**Ad-Hoc Routing Project**

**Project 2 – Wireless Networking**

1. **Overview**

In this experiment you will use 3 laptops configured to communicate in WIFI ad hoc mode and implement a simplified DSR routing protocol. Because it is not easy for us to change the real network layer, out implementation will run at user level.

This handout describes the requirements, how you can set up your computers in ad hoc mode and manage the ad hoc topology, the protocol features you must implement, and hand-in requirements for the project.

1. **Requirement**
2. Teams of 3 students, using 3 linux computers. Please contact the TA if you have trouble forming a three person team that does not have access to at least 3 linux computers.
3. Deadline: Friday, July 29, 2016
4. **Ad Hoc Routing Setup**
	1. **Setting up your laptops in ad hoc mode**

First, disconnect your wireless network. And type the following command.

$ iwconfig wlan0 essid “<essid\_name>” mode ad-hoc

*essid\_name* is the ad-hoc network name, it can be your group name.

Please note that the above command should be typed **twice** and the *essid\_name* value above must be the same on all laptops to make sure that all of them are part of the same ad-hoc network. See the *essid\_name* value by typing:

$ iwconfig wlan0

* 1. **Assign IP address for each computer**

$ ifconfig wlan0 162.105.1.x netmask 255.255.255.0 up

“162.105.1” is the “network identifier” part of the IP address. “x” should be different on each computer on your private network, e.g., 1, 2, and 3.

* 1. **Get the MAC address** **(Hwaddr field) of each computer by typing:**

$ ifconfig wlan0

* 1. **Managing the topology**

If the computers in your experiment are all in the same room, they will all be able to communicate directly with each other, i.e., you will have the topology of Figure 1(a). That is a very boring ad hoc network!

To make it more interesting, we now provide instructions that allow you to block communication between specific computers. This way you can artificially create more interesting topologies that will force some packets to take a multi-hop path. To block computer A and computer C from communicating directly with each other, as if A and C were out of communication range, you can use the following command on Computer A:

$ iptables -A INPUT -m mac –mac--source <MAC address of computer C> -jDROP

and then the following command on Computer C:

$ iptables -A INPUT -m mac –mac--source <MAC address of computer A> -j DROP

You now have the topology in Figure 1(b).



(a)



(b)

**Figure 1: Two topologies in this project. Please set 162.105.1.x as your IP address**.

To move back to the topology in Figure 1(a), you must remove the “block” between A and C, so they can communicate directly again.

$ iptables -L -n --line-number

$ iptables -D INPUT 3

The first command gives you a list of the rules that specify connectivity for the device. The second command deletes the third rule in the list. For example, if you are on Computer A, you should pick the line number that corresponds to the “block” of the link to Computer C.

1. **DSR routing protocol**

Figure 2 shows a block diagram of the DSR router that you must implement. As we discussed in class, it can act as a router and forward packets for other nodes (bottom), and it allows local applications to communicate (top).



**Figure 2: Block diagram of DSR router**

* 1. **Download the code and run the program**

The code for the libraries and data structures shown in orange were written by the TA and is available to you.

After you download the code, you can type ***make*** to compile the code. Note that you should type ***make clean*** before ***make,*** except the first time. Then type**./*ad-hoc*** to run the executable file.

Make sure the program is running on all three computers.

* **How can we send messages to a specific computer?**

Assume that computer A are going to send “123” to computer C. Then we just need to type the IP address of computer C and the message “123” in computer A. The destination IP address and the data will be transmitted to the function **send\_packet** in send\_packet.c/h. The function will do the left things to send the message to computer C.



**Figure 3: computer A**



**Figure 4: computer B**

* 1. **Information of the files**

There is some information for those files.

1. **socket.c/socket.h**

It is used to transmit packets between the three computers using the UDP broadcast to emulate the link-layer. It will submit the packets received to function **receive\_packet** in receive\_packet.c/h. And the function **receive\_packet** has to process the packet and determine what to do next. If two neighbored computer uses ***iptables*** and blocks each other as described above, they cannot receive the UDP broadcast packet from each other. You don’t need to modify this file.

1. **send\_packet.c/send\_packet.h**

As described above, when we need to send data to another computer, the **send\_packet** function would be called. This is what you need to implement.

1. **receive\_packet.c/ receive\_packet.h**

When we received a packet, the **receive\_packet** function would be called. This is what you need to implement. It will be described in detail later.

1. **utils.c/utils.h**

There are some tools for you, such as **send\_unicast**, **send\_broadcast**. You can find the detail in ***utils.h***.

1. **header.h**

The data structure is included here. We use the structure as the following figure.



It is a little tricky and different from the standard. We use the structure here to get rid of the char \* which is a little difficult for some of you.

1. **cache.c/cache.h**

Each time we received a RREP, we would put it into the cache table. If we want to send to that destination again, we can just look up the table instead of sending RREQ. The cache table has been implemented. You can see details in the ***cache.h*** file.

1. **flood.c/flood.h**

When we receive a RREQ packet and we are not the destination, we will broadcast it. So we need some method to know whether we received it before. So we implement this part for you. You can use **get\_unique\_id** to generate a request id for a specified destination and use **check\_new** to check is it a packet we haven’t received. See details in ***flood.h***.

* 1. **Understanding the data structure**

There are Data Packets, RREQ, RREP, RERR in this protocol. Since we do not change the topology when the program is running in this project, which means there are no disconnections, so we only care about the Data Packets, RREQ, RREP. The structures are in the ***header.h***.

* 1. **Sending packets**

When we need to send data to another computer, the **send\_packet** function would be called. You need to first look up the cache table to find the path. If you cannot find the path in the cache table, then you need to discover the route path by sending a RREQ. After the arrival of RREP, add the path into the cache table and send the packet to the destination.

* 1. **Processing the received packets**
		1. **Identifying the packet type**

When we received a packet, the **receive\_packet** function would be called. You need to check what kind of the packet it is, then process it. The following steps tell you how to process those packets.

* + 1. **RREQ**

If node S want to send a packet to node D without knowing the route information, it will send a RREQ. If a computer received a RREQ, it should check if it is the destination and decide what to do next. If it is the destination, it will send a RREP. Otherwise it will forward it and add its address into the packet header. The RREQ should include the source address and the destination and request ID.

* + 1. **RREP**

If a computer receives a RREQ and it is the destination, it will send a RREP to the source address. If a computer received a RREP, it should check if it is the destination and decide what to do next. If it is the destination, it will add the path information into the cache table. The RREP should include the route information.

* + 1. **Data packet**

The data packet should not only contain the data, but also contain the route information. If a computer received a Data packet, it should check if it is the destination and decide what to do next. If it is the destination, it will submit the data to upper layer, while we print the data to the screen in this project. If it is the destination, just forward it using the address list in the packet header.

1. **Hand in**
	1. **requirement**

**a) write the report:** Explain how you implement the protocol and the experiment in detail.

**b) compress your code:** Submit it as the attachment of the email.

* 1. **submit the report**

Send to: **jing.wang@pku.edu.cn**

Attachment format: project 2 name1, name2.zip

The name should be in Chinese.

Please write down your team members and contact information in your report.