BTI 3021: Networking Project - Sprint 2

Christian Grothoff

1 Introduction

For this sprint you will write a precursor to an IP router. This precursor system is to realize the ARP protocol functionality of an IP device.

While the driver and skeleton you are given is written in C, you may use *any* language of your choice for the implementation (as long as you extend the `Makefile` with adequate build rules). However, if you choose a different language, be prepared to write additional boilerplate yourselves.

How the ARP protocol works is expected to be understood from the networking class. If not, you can find plenty of documentation and specifications on the Internet.

The basic setup is the same as in the first sprint.

1.1 Deliverables

There will be four main deliverables for the sprint:

Implementation You must **implement the ARP protocol**. Your implementation must answer to ARP requests, and also itself have the ability to issue ARP requests and to cache ARP replies. For this, you are to extend the `arp.c` template provided (or write the entire logic from scratch in another language).

Testing You must implement and submit your own **test cases** by *pretending* to be the network driver (see below) and sending ARP requests or command-line inputs to your program and verifying that it outputs the correct frames. Additionally, you should perform and **document** *interoperability* tests against existing implementations (i.e. other notebooks from your team to ensure that your ARP protocol implementation integrates correctly with other implementations).

Design and documentation You must design the main data structure (ARP table), and create a **comprehensive test plan**.

All deliverables must be submitted to Git (master branch) by the submission deadline announced on Moodle.

1.2 Functionality

The goal is to implement a program `arp` that:

1. Watches for ARP queries on the Ethernet link
2. Responds with ARP responses for your own IP address
3. Provides an ARP cache so that it does not have to repeatedly make ARP requests to the network for MAC addresses it already knows.
4. If the user just enteres "arp" without an IP address, you should output the ARP table in the format "*IP* -> *MAC* (*IFNAME*)" with one entry per line, i.e.

   ```
   10.54.25.15 -> 28:c6:3f:1a:0a:bf (eth1)
   ```

   (note the leading "0" digit in `0a`).
5. Reads IPv4 addresses from `stdin` (in human-readable format). The interactive command syntax should be "arp *IP-ADDR IFNAME*" (i.e. each line is to be prefixed with the letters "arp ", followed by the IPv4 address and the name of the network interface).
   (a) If the `IP-ADDR` is in the ARP cache, the program must immediately output the associated *MAC*:

   ```
   28:c6:3f:1a:0a:bf
   ```

   NOTE: the format requirement was clarified recently.
   (b) If the `IP-ADDR` is **not** in the ARP cache, the program should *only* issue the ARP query for those IPv4 addresses.
6. If an ARP response is received (at any time), the ARP cache must be updated accordingly (but the MAC address does not need to be output at that time, even if there was an explicit command-line request for this address before).

Your programm should be called with the name of the interface, the IP address[1] for that interface and the network mask. Example:

```
$ network-driver eth0 eth1 - \
  arp eth0[IPV4:192.168.0.1/16] eth1[IPV4:10.0.0.3/24]
```

This means `eth0` is to be bound to 192.168.0.1 (netmask 255.255.0.0) and `eth1` uses 10.0.0.3 (netmask 255.255.255.0).

The file `arp.c` provides a starting point where the parsing of the command-line arguments and the `stdin`-interaction have been stubbed for you.

[1] You may support multiple IPs per network interface, using a comma-separated list of IPs and network masks, but this is not required.

1.3 Testing

For your test plans, please make sure to supply the following information for each test:

- Test Case ID — This field uniquely identifies a test case.
- Test case Description/Summary — This field describes the test case objective.
- Pre-requisites — This field specifies the conditions or steps that must be followed before the test steps executions.
- Test steps — In this field, the exact steps are mentioned for performing the test case.
- Expected Results — successful test
- Author — Name of the Tester.
- Automation — Whether this test case is automated or not.
- Status — Pass/fail with date and code version.

Note that in addition to automated tests similar to the `public-test-switch` you **should** probably do integration tests in a real network (evaluating manually with ping, wireshark, etc.). This will help you find problems in your implementation and ensure that your understanding of the network protocols is correct. Such integration tests should be documented and will be graded as part of the technical documentation.

1.4 Design and Documentation

The final documentation should include:

- Title page with group number and names/kurzel of all participants.
- Table of contents
- Introduction text (mention objectives and targeted audience)
- Data structure (ARP table)
- Test strategy
- Product backlog and sprint backlog (Scrum)
- Glossary
- Bibliography

Text throughout must explain methods as well as justifications for choices made (why not other choices). The organisation should clearly indicate sections and sub-sections. The language should have a consistent voice with a fairly consistent register (semi-formal). The goal is clarity with minimal repetition. Concise language techniques (relative clauses, linking words...) are expected to be employed without forfeiting clarity. You should also use appropriate graphical representations and pay attention to how you visualize code. Algorithms and data structures should be presented in diagrams or with pseudo-code, but NOT in C syntax.

Documentation must be written in Markup and included in the Git, **but** must also be submitted by the deadline as a PDF to the respective Moodle assignment folder for grading by the English professors.

2 Grading

All deliverables must be submitted to Git (master branch) by the submission deadline announced on Moodle. The PDF files generated from the Markup documentation must **additionally be submitted via Moodle**.

You are expected to work in a team of **four students**. If needed, Prof. Grothoff may permit the creation of a team of 5 students or teams of 3 students. Each team is responsible for dividing up the work and coordinating as needed (SCRUM).

You can earn 16, 15 and 19 points in the three networking sprints, for a total of 50 points. You need **37 points** to pass the networking component of BTI 3021.

If your team is **for good reasons** smaller than four students, the passing threshold will be lowered by **2 points** per "missing" student per sprint. So a student going alone would still need **19 points** to pass.

Teams may **request** changes to team membership at the end of a sprint, but must provide a **justification** to the course coordinator, who may approve or decline the request.

2.1 ARP grading

You get points for each of the key deliverables:

Technical documentation (A&D, test plan)	2
Correct implementation	8
Comprehensive test cases	3
Quality of English in documentation	2
Total	15

Partial points are awarded.

You can earn 16, 15 and 19 points in the three networking sprints, for a total of 50 points. You need **37 points** to pass the networking component of BTI 3021.

2.1.1 Correct implementation

- 3 points for passing public test cases
- 5 points for passing secret test cases (see Section 3)

2.1.2 Comprehensive test cases

- 0 points if public reference implementation (see Section 3) fails test cases, **otherwise**
- 1 points for failing public buggy implementation (see Section 3), plus
- 2 points for finding bugs in secret buggy implementation(s)

If you believe you have found a bug in the provided reference implementation, you are encouraged to discuss it with the instructor. If you have found an actual bug in the reference implementation (that is within the scope of the assignment), you will be awarded a **bonus point** per acknowledged bug.

2.1.3 Technical documentation

- 1 point for good description of data structure
- 1 point for good description of test cases

Partial points may be awarded.

2.1.4 Quality of the English in the documentation

- 1 point for organisation / comprehension / consistency
- 1 point for correct grammar / spelling / punctuation

Partial points may be awarded.

3 Provided binaries

You are provided with several binaries:

public-test-arp A public test case, run using "./public-test-arp ./arp" to test your switch. Returns 0 on success.

public-arp Reference implementation of the "arp".

public-bug-arp Buggy implementation of a "arp".

4 Required make targets

You may modify the build system. However, the final build system must have the following `make` targets:

all build all binaries

clean remove all compiled files

arp build your "arp" binary from source; the binary MUST end up in the top-level directory of your build tree.

test-arp build your "test-arp" program from source; the program MUST end up in the top-level directory of your build tree.

check-arp Run "test-arp" against the "arp" binary.

For grading, we will basically run commands like:

```
GRADE=0
$ make test-arp
$ cp public-bug-arp arp
$ make check-arp || GRADE=‘expr $GRADE + 1‘
$ cp private-bug-arp arp
$ make check-arp || GRADE=‘expr $GRADE + 2‘
$ cp public-arp arp
$ make check-arp || GRADE=0
echo "Test grade: $GRADE"
```

You must thus make sure the build system continues to create programs in the right (top-level) location!