

## 第7章 构建网络拓扑

### 7.1 构建总线型网络拓扑

### 7.2 模型、属性和现实

MTU：最大传输单元

巨型帧：jumbo frames

### 7.3 构建无线网络拓扑

### 7.4 ns-3中的队列

7.4.1 ns-3可用的队列模型

7.4.2 修改队列的默认设置

# 第7章 构建网络拓扑

**网络拓扑 (Network Topology)** 是指计算机网络中各个节点 (如计算机、路由器、交换机等) 通过传输介质 (如双绞线、光纤、无线信号) 相互连接的物理布局或逻辑排列方式。它描述了网络的结构形状，包括节点之间的连接关系和数据传输路径。

网络拓扑分为**物理拓扑** (实际的布线和设备位置) 和**逻辑拓扑** (数据流动的虚拟路径) 两种。

网络拓扑的设计直接影响网络的性能、可靠性、可扩展性、成本和故障排查难度。选择合适的拓扑结构需要考虑网络规模、预算、可靠性需求等因素。

**常见的网络拓扑：**



1. Bus Topology (总线拓扑)：在总线拓扑中，所有设备连接到单一通信线路（总线）。数据沿总线传输，每个设备接收到预定的数据。然而，这种拓扑结构可能存在单点故障。
2. Star Topology (星形拓扑)：在星形拓扑中，所有设备都连接到一个中央集线器或交换机。中央枢纽充当中继器，使设备通过它相互通信。它提供了更好的容错能力，因为单个连接的故障通常不会扰乱整个网络。
3. Ring Topology (环形拓扑)：在环形拓扑中，每个器件恰好连接到另外两个设备，形成一个圆形环路。数据沿环向单一方向移动，直到到达预定目的地。数据传输依赖于途中经过每个设备，因此如果其中一个设备故障，传输可能会受到干扰。
4. Mesh Topology (网状拓扑)：在网状拓扑中，每个设备都连接到网络中的其他所有设备。这为数据传输创建

- 了多条路径，提供冗余和容错能力。网格拓扑非常可靠，但实现起来可能既昂贵又复杂。
5. Tree (Hierarchical) Topology (树 (层级) 拓扑)：树形拓扑是星形拓扑和总线拓扑的结合。设备以层级结构排列，中心枢纽（如恒星）通过总线拓扑相互连接。它常用于大型网络，具有可扩展性和容错能力。
  6. Hybrid Topology (混合拓扑)：混合拓扑是上述两种或多种拓扑的组合。这使得组织能够根据自身的具体需求和要求定制网络设计。

## 7.1 构建总线型网络拓扑

使用支持CSMA (Carrier Sense Multiple Access) 的网络设备和信道构建一个总线网络拓扑。现实世界中的以太网 (Ethernet) 则采用CSMA/CD (Carrier Sense Multiple Access with Collision Detection)。

ns-3提供的基于CSMA的网络设备和信道模型，是真实的以太网中CSMA/CD的子集。

**补充知识：**以太网不同速率下的介质访问控制技术对比

以太网类型	速率	支持模式	介质访问控制技术 (MAC)
传统以太网 (10 Mbps, 如10BASE-T)	10 Mbps	半双工 (主流早期)、全双工 (后期支持)	半双工：使用 <b>CSMA/CD</b> (载波监听多路接入/冲突检测) 全双工：无冲突，直接发送 (不使用CSMA/CD)
百兆以太网 (Fast Ethernet, 100 Mbps, 如100BASE-TX)	100 Mbps	半双工、全双工	半双工：使用 <b>CSMA/CD</b> 全双工：无冲突，直接发送 (不使用CSMA/CD)
千兆以太网 (Gigabit Ethernet, 1 Gbps, 如1000BASE-T)	1 Gbps	半双工 (标准支持, 但实际极少使用)、全双工 (主流)	半双工：使用 <b>CSMA/CD</b> (需载波扩展或帧突发以维持最小槽时) 全双工：无冲突，直接发送 (不使用CSMA/CD)
万兆以太网 (10 Gigabit Ethernet, 10 Gbps, 如10GBASE-T)	10 Gbps	仅全双工 (无半双工支持)	仅点到点全双工链路，无共享介质、无冲突，直接发送 <b>(不使用CSMA/CD)</b>

**说明：**

- 现代以太网网络几乎全部使用**交换机 (Switch)** +**全双工** 模式，每个端口独立冲突域，无冲突发生，因此**CSMA/CD**已被淘汰。
- CSMA/CD**主要用于早期共享介质 (集线器Hub或总线拓扑) 的半双工场景，以处理潜在冲突。
- 万兆及以上以太网标准彻底放弃半双工和**CSMA/CD**，仅支持全双工点到点连接。

示例代码：`cp examples\tutorial\second.cc scratch\mysecond.cc`

```

1 #include "ns3/core-module.h"
2 #include "ns3/applications-module.h"
3 #include "ns3/network-module.h"
4 #include "ns3/internet-module.h"
5 #include "ns3/point-to-point-module.h"
6 #include "ns3/csma-module.h"
7 #include "ns3/ipv4-global-routing-helper.h"
8
9 // Default Network Topology
10 //
11 //      10.1.1.0
12 // n0 ----- n1   n2   n3   n4
13 //      point-to-point |   |   |   |
14 //                           =====
15 //                               LAN 10.1.2.0

```

```

16
17 using namespace ns3;
18
19 NS_LOG_COMPONENT_DEFINE("SecondscriptExample");
20
21 int
22 main(int argc, char *argv[])
23 {
24
25
26     bool verbose = true;
27     uint32_t nCsma = 3;
28
29     CommandLine cmd(__FILE__);
30     cmd.AddValue("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
31     cmd.AddValue("verbose", "Tell echo application to log if true", verbose);
32     cmd.Parse(argc, argv);
33
34     if(verbose){
35         LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
36         LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
37     }
38
39     nCsma = nCsma == 0 ? 1:nCsma;
40
41     NodeContainer p2pNodes;
42     p2pNodes.Create(2);
43
44     NodeContainer csmaNodes;
45     csmaNodes.Add(p2pNodes.Get(1));
46     csmaNodes.Create(nCsma);
47
48     PointToPointHelper pointToPoint;
49     pointToPoint.SetDeviceAttribute("DataRate",StringValue("5Mbps"));
50     pointToPoint.SetChannelAttribute("Delay",StringValue("2ms"));
51
52     NetDeviceContainer p2pDevices;
53     p2pDevices = pointToPoint.Install(p2pNodes);
54
55     CsmaHelper csma;
56     csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
57     csma.SetChannelAttribute("Delay", TimeValue(NanoSeconds(6560)));
58
59     NetDeviceContainer csmaDevices;
60     csmaDevices = csma.Install(csmaNodes);
61
62     InternetStackHelper stack;
63     stack.Install(p2pNodes.Get(0));
64     stack.Install(csmaNodes);
65
66     Ipv4AddressHelper address;
67     address.SetBase("192.168.100.0","255.255.255.0");
68

```

```

69     Ipv4InterfaceContainer p2pInterfaces;
70     p2pInterfaces = address.Assign(p2pDevices);
71
72     address.SetBase("192.168.200.0", "255.255.255.0");
73     Ipv4InterfaceContainer csmaInterfaces;
74     csmaInterfaces = address.Assign(csmaDevices);
75
76     UdpEchoServerHelper echoServer(9);
77
78     ApplicationContainer serverApps;
79     serverApps = echoServer.Install(csmaNodes.Get(nCsma));
80     serverApps.Start(Seconds(1));
81     serverApps.Stop(Seconds(10));
82
83     UdpEchoClientHelper echoClient(csmaInterfaces.GetAddress(nCsma), 9);
84     echoClient.SetAttribute("MaxPackets", UintegerValue(1));
85     echoClient.SetAttribute("Interval", TimeValue(Seconds(1)));
86     echoClient.SetAttribute("PacketSize", UintegerValue(1024));
87
88     ApplicationContainer clientApps;
89     clientApps = echoClient.Install(p2pNodes.Get(0));
90     clientApps.Start(Seconds(2));
91     clientApps.Stop(Seconds(10));
92
93     Ipv4GlobalRoutingHelper::PopulateRoutingTables();
94
95     // pointToPoint.EnablePcapAll("mysecond"); // Enable pcap on all PointToPoint
Nodes
96     // csma.EnablePcap("mysecond", csmaDevices.Get(1), true);
97
98     // pointToPoint.EnablePcap("mysecond", p2pNodes.Get(0)->GetId(), 0); // Enable
pcap on specific PointtoToPoint Nodes
99     // csma.EnablePcap("mysecond", csmaNodes.Get(nCsma)->GetId(), 0, false);
100    // csma.EnablePcap("mysecond", csmaNodes.Get(nCsma - 1)->GetId(), 0, false);
101
102    pointToPoint.EnablePcap("mysecond", p2pNodes.Get(0)->GetId(), 0); // Enable pcap
on specific PointtoToPoint Nodes
103    csma.EnablePcap("mysecond", csmaDevices.Get(nCsma), false); // 关闭混杂模式
104    csma.EnablePcap("mysecond", csmaDevices.Get(nCsma - 1), false);
105
106    Simulator::Run();
107    Simulator::Destroy();
108
109    return 0;
110 }

```

**注意：**PointToPoint分别在信道和网络设备上设置 延迟Delay 与 速率DataRate,而CSMA则在信道上设置延时和速率。这是因为：真正的CSMA网络不允许在给定信道上混合10Base-T和100Base-T设备，也即在一种信道上只能存在一种速率的网络设备，否则网络设备是无法工作的。

- **PointToPoint:** 模拟独立的发送器逻辑。在 PointToPoint 模型中， DataRate 被视为**网络设备** (NetDevice) 的属性，而 Delay 被视为**信道 (Channel)** 的属性。

- **DataRate 在设备上:** `PointToPointNetDevice` 建模了一个发送器部件 (Transmitter section) , 它负责将比特流放入“导线”。因此, 速率是发送端的硬件特性。
- **支持非对称链路:** 这种设计允许模拟**非对称链路** (例如 ADSL) 。由于速率设置在设备上, 连接到同一信道的两个设备可以具有不同的 `DataRate` 。
- **Delay 在信道上:** 信道负责模拟比特在物理介质中传播的时间, 因此传播延迟 (Propagation Delay) 是信道的固有属性。
- **CSMA: 模拟共享总线的物理一致性。** 在 `CSMA` 模型中, `DataRate` 和 `Delay` 均被视为**信道 (Channel)** 的属性。
- **共享介质的约束:** `CSMA` 模拟的是类似以太网的**共享总线网络**。在真实的物理总线网络中, **不允许在一个网段上混合使用不同速率的设备** (例如不能在同一根电缆上混合 10Base-T 和 100Base-T 设备) 。
  - **统一速率控制:** 为了反映这种物理现实, `CsmaChannel` 提供的 `DataRate` 属性被用于统一设置连接到该信道的所有设备的发送速率。
- **不可独立设置:** 在 `CSMA` 模型中, 用户**无法独立设置单个设备的速率**, 因为信道状态对所有设备是瞬时可见的, 必须保持一致的物理标准。

特性	<code>PointToPoint (P2P)</code>	<code>CSMA</code>
<b>DataRate 位置</b>	<code>NetDevice</code> (发送器属性)	<code>Channel</code> (信道固有属性)
<b>Delay 位置</b>	<code>Channel</code> (传播媒介属性)	<code>Channel</code> (传播媒介属性)
<b>设计逻辑</b>	模拟两个独立发送器之间的专用线	模拟多个设备共享的单一物理网段
<b>典型应用</b>	支持非对称速率 (如 ADSL)	强制所有接入设备速率同步

(扩展: 现实当中使用的网络设备, 通常就是网卡, 都是千兆、百兆自适应的, 也就是说, 网卡能够根据连接到的信道来自适应的选择传输速率。)

This is because a real CSMA network does not allow one to mix, for example, 10Base-T and 100Base-T devices on a given channel. We first set the data rate to 100 megabits per second, and then set the speed-of-light delay of the channel to 6560 nano-seconds (arbitrarily chosen as 1 nanosecond per foot over a 2000 meter segment). Notice that you can set an `Attribute` using its native data type.

关于: `Ipv4GlobalRoutingHelper::PopulateRoutingTables()`; 为每个节点生成链接通告, 并直接与全局路由管理器通信, 全局路由管理器使用这些全局信息为每个节点构建路由表。

命令行测试:

```
./ns3 run mysecond
./ns3 run "mysecond --PrintHelp"
./ns3 run "mysecond --nCsma=30"
```

复习命令行设置对象的属性(删除mysecond中的两行 `csma.setChannelAttribute` 代码后), 执行如下命令:

```
./ns3 run mysecond #使用csma的默认属性
./ns3 run "mysecond --PrintHelp"
./ns3 run "mysecond --PrintGroups"
./ns3 run "mysecond --PrintGroup=Csma"
./ns3 run "mysecond --PrintAttributes=ns3::CsmaChannel" #查看属性列表和默认值
```

```
./ns3 run "mysecond --ns3::CsmaChannel::DataRate=1Mbps" #对比输出结果中的接收到数据包的时间  
./ns3 run "mysecond --ns3::CsmaChannel::DataRate=100Mbps"
```

## 7.2 模型、属性和现实

要搞清楚模型中具体模拟的内容是非常重要的（即搞清楚哪些东西包含在模型里，哪些是模型没有实现的，毕竟模拟不能与现实世界等同）。

一个仿真模型是对现实世界的抽象。仿真脚本的作者决定了仿真的准确性范围 (`range of accuracy`) 和适用范围 (`domain of applicability`)。例如，在csma模型中，通过查看 `csma.h` 可以很容易发现冲突检测 (collision detection) 机制没有被建模。

通过之前的例子可以发现，ns-3通过提供 `Attributes` 使用户能够很轻松的改变模型的行为。轻松实现一些在现实世界的产品中很难实现或不可能实现的特性。

下面看两个例子：

### MTU：最大传输单元

`CsmaNetDevice` 的 `Mtu` 和 `EncapsulationMode` 属性。

MTU (Max Transmission Unit) 在 `CsmaNetDevice` 的默认值为1500字节，这是设备可以发送的最大协议数据单元 (PDU) 的大小。

- `CsmaDevice` 默认的 MTU 是1500字节，这个 MTU 值是在 RFC894 (“A Standard for the Transmission of IP Datagrams over Ethernet Networks”) 中定义的。
- 1500这个数字实际上来源于10Base5 (full-spec Ethernet) 网络的最大数据包大小——1518字节。如果减去以太网数据包的DIX封装开销 (18字节)，最终可能得到的最大数据大小 (MTU) 为1500字节。
- 我们还可以发现 IEEE 802.3 网络的 MTU 是1492字节。这是因为 IEEE 802.3 规范中 LLC/SNAP 封装给数据包增加了额外的8个字节的开销。
- **不管是 IEEE 802.3 还是以太网，底层硬件最大能够发送出去的数据链路层的最终帧的最大值都是1518字节。**因此，如果网络层按照最大值1518发送数据的话，IEEE 802.3的数据帧中所携带的有效载荷比以太网所携带的有效载荷要少8个字节。

Ethernet || 格式：

目的 MAC 地址 (6byte)	源 MAC 地址 (6byte)	类型 (2byte)	载荷数据 (46~1500byte)	CRC (4byte)
----------------------	---------------------	---------------	-----------------------	----------------

IEEE802.3 格式：

目的 MAC 地址 (6byte)	源 MAC 地址 (6byte)	长度 (2byte)	LLC (3byte)	SNAP (5byte)	载荷数据 (38~1492byte)	CRC (4byte)
----------------------	---------------------	---------------	----------------	-----------------	-----------------------	----------------



ns3支持上述两种帧的建模仿真：`EncapsulationMode` 是 `CsmaNetDevice` 提供的用于设置封装模式的Attribute，可以设置的值包括：

- `Dix` (表示使用Ethernet帧格式进行封装)

- L1c (使用IEEE 802.3 LLC/SNAP进行封装)

```

1 $ ./ns3 run "scratch/mysecond --PrintAttributes=ns3::CsmaNetDevice"
2 Attributes for TypeId ns3::CsmaNetDevice
3   --ns3::CsmaNetDevice::Address=[ff:ff:ff:ff:ff:ff]
4     The MAC address of this device.
5   --ns3::CsmaNetDevice::EncapsulationMode=[Dix]
6     The link-layer encapsulation type to use.
7   --ns3::CsmaNetDevice::Mtu=[1500]
8     The MAC-level Maximum Transmission Unit
9   --ns3::CsmaNetDevice::ReceiveEnable=[true]
10    Enable or disable the receiver section of the device.
11   --ns3::CsmaNetDevice::ReceiveErrorModel=[0]
12    The receiver error model used to simulate packet loss
13   --ns3::CsmaNetDevice::SendEnable=[true]
14    Enable or disable the transmitter section of the device.
15   --ns3::CsmaNetDevice::TxQueue=[0]
16    A queue to use as the transmit queue in the device.

```

**注意：**如果将MTU属性设置为1500， EncapsulationMode设置为 L1c，发送到物理层的帧大小为1526字节，这在很多网络中是非法的帧长度（最大可传输的帧长度为1518字节），按照这种参数仿真的结果是不能真实的映射真实的网络环境。

## 巨型帧： jumbo frames

可以通过参数配置出在现实世界中根本不存在的网络，这样的网络是否有意义完全取决于你想怎样来建模。

来考虑巨型帧 (jumbo frames:  $1500 < \text{MTU} \leq 9000 \text{ bytes}$ ) 和超大型帧(super-jumbo:  $\text{MTU} > 9000\text{bytes}$ )，这些尺寸的帧并非IEEE官方认可，但是在一些高速网络 (Gigabit) 和NICs被采用。

巨型帧 (Jumbo Frame) 和超大帧 (Super Jumbo Frame) 在现实世界中确实有采用，但应用相对有限且存在兼容性挑战。

巨型帧在特定高性能场景中有实际应用，但超大帧主要停留在研究和专用领域。随着网络技术的发展，**RDMA、DCB**等技术往往比单纯增大MTU更能有效提升网络性能。

我们可以设置：

- EncapsulationMode 为 Dix
- MTU 为64000字节
- CsmaChannel DataRate设置为 10 megabits per second。

上述的配置在现实世界中是不存在的，但是通过ns3可以很轻松的进行建模。

NS-3 通常更注重灵活性，许多模型都允许自由设定属性，而不会试图强制实施任何特定的一致性要求或特定的底层规范。

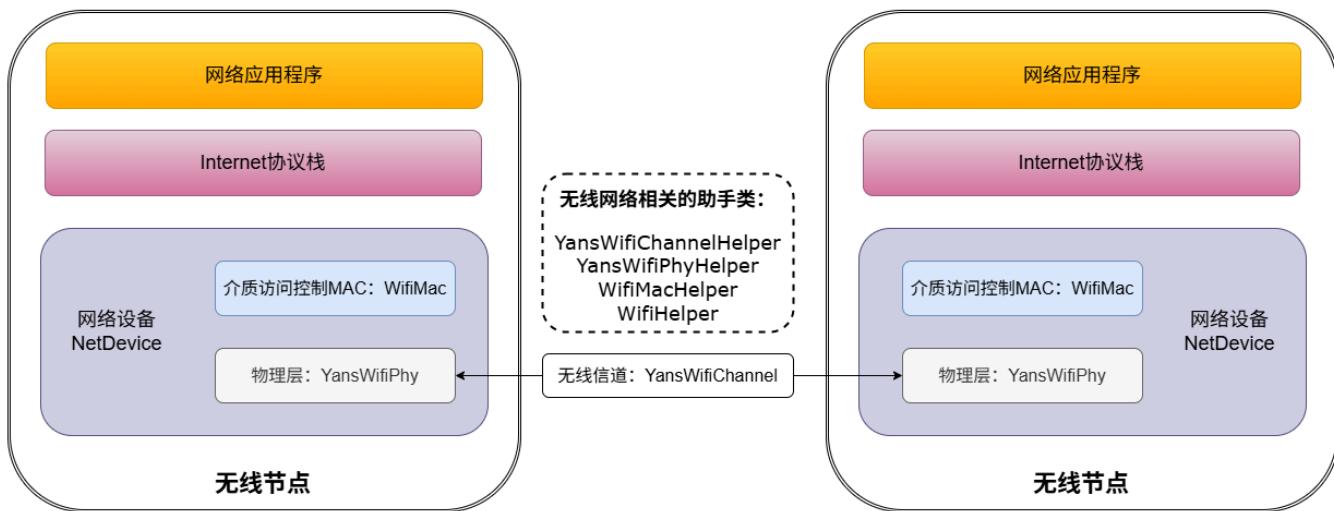
## 7.3 构建无线网络拓扑

ns-3提供了一组802.11网络模型，这些模型试图提供：符合802.11规范的精确的**mac层实现**和符合802.11a规范的“not-so-slow”的**PHY-level模型**。

**“not-so-slow” PHY-level module**: 在ns-3中，“not-so-slow” PHY模型 (YansWifiPhy) 是一种**简化的物理层抽象模型**，旨在在**仿真精度和计算效率之间取得平衡**。它是对Wi-Fi和其他无线网络物理层行为的近似模拟，特别适用于**大规模网络仿真**。较新的ns3中添加了SpectrumWifiPhy，它在干扰和频率相关效果的建模上比YansWifiPhy更精确，但可能在复杂场景中计算代价更高。

YANS: Yet Another Network Simulator

### ns3无线网络节点架构



与PointToPoint和CSMA拓扑类似，ns3也提供了Wifi的助手类，使用方法与之前的类似。

代码文件: `examples\tutorial\third.cc` (注意: ns-3.45源码中的代码与tutorial教程中的代码有差异，tutorial教程中的代码没有及时的更新，以对应版本中的源码为准)。下面代码中的 ASCII 拓扑图，将新的无线网络添加到最左侧（通过命令行参数可以改变有线和无线网络的节点数量）。

```

1  /*
2   * SPDX-License-Identifier: GPL-2.0-only
3   */
4
5 #include "ns3/applications-module.h"
6 #include "ns3/core-module.h"
7 #include "ns3/csma-module.h"
8 #include "ns3/internet-module.h"
9 #include "ns3/mobility-module.h"    // 移动性模块
10 #include "ns3/network-module.h"
11 #include "ns3/point-to-point-module.h"
12 #include "ns3/ssid.h"           // WiFi相关头文件
13 #include "ns3/yans-wifi-helper.h" // WiFi相关头文件
14 // #include "ns/wifi-module.h"    // 该头文件已包含上面两个头文件
15
16 // Default Network Topology
17 //
18 // wifi 10.1.3.0
19 //          AP
20 // *   *   *   *
21 // |   |   |   |   10.1.1.0
22 // n5   n6   n7   n0 ----- n1   n2   n3   n4
23 //                               point-to-point |   |   |   |
24 //                                         =====

```

```
25 // LAN 10.1.2.0
26
27 using namespace ns3;
28
29 NS_LOG_COMPONENT_DEFINE("ThirdscriptExample");
30
31 int
32 main(int argc, char* argv[])
33 {
34     bool verbose = true;
35     uint32_t nCsma = 3;
36     uint32_t nWifi = 3;
37     bool tracing = false;
38
39     CommandLine cmd(__FILE__);
40     cmd.AddValue("nCsma", "Number of \\"extra\\" CSMA nodes/devices", nCsma);
41     cmd.AddValue("nWifi", "Number of wifi STA devices", nWifi);
42     cmd.AddValue("verbose", "Tell echo applications to log if true", verbose);
43     cmd.AddValue("tracing", "Enable pcap tracing", tracing);
44
45     cmd.Parse(argc, argv);
46
47     // The underlying restriction of 18 is due to the grid position
48     // allocator's configuration; the grid layout will exceed the
49     // bounding box if more than 18 nodes are provided.
50     if (nWifi > 18)
51     {
52         std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the
bounding box"
53             << std::endl;
54         return 1;
55     }
56
57     if (verbose)
58     {
59         LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
60         LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
61     }
62
63     // P2P网络设置部分
64     NodeContainer p2pNodes;
65     p2pNodes.Create(2);
66
67     PointToPointHelper pointToPoint;
68     pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
69     pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
70
71     NetDeviceContainer p2pDevices;
72     p2pDevices = pointToPoint.Install(p2pNodes);
73
74     //CSMA 网络设置部分
75     NodeContainer csmaNodes;
76     csmaNodes.Add(p2pNodes.Get(1));
```

```

77     csmaNodes.Create(nCsma);
78
79     CsmaHelper csma;
80     csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
81     csma.SetChannelAttribute("Delay", TimeValue(NanoSeconds(6560)));
82
83     NetDeviceContainer csmaDevices;
84     csmaDevices = csma.Install(csmaNodes);
85
86     // WiFi 网络设置部分
87     NodeContainer wifiStaNodes;
88     wifiStaNodes.Create(nwifi);
89     NodeContainer wifiApNode = p2pNodes.Get(0);
90
91     // 创建信道对象并将其关联到PHY层对象管理器中，确保所有由YanswifiPhyHelper创建的PHY层对象共享相同的底层信道（它们共享相同的通道、无线介质，能够通信和干扰）
92     // 为了简化相关操作，使用默认的PHY层配置和信道模型。
93     YanswifichannelHelper channel = YanswifichannelHelper::Default();
94     YanswiphphyHelper phy;
95     phy.SetChannel(channel.Create());
96     /*
97      YanswiphphyHelper phy; 和 YanswiphphyHelper phy = YanswiphphyHelper::Default();
98      这是由于 ns-3 不同版本的 API 变化。在 ns-3.32 及更早版本的 `examples/tutorial/third.cc` 中，`YanswiphphyHelper` 的初始化使用 `YanswiphphyHelper phy = YanswiphphyHelper::Default();` 形式，因为当时该类的默认构造函数不直接设置所有默认属性，需要通过静态 `Default()` 方法来获取预配置的对象。
99      从 ns-3.33 开始，代码改为 `YanswiphphyHelper phy;`（使用默认构造函数），因为 ns-3 的 WiFi 模块对 Helper 类的实现进行了优化，默认构造函数现在会自动应用与 `Default()` 等效的初始配置（如默认的 PHY 参数、错误模型等），删除了 `Default()`。
100 */
101
102     // 设置MAC层
103     wifiMacHelper mac;
104     Ssid ssid = Ssid("ns-3-ssid"); // 创建一个802.11服务集标识符（SSID）对象，用于设置 MAC 层实现中“SSID”属性的值，对应于无线路由器中设置的无线网络名称。
105
106     // WiFiHelper默认将所用标准配置为 802.11ax（商业上称为 WiFi 6），并配置兼容的速率控制算法(IdealWifiManager)。
107     wifiHelper wifi;
108
109     // 配置无线工作站节点，在节点上安装WiFi网络设备
110     // 在wifi节点上安装WiFi的相关模型，要用到4个Helper objects (YanswifichannelHelper, YanswiphphyHelper, wifiMacHelper, wifiHelper) 和 Ssid object。
111     NetDeviceContainer staDevices;
112     mac.SetType("ns3::stawifiMac", "Ssid", SsidValue(ssid), "ActiveProbing", BooleanValue(false));
113     // 在上面的代码中，助手创建WiFi工作站的MAC层(ns3::stawifiMac)。当标准至少为802.11n或更新时，wifiMacHelper对象的“QosSupported”属性默认设置为true。这两种配置的组合意味着接下来要创建的MAC实例将是基础架构BSS（即带有AP的BSS）中的qos感知的非AP站（STA）。
114     // 将“ActiveProbing”属性设置为false。表示该助手创建的mac将不会发送探测请求，而站点将侦听AP信标。
115     // 设置好station-specific的MAC和PHY层参数后，就还可以使用Install方法在这些节点上创建WiFi网络设备了

```

```

116     staDevices = wifi.Install(phy, mac, wifiSTANodes);
117
118     // 配置无线AP节点，以及节点上的wi-Fi网络设备
119     NetDeviceContainer apDevices;
120     // 无线AP和工作站的mac层的TypeId有所不同，需要设置为“ns3::ApwifiMac”
121     mac.SetType("ns3::ApwifiMac", "Ssid", SsidValue(ssid));
122     apDevices = wifi.Install(phy, mac, wifiAPNode);
123
124     // 使用MobilityHelper来添加移动模型，使工作站STA节点能够移动（AP静止不动）：wandering
125     // around inside a bounding box
126     MobilityHelper mobility;
127
128     // 这段代码告诉Helper使用二维网格来初始放置STA节点。参考ns3::GridPositionAllocator类
129     /*
130         - 行优先排列：节点按水平方向先排满一行，再移动到下一行
131         - 固定列数：每行始终只有3个节点（GridWidth=3）
132         - 均匀间距：X方向节点间距5.0，Y方向行间距10.0
133         - 坐标计算：第n个节点的坐标为：
134             - X = (n % 3) * 5.0
135             - Y = floor(n / 3) * 10.0
136     */
137     this->mobility.SetPositionAllocator("ns3::GridPositionAllocator",
138                                         "MinX", DoubleValue(0.0),
139                                         "MinY", DoubleValue(0.0),
140                                         "DeltaX", DoubleValue(5.0),
141                                         "DeltaY", DoubleValue(10.0),
142                                         "GridWidth", UintegerValue(3),
143                                         "LayoutType", StringValue("RowFirst"));
144
145     /*
146         Y轴
147         ^
148         50 | o(15)   o(16)   o(17)
149         40 | o(12)   o(13)   o(14)
150         30 | o(9)    o(10)   o(11)
151         20 | o(6)    o(7)    o(8)
152         10 | o(3)    o(4)    o(5)
153          0 | o(0)    o(1)    o(2)
154
155             -----> X轴
156             0       5       10
157     */
158
159     // RandomWalk2dMobilityModel：节点以随机的速度在一个边界框内以随机的方向移动
160     mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
161                               "Bounds",
162                               RectangleValue(Rectangle(-50, 50, -50, 50)));
163     mobility.Install(wifiSTANodes);
164
165     // 设置AP位置固定
166     mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
167     mobility.Install(wifiAPNode);

```

```

168     InternetStackHelper stack;
169     stack.Install(csmaNodes);
170     stack.Install(wifiApNode);
171     stack.Install(wifistaNodes);
172
173     Ipv4AddressHelper address;
174
175     address.SetBase("10.1.1.0", "255.255.255.0");
176     Ipv4InterfaceContainer p2pInterfaces;
177     p2pInterfaces = address.Assign(p2pDevices);
178
179     address.SetBase("10.1.2.0", "255.255.255.0");
180     Ipv4InterfaceContainer csmaInterfaces;
181     csmaInterfaces = address.Assign(csmaDevices);
182
183     address.SetBase("10.1.3.0", "255.255.255.0");
184     address.Assign(staDevices);
185     address.Assign(apDevices);
186
187     UdpEchoServerHelper echoServer(9);
188
189     ApplicationContainer serverApps = echoServer.Install(csmaNodes.Get(nCsma));
190     serverApps.Start(Seconds(1));
191     serverApps.Stop(Seconds(10));
192
193     UdpEchoClientHelper echoClient(csmaInterfaces.GetAddress(nCsma), 9);
194     echoClient.SetAttribute("MaxPackets", UintegerValue(1));
195     echoClient.SetAttribute("Interval", Timevalue(Seconds(1)));
196     echoClient.SetAttribute("PacketSize", UintegerValue(1024));
197
198     ApplicationContainer clientApps = echoClient.Install(wifistaNodes.Get(nwifi - 1));
199     clientApps.Start(Seconds(2));
200     clientApps.Stop(Seconds(10));
201
202     Ipv4GlobalRoutingHelper::PopulateRoutingTables();
203
204     // 下面的代码让模拟能够结束，否则模拟永远不会“自然”停止。
205     // 这是因为无线接入点会不停的产生信标，这将导致模拟器事件被无限期地安排到未来，导致模拟不会停止。
206     // 因此必须告诉模拟器何时停止，即使它可能已经安排了信标生成事件。
207     Simulator::Stop(Seconds(10));
208
209     if (tracing)
210     {
211         phy.SetPcapDataLinkType(wifiPhyHelper::DLT_IEEE802_11_RADIO);
212         // 下面的3行代码使我们能够使用最少数量的跟踪文件查看所有流量：
213         // 1. 在2个 Point2Point 节点上启动pcap跟踪
214         // 2. 在Wi-Fi网络上启动混杂（监视器）模式跟踪
215         // 3. 在CSMA网络上启动混杂跟踪。
216         pointToPoint.EnablePcapAll("third");
217         phy.EnablePcap("third", apDevices.Get(0));
218         csma.EnablePcap("third", csmaDevices.Get(0), true);
219     }
220

```

```
221     Simulator::Run();
222     Simulator::Destroy();
223     return 0;
224 }
```

## 运行模拟:

```
1 $ cp examples/tutorial/third.cc scratch/mythird.cc
2 $ ./ns3 run 'scratch/mythird --tracing=1'
3 At time +2s client sent 1024 bytes to 10.1.2.4 port 9
4 At time +2.00926s server received 1024 bytes from 10.1.3.3 port 49153
5 At time +2.00926s server sent 1024 bytes to 10.1.3.3 port 49153
6 At time +2.02449s client received 1024 bytes from 10.1.2.4 port 9
```

查看tracing文件(third-0-0.pcap third-0-1.pcap third-1-0.pcap third-1-1.pcap), 理解网络通信的过程:

```
1 $ tcpdump -nn -tt -r third-0-1.pcap
2 $ tcpdump -nn -tt -r third-0-0.pcap
3 $ tcpdump -nn -tt -r third-1-0.pcap
4 $ tcpdump -nn -tt -r third-1-1.pcap
```

## 查看移动信息:

上面的信息无法体现节点的移动性，可以通过tracing system来获取节点的位置信息。

ns-3的tracing system分为: trace sources和trace sinks

We will use the mobility model predefined course change trace source to originate the trace events. We will need to write a trace sink to connect to that source that will display some pretty information for us. Despite its reputation as being difficult, it's really quite simple. Just before the main program of the `scratch/mythird.cc` script (i.e., just after the `NS_LOG_COMPONENT_DEFINE` statement), add the following function:

在 `NS_LOG_COMPONENT_DEFINE` 代码之后加入如下代码（该函数从移动模型中提取位置信息，并打印节点的x和y坐标值）：

```
1 void
2 CourseChange(std::string context, Ptr<const MobilityModel> model)
3 {
4     Vector position = model->GetPosition();
5     NS_LOG_UNCOND(context <<
6                 " x = " << position.x << ", y = " << position.y);
7 }
```

下面使用 `Config::Connect` 函数设置无线节点（运行echo client应用程序的节点）改变位置时执行上面的代码。在 `Simulator::Run` 之前添加下面的代码：

```

1 std::ostringstream oss;
2 oss << "/NodeList/" << wifiNodes.Get(nwifi - 1)->GetId()
3     << "$ns3::MobilityModel/CourseChange";
4
5 // 使用Config::Connect函数在trace source(节点7上的CourseChange)和trace sink(自定义函数
6 CourseChange)建立连接
7 Config::Connect(oss.str(), MakeCallback(&CourseChange));
8 // Config::Connect("/NodeList/7/$ns3::MobilityModel/CourseChange",
9 MakeCallback(&CourseChange)); // 同上一行代码

```

运行模拟程序：

```

1 $ ./ns3 build
2 $ ./ns3 run scratch/mythird
3 [0/2] Re-checking globbed directories...
4 [2/2] Linking CXX executable scratch/ns3.45-mythird-debug
5 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10, y = 0
6 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.7692, y = 0.639036
7 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.7262, y = 0.349134
8 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.8935, y = 0.902746
9 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.8929, y = 0.93761
10 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.9437, y = 0.622921
11 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.2687, y = 1.56862
12 At time +2s client sent 1024 bytes to 10.1.2.4 port 9
13 At time +2.00926s server received 1024 bytes from 10.1.3.3 port 49153
14 At time +2.00926s server sent 1024 bytes to 10.1.3.3 port 49153
15 At time +2.02449s client received 1024 bytes from 10.1.2.4 port 9
16 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.2746, y = 1.67724
17 /NodeList/7/$ns3::MobilityModel/CourseChange x = 9.46407, y = 2.26291
18 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.4582, y = 2.37091
19 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.3536, y = 3.36543
20 /NodeList/7/$ns3::MobilityModel/CourseChange x = 9.99128, y = 2.43338
21 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.0597, y = 1.43573
22 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.9868, y = 1.06101
23 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.9858, y = 1.10579
24 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.425, y = 1.93372
25 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.8954, y = 1.05124
26 /NodeList/7/$ns3::MobilityModel/CourseChange x = 10.9144, y = 1.24547
27 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.5588, y = 0.480802
28 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.4819, y = 0.86535
29 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.1369, y = 1.80392
30 /NodeList/7/$ns3::MobilityModel/CourseChange x = 13.1133, y = 2.01958
31 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.9241, y = 1.03765
32 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.988, y = 1.38935
33 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.7464, y = 2.04109
34 /NodeList/7/$ns3::MobilityModel/CourseChange x = 11.7554, y = 1.90724
35 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.568, y = 2.49016
36 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.0458, y = 3.34299
37 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.0749, y = 4.34256
38 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.9787, y = 3.91475
39 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.1009, y = 3.43575
40 /NodeList/7/$ns3::MobilityModel/CourseChange x = 12.1078, y = 2.43577

```

## 7.4 ns-3中的队列

对于用户来说，了解默认安装了什么以及如何更改默认设置和观察性能是很重要的。

ns-3中队列规则的选择可能对性能有很大的影响。

在架构上，ns-3将设备层与Internet主机的IP层或流量控制层分开。最近的ns-3版本中，出站的数据包在到达信道对象之前要经过两个队列层：

- 第一层队列层（queueing layer）称为“**流量控制层（traffic control layer）**”。主动队列管理（RFC7567）和基于服务质量（QoS）的优先级通过使用队列规则以设备独立的方式进行。
- 第二层队列层通常存在于NetDevice对象。不同的设备（例如：LTE, Wi-Fi）队列的具体实现各异。

这种两层架构（软件队列提供优先级划分，硬件队列适配特定链路类型）与实际部署情况相符，但实际操作中可能更为复杂。例如，地址解析协议（Address Resolution Protocols）仅有一个小队列，而Linux系统中的Wi-Fi则存在四层队列结构（<https://lwn.net/Articles/705884/>）。（This two-layer approach mirrors what is found in practice, (software queues providing prioritization, and hardware queues specific to a link type).）

在ns-3中上层的流量控制层（traffic control layer），在接收到网络设备通知网络设备队列已满时，才会生效，此时流量控制层将停止向NetDevice发送数据包。否则，在未接收到网络设备的通知时，流量控制层的队列规则（queueing disciplines）是不起作用的，它的积压的工作（backlog）总是空的。

目前，以下NetDevices支持流量控制，即这些网络设备有向流量控制层发出通知的能力。这些NetDevice使用Queue对象（或Queue子类的对象）来存储数据包：

- Point-To-Point
- Csma
- Wi-Fi
- SimpleNetDevice

队列规则（queueing disciplines）的性能表现高度依赖于NetDevices的队列的大小。目前，ns-3中的默认队列并未针对配置的链路属性（带宽、延迟）进行自动调优（即动态调整队列的大小），通常仅采用最简单的实现形式（例如：采用FIFO调度策略和drop-tail行为）。但是，可以通过启用BQL（Byte Queue Limits）动态调整队列的大小，BQL是在Linux内核中实现的一种算法，用于调整设备队列的大小，以防止缓冲区膨胀，同时避免饥饿。目前，支持流量控制的NetDevices是支持BQL的，可以使用BQL来动态调整队列的大小。

下面的报告，通过ns-3模拟和真实实验，分析了设备的队列大小对队列规则（queueing disciplines）有效性的影响：

P. Imputato and S. Avallone. An analysis of the impact of network device buffers on packet schedulers through experiments and simulations. *Simulation Modelling Practice and Theory*, 80(Supplement C):1–18, January 2018.  
DOI: 10.1016/j.simpat.2017.09.008

### 7.4.1 ns-3可用的队列模型

- 在流量控制层，有以下可选queueing models
  - PFifoFastQueueDisc: The default maximum size is 1000 packets
  - FifoQueueDisc: The default maximum size is 1000 packets
  - RedQueueDisc: The default maximum size is 25 packets
  - CoDelQueueDisc: The default maximum size is 1500 kilobytes

- FqCoDelQueueDisc: The default maximum size is 10240 packets
- PieQueueDisc: The default maximum size is 25 packets
- MqQueueDisc: This queue disc has no limits on its capacity
- TbfQueueDisc: The default maximum size is 1000 packets

默认情况下，当给NetDevice关联的接口分配IPv4或IPv6地址时，系统会自动安装 `pfifo_fast` 队列规则 (queueing discipline) ，除非该NetDevice上已存在其他队列规则

- 在设备层，则是针对特定设备的队列：

- PointToPointNetDevice: 默认的配置(as set by the helper) 安装一个DropTail队列，队列的默认大小为 100 packets。
- CsmaNetDevice: 默认的配置(as set by the helper) 安装一个DropTail队列，队列的默认大小为100 packets。
- WiFiNetDevice: 对于non-QoS Station， 默认的配置安装一个DropTail队列，队列的默认大小为100 packets。对于 QoS Stations， 安装4个DropTail队列，队列的默认大小为100 Packets。
- SimpleNetDevice: 默认的配置安装一个DropTail队列，队列的默认大小为100 packets。
- LTENetDevice: Queueing occurs at the RLC layer (RLC UM default buffer is 10 \* 1024 bytes, RLC AM does not have a buffer limit).
- UanNetDevice: There is a default 10 packet queue at the MAC layer

## 7.4.2 修改队列的默认设置

- NetDevice使用的队列类型可以通过 `device helper` 进行修改，代码示例：

```

1 NodeContainer nodes;
2 nodes.Create (2);
3
4 PointToPointHelper p2p;
5 p2p.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue ("50p"));
6
7 NetDeviceContainer devices = p2p.Install (nodes);

```

- 被安装到NetDevice上的队列规则，可以通过 `traffic control helper` 来修改 (The type of queue discipline installed on a NetDevice can be modified through the traffic control helper) :

```

1 InternetStackHelper stack;
2 stack.Install (nodes);
3
4 TrafficControlHelper tch;
5 tch.SetRootQueueDisc("ns3::CoDelQueueDisc", "MaxSize", StringValue ("1000p"));
6 tch.Install(devices);

```

- BQL可以通过 `traffic control helper` 在支持它的设备上启用:

```
1 InternetStackHelper stack;
2 stack.Install(nodes);
3
4 TrafficControlHelper tch;
5 tch.SetRootQueueDisc("ns3::CoDelQueueDisc", "MaxSize", StringValue ("1000p"));
6 tch.SetQueueLimits("ns3::DynamicQueueLimits", "HoldTime", StringValue ("4ms"));
7 tch.Install(devices);
```