MONDAY

REU:

- Gave presenation:
  - Added new abstraction layer -- network os
  - control, network os, and ad-hoc communicate with AIDL
  - 3rd party applications use the network by communicating with the network os using broadcast intents
  - Focusing now on virtual subnetworks (logical networks)
  - 3rd party data sharing applications will use these to send data to other devices running similar software.
  - began implementing an example 3rd party application, a text messenger

Implementation:

- Continued working on messenger demo application
  - added code to:
    - type messages
    - join virtual network via popup dialog
    - select peers on same virtual network via spinner view
    - send message as a generic data packet
  - added code to network OS necessary to:
    - Get list of all nodes on some virtual network via intents (partial impl.)
    - Send generic data packets via intents
  - Modified ad-hoc service to:
    - Accept and send out generic data packets from the network OS
- New Network OS capabilities:
  - Can send generic packets, providing destination address, port number, and a byte array
    - this byte array is created and deciphered by instances of some third party application which sent it
    - Can contain anything -- text, files, etc. Large files should be split manually by the 3rd party app into separate intents.
- Network OS considerations
  - broadcasting everything isn't exactly secure -- any app could catch them. Likewise, any app could listen to received files and messages.

Goals for Tomorrow:

- Implement the receiving code for generic network os packets
- Implement the code to notify peers of logical network membership changes
- Begin testing cross-device 3rd party app communication

TUESDAY

Implementation:

- Completed the messenger demo
- Continued optimizing communication channels between the network OS and 3rd party apps.
- Fixed issues regarding how the ad-hoc service handled packets from the network OS
- Gave first cross-device, 3rd party app data transfer. After working out some kinks, it works
- Status of 3rd Party Network Access
  - Before apps can send data, they must announce themselves with an introduction intent, including:
    - Network Tag
  - Before they can have any meaningful communication, they must be able to ask

the network OS “Which other nodes (devices) are running the same app(s) as me?”. To do this, it sends another intent broadcast to the network OS, containing:
        - Network Tag
    - The network OS replies with an Intent broadcast of its own, containing the nodes.
        - The code which will update virtual network memberships across devices is not currently implemented. Therefore, the network OS cannot currently know of memberships on other devices. It will still attempt to send to any device, however.
        - Currently, we are testing by broadcasting packets, or using hard coded destinations. Messaging is successful.
    - 3rd Party apps (such as messenger demo) can currently send data packets across devices. These packets are packaged into an intent, which they must broadcast for each packet they wish to send:
        - Destination address (node id)
        - Network Tag (common to both local and destination node)
        - A byte array of data to send
            - 3rd party app on receiving end is responsible for parsing these bytes
- In terms of runtime effeciency, the packet parsing is the same as it was when originally coded (bytes from packets are cast to a String, and String functions are used). This implementation may not be the quickest. However, it is simple. It would also not be a huge task to change.
    - Also, the constant sending of intents during network processes which invoke a high number of packets, such as file sending, may introduce a significant overhead. This depends on the efficiency of Intents, and the maximum size of each packet. We will know more about this when we make the photo share demo.
- In terms of programming effeciency, 3rd party apps can send data using only 10 lines of code, plus a broadcast receiver. A Constants file should also be used. Take, for example, this valid ‘hello, world’ (assuming constants are defined):
    - Intent i = new Intent(JOIN_NETWORK_INTENT);
    - i.putExtra(NETWORK_TAG_INTENT_EXTRA, “logical network name”);
    - sendBroadcast(i); //join logical network
    - i.setAction(GET_PEERS_INTENT);
    - sendBroadcast(i); //poll network os for peers on this virtual network
    - //handle received intent containing peers with a BroadcastReceiver
    - //format whatever data you want to send into a byte[]
        - for example, byte[] someByteArray = “Hello, world!”.getBytes();
    - i.setAction(SEND_DATA_INTENT);
    - i.putExtra(DESTINATION_INTENT_EXTRA, someIntVariableNodeId);
    - i.putExtra(BYTES_DATA_INTENT_EXTRA, someByteArray);
    - sendBroadcast(i);
- The above code is how the messenger currently operates (basically). A file share app will need to get the file as a byte array, then break that byte array into pieces. Each of those pieces will need to be placed in a wrapper array, which has at index 0 the current piece index, and index 1 the number of pieces. The rest of the array will contain the data for that piece. Individual intent broadcasts must be done for each piece sent. Granted, 3rd party apps could actually handle this however they wanted. This just seems to be the simplest method (which I plan to use for the photoshare demo).
- Began low-level code for sending logical network changes over the network

Ideas:

- Rather than broadcasting intents to all 3rd party apps, could the network OS receive a specific path when the app announces its network name? This would help efficiency and security.

THURSDAY (Wednesday = holiday)

Implementation:

- Modified ConfigurationRequest Packets for modifying logical networks
    - Added a field for LogicalNetworkTag
    - Modified Parcelable methods for new field
    - Will use already existent targetNode field for the node to add
    - Added LOGICAL_NETWORK_ADD and REMOVE to ConfigurationRequestType
- Added methods and calls necessary to notify all connected peers of LogicalNetwork modifications (made during their connection)
    - A special call will need to be made somewhere to access the state the logical networks before a new node joins -- not too difficult. Can be broadcasted upon 3rd party app joining the logical network.
    - If a logical network by the given tag does not exist, it will be created. Upon receiving the LogicalNetwork update packet, the node is added to the logical network's set in nearly the same way as if it was a purely local addition
- Researched methods of accessing files in Android by bytes (to package into intents) -- should be very easy to get byte-level access to files. Issues:
    - Current packet parsing implementation, as was completed prior to the Software Defined Networking project, replies on casting the byes as a string and splitting on semi-colons....if we are going to send parts of files in these same generic packets from the network OS, there could be spontaneous errors (besides the fact we are using UDP and it may drop packets). If any part of the sent file is cast (by chance) as a utf-8 semi colon, packet parsing may fail. This is, however, unlikely, as the generic user data is appended to the end of the packet, and all significant semicolons should be encountered before the ones in the file.
    - That said, semi-colons actually send fine in the current Text Messenger demo, for the reason mentioned in the last few lines in the point above. We just need to be aware that this desgin flaw exists.
    - Legitimate fixes? There are certainly better packet parsing methods, but this is already implemented and fast/reliable enough for the SDN research. A faster/mroe reliable method would be needed before the software could be commercialized, but in terms of designing an SDN architecture, it makes no difference.
    - Another issue is that we do not yet know the best ratio between the number of intents sent, and the size of each intent extra bundle. For example, unless we implement specialized code to further break down large byte arrays in data send intents, before sending those new subdivisions as packets, we have to make it the responsibility of the 3rd party app developer to only send the amount of data that can fit in a single packet with each send intent. We can arbitrarily increase packet size, but this still leaves the question -- do we prefer a high send-intent count, or a larger packet/intent bundle size? We can't really decide without testing. So...
- Began developing photo-share demo.
    - Will, upon completion:
        - Get a photo file
        - Read it as bytes
        - Split into arrays of some size (TBD and optimized)

        - Each array will be sent as in intent then shipped as a packet with a position integer
        - On the other end, catch the file pieces, and show it when all have been captured.
    - Currently, implemented the GUI, and the code for reading the file as bytes. Began the code for segmenting and labeling the order.
        - Unlike with the messenger, the byte data won't be a string -- it will actually contain multiple independent fields (position, piece count, and raw data)
            - How many bytes to reserve for piece count and position? How many individual parts do we want to support? Optimally, this wouldn't be limited.

Plans for Friday:

- Implement calls to initialize logical network state for new nodes
- Continue with photoshare

FRIDAY

Implementation:

- Continued working on photo-share demo
    - Wrote class to handle the sending of photos:
        - Break photo into pieces (determined by user, limited by packet and intent constraints).
        - send pieces via intents, in formatted byte arrays
            - piece index, total pieces count, and transfer id
    - Code to catch received file pieces
        - Feed those pieces, as captured, into a specific PhotoShareFileBuilder
            - determined by transfer id
        - When all pieces are captured, write to external storage (sd card)
    - Code to find peers, accept user interaction, get local id, etc...
        - Copied from messenger demo -- identical function
- Completed most of Photo Share demo, began debugging
    - Long process...byte arrays everywhere and array index arithmetic is leading to numerous trivial INdexOutOfBoundsExceptions.
    - Began to wonder if I may have made this component over complicated.
    - Should have working complete file transfers by Monday