NS3网络仿真

1. 实验目的

配置NS3需要的环境。自行搭建NS3协议并创建收发端口实现NS3网络通信。

2. 实验要求

- 软件要求：Windows 10及以上版本；RflySim工具链[1]。
- 硬件要求：笔记本/台式电脑1台[2]。

3.实验地址

例程目录：
[安装目录]\RflySimAPIs\9.RflySimComm\3.CustExps\e2.NS3_UAVCommExps\1.NS3

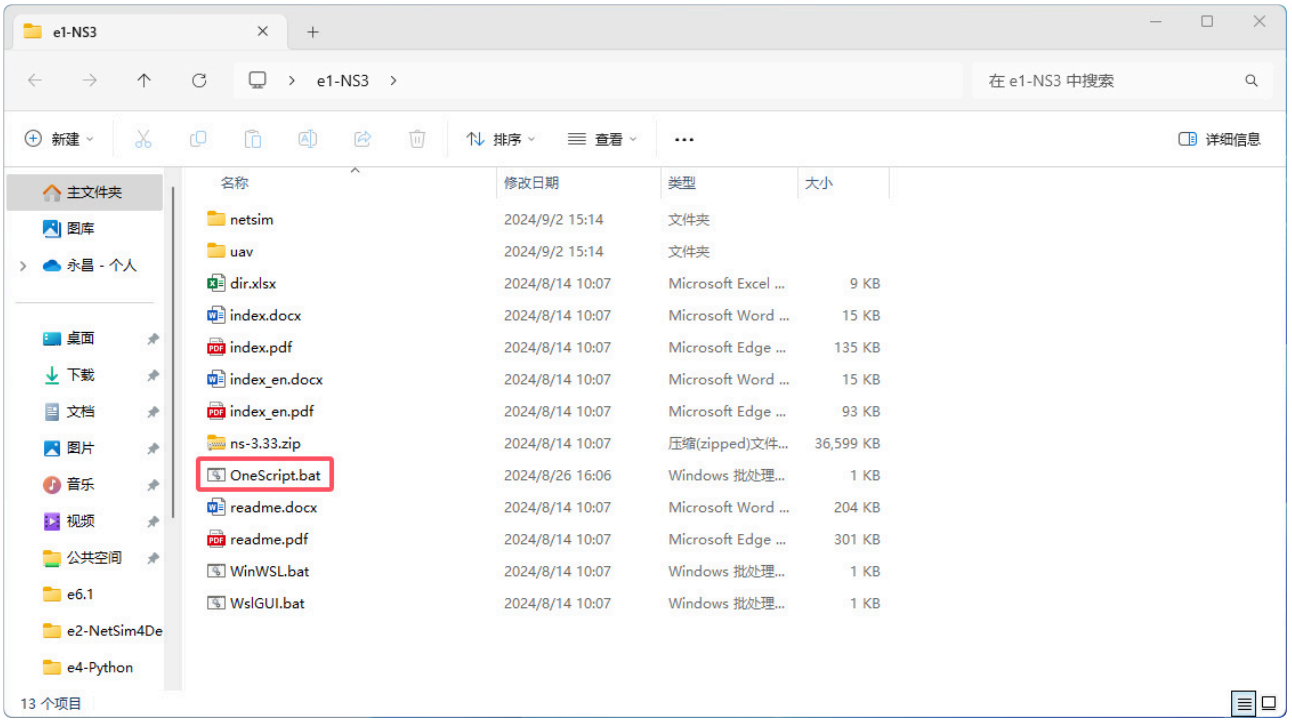
- ./netsim/RflyNet.cc：NS3组网通信仿真转发核心文件。
- ./uav/Python38Run.bat：Python38运行脚本。
- ./uav/SITLRun4MavlinkFull.bat：SITL运行脚本，用于完整MAVLink通信测试。
- ./uav/TestSend.py：测试发送脚本。
- ./uav/UAV1Ctrl.py：无人机1控制器。
- ./uav/UAV2Ctrl.py：无人机2控制器。
- ./uav/UAV3Ctrl.py：无人机3控制器。
- ./uav/UAV4Ctrl.py：无人机4控制器。
- ./uav/UavPythonRunALL.bat：运行所有无人机Python脚本的一键脚本。
- ./OneScript.bat：一键启动脚本，用于启动整个仿真环境。

4. 实验内容或步骤

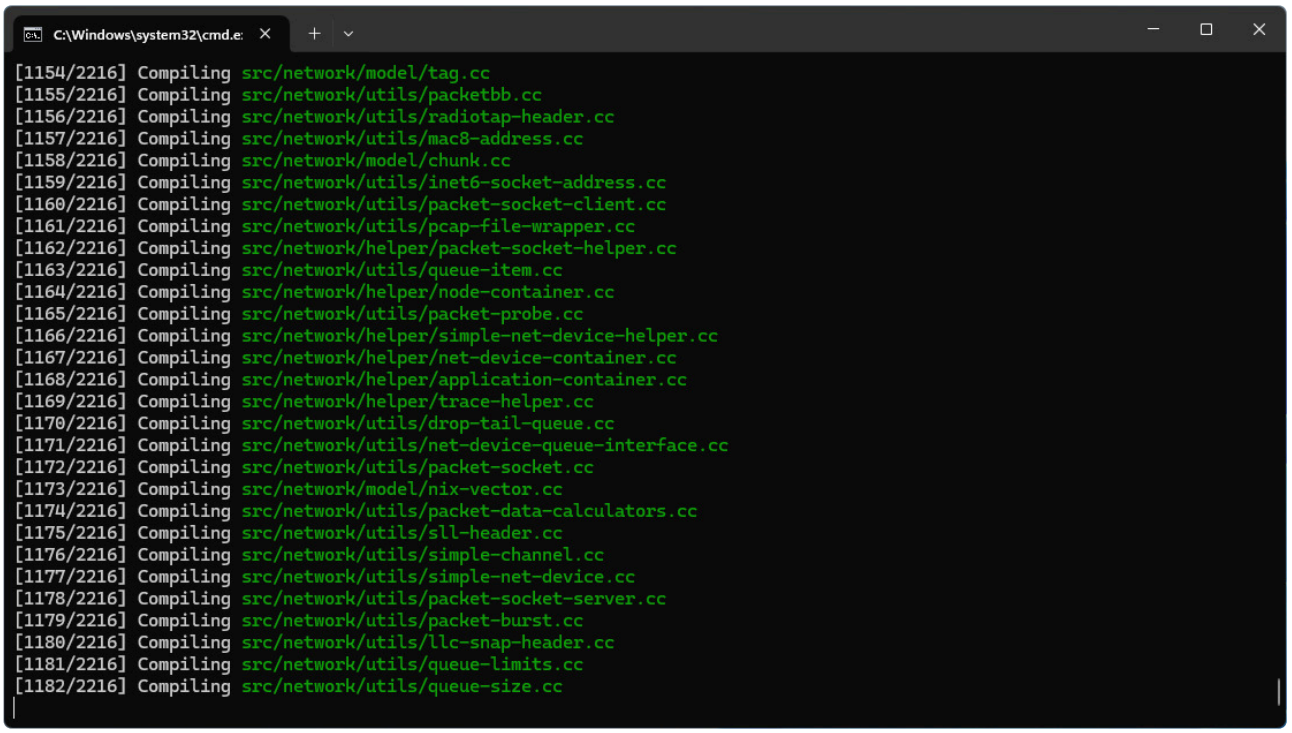
本实验有一个小实验。

4.1 步骤1：本机实验

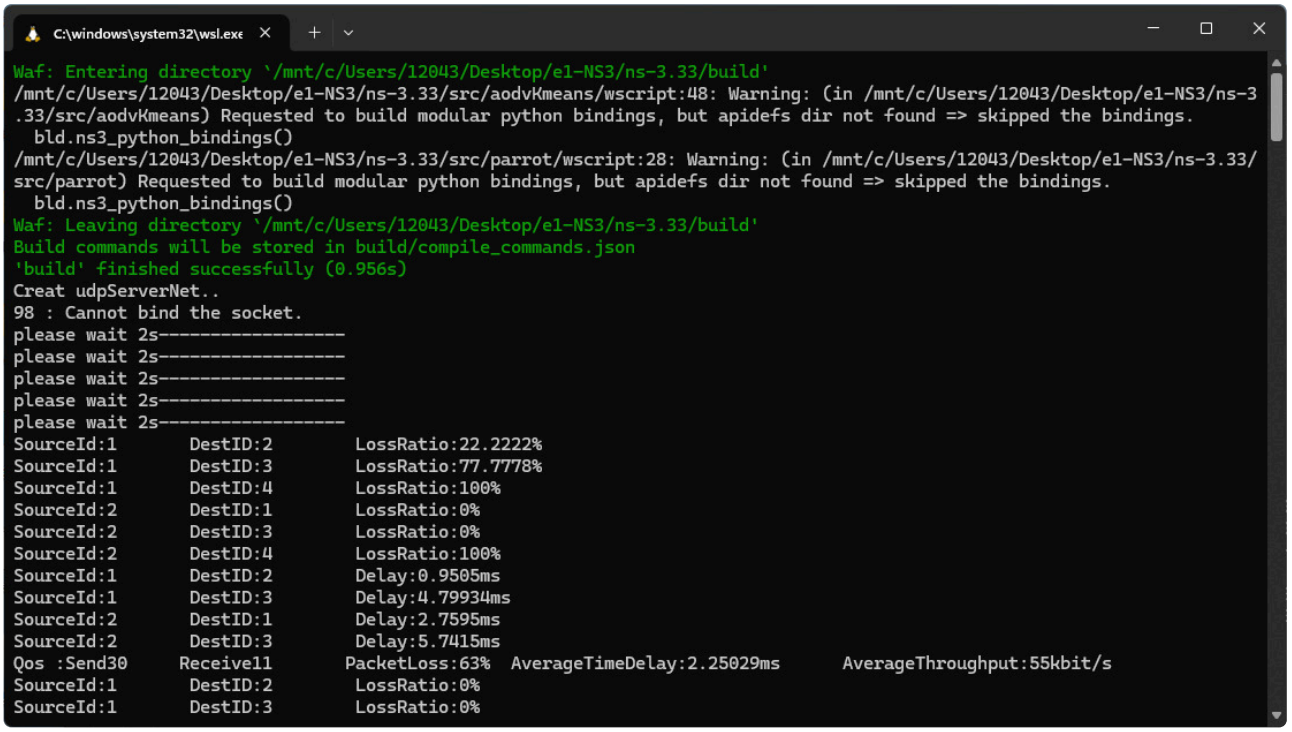
这里推荐将例程复制一份到其他目录中，以防无法复原文件。双击运行"OneScript.bat"一键启动脚本，将飞机仿真与NS网络仿真进行一键配置与启动。



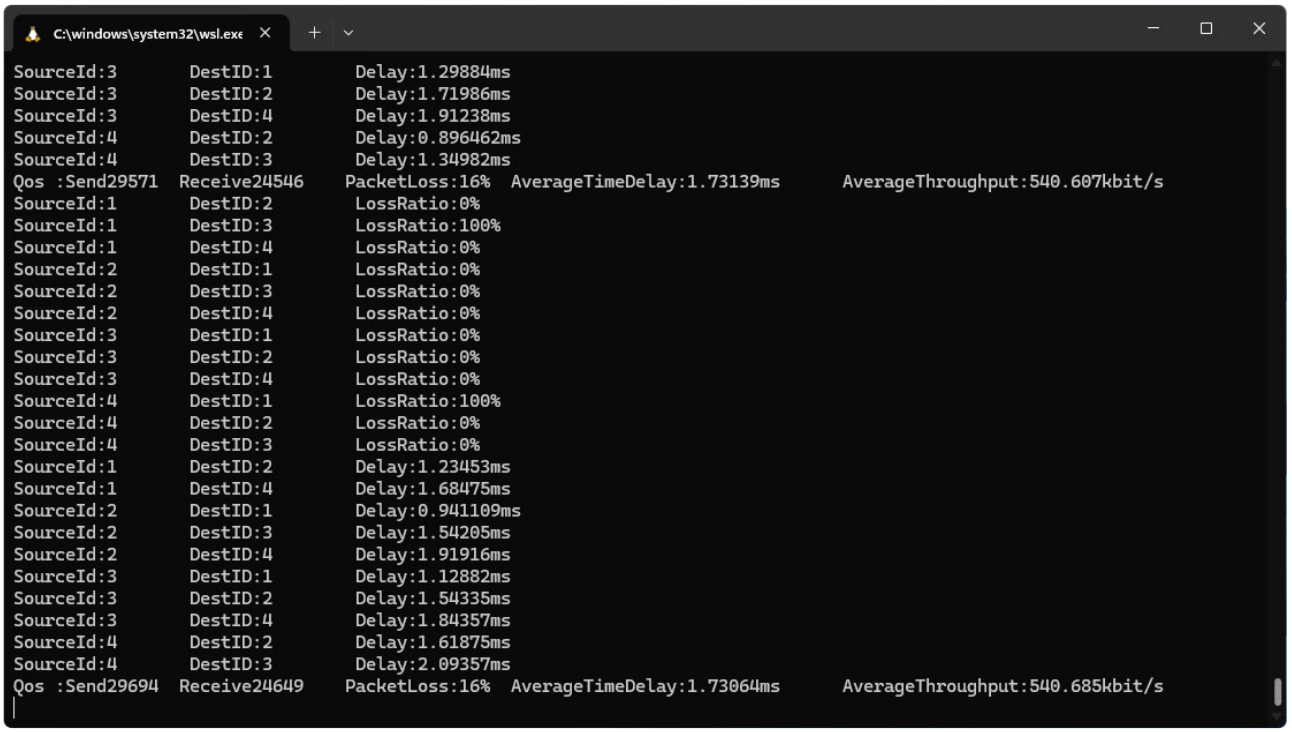
这里如果没有进行过编译，会先进行编译，需要等待一段时间。



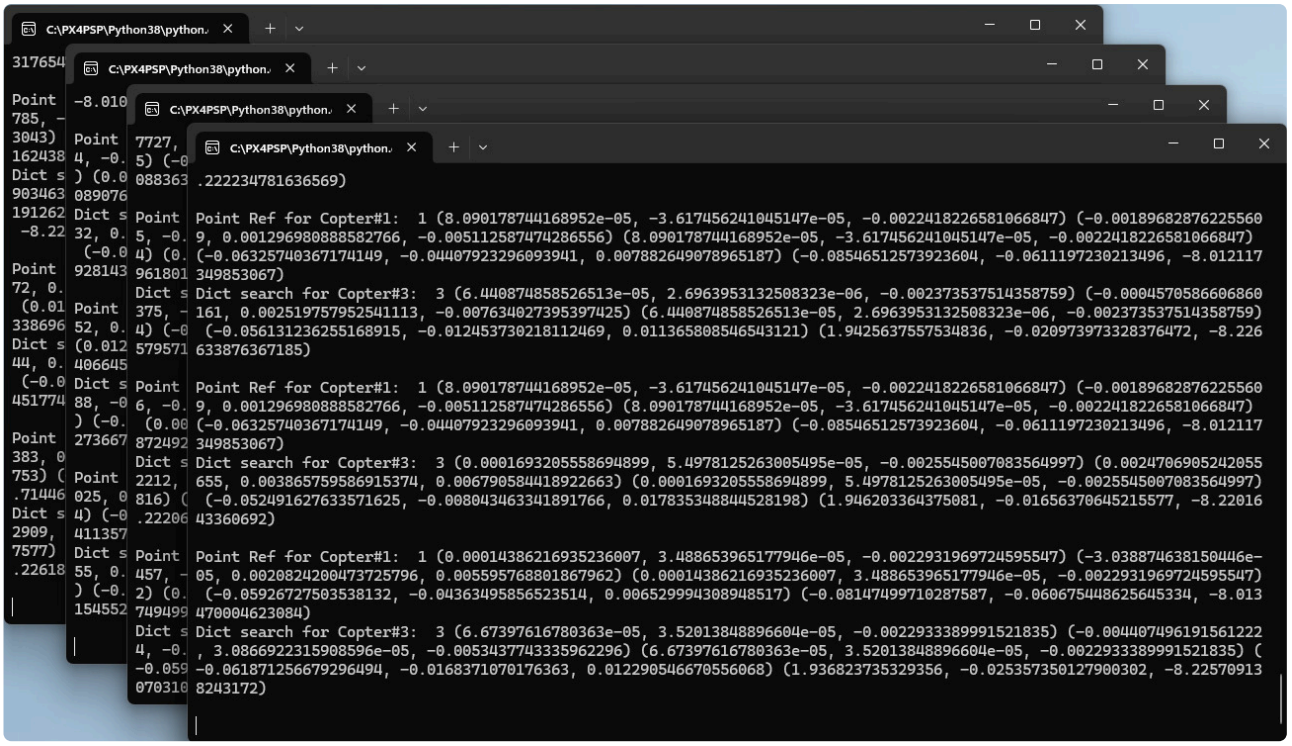
如果之前有已经进行过编译，则会直接配置和启动。



可在命令框中观察到各飞机丢包率等各种信息。



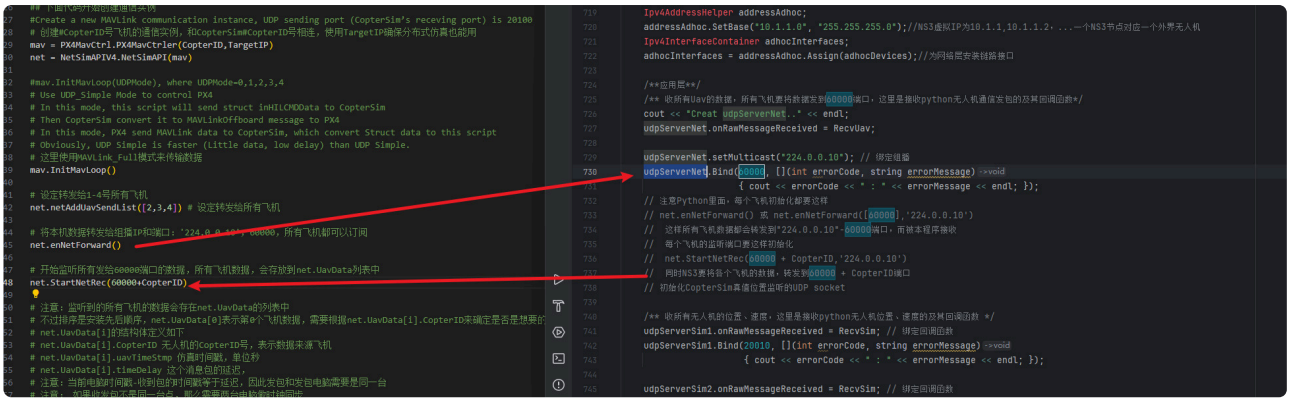
并且在控制程序中，会打印飞机的信息。



注意：此例程飞机并不会起飞。

5. 关键知识点

此实验中，4架无人机的MAVLink消息被设定通过UDP协议发送至统一的目标端口（60000端口）。这些消息被NS3仿真环境接收，并作为输入数据流进一步处理。具体而言，NS3通过一个回调函数ReceiveNs3处理来自各个无人机的数据包。当UDP端口收到来自MAVLink的消息时，回调函数会被触发。在回调函数内部，系统会对接收到的消息进行时延统计、数据分析等操作，并将处理后的数据重新转发。每架无人机的MAVLink数据经过NS3的处理后，转发到特定的端口（如60000 + CopterID），从而将相应的反馈数据返回至对应的控制器。



NS3仿真器设置说明:

该仿真模块基于NS3框架物理层和无线信道建模实现无人机网络通信的延时分析。代码通过WifiHelper模块构建802.11a无线通信标准的基础链路，采用AARF（自适应自动速率回退）算法动态管理链路传输速率，以适配信道质量变化；同时定义了节点间遵循的5GHz频段标

准，理论可支持最高54Mbps的物理层传输速率。信道特性由SpectrumChannelHelper设定，其中基于MultiModelSpectrumChannel支持多信道建模，利用RangePropagationLossModel限制无人机节点的最大通信半径为500米，超过该距离则触发信号完全衰减，仿真实际环境中的通信范围约束。传播延迟通过恒定光速模型计算，精确反映信号在空间中传输的物理时延特性。

物理层具体参数通过SpectrumWifiPhyHelper配置：载波频率（如5210MHz）、20MHz固定信道带宽、收发天线的增益参数（txGainDb/rxGainDb）共同构成信道容量基准值；收发功率阈值（txPowerBaseDbm/txPowerEndDbm）与接收灵敏度（rxSensitivityW）限定了节点动态调整发射功率的边界条件，确保符合无线设备的物理性能约束。这些参数通过绑定channelHelper生成的信道模型生效，完整定义了无人机节点在通信过程中受物理环境、硬件属性影响的实时状态变化。该模块为后续仿真分析端到端传输时延提供了符合实际通信协议规范的底层支撑。

```
/**物理层以及信道设置**/

// 1.设置链路传输速率

WifiHelper wifi;

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

wifi.SetStandard(WIFI_STANDARD_80211a);

// 2.设置信道属性， 包括信道的信道衰减、信道的传播时延

SpectrumChannelHelper channelHelper;

channelHelper.SetChannel("ns3::MultiModelSpectrumChannel");//多信道

channelHelper.AddPropagationLoss("ns3::RangePropagationLossModel", "MaxRange", DoubleValue(MaxRange)); //衰减模型（路径损耗），最大通信500m

channelHelper.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");//传播延时 路径/3*10^8(光速)

// 3.设置物理层参数: 无线物理层（PHY）的详细参数配置

SpectrumWifiPhyHelper phy;

// phy.SetErrorRateModel("ns3::YansErrorRateModel");

phy.Set("Frequency", UintegerValue(frequency));//频率
```

```
phy.Set("ChannelWidth", UintegerValue(20));//信道带宽:20MHz
带宽是802.11a/n的典型配置

phy.Set("RxSensitivity", DoubleValue(rxSensitivityW)); //接收机灵敏度

phy.Set("TxGain", DoubleValue(txGainDb)); //发送天线增益

phy.Set("RxGain", DoubleValue(rxGainDb)); //接收天线增益

phy.Set("TxPowerStart", DoubleValue(txPowerBaseDbm)); //发送功率1

phy.Set("TxPowerEnd", DoubleValue(txPowerEndDbm));//发送功率2

phy.SetChannel(channelHelper.Create());
```

链路层配置采用WifiMacHelper定义自组织网络的MAC层协议行为。通过声明AdhocWifiMac类型将节点设置为无中心化自组网模式，此时所有无人机节点无需依赖路由器或接入点即可直接建立对等通信。虽然典型的自组网无需SSID认证，但代码中仍需构造一个Ssid对象（其内容为空值）作为网络标识符，确保所有节点逻辑上归属于同一无线网络。物理层（phy）与链路层（mac）参数通过wifi.Install方法绑定至节点集合（adhocNodes），生成网络设备容器（adhocDevices），此时每个节点已完成无线通信能力的基础配置。该实现契合无人机集群动态组网的特性，使节点可根据位置变化自主建立或断开无线链路。

```
/**链路层**/

    WifiMacHelper mac;

    /*SSID参数：尽管自组网通常无需SSID验证，但此处仍需定义一个SSID对象
（为空值时所有节点默认在同一逻辑网络）*/

    Ssid ssid;

    /*AdhocWifiMac（自组网MAC模式）

    特点：节点间直接通信，无需基础设施（如路由器或AP），
适用于无人机集群等动态网络。*/

    mac.SetType("ns3::AdhocWifiMac", "Ssid", SsidValue(ssid)); //自组网wifi

    /*将 物理层（phy） 和 MAC层（mac） 配置绑定到所有节点。*/

    NetDeviceContainer adhocDevices = wifi.Install(phy, mac, adhocNodes);
```

对传输层与节点移动性进行了完整定义。通过Config::SetDefault接口设置TCP协议参数：将TCP分段尺寸调整为958字节（总长度1000字节扣除42字节头部，含IP头20字节、TCP基础头20字节及可选扩展字段2字节），适用于模拟无人机网络中小数据包高频传输的场景；同时配置滑动窗口的延迟确认计数（DelAckCount）为1，禁用TCP协议的延迟确认机制以降低传输时延，契合实时性通信需求。网络协议栈通过InternetStackHelper安装至节点，确保所有无人机具备完整的TCP/IP协议处理能力。

```
// 2.设置TCP 分组负载大小
    //  42 = headers size
    /*作用：设置单个TCP段的有效载荷为 958字节（总长度1000字节减去42字节包头）。
    包头组成：20字节IP头 + 20字节TCP头 + 2字节可选字段（NS3默认逻辑）。
    典型用途：模拟特定场景下的小包传输（如物联网设备）。
    */
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(1000 - 42));
    // 3.设置滑动窗口参数
    Config::SetDefault("ns3::TcpSocket::DelAckCount", UintegerValue(1));
    InternetStackHelper internet;
    InternetStackHelper stack;
    /**移动模型，后面会根据python数据更新每一个NS3节点的位置、速度，这里初始化
*/
    /*效果：所有节点在仿真开始时排列成 单行等距直线（间距50米）。
    示例：若 adhocNodes_N=5，节点坐标为 (0,0), (50,0), (100,0), (150,0), (200,0)。*/
    MobilityHelper mobility;
    mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                                  "MinX", DoubleValue(0.0),
                                  "MinY", DoubleValue(0.0),
                                  "DeltaX", DoubleValue(50),
                                  "DeltaY", DoubleValue(50),
```

```
                                "GridWidth", UintegerValue(adhocNodes_N),
                                "LayoutType", StringValue("RowFirst"));
    /*
    模型选择：恒定速度模型（初始速度为0，需外部更新）。
    后续操作：通过Python脚本动态更新节点位置和速度（如模拟无人机飞行轨迹）。
    */
    mobility.SetMobilityModel("ns3::ConstantVelocityMobilityModel");
    mobility.Install(adhocNodes);
```

在应用层设计中，仿真模块通过udpServerNet实现了对无人机数据的集中接收与处理机制。该服务器绑定至60000端口（Bind(60000)），通过加入组播地址224.0.0.10（setMulticast）形成逻辑通信组群，允许无人机集群以组播形式高效传输数据。当任意节点发送数据至该组播地址的目标端口时，服务器将触发预定义的RecvUav回调函数，对原始报文进行解析或转发操作，支持实时处理无人机间的协同控制信息或状态同步数据。

```
/**应用层**/
    /** 收所有Uav的数据，所有飞机要将数据发到60000端口，
这里是接收python无人机通信发包的及其回调函数*/
    cout << "Creat udpServerNet.." << endl;
    /*作用：绑定原始消息接收回调函数
    细节：当收到UDP数据时，触发RecvUav函数处理*/
    udpServerNet.onRawMessageReceived = RecvUav;
    udpServerNet.setMulticast("224.0.0.10"); // 绑定组播
    udpServerNet.Bind(60000, [](int errorCode, string errorMessage)
                { cout << errorCode << " : " << errorMessage << endl; });
```

通过循环遍历所有无人机节点，完成NS3虚拟节点的网络通信与移动模型初始化。具体流程包括：为每个节点创建UDP套接字（ns3::UdpSocketFactory），启用广播功能以支持群组消息传输，并将套接字指针存入SocketVect全局列表；通过InetSocketAddress绑定节点IP地

址与固定监听端口12，使所有节点持续监听该端口数据，地址信息同步存储至AddrList供外部调用查询。当节点收到数据包时，通过SetRecvCallback触发ReceiveNs3回调函数实现协议栈与应用层的数据交互，例如转发或统计网络流量。同时，从节点对象中提取恒定速度移动模型（ConstantVelocityMobilityModel）指针并存入MobList，为后续通过Python脚本实时更新节点坐标（如mob->SetPosition(Vector(x,y,z))）及速度向量提供接口。此过程构建了无人机节点的完整通信与运动控制框架，使网络行为可随动态位置变化实时响应。

```
// 内部NS3虚拟节点接收数据绑定和全局变量赋值
  for (int i = 0; i < adhocNodes_N; i++)
  {
      Ptr<Socket> dstSocket = Socket::CreateSocket(adhocNodes.Get(i), TypeId::LookupByName("ns3::UdpSocketFactory")); //创建UDP套接字（关键步骤分解）
      dstSocket->SetAllowBroadcast(true); // 开启广播
      SocketVect.push_back(dstSocket);    // 存储节点socket指针
      InetSocketAddress dst = InetSocketAddress(
          adhocInterfaces.GetAddress(i), // 获取节点i的IP地址
          12      // 固定监听端口12
      );//ns3节点对应的socket
      AddrList.push_back(dst);  // 存储所有节点的地址信息
      dstSocket->Bind(dst);// 绑定地址与套接字，使socket监听本节点IP:12端口
      dstSocket->SetRecvCallback(MakeCallback(&ReceiveNs3)); // 当NS3节点收到数据包，触发回调函数ReceiveNs3
      // 配置移动模型（关键步骤）
      Ptr<ConstantVelocityMobilityModel> mob = adhocNodes.Get(i)->GetObject<ConstantVelocityMobilityModel>(); // 获取CopterID对应的NS3节点移动模型，例如1号飞机对应Node.get(0)的移动模型
      MobList.push_back(mob);                                                    // 存储节点移动指针。
  }
```

6.参考资料

1. 无

7.常见问题

Q1：***

A1：***

---

1. https://rflysim.com/ ↩
2. 推荐配置请见： https://rflysim.com/doc/zh/HowToInstall.pdf ↩