Description for working with the Hierarchical Network.

A brief

The project is based on mobile AdHoc Network (MANET). We implement the available routing protocols for Hierarchical Network, as opposed to traditional flat networks. The main purpose of Hierarchical Network is to reduce the number of control messages, without affecting the packet delivery This will be a walk through as to how to implement Hierarchical Networks and where the protocol is changed. We are particularly implementing a hierarchical network running OLSR in the backbone. We compare the Network communication Overhead with the other implementations of Hierarchical Network like Silas DHT based solution and Flooding in the backbone network.

Pre Requisites

Knowledge of MANETs.
Understanding a protocol stack of Computer Networks .
Object Oriented Programming Concepts in c++.
Understanding Routing Protocols, especially OLSR and AODV.
Basic Understanding of Omnet++ as a simulator (modules, Network setup..etc).
It is advised to read the OLSR documentation for understand the implementation better.

The NED files.

The NED file specifies the network that we are working on. NED indicates Network Description.

Gateway.ned

We need to define how a gateway looks like. Because the default Adhoc host doesn't support multiple interfaces. We will have to define the extra interfaces in this.

Import inet.nodes.inet.AdhocHost: This is to include the functionality of an Adhoc Host and use it as a base to define the Gateway.

Import inet.networklayer.IManerRouting: This is for associating the extra interface with a routing protocol.

Parameters say about the string backboneRoutingProtocol and the display of the module and the image associated.

Submodules include backbonerouting that is an instance of Routing Protocol.

Since we got another routing protocol, we need to connect that to the Network Layer of the Adhoc Host, to make it a Gateway, which is realised using

backbonemanetrouting.to_ip --> networkLayer.transportIn++;
backbonemanetrouting.from_ip <-- networkLayer.transportOut++;

OLSRRouting.ned

import inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator: This is used to assign ipv4 addresses to all the nodes available.

import inet.nodes.inet.AdhocHost: This is the AdHocHost for nodes to be used in implementing the network.

import inet.world.radio.ChannelControl: Channel Control enables us to assign different channels to different sub networks.

import p2psipmanet.simulations.HierarchicalNetwork.gateway: This imports the Gateway that is defined in Gateway.ned

The name of the network is OLSR, which has variables hosts, gw.
The submodules include ChannelControl, IPv4NetworkConfigurator, gateways defined as an array of gateways (gw number of gateways), hosts defined as an array of host (hosts number of host).

Now that the Network is given with the ingredients required to constitute a network and simulate, we define the network and define the network further and add parameters to make the network further complete.

HierarchicalNetwork.ini:
Ini files are initializations of network and the different parameters that are associated with the network.
[Config]: Describes the configuration
Record-eventlog : Records the event log based on boolean supplied
Sim-time-limit: Simulation Time limit;
description: has the description of the configuration
***.constraintAreaMin&*(& = 'X' or 'Y' or 'Z'): This defines the area constraints of simulation
***.mobilityType*: This is the mobility type of all the nodes. Different mobility modules can be defined.
***.wlan[*].radio.maxDistance*: This is the maximum transmission range of each of the radio.
***.wlan[*].mac.maxQueueSize*: deines the number of messages that can be in the waiting queue.

***.numUdpApps = 1*
***.udpApp[*].typename = "UDPBasicApp"*
***.udpApp[*].startTime=100s*
***.udpApp[*].stopTime=500s*
***.udpApp[*].sendInterval=1s*
***.udpApp[*].messageLength=1024B*
***.udpApp[*].destPort=1000*
***.udpApp[*].localPort=1000*
The above snippet of code says that each node has a UDPBasicApp which starts at 100s and stops at 500s and sends packets at a regular interval of 1s with each message having a size of 1024Bytes and the UDPApp runs on port 1000. The following say about node specific UDPApp.

***.scalar-recording:* Defines a boolean for scalar recording of results of simulation
***.vector-recording*: Defines a boolean for vector recording of results of simulation
***.printOLSRmessages:* Defines a boolean for printing OLSR messages.

***.gw[*].manetrouting.announcedNetworks = "0.0.0.0/0.0.0.0":* This is for announcing the routes to the other interfaces

***.configurator.dumpAddresses = true*
***.configurator.dumpTopology = true*
***.configurator.dumpRoutes = true*
These define that the routes and topology and addresses be printed after simulation

***.configurator.addStaticRoutes:* Define a boolean for adding static routes that are defined in the ini file instead of calculating the routing table.

***.configurator.addDefaultRoutes:* This define about the default routes
***.configurator.addSubnetRoutes:* This define about the adding subnet routes

**.*.routing.typename="OLSR"*
***.routingProtocol = "OLSR"*
***.manetrouting.excludedInterfaces = "wlan1"*
***.manetrouting.interfaces = "wlan0"*
***.manetrouting.interface = "wlan0"*
The above mention the routing protocol associated with manetrouting and the interfaces it can access and the interfaces it cannot access so that there will be distinction between the backbonerouting and manetrouting.

That is followed by the BackboneRouting which gives details about the backbone routing protocol and the associated interfaces and the interfaces it is forbidden to access.

***.wlan[*].bitrate:* Tells about the bitrate at which each interface can operate.
***.isFlooding:* boolean describing whether or not flooding

**.hosts and **.gw are the variables for number of hosts and number of gateways in the network.

***.gw[^].*.initialX = xm*
***.gw[^].*.initialY = ym*
This section tell us about the initial position of the node/gateway. This positions tell us whether the other node is reachable or not.

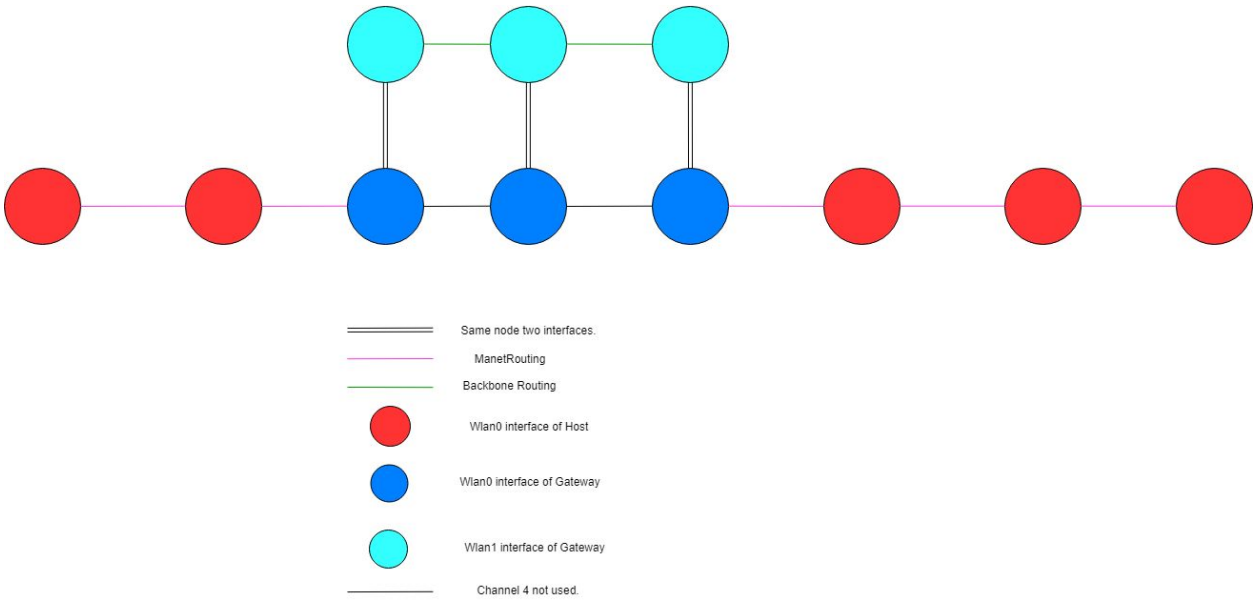
```
*.configurator.config = xml( \
"<config>\
                <interface hosts='gw[0] host[0] host[1]' names='wlan0' address='10.0.1.x'
netmask='255.255.255.0'/>\
                <interface hosts='gw[2] host[2] host[3] host[4]' names='wlan0' address='10.0.2.x'
netmask='255.255.255.0'/>\
                <interface hosts='gw[0] gw[1] gw[2]' names='wlan1' address='10.0.0.x'
netmask='255.255.255.0'/>\
</config>")
```

The above define the configuration for Network configurator so that the ipv4 addresses can be assigned accordingly. It also mentions about which interfaces are connected.

**.gw[*].numRadios = 2*
**.host[*].numRadios = 1*
This defines the number of radios for each of host and gateway. Gateway needs two radios, one for each interface.

***.channelControl.numChannels = 5*: Tells number of channels required
***.gw[0].wlan[0].radio.channelNumber = 0:* Tells about the channel number so that other nodes, even though in range would not communicate.

After defining the network the network looks like:



Each single solid line indicates that the nodes are reachable from each other. The black solid line is a redundant channel, hence given a higher number and never used.

THE OLSR PROTOCOL

The OLSR protocol needs to change significantly to accommodate the requirements of a hierarchical network. The messages like MID and HNA are not implemented in a typical OLSR scenario. We need to add them so that multiple interfaces are supported.

The following description explains about the OLSR as a protocol. This takes RFC3626 as the reference. This tries to address each of important details required to understand OLSR as a routing protocol.

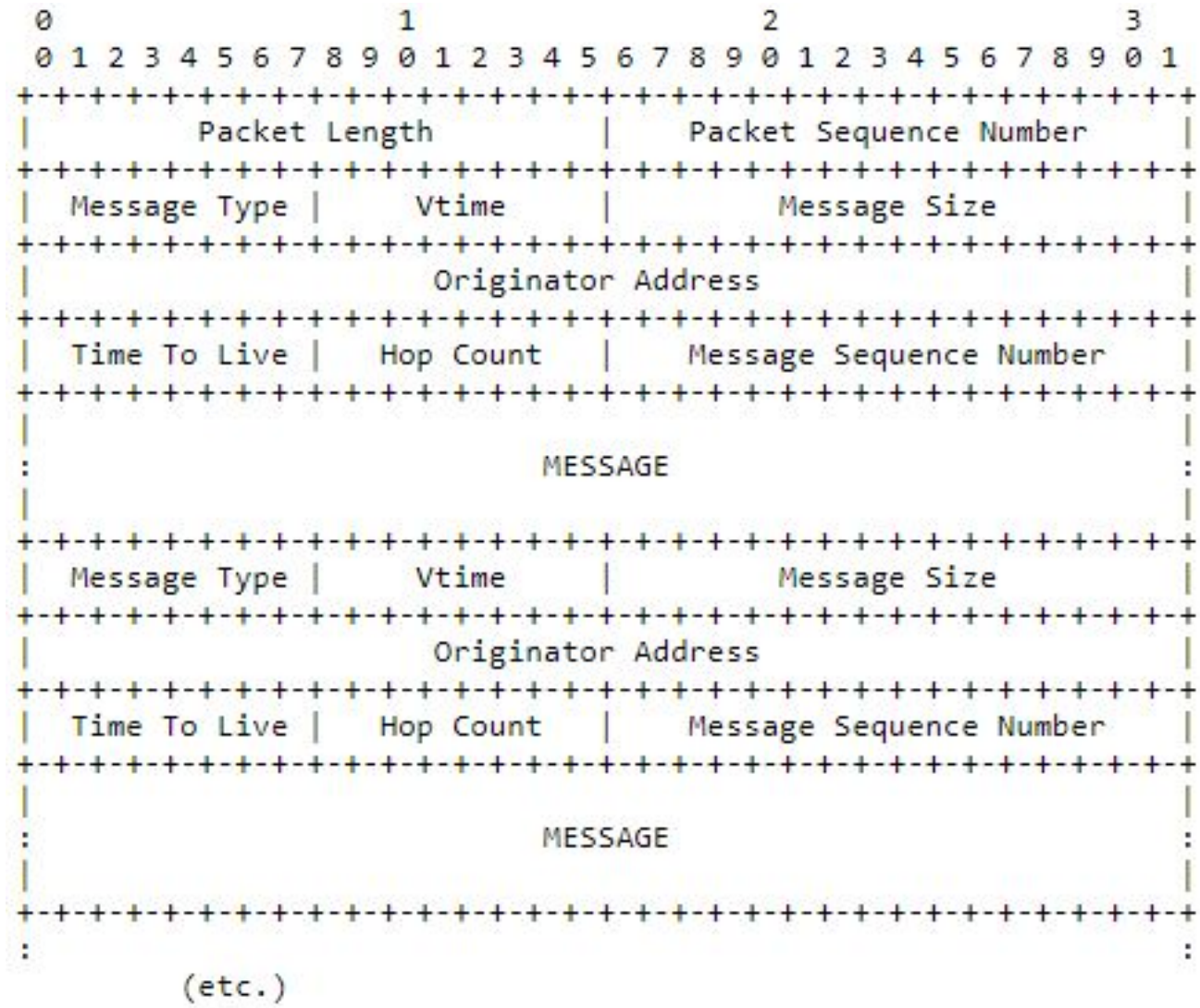
Section 1 and 2 discuss about the basics of OLSR protocol. These sections introduce OLSR and also mention about MPR, which is the main reason OLSR is optimised. The optimizations from the tradition LSR is in the MPR.

Section 3:

3.1 The port number is mentioned in OLSR.cc (RT_PORT)

3.2 Defines about main address of each of node

3.3 The packet format



This packet is defined in OLSRpkt.msg. The skeletal structure being *packet OLSR_pkt{ }* and the definition is extended and all the complexities are defined in OLSR_msg which consists of very wide range of messages like the Hello, MID, HNA, TC and each of the message format is defined in this file .
TC message: *OLSR_tc*
MID message: *OLSR_mid*
Hello message: *OLSR_hello*
HNA message: *OLSR_hna*

3.4:
Packet processing and Message Flooding.
This is addressed by handle_message() , recv_olsr() in OLSR.cc

Default flooding algorithm is mentioned in OLSr.cc forward_default()
The jitter is added in every timer with the same name.