

计 算 机 网 络 实 验

实 验 指 导 书

|  |  |
| --- | --- |
| 实验名称 | NS3基础仿真实验 |

电子信息与通信学院 2018年制

# 实验目的

1．了解网络仿真的意义

2．熟悉NS-3的基本语句

3．安装并熟悉使用NS-3

4．用NS-3搭建最基本的网络仿真场景

# 实验背景

1. **网络仿真技术**

近年来，随着计算机和网络通信技术的不断发展，网络技术的研究也进入到了一个飞速发展的时期。研究人员不断开发出新的网络协议、算法和应用，以适应日益增长的网络通信需要。然而由于网络的不可控、易变和不可预测等特性的存在，给新的网络方案的验证、分析和比较带来了极大的困难。目前网络通信的研究一般分为以下3种方法。

1）分析方法：在理论和协议层面上对网络通信技术或系统进行研究分析，抽象出数学分析模型，利用数学分析模型对问题进行求解。如采用数学建模、协议分析、状态机、集合论以及概率统计等对多种理论分析手段和方法对通信网络及其算法、协议、网络性能等各个方面进行研究。

2）网络模拟：即计算机模拟仿真算法。网络模拟日益成为分析、研究、设计和改善网络性能的强大工具，它通过在计算机上建立一个虚拟的网络平台，来实现真实网络环境的模拟，网络技术研究人员在这个平台上不仅能对网络通信、网络设备、协议以及网络应用进行设计研究，还能对网络的性能进行分析和评价。

3）实验网方法：对网络协议、网络行为和网络性能采用建立实验室测试网络、网络测试平台（network testbed）和小规模商用实验网络的方式对网络进行实战检验。就是设计出研究所需要的合理硬件和软件配置环境，建立测试床和实验室，在现实的网络上进行研究。

以上3种方法有利有弊，相辅相成并各有侧重点。

理论研究适用于早期研究与设计阶段，对新算法和新技术进行理论准备和验证，除了人力和知识，几乎不需要什么额外成本。

实验网方法是网络和系统在投入实际应用前的一次系统的演练，能够发现网络设计与用户需求之间的相合度以及检验网络实际使用的效用和性能。该阶段建设成本很高，要求技术和设备开发相对成熟，网络系统基本成型，主要是对业务、系统稳定性能和服务性能的检验。

而网络模拟阶段可以说是理论方法和实验网方法的中间阶段，它可以对新协议进行初步实现和验证，并有助于新协议的及时调整和改进。网络模拟阶段由于采用计算机软件进行模拟，使得很多研究工作人员能够研究大规模网络和学习新协议新算法的设计和实现，并且能够在网络实用前对其进行检验和改进。此外，它还可以在各种新老系统和算法之间进行比较而不必花费巨资去建立多个实际系统。因此，网络模拟是网络通信研究中一种非常重要的方法。

1. **NS-3简介**

俗话说“工欲善其事必先利其器”。为了方便网络研究人员的研究工作，一款好的网络模拟软件是必不可少的。它必须具备以下特点：

1）能够提供简单易用的使用界面，能方便快捷地建立和修改模拟环境和模拟配置；

2）能够提供常用的绝大部分网络协议、算法和应用模块；

3）能够方便地进行配置和扩展，有利于添加新的协议和算法。

当前有许多优秀的网络模拟软件，如NS-2（network simulator version 2）、OPNET（optimized performance network engineering tool）、OMNet++（objective modular network testbed in C++）、GloMoSim（global mobile information system simulator等，这为网络研究和开发人员提供了很好的网络模拟平台。

相比于以上提到的网络仿真软件，NS-3是比较年轻的网络仿真工具。NS-3是尽量吸取现有网络模拟工具的优点并避开其缺点，应用现代软件工程思想和网络仿真技术而设计开发的新一代网络模拟工具。

NS-3不是NS-2的扩展，而是一个全新的网络模拟器，是由美国华盛顿大学的Thomas R.Henderson教授及其研究小组在美国自然科学基金（NFS）的支持下，于2006年开始应用现代网络模拟技术和软件开发技术设计并开发的一个全新网络模拟工具。NS-3广泛汲取了现有优秀开源网络模拟器如NS-2、GTNetS、Yans等的成功技术和经验，专门用于教育和研究用途的离散事件模拟器，它基于GNU GPLv2许可，可以免费地获取、使用和修改。

NS-3相对简单易学、可扩展性好、节省资源，并且能提供高性能的，与真实网络相近的网络仿真，尤其能够集成到实验床和虚拟机环境，逐渐得到学术界的认可，在和仿真相关的国际会议如SIMUTools、SpringSim、ICST-NSTools等，以及网络方面的顶级会议如Sigcomm、Infocom等研究和使用NS-3的文献逐渐增多，详<http://www.nsnam.org/overview/publications>。

# 实验原理

1. **NS-3仿真基本流程**

使用NS-3进行网络仿真时，一般经过以下4个步骤。

**1）选择或开发相应模块**

根据实际仿真对象和仿真场景选择相应的模块：如是有线局域网络（CSMA）还是无线局域网络（Wi-Fi）；节点是否需要移动（mobility）；使用何种应用程序（application）；是否需要能量（energy）管理；使用何种路由协议（internet、aodv等）；是否需要动画演示等可视化界面（visualizer、netanim）等。

**2）编写网络仿真脚本**

有了相应的模块，就可以搭建网络仿真环境，NS-3仿真脚本支持2种语言：C++和Python，但是2种语言的API接口是一样的，部分API可能还没有提供Python接口。编写NS-3仿真脚本的大体过程如下。

* 生成节点：NS-3中节点相当于一个空的计算机外壳，接下来要给这个计算机安装网络所需要的软硬件，如网卡、应用程序、协议栈等。
* 安装网络设备：不同的网络类型有不同的网络设备。从而提供不同的信道、物理层和MAC层，如CSMA、Wi-Fi、WiMAX和point-to-point等。
* 安装协议栈：NS-3网络中一般是TCP/IP协议栈，依据网络选择具体协议，如是UDP还是TCP，选择何种不同的路由协议（OLSR、AODV和Global等）并为其配置相应的IP地址，NS-3既支持IPv4也支持IPv6。
* 安装应用层协议：依据选择的传输层协议选择相应的应用层协议，但有时需要自己编写应用层产生网络数据流量的代码。
* 其他配置：如节点是否移动，是否需要能量管理等。
* 启动仿真：整个网络场景配置完毕，启动仿真。

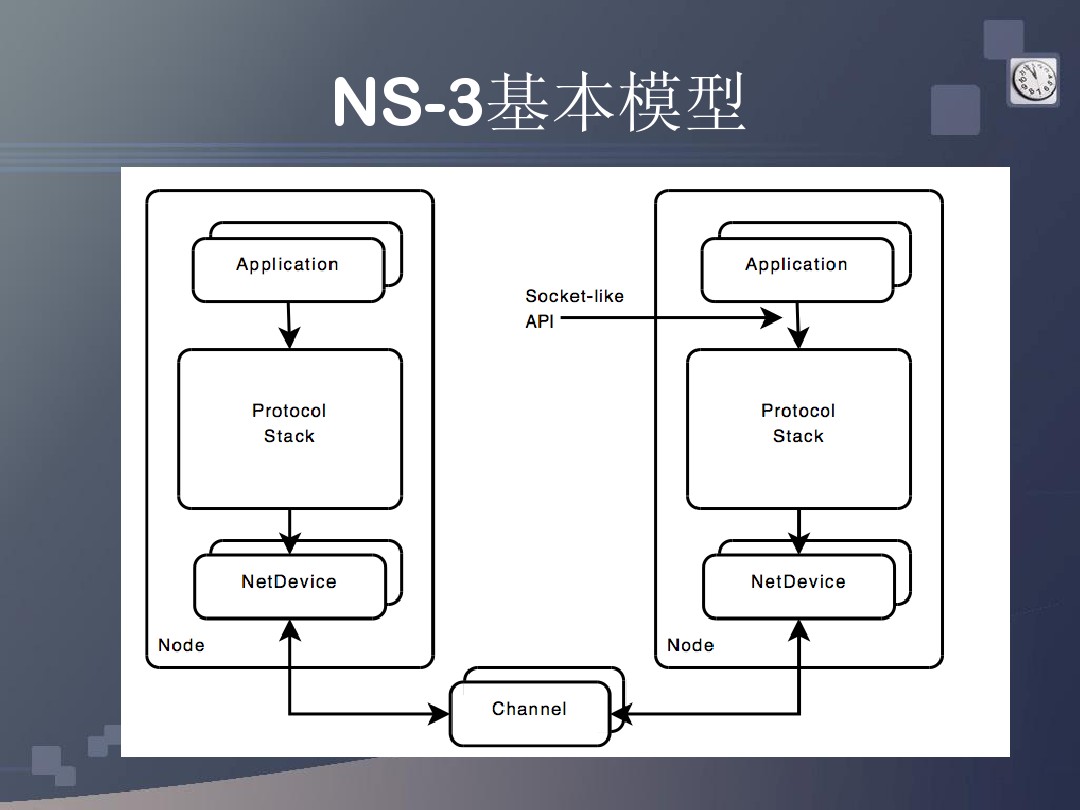


图3-1 NS-3基本模型

NS-3仿真的基本模型如图3-1所示，搭建NS-3网络仿真场景和搭建实际网络类似，首先需要网络节点，NS-3中使用节点的概念；节点需要有网络设备，类似于网络接口卡，NS-3中有相应网络设备的概念；网络设备通过传输媒体连接，NS-3中使用信道的概念来代表传输媒体，设置信道延迟等属性，并且和实际网络相似：信道和网络设备是对应的，CSMA网络设备对应CSMA的信道，Wi-Fi网络设备对应Wi-Fi的信道。

以上概念使网络节点实现了物理连接，但要实现通信，还需要软件支持，也就是协议，应用层产生数据，利用类socket编程（和真实的BSD socket很像）实现数据分组的向下传递，数据分组通过协议栈—TCP/IP向下传递给网络设备（可以简单理解为网卡），该网络设备包括MAC层、物理层协议，于是数据分组就像在真实网络中流动一样，由数据帧转换成二进制流，最终变成信号通过媒体信道传输到目的节点。

目的节点收到数据分组后从下往上逐层转交，由媒体信号转换成二进制，由二进制转换成数据帧，再有数据帧转换成IP数据分组，然后经由传输层的端口号转交给相应的进程（应用程序application），至此在NS-3中完成了一次和真实网络极其相近的完整数据传输过程。

**3）仿真结果分析**

仿真结果一般有2种：一是网络场景，二是网络数据。网络场景如节点拓扑结构、移动模型等，一般通过可视化界面（PyViz或NetAnim）可直观观测到；网络数据也可在可视化界面下有简单的统计，此外，可以通过专门的统计框架（status）或者自行通过NS-3提供的追踪（tracing）系统收集、统计和分析相应的网络数据，如数据分组的延迟、网络流量、分组丢失率和节点消息缓存队列等。

**4）根据仿真结果调整网络配置参数或修改源代码**

有时实际结果和预期相差较远，这时我们要分析原因，是网络参数有问题，还是协议本身有出入等，然后再重新设计，重新仿真，如此反复，直到达到满意的结果。

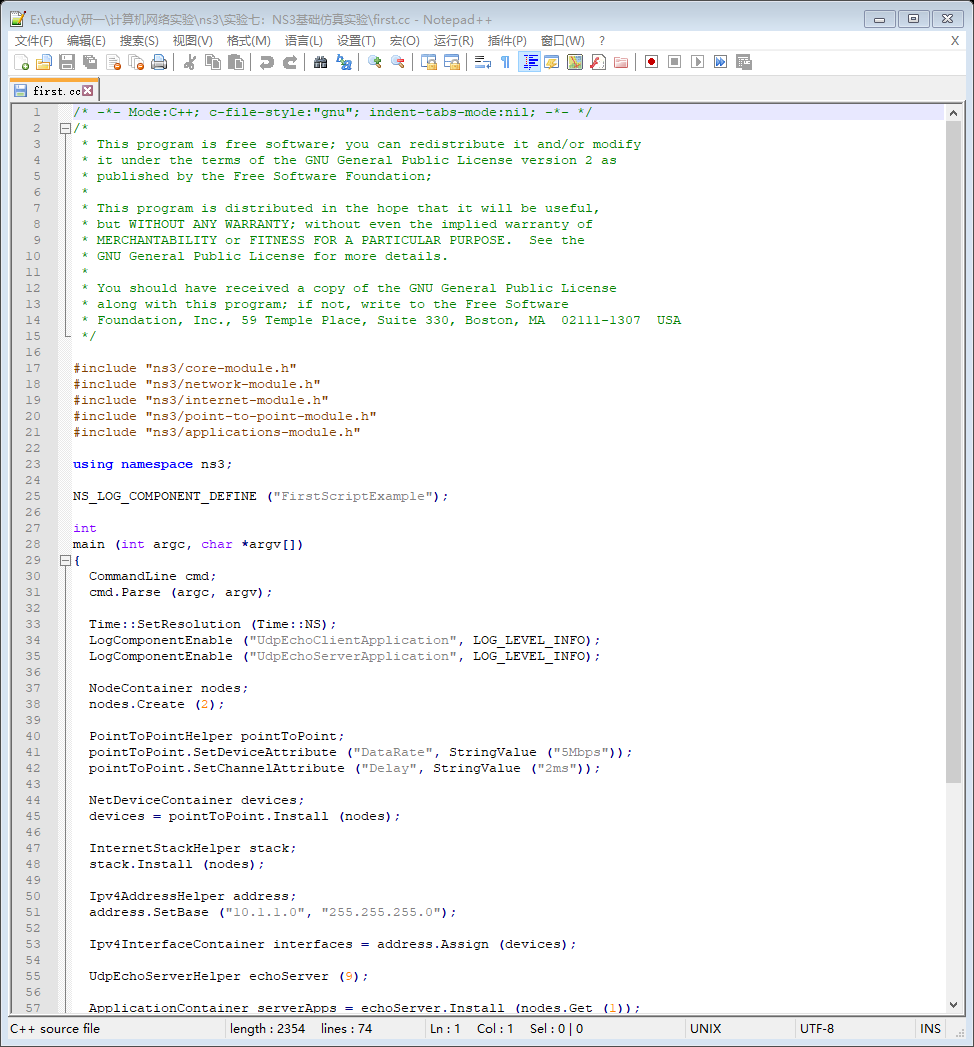
# 实验内容

1. **NS-3基础实验示例**

示例first.cc所在位置为ns-3.16/example/tutorial/，该目录是学习编写NS-3脚本的基础示例。该脚本仿真的网络场景非常简单，只是在2个节点间创建一个简单的点到点通信，但是“麻雀虽小，五脏俱全”，下面大家一起逐行仔细学习，深入体会编写NS-3网络仿真脚本的方法和过程，相信从中会有所收获，以下是对其分析。

1. **代码规范**

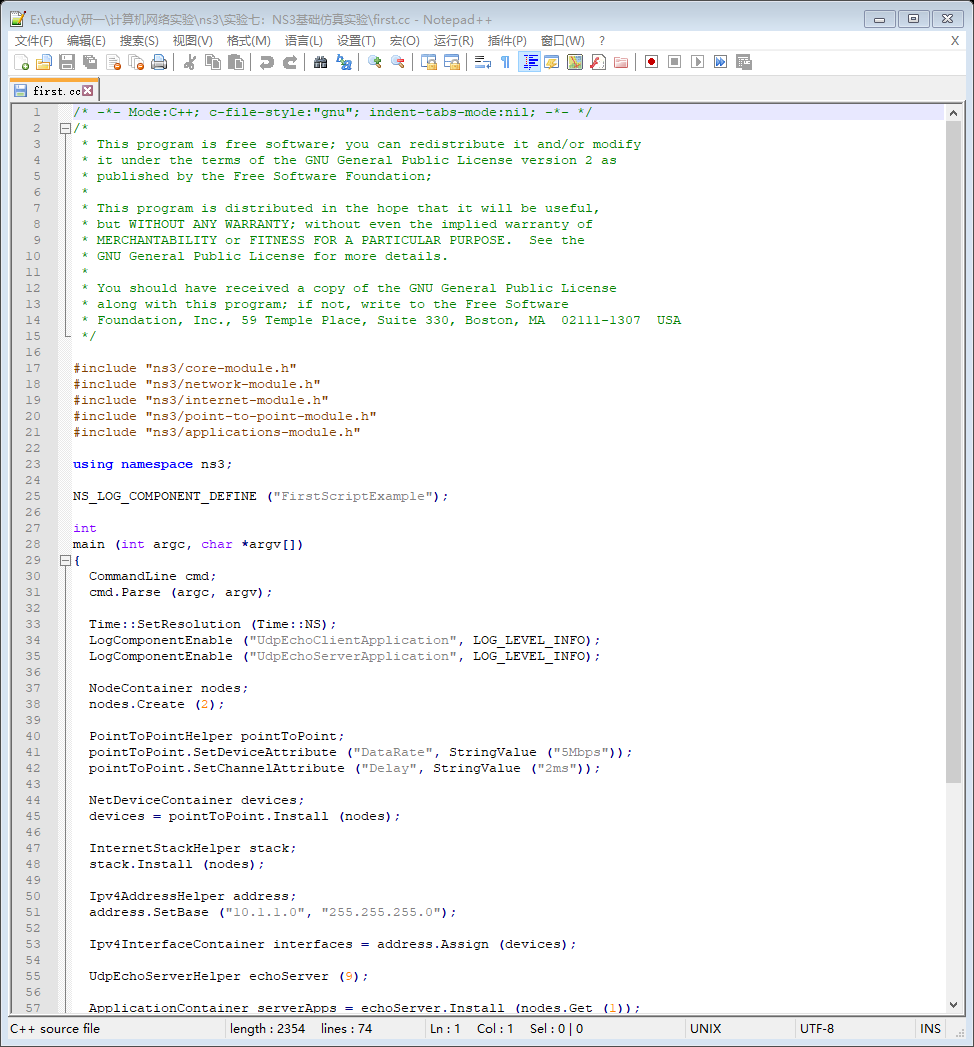
首先看看NS-3的代码规范和基本结构：



在文件中的第一行是emacs模式行，这行告诉了源代码中使用的预定格式（代码风格）。下面的一大段（略去一部分）和许多大型的软件开发项目一样，NS-3项目采用了一套所有贡献代码必须遵守的代码风格。NS-3仿真器使用了GNU General Public License许可。

在NS-3的每一个文件头看到相应的GNU法律条文。通常会在GPL内容的上方看到一个相关机构的版权声明，而在GPL内容的下方会有相应的作者列表。NS-3中好多代码在此都有作者列表和其联系方式（个人主页或邮箱等），这个信息使读者可以和原作者联系、交流，获得更多关于代码以外的信息，这也是学习NS-3的另外一个好方法，不能闭车造门，一定要开放、交流！

1. **头文件**

****

为了帮助高层的脚本用户处理大量系统中的include文件，会把所有的文件根据模块功能进行大致的分类，提供了按大致功能分类的一组include文件，在使用时只需要选择包含几个头文件即可。

在编译的过程中，每一个NS-3的include文件被放在build目录下一个叫NS-3的目录中，这样做可以避免include文件名的冲突。ns3/core-module.h与src/core目录下的ns-3模块相对应。如果你查看ns3目录会发现大量的头文件。当你编译时，Waf会根据配置把在ns3目录下的公共的头文件放到build/debug或者build/optimized目录下。Waf也会自动产生一个模块include文件来加载所有的公共头文件。

1. **ns-3命名空间**



ns3命名空间。C++用using来把ns-3命名空间引入到当前的（全局的）声明域中，这个声明就是说，你不用为了使用ns-3的代码而必须在所有的ns-3代码前打上ns3::作用域操作符。ns-3工程是在一个叫做ns3的C++命名空间中实现的，这把所有与ns-3相关的声明，集中在一个与全局命名空间相区别的命名空间中。但是如果使用标准C++的内容可能就要加上std::前缀了。

1. **日志**



此时会遇到第一个陌生的语句，查阅API文档（遇到不认识的语句，这是最好甚至是唯一的办法），从文档中能够了解到这条语句的宏定义。该语句的定义语句出现在log.h头文件的第122行，功能是用特殊的名字定义一个日志组件。语句实际上是生成一个LogComponent类型的对象g\_log，并且通过构造函数LogComponent(name)初始化，变量name通过宏定义传递参数。

1. **主函数**

****

这就是脚本程序的主函数声明。ns-3脚本没有什么特别的，就和一个普通的C++程序一样，你需要定义一个会被第一个执行的主函数。下面两行脚本是用来使2个日志组件生效的，它们被内建在Echo Client和Echo Server的应用中：

****

这两行代码将“UdpEcho”应用程序的客户端和服务器端的日志级别设为“INFO”级。这样，当仿真产生数据分组发送和接收时，对应的应用就会输出相应的日志消息到相关的日志模块。ns-3日志系统对于读者阅读、调试和编写代码是很有用的，总是通过程序的输出来了解程序的执行的细节。

1. **生成网络节点**

****在脚本中的下面两行将会创建ns-3节点对象，它们在仿真中代表计算机。

NodeContainer类定义包含在头文件<node-container.h>中，是ns-3的一个Helper类，能一次操作多个节点。例如，如果一个设备Helper类想安装设备到大量相似的节点，就可以使用NodeContainer类型的变量做参数。

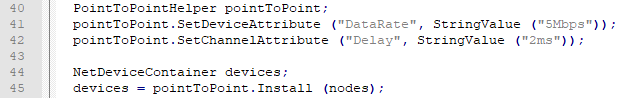
上面的第一行只是声明了一个名为“nodes”的NodeContainer节点容器类。第二行调用nodes对象的Create()方法创建了2个节点。

节点是ns-3对网络设备节点的一个抽象概念，可以简单地认为节点代表一台能够加入诸如协议栈，应用以及外设卡等的计算机。NodeContainer的拓扑助手类提供了一种简便的方法来创建、管理和使用任何节点对象，我们用这些节点来运行模拟器。

NodeContainer容器类调类用ns-3的内部方法Create()来产生2个节点对象，并把指向这2个对象的指针存储在系统之中，但在脚本中它们所代表的节点什么都没有做。下一步把节点连接到网络中，本例子是最简单的网络形式，在2个节点之间建立点到点连接。

1. **物理连接计算机**

实际中物理连接两台计算机成为网络一般来说需要使用网卡和网线，ns-3中将对应的物理实体抽象为网络设备和信道2个概念，下面的语句就实现了网络节点物理连接：

****

在此特别强调一下ns-3当中的Helper类，它是一族类，几乎所有的模块都有相应的一个或若干个Helper类，负责把网络设备连接到节点、信道，配置IP地址等普遍的任务，ns-3把这些工作抽象出来，便于程序开发者使用。

PointToPointHelper类负责设置网络设备和信道属性，并通过Install方法把设备安装到节点中。信道和网络设备是对应的，比如以太网设备和无线信道就不能一起使用。

第一行初始化了一个PointToPointHelper的对象PointToPoint。而紧接着的下一行从上层的角度告诉PointToPointHelper对象当创建一个PointToPointNetDevice对象时使用“5Mbit/s”来作为数据速率。从细节方面讲，字符串“DataRate”与PointToPointNetDevice的一个属性相对应。与PointToPointNetDevice上的“DataRate”类似，PointToPointChannel也有一个Delay属性告诉PointToPointHelper使用“2ms”作为每一个被创建的点到点信道的传输时延值。

正如使用NodeContainer拓扑助手对象来模拟创建节点，我们会让PointToPointHelper来做关于创建、配置和安装设备的工作。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对象来存放自身所创建节点。现在有一个包含2个节点的NodeContainer对象来存放自身所创建节点。现在有一个包含2个节点的NodeContainer对象，有一个准备在2个节点之间创建PointToPointNetDevices和wirePointToPointChannel对象的PointToPointHelper对象。

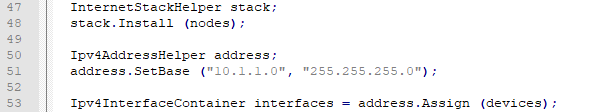
****

这两行代码会完成设备和信道的配置。第一行声明了上面提到的设备容器，第二行完成了主要工作。PointToPointHelper的Install()方法以一个NodeContainer对象作为一个参数。在Install()方法内，一个NetDeviceContainer被创建了。

对于在NodeContainer对象中的每一个节点（对于一个点到点链路必须明确有2个节点），一个PointToPointNetDevice被创建和保存在设备容器内。一个PointToPointChannel对象被创建，2个PointToPointNetDevices与之连接。当PointToPointHelper对象创建时，那些在助手中就被设置的属性被用来初始化对象对应的属性值。

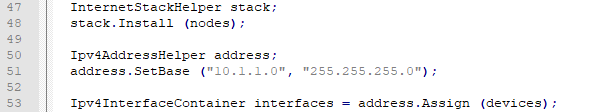
当调用了pointToPoint.Install(nodes)后，会有2个节点，每一个节点安装了点到点网络设备，在它们之间是一个点到点信道。2个设备会被配置在一个有2ms传输时延的信道上以5Mbit/s的速率传输数据。

1. **为计算机安装协议栈**

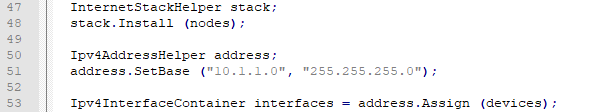
****

类InternetStackHelper会为每一个节点容器中的节点安装一个网络协议栈，主要是IP层。

Ipv4AddressHelper为节点上的设备设置IP地址。提供了一个拓扑助手来管理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。

****

声明了一个地址助手对象，并且告诉它应该从10.1.1.0开始以子网掩码为255.255.255.0分配地址。地址分配默认是从1开始并单调增长，所以在这个基础上第一个分配的地址会是10.1.1.1，紧跟着的是10.1.1.2等。底层ns-3系统事实上会记住所有分配的IP地址。

****

这行代码完成了真正的地址配置。在ns-3中使用Ipv4Interface对象将一个IP地址同一个设备关联起来。正如有时候需要一个被助手创建的网络设备列表一样，有时候需要一个Ipv4Interface对象的列表。Ipv4InterfaceContainer提供了这样的功能。现在有了一个安装了协议栈，配置了IP地址类的点到点的网络。这时所要做的事情是运用它来产生数据通信。

1. **安装应用层**

下面的代码用来安装服务器端应用程序、设置端口号：



第一行声明了UdpEchoServerHelper。像往常一样，这个并非应用本身，是一个用来帮助创建真正应用的对象。本书约定在助手中放置必需的属性。本例中除非我们告知助手服务器和客户端所共知的一个端口号，否则这个助手是不会起任何作用的。本书并没有随机选择，而是把这个端口号作为助手构造函数的一个参数。只要你愿意，就能够使用SetAttribute设置“Port”参数到另一个值。

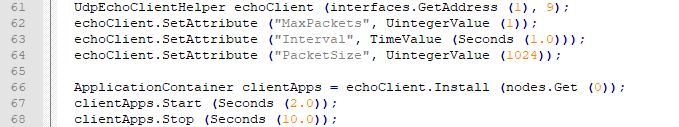
同其他助手对象类似，UdpEchoServerHelper对象有一个Install方法。实际上是这个方法的执行，才初始化回显服务器的应用，并将应用连接到一个节点上去。安装方法把NodeContainer当作一个参数，正如其他安装方法一样。这里有一个C++隐式转换，此转换以nodes.Get(1)的结果作为输入，并把它作为一个未命名的NodeContainer构造函数的参数，最终这个未命名的NodeContainer容器索引号为1的机节点上安装一个UdpEchoServerApplication。安装会返回一个容器，这个容器中包含了指向所有被助手创建的应用指针。

应用对象需要一个时间参数来“开始”产生数据通信并且可能在一个可选的时间点“停止”。我们提供了开始和停止的2个参数，时间点是用Application Container的方法Start和Stop来设置的，这些方法以“Time”对象为参数。在这种情况下，使用了一种明确的C++转换序列来获得C++双精度(double)的1.0，并且用一个Seconds转换来把它转换到ns-3的Time对象。



这两行会使echo服务应用在1s时开始（生效）并在10s时停止（失效）。既然已经声明了一个模拟事件（就是应用的停止事件）在10s时被执行，模拟至少会持续10s。

下面来设置客户端应用层：



客户端应用的设置与服务器端类似。也有一个UdpEchoClientHelper来管理UdpEchoClientApplication。然而，对于echo客户端，需要设置5个不同的属性。首先2个属性是在UdpEchoClientHelper的构建过程中被设置的。按照助手构造函数的格式，本文把“RemoteAddress”和“RemotePort”属性传递给了助手（实际上是作为助手构造函数的2个必须参数传递的）。

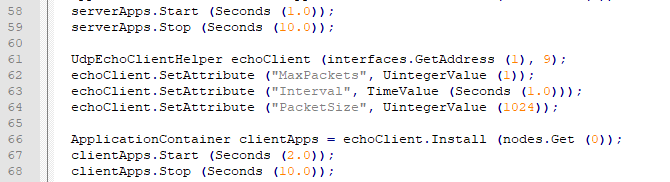
回忆一下使用Ipv4InterfaceContainer来追踪配置给设备的IP地址。在界面容器中位置零的界面对象将会和节点容器中位置零的节点对象对应。同样在界面容器中位置一的界面对象将会和节点容器中位置一的节点对象对应。所以，在上面的第一行代码中，本文创建了一个助手并告诉它设置客户端的远端地址为服务器节点的IP地址。同样告诉它准备发送第二个数据分组到端口9。

“MaxPackets”属性告诉客户端所允许它在模拟期间能发送的最大数据分组个数。“Interval”属性告诉客户端在2个数据分组之间要等待多长的时间，而“PacketSize”属性告诉客户端它的数据分组应该承载多少数据。本例中让客户端发送一个1024byte的数据分组。正如echo服务端一样，告诉echo客户端何时来开始和停止，但是这里本文使客户端在服务端生效1s后才开始（在模拟器中时间2s的时候）。

1. **启动模拟器**



这是用全局变量来运行模拟器。

****

当Simulator::Run被调用时，系统会开始遍历预设事件的列表并执行。首先它会在1.0s时运行事件，这个事件会使echo服务端应用生效。接下来仿真器会运行在t=2.0s时的事件，即让echo客户端应用开始。同样地，这个事件可能会预定更多的其他事件。在echo客户端应用中，开始事件的执行会通过给服务端传送一个数据分组来开始仿真的数据传送阶段。发送一个数据分组给服务端会引发一系列更多的事件。这些事件会被预设在此事件之后，并根据已经在脚本中设定的时间参数来执行数据分组的应答。

我们只发送了一个数据分组，在此之后，那个被单独的客户端应答请求所引发的连锁反应会停止，并且模拟器会进入空闲状态。当这发生时，接下来的事件就是服务端和客户端的Stop事件。当这些事件被执行后，就没有将来的事件来执行了，函数Simulator::Run会返回。整个模拟过程结束。

****

下面剩下的事情就是清理。清理通过调用全局函数来完成，助手函数被执行后，助手安排的钩子函数就被插入到模拟器中来销毁所有被创建的对象，ns-3系统会帮你料理繁杂的任务。

1. **上机实践**

把脚本放到scratch目录下，并运行waf，这样脚本就会被编译运行，其运行结果如图4-1所示。

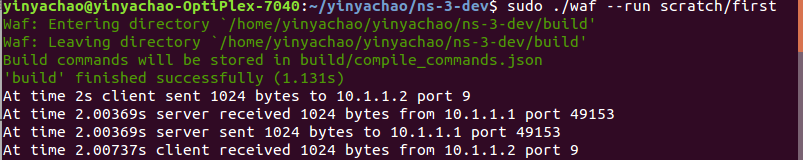


图4-1 first.cc运行结果

编译系统核查文件被编译，接着运行它。可以看到在回显客户端日志构件显示已经通过接口9发送了1024byte到10.1.1.2的服务端，回显服务器端的日志构件显示从10.1.1.1接收到了1024byte。接下来回显服务端应答了数据分组，能看到回显客户端记录它已经接收到从服务端发送过来的回显数据分组。

阅读代码--修改代码—编写代码几乎是程序设计初学者的必经之路，建议自己修改代码中的一些参数，然后执行看看效果，能更好地理解ns-3代码的运行，然后在此基础上学习并尝试搭建其他的网络，如有线、无线局域网等。

1. **CSMA网络中的通信**

示例来源：ns-3.16/examples/tutorial/second.cc。

拓扑结构如图4-2所示。

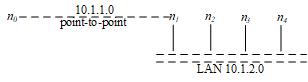
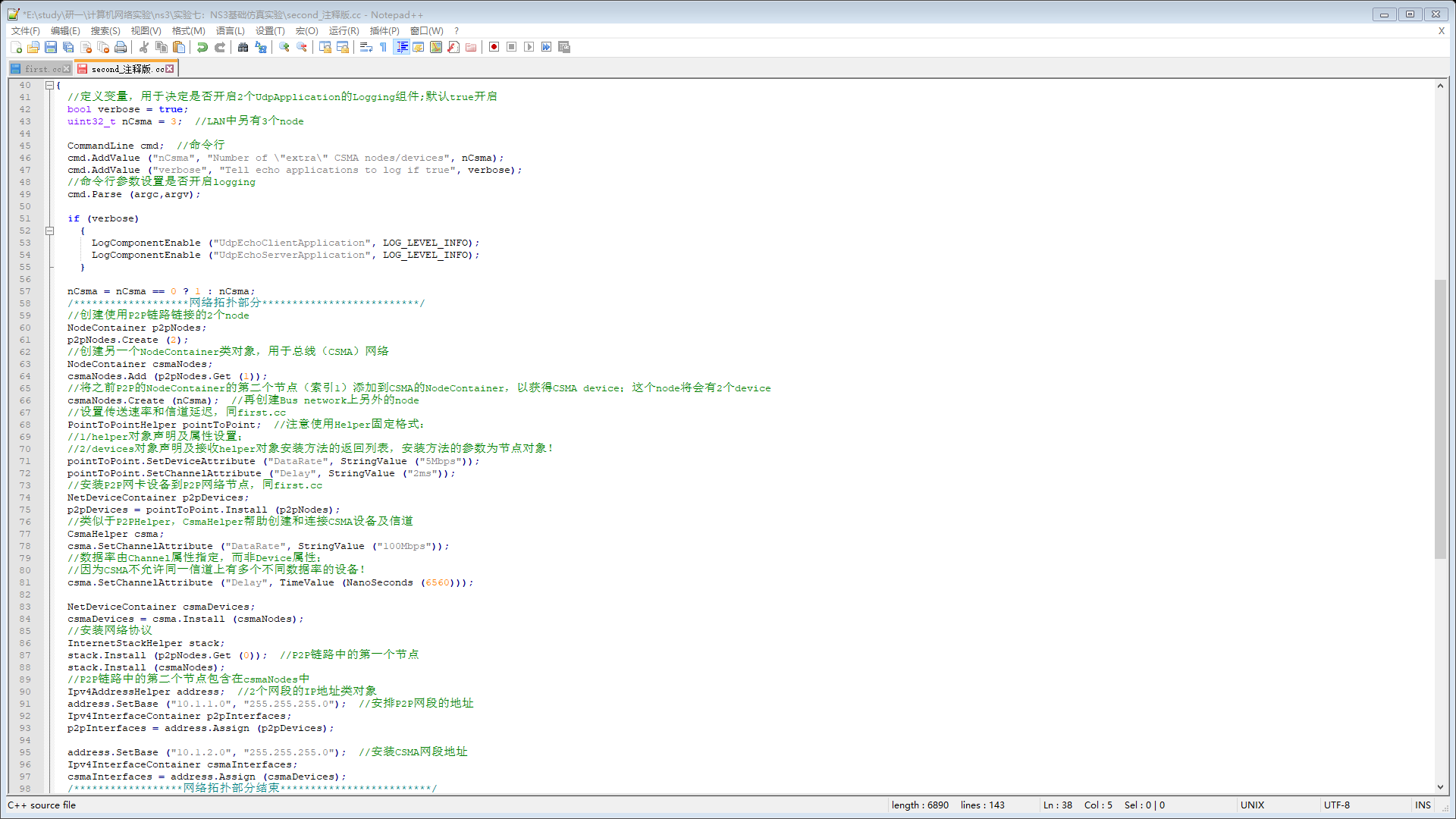
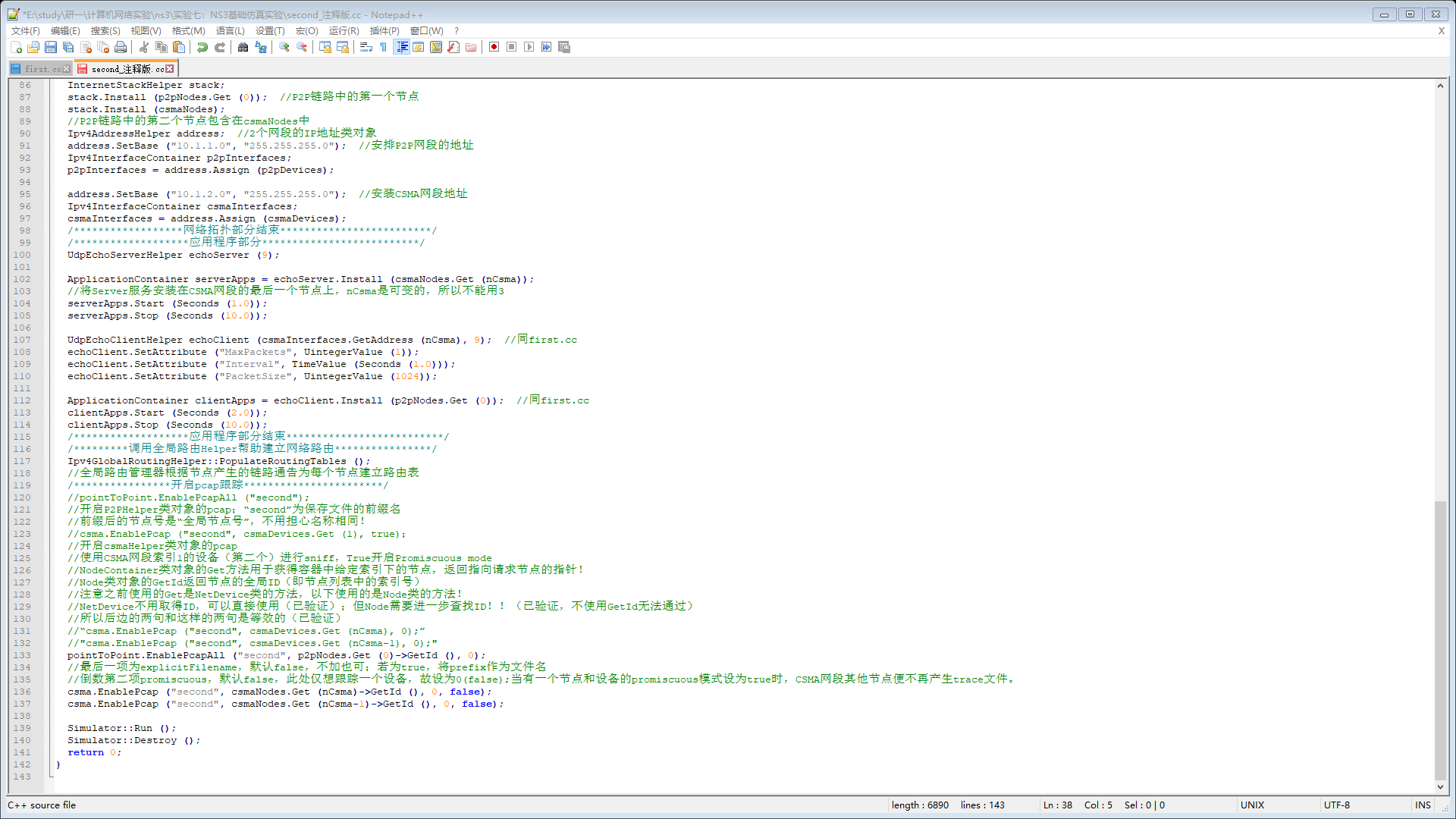


图4-2 second.cc的拓扑结构

n0和n1实现点到点通信，n1~n4是个有线局域网络。网络实现n0经由n1和n4通信。我们希望这个例子能使大家更好地理解前面的关键概念。

下面是部分代码和注释：





在终端中输入编译代码将second.cc进行编译，脚本输出如图4-3所示。从输出信息可以看出UDP回显客户端发送1024byte给地址为10.1.2.4的服务器，服务器在另一个网段（10.1.2.0）上。UDP回显服务器收到来自10.1.1.1客户端的回显数据分组，然后发送相同字节给客户端，而且客户端成功接收。

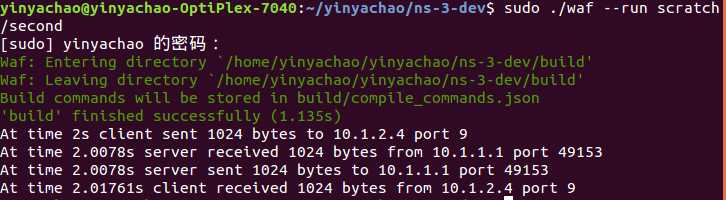


图4-3 脚本输出

回到上一层目录会发现3个跟踪文件（second-0-0.pcap、second-1-0.pcap、second-2-0.pcap）。3个文件的命名具有相同格式，例如，second-0-0.pcap意味着是来自点到点网络设备上节点0、设备0的跟踪文件。如果读者仔细看过前面的拓扑说明，就会看到节点0是点到点链路最左边的节点，节点1既有点到点设备又有CSMA设备。同时会发现节点2是CSMA网络上第一个“额外”节点，其设备0被选为捕捉混杂模式跟踪的设备。

首先查看最左端点到点节点（节点为0），如图4-4所示。从输出可以看到链路类型是PPP（点到点）回显数据分组离开节点0，途径IP地址为10.1.1.1的关联设备，奔向IP地址为10.1.2.4的最右边CSMA节点。这个数据分组移到点至点链路，被节点1上点到点的网络设备接收。

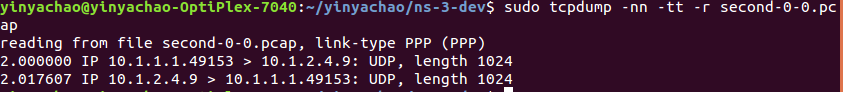


图4-4 节点0

另一个跟踪文件如图4-5所示。在这里链路类型也是PPP，来自IP地址10.1.1.1的数据分组（在2.000 000s被发送到IP地址10.1.2.4）出现在该接口上。在这个节点上，数据分组会被转发到CSMA接口，然后去往最终的目的地。

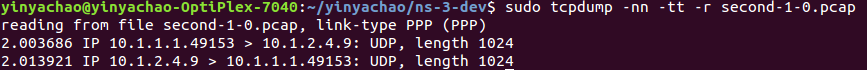


图4-5 另一个跟踪文件

节点2作为CSMA网络混杂嗅探节点的情况如图4-6所示。现在的链路类型为“以太网”，总线网络需要ARP地址解析协议。节点1知道它需要发送数据分组到IP地址10.1.2.4，但它不知道相应节点的MAC地址。于是在CSMA网络（FF:FF:FF:FF:FF:FF）上广播寻找IP地址为10.1.2.4的设备。在这种情况下，最右边的节点回答说其MAC地址是00:00:00:00:00:06。请注意，节点2不直接参与这种交流，但嗅探网络并且报告所有流量。

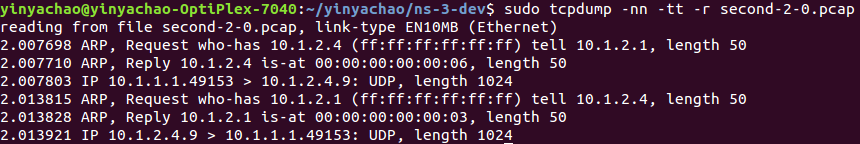


图4-6 节点2作为CSMA网络混杂嗅探节点的情况

节点1、设备1发送回显数据分组给IP地址为10.1.2.4的UDP回显服务器。服务器接收到回显请求后试图将数据分组发回给源头，服务器知道其地址在另一个网络（IP地址为10.1.2.1）上，这是因为初始化了全局路由。但是回显服务器节点不知道第一个CSMA节点的MAC地址，所以如同第一个CSMA节点所做的，它发送了ARP。

# 思考问题

1. **使用NS-3对网络进行仿真相对于使用MATLAB等软件有什么优点？**
2. **基于基础的实验示例，利用NS-3我们还可以进行哪些网络场景的仿真？**