COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 5360/6360
**Wireless and Mobile Networks**
Spring 2013

**Project 3**

# Mobile Phones without Cell Towers Based on OLSR Routing in Mobile Ad Hoc Networks

**Due: 11:55pm April 25, 2013**

## 1 Objective

The goal of this project is to design and implement an *Optimized Link State Routing (OLSR) algorithm over an ad hoc network for a mobile phone voice communication application* using smart cell phones or computers in a WiFi emulation environment, i.e. without the use of cell towers or access points. As in the second project, this voice chat application will allow users to have two-way voice communication over the ad hoc network. You may implement this either in a real wireless environment or an emulation of the wireless environment. In the *real environment*, two devices will be connected through an *ad hoc network* where each pair of nodes is connected through a WiFi link in ad-hoc (IBSS) mode. In an *emulation environment*, you may emulate the underlying wireless link layer with exchange of UDP datagrams between two neighboring hosts through AF_INET sockets. However, the protocol you will implement can be ported to an actual network layer. In this third project, audio packets will be transmitted over wireless mobile and ad-hoc networks (MANET) using the OLSR ad-hoc routing protocols or an emulation of the MANET over a fixed network. Your implementation will also use various software that support sound API (Application Programmer Interface).

## 2 Background

In the events of earthquakes, hurricane, or other wide-spread disasters and even in remote areas such as in the mountains, cell phone towers or access point infrastructures will likely go down or are non-existent. In the absence of cell towers or access points, your sophisticated smartphones will be rendered useless, unless we can make the mobile phones systems more resilient and capable of operating even without cell towers, access points and other communication infrastructures. Our goal is to allow mobile phone users to communicate with each other through a virtual and dynamic network that can be spontaneously established in areas where no cell tower coverage exists. People traveling to remote mountain areas will then be able to communicate with each other for free.

For communication over a wide area, multiple wireless devices spread over the area will form mobile and ad hoc networks (MANETs) that can be used to forward packets from one caller to another who are at the extreme ends of the MANETs. Voice packets from the first caller will be forwarded using MANETs routing protocols through multiple hops of the intermediate wireless devices and eventually to the second user at the other end of

the MANET. Two way voice packets may then be forwarded through similar paths in the reverse direction.

# 3 Requirements

The objective of this third project is to implement an *OLSR* routing algorithm over a MANET for supporting multi-hop forwarding of voice packets for a working two-way voice communication system that will enable users to initiate voice conversations with other users. Using these programs, a user at host computer or smartphone A will be able to speak through a microphone attached to A where a program will extract the voice data and transmit it in a packet through a UDP socket to another computer representing an intermediate node in the MANET. The voice packet is then forwarded to another MANET node and eventually to another smartphone B attached to the MANET. (Smartphones may be replaced by other wireless Android devices for extra credit.) The program in B will receive the voice packet and play the voice message in its speaker. Similarly, the user at B will be able to speak through its microphone attached to B and their voice packet is forwarded through the MANET and eventually delivered to A and played on A's speaker. Make sure that both A and B are separated over a long distance, otherwise there could be feedback from B's speaker into A's microphone and vice versa. An alternative way to avoid the feedback is to use a headphone.

You can code this program in any language of your choice, although we recommend that you code it in Java. This project is usually more easily implemented with Java Sound API, Java socket API and multithreading.

## 3.1 Voice Chat Programs

As in the previous projects, you may implement a simple voice chat program for two-way communication or more sophisticated voice communication program that operates similar to a typical cellphone. The second option will receive extra credits. The simple voice chat system must operate like a typical cellphone and could consist of two programs: (1) a transmitter that extracts the voice data from the microphone and transmits it to the other user through a UDP socket and (2) a receiver that receives the voice packet from a UDP socket and plays it on its speaker. The quality of the voice should be reasonable. Do not be too concerned with the delay of the voice in the receiver's speaker, but the delay should be reasonable. (You can attempt to improve the quality of service and reduce the delay for extra credit.) You can make use of Java Sound package to process the audio signals from the microphone and playing it on the speakers.

## 3.2 Optimized Link State Routing (OLSR) Protocol

### *3.2.1 Overview*

OLSR protocol is an optimization of a pure link state protocol for mobile ad hoc networks. First, it reduces the size of control packets: instead of all links, it declares only a subset of links with its neighbors who are its multipoint relay selectors. Secondly, it minimizes flooding of this control traffic by using only the selected nodes, called multipoint relays, to diffuse its messages in the network. Only the multipoint relays of a

node retransmit its broadcast messages. This technique significantly reduces the number of retransmissions in a flooding or broadcast procedure.

### *3.2.2 Neighbor Sensing*

Each node must detect the neighbor nodes with which it has a direct and bi-directional link. The uncertainties over radio propagation may make some links uni-directional. Consequently, all links must be checked in both directions in order to be considered valid.

To accomplish this, each node periodically broadcasts its HELLO messages, containing the information about its neighbors and their link status. These control messages are transmitted in the broadcast mode. These are received by all one-hop neighbors, but they are not relayed to further nodes. A HELLO message contains: (1) the list of addresses of the neighbors to which there exists a valid bi-directional link; (2) the list of addresses of the neighbors which are heard by this node (a HELLO has been received) but the link is not yet validated as bi-directional if a node finds its own address in a HELLO message, it considers the link to the sender node as bi-directionat.

These HELLO messages permit each node to learn the knowledge of its neighbors up to two hops. On the basis of this information, each node performs the selection of its multipoint relays. These selected multipoint relays are indicated in the HELLO messages with the link status MPR. On the reception of HELLO messages, each node can construct its MPR Selector table with the nodes who have selected it as a multipoint relay.

In the neighbor table, each node records the information about its one hop neighbors, the status of the link with these neighbors, and a list of two hop neighbors that these one hop neighbors give access to. The link status can be uni-directional, bi-directional or MPR. The link status as MPR implies that the link with the neighbor node is bi-directional AND that node is also selected as a multipoint relay by this local node. Each entry in the neighbor table has an associated holding time, upon, expiry of which it is no longer valid and hence removed.

The neighbor table also contains a sequence number value which specifies the most recent MPR set that the local node keeping this neighbor table has selected. Every time a node selects or updates its MPR set, this sequence number is incremented to a higher value.

### *3.2.3 Multipoint Relays*

The idea of multipoint relays is to minimize the flooding of broadcast packets in the network by reducing duplicate retransmissions in the same region. Each node in the network selects a set of nodes in its neighborhood, which retransmits its packets. This set of selected neighbor nodes is called the multipoint relays (MPRs) of that node. The neighbors of any node N which are not in its MPR set, read and process the packet but do not retransmit the broadcast packet received from node N. For this purpose, each node maintains a set of its neighbors which are called the MPR Selectors of the node. Every broadcast message coming from these MPR Selectors of a node is assumed to be

retransmitted by that node. This set can change over time, which is indicated by the selector nodes in their HELLO messages.

Each node selects its multipoint relay set among its one hop neighbors in such a manner that the set covers (in terms of radio range) all the nodes that are two hops away. The multipoint relay set of node N, called MPR(N), is an arbitrary subset of the neighborhood of N which satisfies the following condition: every node in the two hop neighborhood of N must have a bidirectional link toward MPR(N). The smaller is the multipoint relay set, the more optimal is the routing protocol.

### *3.2.4 Multipoint Relays Selection*

Each node of the network selects independently its own set of multipoint relays. The MPR set is calculated in a manner to contain a subset of one hop neighbors which covers all the two hop neighbors, i.e., the union of the neighbor sets of all MPRs contains the entire two hop neighbor set. In order to build the list of the two hop nodes from a given node, it suffices to track the list of bidirectional link nodes found in the HELLO messages received by this node (this two-hop neighbor information is stored in the neighbor table). The MPR set need not be optimal, however it should be small enough to achieve the benefits of multipoint relays.

Multipoint relays of a given node are declared in the subsequent HELLOs transmitted by this node, so that the information reaches the multipoint, relays themselves. The multipoint relay set is re-calculated when:

- a change in the neighborhood is detected when either a bi-directional link with a neighbor is failed, or a new neighbor with a bi-directional link is added; or
- a change in the two-hop neighbor set with bi-directional link is detected.

With information obtained from the HELLO messages, each node also construct its MPR Selector table, in which it puts the addresses of its one hop neighbor nodes which has selected it as a multipoint relay along with the corresponding MPR sequence number of that neighbor node. A sequence number is also associated to the MPR
Selector table which specifies that the MPR Selector table is most recently modified with that sequence number. A node updates its MPR Selector set according to the information it receives in the HELLO messages, and increment this sequence number on each modification.

### *3.2.5 MPR Information Exchanges*

In order to build the routing table for forwarding packets, each MPR broadcasts Topology Control (TC) messages which are forwarded (only by MPRs) like usual broadcast messages in the entire network. This technique is similar to the link state technique used in ARPANET, except that only MPRs are involved for better scalability.

A TC message is sent periodically by each MPR in the network to declare its MPR Selector set, i.e., the message contains the list of neighbors who have selected the sender node as a multipoint relay. A sequence number is associated to this MPR Selector set. The information diffused in the network by these TC messages will help each MPR node

to build its topology table. The interval between the transmission of two TC messages depends upon whether the MPR Selector set is changed or not, since the last TC message transmitted.

Each node of the network maintains a topology table, in which it records the information about the topology of the network obtained from the TC messages. A node records information about the multipoint relays of other nodes in this table. Based on this information, the routing table is calculated. An entry in the topology table consists of an address of a (potential) destination (an MPR Selector in the received TC message), address of a last-hop node to that destination (originator of the TC message) and the corresponding MPR Selector set sequence number (of the sender node).

Upon receipt of a TC message, the following procedure is executed to record the information in the topology table:

1. If there exist some entry in the topology table whose last-hop address corresponds to the originator address of the TC message and the MPR Selector sequence number in that entry is greater than the sequence number in the received message, then no further processing of this TC message is done and it is silently discarded (case: packet received out of order).
2. If there exist some entry in the topology table whose last-hop address corresponds to the originator address of the TC message and the MPR Selector sequence number in that entry is smaller than the sequence number in the received message, then that topology entry is removed.
3. For each of the MPR Selector address received in the TC message:
    - If there exist some entry in the topology table whose destination address corresponds to the MPR Selector address and the last-hop address of that entry corresponds to the originator address of the TC message, then the holding time of that entry is refreshed.
    - Otherwise, a new topology entry is recorded in the topology table.

### *3.2.6 Routing Table Calculation*

Each node maintains a routing table which allows it to route the packets for other destinations in the network. The nodes which receive a TC message parse and store some of the connected pairs of form [last-hop, node] where ''nodes" are the addresses found in the TC message list. The routing table is built from this database by tracking the connected pairs in a descending order. To find a path from a given origin to a remote node R, one has to find a connected pair (X,R), then a connected pair (Y,X), and so forth until one finds a node Y in the neighbor set of the origin.

The route entries in the routing table consist of destination address, next-hop address, and estimated distance to destination. The entries are recorded in the table for each destination in the network for which the route is known. All the destinations for which the route is broken or partially known are not entered in the table.

The routing table is based on the information contained in the neighbor table and the topology table. Therefore, if any of these tables is changed, the routing table is re-

calculated to update the route information about each known destination in the network. The table is re-calculated when a change in the neighborhood is detected concerning a hi-directional link or when a route to any destination is expired (because the corresponding topology entry is expired).

The following proposed procedure may be executed to calculate (or re-calculate) the routing table:

1. All the entries of routing table are removed.
2. The new entries are recorded in the table starting with one hop neighbors (h = 1) as destination nodes.
3. Then the new route entries for destination nodes h + 1 hops away are recorded in the routing table. The following procedure is executed for each value of h, starting with h = 1 and incrementing it by 1 each time. The execution will stop if no new entry is recorded in an iteration.
    - For each topology entry in topology table, if its destination address does not corresponds to destination address of any route entry in the routing table AND its last-hop address corresponds to destination address of a route entry with distance equal to h, then a new route entry is recorded in the routing table where:
        - destination is set to destination address in topology table;
        - next-hop is set to next-hop of the route entry whose destination is equal to above-mentioned last-hop address; and
        - distance is set to h + 1.
4. After calculating the routing table, the topology table entries which are not used in calculating the routes may be removed, if there is a need to save memory space. Otherwise, these entries may provide multiple routes.

### *3.2.7 Audio Packet Forwarding*

Based on the routing tables created by the OLSR algorithm described above, an audio source node can route its audio packets to the destination. The source node will first generate a voice packet from the microphone and forwards it to the next hop MPR node specified in the routing table based on the destination. Eventually the packet will be forwarded through multiple hops and reach the destination address. Every wireless node in the ad-hoc network has a globally unique 16-bit address (not IP address), which you can define. The audio packet header must contain at least the following fields:

1. Sequence number
2. Source address
3. Destination address
4. Previous hop

The audio packets are limited to 128 bytes, including a header. For longer messages, the higher layer protocols or applications must perform the corresponding segmentation and reassembly of the message to/from different packets.

### 3.3 Mobility

The nodes in your network represent smartphones that a group of people carry as they move around in the area. You must model at least 20 smartphones. For this project, you MUST model the mobility of the smartphones which also act as routers. Model the speed of people to be between 0.1 to 1 meters per second (i.e. .25 to 2.5 mph). Each node moves at different random speed, different directions and at different times. It may make random decisions to turn towards different directions. Make sure that the nodes stay within the area that you are modeling. You do not need to model the nodes to avoid them from going over each other, i.e. it is fine for the nodes to overlap in space.

### 3.3 Network Configuration

Each network node will be emulated by a separate execution of your program in one of the tux computers. Nodes will run on a subset of the COE instructional Linux workstations in Davis 123, using the computers tux175 – tux198 with each node corresponding to a (machine IP address, UDP port) pair. Use only port numbers that are assigned to you. Therefore, each execution of your program will represent a virtual node communicating with other virtual nodes, each through its own UDP port. There may be many such virtual nodes running on a single workstation or running on several workstations.

A configuration file will specify the virtual nodes in the network and the links between them. Your program will take the configuration file as a command line argument. Then, the configuration file will have to be parsed and a copy of your program will be executed in each virtual node. The format of the network configuration file is shown next, while the topology of the network represented by that file is shown in Figure 2. The x- and y-coordinates of each node is specified preceding the link keyword. For example, the coordinate of Node 1 is (50, 120).

```
Node 1 tux175, 10010 50 120 links 2
Node 2 tux175, 10011 80 120 links 1 3 4
Node 3 tux180, 10010 180 170 links 2 4
Node 4 tux180, 10011 150 60 links 2 3
```

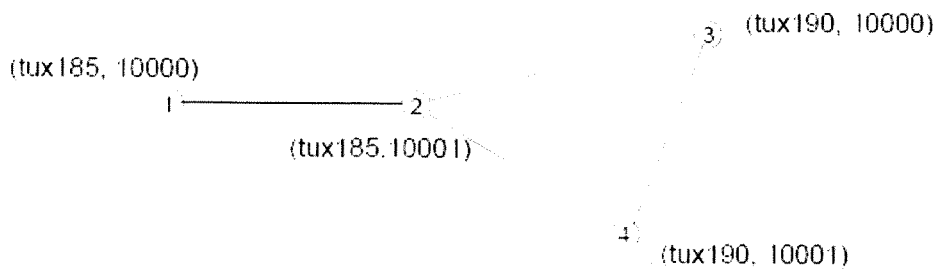


Figure 2. Network Topology

To emulate mobility of hosts, your program running in each virtual node must constantly monitor any change in the configuration file. When a mobile host moves to another location, its link to the fixed hosts in the previous location is disconnected while a new link is established to fixed hosts in the new location. The change in the configuration file can be manually done or automatically updated by a randomized program. The configuration file may also contain location information of each node. Each node periodically read the configuration file to determine the set of neighbors that it can reach through a wireless links and their new locations. The connectivity of the fixed hosts should not change.

## 4 Testing

Your voice chat application programs must run over the virtual MANET which must execute correctly in the College of Engineering Linux tux computers in Davis 123, unless you are doing extra credits and implementing the voice chat system in Smartphones or Android devices.

The voice chat application that broadcasts voice packets to some destination will make use of the OLSR protocol functions that you have implemented above. Make sure that the source and all wireless nodes are executed on different computers. The voice chat application must allow at least two users to communicate with each other, whereby each user must use a different tux computer at opposite end of the MANET. You will supply your own microphones; a speaker is already attached to all the tux computers

You must run this application with at least 20 wireless nodes (representing intermediate smartphones). These wireless nodes are distributed in a square area of 3R X 3R, where communication range is slightly larger than R. Your OLSR protocol must correctly execute with two network configurations: 1) Fixed mesh of 4 by 4 nodes where the horizontal and vertical distance between them is R, 2) Random distribution of the nodes in the 3R by 3R area. Each intermediate wireless node that receives the voice packet must route it correctly onto the next outgoing link. For this project test, make sure that all wireless nodes are mobile except the sender and receiver.

If you choose to implement the extra credit voice chat program over smartphones or Android devices, make sure your programs work in the devices that you provide yourself.

## 5 Extra Credits

For extra credits, you may choose to do the following. There could be *significant extra credits* given if the following tasks are implemented successfully.

1. Implement your voice chat programs on smartphone (e.g. iphones) or wireless Android devices. These devices include tablets, media player, and smartphones, all of which must have WiFi capability. You may implement this on the Android Operating System or alternatively over Linux that you can install on these devices. Your program must execute correctly over the Android devices that form

a spontaneous mobile and ad-hoc networks (MANETs) whereby these devices communicate through WiFi in Ad-hoc (IBSS) mode.

2. Improve the quality of your voice chat application programs by adding the following enhancements:
    a. Improve the Quality of Service (QoS) of the voice communication, e.g. reducing the delay and noise, e.g. using a voice codec, so that it is comparable to typical cell phones services. You may also incorporate echo cancellation program so that you can use speakers as well as headphones.
    b. Implement a capability for dialing a certain user using some form of numbering or naming scheme and provide a service for looking up a number or name in the server.
    c. Provide an efficient and nice GUI for the users to execute the voice chat programs and manage the connections. This GUI must have the functionalities of typical cellphones, including but not limited to allowing users to dial the destination node.

## 6 What to do and turn in

After you have implemented, debugged, and thoroughly verified your voice chat implementation, submit the following in Canvas on or before the due date.

a. Complete two-way voice chat application codes and the OLSR protocol codes (which should include comments for full credit).
b. An execution trace of some important nodes (of important events only (packets received and sent), just enough to demonstrate the protocols and the voice chat application works correctly.
c. A brief (2 pages) report that describes your system architecture, design and implementation issues, including sections on introduction and motivation, problem statement, details of your algorithms, experimental results, and conclusions.

*You will demonstrate your program to the TA or me on April 26, 2013 in Davis 123 on the COE network of Linux computers.*