Lab BSY NET

Introduction & Prerequisites

In this laboratory you will expand on knowledge gained from KT and ITSec.
This laboratory is to learn or gain:

- Handle routing using the `ip` utility
- In-depth experience with `nftables` hooks
- Gain some experience with namespaces

The following resources and tools are required for this laboratory session:

- A ZHAW VPN session
- Any modern web browser
- Any modern SSH client application
- OpenStack Horizon dashboard: https://ned.cloudlab.zhaw.ch
- OpenStack account details
  - See Moodle
- Username to login with SSH into VMs in ned.cloudlab.zhaw.ch OpenStack cloud from your laptops
  - **`Ubuntu`**

Time

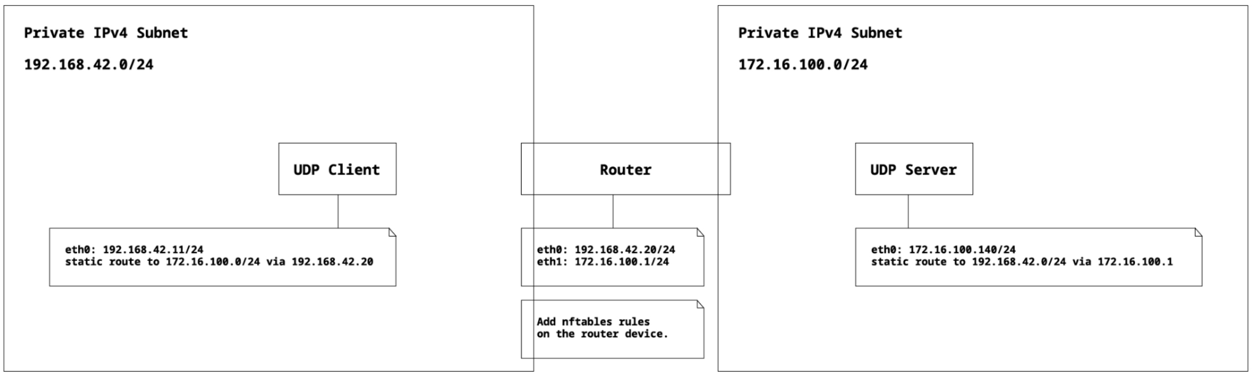
The entire session will take 90 minutes.

Assessment

No assessment foreseen

Task 1 – Setup & Basic Tasks

We want to build the following networking situation:



There are several ways of doing this. One is to instantiate three machines, virtual or not, and use networking facilities. A second is using containers, Docker etc. In this lab we will use netspaces on a single Openstack VM. To do this we need:

1.) Ensure `nftables` is installed
2.) Download a repo with a set of bash script files that will setup the name spaces

Make sure `nftables` is installed:
Type `nft --version`. If `nft` is not installed then do so using

```
sudo apt install nftables
```

Clone the appropriate repo using your ZHAW username/password and:

```
git clone
https://github.zhaw.ch/InES-RT-Ethernet/bsy-lab-ip-route-students.git
```

In the sub-directory `/netenv` call

```
sudo bash netenv setup
```

This will set up a networking environment with three IP stacks running, each in its own namespace `(golden-<xyz>`). For the purposes of this lab this setup is *functionally* equivalent to having three separate virtual machines or three networked docker images on one machine. Netspaces are sometimes used by hypervisors to separate network accesses by different guests and by containers for the same reason. Netspaces are also useful in setting up small networking environments for test and development purposes. When you are finished or want to start the lab again, you can delete the instances by using:

```
sudo bash netenv teardown
```

You can access `ip` features in the context of the namespaces, for instance

```
sudo ip netns exec golden-router bash
```

Replace `router` with `client` or `server` to access the respective namespace

Task 2 – Routing

Subtask 2.1 – Setting up the routing

Enter the router namespace and - using the `ip` utility - examine the routing table. The `ip` utility features a number of sub-utilities, for instance `link`, `route`, `address (addr)`, `monitor`.

Using the `ping` utility, are the client and server reachable?

Using the `ping` utility on the client, are the router and server reachable and if not why?

Let's make a static route from the client to the router - what parameters do you think are needed, and why? Using the `ip` utility setup the route. Check the router table contains the entry.

```
ip route add 172.16.100.0/24 via 192.168.42.20 dev enp0s8
```

Check the route using

```
ip route list
```

Ping again, what happens and why?

Setup the routing table on the server and configure the router as a router - you may want to find the egress device name using

```
ip link <parameter>
```

To enable the routing on the network stack

```
On the router -> echo "1" > /proc/sys/net/ipv4/ip_forward
On the router -> cat /proc/sys/net/ipv4/ip_forward
```

Ping the client from the server (or the other way round).

Subtask 2.2 – Running the application

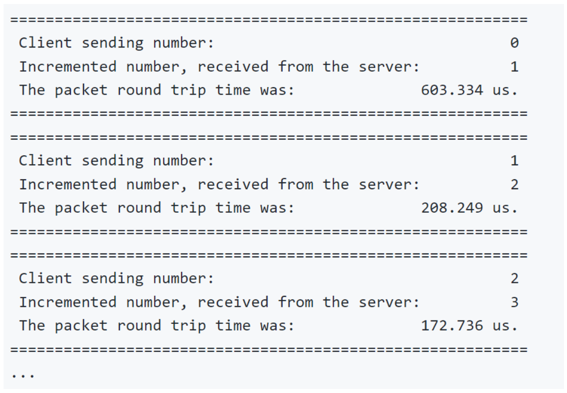
On the server machine, issue the command:

```
./srvr/bin/srvr
```

On the client machine, issue the command:

```
./clnt/bin/clnt
```

You should see the following:



```
==========================================================
 Client sending number:                                 0
 Incremented number, received from the server:          1
 The packet round trip time was:              603.334 us.
==========================================================
==========================================================
 Client sending number:                                 1
 Incremented number, received from the server:          2
 The packet round trip time was:              208.249 us.
==========================================================
==========================================================
 Client sending number:                                 2
 Incremented number, received from the server:          3
 The packet round trip time was:              172.736 us.
==========================================================
...
```

The client part of the round-trip application automatically stops after the incrementing number has reached a value of 1000. If desired, the client can just be restarted and the sequence starts over again.

When running, the server as well as the client part of the round trip application can be stopped by hitting Ctrl + C on the respective console.

Task 3 – Setting up Netfilter rules

`netfilter` - name of the subsystem within the linux kernel that allows access, through hooks, to packets traversing between interfaces.

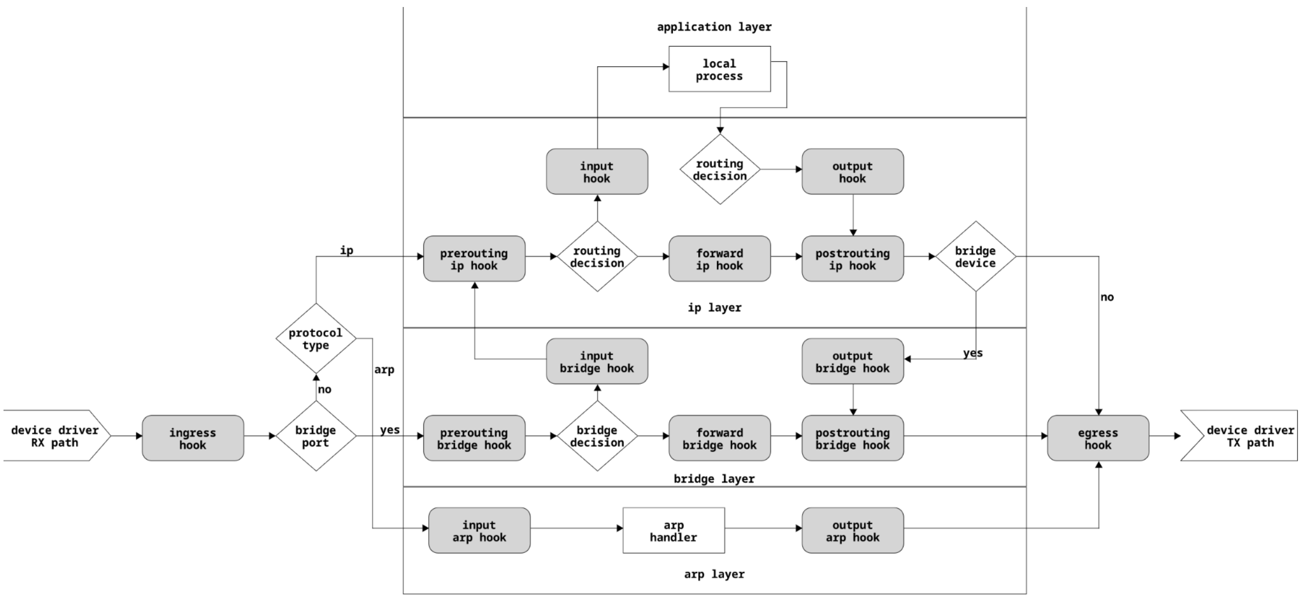
`netlink` socket - the socket interface through which user-land processes can communicate with the netfilter.

`nftables` (`nft`) The user space front-end utility used to configure the netfilter subsystem.

The `netfilter` subsystem of the Linux kernel can be configured from user space through `netlink` sockets. The user space front-end which is used to configure the netfilter subsystem is called `nftables` or just `nft`. The `nft` front-end unites and, at the same time, replaces the front-ends: `iptables`, `ip6tables`, `arptables` as well as `ebtables`. Hence, one has one and the same application to configure network filtering rules at all the possible hook-in points from an operating system point of view, e.g. even before entering the IP networking protocol stack of the operating system.

In the scope of this laboratory, the aim is to set up a packet counter which counts the UDP network packets that stem from the UDP round-trip application. A counter that should count ingress packets at the network interfaces of the router which are connected to the two distinct private IPv4 subnets, see also fig. 1. On the transport layer, the application uses the user datagram protocol (UDP). The application identifying characteristic is the static server port number that is 42000. The packets shall be counted before they enter the internet protocol (IP) networking protocol stack of the operating system. I.e. right after the packets are passed from the networking driver to the operating system.

Subtask 3.1 – Understanding the structure of `netfilters` and `nft`



In the man page of `nft,` compare the hooks mentioned in the "Address Families" section with the diagram above. Relate the address families and hooks from the man page to the diagram above. Note: the availability of the `netdev` egress hook is `nft` version dependent.

Trace the path of the client UDP frame through the netfilter. Where would be a good place to set up a counter counting all frames entering the netfilter? Where would be a good place to set up counters for the following

1.) Counting all UDP frames entering the IP layer from an external port
2.) Counting all UDP frames coming from an application
3.) Counting the sum of frames from these two sources?

Subtask 3.2 – Setting up a `netdev` counter

Let's count frames coming into the netfilter. To do this `nft` offers a counter facility and counters need to be instantiated. The counters are triggered by a rule. Since in the wider scheme of things there can be many rules, rules need to be organized and are done so in chains. The chains are stored in a table which is address family specific.

So we need to set up a table. Given the table we can set up a counter and we can set up a chain.

Once a rule is associated with a chain we can associate the rule with the counter. Awkward but there you have it.

On the router machine we set up a table and an associated named counter - we look at client-sided ingress - it is **important** you take the time to understand the syntax.

```
nft add table netdev tbl-lowlevel-cnt
```

After every action on this table you can view the effect by using
```
nft list ruleset
```

We add the counter using
```
nft add counter netdev tbl-lowlevel-cnt round-trip-from-clnt
```

The name of the table and counter can be arbitrarily selected. In this case, the name `tbl-lowlevel-cnt` and `round-trip-from-clnt` was respectively chosen.

Now we setup a chain
```
nft add chain netdev tbl-lowlevel-cnt \
    udp-ping '{ type filter hook ingress device enp0s8 \
                priority -300; policy accept; }'
```

Finally we set up a rule
```
nft add rule netdev tbl-lowlevel-cnt udp-ping \
             udp dport 42000 \
             counter name round-trip-from-clnt
```

Run the client and server application again. Use `nft list ruleset` to view the state of the counters.

Subtask 3.3 – Comfortable debugging

We can also add a rule to monitor whenever a chain is traversed:

```
nft add rule netdev tbl-lowlevel-cnt udp-ping \
             meta nftrace set 1
```

Check how the table is changed by this command (`nft list ruleset`). To have live tracing, use

```
nft monitor trace
```

Subtask 3.4 – Exercise

Set up a counter on the postrouting hook in the ip address family

Cleanup

IMPORTANT: At the end of the lab session:

- **Delete** all -unused - OpenStack VMs, volumes, security group rules that were created by your team.

Additional Documentation

OpenStack Horizon documentation can be found on the following pages:

- User Guide: https://docs.openstack.org/horizon/latest/