Week 5 Project Notes

MONDAY

REU:

- Gave presentation on week 4 progress:
    - Research Summary:
        - Researched NOX documentation
        - Noted responsibilities of the Network OS
        - Defined it's relative location between the Control application and the low level packet forwarding software.
    - Implementation Summary:
        - Completed the RouteTableEntry class
            - Parcelable data holder for sending between applications on the same device over AIDL -- class is in a package visible to both the control application and the low level application
            - Has methods to easily convert to and from PDU tokens for sending such information between different devices.
            - Must be converted to the native AODV ForwardRouteEntry class before the low level software can actually add it to table, or remove, etc. This is just a data holding and sending class.
        - Added GUI/Activities for displaying forwarding tables, details of table entries, and adding/modifying entries.
        - Implemented the 'entry removing' code, via a block list, and added exceptions for control packets.
        - Defined a new PDU type for inter-device forwarding table polling and modification, which itself currently has 4 subtypes:
            - ADD_FORWARDING_ENTRY
            - REMOVE_FORWARDING_ENTRY
            - GET_FORWARDING_TABLE
            - TAKE_FORWARDING_TABLE
        - Added the ConfigurationManager to the low level application, which from now on will handle received control request packets, making the requested changes locally, or sending the requested data to the requester (using ConfigurationRequest PDUs from above).
- Gave week 5 plans:
    - Complete inter-device flow table access and modifications
    - Determine and possibly implement a sort of network OS (if one would be feasible or useful)
    - Provide an interface for other applications to send data over the ad hoc network
    - Begin writing research paper

Implementation:

- Migrated many distributed methods to the ConfigurationManager, in order to centralize the handling of SDN requests.
- Trying a new algorithm for accessing the forwarding tables of foreign nodes
    - Rather than polling the foreign nodes and waiting for a response to return in the same method, so that the returned information can be given back immediately, the process will be split.
    - Much like the MapManager, there will be two separate methods; one to poll foreign nodes, and another to view all received data. Two separate buttons will call these two methods. I think the issue with my original design had to do with

threading, or something like that. Splitting it up should work much better.

- Added both buttons, now experiencing an issue where the pdu does not contain the forwarding table as it should. Debugged this for the last few minutes of the day.

TUESDAY

Implementation:

- Finally figured out why forwarding tables were not correctly sent over the network -- it was a dumb string formatting issue. Anyway, it is fixed. Cross-device route table editing (viewing, adding, removing, and modifying) works 100%.
- Finished up odd jobs; GUI related things, usability issues, not loading defaults, etc.
- Created a system of checking whether routing rules are being adhered to:
    - Created the RouteTracer PDU
    - Modified the Sender class:
        - When a RouteTracer is being sent from one node to the next, it will add the address of the current node, and display a toast message stating the last hop and next hop of the RouteTracer.
        - When the RouteTracer pdu reaches its ultimate destination, the wifi application displays a toast stating that a RouteTracer has been received, then displays the pdu's trace (the path it took through the nodes).
    - Initial tests are very good
        - At first connecton, route tracers are sent directly to the target, but these paths can be modified via SDN. For example, in one test using 4 devices...
            - A route trace from device B to D resulted in
                - B stating that it was forwarding a RouteTracer to D
                - D showing the path "B-D".
            - Device A was able to view the forwarding tables of devices B and C.
            - Device A was able to modify device B's forwarding entry to D; instead routing to a next hop of C (originally, it was routing directly to D), and listing 2 hops.
            - Device A then added an entry to device C's forwarding table, with a destination of D.
            - When running the route trace again
                - B stated it was forwarding a RouteTracer to C
                - C stated it was forwarding a RouteTracer to D
                - D displayed the path "B-C-D"
            - Device A could confirm that the route table changes had been made by viewing its peers' tables.

Planning:

- One forward route entry per destination is not enough... for example, in the above test, when I added the destination to device C's table for destination D after coming from B, it resulted in overwriting C's original entry for destination D....
- AODV currently maps an integer address to a specific forwarding table entry. I want to modify this to map to a list of forwarding table entries. This way, AODV can be modified to search for the next hop not only just by destination, but also by the last hop, etc.
    - This would allow devices to reach destinations differently depending on where the packet came from. Again, relating to the example above, device C could concurrently have a forwarding entry to D as a default, and a secondary entry destined to D for a specific last hop. This concept is very similar to that of flow control in SDN.

    - For example, a received packet to forward would have its last hop list extracted (if not currently possible, this would be implemented. I must discuss with Paul tomorrow).
    - Second, the List to which the address is mapped would be searched for that specific last hop, or other values. If no match is found, the default is used (whatever that may be). If one is found, forwarding is done in accordance with it.
- Would it be wise to add entries in a more protected way, such as how entries are removed (added to block list)? For instance, because the current AODV setup uses a map to a single entry, while we cannot truly remove entries (to prevent breaking mapping and control), it is possible to overwrite entries. This results in the same failure. Could this safety issue be solved with the map-to-list setup mentioned above?

WEDNESDAY

Implementation:

- The flow and action table, within the AODV ForwardRouteTable class. Added a Map from an integer (ultimate address) to a Set of ForwardRouteEntrys. Matching entries will be pulled from this table before they are pulled from the AODV default map (unless the block check is skipped).
- Modified the commands AODV uses to get forwarding entries. Rather than sending the destination address and a boolean to signify whether or not the block list should be checked, the methods accept the entire packet, and the boolean now designates, generally, whether or not SDN entries should be ignored entirely.
    - Passing the entire packet enables the ForwardRouteTable (modified to behave very similarly to an SDN FlowTable) to find custom SDN entries based on destination, pdu type, source address, port number, and source port.
    - A passed true value means that not only will the blocked entry list be ignored, but special SDN routes will be skipped as well.
- Implemented a parcelable FlowFilter class, which carries data between the control application and the wifi service for handling forwarding based on ports and addresses.
- Implemented the FlowFilterManager, which manages matching packets to filters, etc, in the wifi service application.
    - Use of this class should dramatically reduce the amount of modifications I need to make to AODV directly -- time saving and shouldn't cause too many bugs.
    - FlowFilterManager maintains a sorted set of FlowFilters, which themselves implement Comparable based on their respective priorities.

THURSDAY

REU:

- Planned and began on Monday's slide show
    - Explanation of limitations of current networking scheme
    - Intro to SDN as a solution
    - Intro to the AODV android network
    - Explanation of how we implemented SDN on the android network, and how it opposes traditional SDN
    - Current progress (core structure is complete)
    - Remaining work (additional functions; additional filter types, etc)

Implementation:

- Deleted everything from Wednesday. Literally restored from Tuesday's backup
- Wednesday's work was the right idea, but I went about it in the wrong way entirely. Way too many files modified, way to many senseless approaches that should have been centralized elsewhere.

- Began redoing wednesday's work in a better way, on a smaller initial scale.

FRIDAY

REU:

- Began writing interim progress report for Monday
    - Summarize project (Title and goal)
    - Progress made
    - Observations
    - Lessons, if any
- Completed presentation

Planning:

- Considering different strategies for handling flows
    - Could port and destination level routing rules be managed by the ConfigurationManager?
- Talked to Paul; will not be implementing code to filter packets by ports. It turns out that the ad-hoc software may ignore ports
    - Instead, next goal is to pull some code up from the wifi service to make a distinguishable network os layer.