CS168 Programming Assignment 2: IP over UDP

Assignment Out:	September 26, 2013
Milestone:	October 04, 2013
Assignment Due:	October 11, 2013, 11:59pm

1   Introduction

In this assignment you will be constructing a *Virtual IP Network* using UDP as the link layer. Your network will support dynamic routing. Each node will be configured with its (virtual) links at startup and support the activation and deactivation of those links at run time. You will build a simple routing protocol over these links to dynamically update the nodes' routing tables so that they can communicate over the virtual topology. The relevant class lectures and textbook will be especially helpful with this part of the project.

This is a 2-person group project. You should find a partner to work with right away, and email TAs to inform us of your pairing. If you are having problems with this (there could be an odd number of people in the class), send mail to the class mailing list, `cs168.2013-14.f@lists.cs.brown.edu`, or ask us. Once the groups are set, you'll be assigned a mentor TA to help you through this project and the next, TCP. TCP will build on this project, so your effort on design will pay off twice.

2   Requirements

Before you start coding, you need to understand what you're doing. It will take a little while to wrap your head around, but once you do, it will seem straightforward, we promise.

There are a two main parts to this assignment. The first is IP in UDP encapsulation, and the design of *forwarding* — receiving packets, delivering them locally if appropriate, or looking up a next hop destination and forwarding them. The second is *routing*, the process of exchanging information to populate the routing tables you need for forwarding.

Your network will be structured as a set of cooperating processes. You might run several processes on a single machine or use separate machines; it doesn't matter because your link layer is UDP.

Files you will need to bootstrap your network are available in `/course/cs168/pub/ip/`. You will write a network topology file (we've supplied two examples) describing the virtual topology of your intended network. After running our script `net2lnx` on the complete topology, you'll have a file for each node that specifies only that node's links. You will run your process, which must be called `node`, for each virtual node, and must accept the name of that node's link file as its first argument on the command line.

3 Implementation

Your nodes will come up, and begin running RIP on the specified links. Each node will also support a simple command line interface, described below, to bring links up and down, and send packets. When IP packets arrive at their destination, if they aren't RIP packets, you should simply print them out in a useful way. In the next assignment, you'll deliver them to your TCP implementation.

3.1 Forwarding

You will use UDP as your link layer for this project. Each node will create an interface for every line in its links file — those interfaces will be implemented by a UDP socket. All of the virtual IP packets it sends should be directly encapsulated as payloads of UDP packets that will be sent over these sockets. You must observe an Maximum Transfer Unit (MTU) of 1400 bytes; this means you must never send a UDP (link layer) packet larger than 1400 bytes. However, be liberal in what you accept. Read link layer packets into a 64k buffer, since that's the largest allowable IP packet. To enforce the concept of the network stack and to keep your code clean, we require you to provide an abstract interface to your link layer rather than directly make calls on socket file descriptors from your forwarding code. For example, define a network interface structure containing information about a link's UDP socket and the physical IP addresses/ports associated with it, and pass these to functions which wrap around your socket calls.

You will design a network layer that sends and receives IP packets using your link layer. The IP packet header is available in $</usr/include/netinet/ip.h>$ as `struct ip`. Those of you not using C/C++ may use `/usr/include/netinet/ip.h` or other sources as a reference for crafting your headers. Although you are not required to send packets with IP options, you must be able to accept packets with options (ignoring the options). You network layer will read packets from your link layer, then decide what to do with the packet: local delivery or forwarding.

You will need an interface between your network layer and upper layers for local delivery. In this project, some of your packets need to be handed off to RIP, others will simply be printed. Next time, you'll be handing packets off to your TCP implementation. These decisions are based on the IP protocol field. Use a value of 200 for RIP data, and a value of 0 for the test data from your send command, described below. We ask you to design and implement an interface that allows an upper layer to register a *handler*[1] for a given protocol number. We'll leave its specifics up to you.

Even without a working RIP implementation, you should be able to run and test simple forwarding, and local packet delivery. Try creating a static network (hard code it, read from a route table, etc.) and make sure that your code works. Send data from one node to another one that requires some amount of forwarding. Integration will go much smoother this way.

---

[1]Suggested C prototypes (where the declaration of interface_t is up to you):
```
typedef void (*handler_t)(interface_t *, struct ip *);
void net_register_handler(uint8_t protocol_num, handler_t handler);
```

3.2 Routing - RIP

The second part of this assignment is implementing routing using the RIP protocol described in class.

You must adhere to the following packet format for exchanging RIP information:[2]

```
uint16_t command;
uint16_t num_entries;
struct {
  uint32_t cost;
  uint32_t address;
} entries[num_entries];
```

command will be 1 for a request of routing information, and 2 for a response. num_entries will not exceed 64 (and must be 0 for a request command). cost will not exceed 16; in fact, we will define infinity to be 16. address will be an IPv4 address.

As with all network protocols, all fields must be sent on the wire in network byte order.

Once a node comes online, it must send a request on each of its interfaces. Each node must send periodic updates to all of its interfaces every 5 seconds. A routing entry should expire if it has not been refreshed in 12 seconds[3]. If a link goes down, then the network should be able to recover by finding different routes to nodes that went through that link.

You must implement split horizon with poisoned reverse, as well as triggered updates.

3.3 Driver

Your driver program, node, will be used to demonstrate all features of the system. You must support the following commands.

interfaces Print information about each interface, one per line.

routes Print information about the route to each known destination, one per line.

down *integer* Bring an interface "down".

up *integer* Bring an interface "up" (it must be an existing interface, probably one you brought down)

send *vip proto string* Send an IP packet with protocol *proto* (an integer) to the virtual IP address *vip* (dotted quad notation). The payload is simply the characters of *string* (as in snowcast, do not null-terminate this).

You should feel free to add any additional commands to help you debug or demo your system, but the above the commands are required. It would be to your advantage to add bandwidth-intensive test commands to help prepare your implementation for TCP.

---

[2]If you are writing in C or C++, consider using flexible array members for allocation of your packet structure

[3]When testing your project, feel free to make these times longer if it assists with using a debugger.

4 Getting Started

We've created a few tools that you can use to help you with your project. They are available in

/course/cs168/pub/ip/

4.1 Scripts

- net2lnx - A tool to convert a .net file into a series of .lnx files that each node can read separately.

- runNode - Takes a .lnx file as input and runs that node, ssh-ing to the remote machine it is specified to run on, if necessary.

- runNodeWin - runNode, but in a different xterm window.

- runNetwork - Given a .net file, starts all nodes that are part of that network. Much more convenient than starting all nodes manually!

4.2 Sample Networks

- AB.net - Simple network with two nodes. It may looks like this:
  node A localhost
  node B localhost
  $A < - > B$

  which tells you the physical location of each node and how they are connected. After running net2lnx on it, you will have something look like:
  A.lnx:
  localhost:17000 10.10.168.73 localhost:17001 10.116.89.157
  B.lnx:
  localhost:17001 10.116.89.157 localhost:17000 10.10.168.73
  which you can feed each node as their link information. For example, "localhost:17000 10.10.168.73 localhost:17001 10.116.89.157" means A has one interface at physical location with ip "localhost" and port 17000, and its virtual ip is 10.10.168.73. It is connected to another interface with virtual ip 10.116.89.157 at localhost:17001.

- loop.net - More complicated network with the following shape:

  ```
  src -- srcR --  short  -- dstR -- dst
          |                  |
          \-- long1 -- long2 -/
  ```

  A useful test for routing is to start the network and make sure src goes to dst through short. Then stop the short node and see what happens.

4.3 C support code

- `parselinks.c parselinks.h` - Functions to parse a `.lnx` file into a lnxlink struct. Non-C users can look at this for reference.

4.4 Utilities for C

We've provided several utility files for C with useful functions in the `util` directory:

- Debugging: `dbg.c dbg.h dbg_modes.h colordef.h`. Print colored debugging messages. You can enable and disable categories of messages based on the environment variable `DBG_MODES`. See `node.c` for an example of how to use them in your code. By default, `runNode` enables all possible debugging messages. If you want to enable only, say, net layer and routing messages, then you can run:

  ```
  ./runNode file.lnx net,route
  ```

  See `dbg_modes.h` for a full list of debugging modes - feel free to add your own!

- IP checksum calculation: `ipsum.c ipsum.h`. Use this function to calculate the checksum in the IP header for you.

- Linked list: `list.h`. See `parselinks.c` for examples on how to create a list, add elements, and iterate through it.

- Hash table: `htable.c htable.h`.

4.5 Reference Implementation

A reference implementation of the project will be released shortly (we'll send an announcement to the email list). Copy `/course/cs168/pub/ip/node` into your project directory, then do:

  ```
  ./runNetwork loop.net
  ```

and watch it run!

5 Getting Help

This project isn't intended to be painful, and you have many resources to help you. Make sure you've read this handout and really understand what we mean when we say that UDP is your virtual network's link layer. Piazza is always a good place to get help on general topics, and the TAs will, of course, be holding TA hours.

Make sure that you work together with your group partner, and try to split the project up so that neither of you has too much to handle. An obvious way to split things up is for one person to implement routing (RIP) and the other to be responsible for everything else (packet forwarding,

send/recv interface, etc), but you can do whatever you feel is appropriate. It will *not* be possible for you to go off into separate rooms, implement your half, and "just hook them up." You should work together, there is a lot that should be designed together. The routing table is the most obvious example.

We request you to use a revision control system such as Git so that you can update each other periodically (commit often, but only when the build succeeds!). We are going to give out github repos to teams. Register accounts with github and email the account names to the tas. These repos are private and you are not allowed to share code with other groups. You can talk to other groups about concepts, algorithms, *etc.*, but each group's code must be their own.

Finally, each group will have a mentor TA. This means that you'll have one of the three TAs as your group's advisor. You'll need to set up an appointment to meet with your mentor TA for some time this week to discuss the project design. Once you've got the ok on this, you should stay in contact with your mentor, who will be grading your project and will be able to explain what the project ultimately should be doing. Your mentor also will do his best to help outside of TA hours, debugging, discussing design, *etc.* Just because your mentor is helping you out, however, doesn't mean that he/she is at your beck and call. Understand that the TA staff is busy too, and while they'll try to help you as much as possible, there may be times when they simply won't be able to.

6   Extra Credit (15%)

You must implement IP fragmentation for extra credit. The MTU for each link should be set as an additional command line interface option as:

mtu *integer*0 *integer*1 Sets the MTU for the link integer0 to integer1 bytes.

7   Handing In and Interactive Grading

Once you have completed the project you should run the electronic handin script
/course/cs168/bin/cs168_handin ip to deliver us a copy of your code. Your mentor TA will arrange to meet with you for your interactive grading session to demonstrate the functionality of your program and grade the majority of it. This meeting will take place at some point shortly after the project deadline. Between the time you've handed in and the demo meeting, you can continue to make minor tweaks and bug fixes (and you should, since it will be the code base for your next project). However, the version you've handed in should be nearly complete since it could be referenced for portions of the grading.

8   A Warning

You should start on this project *now*. We expect all of the projects in CS168 to take the full amount of time we give you. It can be tricky so we want to make sure that you stay on top of it. You have to have your first design meeting with your mentor TA by Oct 4th. For this meeting you should have a clear sense of what your program is going to look like and a complete list of what you don't

understand. We will also ask questions that you should have answers by the milestone. Make no assumptions about what will be asked and tested. Ask questions now if in doubt. Start talking with your partner right away, and get ready to get connected!