COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 5360/6360
Wireless and Mobile Networks
Spring 2013

Project 2

Mobile Phones without Cell Towers With Flooding Routing in an Ad Hoc Network

Due: 11:55pm March 21, 2013

1 Objective

The goal of this project is to design and implement a *flooding routing 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 first project, this voice chat application will allow users to have two-way voice communication over the ad hoc network. You ay 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 second project, audio packets will be transmitted over wireless mobile and ad-hoc networks (MANET) using a flooding 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 second project is to implement a flooding 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 project, 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 Flooding Protocol for MANETs

You will design and implement the flooding protocol on top of the simple UDP socket interface that emulates the link layer represented by the shaded portion of Figure 1. The shaded portion is not part of this project, but represents protocols that could conceptually be added or substituted at the layer where it is shown.

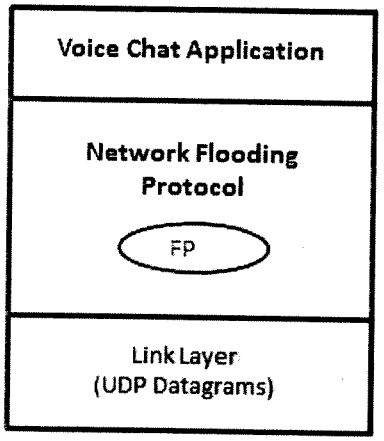

Figure 1. Overview of the Protocol Components

Initially, a source node will generate a voice packet from the microphone and forwards it to all its neighbor nodes through its FP layer. Eventually the packet will be forwarded through multiple hops and reach the destination address. The FP layer of the intermediate nodes performs the task of forwarding packets from one node to another. The packet header must contain at least the following fields:

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

One issue in FP forwarding is the addresses in the packet header. Every wireless node in the ad-hoc network has a globally unique 16-bit address (not IP address), which you can define. Since we will not be focusing on unicast or multicast forwarding, the voice packet's destination address will contain a special broadcast address. Although typically the higher-half of the address space is reserved for distant network broadcast addresses, you will use only a single network in this project, in which case, all the address bits of the broadcast destination address is set to 1.

The FP 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.

You must calculate the packet drop rate for each link which is proportionate to the square of the distance. Develop a function to compute the packet drop rate. When the error rate is above a certain threshold value, consider the node is out of range. Set your packet loss function so that the range of each node's transmission is about 100 meters. The error rate should capture the propagation loss (optional to include the effect of slow fading and fast fading). You can calculate the distance between nodes based on the current location.

Each router will run the flooding protocol as follows. When a node with address j receives a packet, it first checks the source address i of that packet. If $i = j$, then the node will discard the packet, since it is the source of the packet. Otherwise it checks in its

cache table the largest sequence number of broadcast packets that it has received from source *i*. (Each FP layer maintains a *cache table of broadcast packets* that it has received from each source with the largest sequence number of packets that it has received from it up to that point, i.e. the cache table contains information on flooding packets that the node has forwarded previously.) If that sequence number in the cache table is smaller than the sequence number in the new packet, then that packet is new and *needs to be forwarded.* The router *j* then updates the cache table with the largest sequence number that it has received from *i*. It then forwards a copy of the packet to *each* of its neighbors that is not the neighbor *k* that it received the packet from. The neighbor *k* can be determined from the "Previous Hop" field specified in the packet header. Eventually the packet is sent to all reachable wireless nodes, including the destination of the voice packets.

Some of the design issues for FP that you must address are as follows:

1. FP packet header
2. Forwarding mechanism for flooding packets
3. Structure of cache tables
4. Memory management of packet buffers

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
```

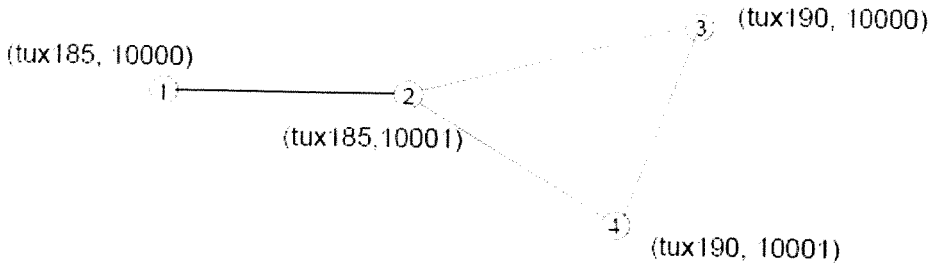

Figure 2. Network Topology

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 network flooding 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 16 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 flooding 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 flood it onto all other outgoing links. For this simple test, assume that all wireless nodes are stationary.

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 flooding 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 March 22, 2013 in Davis 123 on the COE network of Linux computers.