MONDAY, TUESDAY, WEDNESDAY

REU:

- Gave Interim progress report
  - Project is on track
  - No major issues persist

Implementation:

- Created a third application -- Network OS.
  - The Network OS application runs as a background service. It sits above the wifi/ad-hoc service, and communicates directly with both the control application, and third party applications.
  - Responsible for retrieving settings and configurations from the control application and sending them to the ad-hoc service.
  - With the help of the ad-hoc service, maintains the network map
  - Manage multiple mid-level abstractions and responsibilities (see below)
- Spent 3 days moving code from the ad-hoc application and control application to the network OS application..
  - Moved all AIDL communication from the control app to the network OS app
  - Added a new AIDL interface, and implementing code, to the control app, which will communicate with the network OS service instead
  - Implemented the network-os side of the new AIDL interface (most of which just call similar AIDL methods on the ad-hoc service, caching relevant data).
  - Implemented higher level ContactManager, MapManager, and ConfigurationManager in the Network OS (granted, much of their functionality is just to pass commands down to the ad-hoc network).
  - Debugged, etc. By Wednesday afternoon, the applications were functioning just as they were Monday (fully functional), but now with an additional abstraction layer.
    - Was this a waste of time? Nope. Because this new layer is much more apt to do a number of things than either the ad-hoc service or the control application would be, such as:
      - Communicating with third party applications
      - Load balancing
      - Managing ‘OpenFlow-like’ network slices (logical subnetworks)
      - Collecting real time network traffic info
      - Saving and managing network state information
      - Resolving network address collisions

THURSDAY

Planning:

- Talked to Paul about project progress and future plans
  - We discussed the basic structure of the software (The three core applications, third party applications, communications between all 4 application types, and between devices)
  - Agreed on the following structure, at least for the time being:
    - Control application -> Network OS via NetworkOsAidlInterface
    - NetworkOs -> Ad-Hoc Service via AdHocServiceAidlInterface
    - Third party applications <-> Network OS via broadcast intents (?)
  - It is unclear how to communicate with third party applications (messengers, file senders, photo share)...Intents, more than likely. Certainly not AIDL, as is currently used between the control and network OS, and between the network OS and ad-hoc service. What kind of communication would be needed?

        - According to Paul, the third party applications would only need to know the next hop...I am not sure how to implement such a third party application. My original plan was to just broadcast the intent type (perhaps as generic as “AD_HOC_SEND_BYTES”), and a byte array in extras that would be passed up to the same third party application (on the foreign device, via wifi packets) for interpretation there. However, I suppose this is too high-level. If we are supposed to be able to open sockets in the third party applications, then I have no how to approach that. Will need to discuss with Paul when this part of the project comes around.
        - Third party access to, I guess Sockets (if that is what he means) would be more powerful and useful, but less simple for third party developers. Then again, it may be the only way to handle file transfers reasonably. Maybe both could be implemented? Will be seeking clarification on this soon.
    - Discussed implementing network slicing -- creating virtual subnetworks.
        - Different applications would each run on their own virtual subnetwork
        - These virtual subnetworks would run on the real network, but have limited visibility, and make it easy to find peers with similar apps available.

Implementation:

- Began to implement slicing:
    - Created the LogicalNetworkManager and defined the LogicalNetwork class (in the Network OS)
        - Each class will have a String tag and list of nodes which are members of that network
        - Nodes join the network when their third party applications broadcast an intent with some logical network tag.
            - The logical network manager adds the local node id to the logical network with the matching tag, and tells network OS instances on foreign devices via packets (foreign updates not yet implemented).
            - To the end user, they need only start the application. The network OS will then add their device to the logical network for that application.
        - LogicalNetworkManager maintains a set of LogicalNetworks, adds convenience methods, etc.

FRIDAY

Implemenation:

- Completed integrating the LogicalNetworkManager into the Network OS.
- To test the logical subnetwork implementation, it was necessary to begin implementing Intent communication for third party applications
    - Implemented a BroadcastReceiver, and a class dedicated to sending Network OS related broadcasts
        - Receives (from 3rd party to network OS):
            - GET_NODE_ID : “Please broadcast the local node ID”
            - JOIN_LOGICAL_NETWORK : “Please add this device to the logical network enclosed in the Extras Bundle”
            - eventually, intents to send data packets, with byte arrays in the extras bundle
            - requests to get information about various logical networks
        - Sends (Network OS to 3rd party):
            - LOCAL_NODE_ID : “Enclosed in extras is the local node id”

        - logical network information
        - possibly delivery confirmations?
- Also, began implementing a ‘third party” application..messenger demo
    - Unlike the previous 3 applications, this one does not have access to shared code (AIDL interfaces, shared data structures such as RouteTableEntry, Contact, or Constants), nor is it in a package common to any other.
    - Currently, just sends a broadcast asking the network OS to, in turn, broadcast the node address.
    - Has a broadcast receiver which will catch the network os’s reply, and display the node address in a Toast.

Issues:

- The network OS catches the demo’s broadcast, but the demo cannot seem to catch the network os’s replied broadcast (but it is being sent). No idea what is going on here..manifests look fine, and intents actions/filters all match. Additionally, sometimes this broadcast causes the control application to lose its connection to the network OS. Other times, there are mysterious null pointer exceptions. It seems as though the broadcast, on some end, is not going to the running instance, but starting a different instance, of one of the applications or services.

To-Do:

- Determine the PDU structure for communicating LogicalNetwork information
- Figure out why broadcasts are not totally functioning
- Determine how low-level third party application access will be