第6章 微调(Tweaking)

本章介绍ns3内置的的一些实用工具，这些工具对于开发网络模拟程序非常有帮助，主要包括：

- 日志模块：用于调试输出
- 命令行参数：用于动态修改模拟配置
- 跟踪系统：用于生成 ASCII 或 PCAP 格式的输出数据

这些工具共同的特点是：**不需要修改ns-3核心代码**就能对网络模拟程序进行**调试、配置和分析。**

由于不需要修改ns3核心源码，因此本章的标题为**微调（“Tweaking”）**。

6.1 日志模块Logging Module

6.1.1 日志模块概述

ns3提供了一种可选择的、多级日志功能模块。

ns3支持的日志功能如下：

- 彻底关闭日志功能
- 全局（Global）激活日志功能
- 基于特定组件来激活日志功能
- 支持多级日志输出（基于输出信息的详细程度）

提示：日志模块主要用途是网络模拟程序的调试。同时，ns3提供了`tracing system`获取仿真模型输出的数据用于研究目的。

日志模块与`tracing system`的侧重点不同：

- **日志系统通常被用来输出：**调试信息、警告信息、错误信息，以及当你希望在脚本或模块中快速输出消息的场合。
- **tracing system:** provide a general purpose mechanism — tracing — to get data out of your models. 例如：捕获数据包

ns3基于输出日志信息的详细程度，提供了7个level的日志级别（从上往下，日志的详细程度递增）：

- LOG_ERROR：输出错误信息，相关的宏：`NS_LOG_ERROR`
- LOG_WARN：输出警告信息，相关的宏：`NS_LOG_WARN`
- LOG_DEBUG：输出调试信息，相关的宏：`NS_LOG_DEBUG`
- LOG_INFO：输出程序执行过程信息，相关的宏：`NS_LOG_INFO`
- LOG_FUNCTION：每次函数调用都输出一条信息，相关的宏：`NS_LOG_FUNCTION`用于成员函数，`NS_LOG_FUNCTION_NOARGS`用于静态函数
- LOG_LOGIC：输出函数中的逻辑流程，相关的宏：`NS_LOG_LOGIC`
- LOG_ALL：输出以上所有的信息，没有对应的宏定义。

以上7个`LOG_TYPE`，每一个日志`TYPE`都有对应的`LOG_LEVEL_TYPE`，例如：LOG_DEBUG <---> LOG_LEVEL_DEBUG.

它们的区别是，`LOG_TYPE`仅输出指定level的日志，`LOG_LEVEL_TYPE`除了输出指定level的日志信息之外，还要输出比指定level低的level的日志信息。举例说明：

- 激活`LOG_INFO`：仅输出`NS_LOG_INFO`宏定义的日志信息；
- 当激活`LOG_LEVEL_INFO`时，将输出`NS_LOG_INFO`，`NS_LOG_DEBUG`,`NS_LOG_WARN`,`NS_LOG_ERROR`定义的所有日志信息；（ns-3.45实践中不会输出NS_LOG_DEBUG定义的日志信息）
- 当激活`LOG_LEVEL_DEBUG`时，输出所有的日志信息（包括：`NS_LOG_LOGIC`,`NS_LOG_FUNCTION`,`NS_LOG_INFO`,`NS_LOG_DEBUG`,`NS_LOG_WARN`,`NS_LOG_ERROR`）；
- 激活`LOG_ERROR`等价于`LOG_LEVEL_ERROR`；显而易见：ERROR之前没有啥能输出的了::)
- 激活`LOG_ALL`等价于`LOG_LEVEL_ALL`；

查阅源代码：`src/core/model/log.h`

此外，ns3提供了一个特殊的宏定义`NS_LOG_UNCOND`，使用此宏定义的日志总是会被显示，不受日志level的影响。

6.1.2 启用日志

scratch/myfirst.cc

```
/*
 * SPDX-License-Identifier: GPL-2.0-only
 */

#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"

// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1
//    point-to-point
//

using namespace ns3;
```

```
19
20  NS_LOG_COMPONENT_DEFINE("FirstScriptExample");
21
22  int
23  main(int argc, char* argv[])
24  {
25      CommandLine cmd(__FILE__);
26      cmd.Parse(argc, argv);
27
28      Time::SetResolution(Time::NS);
29      LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
30      LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
31
32      NodeContainer nodes;
33      nodes.Create(2);
34
35      PointToPointHelper pointToPoint;
36      pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
37      pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
38
39      NetDeviceContainer devices;
40      devices = pointToPoint.Install(nodes);
41
42      InternetStackHelper stack;
43      stack.Install(nodes);
44
45      Ipv4AddressHelper address;
46      address.SetBase("10.1.1.0", "255.255.255.0");
47
48      Ipv4InterfaceContainer interfaces = address.Assign(devices);
49
50      UdpEchoServerHelper echoServer(9);
51
52      ApplicationContainer serverApps = echoServer.Install(nodes.Get(1));
53      serverApps.Start(Seconds(1));
54      serverApps.Stop(Seconds(10));
55
56      UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);
57      echoClient.SetAttribute("MaxPackets", UintegerValue(1));
58      echoClient.SetAttribute("Interval", TimeValue(Seconds(1)));
59      echoClient.SetAttribute("PacketSize", UintegerValue(1024));
60
61      ApplicationContainer clientApps = echoClient.Install(nodes.Get(0));
62      clientApps.Start(Seconds(2));
63      clientApps.Stop(Seconds(10));
64
65      Simulator::Run();
66      Simulator::Destroy();
67      return 0;
68  }
```

使用NS_LOG环境变量启用日志

使用 `NS_LOG` 环境变量来打开日志功能。

```
$ ./ns3 run scratch/myfirst
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

上面输出中的发送和接收日志信息分别来自`UdpEchoClientApplication`和`UdpEchoServerApplication`。我们可以通过命令行的环境变量（NS_LOG）来告诉客户端应用程序打印更详细的日志信息。

scratch/myfirst.cc中的代码：

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

启用**LOG_LEVEL_INFO** level的日志， 比 `LOG_LEVEL_INFO` 更低Level的日志（`NS_LOG_DEBUG,NS_LOG_WARN and NS_LOG_ERROR`）也将被启用。

这种将日志level直接写到代码中的方式，灵活性差，如果要修改激活的日志level，模拟脚本需要重新编译。

更简便的修改日志level的方法：可以通过设置`NS_LOG`环境变量,而无需直接修改仿真脚本的代码。

几种使用方法举例：

```
$ export NS_LOG=UdpEchoClientApplication=level_all
$ export NS_LOG=UdpEchoClientApplication
$ export NS_LOG=UdpEchoClientApplication=level_info
$ export NS_LOG=UdpEchoClientApplication=info  # 注意区分：level_info
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func'  # 在日志信息前显示对应的组件名
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func:UdpEchoServerApplication=level_all|prefix_func'  # 设置多个组件的日志level
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func|prefix_time:UdpEchoServerApplication=level_all|prefix_func|prefix_time'  # 显示组件名和时间前缀
$ export 'NS_LOG=*=level_all|prefix_func|prefix_time' # 使用通配符匹配所有组件的日志配置
```

> 注意：
>
> 1. 上面的示例中如存在"|"字符时需要加引号，因为在shell中“|”是管道符号。
> 2. prefix_func、prefix_time：我怎么知道有这些东东？？？ **查阅源代码**：`src/core/model/log.h` 大概在118行

具体的运行实例（对于UdpEchoClientApplication的 level_all 和 level_function 的输出结果相同）：

```
$ export NS_LOG=UdpEchoClientApplication=level_all
$ ./ns3 run scratch/myfirst --no-build
UdpEchoClientApplication:UdpEchoClient(0x608999b58dc0)
UdpEchoClientApplication:SetRemote(0x608999b58dc0, )
UdpEchoClientApplication:SetPort(0x608999b58dc0, 0)
```

```
UdpEchoClientApplication:SetDataSize(0x608999b58dc0, 1024)
UdpEchoClientApplication:SetRemote(0x608999b58dc0, 04-06-0a:01:01:02:09:00)
UdpEchoClientApplication:SetPort(0x608999b58dc0, 0)
UdpEchoClientApplication:StartApplication(0x608999b58dc0)
UdpEchoClientApplication:ScheduleTransmit(0x608999b58dc0, +0ns)
UdpEchoClientApplication:Send(0x608999b58dc0)
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
UdpEchoClientApplication:HandleRead(0x608999b58dc0, 0x608999c05d30)
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
UdpEchoClientApplication:StopApplication(0x608999b58dc0)
UdpEchoClientApplication:~UdpEchoClient(0x608999b58dc0)

$ export NS_LOG=UdpEchoClientApplication=level_function
$ ./ns3 run scratch/myfirst --no-build
UdpEchoClientApplication:UdpEchoClient(0x56b721de6dc0)
UdpEchoClientApplication:SetRemote(0x56b721de6dc0, )
UdpEchoClientApplication:SetPort(0x56b721de6dc0, 0)
UdpEchoClientApplication:SetDataSize(0x56b721de6dc0, 1024)
UdpEchoClientApplication:SetRemote(0x56b721de6dc0, 04-06-0a:01:01:02:09:00)
UdpEchoClientApplication:SetPort(0x56b721de6dc0, 0)
UdpEchoClientApplication:StartApplication(0x56b721de6dc0)
UdpEchoClientApplication:ScheduleTransmit(0x56b721de6dc0, +0ns)
UdpEchoClientApplication:Send(0x56b721de6dc0)
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
UdpEchoClientApplication:HandleRead(0x56b721de6dc0, 0x56b721e93d30)
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
UdpEchoClientApplication:StopApplication(0x56b721de6dc0)
UdpEchoClientApplication:~UdpEchoClient(0x56b721de6dc0)

$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func'
$ ./ns3 run scratch/myfirst --no-build
UdpEchoClientApplication:UdpEchoClient(0x6445bdeafdf0)
UdpEchoClientApplication:SetRemote(0x6445bdeafdf0, )
UdpEchoClientApplication:SetPort(0x6445bdeafdf0, 0)
UdpEchoClientApplication:SetDataSize(0x6445bdeafdf0, 1024)
UdpEchoClientApplication:SetRemote(0x6445bdeafdf0, 04-06-0a:01:01:02:09:00)
UdpEchoClientApplication:SetPort(0x6445bdeafdf0, 0)
UdpEchoClientApplication:StartApplication(0x6445bdeafdf0)
UdpEchoClientApplication:ScheduleTransmit(0x6445bdeafdf0, +0ns)
UdpEchoClientApplication:Send(0x6445bdeafdf0)
UdpEchoClientApplication:Send(): At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
UdpEchoClientApplication:HandleRead(0x6445bdeafdf0, 0x6445bdf5cc20)
UdpEchoClientApplication:HandleRead(): At time +2.00737s client received 1024 bytes
from 10.1.1.2 port 9
UdpEchoClientApplication:StopApplication(0x6445bdeafdf0)
UdpEchoClientApplication:~UdpEchoClient(0x6445bdeafdf0)

```

```
$ export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func|prefix_time'
$ ./ns3 run scratch/myfirst --no-build
+0.000000000s UdpEchoClientApplication:UdpEchoClient(0x58b70d657e10)
+0.000000000s UdpEchoClientApplication:SetRemote(0x58b70d657e10, )
+0.000000000s UdpEchoClientApplication:SetPort(0x58b70d657e10, 0)
+0.000000000s UdpEchoClientApplication:SetDataSize(0x58b70d657e10, 1024)
+0.000000000s UdpEchoClientApplication:SetRemote(0x58b70d657e10, 04-06-
0a:01:01:02:09:00)
+0.000000000s UdpEchoClientApplication:SetPort(0x58b70d657e10, 0)
+2.000000000s UdpEchoClientApplication:StartApplication(0x58b70d657e10)
+2.000000000s UdpEchoClientApplication:ScheduleTransmit(0x58b70d657e10, +0ns)
+2.000000000s UdpEchoClientApplication:Send(0x58b70d657e10)
+2.000000000s UdpEchoClientApplication:Send(): At time +2s client sent 1024 bytes to
10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
+2.007372800s UdpEchoClientApplication:HandleRead(0x58b70d657e10, 0x58b70d704c40)
+2.007372800s UdpEchoClientApplication:HandleRead(): At time +2.00737s client received
1024 bytes from 10.1.1.2 port 9
+10.000000000s UdpEchoClientApplication:StopApplication(0x58b70d657e10)
UdpEchoClientApplication:~UdpEchoClient(0x58b70d657e10)
```

注意：

- ns3并不强制模型一定要提供日志功能，由模型开发者自行处理是否添加日志功能。通常，官方提供的模型都提供了日志功能。

也可以将日志转存到文件中而不再终端中打印，例如：运行脚本，并将日志存入文件log.out，如果出现错误则打印到屏幕输出：

```
$ ./ns3 run scratch/myfirst > log.out 2>&1
```

6.1.3 代码中加入日志功能

上面介绍了如何显示系统模块当中内置的日志信息，那么在开发脚本或者模块时，研究人员也可以通过NS3内置的几个宏来自定义自己的日志信息。

scratch/myfirst.cc中的：`NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");` 定义了一个名称为"FirstScriptExample"的日志组件，我们通过环境变量来激活此日志组件：

scratch/myfirst.cc中添加如下代码：

```
int
main (int argc, char *argv[])
{
  NS_LOG_INFO("My First ns3 Simulation Script INFO......");
  NS_LOG_ERROR("My First ns3 Simulation Script Error......");
  CommandLine cmd (__FILE__);
  cmd.Parse (argc, argv);

  ........

}
```

设置环境变举例：

```
$ export NS_LOG=""
$ export 'NS_LOG=FirstScriptExample=level_error'
$ export 'NS_LOG=FirstScriptExample=level_info'
$ export 'NS_LOG=FirstScriptExample=error'
$ export 'NS_LOG=FirstScriptExample=info'
```

> info和error仅显示代码中对应的宏定义的日志信息，level_info则会显示NS_LOG_ERROR和NS_LOG_INFO定义的日志信息。

运行脚本：

```
$ ./ns3 build   # 编译那些未编译的，或者上次编译后发生变化的程序。同：./ns3
[  0%] Building CXX object scratch/CMakeFiles/scratch_myfirst.dir/myfirst.cc.o
[  0%] Linking CXX executable /home/ns3/workspace/ns-allinone-3.45/ns-3.45/build/scratch/ns3.45-myfirst-default
Finished executing the following commands:
/usr/bin/cmake --build /home/ns3/workspace/ns-allinone-3.45/ns-3.45/cmake-cache -j 1
$ export NS_LOG=FirstScriptExample=info
$ ./ns3 run scratch/myfirst --no-build  # 加上 --no-build 执行会快一点
My First ns3 Simulation Script INFO......
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
$ export NS_LOG=FirstScriptExample=level_info
$ ./ns3 run scratch/myfirst --no-build
My First ns3 Simulation Script INFO......
My First ns3 Simulation Script Error......
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9

```

6.2 使用命令行参数

6.2.1 覆盖默认属性值

命令行参数是ns3提供的另一种不用编辑和重新构建仿真脚本就可以改变脚本行为的方法。

首先，需要在主程序中声明命令行解析器：

```
int
main (int argc, char *argv[])
{
  ...
  CommandLine cmd;
  cmd.Parse (argc, argv);
  ...
}
```

构建和运行脚本： `$ ./ns3 run "scratch/myfirst --PrintHelp"`

运行时，会将`--PrintHelp`传递给myfirst脚本，脚本中的命令行解析器将读取到`--PrintHelp`参数并响应此参数（打印出帮助信息）：

```
$ ./ns3 run "scratch/myfirst --PrintHelp" --no-build
myfirst [General Arguments]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintVersion:              Print the ns-3 version.
    --PrintHelp:                 Print this help message.

```

`--PrintGlobals`：打印当前仿真环境中所有可用的全局变量（全局属性）。

`--PrintGroups`：显示所有的模块分组。例如：`Network`，`Internet`，`Applications`，`Mobility`，`Wifi`等等。

`--PrintGroup=[group]`：显示指定模块组内所有的`TypeIds`

`--PrintTypeIds`：打印所有的`TypeIds`

重点来看`--PrintAttributes`：打印指定typeid的属性以及这些属性的默认值。例如：

例1：

```
$ ./ns3 run "scratch/myfirst --PrintAttributes=ns3::PointToPointNetDevice"
Attributes for TypeId ns3::PointToPointNetDevice
    --ns3::PointToPointNetDevice::Address=[ff:ff:ff:ff:ff:ff]
        The MAC address of this device.
    --ns3::PointToPointNetDevice::DataRate=[32768bps]
        The default data rate for point to point links
```

```
    --ns3::PointToPointNetDevice::InterframeGap=[+0ns]
        The time to wait between packet (frame) transmissions
    --ns3::PointToPointNetDevice::Mtu=[1500]
        The MAC-level Maximum Transmission Unit
    --ns3::PointToPointNetDevice::ReceiveErrorModel=[0]
        The receiver error model used to simulate packet loss
    --ns3::PointToPointNetDevice::TxQueue=[0]
        A queue to use as the transmit queue in the device.
```

例2:

```
$ ./ns3 run "scratch/myfirst --PrintAttributes=ns3::PointToPointChannel"
Attributes for TypeId ns3::PointToPointChannel
    --ns3::PointToPointChannel::Delay=[+0ns]
        Propagation delay through the channel
Attributes defined in parent class ns3::Channel
    --ns3::Channel::Id=[0]
        The id (unique integer) of this Channel.
```

例3:

```
$ ./ns3 run "scratch/myfirst --PrintAttributes=ns3::CsmaChannel"
Attributes for TypeId ns3::CsmaChannel
    --ns3::CsmaChannel::DataRate=[4294967295bps]
        The transmission data rate to be provided to devices connected to the channel
    --ns3::CsmaChannel::Delay=[+0ns]
        Transmission delay through the channel
Attributes defined in parent class ns3::Channel
    --ns3::Channel::Id=[0]
        The id (unique integer) of this Channel.
```

从上面的例子中可以查看到相关属性的默认值，如果在代码中不显式的指定相应的值，会使用默认值。将scratch/myfirst.cc中如下代码注释掉:

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

构建并运行脚本，对比前后输出结果的差异（**网络设备的速率和信道延时都使用默认值**）：

```
$ vim script/myfirst.cc
$ export 'NS_LOG=UdpEchoServerApplication=level_all|prefix_time'
$ ./ns3 run scratch/myfirst --no-build
+0.000000000s UdpEchoServerApplication:UdpEchoServer(0x56b47fb756a0)
+1.000000000s UdpEchoServerApplication:StartApplication(0x56b47fb756a0)
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
+2.257324219s UdpEchoServerApplication:HandleRead(0x56b47fb756a0, 0x56b47fb69120)
+2.257324219s At time +2.25732s server received 1024 bytes from 10.1.1.1 port 49153
+2.257324219s Echoing packet
+2.257324219s At time +2.25732s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.51465s client received 1024 bytes from 10.1.1.2 port 9
+10.000000000s UdpEchoServerApplication:StopApplication(0x56b47fb756a0)
UdpEchoServerApplication:~UdpEchoServer(0x56b47fb756a0)
```

怎么查询组件有哪些属性：

```
$ ./ns3 run "scratch/myfirst --help"
myfirst [General Arguments]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintVersion:              Print the ns-3 version.
    --PrintHelp:                 Print this help message.

$ ./ns3 run "scratch/myfirst --PrintAttributes=ns3::PointToPointNetDevice" --no-build
Attributes for TypeId ns3::PointToPointNetDevice
    --ns3::PointToPointNetDevice::Address=[ff:ff:ff:ff:ff:ff]
        The MAC address of this device.
    --ns3::PointToPointNetDevice::DataRate=[32768bps]
        The default data rate for point to point links
    --ns3::PointToPointNetDevice::InterframeGap=[+0ns]
        The time to wait between packet (frame) transmissions
    --ns3::PointToPointNetDevice::Mtu=[1500]
        The MAC-level Maximum Transmission Unit
    --ns3::PointToPointNetDevice::ReceiveErrorModel=[0]
        The receiver error model used to simulate packet loss
    --ns3::PointToPointNetDevice::TxQueue=[0]
        A queue to use as the transmit queue in the device.
```

通过命令行参数来设置组件的属性：

```
$ ./ns3 run "scratch/myfirst --PrintAttributes=ns3::PointToPointChannel" --no-build
Attributes for TypeId ns3::PointToPointChannel
    --ns3::PointToPointChannel::Delay=[+0ns]
        Propagation delay through the channel
Attributes defined in parent class ns3::Channel
    --ns3::Channel::Id=[0]
        The id (unique integer) of this Channel.
```

```
$ ./ns3 run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5Mbps --ns3::PointToPointChannel::Delay=2ms" --no-build
+0.000000000s UdpEchoServerApplication:UdpEchoServer(0x5da0f6fc16a0)
+1.000000000s UdpEchoServerApplication:StartApplication(0x5da0f6fc16a0)
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
+2.003686400s UdpEchoServerApplication:HandleRead(0x5da0f6fc16a0, 0x5da0f6fb5120)
+2.003686400s At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
+2.003686400s Echoing packet
+2.003686400s At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
+10.000000000s UdpEchoServerApplication:StopApplication(0x5da0f6fc16a0)
UdpEchoServerApplication:~UdpEchoServer(0x5da0f6fc16a0)
```

通过命令行来修改属性的默认值:

```
# 注意加引号
# 指定一个参数值
$ ./ns3 run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5Mbps"

# 指定两个参数值
$ ./ns3 run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5Mbps --ns3::PointToPointChannel::Delay=2ms"

## 指定三个参数的值
$ ./ns3 run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5Mbps --ns3::PointToPointChannel::Delay=2ms --ns3::UdpEchoClient::MaxPackets=2"
```

如何查询到这些参数列表?

可以查询ns-3的 Doxygen文档（https://www.nsnam.org/docs/release/3.45/doxygen/d3/d79/_attribute_list.html），或者在命令行使用如下的方法:

首先，打印帮助信息:

```
$ ./ns3 run "scratch/myfirst --PrintHelp"
myfirst [General Arguments]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintVersion:              Print the ns-3 version.
    --PrintHelp:                 Print this help message.

```

然后，使用`--PrintGroups:`查询有哪些分组:

```
$ ./ns3 run "scratch/myfirst --PrintGroups"
Registered TypeId groups:

```

```
    Antenna
    Aodv
    Applications
    Bridge
    Buildings
    ConfigStore
    Core
    Csma
    Dsdv
    Dsr
    Energy
    FdNetDevice
    FlowMonitor
    Internet
    Internet-Apps
    LrWpan
    Lte
    Mesh
    Mobility
    Network
    NixVectorRouting
    Olsr
    PointToPoint
    Propagation
    SixLowPan
    Spectrum
    Stats
    TapBridge
    TopologyReader
    TrafficControl
    Uan
    VirtualNetDevice
    Wifi
    Wimax
    Zigbee
```

然后，使用`--PrintGroup`查询分组内的组件：

```
$ ./ns3 run "scratch/myfirst --PrintGroup=Zigbee"
TypeIds in group Zigbee:
    ns3::zigbee::ZigbeeNwk
    ns3::zigbee::ZigbeeStack
```

最后，使用`--PrintAttributes`查询具体组件的属性列表：

```
$ ./ns3 run "scratch/myfirst --PrintAttributes=ns3::zigbee::ZigbeeNwk"
Attributes for TypeId ns3::zigbee::ZigbeeNwk
    --ns3::zigbee::ZigbeeNwk::MaxPendingTxQueueSize=[10]
        The maximum size of the table storing pending packets awaiting to be
transmitted after discovering a route to the destination.
    --ns3::zigbee::ZigbeeNwk::NwkcCoordinatorCapable=[true]
```

```
        [Constant] Indicates whether the device is capable of becoming aZigbee
coordinator.
    --ns3::zigbee::ZigbeeNwk::NwkcInitialRREQRetries=[3]
        [Constant] The number of times the first broadcast transmissionof a RREQ cmd
frame is retried.
    --ns3::zigbee::ZigbeeNwk::NwkcMaxRREQJitter=[128]
        [Constant] The duration between retries of a broadcast RREQ (msec)
    --ns3::zigbee::ZigbeeNwk::NwkcMinRREQJitter=[2]
        [Constant] The minimum jitter for broadcast retransmission of a RREQ (msec)
    --ns3::zigbee::ZigbeeNwk::NwkcProtocolVersion=[2]
        [Constant] The version of the Zigbee NWK protocol inthe device (placeholder)
    --ns3::zigbee::ZigbeeNwk::NwkcRREQRetries=[2]
        [Constant] The number of times the broadcast transmission of aRREQ cmd frame is
retried on relay by intermediate router orcoordinator.
    --ns3::zigbee::ZigbeeNwk::NwkcRREQRetryInterval=[+2.54e+08ns]
        [Constant] The duration between retries of a broadcast RREQ cmd frame.
    --ns3::zigbee::ZigbeeNwk::NwkcRouteDiscoveryTime=[+1e+10ns]
        [Constant] The duration until a route discovery expires
```

6.2.2 DIY自己的命令行参数

(1) 仿真脚本中示例代码:

```
int
main (int argc, char *argv[])
{
  uint32_t nPackets = 1;     //参数的默认值
  CommandLine cmd;
  cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
  cmd.Parse (argc, argv);
  ...
  echoClient.SetAttribute("MaxPackets", UintegerValue(nPackets));
  ...
}
```

(2) 查看新增的命令行参数

```
$ vim scratch/myfirst.cc
$ ./ns3 build
[  0%] Building CXX object scratch/CMakeFiles/scratch_myfirst.dir/myfirst.cc.o
[  0%] Linking CXX executable /home/ns3/workspace/ns-allinone-3.45/ns-
3.45/build/scratch/ns3.45-myfirst-default
Finished executing the following commands:
/usr/bin/cmake --build /home/ns3/workspace/ns-allinone-3.45/ns-3.45/cmake-cache -j 1
$ ./ns3 run "scratch/myfirst --PrintHelp"
myfirst [Program Options] [General Arguments]

Program Options:
    --nPackets:  Number of packets to echo [1]

General Arguments:
    --PrintGlobals:              Print the list of globals.
```

```
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintVersion:              Print the ns-3 version.
    --PrintHelp:                 Print this help message.
```

`Program Options:`中列出的即为程序中添加的参数名称和默认值。

（3）覆盖默认参数值

`./ns3 run "scratch/myfirst --nPackets=3"`

小结：

- ns-3用户：通过命令行参数控制全局值、模型中组件的属性（Attributes）
- ns-3模型开发者：为模型组件对象添加新的属性（Attributes）；ns3模型用户：通过ns-3的命令行系统来设置属性的值。
- ns-3脚本（网络模拟程序）作者：为网络模拟程序添加新的新的变量，变量的值可以在命令行中设置。

6.3 使用追踪系统

网络模拟的重点是生成输出网络模拟数据以供进一步研究。

生成输出内容的几种方式：

（1）可以使用C++的标准输出功能

ns-3模拟程序是C++程序，因此可以使用C++中惯用方式来生成输出内容。

```cpp
#include <iostream>
...
int main ()
{
...
std::cout << "The value of x is " << x << std::endl;
...
}
```

（2）使用ns-3的日志模块

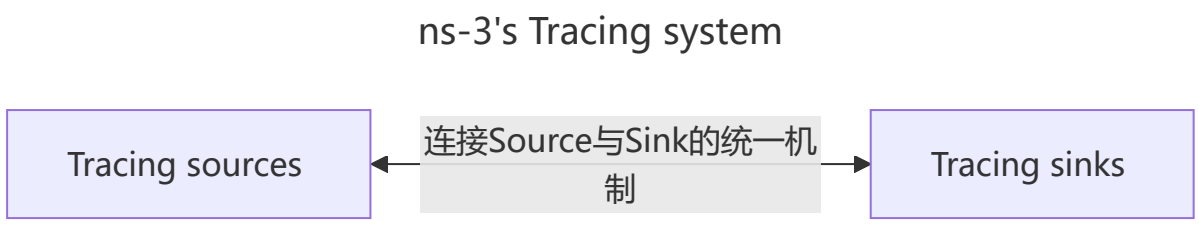
（3）事件追踪子系统（event tracing subsystem）

ns-3使用tracing system来实现输出网络模拟信息，其主要目的：

- 对于基本任务（例如：观察数据包的发送/接收、监测协议状态变化、或者测量吞吐量等），跟踪系统应允许用户为常用的、流行的跟踪源（ns3中那些最常用、最重要的组件。例如，网络设备：例如网卡、通信信道Channel、协议：例如TCP、UDP，等核心模块）生成标准跟踪（例如：将数据包信息以 ASCII 文件或 Pcap 格式输出），并自定义从哪些对象中生成跟踪信息。
- 中级用户能够扩展跟踪系统，以修改生成的输出格式，或插入新的跟踪源，而无需修改模拟器的核心。
    - 可以编写新的 **Tracing Sinks**，例如，他们可以决定不使用默认的 ASCII 格式，而是编写一个 C++ 函数来接收追踪数据，并将数据直接写入 CSV 文件、JSON 格式，或者实时发送到另一个分析工具，所有这些操作都通过**连接**到现有的追踪源来实现。

    - 可以在 **应用程序层** 或 **自定义模块** 中定义新的追踪源。例如，如果用户编写了一个新的应用程序模型，可以在这个新模型中定义一个追踪源，而无需修改 ns-3 现有的协议栈或设备模型。
- 高级用户可以修改模拟器核心以添加新的跟踪源和接收器。

ns-3 的跟踪系统是建立在独立的追踪源（tracing sources）和追踪接收器（tracing sinks） 概念之上，并用一种统一的机制建立`sources`与`sinks`之间的连接 。



tracing source：跟踪源（Trace sources）是能够发出仿真中发生的事件信号，并提供对感兴趣的底层数据进行访问的实体。例如，一个跟踪源可以指示网络设备何时接收到数据包，并为感兴趣的跟踪接收器（trace sinks）提供对数据包内容的访问。

> **提示：**跟踪源本身并没有什么作用，它们必须被"连接"到其他代码片段中才能发挥作用，这些连接了tracing sources的代码利用tracing sinks提供的信息做一些有意义的事情。

trace sinks：跟踪接收器（Trace sinks）是跟踪源提供的事件和数据的消费者。例如，我们可以创建一个跟踪接收器，它（当连接到前述示例的跟踪源时）能够打印出接收到的数据包中感兴趣的部分。

这种明确划分的基本原理是：**允许用户将新型接收器连接到现有跟踪源，而无需编辑和重新编译模拟器的核心。**因此，在上面的示例中，用户可以在脚本中定义新的tracing sink，并通过仅编辑用户脚本将其附加到`simulation core`中定义的现有tracing sources。

本教程通过几个预定义的跟踪源（tracing sources）和接收器（tracing sinks）演示如何进行定制化开发。

有关高级跟踪配置的信息，请参阅ns-3手册或操作步骤部分，包括扩展跟踪名称空间和创建新的跟踪源。

6.3.1 ASCII Tracing

在`scratch/myfirst.cc`的`Simulator::Run();`之前添加如下的代码：

```
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("myfirst.tr"));
```

运行（默认保存输出文件到ns3最上层目录，即ns3脚本所在的目录）：`./ns3 run scratch/myfirst`

使用`--cwd`将输出文件保存到其他位置：`$ ./ns3 run scratch/myfirst --cwd=./other_out_dir`

解析Ascii Traces文件

```
+ 2 /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)
- 2 /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Dequeue ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)
r 2.25732 /NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacRx ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)
+ 2.25732 /NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.2 > 10.1.1.1) ns3::UdpHeader (length: 1032 9 > 49153) Payload (size=1024)
- 2.25732 /NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Dequeue ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.2 > 10.1.1.1) ns3::UdpHeader (length: 1032 9 > 49153) Payload (size=1024)
r 2.51465 /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/MacRx ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.2 > 10.1.1.1) ns3::UdpHeader (length: 1032 9 > 49153) Payload (size=1024)
```

每一行代表一个“trace event”。本例中，我们正在跟踪网络模拟中每个点对点网络设备中存在的传输队列上的事件。

每一行的第一个字符的含义：

- +: An enqueue operation occurred on the device queue; 入队列
- -: A dequeue operation occurred on the device queue; 出队列
- d: A packet was dropped, typically because the queue was full; 包被丢弃，典型的原因：队列满
- r: A packet was received by the net device. 物理设备接收到数据包

6.3.2 PCAP Tracing

输出`.pcap`文件格式，可以在Wireshark、tcpdump中打开解析。

将如下代码添加到`scratch/myfirst.cc`的`Simulator::Run();`之前：

```
pointToPoint.EnablePcapAll ("myfirst");
```

运行(将输出保存到`/home/ns3`目录,不加`--cwd`时默认保存输出文件到当前目录)：

`$ ./ns3 run scratch/myfirst --cwd=out`

生成文件：`myfirst-0-0.pcap`, `myfirst-1-0.pcap`.

使用tcpdump读取pcap文件：`tcpdump -r out/myfirst-0-0.pcap`：

```
$ tcpdump  -r out/myfirst-0-0.pcap
reading from file out/myfirst-0-0.pcap, link-type PPP (PPP)
```

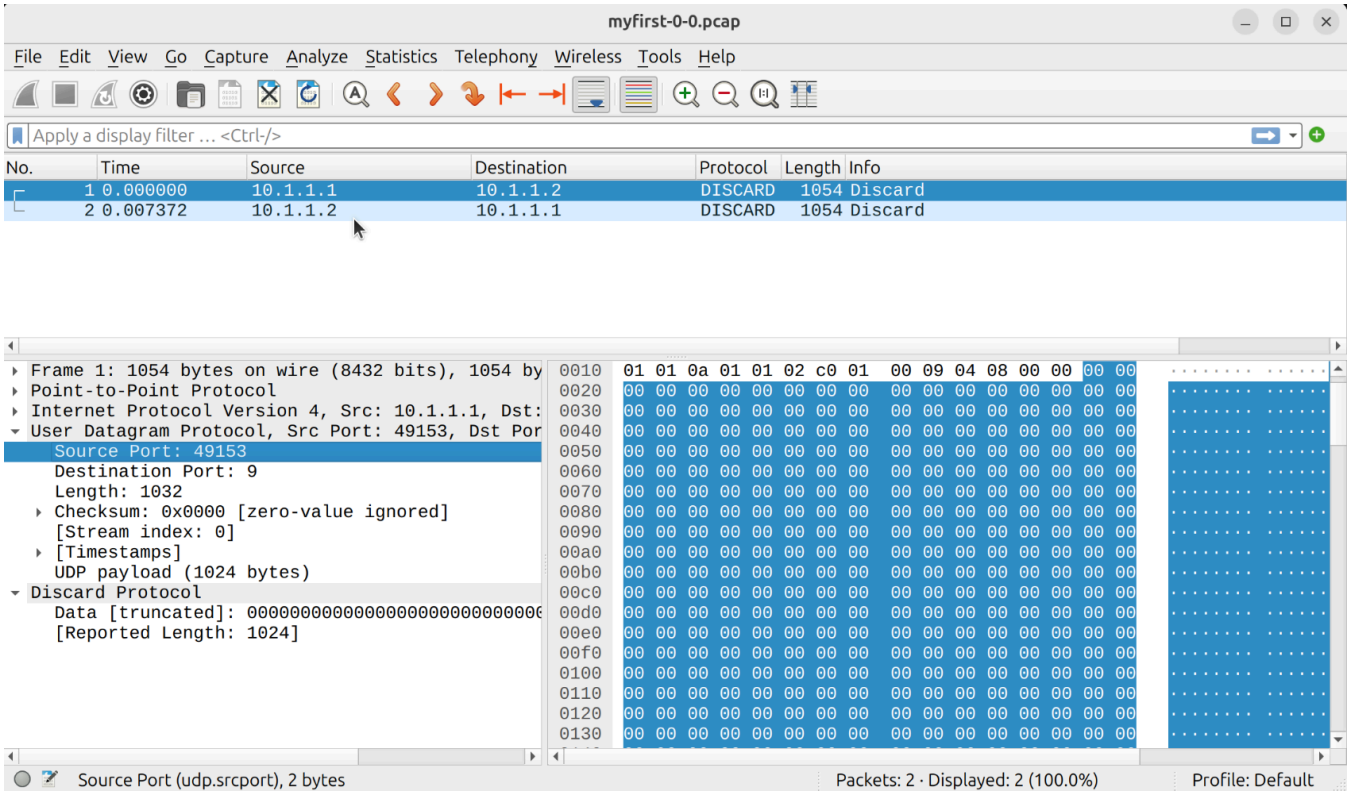
```
08:00:02.000000 IP 10.1.1.1.49153 > 10.1.1.2.discard: UDP, length 1024
08:00:02.514648 IP 10.1.1.2.discard > 10.1.1.1.49153: UDP, length 1024
08:00:03.000000 IP 10.1.1.1.49153 > 10.1.1.2.discard: UDP, length 1024
08:00:03.514648 IP 10.1.1.2.discard > 10.1.1.1.49153: UDP, length 1024

$ tcpdump -nn  -r out/myfirst-0-0.pcap
reading from file out/myfirst-0-0.pcap, link-type PPP (PPP)
08:00:02.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
08:00:02.514648 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
08:00:03.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
08:00:03.514648 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024

$ tcpdump -nn -tt -r out/myfirst-0-0.pcap
reading from file out/myfirst-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.514648 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
3.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
3.514648 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

- `-nn`：关闭主机名和端口号的解析，只显示原始的 IP 地址和数字端口号。
- tcpdump关于时间的参数：
  - `-t`：不显示时间戳
  - `-tt`：显示自纪元以来的秒数（不做本地化local time转化）
  - `-ttt`：显示相对时间间隔
- `-tt`：以 Unix 时间戳格式显示时间（时间基点为：`1970年1月1日 00:00:00 UTC`，即相对于该时间基点流逝了多长时间，通常以秒计数，可以更精确），便于后续处理和分析。
  - 时间格式：**时:分:秒.微秒**，例如：`08:00:02.000000`，表示：
    - **08**：小时（08点，即上午8点）
    - **00**：分钟
    - **02**：秒
    - **000000**：微秒（这里是0微秒）
  - 如果系统时钟的时区为中国的东八区，tcpdump默认的输出的时间最前面是`08`，该数字对应的是东八区。如果把系统时区做调整，则tcpdump的输出也会变化。这是因为：ns-3中的`网络模拟时间的0`会被映射到现实世界中的`Unix 时间戳 0`，如果以local time的方式显示时，第一个数字会根据当前系统设置的时区而变化。

也可以使用wireshark GUI（http://www.wireshark.org/）工具查看PCAP文件。


myfirst-0-0.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 10.1.1.1 10.1.1.2 DISCARD 1054 Discard
2 0.007372 10.1.1.2 10.1.1.1 DISCARD 1054 Discard
Frame 1: 1054 bytes on wire (8432 bits), 1054 by
Point-to-Point Protocol
Internet Protocol Version 4, Src: 10.1.1.1, Dst:
User Datagram Protocol, Src Port: 49153, Dst Por
Source Port: 49153
Destination Port: 9
Length: 1032
Checksum: 0x0000 [zero-value ignored]
[Stream index: 0]
[Timestamps]
UDP payload (1024 bytes)
Discard Protocol
Data [truncated]: 0000000000000000000000000000
[Reported Length: 1024]
0010 01 01 0a 01 01 02 c0 01 00 09 04 08 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Source Port (udp.srcport), 2 bytes
Packets: 2 · Displayed: 2 (100.0%)
Profile: Default