MyvSwitch — Subject

version **#1.0**

CRI `<tickets@cri.epita.fr>`

Contents

[*]. https://intra.assistants.epita.fr

Obligations

*Obligations are **fundamental** rules shared by all subjects. They are non-negotiable and to not apply them means to face sanctions. Therefore, do not hesitate to ask for explanations if you do not understand one of these rules.*

Obligation #0: **Cheating**, as well as sharing source code, tests, test tools or coding-style correction tools is **strictly forbidden** and penalized by not being graded, being flagged as a cheater and reported to the academic staff.

Obligation #1: If you do not submit your work before the deadline, it will not be graded.

Obligation #2: Your submission repository must be **clean**. Except for special cases, which (if any) are **explicitly** mentioned in this document, an *unclean* repository may contain:

- binary files;[1]
- files with inappropriate privileges;
- forbidden files: `*~`, `*.swp`, `*.o`, `*.a`, `*.so`, `*.class`, `*.log`, `*.core`, etc.;
- a file tree that does not follow our specifications (see *Technical constraints*).

Obligation #3: All your files must be encoded in ASCII or UTF-8 without BOM.

Obligation #4: When examples demonstrate the use of an output format, you must follow it scrupulously.

Obligation #5: Your code should compile with the flags "`-std=c11 -pedantic -Werror -Wall -Wextra`".

Obligation #6: If a function, a command or a library is not *explicitly* forbidden, it is **authorized**. If you use a forbidden symbol, your work will not be graded.

Advice

- ▷ Read the *whole* subject.
- ▷ If the slightest project-related problem arise, you can get in touch with the CRI.
  Post to the dedicated **newsgroup** (with the appropriate **tag**) for questions about this document, or send a **ticket** to **<tickets@cri.epita.fr>** otherwise.
- ▷ In examples, `42sh$` or `42sh#` is our prompt: use it as a reference point.
- ▷ Do **not** wait for the last minute to start your project!

1. If an executable file is required, please provide its sources **only**. We will compile it ourselves.

Project data

Instructors:

- Kévin Sztern `<kevin.sztern@epita.fr>`
- Marin Hannache `<marin.hannache@epita.fr>`

Dedicated newsgroup: `cri.projets` with `[VSW]`

Members per team: 1

1 MyvSwitch: a userland virtual Ethernet switch

1.1 Objectives

An Ethernet switch (or bridge) is a layer 2 appliance, central to the deployment of an Ethernet based *Local Area Network* (*LAN*). All machines connected to the switch are part of the LAN, and the switch forwards the frames incoming in from host machines (or other switches!) to the correct destination.

For this subject, you will build a **virtual** switch, which connects **virtual** machines via **virtual** Ethernet cables to form a **virtual** local network. This is similar to *Linux bridges* or the *Open vSwitch* project, with the exception that those products are implemented partly in kernel space.

If you do this project seriously, you will learn a great deal about Ethernet, how a switch works and its advanced configuration options, along with the basics of network virtualization. You should expect a lot of research and reading if you want a decent switch!

1.2 Technical constraints

Files to submit:

— `./Makefile`
— `./README`
— `./src/*`
— `./tests/*`

Forbidden library: pcap

Makefile: The makefile should define at least the following targets:

— **all**: Produces the myvswitch binary
— **tests**: Runs your test suite
— **clean**: Removes compilation byproducts

Guidelines:

— You should respect the “C11” standard of the C programming language.
— You should respect the EPITA coding style.
— Your project must compile and run correctly on Arch Linux.
— Your virtual switch must free all its allocated memory and close all its opened file descriptors.
— You must not strip your program, nor link it statically.
— You must use the Git versioning tool.
— You should comment your code when appropriate.
— You must not include any code *not* written by you, unless permitted beforehand by an instructor.
— Your git repository must not contain extraneous authors, that is to say `git shortlog -s` must display your name only.
— Your *myvswitch* binary **must** be generated in the same directory as the *Makefile*.

2 Getting started

2.1 How to test your implementation

2.1.1 Network namespacing

Since we want to simulate multiple hosts, we need some sort of virtual machine or container. Using a full-blown solution is a bit overkill though, so we're only using the *namespaces* feature of the Linux kernel to represent our virtual hosts, in particular the *network namespaces*. Namespaces are actually how container solutions like *Docker* and *LXC* are implemented, so this is very similar to a real-world application. See the man-pages `namespace(7)` and `ip-netns(8)` for more information.

Then, we need some sort of a virtual cable to plug the host to the switch. This is easily achieved with *veth pairs*, which can be created with `ip-link(8)`:

```
42sh# ip link add veth-a type veth peer name veth-b
```

This command will create two new virtual interfaces, `veth-a` and `veth-b`. Every packet sent to `veth-a` will appear in `veth-b`, and vice-versa. Of course, each end of the pair can be in a separate network namespace!

2.1.2 The CRI wrappers

Creating network namespaces and veth pairs both need root permissions (more precisely the `CAP_NET_ADMIN` capability), so we provide you with a set of shell script with the appropriate permissions on the school computers:

— "`ns-init NAME`" to create a new namespace.
— "`ns-add-if NAME NS1 NS2`" to add a new virtual interface pair in two namespaces.
— "`ns-exec NS CMD [ARG1 [ARG2...]]`" to execute a command in the specified namespace.
— "`ns-del-if NAME NS`" to delete a previously created virtual interface.
— "`ns-destroy NAME`" to delete a previously created namespace.

Since these commands need to be run with root privileges, you must execute them with `sudo` to get them to work:

```
42sh$ sudo ns-init sw host1 host2
Namespace 's2' created.
Namespace 'host1' created.
Namespace 'host2' created.
42sh$ sudo ns-add-if p1 sw host1
Interface 'p1-1-sw' created in namespace 'sw'.
Interface 'p1-2-host1' created in namespace 'host1'.
42sh$ sudo ns-add-if p2 sw host2
Interface 'p2-1-sw' created in namespace 'sw'.
Interface 'p2-2-host2' created in namespace 'host2'.
42sh$ sudo ns-exec sw ./myvswitch p1-1-sw p2-1-sw
```

After executing `ns-add-if` you will end up with *two* new interfaces, each in the specified namespace:

```
42sh$ sudo ns-exec sw ip link show
[...]
4: p1-1-sw@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP␣
↪mode DEFAULT group default qlen 1000
    link/ether 4e:8d:8c:2f:72:ca brd ff:ff:ff:ff:ff:ff link-netns host1
[...]
42sh $ sudo ns-exec host1 ip link show
[...]
5: p1-2-host1@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP␣
↪mode DEFAULT group default qlen 1000
    link/ether e2:4f:45:8b:56:1a brd ff:ff:ff:ff:ff:ff link-netns sw
```

Be careful!

Be careful during your tests: you **must** send the packets from another namespaces than the one where your switch is residing, otherwise the kernel might try to get clever and capture it before you have a chance to see it in your switch!

2.2 What you will need to learn on your own

Enough hand-holding, the following concepts will have to be researched by yourself!

— Network programming in Linux is done through the socket interface, some reading on that could be a good start. Try the famous *Beej's guide*, freely available online.
— Internet sockets are fine and dandy for TCP and UDP, but your switch needs to operate on the lower level link-layer. Maybe the `socket(7)` has information you need on an appropriate protocol family?
— You will need to wait for data on multiple sockets at a time. Could `epoll(7)` hold the answers?
— Don't know where to start? Looking up how Linux implements its own bridge could give you some pointers.

3 Required features

Those features are the minimum requirements for your program to be called a switch. As such, implementing them is mandatory.

They're not enough to make it a *good* switch, though. If you only implement the basic features, you can only expect a slightly above average mark.

3.1 Threshold 1 - Unidirectional forwarding

For this threshold, the goal is to have your socket programming working, without switching magic for now.

The topology for this threshold is extremely simple: only two ports are connected to the switch, and only the first sends frames to the second one. All you have to do is receive the frames from the first port and send them unmodified to the second port.

Your switch must accepts as parameters the two interfaces on which it will operate, and simply wait for data from the first one.

3.2 Threshold 2 - Frame flooding

Your switch should now accept an arbitrary number of interfaces as parameters, each representing a different port.

To complete this threshold you have to find a way to wait for frames on multiple sockets at once. Once a frame is received, you have to send it to all ports except the one from which it came. This behavior is called *flooding*.

3.3 Threshold 3 - Accounting

Time to manipulate the frames a bit. For each frame received, you should print information about it (which port it came from, source and destination MAC addresses, and its *EtherType*) before sending it on its merry way.

Just to make you dig a bit more on *EtherTypes*, we ask you to print its name in addition to its value if it's a common type. We consider the following to be common types: `IPv4`, `IPv6`, `ARP`, `LLDP`, `802.1Q` and `802.1ad`.

Additionally, you should maintain counters for each port (stuff like frames and bytes sent/received). This is very useful for debugging.

Be careful about the byte order of stuff you read! Not everyone on the Internet interprets bytes the same way your machine does. Look up `byteorder(3)` for more information...

3.4 Threshold 4 - Frame switching

What you have right now is more of a *hub* than a switch. That is, every frame received is flooded to all other ports without discrimination, which is not a very good idea in real life.

One of the defining characteristics of a *switch* is the *Forwarding Information Base* (*FIB*), an association table between MAC addresses and the ports on which they live.

Note that you must not pre-configure the FIB before launch: your switch has to **learn** the association by inspecting the source MAC address of the frames it receives. If the destination is not yet known, simply fall back to the flooding behavior.

Of course, *broadcast* and *multicast* frames should still be flooded in all cases (You read that correctly: multicast frames are flooded just like broadcast frames).

3.5 Threshold 5 - CLI and configuration

By adding a CLI and a configuration file, you will be able to implement the other features as most of them require some way of altering the way ports handle frames.

You are free to implement the CLI however you like (though you should really use the libreadline "alternate interface" to preserve your sanity), but we encourage you to read up on the documentation of commercial switches for inspiration (maybe not Cisco, since their CLI has been made intentionally cryptic to sell overpriced certifications).

As a minimum for this threshold, we require the following features:

- A way to show the values of the various counters.
- A way to enable and disable threshold 3 output (defaulting to disabled).
- A way to bring ports up or down dynamically.
- A way to show the content of the switch FIB.
- A way to save the configuration in a file.
- A way to load the configuration from a file.

Tips

No need to get fancy with the configuration file: a collection of lines to be executed on startup as if they were entered on the CLI works fine.

4 Expected features

Expected features are the first steps towards an industry-grade switch, and should be your next objective after your basic features are done. In short: this is where the fun begins.

4.1 (very easy) Port mirroring support

Port mirroring, sometimes called *SPAN* (*Switch Port ANalyzer*) or *RAP* (*Roving Analysis Port*), is a common network monitoring technique for switches. Its principle is very simple: in addition to normal forwarding, send an extra copy of unicast frames to a designated port (typically, to a network monitoring software). There are three levels of increasing difficulty for this feature:

1. Augment the CLI to designate a port as an analyzer port. Every unicast frame from all other ports gets forwarded to this analyzer port in addition to normal forwarding.
2. Augment the CLI to also designate ports as to be analyzed. Only forward the copy to the analyzer port if the source port is configured to do so.
3. (When you have VLANs) Augment the CLI to configure the analyzer port to receive a copy of all frames entering a specific VLAN through any port.

You should never mirror a frame back to the port it came from. You should not modify the mirrored frames, even to add or remove a VLAN tag.

4.2 (very easy) 802.1Q (VLAN) support

Support of “Dot1q” VLANs is the first thing to look for in any halfway decent switch. If your switch at home doesn’t support even basic VLANs, throw it in the garbage or sell it for crack.

There are two major nomenclatures for VLAN configuration: *access/trunk* (for a *port* based approach) for Cisco/Arista and *tagged/untagged* (for a *frame* based approach) for others like HPE. Which nomenclature you choose, or even inventing a new one of your own, is entirely up to you: pick the one that makes the most sense to you, they are functionally equivalent.

Whichever you choose, we expect the same flexibility as real-world switches, and the ability to setup basic topologies like the famous “router on a stick” topology:

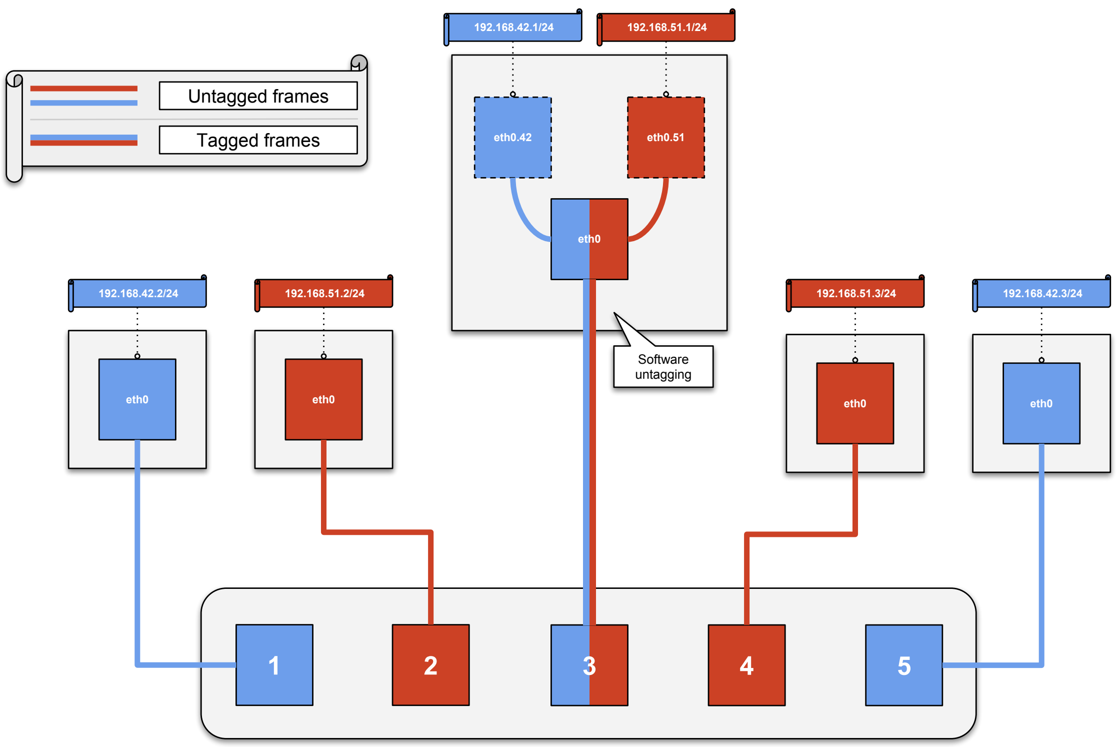

Fig. 1 – “Router on a stick”

Note that the router receives tagged frames from a trunked port. Then, it creates two more virtual interfaces, one for each possible tag on the incoming frame. The frame will be untagged and sent to the appropriate interface, each one with an appropriate IP address (no IP address on the tagged interface!). You can create this kind of interfaces with the following command:

```
42sh# # Assuming the tagged interface is called "veth"
42sh# ip link add name veth.42 link veth type vlan id 42
42sh# ip link add name veth.51 link veth type vlan id 51
42sh# ip addr add 192.168.42.1/24 dev veth.42
42sh# ip addr add 192.168.51.1/24 dev veth.51
42sh# ip addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: veth: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 1a:8d:89:54:2a:34 brd ff:ff:ff:ff:ff:ff
3: veth.42@veth: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group␣
↪default qlen 1000
    link/ether 1a:8d:89:54:2a:34 brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.1/24 scope global veth.42
       valid_lft forever preferred_lft forever
4: veth.51@veth: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group␣
↪default qlen 1000
    link/ether 1a:8d:89:54:2a:34 brd ff:ff:ff:ff:ff:ff
```

(continues on next page)

(continued from previous page)

```
    inet 192.168.51.1/24 scope global veth.51
       valid_lft forever preferred_lft forever
```

5 Additional features

Additional features are just that: additional stuff you can add to your switch so you can learn more.

Those features are sorted by difficulty, and shouldn't be that hard once you get the hang of it.

5.1 (easy) 802.1ad (QinQ) support

QinQ is an amendment to the 802.1Q standard which allow for multiple VLAN headers (usually just called "tags") stacked on top of each other on a single frame. A common use case is giving a VLAN to a customer, who will in turn have multiple separate VLANs of its own.

Not very complicated once you have basic VLANs working, just read up on the standard and you'll be fine.

5.2 (easy) VLAN mapping support

Sometimes you need to interconnect multiples networks from two entities which does not share the same ID for their VLANs (after a merger and acquisition for instance).

To avoid renumbering the whole network infrastructure, you can use switches performing *VLAN mapping* (or *VLAN translation*) at the network edges to transparently handle this kind of disparity.

To implement this feature, you need to augment your CLI in order to be able to express that specifics VLAN IDs comming from designated ports need to be changed to specified others IDs.

5.3 (medium) 802.1AX (LACP) support

The *Link Aggregation Control Protocol* allows you to combine multiple links together, either for bandwith sharing, load balancing, or resiliency purposes. This is mostly known as *bonding* in common parlance. You can be sure that any critical link in a datacenter is bonded.

LACP is a negotiation protocol: frames with a specific structure called *LACPDU* are exchanged between the peers to setup the aggregation.

LACP is VLAN-independent, so LACPDU should always be sent untagged.

Nota bene: LACP was initially standardized in 802.3ad, but was moved to 802.1AX in 2008.

6 Advanced features

6.1 (hard) 802.1ak (MVRP) support

Multiple VLAN Registration Protocol is described in an amendment to the 802.1Q standard. It is designed to ease VLANs configuration in network based on multiple switches.

The idea is to tag the VLAN only on two ports across any number of switches and let the protocol negotiate with the intermediary switches to tag the uplinks accordingly.

MVRP is obviously VLAN-independent, so MRPDU should always be sent untagged.

6.2 (hard) 802.1D (STP) support

The *Spanning Tree Protocol* is designed to detect and remove loop in a switched network, making its topology a *spanning tree*, hence the name.

Loops are highly dangerous in a switched network, since broadcast are sent to every port, a loop ends up duplicating these frames indefinitely, saturating every port of the *entire network* in just a few seconds.

STP is VLAN-independent, so BPDU should always be sent untagged. For a VLAN-aware spanning tree protocol, you should look up *MSTP*.

6.3 Other ideas

A real-life entreprise-grade switch would also implement *SPB*, *IGMP snooping*, *DHCP snooping*, *801.1X*, etc. While implementing one of these features would be impressive for a two weeks project, you are not expected to do so. We even **strongly** advise against it. You may nonetheless do some research to at least get to know what they do.

Dans le doute, reboot.