# NS3网络仿真及协议分析

## 实验要求

1. 利用ns3等网络仿真软件进行AODV协议及至少另一种路由协议。
2. 简要叙述评估协议的基本原理。

（3）给出评估环境、移动模式等参数的具体设置。

（4）给出评估结果的图示和分析。

## 二、相关知识

本章节主要介绍实验采用的NS3网络仿真软件的相关内容，以及在此次实验中仿真的两种Ad-Hoc路由协议，分别是AODV协议及DSDV协议，介绍一些他们的区别和相关基础知识，以便于仿真实验后的评估。最后，介绍在实验中我们采用的性能评估的参数，并解释这些参数的具体含义。

### NS3网络仿真软件

NS3(networdsimulator-version3),是面向对象的离散时间驱动的网络仿真器，它是一个全新的模拟软件并非是NS-2的升级版，它摒弃了主流网络模拟软件OPNET和NS-2的缺点，整合了它们的优点。

### 1.1 NS-3的网络组件：

1）节点node：计算机设备抽象为节点，用Node类来表示

2）网络设备NetDevice：网络设备被安装在节点内，从而使得节点可以通过信道和其他节点通信。网络设备抽象用NetDevice类表示，它提供连接节点和信道的方法，节点只有绑定网络设备之后才能具有通信功能。

3）信道Channel：将节点和其他对象相连的为信道，是无力传输媒介的抽象，它的抽象用Channel类表示。

4）应用Application：应用程序的抽象用类Application表示，应用程序是用户程序产生模拟活动的最基本的抽象，来产生和消费网络流量，驱动网络模拟器，它提供了许多方法来管理模拟用户层的各种应用。

### 1.2 NS-3仿真的基本步骤

1）创建网络；2）建立节点间的连接；3）创建流量；4）创建事件调度器，即规定节点何时采取什么动作（包括发数据包、移动等）；5）跟踪仿真过程，提取重要数据结果输出到trace文件中保存，仿真结果是进行参数分析的基础。对于无线网路的仿真，创建网络之前需要配置节点。

### Ad-Hoc路由协议

移动 Ad Hoc 网络是由一组自主的无线节点或终端相互形成的，独立于固定的基础设施、采用分布式管理的多跳网，是一种自创造、自组织和自管理的网络。 与有线网络和其他无线网络相比，移动Ad Hoc网络具有许多独特之处，如网络拓扑结构的动态变化、无线传输带宽有限、移动节点的能力有限、需要采用分布式控制方式、安全性差、网络的扩展性不强、存在单向无线信道以及生存时间短 等。根据路由建立方式的不同，可以把路由协议分为表驱动路由协议，按需驱动路由协议，混合式路由协议三类。DSDV是点行的表驱动路由协议，AODV是比较典型的按需路由协议，本实验主要利用NS3仿真实现了AODV协议和DSDV协议，并对他们进行了评估。

### 2.1表驱动（先验式）协议

**2.1.1 DSDV协议**

目的节点序列距离矢量协议（DSDV）协议基于DV算法，具有简单，易于实现，需要的存储空间小等特点，DSDV协议通过给每个路由设定序列号来避免路由环路的产生，DSDV协议能对拓扑变化做出快速的反应，当路由表有显著变化时，会立即启动路由公告，同时延缓不稳定路由的公告，以减缓路由波动。因为DSDV是先验式路由，采用时间和事件驱动技术控制路由表的传送，即每个移动节点在本地都保留一张路由表，其中包括所有可达到的目的节点、达到目的节点路由的条数、目的节点路由序列号等信息。除此之外，路由信息会周期性的更新，即使网络拓扑无变化，也会存在通信开销，维护的路由可能从未使用过，因此存在开销大等缺点。

**2.1.2 OLSR协议**

优化链路状态路由协议（OLSR）是基于链路状态算法的路由协议。作为一种主动路由协议，它的优点是在需要时可以在标准路由表中立即使用路由。由于优化的本质，在高度连接的网络中出现了最小的泛洪重复。网络中的每个节点选择一组邻近的节点来重新传输数据包，这组节点称为该节点的多点中继。OLSR协议在网络中采用多点中继(MPR)来代替纯泛洪，以减少可能的开销、广播泛洪和控制消息传输的时间间隔。只有MPR以这样的方式转发控制报文，信息才能到达整个网络，这些MPR负责声明LS信息。每个节点定期广播一个单跳邻居列表，在hello消息的帮助下选择MPRs。MPR完成从源节点到目的节点的路由计算。OLSR支持三种机制:邻居感知、有效的控制流量泛洪和足够的拓扑信息。

OLSR使用两种类型的控制消息:Hello和拓扑控制(TC)。Hello消息用于查找链路状态和节点邻居的信息，而TC消息用于广播自己通告的邻居的信息，包括至少MPR选择器列表。

### 2.2 按需（反应性）协议

**2.2.1 DSR协议**

DSR是一种基于源路由概念的按需（反应性）路由协议，它要求每个包携带从源到目的地的完整地址(路由中的每一跳)。它基于按需的路由发现和路由维护机制。DSR协议的一个优点是，节点可以在其路由缓存中存储多条路由。源节点可以在启动路由发现前检查其路由缓存中是否存在有效路由，如果存在有效路由，则不需要进行路由发现。另一方面，如果一个节点没有这样的路由，它会通过广播RREQ报文来发起路由发现。RREQ包包含目的地址、源地址、路由记录字段和唯一标识号。一旦RREQ到达目的地或一个知道到目的地路由的节点，它就会响应一个RREP，并将RREQ收集到的路由的相反方向。当链路发生故障时，可以通过主动监视确认或被动地以混杂模式运行来检测，以偷听数据包被邻居节点转发。用RERR报文通知失败的链路到源节点。源节点可以使用到目的节点的其他已知路由，也可以重新启动路由发现过程，以找到到目的节点的新路由。另一件需要注意的事情是，它不需要hello消息交换，因此节点可以进入休眠节点以保存它们的能量。也节省了相当数量的网络带宽。但是，它也存在一些缺点，因为每个分组都需要携带完整的路由信息，因此会导致整个路由协议的开销增大，降低了网络带宽的利用率，并不适合于直径大的自组网络，同时网络的扩展性不强。

**2.2.2 AODV协议**

自组织按需距离向量路由（AODV）协议是按需（反应式）路由协议，它为了找到通往目的节点的路由，源端将广播一个路由请求分组，邻居节点依次向周围节点广播此分组直到该分组被转发到一个一致的目的节点路由信息的中间节点或目的节点本身。一个节点将丢弃重复收到的请求分组，路由分族中的序列号用来防治路由环路，并能判断中间节点是否响应了相应的路由请求，当节点转发路由请求分组时，它会将其上游节点的标志ID录入路由表，从而能够构建一条从目的节点到源节点的反向路由，但源端移动时，它会重新发起路由发现过程，如果中间节点移动，那么与其相邻的节点会发现链路失效并向其上游节点发送链路失效消息并一直传到源节点，而后源节点根据情况重新发起路由发现过程。从本质上来说，AODV是DSR和DSDV的结合，它借用了路由发现的基本按需机制和DSR中的路由维护，再加上逐条路由，序列号和DSDV中的周期性的信标，因此，AODV的优点是它可以利用多播路由的优势，这正是其他协议所缺少的，而缺点则是它依赖于对称的链路而不能处理非对称的网络。

### 2.3路由协议性能评估

对协议进行性能评估前，需要进行环境参数设定，包括区域的大小，区域内节点的分布，节点的密度，节点的移动速度和移动模式，无线传播半径（功率），无线通信技术和数据传输模式都需要确定。

协议的性能评价指标有：

1）吞吐量：单位时间内成功地传送数据的比特数。

2）路由开销：单位时间内平均传递一个数据分组，需要传递的控制消息的比例。

3）端到端平均时延：一个数据包从源节点路由层成功到达目的节点路由层所需要的平均时间。

4）数据报接受率：一定时间内正确接受的数量与发送总量的比值。

5）丢包数：传输过程中未能成功到达目的节点的数据包的数目。

6）抖动：所有节点之间的数据包延迟的标准方差。

除了以上提到的评价指标外，评价路由协议的性能还有一些其他的指标，在这里我们就不在一一赘述了。

## 三、实验过程

### 1. NS3网络仿真软件的安装与测试

1）首先更新源：

sudo apt-get update

sudo apt-get dist-upgrade

编写vim脚本安装依赖库：

sudo apt-get install gcc g++ python -y

sudo apt-get install gcc g++ python python-dev -y

sudo apt-get install mercurial -y

sudo apt-get install bzr -y

sudo apt-get install gdb valgrind -y

sudo apt-get install gsl-bin libgsl0-dev libgsl0ldbl -y

sudo apt-get install flex bison libfl-dev -y

sudo apt-get install g++-3.4 gcc-3.4 -y

sudo apt-get install tcpdump -y

sudo apt-get install sqlite sqlite3 libsqlite3-dev -y

sudo apt-get install libxml2 libxml2-dev -y

sudo apt-get install libgtk2.0-0 libgtk2.0-dev -y

sudo apt-get install vtun lxc -y

sudo apt-get install uncrustify -y

sudo apt-get install doxygen graphviz imagemagick -y

sudo apt-get install texlive texlive-extra-utils texlive-latex-extra -y

sudo apt-get install python-sphinx dia -y

sudo apt-get install python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev -y

sudo apt-get install libboost-signals-dev libboost-filesystem-dev -y

sudo apt-get install openmpi\* -y

2）根据NS3官方手册，常见的安装方法有tarball方法与bake方法。本文使用tarball方法，先创建创建Tarballs目录，在该目录下载ns-allinone-3.28.tar并解压。

3）编译及测试：

cd ns-allinone-3.28

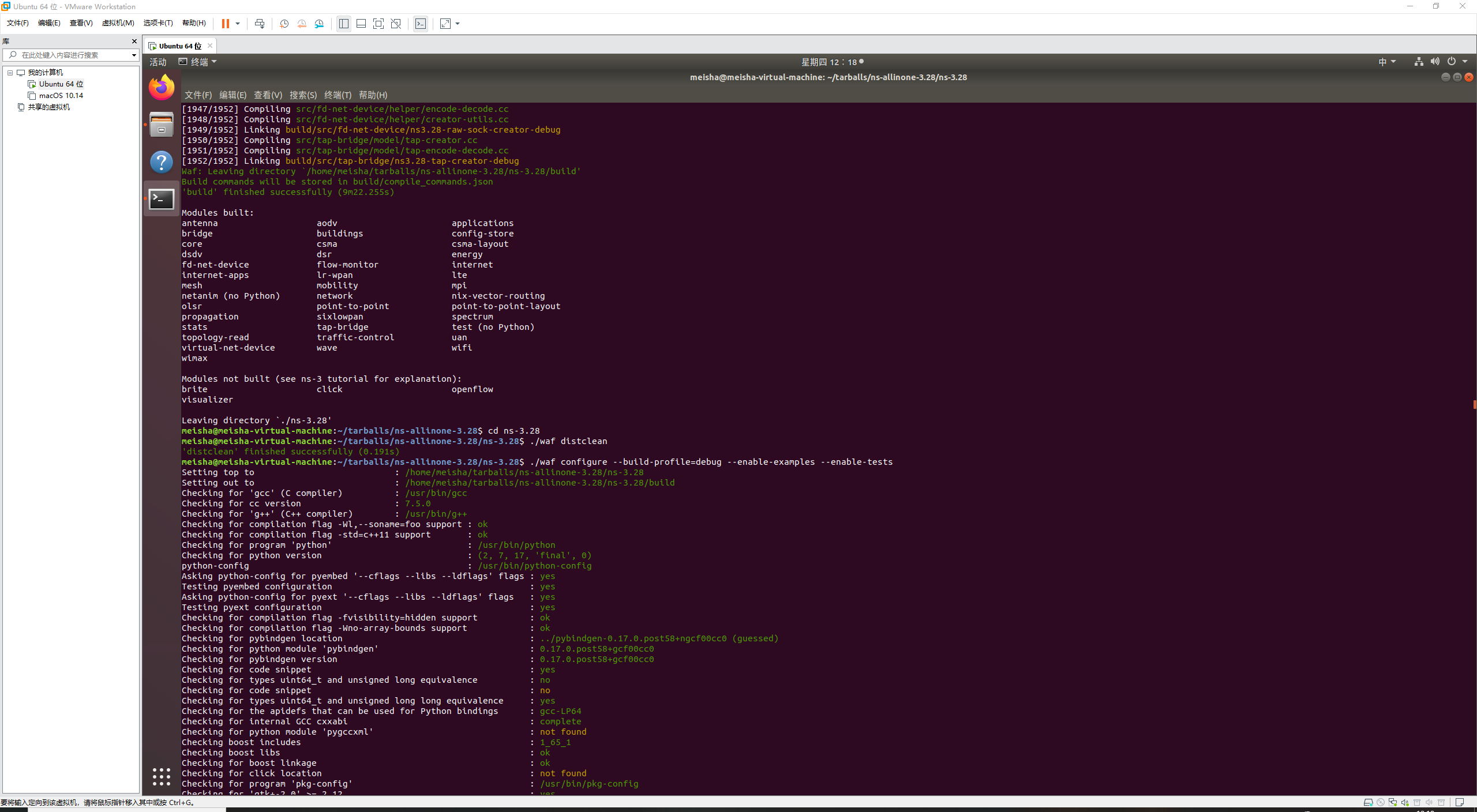
./build.py

cd ns-3.28

./waf distclean （清除整个build目录）

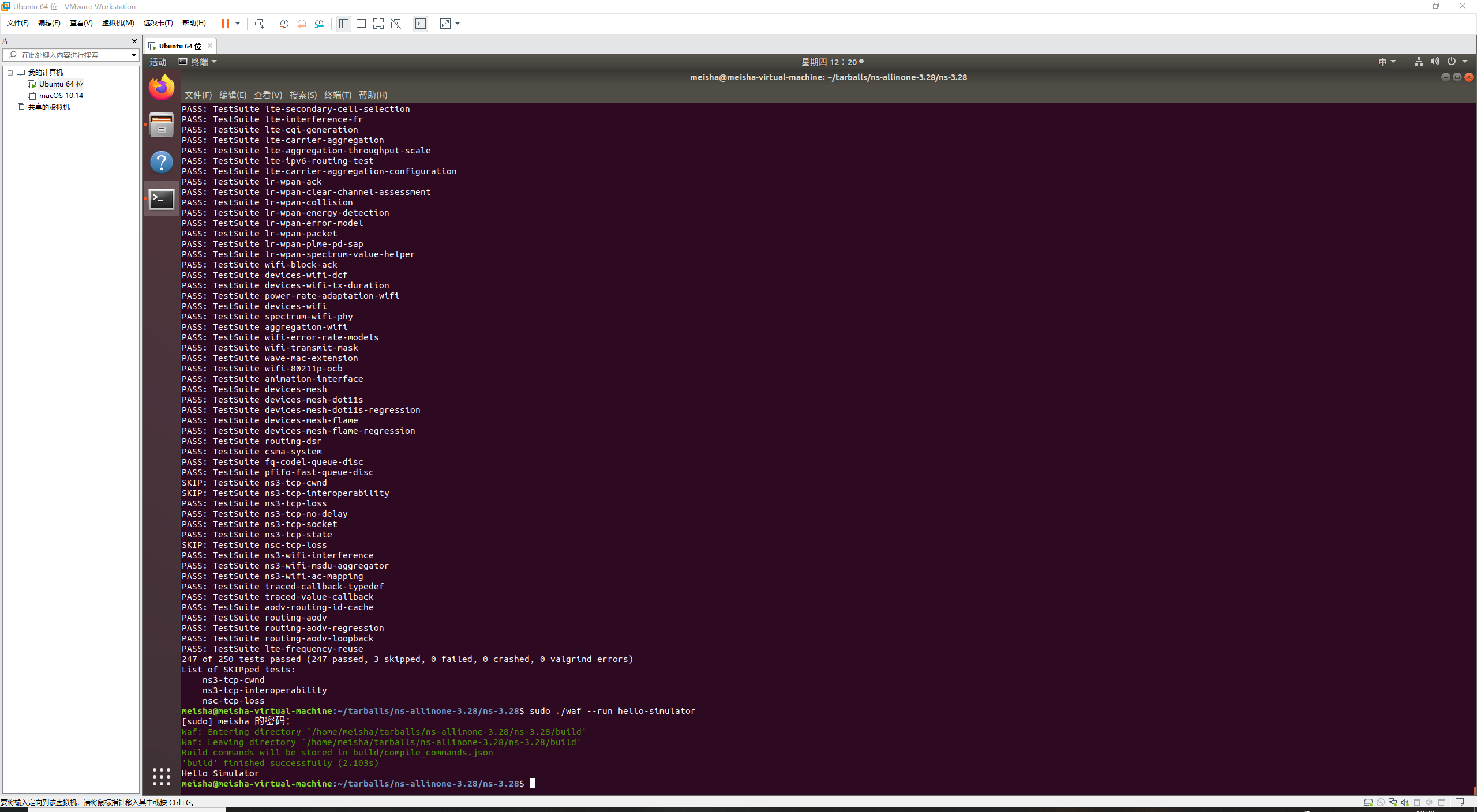
./waf configure --build-profile=debug --enable-examples --enable-tests（打开debug并开启例子及帮助）

./waf build



4）运行./test.py -c core，测试通过

5）运行hello simulation，输出正确，证明NS3安装成功



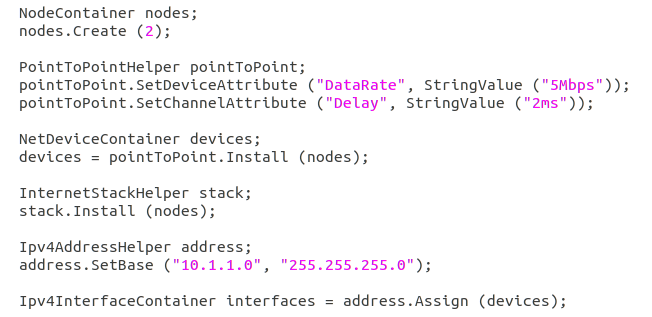
### 2.测试代码编写

### 2.1示例代码介绍

在example/tutorial文件夹下，存有7个教程文件，通过学习这些简单的教程，可以初步掌握NS3仿真测试代码的编写方法。

first.cc文件模拟了一个简单的点到点连接。在文件首行，介绍了源代码的预定格式，之后是仿真器的GNU许可。代码以一系列声明开始，接下来是命名空间，日志模块的Doxygen文件系统。

在主函数中，为了实现拓扑生成，使用了NodeContainer类、PointToPintHelper类、NetDeviceContainer类、InternetStackHelper类、Ipv4AddressHelper类。



NodeContainer类会创建NS3节点对象，它们在仿真中代表计算机。

PointToPintHelper类构建了一个点到点的连接。

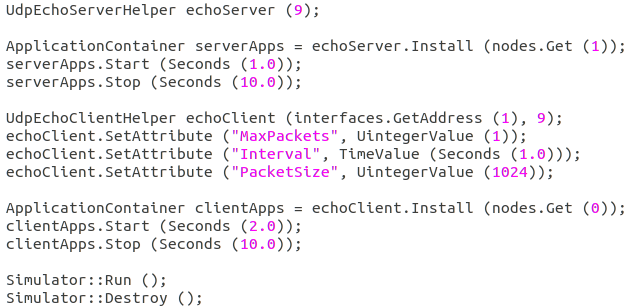
NetDeviceContainer类在两个节点之间创建PointToPointNetDevices和wirePointToPointChannel对象的PointToPointHelper对象。需要一个所有被创建的NetDevice对象列表，所以使用一个NetDeviceContainer对象来存放它们，就像使用一个NodeContainer对象来存放我们所创建节点。

类InternetStackHelper是一个安装PointToPointHelper对象和点到点网络设备的网络协议栈的拓扑生成器类。其中Install()方法以一个NodeContainer对象做为一个参数，当它被执行后，它会为每一个节点容器中的节点安装一个网络协议栈（TCP,UDP,IP等等）。

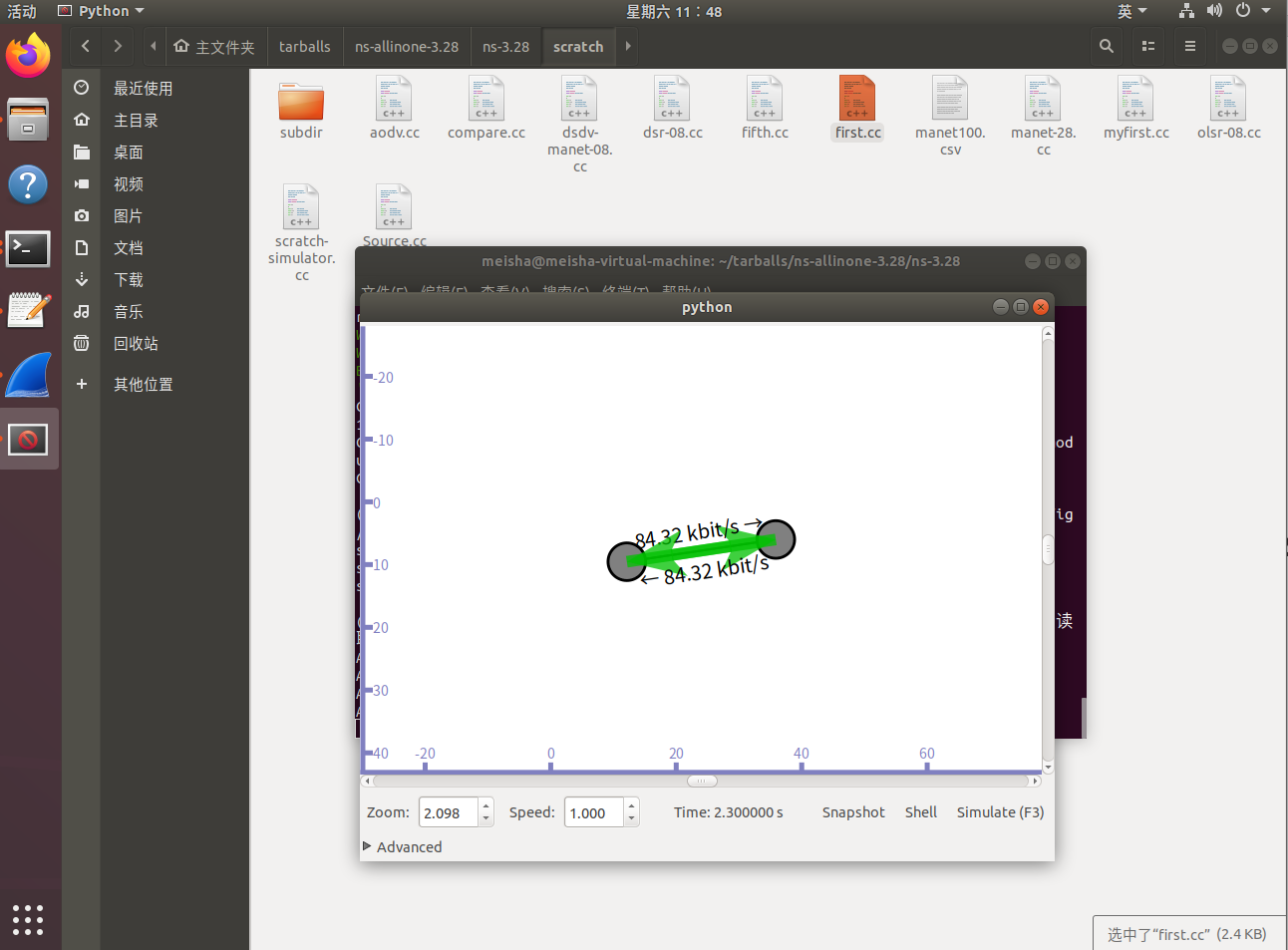
Ipv4AddressHelper提供了一个拓扑生成器来管理IP地址的分配。当执行实际的地址分配时唯一用户可见的API是设置基IP地址和子网掩码。

NS3系统的核心抽象是Application类。在这个脚本中用两个特定的Application类：UdpEchoServerApplication和UdpEchoClientApplication。

最后，使用全局函数Simulator::Run.来运行模拟器。



将first.cc置于scratch文件夹之下，使用sudo ./waf --run scratch/first –vis命令即可进行可视化仿真实验。



在本节，介绍了first.cc这一简单的实验场景的代码，并介绍了如何使用NS3进行可视化仿真实验。

### 2.2编写测试代码

Ns3内置了大量的协议源码，在src文件夹下，有aodv、dsdv等协议的仿真实验示例。与此同时在examples/routing文件夹下，还有一个manet-routing-compare实验示例，该实验旨在设置不同的路由协议以进行性能评估。个人参考了上述文件并编写了自己的仿真测试代码，对于示例实验代码将不再冗述，下面将直接介绍个人编写的测试代码。

在头文件中，引入了aodv、olsr、dsdv、dsr的相关module与helper以使用不同路由协议构建网络。特别的，引入了flow-monitor模块进行性能评估标准计算。



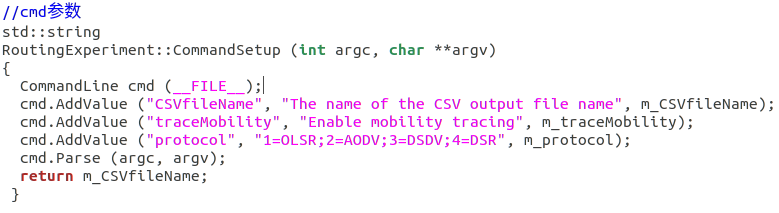
设置包接收的相关方法，吞吐量计算的相关方法。设置端口、接收器数量、csv文件名称、协议等相关变量，并完成初始化。



在接收包时，计算总包数与字节数，将相关信息写入csv文件。



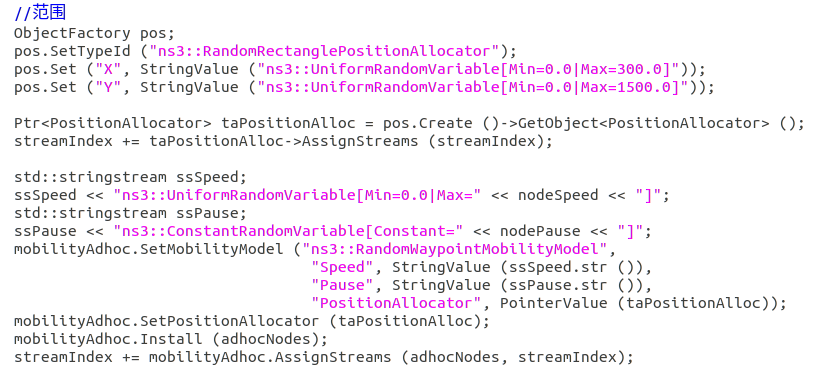
设置cmd参数，在终端执行命令时，可以增加参数实现对csv文件名的设置，是否启用trace，选择何种协议进行仿真实验。



对于网络速度、节点总数、运行时间、物理模式、节点移动的速度、节点停顿时间、包大小等参数进行设置。



对移动的范围进行设置，使用随机移动。

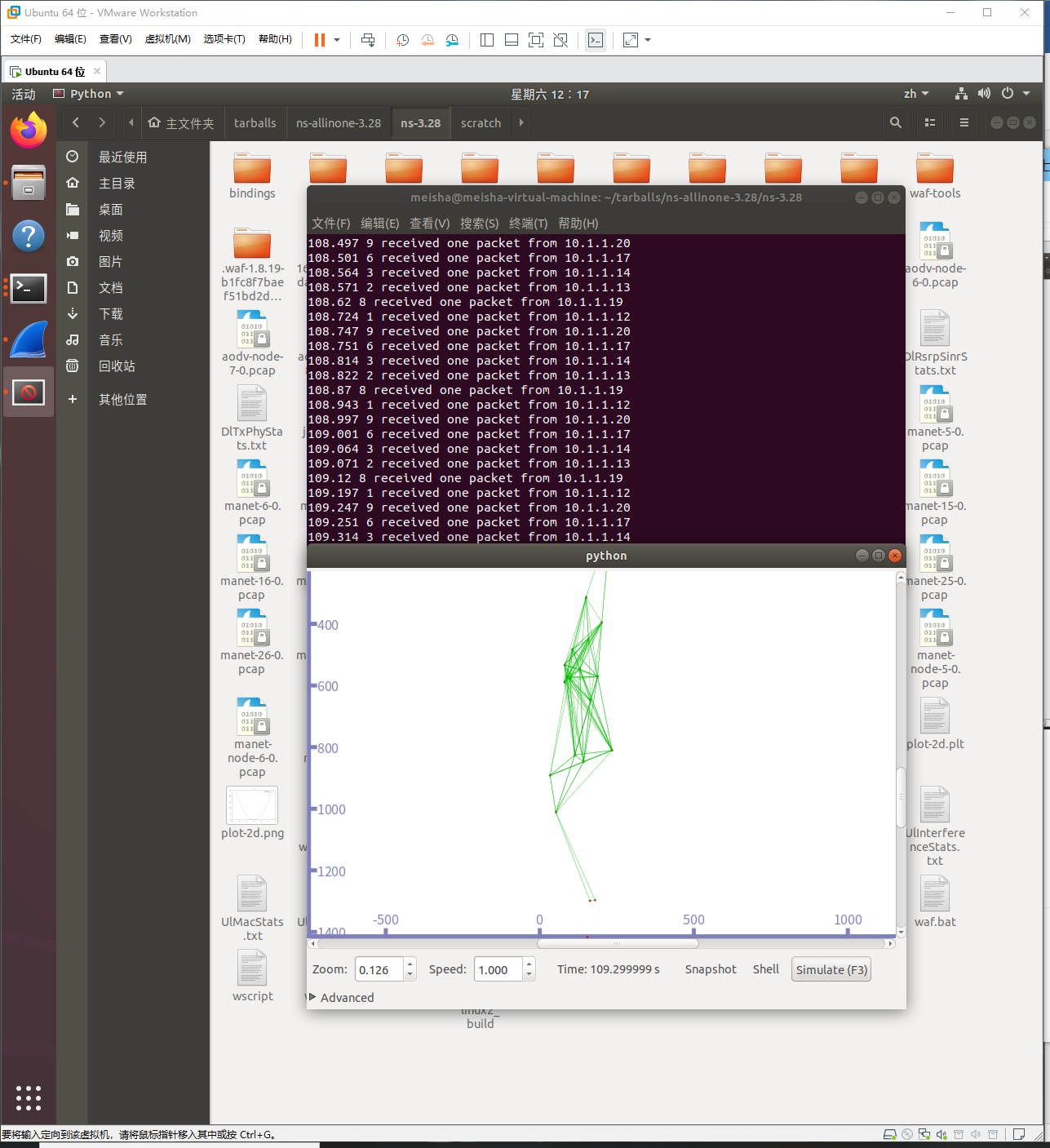


至此，所编写的代码的主要部分已全部介绍完毕。

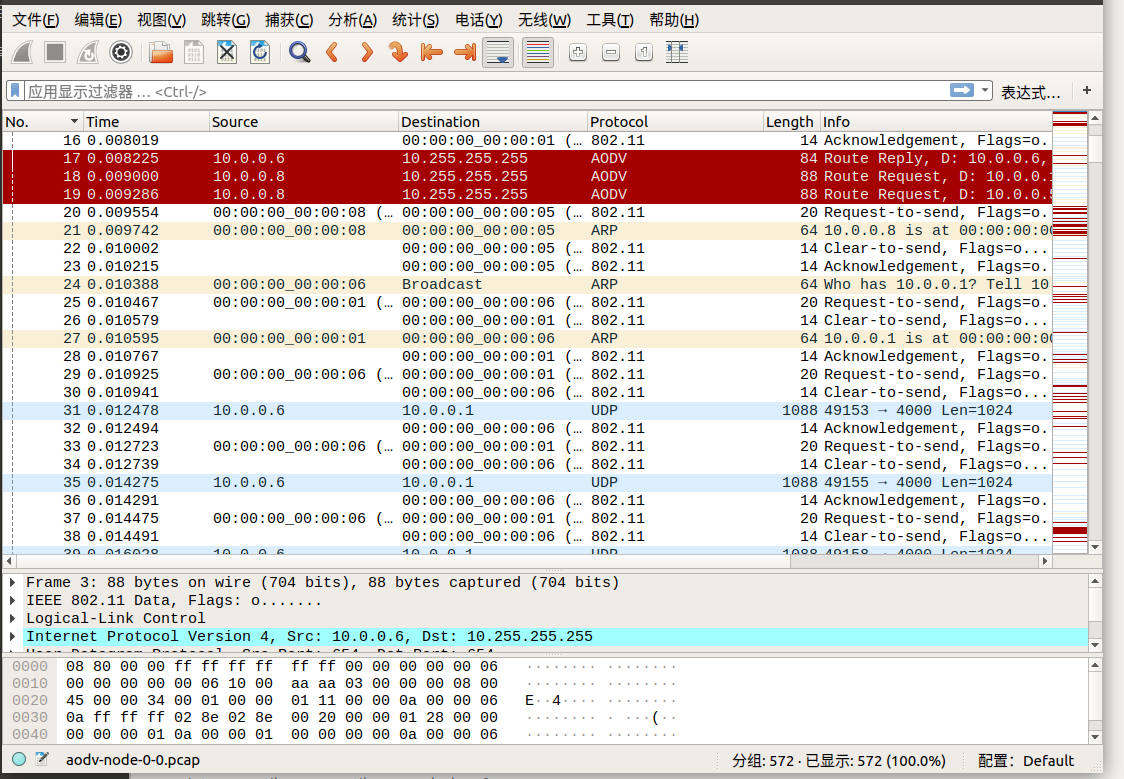
### 3.仿真实验

首先说明默认的评估环境、移动模式参数的具体设置。在默认情况下，模拟运行200模拟秒，节点数为20。节点的移动方法为RandomWaypoint，其速度为20m/s，移动范围为300×1500m，移动不设停顿时间。WiFi以2mb/s速率，802.11b和Friis损耗模型的ad hoc模式，发射功率为7.5dBm。默认直接使用AODV协议，有10个发送UDP数据的源/接收数据对，每个应用程序的速率为2048kb/s，每秒4个64字节数据包。在实验中，每次仅改变节点数、节点移动速度对不同协议的吞吐量、时延、数据包接受率、抖动等性能评估标准进行统计。

以默认情况为例，进行仿真实验。输入：sudo ./waf --run scratch/Compare –vis开启可视化界面，如下图所示。



当实验结束时，如果选择生成了pcap trace文件，则可以直接使用Wireshark对整个仿真过程的细节进行查看。如下图所示，红色的的代表AODV协议的捕获内容。通过查阅，可以自行计算吞吐量等性能指标。

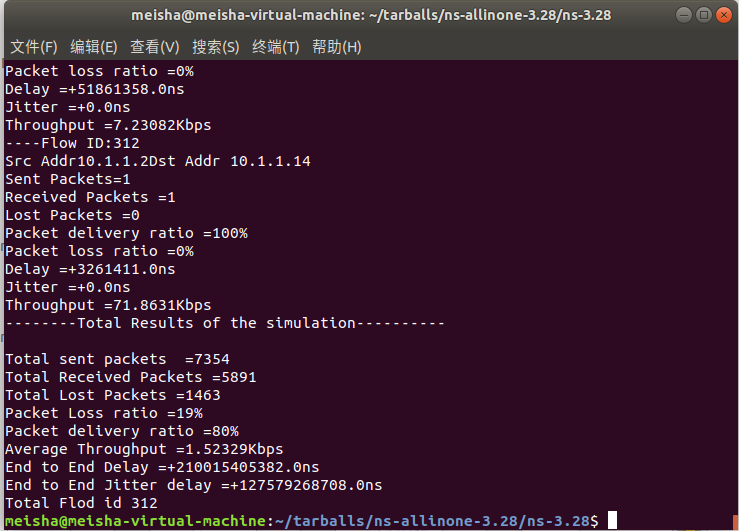


但是这样做费时费力，查阅NS3官方文档，提供了两种常用的方法，一直是编写awk脚本进行计算，另一种是使用FlowMonitor模块进行统计输出。在此，我选择了使用FlowMonitor模块。

为所以节点后添加FlowMonitor模块，之后在Simulator::Run()之前添加数据统计代码。并把trace默认设置改为false。



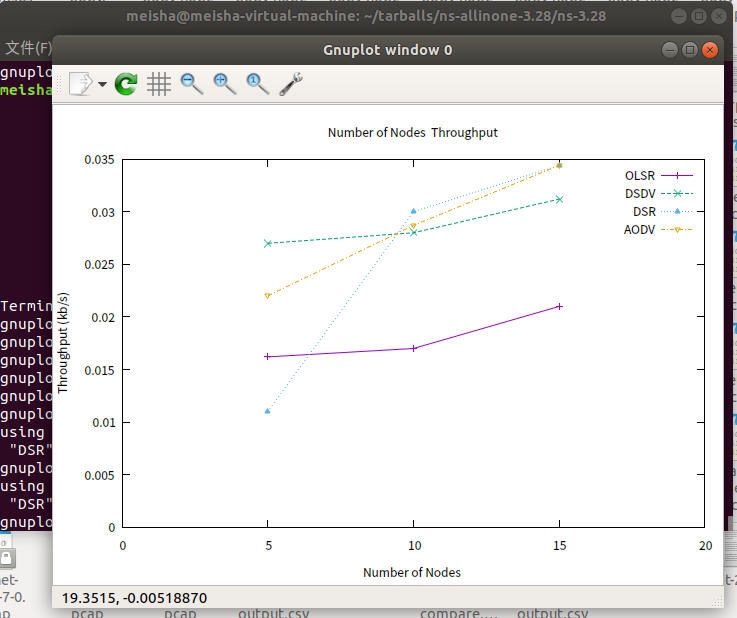
在控制台中，输出每个节点的性能指标。在最后，输出最终的统计结果，包括：总发包数、总收包数、总丢包数、丢包率、包接收率、平均吞吐量、端到端时延、端到端抖动，如下图所示。



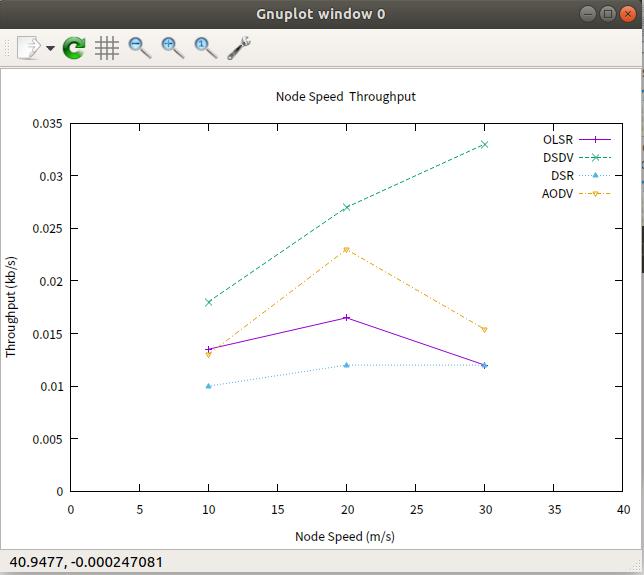
## 四、实验结果统计与分析

在上一章中介绍了实验的过程，在本章将对实验结果进行统计与分析。通过输出的csv文件，可以使用Gnuplot自动绘制对应的图像。

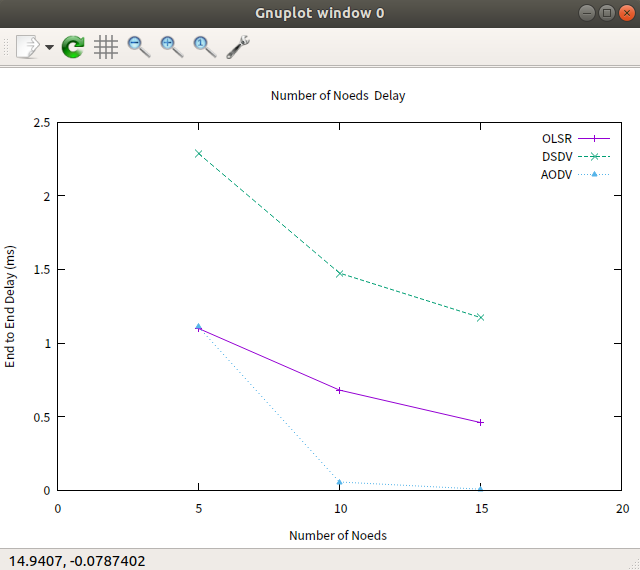
随着节点数量的增加，尽管观察到所有协议的吞吐量都有增加，但毫无疑问DSR是提升最大的协议。OLSR与DSDV的提升比较小。在DSDV中，我们可以观察到节点从5增加到10时增量很小，但节点从10增加到15时略有增加。随着节点数量的增加，AODV的吞吐量不断增加。



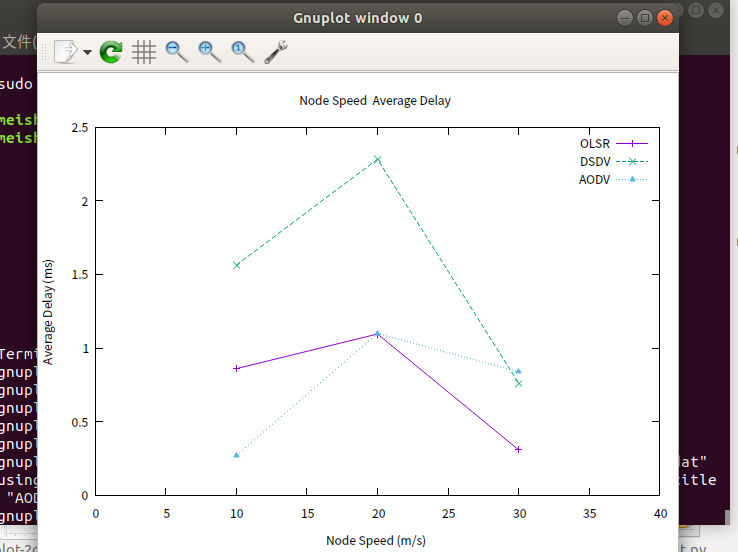
在OLSR的情况下，可以观察到从10m/s到20m/s时吞吐量略有增加，但从20m/s到30m/s时，吞吐量下降。DSDV的吞吐量随着节点移动速度不断上升。DSR在所有情况下提供几乎相同的吞吐量。AODV吞吐量在10m/s到20m/s上升，然后在20m/s到30m/s下降。



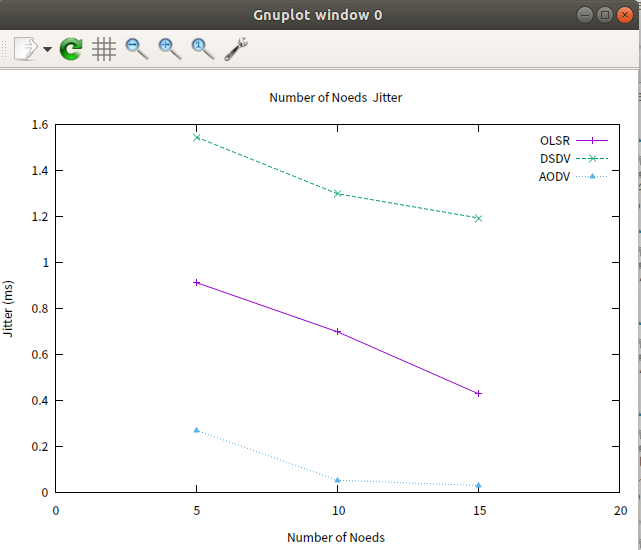
下图为随着节点数量的变化而变化的端到端时延的信息。我们看到这在这三种协议的情况下时延都是递减的。但与DSDV、DSR相比，AODV的时延下降速度较快。DSDV在所有情况下都有最高的端到端时延。



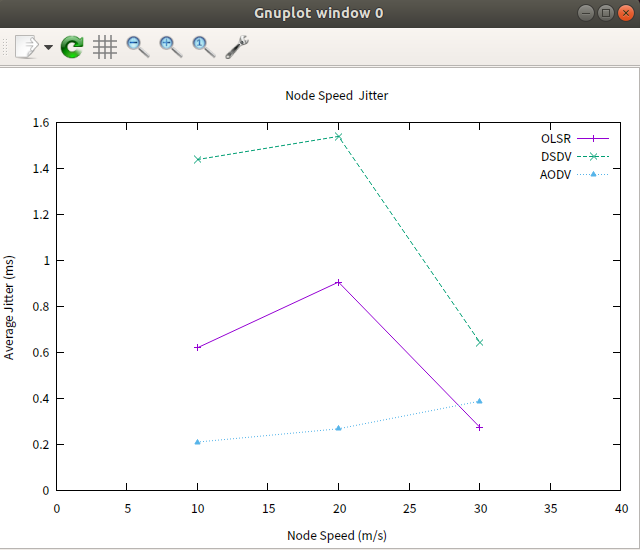
通过改变节点的移动速度，在不同的节点移动速度下对OLSR、DSDV、AODV的平均时延进行了测试，绘制了如下图像。可以看到，在10m/s的情况下，DSDV具有最高的平均时延，即使节点速度增加到20m/s，同样的情况仍在继续，但在30m/s的情况下，DSDV的平均延迟小于AODV。并且在30m/s时，OLSR具有最小的平均时延，而在10m/s时，AODV具有最小的平均时延。



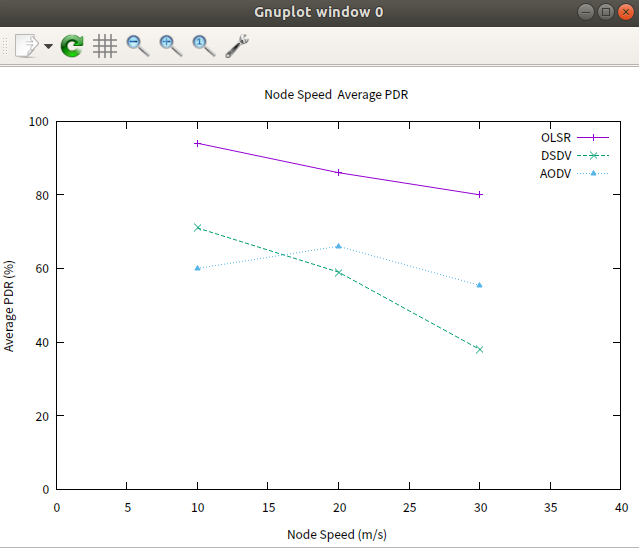
随着节点数量的增加，抖动都会有所减少。通常情况下，DSDV抖动更大，其次是OLSR，最后才是AODV。



观察到，当节点移动速度达到20m/s时，OLSR、DSDV均取得最大平均抖动，当节点移动速度继续增大时，平均时延反而下降。而对AODV来讲，随着节点速度增加，平均时延也随之增加，并且节点移动速度的改变对平均抖动的改变影响较小。



在总的趋势上，随着节点移动速度的增加，包接收率呈下降趋势。在OLSR、DSDV、AODV三个协议中，OLSR协议的包接收率最高。在10m/s时，DSDV的包接收率高于AODV，但是在20m/s与30m/s时，AODV的包接收率更高。



在OLSR和DSDV中，包接收率的变化随着节点数量的改变变化很小，但在AODV中，将节点数从5增加到15时，可以观察到一个较为明显的提升。在所有情况下，OLSR的包接收率都是最高的。

