Project & Training 3: Networking Project

Gerhard Hassenstein & Christian Grothoff

Berner Fachhochschule

KW 47/2020

Agenda

Ethernet Labs

There are three main Ethernet labs deliverables:

1. Implement a Hub & Switch
2. Implement ARP
3. Implement a Router

The Hub is not graded and serves as a quick warm-up.

Team

Suggested are teams of 4 students:

- One Scrum-Master (documentation, planning)
- One for main product deliverable
- Two testers

This is a *suggestion*, not a requirement!

Process

Three 2-week sprints:

- ▶ Plan algorithms, data structures and testing
- ▶ Implement
- ▶ Unit-test
- ▶ Integration test

$\Rightarrow$ Version in Git at submission time is graded!

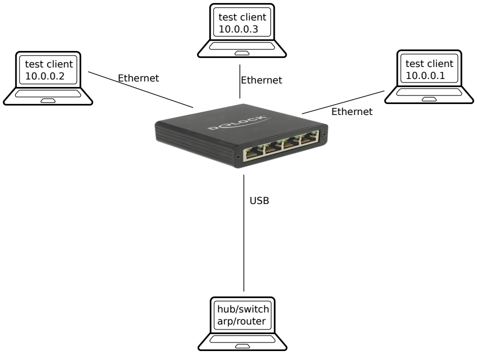
Virtual Hardware

`https://gitlab.ti.bfh.ch/demos/vlab`

Hardware

- There are two custom USB-to-4x-Ethernet adapters for each team in N.111 (below secretary's office) in the lab.
- You can open the door with your BFH card. Please add your name to the list with the number of the adapter taken.
- Also take single USB-Ethernet adapters if your notebook/PC does not have an Ethernet port. You may also take an Ethernet cable if needed.
- You must bring everything back after the final submission deadline. When you have returned the adapter, you may cross your name off the list again.

Suggested setup



Skeleton

Your Gitlab team repository should have been provisioned with a skeleton that is a starting point. It includes:

hub.c	Template for the `hub` project. Add 3 lines to get a working hub!
switch.c	Template for the `switch` project.
arp.c	Template for the `arp` project.
router.c	Template for the `router` project.
Makefile	Build system. Modify as needed.
network-driver.c	Completed driver to allow you to send and receive Ethernet frames. You do not need to modify this code!

Running the network-driver

- LAN driver provided in C in Git (`glab/network-driver.c`)
- Launch driver with list of names of physical network device (i.e. “lan0”) followed by “-” followed by the command to run your program:

  ```
  $ sudo network-driver eth0 eth1 eth2 - ./my-hub eth0
  ```

  The network devices MUST be passed twice: once to the network-driver as arguments, and once to your program (`hub`, `switch`, `arp`, `router`)!
- Driver will pass received frames to your `stdin`
- Driver will read frames from your `stdout` and pass to network
- Driver will not touch `stderr`, you can use `stderr` for logging
- Terminate driver via signal (kill) or closing `stdout`

Your code can be in *any* language!

The Driver I/O Format

- First 6 bytes written by driver to your `stdin` are the HW MACs for each of the network interfaces (in the same order).
- Henceforth, the message format is 16-bit length prefix (in big endian), followed by 16-bit interface identifier, followed by Ethernet frame (destination MAC, source MAC, etc.).
- To send frames, also use 16-bit length prefix followed 16-bit interface identifier, followed by Ethernet frame.
- Interace number 0 is **reserved** for interacting with the console.
- You must set the source MAC correctly (in particular for `arp`/`router`)!
- Some hardware may not support using source MACs other than your HW MAC. If you do not use the provided equipment, check that you have hardware that supports sending with arbitrary MACs!

Skeleton: Helper files

Other code in the Gitlab team repository:

loop.c	Shared logic for all programs. Functions you may use, but do not need to modify.
print.c	Replacement for `printf()` given that `stdout` is for Ethernet frames and MUST NOT be used for program output.
crc.c	Internet checksum. Feel free to use.
glab.h	Packet format for the interaction with the network-driver.

You do NOT have to modify this code, but it may be useful to understand it. You MUST re-implement this logic if your project is in languages other than C.

Suggested Strategy

- Understand what a hub/switch/arp/router really has to do.
- Plan your data structures first. The algorithms are always trivial. Use simple tables (except for routing).[1]
- For testing, write a program that pretends to be the network driver. Remember the `shell` project from CS Basics. Use `dup`, `fork` and `exec` to run your main program with `stdin`/`stdout` being controlled by your test harness.[2]
- Perform compatibility tests with real hardware once above tests work.

Tests and main program do **not** have to be in the same language.

[1] Until the `router`, you do not need `malloc()` at all!

[2] `java.lang.ProcessBuilder` can also be used.

Test requirements

The quality of your tests will **also** be graded.

- Make sure your tests are run via the `make check` target.
- The tests should succeed by returning 0, and fail with non-zero.
- `make check` MUST NOT create binaries like `switch`, `arp` or `router`.
- If you write unit tests for individual functions, please put them under a different target, like `make tests`. Those will NOT be graded.

Why? We will run your test suite against our reference implementation and implementations of other students, and vice-versa!

Test starting point

```
int meta (int argc, char **argv) {
  int cin[2], cout[2]; pipe (cin); pipe (cout);
  if (0 == (chld = fork ()) {
    close (STDIN_FILENO); close (STDOUT_FILENO);
    close (cin[1]); close (cout[0]);
    dup2 (cin[0], STDIN_FILENO);
    dup2 (cout[1], STDOUT_FILENO);
    execvp (argv[0], argv);
    printf (stderr, "Failed to run binary ‘%s’\n", argv[0]
    exit (1);
  }
  close (cin[0]); close (cout[1]);
  child_stdin = cin[1]; child_stdout = cout[0];
  // send MACs, run test, cleanup
  kill (chld, SIGKILL);
}
```