**通用**

【关于“L2 设备”和”L2层”】

“L2 设备”指的是交换机，不具备划分子网功能。“L2”层是数据链路层。

【关于WAN和LAN】

Wide Area Network的首字母所写，即代表广域网;而LAN则是Local Area Network的所写，即本地网(或叫局域网)。一个LAN内部由交换机

【关于inet和inetmanet的区别】

Manet比inet多了关于车联网的模块,包括比如src/inet/routing里的extra文件夹，里面的一些路由方式应该都是专门针对车联网进行设计的。UAVSim用了OLSR协议，而AVEN用了AODV协议。

【关于inet包本身src和其他】

因为inet既提供了基本的扩展例子(比如examples中的一系列),又作为一个完整的扩展库提供了他们的实现底层源码(在src中)，我们实际上在调用inet包时候(调用方法见下方)，调用的是他们实例的小模块，不必仔细挨家挨户地调整源码!

如何正确导入inet(或inetmanet)到其他项目中??

1. 先常规import inet(或inetmanet)到IDE中，然后build,debug和release都来,release需要依靠run像examples这种里面的ini文件触发。注意其版本问题，也可以完事儿先run example的例子看看是否编译好了。
2. Import主要的project(比如UAVSim)到IDE中，然后右键🡪properties🡪Project References,把前面已经import并编译好的inet(或manet)勾选上即可。

【关于omnet的packet工程机制】

1.package包名的拼接

首先在package.ned中定义:

package foo;

那么其他模块，比如inet/src/transportlayer/TCP.ned:

package foo.transportlayer;

即foo代表inet这整个大的文件夹组成的package，而下面的各种子模块，就直接按照路径进行拼接，最后一层到它紧挨着的第一个文件夹名字为止。

【import中”\*”和”\*\*”的区别】

In wildcard patterns, one asterisk ("\*") stands for "any character sequence not containing period", and two asterisks ("\*\*") mean "any character sequence which may contain period".

也就是说\*只能同级路径(不能含有”.”),而\*\*可以直接跳多级路径(可以含有很多”.”)。



【如何导入自定义的图片】

在inet—images里先加载进去。

【调试小方法】

关于不同的模块，可以在仿真图标的右键点击“open details for ……”来进行来进行具体的变量值查看。



【什么是各种Host的”PingApp”】

实际上针对某个Host，先设置一下numPingApps，然后针对每个pingApp[\*],设置一下type和目的Host，则“App“实际上就是建立了两个Host之间的通信，而不是什么复杂的应用！至于UDP什么的，会有专门的numTcpApps/numUdpApps等来设置，跟ping不是一类！

【关于StandardHost和WirelessHost和AdHocHost】

StandardHost是基础的，它包含像TCP，UDP和IP的协议组件，用于插入应用程序模型的插槽以及各种网络接口（NIC）。WirelessHost是它的变体，它基本上是为无线方案预先配置的StandardHost。

在此基础上，还有**AdHocHost:**

**包含:路由、移动性、电池组件的WirelessHost。**

仅支持IPV4，TCP/UDP/Ping/ICMP。

关于网卡类型的配置:

\*\*.wlan.typename（可参考Wireless的tutorial的\*.host\*.wlan[0].typename）

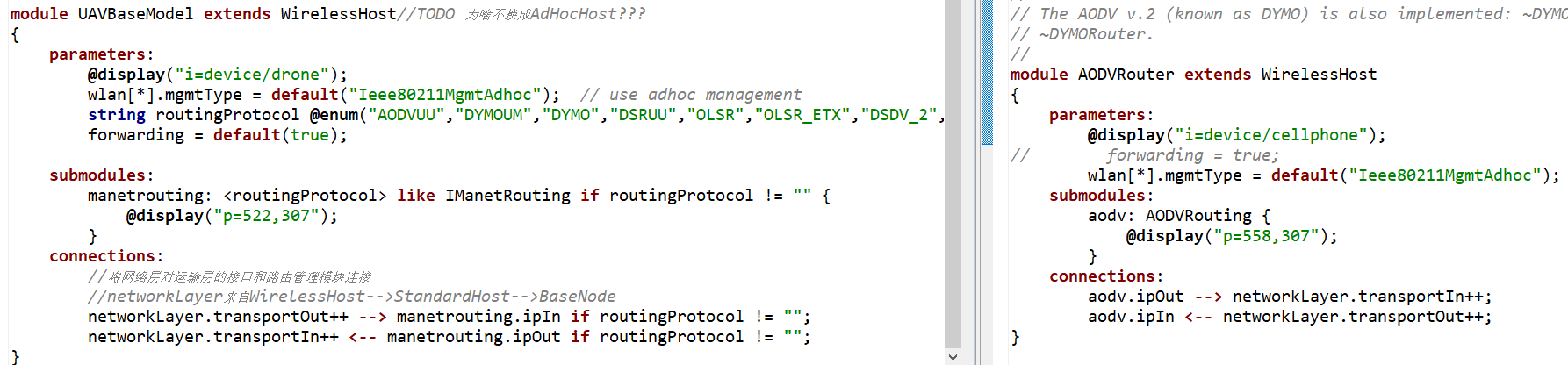
关于移动性的配置:

\*\*.mobility.typename(可参考Wireless的tutorial的\*.hostR\*.mobilityType = "LinearMobility")，当然推荐Mass方式移动。

**UAVSim项目理解**

**Wireless的第10个仿真AODV非常关键！可以重点参考！后续11加了障碍物也很好！**

【关于UAVBaseModule和AODVRouter的区别】



可以看到，作者实际上是希望通过自己扩展出的“UAVBaseModel”来获得一个快捷的路由选择方式，不论是AODV还是OLSR之类的，都被集成在一个叫IManetRouting的路由集合里，它可以加载AODVRouting，OLSRRouting等等，这样仅仅是比较方便管理，但其实也没别的太多新鲜的。注意，这个IManetRouting在src源文件也是有的，networklayer/contract/IManetRouting.ned。

我们真正做的话 还是直接用AODVRouter这种就行。

【关于AttackHost和UAVHost的区别】

很坑爹！乍看之下UAVBaseModel是WirelessHost；而AttackHost是AdhocHost，但是仔细点进去UAVBaseModel和AttackHost，发现除了图标不一样，其他都一样！也就是说**实际上UAVBaseModel仍然是AdhocHost！除了图标样子不一样！没有区别了！**

而UAVHost实际上和AttackHost是平级的，同样，这俩除了图标不一样之外，其他的没啥区别。。。都可以认为是移动性的节点(至少本来是可以移动的，只不过在ini文件中被用作固定点或通信塔了。)

【关于TestNetwork和UAVNetwork的异同】

两者都是network，TestNetwork多了个**IntegratedCanvasVisualizer**(虽然还被注释掉了..)，应该就是跟IntegratedVisualizer那种差不多，算是debug时候的一个可视化界面?

【基本元素——不可或缺】

**A.Ieee80211ScalarRadioMedium**

在OMNeT++中，所有的无线仿真都要经过一个无线电媒体模块！无线电发射机，它负责定义各个Host的通信范围，考虑信号的传播/衰减/干扰等。

也可以用IdealRadioMedium。

控制通信范围的两种手段(需要视radio的type来选择设置方法):

①\*\*.wlan[\*].radio.transmitter.power = 1mW

②\*.host\*.wlan[0].radio.transmitter.communicationRange = 500m

注意在一般的UAV网中，不应该所有UAV都能直接进行通信，他们之间需要routing来转发路由。

不考虑碰撞干扰:

\*.host\*.wlan[0].radio.receiver.ignoreInterference = true

无线电数据传输速率设置:

\*.host\*.\*\*.bitrate = 1Mbps

与他配套的，还有就是数据链路层(MAC子层)的NIC网卡设置，对应关系:

IdealRadioMedium无线电对应 \*.host\*.wlan[0].typename = "IdealWirelessNic"的NIC(最理想)

[UnitDiskRadio](https://doc.omnetpp.org/inet/api-current/neddoc/inet.physicallayer.unitdisk.UnitDiskRadio.html)无线电对应 \*.host\*.wlan[0].typename = " WirelessInterface"的NIC

ApskScalarRadio对应ApskScalarRadioMedium(最现实)

详见:

https://inet.omnetpp.org/docs/users-guide/ch-physicallayer.html?highlight=ignoreinterference

**B.IPv4NetworkConfigurator**

能够进行自动IP地址分配，自动路由表配置，通过XML文件进行手动指定上述两者或进行层次网络的划分……

剩下3个暂时先去掉了 因为不知道干啥的:

**C.RoutingTableRecorder**

**D.LifecycleController**

**E.ScenarioManager**

**[[可以添加的&待注意的]]**

A.IntegratedCanvasVisualizer可以添加这个模块以进行路由表的可视化等等！Wireless教程中也用到这个模块，可以显示物理环境中的物理对象，移动轨迹，发现的网络连接性，发现的网络路由，正在进行的传输，正在进行的接收，传播的无线电信号，统计信息等等。详细见wireless第二个。

[路由]

\*.visualizer.routingTableVisualizer.displayRoutingTables=True 路由表显示

\*.visualizer.routingTableVisualizer. destinationFilter = "host7" 指定目的地点的范围

nodeFilter 指定路过的节点范围

注意:当有无法直接到达的两个节点需要通信时候，一定要写上\*.host\*.forwarding = true，这样才能中继转发进行路由!

[数据链路层]

\*.visualizer.dataLinkVisualizer.displayLinks = true

[物理层]

\*.visualizer.physicalLinkVisualizer.displayLinks = true 物理层传输显示

\*.visualizer.physicalLinkVisualizer.packetFilter = "UDPData\*" 指定数据链路层报文类型

**[网络层]**

\*.visualizer.networkRouteVisualizer.displayRoutes = true 蓝色的线 没啥事儿的话就用这个就行

[其他]

\*.visualizer.mediumVisualizer.displaySignals = true 显示传输为彩色圆环(一圈圈)

\*.visualizer.interfaceTableVisualizer.displayInterfaceTables = true 可视化接口信息

#注意忽略Switch和AP接入点，因为他们没有IP地址

\*.visualizer.interfaceTableVisualizer.nodeFilter = "not (\*switch\* or \*Switch\* or \*AP\*)"

**B. 在xml文件中，<autoroute metric="errorRate" sourceHosts="\*\*"/>以错误率为度量标准，最适合用到无线网络中！这个暂时先去掉了 因为我看统计信息都是-2147xxxxx，我认为因为aodv本身就是反应式的 我们在初始化时候只建立ip而不进行路由设置。**

<autoroute metric="errorRate"/>

实际上在这里的后续，我们如果真的和python交互了则并非通过ip地址进行聚类和子网划分的，因此ip地址也就不用非写的很详细了，甚至顺序排列都可。

C. **在非常高动态，或者是使用了反应路由协议这种的时候，**\*.configurator.addStaticRoutes = false,默认是true，在Ad Hoc Network里，路由表往往是变化的而不是静态！比如AODV这种过一会不反应就去掉的…所以只有IP地址是固定的，但是路由信息最好设置为False！一旦用了AODV，它就能反应式地接管路由表更新，因此不要初始静态路由设置。

D.消息的发送间隔(暂时没用):

比如这种\*.hostA.udpApp[0].sendInterval = exponential(12ms),指数分布的时间间隔，均值12ms

E.关于信道之间的干扰

实际上是在radio和mac层以及pathloss等体现的。

关于可视化:

\*.host\*.wlan[0].radio.receiver.ignoreInterference = false(默认即是它，要考虑干扰)

\*.host\*.wlan[0].radio.transmitter.interferenceRange = 500m

F.关于能耗建模(暂时没用):

\*.host\*.wlan[0].radio.energyConsumerType = "StateBasedEpEnergyConsumer"

在radio里设置，详细见wireless里的第“八”实例。

**G.** 节点的mobilityType 设置:VehicleMobility”/”MassMobility”/”LinearMobility

还包括节点的移动轨迹，速度矢量可视化:

\*.visualizer.mobilityVisualizer.displayVelocities = true

\*.visualizer.mobilityVisualizer.displayMovementTrails = true

**H.** 节点的MAC层的处理数据包的队列长度限制——网内拥塞！

\*.host\*.wlan[0].mac.maxQueueSize = 10

I.给环境添加障碍物

可以给环境添加一个森林障碍物，即forest，详见Wireless的第11个仿真。

【关于网络层协议】

在UAVBaseModel里，routingProtocol基本上选择OLSR,然后实例化一个子model,叫manetrouting,其中选用了routingProtocol类的IManetRouting。IManetRouting其实就是一个in和out的router，其最大作用在于将**WirelessHost-->StandardHost-->BaseNode**中的networkLayer.transportOut和in与ImanetRouting对接了!所以实际还是这个**WirelessHost**!

**重点关注所有画红色的部分**

**总学习**

学习顺序:

A．visualization

<https://inet.omnetpp.org/docs/tutorials/visualization/doc/step1.html>

B．Configurator——IPv4 network configurator

<https://inet.omnetpp.org/docs/tutorials/configurator/doc/step1.html>

C.Wireless

<https://inet.omnetpp.org/docs/tutorials/wireless/doc/step1.html>

**第一部分 visualization**

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------------------------------------------------------------------

1. Enabling visualization

1.什么是visualization

集成的可视化工具，将所有单独的、分散的可视化工具集成到一个模块中。每个**IntegratedVisualizer**有一个canvas和一个osg visualizers。他们两个可以一起设置，也可以单独设置。

结果是一个空的2D空间平面。

1. Showing Boston downtown

1.添加坐标系统

coordinateSystem: OsgGeographicCoordinateSystem {

**parameters**:

**@display**("p=100,150");

}

OsgGeographicCoordinateSystem使用内置的OSG API提供了一个精确的地理坐标系统,包括经(Longitude)纬(latitude)度的操作，原点定义等。

2.定义visualizer的相关属性

在ini文件中，包括显示的类型和地图。

结果是显示了波士顿街道。

1. 添加坐标轴

仍然用的VisualizationB.ned。在ini中设置了包括轴长axisLength、原点坐标位置playgroundAltitude等。并设置阴影playgroundShading、不透明度playgroundOpacity。

1. 啥也没干
2. 用了3D模型

用的VisualizationD.ned。**加入了两个WirelessHost移动节点和一个AccessPoint(AP)。两个WirelessHost通过AccessPoint(AP)进行通信。**

1. **移动节点可视化(\*\*)**

仍然用VisualizationD.ned。设置了节点的移动性。行人采用坐标定义:

\*.pedestrian\*.mobilityType = "MassMobility"**#乱走 移动类型**

#所有行人z方向移动范围都一样

\*.pedestrian\*.mobility.initialZ = 0.5m

\*.pedestrian\*.mobility.constraintAreaMinZ = 0m

\*.pedestrian\*.mobility.constraintAreaMaxZ = 1m

#移动更换频率

\*.pedestrian\*.mobility.changeInterval = 2s

\*.pedestrian\*.mobility.changeAngleBy = uniform(-10deg, 10deg)

\*.pedestrian\*.mobility.speed = 1.3mps

之后设置了各个行人的移动范围，**防止走出通信范围。**

行人0

\*.pedestrian0.mobility.initialX = 395m

\*.pedestrian0.mobility.initialY = 65m

\*.pedestrian0.mobility.constraintAreaMinX = 390m

\*.pedestrian0.mobility.constraintAreaMinY = 63m

\*.pedestrian0.mobility.constraintAreaMaxX = 440m

\*.pedestrian0.mobility.constraintAreaMaxY = 230m

AP设备采用经纬度定义(initialLatitude\initialLongitude)。注意需要先定义coordinateSystemModule。

之后，还可以设置一些可视化的属性:展示运动轨迹(displayMovementTrails)运动速度(displayVelocities)和运动方向(displayOrientations)。

1. **显示通信和干扰范围**

注意要调小AccessPoint和WirelessPoint的RadioMedium的发射功率！因为无线电的发射机功率决定了他们的通信范围。 好像太大则通信、干扰的范围也很大。之后设置显示就行。

1. 显示IP地址

IPV4的地址是**IPv4NetworkConfigurator**自己设置的！在之后学IPV4网络配置器时候会详细说。

考虑到这个可视化工具会显示所有节点所有接口的地址，因此限定范围为“pedestrian”的“wlan\*”接口。这样就只显示行人的wlan接口地址(**我们估计也仅显示UAV的wlan接口!**)format参数设置显示一些额外信息，比如MAC地址“%N %\\%m”。

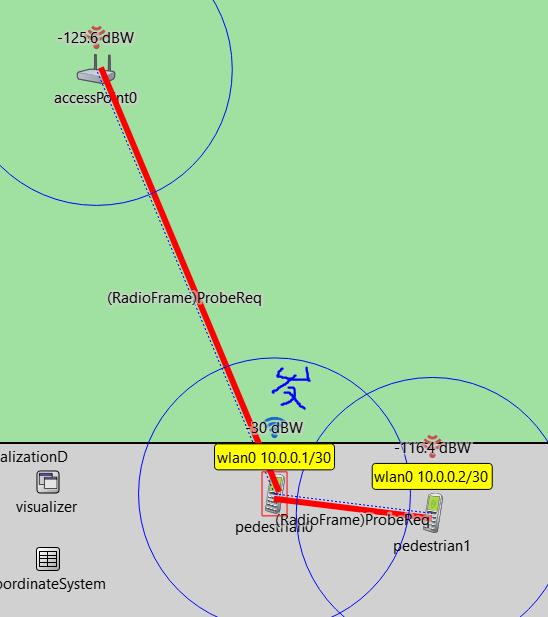
1. 显示WIFI关联(只有用到IntegratedVisualizer的ieee80211Visualizer才有wifi)

所谓“关联“，就是每个行人(WirelessHost)和临近的AdHocHost是怎么配对的，给显示出来。

1. **显示发送和接收**

可视化每个节点是处于发送状态，还是处于接受状态。假设行人0是发送方，行人1是接收方，俩人进行通信，用的是Voip恒定比特率源信息。

通过mediumVisualizer分别设置显示传输状态(displayTransmissions)和接收状态(displayReceptions)。这俩状态会以图标形式显示在节点周围(下图中p0在发，其他在接收)。



1. 显示传播信号

进一步可视化传播信号。mediumVisualizer.displaySignals设置为true。

1. 可视化物理层连接活动

显示网络节点之间的流量。由physicalLinkVisualizer.displayLinks设置为True，设置packetFilter为 “\*VoIP\*”来仅考虑VoIP包的流量传输活动，fadeout设置箭头淡入淡出时间。没啥用

1. 可视化data link数据链路层活动

由dataLinkVisualizer.displayLinks设置为True。也看不出啥用

1. 显示静态数据

理论来说可以显示端到端的数据延迟，在此是针对VoIP的ms级统计信息。但是看不出啥效果，暂时先不管。

1. 直观地可视化路由表的条目(路径)

首先复杂了网络的结构，加入了pedestrainVideo作为视频流应用程序的发送端(客户端),注意这个和pedestrians暂时还没有直接关系。又加入了videoStreamServer作视频流应用程序的接收端(服务器)，还加入了它的邻居server(暂时没啥用)，因此通过switch来区分子网。最后，通过router把这俩不同局域网连接起来作路由交换。

在ini文件中先定义这些新模块的位置，然后设置UDP的视频流信息，包括消息大小等，然后设置videoStreamServer为服务器,pedestrianVideo为客户端，对应好端口。最后，定义了routingTableVisualizer的目的地址，手动加入从pedestrianVideo到videoSteamServer要经过的节点，用绿线显示出来。

其实就是路由路径显示……

1. 显示802.11标准下的节点访问状态的信息(信道争用状态)

暂时没啥用

1. **网络路径活动可视化(这里比较复杂了)**

为视频流启用网络路径活动可视化。可视化有助于验证videoStream数据包是否通过了客户端节点的网络层。这里面源码有一处细节:

\*.configurator.config = **xmldoc**("configurationF.xml")

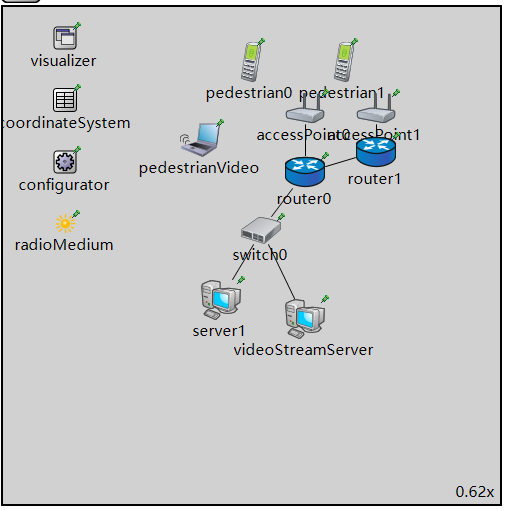
意思是通过IPv4NetworkConfigurator，设置指定的xml文件代表该仿真的网络信息。

注意到.config属性本身就是一个xml文件的格式，因此IPv4NetworkConfigurator的信息一定本身就和xml有关的，用xml文件记录一些基本信息。

**xml** config = **default**(**xml**("<config><interface hosts='\*\*' address='10.x.x.x' netmask='255.x.x.x'/></config>")); *// XML configuration parameters for IP address assignment and adding manual routes*

针对这个仿真本身，其在原基础上加了一个AP1，一个router1，并且用到了router协议，叫做DHCP，查阅发现路由协议DHCP(Dynamic Host Configuration Protocol，动态主机配置协议)是一个局域网的网络协议，对应的数据链路层协议为UDP协议工作， 主要有两个用途:给内部网络或网络服务供应商自动分配IP地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段，在RFC 2131中有详细的描述。

它认为两个路由器都有DHCP来管理网络，因此行人也要进行DHCP的app设置。此外还要进行一些wlan设置。但是**由于例子无法运行，因此我还不太明白**！



【关于DHCP】

DHCP基于广播，操作四个阶段:IP租用请求、IP租用提供、IP租用选择、IP租用确认。

1. lan广播找到DHCP服务器，那么发起lan广播的这一端就是DHCP客户端。
2. 可能有很多DHCP服务器返回了OFFER报文进行IP地址预分配。
3. 然后DHCP客户端选择（一般是第一个收到的offer）进行确认REQUEST

D.DHCP服务器收到确认后，带着预先准备好的ip发送ACK报文，完成整个过程。

E.那么这里的服务器就是我们的网关咯，也就是CPU，它是装过dhcp服务的openwrt系统。

1. **显示物理对象**

将一些物理阻碍添加进环境，因为他们会对信号传输产生影响。他说模型太大了出圈了啥的。。。我也不知道咋回事 先不管

1. 显示障碍物丢失

遇到障碍物时，无线信号强度通常会降低，并可能导致数据丢失。但是降低信号强度的地方并不总是很清楚，因此可视化可以帮助促进对仿真工作的理解。在此步骤中，我们将可视化障碍物丢失。obstacleLossVisualizer。

1. 可视化数据包丢弃

这个部分仿真又多了一些节点之类的，但是我们暂且不管，主要是VisualizationH的car是个WirelessHost，也可以自由移动(下图那个笔记本就是它)。**这边可以借鉴！但是注意它的mobilityType = "VehicleMobility"。**

*# car mobility parameters*

\*.car.mobilityType = "VehicleMobility"

\*.car.mobility.coordinateSystemModule = "coordinateSystem"

\*.car.mobility.updateInterval = 100ms

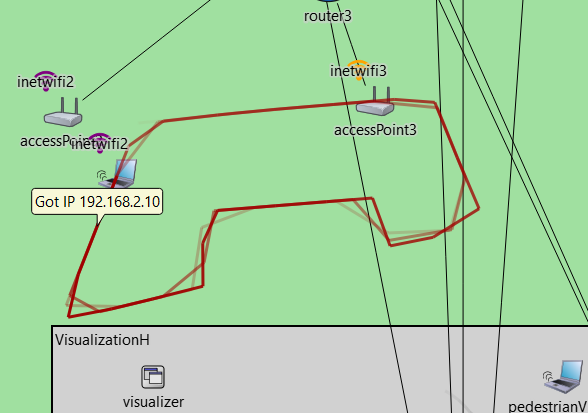
\*.car.mobility.waypointFile = "waypoint1.txt"

\*.car.mobility.waypointProximity = 4m

\*.car.mobility.speed = 12mps

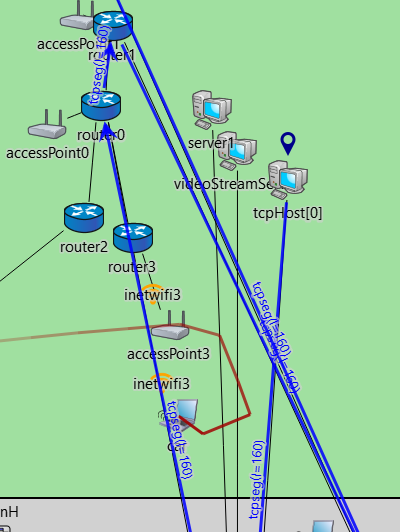
然后仍然是DHCP和UDP两个App的相关设置等。最后由packetDropVisualizer.

displayPacketDrops设置可视化数据包丢弃，效果如下图:



1. 可视化传输连接

针对运输层应用的可视化。蓝色的线显示了tcp啥的消息。



1. **显示网络节点之间的链接断开**

移动节点走出接入点的通信范围时候，就会链路断开，当发送方节点长时间未从接收方节点获得ACK帧时，导致丢包。我把car的移动路径(waypoint1.txt)改了一个很远的点，但是**好像没触发断开的提示**，就先不管了。但是这个**还是可以用到的!UAV断开。**

1. 可视化移动节点的wifi切换(无缝切换???)

还没实现。。

1. 可视化路由信息变化

还没实现

1. **总结，可视化全部信息**

可用的:

1. 可视化通信和干扰范围

\*.visualizer.\*.mediumVisualizer.displayCommunicationRanges = true

\*.visualizer.\*.mediumVisualizer.displayInterferenceRanges = true

1. 可视化动作轨迹

\*.visualizer.\*.mobilityVisualizer.displayVelocities = true

\*.visualizer.\*.mobilityVisualizer.displayOrientations = true

1. 可视化一圈圈扩散的传播信号

\*.visualizer.\*.mediumVisualizer.displaySignals = true

\*.visualizer.osgVisualizer.mediumVisualizer.signalShape = "both"

其他还有显示物理链路活动和障碍损失，暂时不用。

哦，输入参数设置:30,10,1s…….

**第二部分 IPv4 network configurator**

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

【关于General基本设置】

本部分的设置是全局都可以显示的。

**A.Dump\_XXX**

Dump会让configurator的信息转存到log日志文件，方便调试。

需要在每个子ini文件中设置以生成日志文件,如:

\*.configurator.dumpConfig = "step4\_dump.xml"

B.路由设置

arpType = "GlobalARP"预先填充所有节点的ARP表，因此在模拟开始时无需ARP交换。

routingTable.netmaskRoutes =“”防止将网络掩码路由也添加到路由表。网络掩码路由也是配置器默认添加的，因此需要netmaskRoutes将其关闭以避免重复的路由。

C.无线网比特率

设置成54Mbps，ACK报文传输更快。

**D .interfaceTableVisualizer**

可视化接口信息，包括接口名称，IP地址。但是要去除Switch和AP的，因为他们没有地址。

1. **全自动IP地址分配**

很多情况下，**IPv4NetworkConfigurator模块**仅使用**默认设置**就可以正确配置网络。因此，omnetpp.ini中此步骤的配置基本上为空(后续可能改一改)。其主要包括两个内容:

assignAddresses = default(true)。Configurator默认自动进行地址分配。可以在ＸＭＬ文件中修改详细信息。

assignDisjunctSubnetAddresses = default(true)。Configurator是否为不同链接上的节点分配不同的地址前缀和网络掩码，默认进行分配。它可以同时确保：相同子网有相同掩码和前缀，不同子网之间的掩码和前缀不同。

对于IPv4NetworkConfigurator模块，如果不设置XML模块，就默认如下：

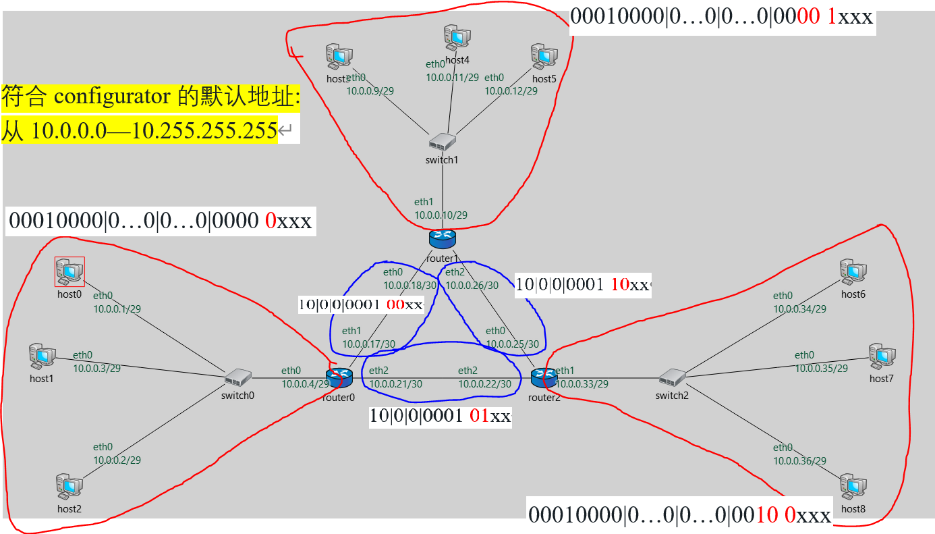
config=

default(xml("<config><interface　hosts='\*\*' address='10.x.x.x' netmask='255.x.x.x'/></config>

"))

因为host是＂\*\*＂所以会把IP地址分配给所有主机接口。从10.0.0.0—10.255.255.255，网络掩码范围255.0.0.0-255.255.255.255。

结果是划分了3个子网：



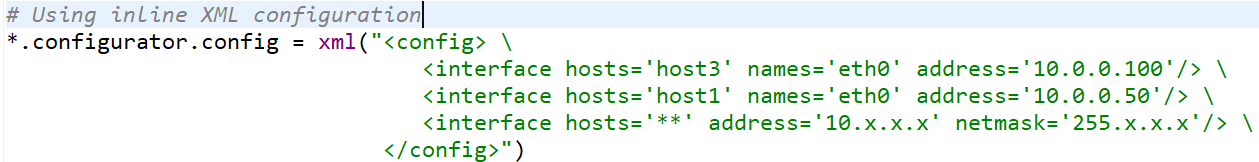
1. **修改IP地址**

在这个仿真里面，手动为某些节点或接口指定IP地址,同时保留对网络其余部分的自动分配。

将10.0.0.50(10|0|0|0011 0010)地址给host1,10.0.0.100(10|0|0|1100 0010)给host3，其他自动分配。

推荐使用**xmldoc() function与外部xml文件交互，读取**。

对此，在ini文件中直接写出了xml文件(configurator.config),前两条指定，第三条默认！:



**【关于XML文件格式】**

<config></config>是根。

每个<interface></interface>(或者<interface ……/>)定义一个或一组接口，其中属性:

hosts:选主机，可以用”\*.”来选全部。

names:选主机的接口。

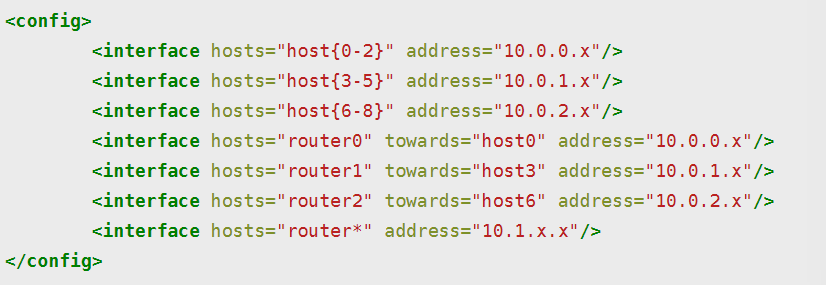
address:地址，”x”代表自动配置。“”空值代表不分配地址，可以用于后续动态配置。

netmask:网络掩码。跟地址命名格式差不多。

每个xml文件必须至少包含一个<interface>，尽量包含自动划分的配置！不然就空值。

1. 自动将一定范围内的IP地址分配给子网

为子网划分一定范围内的地址。这次用xmldoc()方法读取xml文件了!



前3行为不同子网的主机分配不同前缀的IP地址。接下来3行设定每个路由器对应着子网的那个接口，认为这个接口也应该对应(“towards”)地在这个子网中。

最后把所有路由器的其他接口(也就是和相邻路由器相连接的接口)默认配置。

1. **全自动静态路由配置**

跟IP地址配置一样，**默认情况下IPV4NetWorkConfigurator就可以进行路由配置了**!不用再添加什么。

和自动配置IP地址一样，自动配置路由表也有一些可以修改的属性:

**addStaticRoutes = default(true): 是否开启自动填充路由表，如不开启只能XML手动设置了。**

**addDefaultRoutes = default(true): 是否自动添加默认路由(比如只有10.0.1.x指向某个主机，其他地址都走默认路由)。默认是开的，但是如果对于某路由器来说，如果它对应的链路上的节点，有一些不能直接连到它的网关(比如超出范围了),那么就需要设置为Flase，不然那个host的路由表里，走默认路由直连了！**

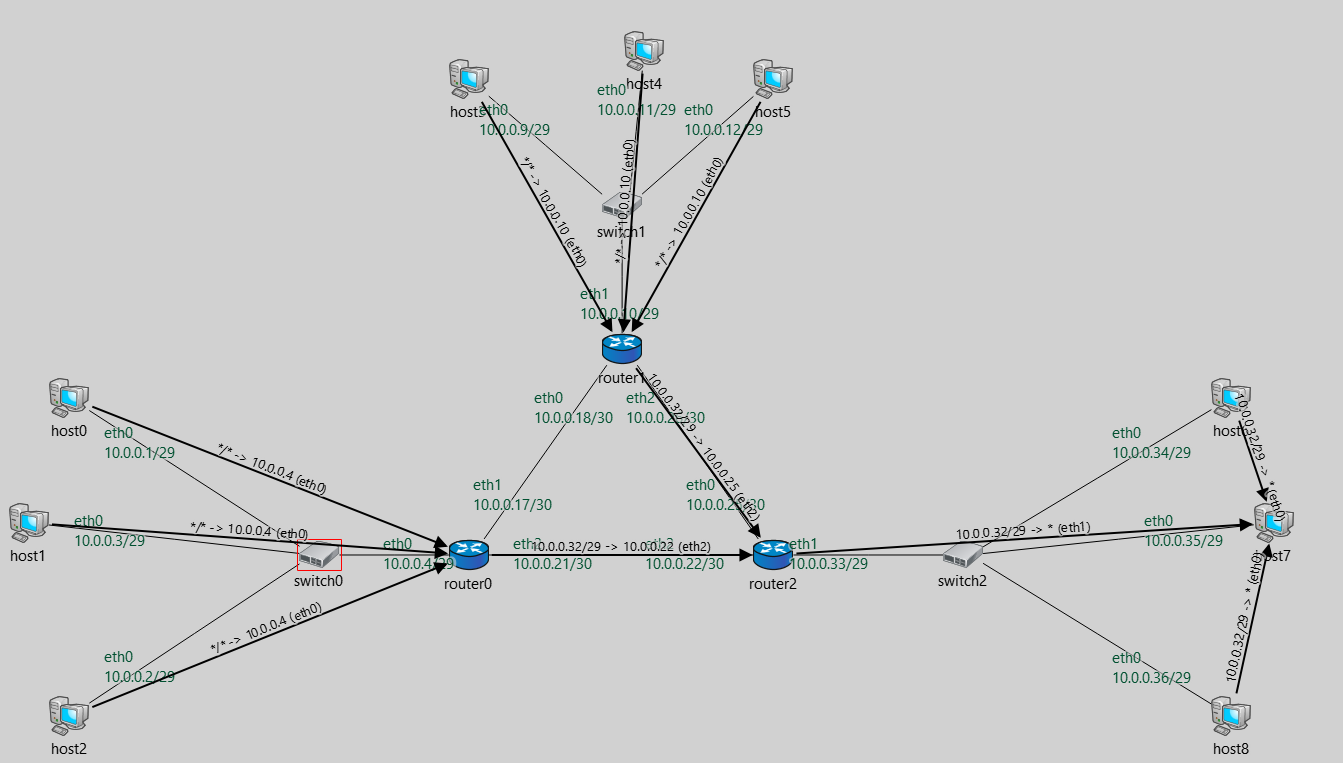
addSubnetRoutes = default(true): 通过把某条路由信息指向某个子网而不是单一接口来简化路由结构。

optimizeRoutes = default(true): 可能的话通过合并条目来优化路由表。

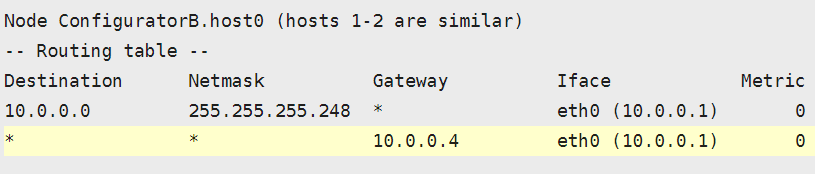
加了一个从host1到host7的pingApp，进行相互通信。

routingTableVisualizer.displayRoutingTables = true激活可视化。默认会把所有路由路径都以箭头表示出来，在这里通过destinationFilter和nodeFilter约束显示范围。

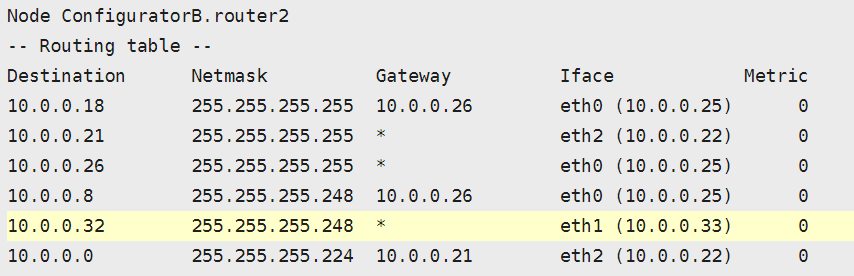
另外，arpType=GlobalArp，以使数据包交换更简单: GlobalArp会预先填充所有节点的ARP表，因此在模拟开始时，无需进行ARP交换;而\*\*.routingTable.netmaskRoutes =“”可防止路由表模块将**网络掩码路由**添加到路由表。**网络掩码路由**意味着具有**相同网络掩码但不同IP地址的节点应直接相互到达**。这些路由也由配置器添加，因此netmaskRoutes被**关闭以避免重复的路由**



显示了到host7的路由路径。此外，在动态图中还可以一开始Switch0和Switch2经过时候的**广播**，还有握手之后不断进行单播。以host1发出的msg到Switch0广播为例，看输出信息(\*.configurator.dumpConfig = "step4\_dump.xml)：网关的\*表示网关与目的地相同。

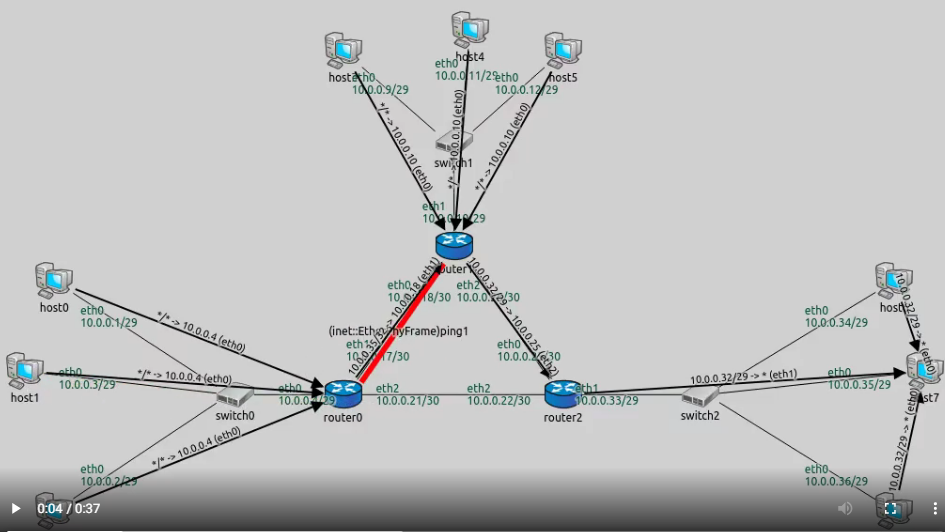


主机:一个路由表条目，可以直接到达同一子网上的其他节点;一个默认路由，该路由以路由器为网关，用于将数据包发送到与自己不同的子网外地址。



路由器:和主机差不多，在自己的子网里则gateway是“\*“。

1. 手动配置部分路由 其他保持自动配置
2. 配置一台特定主机的路由



host0—host7本来是经过router2，但是现在我们希望它经过router1，因此修改router0的路由表:到达10.0.0.35(host7)的数据包要经过router1。xml设置：

**<config>**

**<interface** hosts="\*\*" address="10.x.x.x" netmask="255.x.x.x"**/>**

**<route** hosts="router0" destination="10.0.0.35" netmask="255.255.255.255" gateway="10.0.0.18" interface="eth1" metric="0"**/>**

**</config>**

先统一设置默认的，然后再修改指定的。注意address, netmask, gateway, interface, metric是可选参数。注意，为什么10.0.0.35的子网掩码最后也是255？两个原因:

* 1. 唯一的确定了到host7的主机的路由，符合题目要求。
  2. 该路由为新添加的路由，比默认的自动路由约束更强，才会生效。

但是注意它只是把destination设置成了较强的主机和掩码，并不改变他们的地址掩码。

另外，注意这里的黑线还是用step4的只能检测到host7的路径，因为destinationFilter还是host7.

B.配置一组主机的路由

把主机0-2到主机6-8的路由路径都改成router0—router1—router2.实际上在xml里是设置到10.0.0.32的都走10.0.0.18.

1. **使用其他度量标准进行自动路由表配置**

默认情况下基于最短路径。还可以基于其他标准。

在XML文件的配置中，用<autoroute>元素指定自动静态路由表配置的参数。

它包括这些属性:

1. sourceHosts:用于选择应修改哪些主机的路由表。默认值为“ \*\*”。
2. destinationInterfaces: 用于选择要为其计算最短路径的目标接口。默认值为“ \*\*”
3. metric: 用于设置在计算最短路径时要使用的度量基准。默认值为“ hopCount”。其他的值:

hopCount：针对跳数，默认指标。

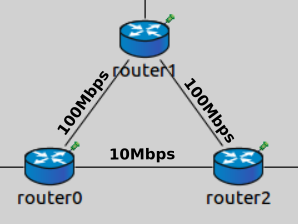
dataRate：首选具有更高带宽的连接。边缘成本与连接的数据速率成反比。

delay：首选降低延迟。边缘成本与连接延迟成正比。

errorRate：首选降低错误率。边缘成本与连接的错误率成正比。这**对于无线网络最有用**，因为有线连接的错误率通常可以忽略不计。

1. 数据速率作标准

注意把destinationfilter改成host1了。在xml文件里，添加<autoroute>属性，sourceHosts=”\*\*”(所有主机),metric=”dataRate”。需要注意，在ConfiguratorA.ned里，三个router之间的数据速率是不同的:所以不走router0-router2，改为router0-router1-router2.



最后的结果就是不管是来还是回的路线，都不会再经过router0—router2.(楼上手动配置路由只有去的时候不过router0-2，回来还是默认过的。)



最后那个掩码是192，即11|00 0000，考虑到最右边的子网掩码是10.0.0.32，即最后是00|10 0000，因此刚好能到。

1. 手动设置链路成本

把router0—router2的开销指定为无穷大，指示配置器在设置路由时不要使用router0和router2之间的链接。

<autoroute metric='hopCount'>

<link interfaces='\*.router0.eth2' cost='infinite'/>

</autoroute>

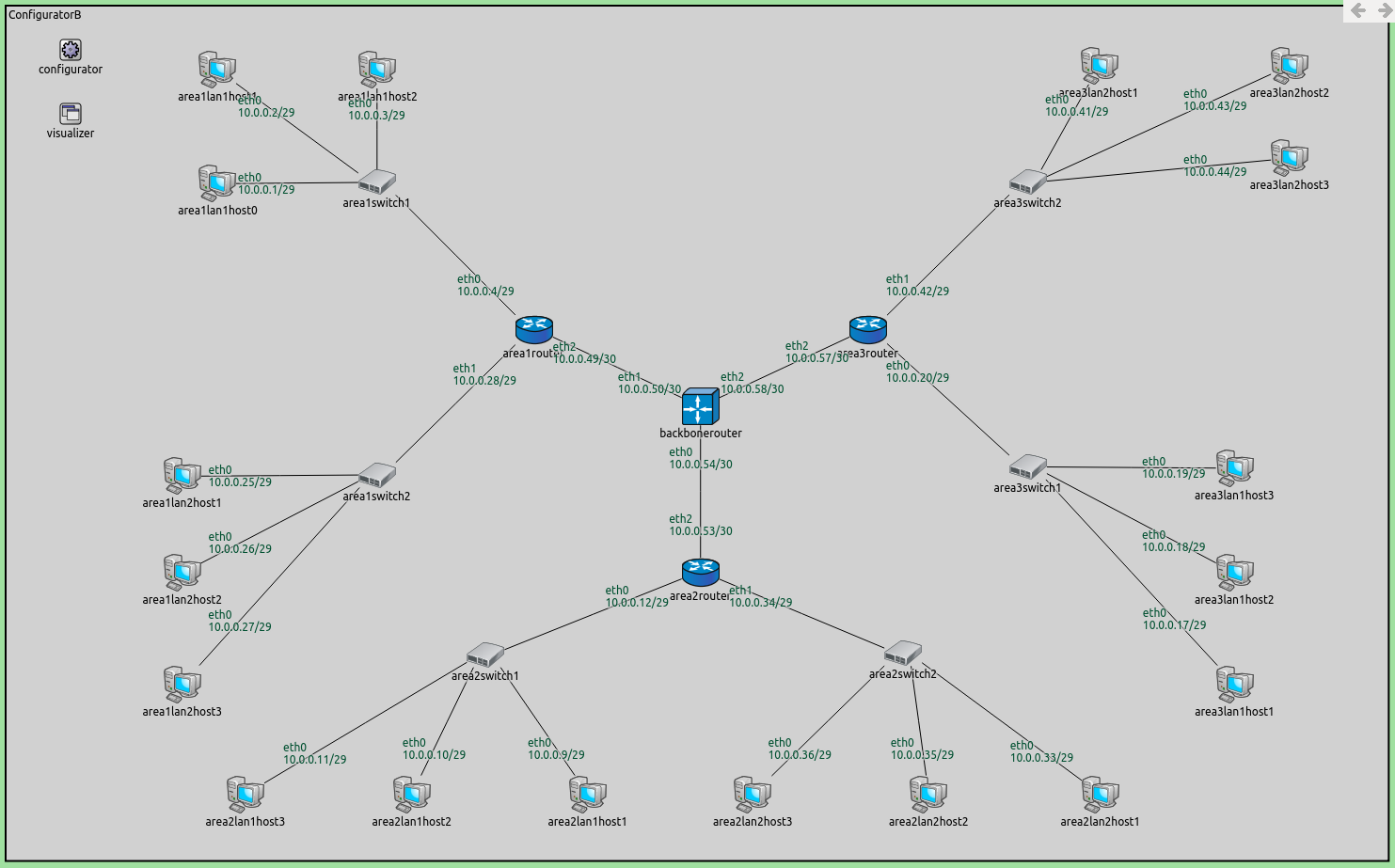
router0的eth2就是对着router2的。<link>这个位置还可以换成:<node>。当然里面还是要包括interfaces和cost这俩。

最后效果和A一样。

1. **配置层次化网络**

优化复杂的网络，在必要时候进行分层。

网络的结构:中心骨干路由器—3个区域路由器---每个区域中包括两个LAN(局域网)---每个LAN中包含3个主机。



1. 自动分配地址 不优化

最low的对照组，不分层，不优化，自动配置。



关闭了各种优化，最后节点将对每个目标接口都有一个单独的路由表条目。

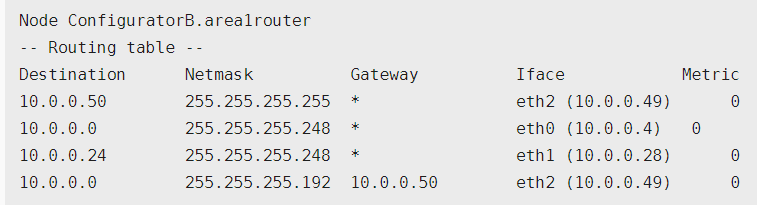
路由表超级多，每一个节点都要生成一个条目: **所有路由表条目都有255.255.255.255网络掩码，即到所有目标接口的单独路由**。网络中有30个接口（18个主机\* 1个接口+ 4个路由器\* 3个接口）。因此，对于其他29个接口，主机在其路由表中有29个条目。同样，路由器也有27个条目。

1. 自动分配地址 优化

仍然不进行分层，但是开启全部路由表的优化以减少表大小。

主机:一个指向自己局域网的，一个默认路由。

区域路由器:



①到骨干路由器,②③到达与它们连接的两个LAN的两条规则**(子网掩码多，限定范围的优先级高)**④通过骨干路由器到达网络其余部分的默认规则。注意虽然也有10.0.0.0，但其实表示一个默认的路由地址。

骨干路由器:

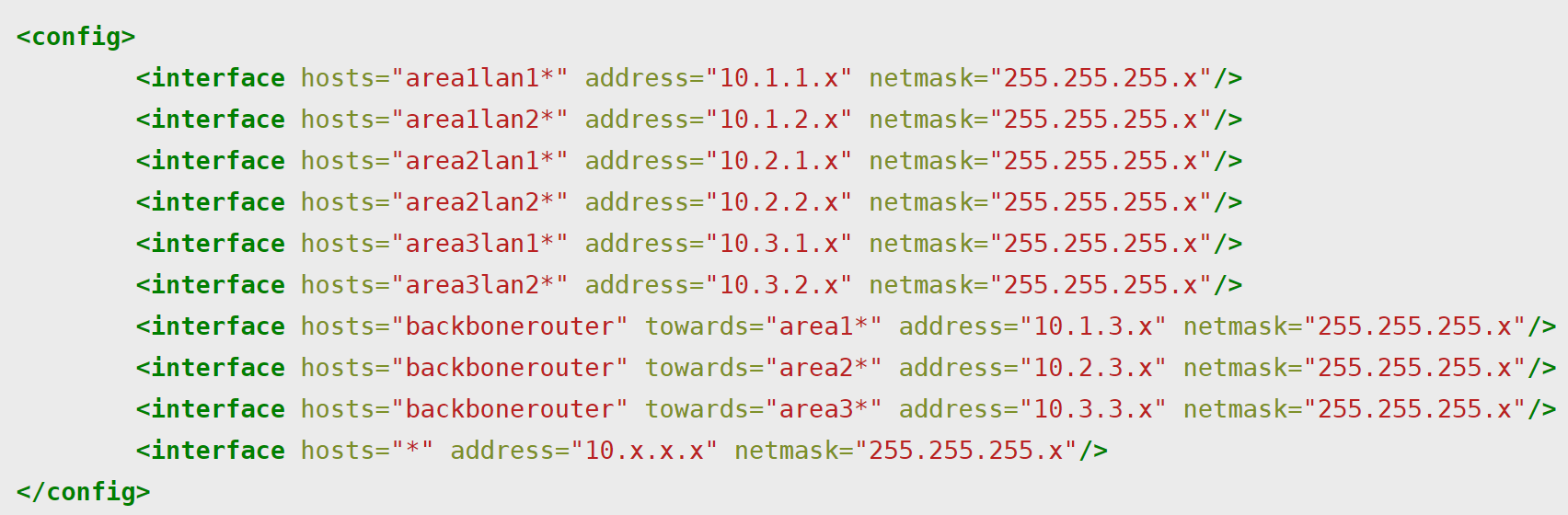
注意同一个区域路由器对应的两个LAN的IP地址并不连着(area2的LAN1是10.0.0.8,area2的LAN2是10.0.0.32)，因此不能把同一个区域作为同一条路由条目覆盖，还是得要俩条目(但若是自动分层就好了!)。

1. **自动分层分配地址 优化**

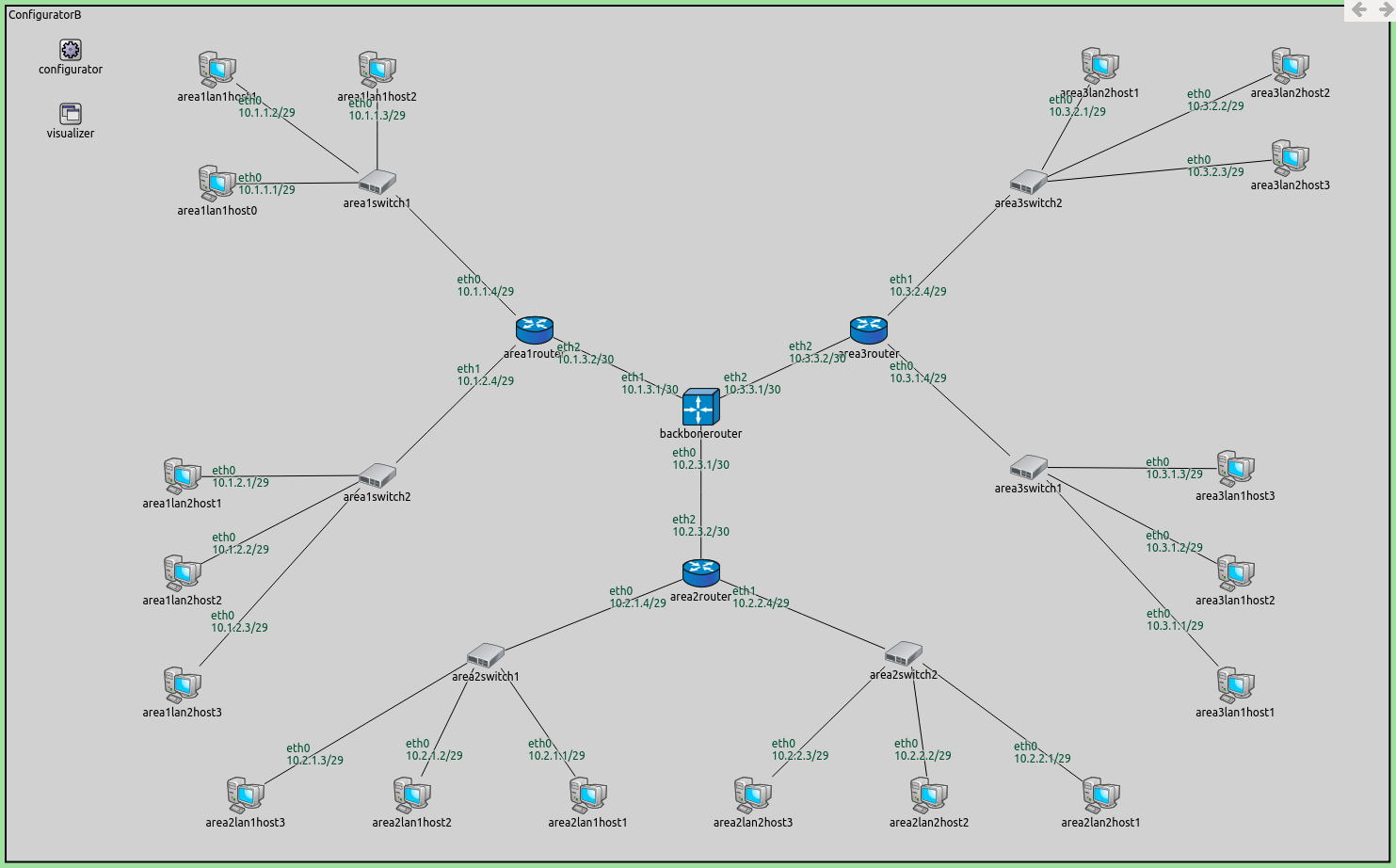
分层会进一步减少路由表的长度，因为在核心路由器的路由表中，只有一条来覆盖较大的一层网络。

每个主机的划分规则:以每8位为基础，所有的节点都以“10”开头。10.2.1.4表示area2,lan1,host4.

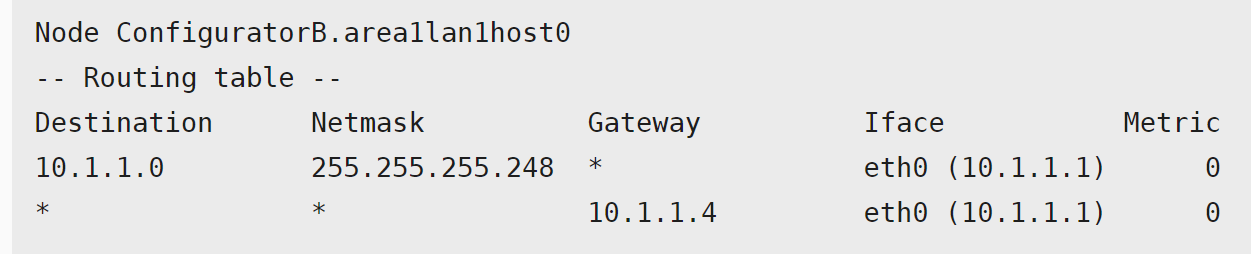
还将中间的backbonerouter到三个area的IP地址进行了手动划分(这样，**中间的骨干路由器每个区域只需要一条就行**)，但是各area的区域路由器地址是自动划分的(实际效果是**每个区域路由器对于每个LAN只需要一条**，比如area1lan1都是10.1.1.x):



结果如下:

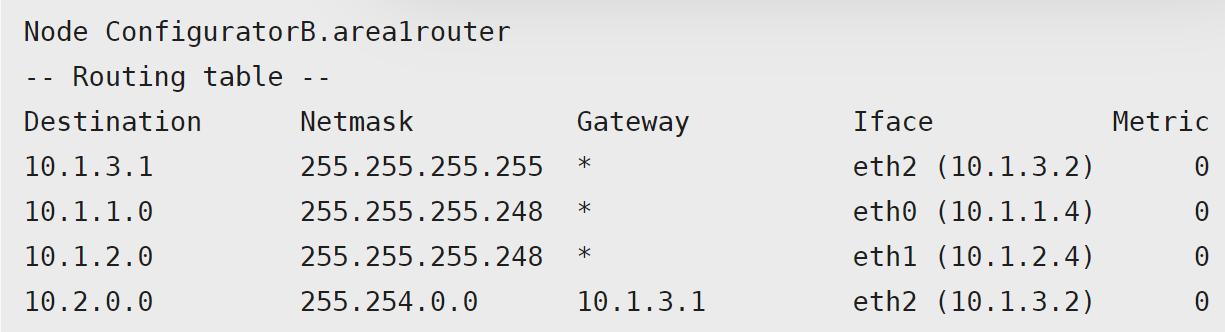


主机路由表:



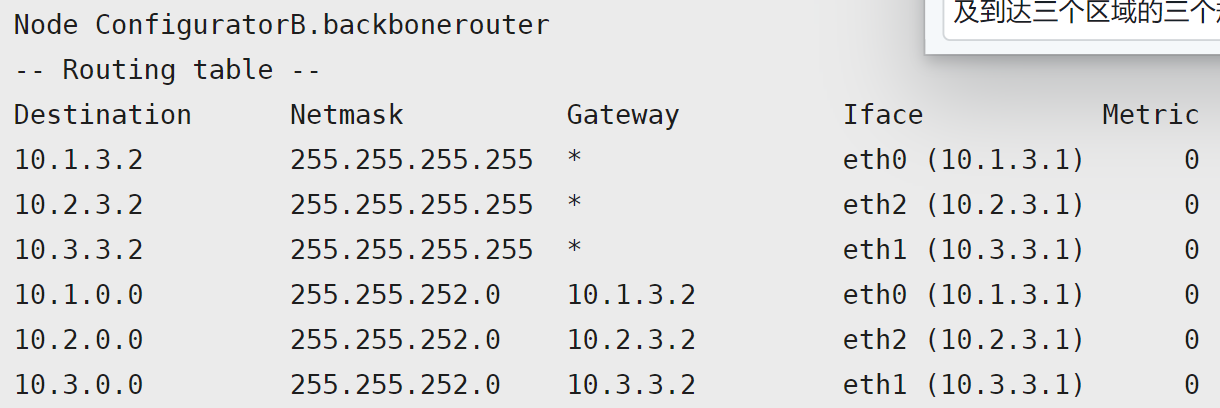
仅包含两个，一个指向自己的LAN，一个向外头(默认路由)。

区域路由器的路由表：



* 1. 访问骨干路由器，②和③访问自己区域的两个LAN，④通过骨干路由访问其他区域。

骨干路由器路由表：



①②③都是到达区域路由器，④⑤⑥到达新的区域。最大的贡献就是解决了自动划分时候同一个区域的俩LAN地址不连续，还得重新一一对应！

1. **配置混合有线/无线网络**

用ConfigurationC,比ConfigurationB多了两个无线局域网，分别在区域1和区域3。每个无线局域网包括:1个AccessPoint和两个WirelessHost。

1. 使用<wireless>元素确定无线网络的成员

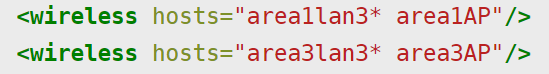
假设有相同SSID的节点在同一个无线网络中，<wireless>确定无线网络的成员，其中host和interface需要指定。

**在OMNeT++中，使用同一个RadioMedium(比如Ieee80211ScalarRadioMedium)模块的节点视为在相同无线网络中！**

考虑到节点默认的SSID相同，这导致area1lan3的host直接能连到area3lan3的host，显然错误。这俩无线LAN应该在不同无线网络中，即不同的RadioMedium。

设置displayRoutingTables显示路由表，还把destinationFilter设置为area3的lan3，限定目的范围。

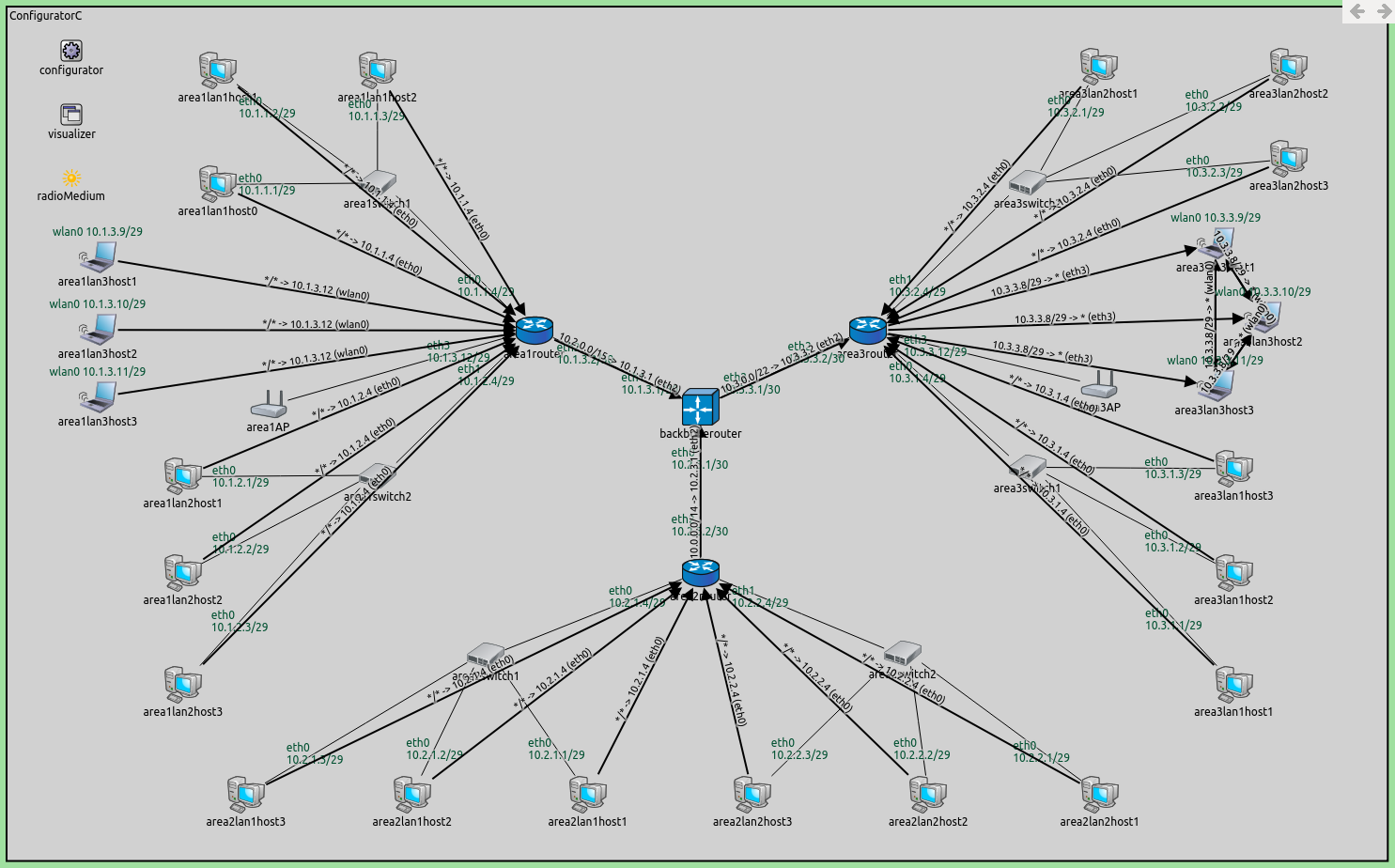
XML文件的设置规则还是划分子网，除此之外还新添加了:



**area1lan3的无线host和AP 与 area3lan3的无线host和AP 位于单独的无线网络中！**

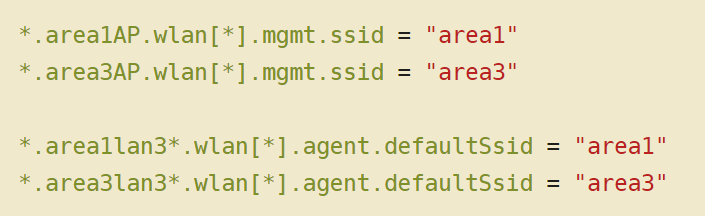
WirlessHost通过AP接入到路由器，而AP就像交换机一样是L2设备，没有自己的地址。在传输过程中，area1lan3Host2的消息会**首先广播到所有无线节点**(包括area3的)，但只有**自己通信范围内(只有area1)**的AP和节点能收到它，再由AP给router进行转发！

结果：



B.通过SSID确定无线网络的成员

直接设置两个无线LAN的SSID来划分，通过ini文件定义了。



由于ini定义了，因此xml文件不用再写<wireless>.

1. 保留部分网络未配置

考虑到有时候需要动态配置网络的某些部分，本例子不指定部分有线或无线的地址，用DHCP获取(动态路由协议，用于自动分配IP地址)。

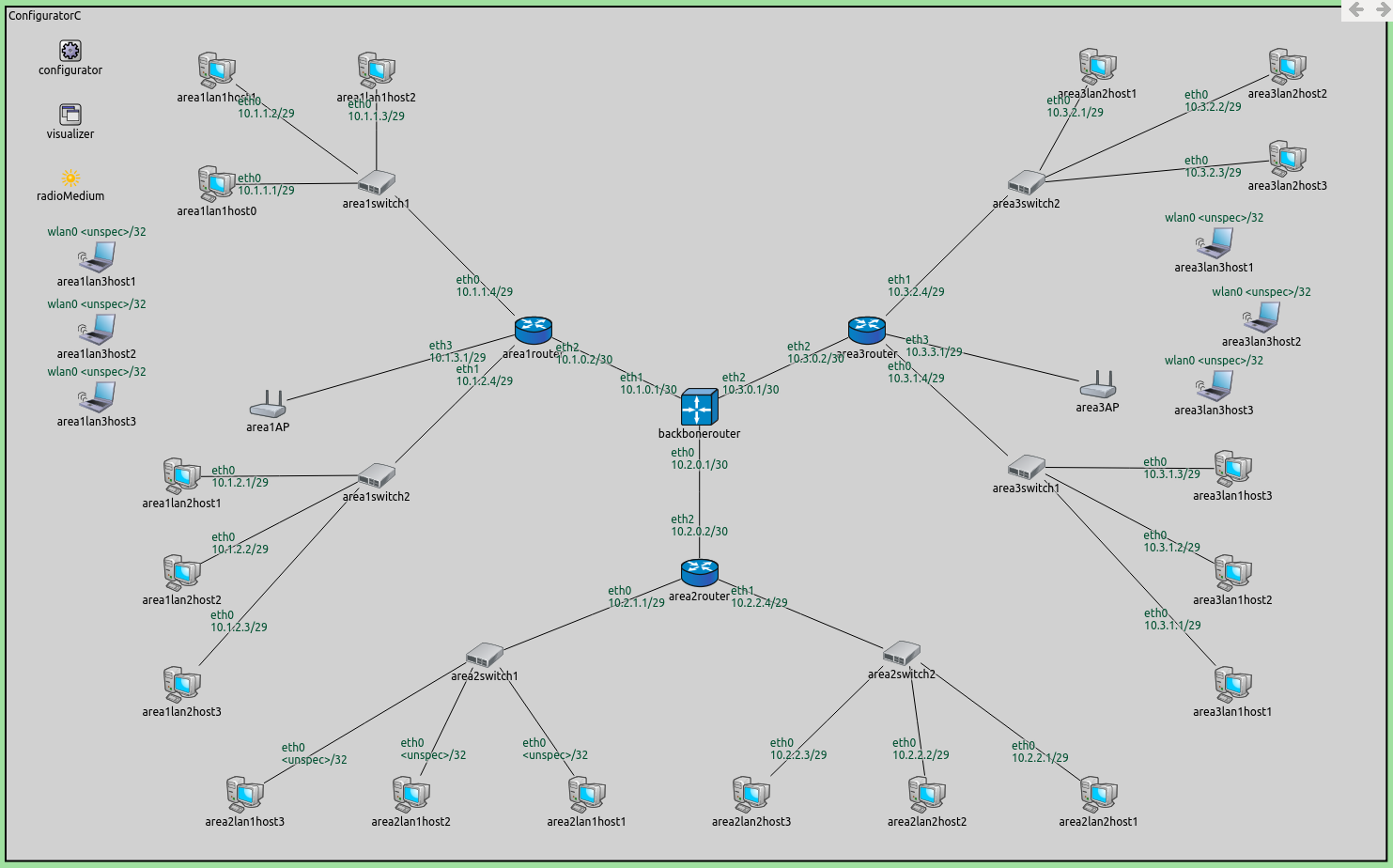
在区域路由器上设置为DHCPServer，注意指定接口”eth\_x”，由于area2没有无线的LAN所以就用了lan1，另外numReservedAddresses/leasetime啥的都是什么????

area1lan3/area3lan3的无线局域网以及area2lan1的普通局域网里的全部host设置为DHCP的主机，注意用于DHCP的叫"**numUdpApps**"(不是常规的ping的numPingApps)，因为DHCP在运输层用的就是UDP协议。这些LAN中的主机从相应区域路由器中的DHCP服务器获取地址。

XML文件:

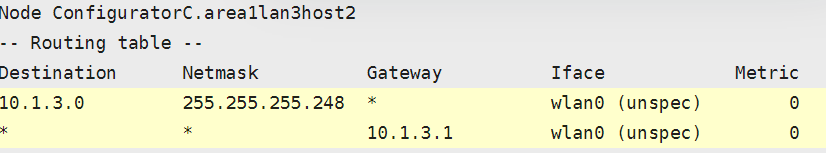


注意，area1lan3，area2lan1，area3lan3没有层次化划分地址，而是在后面，通过**指定各自的area\_x\_router的对应接口”eth\_x”的地址**，并辅以DHCP的Server和Client相对应从而间接分配了地址。



在初始运行时候，DHCP的Client并没有被分配网络地址，但是area1router这种与他们对应的eth3接口的地址已经有了(因为是xml文件里已经写清楚的)，就等着DHCP一对应就行了！

此外，尽管那些host未指定地址，但他们通往自己LAN的路由已经添加到了路由表。且他们还有一条默认路由。



对于区域路由器，它也有到达Area1lan3的路由以及到达Area2和Area3的默认路由。



骨干路由器没啥区别。

在实际仿真进行中，IP地址会逐步地、动态分配，会显示为”got ip xxx”。在此之后，才会进行一个正常的ping过程。

1. **完全无线网络配置**

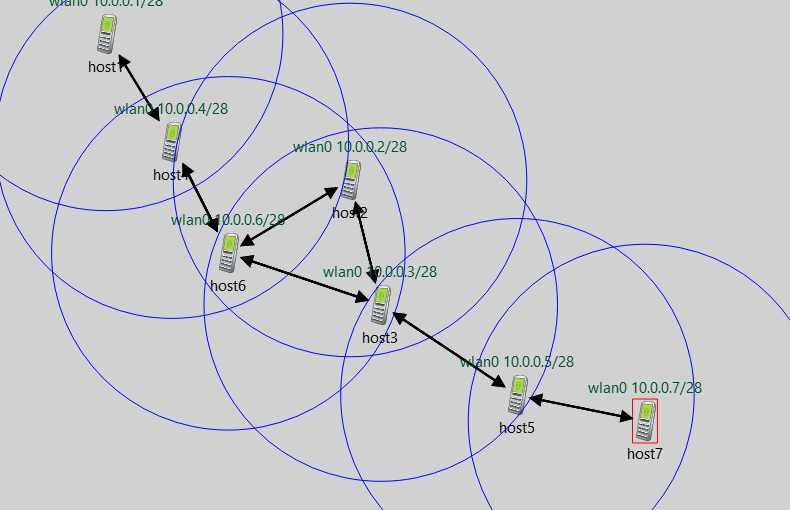
使用ConfiguratorD.ned，七个**AODV\_Routers**，继承了WirlessHost，属于无线节点的一种。包括了一个AODVRouting的子模块。

1. **使用错误率为指标配置的静态路由**

对于无线路由，用错误率为指标(因为无线网络很不稳定，适合用错误率为指标)进行静态路由配置。

先把**radio的transmitter.power设置为1mW**来降低各节点的通信范围，使他们仅位于链中相邻主机的范围内。此外，routingTableVisualizer能够显示所有节点的路由，但是关闭了箭头标签，箭头线偏移为0，简化可视化。通信范围之外的错误率可以看作是无限的，但是就算在通信范围内也不一定就没有错误率了。

在XML文件中设置了<autoroute metric=”errorRate”/>,用默认路由配置。



1. 为MANET准备的未配置的路由表

作为MANET，是动态性的，这导致拓扑结构可能会更改，因此需要**动态路由协议**。这个时候configurator仅用于配置地址，而**路由表配置是动态协议**。这个时候就是A去掉了黑色的箭头。

1. 使用AODV协议配置的路由表

使用Ad-hoc基于距离的动态路由协议AODV建立真正的动态拓扑结构！仍然**把addStaticRoutes设置为false**，不进行静态路由的分配(configurator仅配置IP地址)。由于**AODV是反应性路由协议**，因此需要**ping才能触发AODV协议来建立路由**。

静态路由表最初是空的，由于host1 ping host7，触发了AODV的路由发现过程，配置了路由。由于AODV是反应性协议，因此使用的路由会在一段时间后过期(就像host2，前期还有路径，后期路由表就空了，因为从host1—host7并不经过它！)。

1. 手动修改自动创建的配置

把完成的路由表配置等信息转存到文件中，在这个文件中再一次进行编辑就可以代替原始文件。

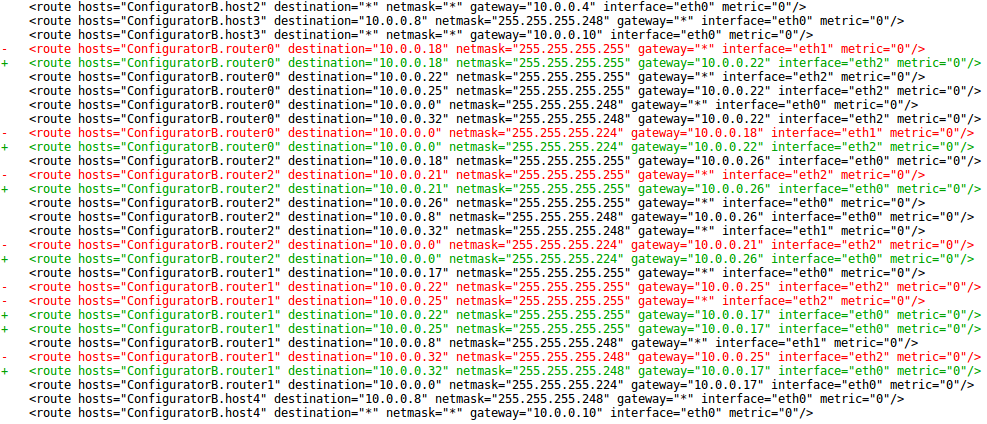
1. 转存完整的设置

转存的文件包括所有分配后的地址，路由表信息等，可以对其进行修改，作为新的xml配置文件重新导入。实际上就是configurator.dumpConfig()。

B.修改dump后的文件，再复用使用修改后的配置

编辑dump出来的文件用作xml配置，目标是数据包在三个路由器的三角形中逆时针传播，即每个路由器都应通过其右侧的接口转发三角形中的数据包。

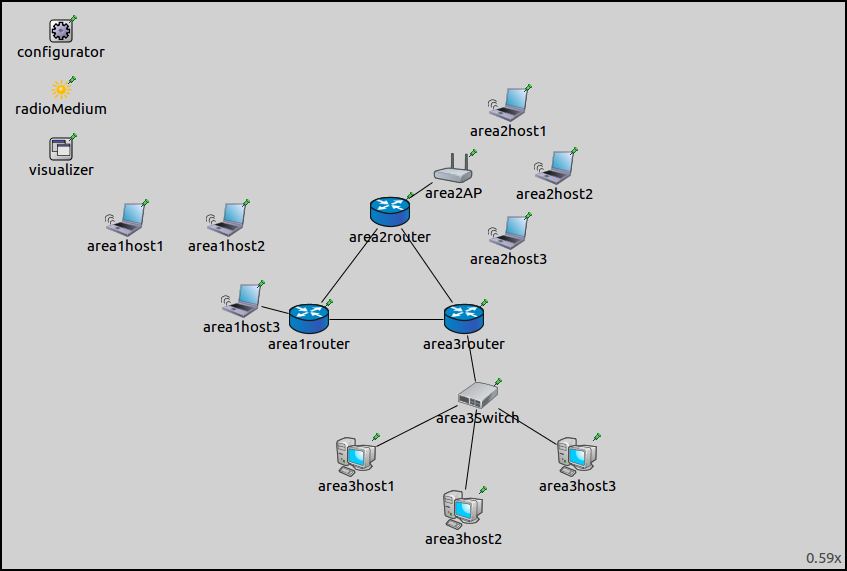
“step11b.xml”就是”step11a\_dump.xml”的一个复制，因为已经进行了全部路由的设置，所以可以设置addStaticRoutes=false。另外，“step11b.xml”对路由表进行了修改，以实现三个router的逆时针传输packet。



上图红色的是”step11a\_dump.xml”，绿色的是替换后的“step11b.xml”，可以看到改的时候，destionation和netmask不变，即指向的区域不变，但是gateway和interface变了，即接口对应关系变了。

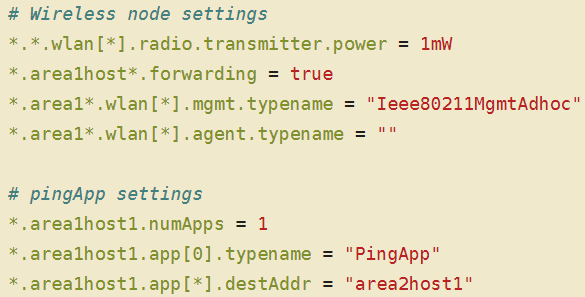
1. 混合不同种类的自动路由

对于有线/无线网络，分别使用跳数和错误率指标。



三个区域，由三个区域路由器为代表。Area1是三个WirelessHost，其中有一个是有线连接(**没有接入点，即一个Ad Hoc NetWork，**用错误率为度量标准);Area2是三个WirelessHost和一个AP，AP接入router;Area3是通过交换机连接的三个StandardHost。注意area1host1的通信范围与area1host3的通信范围没有交集，因此需要通过area1host2进行数据包转发，以错误率” errorRate”为度量标准。

关于area1以ad-hoc模式运作的ini部分代码:



设置radio功率以限定通信范围，forwarding默认就是True?????(可测试1)官网说是进行IP转发，**”Ieee80211MgmtAdhoc”是管理模块，代表着Ad-hoc管理模块**。另外，设置了area1host1到area2host1的ping。

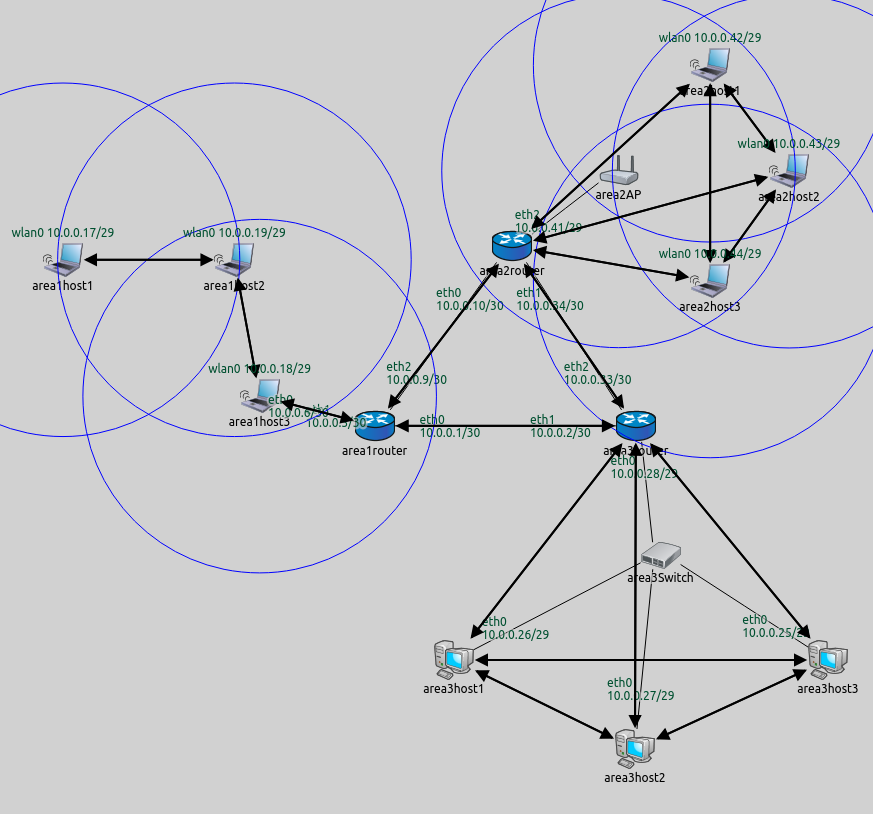
XML文件:



对于区域1，根据错误率进行指标配置，对于其他区域则根据hopCount(虽然是默认的，但是因为要区分给area1的metric，因此都写明更好阅读)。**对于默认的路由，假定链路上所有的节点都能到达该路由的网关，但是area1host1并不能直接通信到area1host3，因此区域1没有默认路由配置(这一点需要特别注意，基本上所有的Ad Hoc Network都有这个问题)**，

**注意，addStaticRoute是自动进行静态路由配置，如果用AODV这种可能就不需要，效果为路由表是空的；而addDefaultRoute才是这里我们提到的！**

结果：



对于area1host1，它给其他的出口都是10.0.0.19，但是:



这个是访问自己的LAN的所有主机。注意，这里其实只包括18和19，但是18的已经有其他的规则指定了，因此实际上这就是针对10.0.0.19的。注意因为area1host1不能直接到area1host3，所以无论来回一定都是过area1host2的！

**第三部分 Wireless**

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. **两个节点之间无线通信**

在ini文件里设置了\*.host\*.networkLayer.arpType = "GlobalARP"来简单地使各host知道对方MAC地址，进行通信。

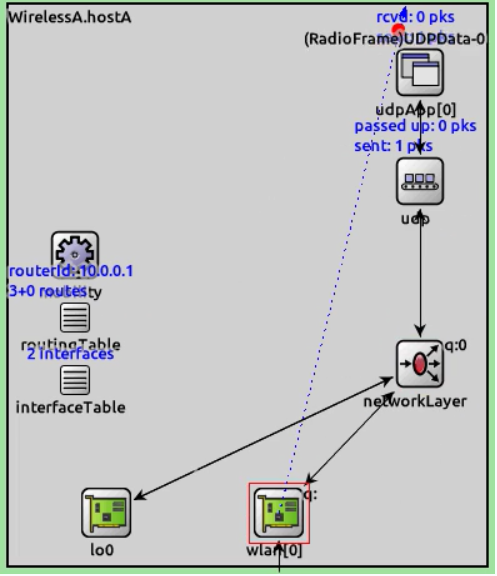
**关于物理层无线电，用最简单的IdealRadioMedium**，很理想，没考虑信号衰减之类的实际问题。因为物理层建模不是主要问题，所以越简单越好!

**关于MAC层(数据链路层)的网络接口卡，用简单的默认值。\*.host\*.wlan[0].typename = "IdealWirelessNic"，即网络接口卡**，与IdealRadioMedium对应的是IdealWireless，因为无线电是NIC的一部分(对应关系还有:无线电UnitDiskRadio——NIC:AckingWirelessInterface)

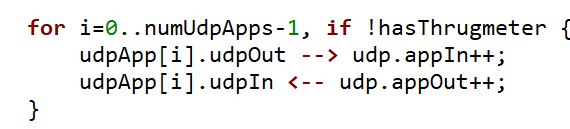
ini文件中，设置hostA作发射点，”UDPBasicApp”，hostB是接收方，但它”UDPSink”,一收到就给丢弃，啥也不干。之后以12ms为均值的指数分布发消息。

结果:

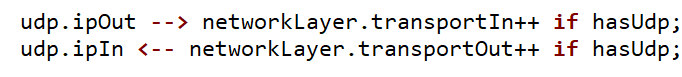
主机A的内部(StandardHost):



首先udpApp是产生udp数据包的，也就是”UDPBasicApp”,它和下面udp连接的代码在StandardHost.ned里:



其次是udp转给到网络层的networklayer，也是StandardHost.ned里定义的:



总之，节点A的udpApp产生了数据包，最终传出到wlan接口。

1. 设置可视化动画

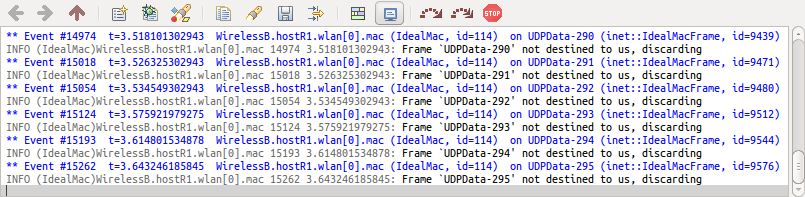
专注于物理层的无线电传输和信号传播来可视化。用IntegratedCanvasVisualizer来进行: **通信范围表示，信号传播和最近成功的物理层传输**。

相关的都写在最上面关于添加可视化模块的地方了，结果如图:



1. 添加更多节点，减小通信范围

增加3个节点，减小通信范围使主机之间无法相互直接访问，所以才需要进行routing转发等。在ini中减少通信范围为250m，因为hostA和hostB之间距离为400m所以无法直接通信。中继节点还没配置路由所以没法通信。注意这个时候因为是继承的之前的仿真，因此实际上HostA还在尝试与HostB的UDP报文传输，可惜无法到达(从日志文件R1丢弃就可以看出)!

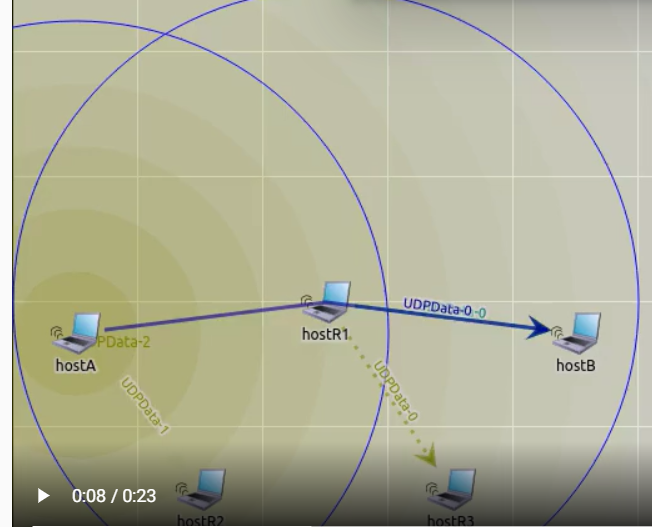


1. 用IPv4NetworkConfigurator设置静态路由

把中间节点充当路由器进行静态路由转发。Ipv4NetworkConfigurator对于无线网络仍然是针对错误率进行路由表建立的，在通信范围外的错误率是1，范围内的错误率很小。

可视化还是用IntegratedCanvasVisualizer，但是用其中的networkRouteVisualizer子模块了！能显示最近在两个终端主机的网络层之间发送数据包的路径(彩色箭头)。

如图，黄色的虚线是物理层广播传输，只不过因为数据链路层寻址没找到他们而丢弃部分；不太明显的青绿色箭头对应数据链路层的交换，最后蓝色箭头代表完整的网络层的路径。



注意:当有无法直接到达的两个节点需要通信时候，一定要写上\*.host\*.forwarding = true，这样才能中继转发进行路由!

我自己加了两个属性:  
##自己新加的,可视化接口地址

\*.visualizer.interfaceTableVisualizer.displayInterfaceTables = true

#注意忽略Switch和AP接入点，因为他们没有IP地址

\*.visualizer.interfaceTableVisualizer.nodeFilter = "not (\*switch\* or \*Switch\* or \*AP\*)"

\*.configurator.dumpConfig = "wireless4\_dump.xml" #方便看xml文件。

1. **信道之间的干扰**

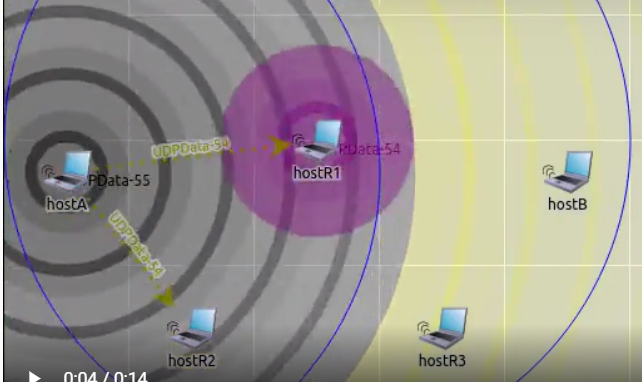
开启radio之间的干扰建模，以使到达同一接收器的信号产生冲突，接收失败。radio的干扰范围设置为500m，是通信范围的两倍。干扰范围即自身信号随着这个距离变弱，但是在该范围内能够干扰其他信号的正常传输。

设置如下:

\*.host\*.wlan[0].radio.receiver.ignoreInterference = false

\*.host\*.wlan[0].radio.transmitter.interferenceRange = 500m

虽然主机B无法直接得到主机A上的传输，但是主机A范围内的干扰能影响主机B上的其他传输。可视化关闭了数据链路层的箭头。



如图所示，UDP-54由R1继续往B发的同时，hostA又发了UDP-55，虽然A不能直接与B主机进行通信，但是由于这俩数据包的目的地址都是B，因此产生了冲突，即此过程结束后并没有hostA🡪hostB的网络层蓝色箭头出现。直到主机A达到发送间隔在等待，主机R1把最后一条到它这里的消息中继出去，此时hostA🡪hostB的连接才真正建立。注意，hostR1在发完UDP54之后是不会发UDP55的，因为它由于冲突压根接收不到UDP55，只能是等着A发的下一个UDP数据包！

干扰是一种正常现象，为了降低其损害，需要某种媒体访问协议来控制可以传输哪个主机以及何时传输。

1. **使用CSMA(MAC层)更好地利用介质，在有干扰地情况下提高信道利用率**

通过选择更适合无线通信的MAC层（medium access control）协议来提高通信信道的利用率。在上一个例子，节点发送数据包就是直接发送，CSMA则基于“传输前感知”（或“通话前聆听”）原理，在发送数据包之前先进行信道的监听。

CSMA（载波侦听多路访问）的节点在共享性的传输介质上进行传输之前，会先验证是否缺少其他流量。我们使用CSMA / CA（其中CA表示避免碰撞），在此协议中，首先要发送数据的节点等待通道变为空闲状态，然后再等待随机的退避周期。如果在退避周期结束后通道处于空闲状态，则该节点实际上可以开始传输。否则，该过程将重新开始，可能在退避期间内重新选择退避时间。我们期望使用CSMA可以提高吞吐量，因为将减少冲突，并且可以更好地利用介质。

对于无线网卡，需要**将”WirelessNic”替换”IdealWirelessNic”，它的radio和MAC模块都保持打开的状态，因此进一步设置radio模块”IdealRadio”,MAC模块”CsmaCaMac”。 CsmaCaMac模块通过可选的确认和重试机制来实现CSMA / CA**。它有许多可设置的参数，因此可以近似基本的**802.11b ad-hoc模式操作**。可设置参数如下”

acknowledgments 开/关

bit rate 比特率，用于数据和ACK帧

protocol overhead: 协议开销，包括MAC的标头长度和ACK帧长度

backoff parameters: 退避参数，包括最小/最大竞争窗口，时隙时间，最大重试次数

timing: 时序，除退避时隙（DIFS）外，在传输ACK帧（SIFS）和数据帧之前要等待的间隔

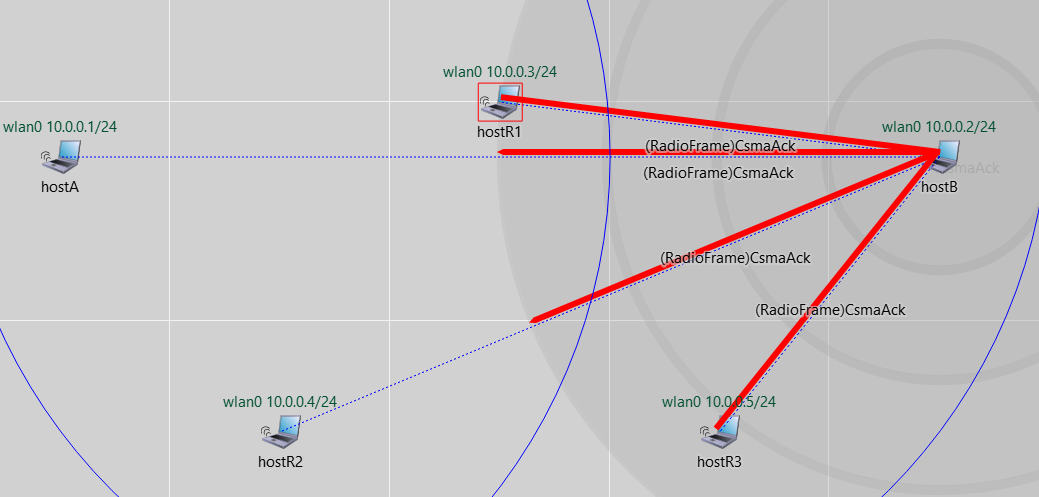
暂时把acknowledgments关了不确认，以看到“通话之前先监听”的效果(当然要随机等待退避时间)。

仿真里看，这个时候就好起来了！hostA发送的数据包知道等待一个退避周期了，这样中继到hostR1的数据包就能正确传递到hostB了。

1. **打开CSMA中的ack，来使链路层通信更可靠**

\*.host\*.wlan[0].mac.useAck = true

打开ACK后会导致：对于接收方而言，当它接收到寻找自己的MAC的数据帧时候，经过一个SIFS时间间隔会向发送方发ACK进行确认；对于发送方而言，如果它在适当时间内没有继续收到ACK则会重传。每次重传，竞争窗口(退避周期的最大值)加倍，直到达到最大值。重传次数过高则丢弃，初始化竞争窗口等参数，取出下一个数据帧(这个时候，MAC层会发出链路中断信号，这个信号可能会被网络层的AODV之类的路由协议解释为信号，表明路由状态需要进行变更)。



如图所示，其实A发给R1之后，R1也要ACK。

相比较没有ACK机制，有ACK会使得每一个数据包必须确认成功接收到，才能继续发送下一个数据包。宏观表现为:只有hostA的UDP1能完整发到hostB，hostA才会继续发送下一个UDP2，而不是连续UDP1，UDP2这样传输。

**ACK机制的优点:**

只要发过去的数据包，除非是重传次数过高导致丢弃，否则基本不会丢弃数据包。综合体现为数据包的序号连续。但是没有ACK则因为连续发送但是有些丢弃的数据包，从而导致接收方收到的数据包的需要可能是断断续续的。

**ACK机制的缺点:**  
 每次发送的完整过程都需要每跳进行确认，因此完整传输时间变慢，在提高信道可靠性的同时降低了吞吐量。

1. **能耗建模**

考虑电池的剩余电量之类的，在本例中使用组件扩展节点，以建模能耗。**忽略了能源约束，仅将无限能源安装到节点中(即一个完整的能耗模型，但是总能源无限)。**

在INET中，能耗模型由radio的energyConsumerType指定，在这个例子中，设置:

\*.host\*.wlan[0].radio.energyConsumerType = "StateBasedEpEnergyConsumer"

StateBasedEpEnergyConsumer基于无线电模式，发射器和接收器状态等状态对无线电功率消耗进行建模。无线电维护两个状态变量:接收状态和发送状态，要么处于接收要么处于发送状态，以接收状态为例，它可以是空闲/忙碌(这俩是指的信道状态)或接收。在接收状态时，子状态会存储正在接收的信号是哪一部分：前同步码，（物理层）标头，数据或其他。类似地，发送状态可以是空闲或正在发送，子状态存储正在发送信号的哪一部分（如果有）。

对于StateBasedEpEnergyConsumer，需要指定sleepPowerConsumption\receiverBusy

PowerConsumption\transmitterTransmittingPreamblePowerConsumption等参数。也就是一些特定阶段的能耗指标。

与能耗模型相对应的，还有储能组件，在这个例子中，设置:

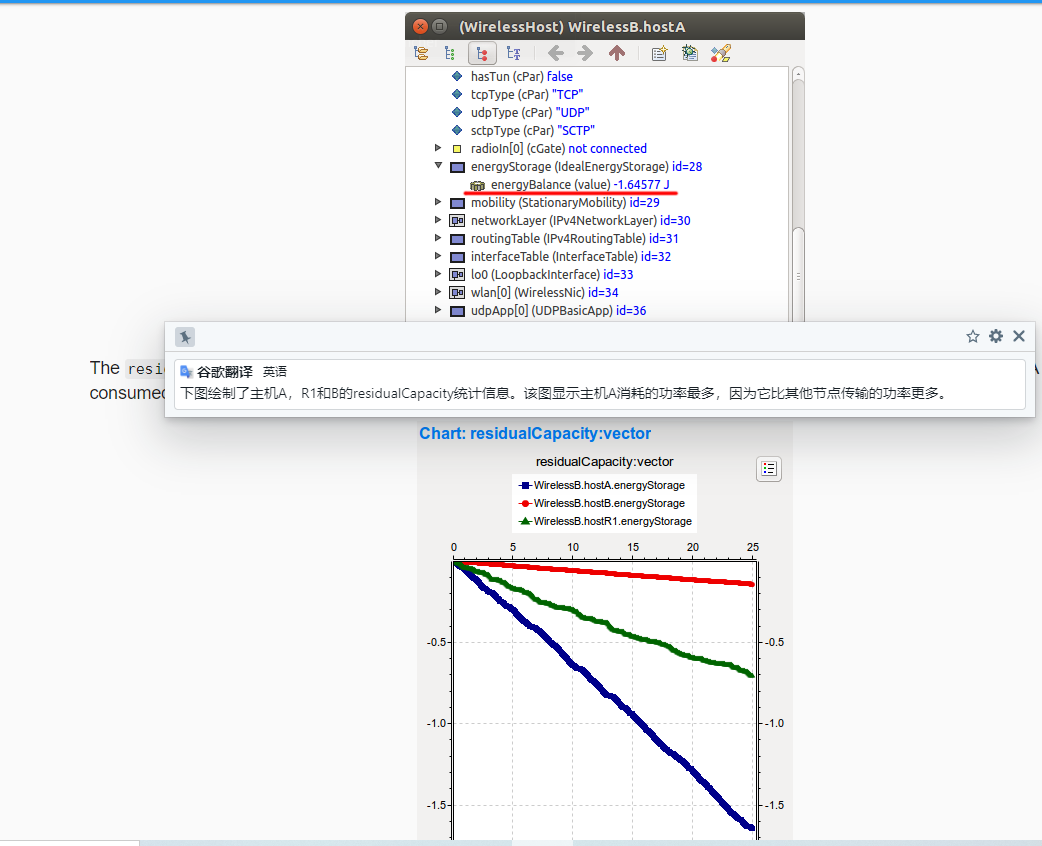
\*.host\*.energyStorageType = "IdealEpEnergyStorage"

它代表着无限量的能量，用之不竭。

对于能耗的一些统计信息，一来可以通过储能模块中的energyBalance监视变量来跟踪能耗(好像可以setWatch??)同样，也可以绘制残余的EnergyCapacity统计信息来获得能耗随时间的变化。需要**详细学习关于OMNeT++的统计信息如何使用！**

对于这一部分，我们可以很简单的添加能耗的统计，难的是进行总结和可视化!这些图怎么画我还不会！

关于下图中图1的查看方法:在仿真中右键点hostA，”open Details for”,点两个红点的那个上面排列方式按钮，就能找到。但是不知道为啥我的value是Nan！！



1. **配置节点的移动方式**

添加节点的移动性，使中间节点一直向北移动这样一段时间后，他们就移出了主机A和B的通信范围，破坏了通信路径。

移动性由节点的mobilityType来设置(在之前的configurator例子里也有，比如对car或pedestrian的)，目前已知的就是” VehicleMobility”/”MassMobility”/”LinearMobility”等。在这里用”LinearMobility”，沿直线走，给定一个初始角度和初始速度就行。

\*.hostR\*.mobilityType = "LinearMobility"

\*.hostR\*.mobility.speed = 12mps

\*.hostR\*.mobility.angle = 270deg

另外，L2层的队列长度一直是无限的，这样就不会因为拥塞而丢弃数据包。但其实拥塞一定是存在的！**在本例中，主机R1和主机A之间存在拥塞，因为主机A中的应用程序以比网络所能承载的速率更高的速率生成数据包。**

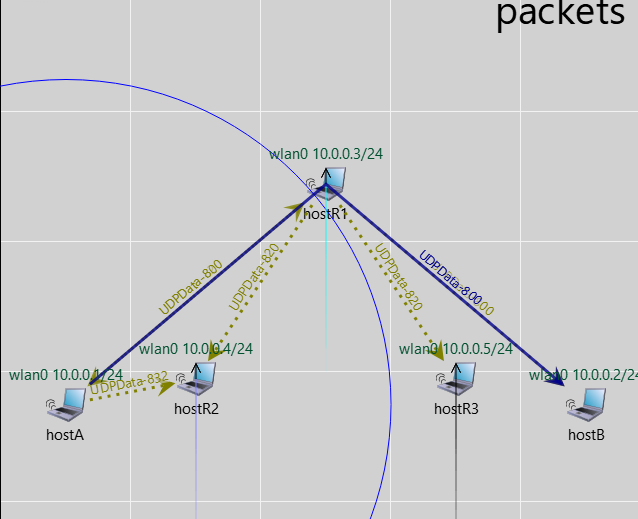
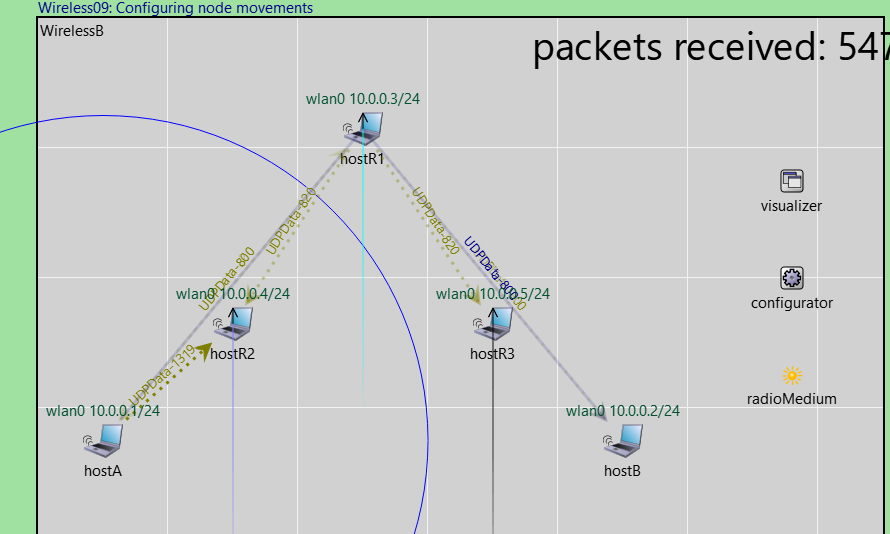
**因此，限制队列长度为10个数据包。**由于数据包丢失，我们期望主机B接收到的数据包的序列号将不再连续。设置如下:  
\*.host\*.wlan[0].mac.maxQueueSize = 10

还打开了关于速度矢量和移动轨迹的可视化。

\*.visualizer.mobilityVisualizer.displayVelocities = true

\*.visualizer.mobilityVisualizer.displayMovementTrails = true

仿真就是hostR\_x逐渐移动走，通信断开了(路由表是静态的，不会发生啥改变)。其实R2和R3还能用于通信，但是由于路由表静态所以不太行。

我们需要**根据网络的拓扑变化来实时配置路由，也就是AD-Hoc Routing Protocol！这个将在下次讲！**

1. **加入AODV动态路由！！！！！！**

AODV (ad hoc on-demand distance vector routing)是反应式路由协议，在本例中它能够适应不断变化的路由拓扑结构，在R1离开通信范围时候通过R2和R3继续中继数据包。

所谓”反应式”,就是它基于通信数据包的source-destionation进行路由表的动态调整，如果没有数据驱动它的路由表就不会更改。

因为AODV完全管理了路由表，因此初始**不需要添加静态路由**了！做法:关闭Configurator的自动分配功能:

\*.configurator.addStaticRoutes = false

为了使用AODV协议，必须将所有主机的hostType从“WirelessHost”(WirelessA.ned)中定义的类型，改为” AodvRouter”，这样每个主机就都是AODVRouter了:

\*.hostType = "AODVRouter"

别担心，其实AODVRouter跟WirelessHost没啥区别，是在其基础上延展了子模块。

AODV初始时候，网络处于静默状态，因此初始的IP地址如下:

当有UDP或其他数据连接请求时，源节点广播连接请求，其他AODV节点转发此消息，各自都记录下他们发现的，能听到这条消息的节点，从而形成大量返回到源节点的临时路由。当源节点接收到这些消息，还发现找到符合自己所需节点的路由时，它将从多个返回得到的路由路径中**挑选跳数最短**的一条，建立临时路由，向后发送消息到请求节点。注意，路由表中长久未使用的条目将在一段时间后回收。

考虑到AODV消息包括:路由请求（RREQ），路由应答（RREP）和路由错误（RERR）。希望通过可视化，在开始时（需要建立初始路由时）以及稍后由于节点的移动而改变网络拓扑时看到这些消息。



在本例子中，其实在一开始hostA到hostB的路径可以是:

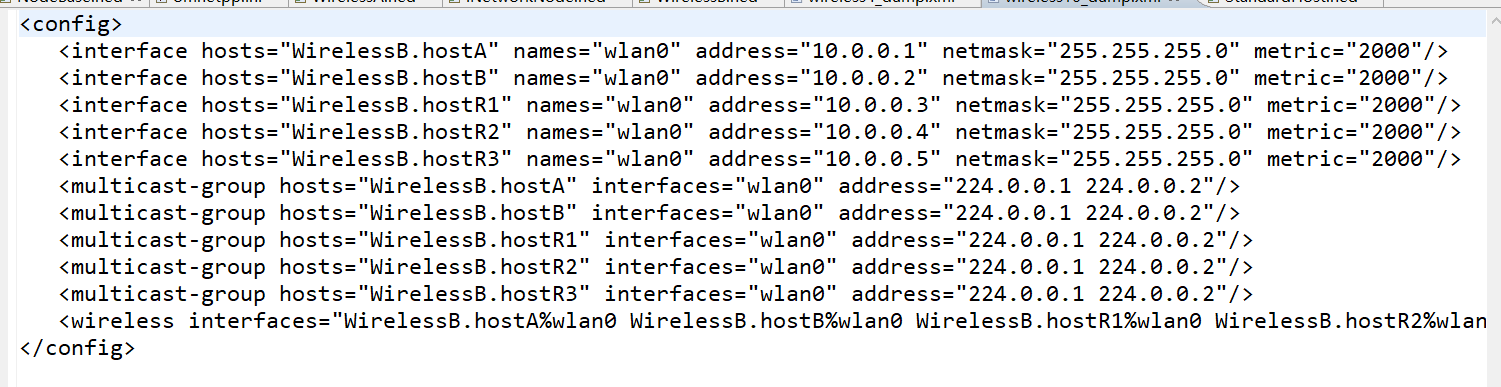
hostA—hostR1—hostB

hostA—hostR2—hostR3—hostB

……

AODV在广播，返回临时路由时候也发现了他们，但是一直基于**跳数最短**选临时路由，因此一开始是hostA—hostR1—hostB！

当hostR1出圈时候，hostA收不到它的ACK了，所以才会触发AODV的重新配置。



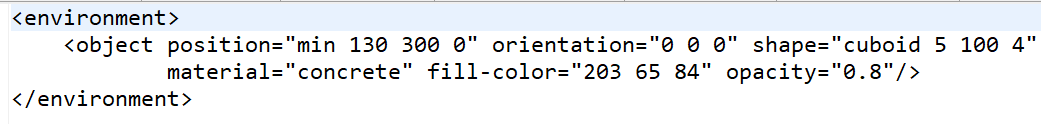
在dump出来的xml文件中可以看到:在初始化时候就确定了各个节点的IP地址，然而并不设置默认的路由表。也就是说，**在Ad-Hoc Network中，各个节点的IP地址是确定的，但是其路由表是动态的！**

1. **给环境添加障碍**

现实环境中，障碍物会吸收并反射无线电波，导致通信质量的下降。

在这个仿真中，在hostA和hostR1之间添加混凝土墙，这个时候障碍物会将无线电信号全部吸收，无法穿透。

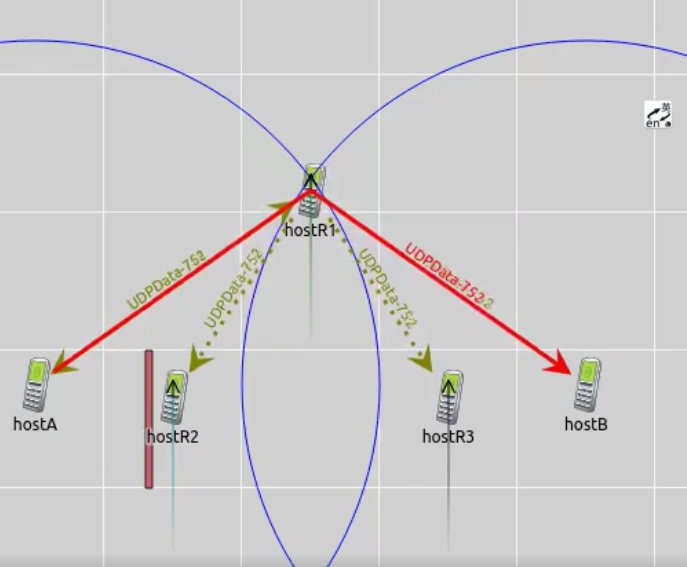
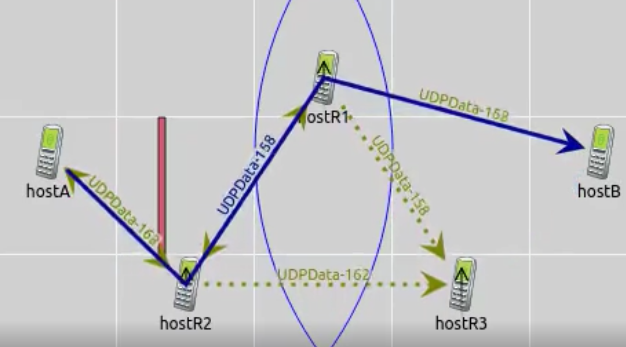
这个仿真用了WirelessC.ned，它在WirelessB.ned的基础上扩充了一个子模型:PhysicalEnvironment，因为障碍物是PhysicalEnvironment中的一个子模块。

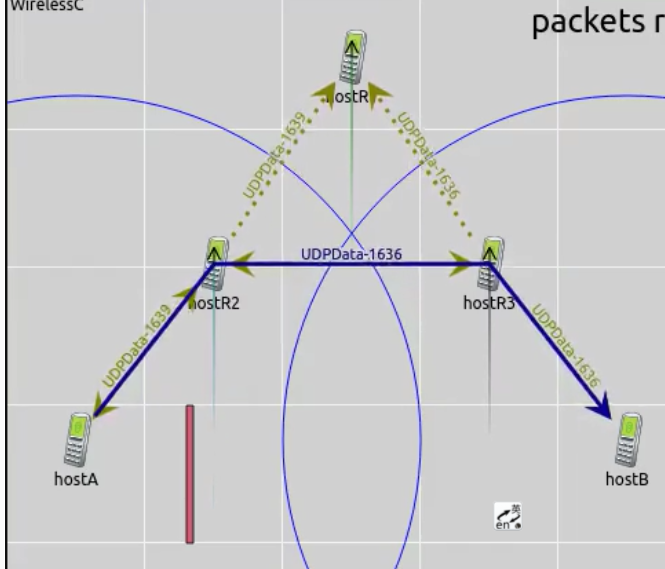
另外，在”walls.xml”描述了障碍物的各种参数，包括形状，位置，方向和材料定义等等。

注意”material”这里，本例用的是”混凝土”，还可以用concrete, brick, wood, glass, **forest**。除此之外，还需要把无线电模型和障碍物模型结合在一起。在这里设置为:  
\*.radioMedium.obstacleLossType = "IdealObstacleLoss"

所谓Ideal就是障碍物会完全地阻挡无线电信号，不能穿过去传播。

注意xml文件中定义了”cuboid 5 100 4”,即5\*100\*4的立方体，因此我们又把主机的高度定义为了1.7m，使墙完全挡住它。





上面是一些转发的路径图，因为AODV是反应式Ad Hoc协议，所以它的路由一直是随着移动性和障碍物的位置来变化的。

1. 更现实化的无线电模型

把一直继承的WirelessA.ned的【mediumType = default("IdealRadioMedium");】改成:

\*.mediumType = "APSKScalarRadioMedium"

与之对应的,把【host\*.wlan[0].radioType = "IdealRadio"】改成:

\*.host\*.wlan[0].radioType = "APSKScalarRadio"

APSK调制方案是更现实的无线电模型，默认情况下，它使用BPSK，但是也可以配置QPSK，QAM-16，QAM-64，QAM-256和其他几种调制。（调制是无线电发射器组件的参数。）

对于” APSKScalarRadioMedium”，它具有各种可扩展的属性，用于插入各种传播模型，路径损耗模型，障碍物损耗模型，模拟模型和背景噪声模型。在这个仿真，仅利用默认背景噪声模型为均质各向同性白噪声这一事实，并将噪声级别设置为非零值（-90dBm）。

在APSK中，我们需要定义无线电的载波频率，信号带宽和传输功率。注意在这些改变的时候，之前定义的CsmaCaMac这些定义完全不与他们冲突，包括ack机制仍然有效。

无线电接收器的物理参数:

A.灵敏度[dBm]: 若信号功率低于此阈值，则无法接收（接收器无法从信道繁忙状态变为接收状态）

B.能量检测阈值[dBm]：若接收功率低于此阈值，则检测不到信号，信道仍为空（对CSMA的“载波侦听”部分很重要）

C.SNIR阈值[dB]：如果SNIR低于此阈值，则接收不成功。

这些参数都在ini文件中进行了相关设定。

对于仿真结果，跟上一个没啥区别，但是实际却更真实了。

1. 更准确的路径损耗模型

默认的路径损耗模型是”free-space loss model”，即只考虑由于障碍物造成的反射或衍射。在例子中，由于主机是mobile，因此在地面上移动还应该考虑来自地面的二次反射(two-way ground reflection)。

在上面已经提到，APSK的radiomedium能够进行扩展，因此只需要把它的pathLossType设置为”TwoWayGroundReflection”(默认是FreeSpacePathLoss,还可以修改成RayleighFading，RicianFading，LogNormalShadowing等)。

\*.radioMedium.pathLoss.typename = "TwoRayGroundReflection"

TwoWayGroundReflection使用地面上方的发射器和接收器天线的高度作为输入。要计算海拔高度，我们需要主机的（x，y，z）位置以及这些点处的地面标高。在较早的步骤中，主机的z坐标已设置为1.7m。地面标高由地面模型定义，该模型是物理环境模型的一部分。

在此模型中，我们将FlatGround用于地面模型，并将其指定给physicalEnvironment模块。 （请注意，当我们引入障碍物时，我们之前已经在网络中添加了PhysicalEnvironment。）地面的标高是FlatGround的标高参数。我们将此参数设置为0m。

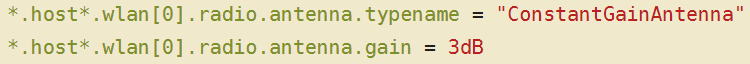
\*.physicalEnvironment.ground.typename = "FlatGround"

\*.physicalEnvironment.ground.elevation = 0m

关于结果，也是需要通过统计信息画图来产生的！还是得学习关于统计信息的可视化图方法。

1. 天线增益

进一步增强仿真的现实性，在之前一直假设无线电的各向同性天线的增益为1（0dB），但是在这里，更改天线增益来增强仿真。



将主机配置为ConstantGainAntenna,它建模了具有恒定增益的天线。例如，如果模拟无线网络的所有节点都在同一平面上，则ConstantGainAntenna可能对应于全向天线，例如垂直偶极子。（INET也包含对定向天线的支持。）

在仿真结果中可以看到，由于进行了天线的增益，传输足够强大，每次仅需要两次跃点即可到达主机B，而之前的步骤有时需要三个跃点。因此，在仿真开始时，主机R1可以直接到达主机B。此外，主机R1仅在模拟结束时才超出主机A的通信范围。发生这种情况时，主机A的传输将通过主机R2进行路由，这又只是两跳。

【在仿真工程中出现的问题】

1. ping的destAddr属性无法使用类似于循环之类的东西进行写入。只能手动逐个指定，且指定之前还需要把count和continue进行手动设置。期望解决办法是通过类似于xmldoc这样，单独生成destAddr的字段，再读取即可。即**普通host节点的簇内通信问题**！

【仿真工程总结】

1. 关于分簇问题:
   1. UAV节点和普通host节点的区别:

他们俩都是AODVRouter，但是UAV簇头节点能够进行消息的转发即forwarding，而普通的host节点不能。另外，UAV簇头节点通信范围广，移动性快，但是消息发送和接收的频率慢(和其他簇头进行通信);普通簇内节点通信范围小，移动性低，但是消息发送和接收的频率快(和本簇内节点进行通信。)

* 1. 后续的分簇工作还是要跟python交互 聚类算法，详细可以看[[可以添加的&待注意的]]的第二点。