# Detailed Design

## Introduction

This is a code level description of the major components of the system.

## Monitor

### Execution

Execution starts in `MainWindow::MainWindow()`. A timer is initialized in this method by calling `connect`. The timer calls `MainWindow::updateList()` every `timeout()` seconds. `updateList()`...

1. polls the `Model` by calling `comm->get_all_nodes()`
2. updates the GUI tables by calling `setNodeInAllModel()`
3. builds a renderable list of node by calling `widget->updateGraph()`
4. and renders the nodes by calling `widget->show()`

Note: The MKCommModel is constructed in `MainWindow::MainWindow()`.

### Behavior Reference

| To change | Edit file(s)[1] |
|---|---|
| Graph appearance (colors, shapes, etc...) | node.*<br>edge.* |
| Graph physics behavior | point.* |
| Graph Rendering | graphwidget.* |
| GUI options and settings | settingsdialoug.* |
| GUI tables | mainwindow.* |

[1]These files are relative to MeshKit/src/gui/Monitor/

## Comm Model

The `MKCommModel` stores its model in the data structure `nodes` in MKCommModel.h. The interface to the Monitor is also defined in MKCommModel.h. When an

MKCommModel is constructed, it creates the HeartBeatListener thread and an MKControl thread. These threads listen for packets sent to the OCU from the MESH.

When the HeartBeatListener thread receives a packet, it calls MKCommModel::update(HeartBeatPacket). This method adds the sender of the packet to the model if the model does not contain the sender. Next, it clears the sender's list of neighbors and adds each neighbor that is listed in the packet.

When the MKControl thread receives a packet, it calls MKCommModel::update (mongo:BSONObj, MKRequest). This method handles latency requests and route request. For a latency request, it finds the the node that was requested to ping the other node and sets the latency for the request. For a route request, it finds the node that was requested to traceroute the other node and sets the route for the request.

# Listener

The listener is composed of two threads: HeartBeatListener and MKControl. HeartBeatListener listens for heartbeat packets, and MKControl listens for command and control (C&C) packets. When the HeartBeatListener thread receives a packet, it calls MKCommModel::update(HeartBeatPacket), and when the MKControl thread receives a packet, it calls MKCommModel::update(mongo:BSONObj, MKRequest).

# Heartbeat

The heartbeat script is very similar to the original one that you (Tim) gave us, so it will not be covered in depth here. This script unicasts to the target ip address as defined by the global variable OCU_IP.

# Command and Control

CCResponse.py starts in main by creating and running a CNCListenerThread thread. This thread listens for C&C request packets and, upon receiving one, calls parseCNC. ParseCNC verifies that the packet received is indeed a C&C request packet and, if so, creates a CNCResponderThread thread to handle the response. This thread determines which kind of request was sent (either a "ping" or a "traceroute" request) and responds accordingly. (This script unicasts to the target ip address as defined by the global variable OCU_IP.)