Project & Training 3: Networking Project

Hansjürg Wenger & Christian Grothoff

Berner Fachhochschule

KW 47/2022

Agenda

Networking project

There are three main sprints:

1. Implement an Ethernet switch
2. Implement the ARP protocol
3. Implement an IPv4 router

Today, we will implement a `hub`, which serves as a quick warm-up.
The `switch` is **not required** for students in BTI 3022.

Networking project: repeating students

There are **special rules** for students who failed the course previously:

1. Only one project: **vswitch**
2. Special team size: **two students max**
3. Different passing threshold: **24 points (team of 2), 20 points (team of 1)**
4. Only one deadline: **same as Sprint 3 for regular students**

Team
Suggested

are teams of 4 students. This is a *suggestion*, not a requirement! If your team is smaller, the required thresholds for passing the course will be reduced.
Teams have been pre-assigned, if there are problems within your team, do talk to the professors who will try to find a solution.

Process

Three 2-week sprints:

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

⇒ Version in Git at submission time is graded!

Virtual Hardware

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

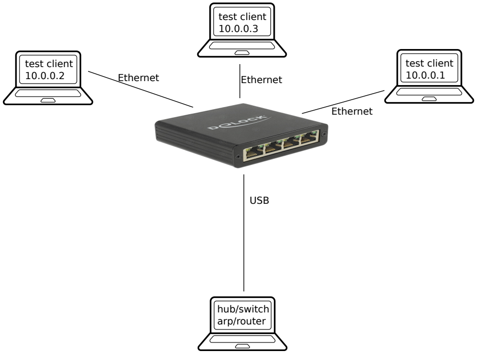
Username: root, password: (given via Mumble)

```
# ./network-driver ens4 ens5 ens6 ens7 - ./switch ens4 ens5 ens6 ens7
```

Hardware

- There are two custom USB-to-4x-Ethernet adapters for each team in the lab.
- 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-XXX` target(s).
- The tests should succeed by returning 0, and fail with non-zero.
- `make check-XXX` 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 as well as buggy implementations!

Grading of your code

- If you code does not compile with `make`: 0 points. No discussions.
- If you code exhibits undefined behavior and thus works on your system but not ours: 0 points. No discussions.
- We run tests against your implementation. Passing our tests gets you points.
- We run your test suite against our correct and buggy (`bugX-XXX`) implementations — you get points if your test suite finds our bugs **while passing our correct reference implementation**. Bonus points will be awarded for teams that find previously unknown bugs in the reference implementation within the scope of the specification.[3]
- Test your tests against the provided correct and buggy reference implementations!

[3] Contact us, if we confirm it is a bug in the reference implementation, you get your bonus.

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);
}
```

Git

The slides and other materials related to the lecture and the project are on `https://gitlab.bfh.ch/`. You should have obtained the links from the official Moodle page of the course. You must also use Git for your software development. Make sure to only share the GitLab repository with your team and the professors. You are responsible that your solution is only submitted by your team.
When you have questions about your code, it is helpful if the current version of the code is available to the professor on GitLab for inspection.

Cryptpad

`https://pubcryptpad.pep.foundation/` can be used for collaborative editing, be it during lectures or your team work. A cryptpad has the advantage that the data is encrypted before it is transmitted to the server. Thus, unlike proprietary Web-based text editors offered by major tech companies, this Free Software solution is compatible with privacy regulation.
The key for decryption is in the URL, so be careful with whom you share the link to the pad! When used on-the-spot in lectures, the professor will share a link to the respective pad using the Mumble chat.

Jitsi

BFH's official Jitsi server **bfh.meet.switch.ch** will be primarily used for voice conferencing, including interactive help with the project. The server is running 24/7, so you are welcome to use it at any time.
Jitsi can be used to record sessions. You MUST have the consent of everyone in the room before recording. Lectures MUST NEVER be recorded.

Scheduling a session

To receive support via Jitsi, you must first send an e-mail to the respective instructor asking for help. Support will be provided only during the official support period. The instructor will reply with a URL and time.
You may ask on the day itself.