RflySim 平台分布式组网通信 NetSimAPIV4.py 接口介绍

平台使用一个 Python 库 NetSimAPIV4.py 来实现不同飞机之间的数据通信（交互）与网络仿真模拟，库文件的位置在
“RflySimAPIs\RflySimSDK\comm\NetSimAPIV4.py”。

1.1 功能总体介绍

无人机通信接口 API，MavOrCopterID 可以指定 mav=PX4MavCtrlV4 或 CopterID 号
如果传入 mav，则默认启用无人机数据转发模式，会将 UAVSendData 转发到指定端口（由 enNetForward 或 enUavForward 指定）

本接口有四种用法

1.1.1 单点通信模拟。

用 enNetForward 或 enUavForward 指定数据包发到指定的飞机（端口）列表，然后 StartNetRec 指定接收自己飞机（端口）数据
例如，1 号飞机，指定将数据转发给 2 3 4 5，并指定接收发往 1 号飞机数据
……
5 号飞机，指定将数据发往 1 2 3 4，并指定接收 5 号飞机数据
通过上述配置，可以指定组建一种指定飞机 ID 的通信方式（对应单播）。

1.1.2 广播(组播)模拟

用 enNetForward 或 enUavForward 指定数据包发往默认端口 60000（或 0 号飞机），然后 StartNetRec 绑定并接收 60000（或 0 号飞机）数据
通过这种方式，相当于构建了一个广播（组播）网络，每个飞机都能收到其他飞机的数据，通过数据包内的 CopterID，可以识别出数据来自哪个飞机

1.1.3 连接网络仿真器

在“单点通信模拟”的基础上，还可以通过 enNetForward 或 enUavForward 将数据包统一发给网络仿真器的端口（例如，65000）
其次，每个飞机订阅自己的数据接口（例如，60000+CopterID）
网络仿真器，对于每个飞机本应该收到的数据，会经过网络仿真模拟，加入延迟、丢包等环节，再将数据包转发到各个飞机
例如，1 号飞机本身要将数据发给 3 号飞机，但是它们之间被高墙遮挡，网络仿真器将会把 1 号飞机发给 3 号飞机的数据包拦截

或者，3 号飞机被对方网络干扰器锁定，则网络仿真器将不会转发任何数据包给 3 号飞机

1.1.4 网络仿真器+收发定制

在“连接网络仿真器”的基础上，每个飞机可以向网络仿真器发送消息，指定数据包发给的飞机，以及指定请求接收的飞机数据。

1） 可以通过 netResetSendList 或 netAddUavSendList 来维护一个飞机列表，使得数据包只会被指定的飞机响应

2）可以通过 StartReqUavData 函数，来请求别的飞机，将自己加入飞机列表，以便能收到指定飞机的数据

通过上述列表控制功能，可以实现网络数据的筛选

本接口能够转发和接受的数据类型主要有两种：

1. 来自 mav 的底层飞控数据，包括位置点等（需要将 mav 传入 NetSimAPI 构造函数），数据包结构见 UAVSendData

2. 使用 netForwardBuf 直接将某个 buf 包转发出去，对方需要用同样的方法进行解包

注意：后一种方法，需要在外层 python 中，写一个死循环，并调用 bufEvent.wait()来接收 buf 包来的信号

然后，从 bufData 中，获取 buf 包的值，并进行数据解析。

注意，还可以从 bufHead=[checkSum,CopterID,sendMode,StartIdx,SendMask,TimeUnix]中获取数据包源头信息

1.2 class NetSimAPI 类

```
# RflySimNet 通信与仿真接口类

class NetSimAPI:

    def __init__(self, MavOrCopterID=1):  # 默认构造函数

    # mav 如果赋值为 PX4MavCtrlV4 的实例，则 self.SendState 为 True，会自动转发无人机的位置、速度等状态信息
```

mav 如果赋值为 CopterID，则不会转发飞机自身信息，而是需要用 netForwardBuf 来发送 buf 到其他飞机

使用方法示例 1（需要转发本飞机数据）：

```
# 创建#1 号飞机的通信实例，和 CopterSim#1 号相连
mav = PX4MavCtrl.PX4MavCtrler(1)
net = NetSimAPIV4.NetSimAPI(mav)

# 创建#2 号飞机的通信实例，和 CopterSim#2 号相连
mav2 = PX4MavCtrl.PX4MavCtrler(2)
net2 = NetSimAPIV4.NetSimAPI(mav2)
```

使用方法示例 2（不需要转发本飞机数据）：

```
# 为 1 号飞机创建通信实例，不需要与 CopterSim 相连，只转发 buf 包
net = NetSimAPIV4.NetSimAPI(1)

# 为 2 号飞机创建通信实例，不需要与 CopterSim 相连，只转发 buf 包
net2 = NetSimAPIV4.NetSimAPI(2)
```

```
def enNetForward(self,PortList=[60000],targetIP='224.0.0.10',Interval=0): # 开启网络转发函数

# 启用网络转发，将本机收到的消息，转发给 targetIP 上的 PortList 系列端口
# 默认发给 0 号节点，也就是假设是地面中心，以及组播 IP 224.0.0.10
# Interval 表示每隔数据点，向外发一次包。如果 Interval=0 表示，来包就转发；
# 如果 Interval=5，表示每 5 个数据向外发一次。这个接口可以降低发包速率，调节通信数量。
```

使用方法示例：

net. enNetForward() # 默认使用 PortList=[60000],targetIP='224.0.0.10',Interval=0 的配置，即将本飞机的包，转发到组播 IP+端口'224.0.0.10'+[60000]，同时每个包收到后立刻转发

net. enNetForward([60002,60003]) # 将数据包转发到[60002,60003]两个端口

net. enNetForward([65000], '224.0.0.10',5) # 将数据包转发到[65000]端口，IP 地址为'224.0.0.10'，每隔 5 个包发出一次（即降频为原来的六分之一）

注：通过本接口，可以将数据包转发到网络仿真器，后者加入通信延迟、丢包、误码后，再转发给指定飞机，实现网络通信模拟的功能。

```
def enUavForward (self,CopterIDList=[0] , Interval=0): # 开启飞机数据转发函数

    # enNetForward 的简化版，只需要指令期望发送的飞机列表，会自动用 224.0.0.10 + 60000 系列端口转发
    # uavList=[0 1 2 3 等]，表示想要将数据或 buf 转发给哪些飞机
    # uavList=0 对应 60000 端口，是默认的所有飞机共同的通信端口（也就是能收发所有飞机的数据）
    # Interval 表示每隔数据点，向外发一次包。如果 Interval=0 表示，来包就转发；
    # 如果 Interval=5，表示每 5 个数据向外发一次。这个接口可以降低发包速率，调节通信数量。

    使用方法示例：
    net.enNet2UavID() # 默认发给 0 号飞机，即广播给所有飞机，Interval=0 表示数据立刻发出
    net.enNet2UavID([2,3]) # 发给 2 号和 3 号飞机，Interval=0 表示数据立刻发出
    net.enNet2UavID([2,3],5) # 发给 2 号和 3 号飞机，Interval=5 表示每收到底层飞机的 5 个状态包（自己的位置、速度等信息），向外转发一个包，及降频为原来的六分之一。


def endNetForward(self): # 结束网络转发函数

    # 中止通信转发的函数，在结束仿真时调用

    使用方法示例：
    net.endNetForward() # 中止数据转发程序，退出线程与循环


def netResetSendList(self):  # 清空发送飞机 ID 列表函数

    # 清空发送飞机的列表

    使用方法示例：
    net.netResetSendList() # 清空发送飞机的列表


def netAddUavSendList(self,uavList=[]): # 扩充发送飞机 ID 列表

    # 添加发送飞机列表，可多次调用，最后维持一个总表
    使用方法示例：
    net.netAddUavSendList([2,100,900,901]) # 增加一组飞机到发送列表
    net.netAddUavSendList([6,107,909,1000]) # 增加一组飞机到发送列表，并与前面列表合并


def netResetReqList(self): # 清空请求飞机 ID 列表

    # 清空请求发数据给本飞机的列表
    使用方法示例：
    net.netResetReqList() # 清空发送飞机的列表
```

def netAddUavReqList(self,uavList=[]): #扩充请求飞机 ID 列表

扩充请求发数据给本飞机的列表，可多次调用，最终维护一个总表
使用方法示例：
net. netAddUavReqList ([2,100,900,901]) # 增加一组飞机到请求列表
net. netAddUavReqList ([6,107,909,1000]) # 增加一组飞机到请求列表，并与前面列表合并

def StartReqUavData(self,uavList=[]): #开始请求飞机数据

开始以 1Hz 频率广播数据给请求飞机，后者收到消息后，将会将自己数据传过来
使用方法示例：
net. StartReqUavData () # 开始发送请求数据包
net. StartReqUavData ([2,3,4]) # 将[2,3,4]号飞机加入请求列表，并开始发送请求数据包

def EndReqUavData(self): #结束请求飞机数据

结束请求飞机数据循环
使用方法示例：
net. EndReqUavData() # 停止发送请求数据包

def sendReqUavLoop(self): # 发送飞机数据循环，内部函数

发送本飞机数据的循环线程
注：内部函数，不需要从外部调用

def netForwardBuf(self,buf): # 转发数据包函数

发出一个 buf 包，给到指定飞机或端口
将数据发送到 ForwardIP 和 ForwardPort（端口或列表）
使用方法示例：
buf = struct.pack('iiii',123456789,2,3,4) # 封装一个数据包
net. netForwardBuf(buf) # 将数据包转发出去

def getMavEvent(self): #监听底层 mav 数据循环函数，内部函数

监听底层 mav 的数据更新事件，并将数据转发出去
注：内部函数，不需要从外部调用

```
def StartNetRec(self,MultiPort=60000,MultiIP='224.0.0.10'): # 开始监听端口数据函数

# 开启接收发往指定飞机的数据线程循环
# 默认接收 1 号飞机端口 60001，和组播 IP 224.0.0.10
```

使用方法示例：

```
net. StartNetRec() #开始监听指定端口 IP 数据，默认是 60000 端口+组播 IP 224.0.0.10
net. StartNetRec(60001) # 开始监听 60001 号端口数据，通常对应 1 号端口
```

```
def StartNetRecOwn(self): # 开始监听自身飞机数据函数

    self.StartNetRec(60000+self.CopterID)
# 开始监听发往自己飞机的数据，端口根据 CopterID 自动设置
```

使用方法示例：

```
net. StartNetRecOwn () #开始监听发给自己的数据，默认使用端口 60000+self.CopterID
```

```
def endNetLoop(self): # 结束接听数据函数

# 退出网络数据监听线程循环
```

使用方法示例：

```
net. endNetLoop () #退出监听发给自己的数据循环
```

```
def getMavMsgNet(self): # 监听数据循环函数，内部函数

# 监听网络发往本机数据的线程循环函数
```

注：内部函数，不需要从外部调用

```
def getUavData(self,CopterID): # 获取指定 ID 飞机的数据指针

# 从所有飞机状态数据库中，读取指定飞机的数据指针
```

使用方法示例：

```
TargetCopter = 1
uav = net. getUavData (TargetCopter) #获取 1 号飞机的指针数据指针
if uav. CopterID != TargetCopter:
    print('获取数据失败')
else:
    print(uav. uavAngEular) # 飞机欧拉角
    print(uav. uavVelNED) # 飞机速度
    print(uav. uavPosGPSHome) # 飞机 GPS 原点
    print(uav. uavPosNED) # 飞机本地坐标（相对于起飞或上电的位置）
```

```
print(uav. uavGlobalPos) # 飞机全局坐标，和 CopterSim 保持一致
```

1.3 NetSimAPI 成员变量（给外部访问）

```
self.UavData = [] # 无人机数据结构体向量

# 收到的所有飞机的数据会更新存在本列表
# 注：self.UavData[0], self.UavData[1]，表示最先收到的飞机的数据包，不一定和 CopterID 匹配
```

```
self.UavDict = {} # 无人机数据结构体字典

 # 收到的所有飞机的数据会更新存在本字典中
 # self.UavDict['1'] 表示 CopterID 为 1 的飞机的数据指针，便于快速获取指定飞机数据
 使用方法示例：
 if net.UavDict.__contains__(str(2)): # 检查 2 号飞机是否在字典中
       uav1 = net.UavDict['2'] # 获取 2 号飞机的数据指针
       # 使用指针访问 2 号飞机数据，这里以打印为例
       print(uav2.CopterID,uav2.uavAngEular,uav2.uavVelNED,uav2.uavAngEulars)
```

```
示例：等待期望飞机都在字典中，再开始执行程序
 下面的代码，会进入死循环，每隔 0.5s 检查一次，直到期望接收的飞机[1 2 3 4]全部收到，才退出死循环，进行后续操作。在后续操作中，就能直接访问飞机数据了。
 # 请求的飞机列表，等到列表中飞机数据都收到后，再开始运行后续程序
 reqUavList=[1,2,3,4]

 curList=[]
 print('Waiting for all uavs ...')

 # 下面的函数检查是否收到指定的无人机数据，这里以 1 2 3 4 号飞机为例
 while True:

     isAllRec=True

     for ID in reqUavList:
         if not net.UavDict.__contains__(str(ID)):
             isAllRec=False # 只要有一个飞机数据没收到，就继续等待
             break
```

```
            else:
                if ID not in curList:
                    curList=curList + [ID]
                    curList.sort()
                    print('Current list:',curList)
        if isAllRec: # 所有飞机数据收到后，往下跳出等待循环
            print('All uav data received.')
            break

        time.sleep(0.5)

    # 因为 1234 号飞机数据都拿到了，就能直接通过字典访问飞机数据了。
    # 分别获取 1/2/3/4 号飞机的数据指针
    uav1 = net.UavDict['1']
    uav2 = net.UavDict['2']
    uav3 = net.UavDict['3']
    uav4 = net.UavDict['4']
```

self.bufEvent = threading.Event() # 线程事件，用于通知外层 Python 程序有数据更新了

用于网络通信的事件信号

使用方法示例：

接收程序如下 testRec.py：

```
import NetSimAPIV4
import struct
import copy # 导入 copy 库

net = NetSimAPIV4.NetSimAPI(1)

net.StartNetRec() # 默认开始监听 60000 端口，即所有飞机。

# 假设对方使用如下接口发包
# checksum=123456789 # 预设的校验码，用于确保包的正确性
# CopterID=2 # 数据包来之飞机的 ID，用于区分包的来源
# Data1=3 # 自定义数据位
# Data2=4 # 自定义数据位
# buf = struct.pack('iiii',checksum,CopterID,Data1,Data2) # 封装一个数据包
# 包长度为 4*4 = 16，第一位为校验位 123456789
# net. netForwardBuf(buf) # 将数据包转发出去
# 我们通过如下方法，将包解析出来

# 创建一个类，用于存储飞机数据
```

```
class UAVData:
    def __init__(self):
        self.hasUpdate=False
        self.CopterID=False
        self.Data1=False
        self.Data2=False

    def update(self,Data):
        self.hasUpdate=True
        self.CopterID=Data[1]
        self.Data1=Data[2]
        self.Data2=Data[2]

# 创建一个全局变量字典，用于存储飞机数据
UAVDataDict={}

print('Waiting for data...')
while True:
    net.bufEvent.wait() # 会一直阻塞在这里，直到有新的数据包收到
    bufData = copy.deepcopy(net.bufData) # 立刻将包拷贝出来
    bufHead = copy.deepcopy(net.bufHead) # 立刻将包头拷贝出来
    net.bufEvent.clear() # 清除 set 位，使得下层能够继续更新数据

    if len(net.bufData) == 16:
        Data = struct.unpack('iiii',net.bufData) # 获取所有数据向量
        checksum = Data[0]
        if checksum==123456789: # 校验通过，说明是对的包
            CopterID = Data[1]

            if UAVDataDict.__contains__(str(CopterID)): # 如果飞机数据已经在字典中
                UAVDataDict[str(CopterID)].update(Data) # 直接更新数据
            else:
                uData = UAVData() # 创建一个新的空结构体
                uData.update(Data) # 更新数据
                UAVDataDict[str(CopterID)] = copy.deepcopy(uData) # 拷贝加入字典

            print('Data:
',UAVDataDict[str(CopterID)].CopterID,UAVDataDict[str(CopterID)].Data1,UAVDataD
ict[str(CopterID)].Data1) # 打印数据

            continue # 成功解析数据包，就跳过本循环，后续代码将不会执行

    if len(net.bufData) == 24:
         # 如果未被处理，则继续处理其他的包
```

```
        # 解析包
        # 解析成功
        continue # 跳出本循环
```

发送程序如下 testSend.py：

```
import NetSimAPIV4
import time
import struct

net = NetSimAPIV4.NetSimAPI(2)

net.enNetForward() # 默认发给 60000 端口，即所有飞机。

time.sleep(5)

buf = struct.pack('iiii',123456789,2,3,4) # 封装一个数据包，备注 2 号飞机
net.netForwardBuf(buf)
print('Send Data CopterID 2')

time.sleep(5)

buf = struct.pack('iiii',123456789,3,3,4) # 封装一个数据包，备注 3 号飞机
net.netForwardBuf(buf)
print('Send Data CopterID 3')
```

self.bufData=[] # 存储当前的数据 buf

使用方法见 self.bufEvent 注释

self.bufHead=[] # 存储当前的包头信息

```
# bufHead=[checkSum,CopterID,sendMode,StartIdx,SendMask,TimeUnix]
#checkSum=12345678 # int32 型，校验位
#CopterID, int32 型，当前飞机的 ID 号
#sendMode,  int32 型，发送模式，有组播，也有单拨
#StartIdx,  int32 型，发送列表的起始飞机序号
#SendMask,  uinit64 型，从序号开始的 64 个飞机是否发送
#TimeUnix, double 型，发送前获取到的电脑时钟
```

使用方法见 self.bufEvent 注释

1.4 UAVSendData 数据结构体

平台提供了一个 UAVSendData 类（结构体），记录从内层 PX4MavCtrl 通信实例读取到的飞控信息（主要是姿态和位置等），并转发到局域网给其他飞机，便于实现多机的编队分布式控制。

1.4.1 数据说明

```
# UAVSendData
# double uavTimeStmp 仿真时间戳（单位秒）
# double timeDelay 这个消息包的延迟，
# 注意：当前电脑时间戳-收到包的时间戳等于延迟，因此发包和发包电脑需要是同一台
# 注意： 如果收发包不是同一台点，那么需要两台电脑做时钟同步
# float uavAngEular[3] 欧拉角 弧度
# float uavVelNED[3] 速度 米
# double uavPosGPSHome[3] GPS 维度（度）、经度（度）、高度（米）
# double uavPosNED[3] 本地位置 米 （相对起飞点）
# double uavGlobalPos[3] 全局位置 （相对与所有飞机的地图中心）
# d6f9d = 长度 104
```

1.4.2 详细定义

```
class UAVSendData:
    # 无人机数据类，从 mav 中拿到无人机的数据，并转发到网络，给别的飞机
    # 包含飞机的位置、速度、姿态角等
    def __init__(self):
        self.hasUpdate=False # 数据包是否更新
        self.timeDelay=0 # 数据包延时时间
        self.CopterID=0 # 来自飞机的 CopterID
        self.uavTimeStmp=0  # 本数据包发送的时间戳
        self.uavAngEular=[0,0,0] # 飞机欧拉角
        self.uavVelNED=[0,0,0] # 飞机速度
        self.uavPosGPSHome=[0,0,0] # 飞机 Home 点 GPS 坐标
        self.uavPosNED=[0,0,0] # 飞机局部位置（相对起飞点或解锁点）
        self.uavGlobalPos=[0,0,0] # 飞机全局（和 CopterSim 真值一致，相对地图中心）
```

1.5 接口例程汇总简介

例程目录在：RflySimAPIs\9.RflySimComm\0.ApiExps\0.RflyNetPy （加入超链接引用）
待补充…
下面准备一个表，介绍有哪些例程，展示了什么功能。