TUESDAY

Notable Readings

- ITC Keynote by Nick McKeown
  - http://www.itc23.com/fileadmin/ITC23_files/slides/K1_McKeown-ITC_Keynote_Sept_2011.pdf
- Virtual Network Mapping
  - http://pc2.uni-paderborn.de/uploads/tx_sibibtex/visa05d-lischka.pdf

Implementation:

- Studied the AODV ad-hoc architecture
- Defined a new PDU type for mapping requests (called MappingRequest). PDU type contains (currently):
  - PDU Type (defined in Constants as PDU_MAPPING_REQUEST)
  - A randomly generated (hopefully unique) ID from 0-99999
  - ID of original requester
  - Blacklist (where the request has already been sent)
  - Status (building = ignore blacklisted, returning = ?? working on that)
  - Global View Table (cumulatively created table of network links)
- Defined MappingManager, which handles all incoming MappingRequests
- Added appropriate entries to AppControl to properly direct incoming MappingRequests

WEDNESDAY

Notable Readings:

- Virtual Networking
  - http://www.wired.com/wiredenterprise/2012/05/what-is-a-virtual-network/all/1
  - Steve Herrod, chief tech officer at VMWare. Virtual servers run 65% of all server tasks on earth. Same concepts can be applied to networks.
- Uses of SDN
  - http://www.technologyreview.com/web/22120
  - Modify flow tables
  - Test new switching and routing protocols
  - EX: video priority over email
  - Quarantine suspicious devices
- More uses
  - http://www.openflow.org/documents/openflow-wp-latest.pdf
  - New routing protocols/IP Alternatives

Implementation:

- Added activity to SingleHopWifi.view (Accessible via menu option in ContactsView) to view current network.
- Currently, the NetworkMapActivity can be used to send mapping requests to all connected peers, which in the SingleHopWifi application means only the adjacent nodes.

The requests are appropriately forwarded, and the current network map is appropriately and consistently built. Even when connections are artificially blocked (to simulate two phones out of range), MappingRequest forwarding ensures that all phones generate the same network map.

THURSDAY

Notable Readings:

- The Future of Networking and the Past of Protocols
  - http://opennetsummit.org/talks/shenker-tue.pdf
  - Scott Shenker, Nick McKeown, et al.
  - 3 Layers of abstraction:
    - Highest: Control Program, specify behavior on an abstract network model
      - (What is this abstract model..?)
    - Mid: Network Virtualization, convert the specified abstract model to the global network view (topological graph of network nodes).
    - Low: Forward details of the global network view to the physical devices.
      - Distributed state abstraction
      - Forwarding abstraction
- Open Networking Foundation Whitepaper
  - https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf
- Lippis Report 173
  - http://lippisreport.com/2011/06/lippis-report-173-software-defined-networking-the-openflow-way-grabs-industry-attention/
- Application Tier of Software Defined Networking
  - http://searchnetworking.techtarget.com/feature/The-application-tier-of-a-software-defined-networking-architecture

Implementation:

- Refined how mapping requests are handled, and how the network map is displayed.
- Determining a reasonable way to pass this network map info to control programs. It seems like, for this Android ad-hoc network, where all the nodes are homogeneous, the low and mid levels of abstraction may not be significantly distinguishable from one another; they can probably be combined into one utility.
- Trying to determine if there is some way to encapsulate the current SingleHopWifi or WifiTransfer V3 applications as a service, with which the control program can communicate. Else, write a new service using the libraries and code from the original two applications. The newly written (or assembled) Service will emulate the lower two levels of abstraction, and via interprocess communication (AIDL or Messengers...?) will connect to the control application.
- The highest level (control application) could be a standalone application which uses the lower/mid level Service utility to send out configuration requests and accumulate network details (providing a network map).

- The control application should be able to define specific paths when launched normally, and also act as a ContentProvider, which other applications can hook into to send data (photo share, or file transfer).
- Rough idea of how I visualize the flow of control currently:
    - User launches any generic application, and fires a generic share intent
    - User chooses the control program from a list of possible file sharing utilities (handled by the Android OS)
    - The control program receives the data, and provides a UI for the user to....choose a path, given a topological network graph? User should be given some SDN options.
        - Perhaps blacklisting specific nodes? Only sending along trusted nodes? Avoiding nodes, but determining the actual path automatically? Seems like there is a lot of graph theory that can be built into the control program.
    - The control program passes the specified path (or SDN rules, something SDN related) and the data to be sent to the lower level service, which translates the provided path and SDN rules into specific RoutingRulesRequests to be sent to the relevant nodes, and a PDU container which specifies the path on which it is to be sent. This PDU container could perhaps contain other PDUs.
        - PDU fields would include the type first, then the path to send the data along, then the third field would contain the actual PDU item being sent (some request or data packet)
    - As the container PDU is sent along, nodes visited are popped off of the second field, and when the second to last node is reached, the entire Container PDU is removed, and the stored packet is sent directly to its destination (using a single hop service to guarantee that no alternate forwarding is being used).
- Began toying with a UI for the control layer application to better visualize how the three layers may interact.
- Began looking into Android Interface Definition Language to communicate between the service and control application. Types of communication:
    - Control -> Service: request topological network map
    - Control <- Service: topological network map
    - Control -> Service: Network settings
        - blacklist nodes
        - propose specific forwarding rules
    - Service -> Devices: Specific rules
    - Control -> Service: Data to send and path to send along
    - Control <- Service: Success or failure of data sent

FRIDAY

Notable Readings:

- Implementing an OpenFlow Switch on the NetFPGA Platform
    - http://yuba.stanford.edu/~jnaous/papers/ancs-openflow-08.pdf
    - Nick McKeown, et al

- OpenFlow acts over unmodified switches and routers
- It pushes the complexity of determining network paths from the network hardware (routers, switches) to user modifiable software controllers.
    - Controller administrator has full control over forwarding decisions
    - Can handle traffic belonging to different networks in different ways
        - Experimental traffic can be separated from general traffic; safe experimentation on widely used networks
- Data-path elements are flow-switches which:
    - have a flow-table
    - can talk to remote controllers using the OpenFlow protocol
- A flow is all packets matching a flow-entry in a switch's flow table
    - flow entries are general
    - Controller decides which flows to admit, and the path their packets should follow
- OpenFlow is simple; only defines:
    - flow-based data-path switches
    - protocol for adding and deleting flow entries
    - compatible with any controller which speaks the OpenFlow protocol
- Flow-entries contain a flow description and an action associated with that flow
- OpenFlow controllers communicate with the OpenFlow switch via SSL
- If a packet cannot be matched to a flow entry, it is encapsulated and sent to the controller over the SSL channel The controller examines it, updates the networks flow tables, and returns the packet. Following packets from flow need not consult the controller.
- Flows are classified based on 10 properties:
    - Switch input port
    - Source MAC address
    - Destination MAC address
    - Ethernet type
    - VLAN ID
    - IP source address
    - IP destination address
    - IP protocol
    - TCP/UDP source port
    - TCP/UDP destination port
- Openflow Switch has three responsibilities:
    - Forwarding packets to a specified set of output ports
    - Encapsulate and send packet to controller
        - first packet of flow to determine path
    - Drop packets (security)
    - Optional:
        - VLAN editing
        - IP address rewriting

- TCP/UDP port rewriting

- OpenFlow Switch Specification
  - http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf
  - Version 1.0.0
  - Flow Table:
    - Set of flow entries
    - Activity counters
      - Per-table, per-flow, per-port, per-queue
    - Set of actions to apply to matching packets
  - If a matching flow entry is found for a packet, any actions for that entry are performed
    - for example, forward out a specific port
  - If no match is found, the packet is sent to the controller via the SSL connection.
    - The controller then is responsible for adding and removing the relevant flow entries to get the flow moving
    - A value of ANY for a header entry serves as a wildcard
      - Matches with no wildcards are preferable to those with
      - Different header values are ranked differently when both have wildcard values
  - Flow entries may forward to one or more OpenFlow ports
  - Required OpenFlow Action Support:
    - Forward to...
      - All interfaces
      - Controller
      - Local networking stack
      - Table
      - Input port
    - Drop
  - Optional OpenFlow Action Support
    - Forward to...
      - Normal networking interface (regular routing hardware)
      - Flood along minimum spanning tree, not to sender
    - Enqueue, or forwarding to a queue where the queue configuration dictates further forwarding behavior
    - Modify-field (although VLAN modification at least is very much suggested)
  - Relevant to the ad-hoc network: Support for multiple simultaneous controllers is currently undefined
  - Controller to switch messages:
    - Features: Controller asks switch whether a capability or feature is supported, switch confirms or denies
    - Configuration: Set and query switch configuration; switch replies only to queries
    - Modify-state: add/delete/modify flow tables, and set port proerties

        - Read-state: collect statistics from flow-tables, ports, and flow-entries
        - Send-packet: Send packets through a specified port on switch
        - Barrier: Ensure message dependencies, or receive notifications for completed operations
    - Switch to controller messages:
        - Packet-in: packet did not have a flow-entry
        - Flow-removed: entry timed-out
        - Port-status changes
        - Error messages
    - Bi-directional
        - Hello: initial connection
        - Echo: test latency, bandwidth, liveliness of connection
        - Vendor: custom
    - Project relevant: If connection to controller is interrupted, default to a backup controller.

Implementation:

- Where will the controller be?
    - OpenFlow uses a client-server model for controlling the network
    - AODV Android network is peer-to-peer...which phone runs the controller?
        - Alternative 1: distributed control: each phone can modify the routing rules of others.
            - When a routing decision needs to be made for a packet, who is consulted?
                - Sender?
                - Current node?
        - Alternative 2: Run the controller software on a pc (acting as SDN server)
            - probably not worth the complications, and destroys the purpose of the mobile ad-hoc network in the first place.
        - Alternative 3: Choose an arbitrary device as the control center
            - Nodes no longer homogenous
            - less flexible; what happens if the controller connection is lost?
- What fields do we want to keep for the proof of concept?
    - Source address
    - Destination address
    - …?
- Again, rather than SSL, how to communicate with the controller? AIDL seems to be overkill, and only works for the model wherein we have multiple controllers.