WEEK 4 NOTES

MONDAY

REU:

- Gave presentation on Week 3 progess:
    - Converted Connect activity to a service
    - Began coding the control application, including most of the UI
    - Implemented an AIDL protocol for communication between the control application and the background service
    - Implemented Parcelable classes for Contact, and flow table entries (incomplete) to send between the service and control application via AIDL
    - Researched Nypervisor and Network Operating Systems/NOX

Implementation:

- Running behind from last week -- mapping still broken
- Traced problems to incorrectly parsed packet info, and blacklisting errors preventing data from being returned to the original device
- Later: Fixed mapping. Still some funny behavior, but in general it appears to work.
- The parsing issue was resolved (mostly) and blacklisting was not an issue

TUESDAY

Research:

- Completed reading Nox: Towards an OS...
    - Purpose: Observe and control network
        - Controlling done via applications interfacing the OS; not the OS autonomously
        - Operate as if the entire network were present on one machine; centralized development and management
        - Applications written using high-level abstractions, not low level configuration parameters.
    - Set of switches and one or more servers (NOX runs on the servers)
        - Despite the fact that NOX runs on multiple servers, only one holds a global network view.
    - Switches should allow programmatic configuration modifications, such as by implementing OpenFlow
    - Observation granularity: Includes switch level topology; users, hosts, and middle-boxes, as well as other network components.
    - Does not provide real-time network traffic info
    - Control granularity: Control flows based on matching headers
    - Programmatic interface is built around events, namespace, and the network view.
    - Events can be passed up from OpenFlow (such as a new switch connecting), or created by NOX (such as a user joining a network).

Implementation:

- Completed implementation of the Parcelable RouteTableEntry class
    - Can now be passed via AIDL
    - Added an AIDL method (getForwardingTable) which passes a List of RouteTableEntrys back to the control program
- Implemented a new activity, ManageNodeActivity, which is activated when a node is selected from the listview of all network nodes in the control application.
    - This activity receives an integer contact id via the passed Intent, and

then queries the AdHocService for that contact id's forwarding table (with getForwardingTable(int))
    - Currently, when the AdHocService receives the getForwardingTable method...
        - If the desired contact id is that of the current device, it iterates through the Node's->RouteTableManager's->ForwardingRouteTable and for every ForwardRouteEntry (defined in aodv) it creates a new RouteTableEntry (the Parcelable and AIDL acknowledged class defined in com.SharedResources.DataTypes, accessible to both the service and the control application). It then adds all these RouteTableEntries to a List, and returns it over AIDL.
        - If the desired contact id is NOT the current device, it returns an empty list. To continue this, a new packet type must be implemented which can query other nodes for their route table information, and return it to the requester.
    - The activity takes the received list of entries and pushes them into a custom list view:
        - The OnItemClickListener for each entry provides all known details of that entry:
            - precursors
            - validity
            - next hop
            - destination address
            - hop count
            - destination sequence number
        - ...and a set of options for handling the entry:
            - remove entry
            - modify entry (which will just remove the entry, then request a new entry with the modified data)
    - Contains buttons to...
        - sort the forwarding table by different properties (important as the network grows/not significant at the current scale of testing -- 4 devices). Currently unimplemented.
        - add forwarding table entries (not yet implemented)
            - Need a new packet type which describes a forwarding table entry and contains a field to either remove all matching, or add, that described entry, before this button can be used for distant nodes
- Talked to Paul about how an issue with the current mapping/forwarding table modification scheme
    - If the network map is supposed to show all possible connections (ie all devices in wifi range), and it is formed by listing all nodes accessible in one hop, then wouldn't removing a flow table entry between two adjacent phones erroneously cause the map to show that they are not within range of each other? Implying that the map should show two phone's as adjacent if they are in range, even if they do not currently possess a flow table entry which makes this connection possible
        - Solution A) Rather than basing mapping on the flow table entries, make it autonomous and let it manage itself (find peers using a special broadcast, etc.)
        - Solution B) Implement some sort of blocking code which, when the control application asks to remove an entry, prevents that entry from being used for forwarding....but at the same time, allows the map

manager to see the blocked entries and retrieve adjacent nodes.
    - Solution A seems far better and cleaner, but, as long as implementing B doesn't become too messy, it will be quicker and work well enough. The tentative decision is to use B; blocking code, rather than an independent search for nodes in range.

WEDNESDAY

Implementation:

- Implemented the code necessary to remove route table entries from the current device
    - Pressing the 'remove entry' button calls the aidl method 'removeEntry(int, RouteTableEntry)'.
    - The service reacts...
        - if the integer passed is the current device's id, it adds the passed RouteTableEntry (after converting it to a native, non-aidl, ForwardingTableEntry) to the block list in the route table manager.
        - if the integer passed is the ID of another device, it currently does nothing. In the future, it should send this request to the device in question, which will react accordingly.
- Began implementing code to unblock/add new routing entries to the current device.
    - Behaves similarly to above
- Began implementing the code for the routing entry block list
    - The method used by AODV's route tables to find a corresponding route table entry was modified to accept a boolean skipBlockCheck. If the passed value is true, the function's behavior is unmodified. If the passed value is false, the block list is checked, and blocked entries are ignored.
    - The boolean parameter was added to methods up the call chain until the AODV Sender and Receiver classes...
        - The Receiver class passes the value true for all routing request packets. This means that blocking forwarding entries will not prevent routing requests from being passed around. This will keep the map manager from breaking (I hope). Update: yes, mapping does not break anymore.
        - The Sender class passes false for all user packets, unless the pdu type of the data in that packet is a map request (which should ignore block entries to properly map the network). Other types (such as configuration pdus, which will later be implemented to modify the routing tables of other devices) will also be permitted to skip the block check.
        - The idea that I am trying to implement is that map packets, route request packets, and control packets will be unaffected by SDN settings. Settings requested by SDN (ie the control application) should only affect generic user data...messages, file transfers? etc).
- Added an activity to modify and create entries with custom data.
    - Because of the current limitations of the addEntry method, this can only be used on the current device. That should change by tomorrow.

Remaining goals for the week:

- Get cross-device flow table editing to work
    - pdu type(s?)
        - One pdu for all configuration requests? (favored idea)
        - Different pdus for different request types?
    - Test the mapping and flow table mods (are they actually working as they appear to?)

THURSDAY

Implementation:

- Completed code for adding new forwarding table entries on the local device
    - Implemented a new Activity which shows editable fields for all components of an ForwardingTableEntry. This Activity can be called 2 ways:
        - For modifying an entry:
            - Pass Intent extras to load fields from the cached flow table entry
            - Pass intent extras to remove the cached flow table entry after adding a new one with the modified data
        - For creating an entry:
            - Do not pass the Intent extras for loading from cache or removing the cached entry; fields start blank.
- All routing requests and map requests seemt to ignore blocked entries (as desired). Whether or not general packets will adhere to the block list is yet to be seen (but hoped). Will need to test this in the future.
- Everything works from the ground up to edit flow tables on the current device. The rest of the day was used to design and implement a way to forward these requests to other devices.
- Implemented a new PDU type for configuration requests
    - Added exception to the blockListCheck.
    - This PDU has a field for configuration type
        - sender id
        - intended receiver
        - configuration request type
        - list of forwarding table entries (sometimes 1 entry, sometimes empty)
    - ...which is managed in code by the Enum class ConfigurationRequestType:
        - ADD_FORWARDING_ENTRY
            - Add the first entry contained by this PDU's forwarding entry list to your local entry list (The list should only contain one entry in this case)
        - REMOVE_FORWARDING_ENTRY
            - Remove the first entry contained by this PDU's...blah blah, same as above.
        - GET_FORWARDING_TABLE
            - The sender wants to know the receiver's forwarding table contents. This request itself should not contain any forwarding table entries
            - if received, create a new ConfigurationRequest pdu and fill the contained list of entries with the local forwarding table. Set the ConfigurationRequestType of the ConfigurationRequest to TAKE_FORWARDING_TABLE and reply to the sender.
        - TAKE_FORWARDING_TABLE
            - Effectively a reply to the GET_FORWARDING_TABLE type. If received, update your cached knowledge of the node which sent it.
- Wrote the code which accepts the control application's requests for forwarding table modifications, and either makes the modifications locally, or sends them to the appropriate devices.
- Wrote ConfigurationManager, which is called whenever some sort of ConfigurationRequest is received.

    - It handles converting RouteTableEntry objects (a Parcelable which I defined for passing over AIDL and easily converting to PDU bytes) to ForwardTableEntry objects (the native class used by AODV to manage routes).
    - It also directly manages the node's block list and local forwarding table modifications.

Persisting Issues:

- nondescript Parcel ReadException. I'll have to toy around with the parcelable classes, but so far I cannot test the configuration PDUs (sending or receiving), because my connection to AIDL no longer works for the method getFlowTable().

FRIDAY

Implementation:

- Debugging; a lot of code was written on Thursday; a lot did not immediately work. Friday was spent fixing much of Thursday's code and getting it to run.
- Tried a few methods to handle querying other devices for their forwarding tables, waiting for a response, then returning it after receiving it. Currently having some issues receiving replies. Hopefully I can have this fixed before Monday.