BTI 3021: Networking Project - Sprint 3

Christian Grothoff

1 Introduction

For this sprint you will write an IP router, building upon the results from your sprint.

While the driver and skeleton you are given is written in C, you may again 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 an IP router 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 two sprints.

1.1 Deliverables

There will be three main deliverables for the sprint:

Implementation You must implement an IPv4 router. Your implementation must answer to IP packets, and route them. For this, you are to extend the `router.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 IP packets 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 IP router implementation integrates correctly with other implementations).

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

1.2 Functionality

Implement `router` which routes IPv4 packets:

1. Populate your routing table from the network interface configuration given on the command-line using the same syntax as with the `arp` program.

2. Use the ARP logic to resolve the target MAC address. Simply drop the IP packets for destinations where the next hop's MAC address has not yet been learned, and issue the ARP request to obtain the destination's MAC instead (once per dropped IP packet).

3. Make sure to decrement the TTL field and recompute the CRC.

4. Generate ICMP messages for "no route to network" (ICMP Type 3, Code 0) and "TTL exceeded" (ICMP Type 11, Code 0),

5. Support the syntax `IFC[RO]=MTU` where `MTU` is the MTU for IFC. Example: `eth0=1500`. Implement and test IPv4 fragmentation (including *do not fragment*-flag support), including sending ICMP (ICMP Type 3, Code 4).

6. Support dynamic updates to the routing table via `stdin`. Base your commands on the `ip route` tool. For example, "route list" should output the routing table, and "route add 1.2.0.0/16 via 192.168.0.1 dev eth0" should add a route to `1.2.0.0/16` via the next hop `192.168.0.1` which should be reachable via `eth0`. Implement at least the `route list`, `route add` and `route del` commands. The interface-specific (connected local network) routes that are added upon startup from the command-line must not need to be `del`etable.

The output of your routing table should have the following format:

```
192.168.0.0/255.255.0.0 -> 1.2.3.4 (eth0)
```

Use 0.0.0.0 if there is no next hop (the target host is in the connected LAN on the specified interface). You may print the routing table in any order. Do include locally connected networks.

Routing table entries for locally connected networks MUST NOT be configured explictly (via "route add") but must be automatically created when your router starts (from the command-line arguments). You do not have to support removal of those routing table entries.

Note that your implementation must realize following functions of a router:

- Basic IP handling (TTL, ICMP, Checksum)
- Forwarding and routing
- Address resultion and caching
- IP fragmentation

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 (routing table)
- Routing algorithm (use UML state diagram)
- Fragmentation algorithm (use UML state diagram)
- 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 Router grading

You get points for each of the key deliverables:

Correct implementation	10
Comprehensive test cases	4
Technical documentation (A&D, test plan)	3
Quality of English in documentation	2
Total	19

2.1.1 Correct implementation

- 3 points for passing public test cases
- 7 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**
- 2 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 switching algorithm
- 1 point for good description of test cases

Partial points may be awarded.

2.1.4 Quality of the English in the documentation

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

Partial points may be awarded.

3 Provided binaries

You are provided with several binaries:

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

public-router Reference implementation of the "router".

public-bug-router Buggy implementation of a "router“.

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

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

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

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

For grading, we will basically run commands like:

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

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