动模型**。

- 移动模型一般在网络设备安装完毕后设置，因为需要分开设置移动结点和AP结点的移动模型。使用助手类**MobilityHelper**设置移动模型。
- 移动模型涉及结点位置表示的问题，在NS-3中使用**直角坐标系**表示结点的位置。



三、一个NS-3的例程：third.cc

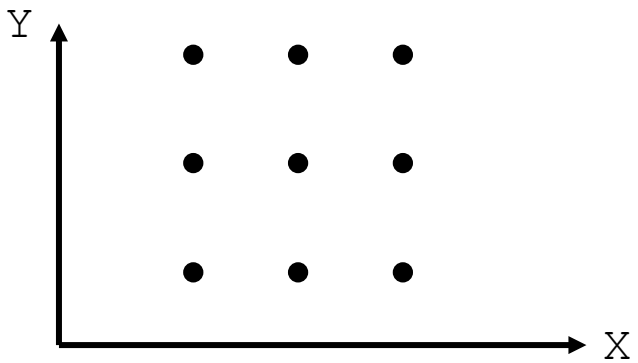
◆ 创建网络拓扑--Wifi网络

移动模型：移动结点的配置

```
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));
```

- SetPositionAllocator函数设置结点的**初始位置**。

  ns3::GridPositionAllocator表示初始位置的**设置策略**：在矩形**二维网格**上分配位置。（可以查阅PositionAllocator类的子类了解更多的设置策略）

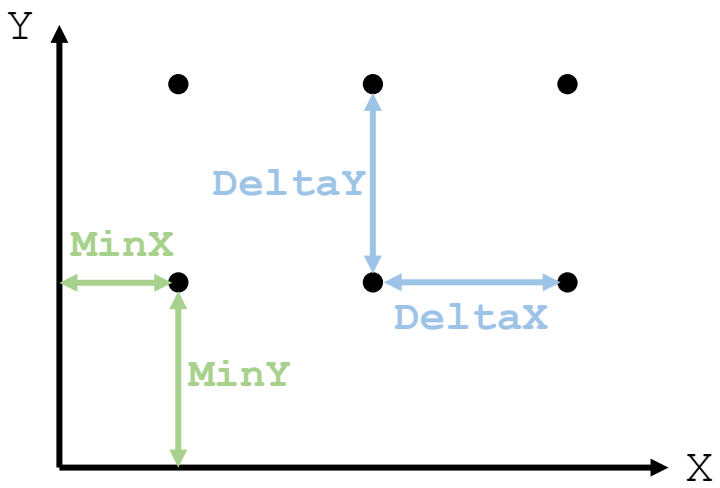


三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

移动模型：移动结点的配置

```
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));
```



- (MinX,MinY)决定的点为初始点，其余的网格点的坐标都是在X或Y坐标上加上整数倍的DeltaX或DeltaY得到的。
- LayoutType决定位置分配的方向，RowFirst则优先朝X轴方向分配位置。
- GridWidth决定每一行的点的最大个数。

三、一个NS-3的例程：third.cc

◆ 创建网络拓扑--Wifi网络

移动模型：

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Bounds",
                           RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

- SetMobilityModel设置结点的**移动模型**，

  ns3::RandomWalk2dMobilityModel为在一个矩形内以随机速度和随机方向移动的移动模型，Bounds指定了矩形的大小。

  ns3::ConstantPositionMobilityModel为固定位置的移动模型，初始位置为(0,0)。

- Install函数则为结点安装移动模型。

◆ 至此，网络拓扑创建完毕。

三、一个NS-3的例程：third.cc

◆ **安装TCP/IP协议族：**简单来说，分为两步。

- **第一步**为结点安装TCP/IP协议栈：使用InternetStackHelper助手类安装。

```
InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);
```

- **第二步**为结点分配IP地址。以PPP网络设备为例（其余同理）：

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
```

  - SetBase函数设置起始地址为10.1.1.0，网络掩码为255.255.255.0。
  - Assign函数为结点分配地址，分配的地址从**10.1.1.1, 10.1.1.2,.....**开始依次分配。
  - 注意这里是给网络设备分配IP地址，因此一个结点可以有多个IP地址。

三、一个NS-3的例程：third.cc

◆ 安装应用程序

NS-3中应用层对应Application类，其子类定义了各种的应用程序，每种应用程序都有不同的**分组收发行为**。举例：

- UdpEchoServer和UdpEchoClient：客户端发送数据包给服务器后，服务器会返回一个相同大小的数据包给客户端。
- OnOffApplication：该应用程序有两种模式：On/Off，在On模式下应用程序会向一个指定的目的地（由套接字决定）发送数据包，而在Off模式下则不会产生流量。
- BulkSendApplication：应用程序会发送尽可能多的流量，并尝试达到最大带宽。

使用哪种应用程序取决于具体的应用场景。在third.cc中使用的是UdpEchoServer和UdpEchoClient，相应的使用UdpEchoServerHelper和UdpEchoClientHelper助手类。

三、一个NS-3的例程：third.cc

◆ 安装应用程序

第一步，安装服务器端应用程序：

```
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps =
    echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

- 第一行代码创建了一个助手类对象，参数“9”表示监听9号端口。
- 在serverApps容器中为第nCsma个结点安装服务器端应用程序（注意csmaNodes容器中有nCsma+1个结点）。
- Start函数使serverApps容器中所有对象在1.0秒的时候启动，Stop函数则使其在10.0秒的时候停止。
- “Seconds(1.0)”表示1秒，在NS-3中提供的**时间单位**还有：MilliSeconds、MicroSeconds、NanoSeconds、Minutes、Hours等等（这些都是类），使用方法与Seconds相同。

三、一个NS-3的例程：third.cc

◆ 安装应用程序

第二步，安装客户端应用程序：

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps =
  echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

- 第一行代码创建了一个助手类对象，创建时需要添加服务器的IP地址和端口（前文中提到，**与同一个结点容器相关联的所有容器中，相同索引号的对象属于同一个结点**，因此在csmaInterfaces容器中的第nCsma个对象属于csmaNodes容器中的第nCsma个结点）。
- 第2-4行代码设置了助手类对象的属性值MaxPackets、Interval、PacketSize，分别代表：最大发送分组个数、分组发送间隔和分组字节数。接下来只在一个结点中安装客户端，并设置启动、停止时间。

三、一个NS-3的例程：third.cc

◆ 设置路由

```
// Default Network Topology
//
//   Wifi 10.1.3.0
//                  AP
//  *     *     *     *
//  |     |     |     |    10.1.1.0
// n5    n6    n7    n0 -------------- n1   n2   n3   n4
//                    point-to-point  |    |    |    |
//                                    ================
//                                      LAN 10.1.2.0
```

- 有线子网之间通信时，需要使连接多个子网的结点实现**路由功能**（假设n5与n2通信则需要n1实现路由功能）。
- third例程中使用的是全局路由(GlobalRouting)协议，其内部使用的是OSPFv2算法(RFC2328)。
- 其它路由协议参考Ipv4RoutingProtocol类的子类。

三、一个NS-3的例程：third.cc

◆ 设置路由

```
// Default Network Topology
//
//   Wifi 10.1.3.0
//                  AP
//  *     *     *     *
//  |     |     |     |    10.1.1.0
// n5    n6    n7    n0 -------------- n1   n2   n3   n4
//                    point-to-point  |    |    |    |
//                                    ================
//                                      LAN 10.1.2.0
```

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

- 这一句代码就完成了路由数据库的创建和所有结点的路由表的初始化。此时，n1结点就具有了路由功能。

三、一个NS-3的例程：third.cc

◆ **采集数据：**使用NS-3进行模拟的关键的一步就是记录重要的网络行为。例如，执行sudo ./waf --run third 可以看到有以下的内容输出在终端：

```
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01794s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01794s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03371s client received 1024 bytes from 10.1.2.4 port 9
```

- 以上的内容是通过**Log系统**输出的。

◆ **Log系统：**Log系统就是一种输出系统，用来记录或调试模拟过程的网络行为。

- **Log组件：**Log组件是Log系统的最小管理单元。一个C++(.cc)文件就可以是一个Log组件。使用NS_LOG_COMPONENT_DEFINE函数可以将一个C++文件注册为Log组件，但是注册名必须**全局唯一**。注册后，此C++文件中定义的Log函数才能在其它文件中使用。

```
NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");
```

例如，在third.cc中，我们没有定义UdpEchoClientApplication组件但仍然可以使用它的Log函数。

三、一个NS-3的例程：third.cc

◆ Log系统

- 使用Log系统：LogComponentEnable函数，该函数有两个参数：第一个参数表示Log组件名称，第二个参数表示要输出的**Log等级**。该函数的功能是激活指定的Log组件，并且输出指定的Log等级的信息。

```
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

  - **Log等级**：在NS-3中定义了多个Log等级，每个Log等级输出的信息和使用场景都不一样。请在API手册中查询ns3::LogLevel中定义的所有的Log等级。
  - 注意，NS_LOG_COMPONENT_DEFINE函数并非是使用Log系统的前提条件。

- 将Log信息输出到文件中：`./waf --run third 2>log.log`

  此时会在ns-3.27目录下生成log.log文件，并把信息输入到此文件中。

三、一个NS-3的例程：third.cc

◆ Log系统

- 自定义Log输出：

  使用一系列在log.h中定义的函数输出Log信息，例如：NS_LOG_INFO函数输出Log等级为LOG_INFO的信息。

  - 在third.cc中添加以下代码：

```
LogComponentEnable ("ThirdScriptExample", LOG_LEVEL_INFO);
NS_LOG_INFO ("Hello Log");
```

  - 执行sudo ./waf --run third 查看结果：

```
Hello log
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01794s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01794s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03371s client received 1024 bytes from 10.1.2.4 port 9
```

三、一个NS-3的例程：third.cc

◆ **采集数据**的第二种方式：NS-3中为了丰富数据收集的功能，定义了EnablePcap和EnablePcapAll函数。

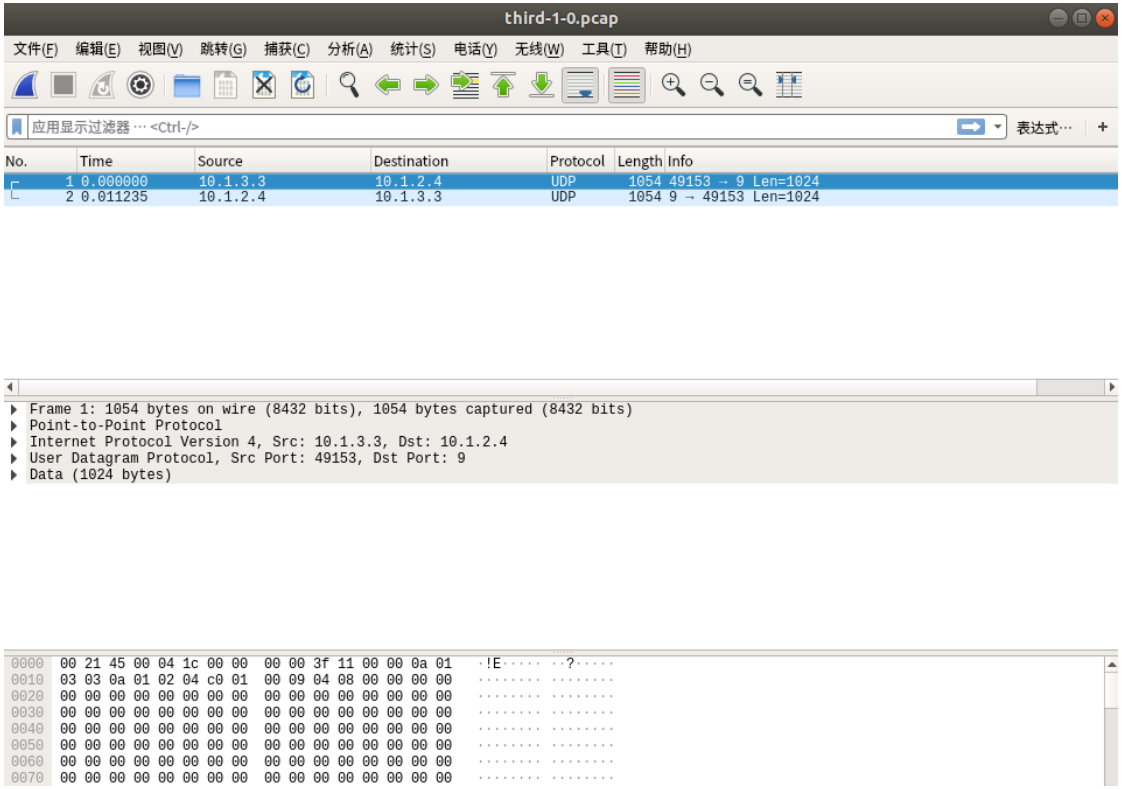
```
PointToPointHelper pointToPoint;
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
CsmaHelper csma;
pointToPoint.EnablePcapAll ("third");
phy.EnablePcap ("third", apDevices.Get (0));
csma.EnablePcap ("third", csmaDevices.Get (0), true);
```

- EnablePcap和EnablePcapAll函数属于**PcapHelperForDevice**类，这个类被PointToPointHelper、YansWifiPhyHelper、CsmaHelper继承。
- EnablePcap和EnablePcapAll函数用于记录网络设备的分组收发情况，记录的数据会保存在.pcap文件中，命名格式为：**third-结点号-设备号**。如third-0-0.pcap记录的是结点号为0的第0个设备的分组收发情况。执行：sudo ./waf --run "mythird --tracing true" 查看结果。
- 使用**wireshark**打开pcap文件查看结果。

三、一个NS-3的例程：third.cc

◆ 使用wireshark查看结果：

- 安装wireshark：sudo apt-get install wireshark
- 打开wireshark：sudo wireshark
- 用wireshark打开其中一个pcap文件：



三、一个NS-3的例程：third.cc

◆ 补充内容：如何广播分组

只需把分组的目的IP地址设为**255.255.255.255**即可。

原代码：

```
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps =
    echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
```

修改后代码：

```
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps =
    echoServer.Install (csmaNodes.Get (nCsma));
ApplicationContainer serverApps_2 =
    echoServer.Install (WifiStaNodes);
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
serverApps_2.Start (Seconds (1.0));
serverApps_2.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (Ipv4Address("255.255.255.255"), 9);
```

三、一个NS-3的例程：third.cc

◆ 补充内容：如何广播分组

从网络拓扑来看：

```
// Default Network Topology
//
//   Wifi 10.1.3.0
//                  AP
//  *     *     *     *
//  |     |     |     |    10.1.1.0
// n5    n6    n7    n0 -------------- n1    n2    n3    n4
//                    point-to-point  |     |     |     |
//                                    ================
//                                      LAN 10.1.2.0
```

上述代码做的变动为：为全部Wifi结点安装UdpEcho服务器程序，发送地址改为255.255.255.255。执行程序，观察终端输出的结果。

三、一个NS-3的例程：third.cc

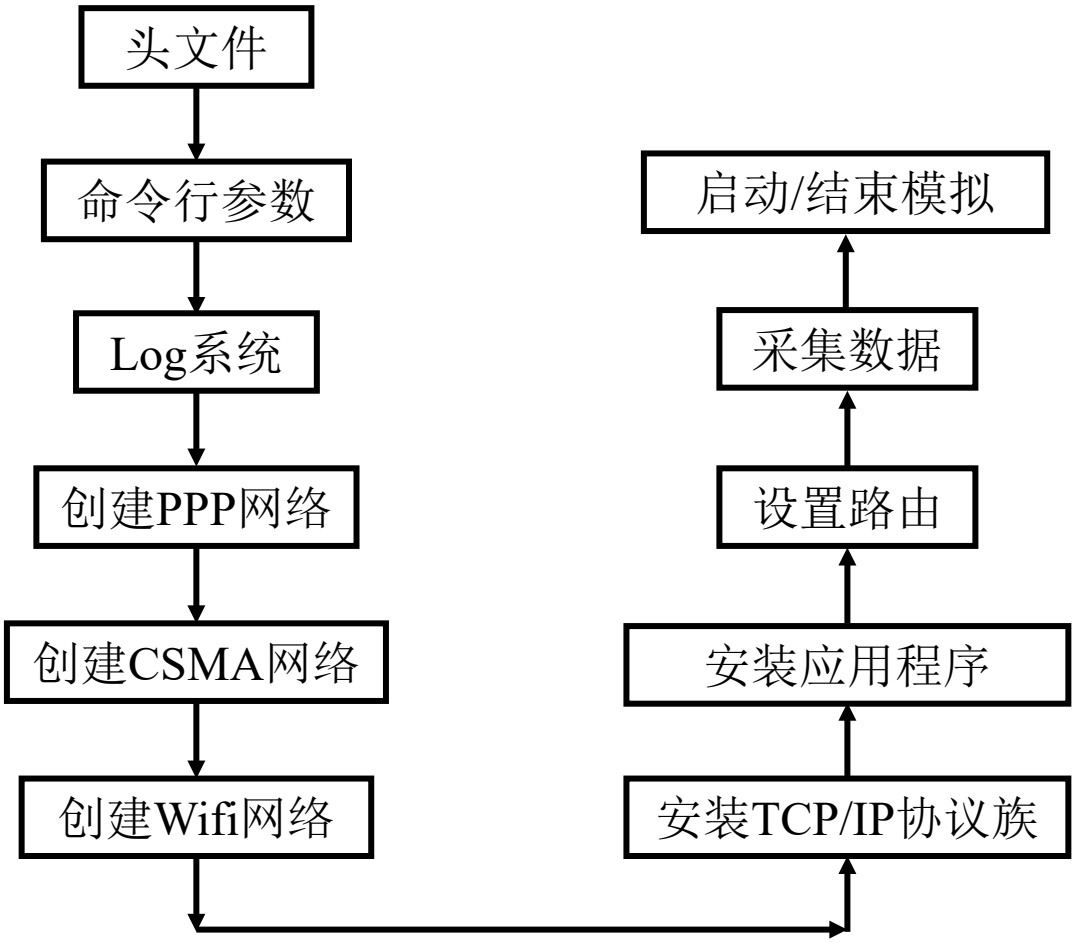
◆ 启动模拟与停止模拟

```
Simulator::Stop (Seconds (10.0));
Simulator::Run ();
Simulator::Destroy ();
```

- **Stop**函数表示模拟在第10秒停止。在third.cc中必须使用Stop设置停止时间，否则模拟不会停止。
- **Run**函数表示模拟开始执行，直到**没有其它事件发生**或**通过Stop函数设置的停止时间到达**。
- **Destroy**函数表示模拟结束执行。

四、总结

◆ NS-3脚本的流程：



四、总结

◆ NS-3重点概念：

- 类/助手类
- Node类
- 模块
- 属性
- 移动模型
- Log系统
- 智能指针
- trace变量
- 回调函数
- Object类
- 对象聚合

◆ NS-3参考资料：

官方文档：

- ns-3-tutorial.pdf
- ns-3-manual.pdf
- API手册(Doxygen)

参考书籍：

- 开源网络模拟器ns-3：架构与实践
- ns-3网络模拟器基础与应用