pete > courses > CS 431 Spring 23 > Assignment 03

Assignment 3: IP, ARP, ICMP

If any of this is unclear, please get in touch with me ASAP. There are definitely things in the course I want you to struggle with. I absolutely do NOT want you struggling with understanding my expectations.

The goals of this assignment are for you to cement your knowledge of IP, routing, ARP, and ICMP.

The three parts of this assignment build on each other, so do not proceed to Part III until you have finished Part II, and do not proceed to Part II until you have finished Part I. Additionally, Part I of this assignment builds on the `stack.c` code you wrote for Part I of the previous assignment, so make sure it is working correctly before beginning work.

ChangeLog: - 10 Apr 23: added requirements related to IHL

Part I: Route packets from a single interface

Beginning with the `stack.c` program you wrote for the previous assignment, extend it to implement an IP router. The router will simulate multiple interfaces, each attached to a different network. When a frame arrives, if that frame contains an IP packet, the router should route the packet according routes stored in a routing table. For Part I, you may assume that frames will only ever arrive on one of those interfaces, but may need be sent out over any of them.

If a frame arrives destined for the broadcast Ethernet address, your router MUST note it (and then ignore it).

Your route MUST implement a routing table whose contents are easily identifiable and modifiable (ie, so that I can write tests that assume a particular network configuration and only have to change data in your code to work with it).

If a packet is destined for one of the router interfaces (ie, it will not be forwarded), your router MUST print a message to this effect. In future assignments, the packet will be handed off to another component.

Your router MUST implement an ARP cache whose contents are also easily identifiable and modifiable.

If any problems are encountered, your router MAY just drop the packet but it MUST note that it has done so and why. This includes circumstances in which an ICMP error would be returned (eg, TTL exceeded, host unreachable, net unreachable).

When your router receives a packet, it MUST correctly interpret the "IHL" field in the IP header, but it MUST also ignore any options that might be present. **[Added 10 Apr 23]**

You MAY hard-code the following:

- the quantity, IP addresses, MAC addresses, and network parameters for each simulated interface;
- the size and contents of the routing table;
- the size and contents of the ARP cache.

You will need to run one instance of `vde_switch` for each network to which your router is connected. To differentiate these instances, you will need to run each with a different *control file*: this is a file in the filesystem that is used to communicate with that particular switch. Control files typically live in `/tmp` and can be named anything you want (as long as it's unique).

To run `vde_switch` with the control file `/tmp/net1.vde`, run this:

```
$ vde_switch -sock /tmp/net1.vde
```

It will probably be easiest for you to write a shell script that sets up all the switches, rather than running them manually. If you need help with this, talk to me.

To cause your program to connect to this switch, you will need to pass a different second parameter to `connect_to_vde_switch`: instead of `{ "vde_plug", NULL }`, you will need to send `{ "vde_plug", "/tmp/net1.vde", NULL }`.

Here is an example output (which does not include any indication of packets successfully routed: those will be apparent to separate receivers connected to the various simulated switches):

```
$ ./stack
received 146-byte broadcast frame from 77 88 99 aa bb cc
ignoring 1078-byte frame (not for me)
ignoring 96-byte frame (bad fcs: got 0xdeadbeef, expected 0xe88f7195)
ignoring 54-byte frame (short)
ignoring 800-byte frame (unrecognized type)
dropping packet from 1.2.3.4 to 5.6.7.8 (no route)
dropping packet from 1.2.3.4 (bad IP header checksum)
dropping packet from 1.2.3.4 to 6.7.8.9 (no ARP)
dropping packet from 1.2.3.4 (wrong length)
dropping packet from 1.2.3.4 to 7.8.9.a (TTL exceeded)
```

Part II: ARP and ICMP

Modify your implementation from Part I to a) respond to ARP requests and b) send ICMP messages back when certain problems are encountered, described below.

If your router receives an ARP request on the listening interface that corresponds to that interface, it MUST respond with a well-formed response. Any other ARP request MUST be ignored.

If your router receives a packet to be routed, it MUST send the following ICMP messages under the appropriate conditions:

- ICMP TTL Exceeded
- ICMP Network Unreachable
- ICMP Host Unreachable

Part III: Route packets from multiple interfaces

Modify your implementation from Part II to handle packets and ARP requests arriving on any interface.

You MUST implement this without using multiple threads or processes. The `poll(2)` system call will be vital here. `poll` is the modern variant of the venerable `select(2)` system call, and is the standard way for a single-threaded process to handle multiple clients efficiently. (If you want to be ambitious, use the more general-purpose, FreeBSD-specific `kqueue(2)` system call.)

Style requirements

Beyond the function prototypes as required, none. You are, however, *strongly* encouraged to use good style (comments, appropriate names, etc), as it will benefit you greatly when you need to revisit your code throughout the semester. The idea here is to demonstrate why good style is a good thing, not by forcing you to use it, but by forcing you to deal with the consequences if you don't.

I MUST be able to build your router program using the following command:

```
$ gmake stack
```

I MUST be able to run your router program using the following command:

```
$ ./stack
```

Submission Process

None. I will pull from your git repo at 12pm (noon) on Friday, 21 April and consider that your submission. I will review your code and provide feedback that enumerates improvements required to earn credit, upon receipt of which you will have one week to make revisions.

Assessment

Part I is the core requirement, which you must complete to earn a C.

Part II is the intermediate requirement; you must complete both Parts I and II to earn a B.

Part III is the advanced requirement; you must complete all three parts to earn an A.

Last modified: 04/10/2023 12:47:05