How networks work: what is a switch, router, DNS, DHCP, NAT, VPN and a dozen of other useful things |…

We'll figure out how networks work and solidify the knowledge in practice using libvirt and Linux net.....

The subject of networking, unfortunately, is boring for the most of our colleagues. All the used technologies, protocols and best practices are pretty old, they have been surrounding us and ensuring the communication between millions of devices around us for a long time. Even programmers most often take networks for granted and don't think about how they work.

It often happens with us: we use words like IP and DNS every day, but there's no understanding how it all works, and how to try it on. Such an attitude is not only incorrect, but also wrong for every self-respecting IT engineer's career. It doesn't matter how many frameworks you have learned, without the networking knowledge you won't be taken seriously. No part of the infrastructure should remain a blackbox neither for developers nor for administrators nor, of course, for you, the future DevOps engineer.

The goal of this article is not to give a comprehensive guide on networks. Inside of the article and in the end of it, I will give a lot of links to the sources, which can help you deepen the knowledge you've gained. Don't be lazy, click all the links and read everything.

But in this text we will focus on a network structure, its basic components and see how they are used in practice with the help of virtual machines and libvirt/KVM in particular, which we have become familiar with in the previous article .

OSI model

First of all, we need to get familiar with the OSI model . This model standardizes the communication between network protocols.

OSI divides the communication into 7 layers, each one having its protocols. You will hear things like "it happens on the 3rd layer" a lot. Here are these layers:

- Physical layer
- Data link layer
- Network layer
- Transport layer
- Session layer
- Presentation layer
- Application layer

Physical layer

The protocols of this layer are responsible for hardware communication on the lowest level. The very transmission of

data by wire (or wireless) is described in this layer. Examples of protocols: Wi-Fi, Bluetooth, DSL.

Data link layer

Data link layer is responsible for the transmission of data between two devices in one network. Data is transmitted in frames, a frame contains the physical address of the sender and the receiver. This address is called MAC-address.

So, who are the sender and the receiver?

First of all, every device (including your laptop) has NIC — Network Interface Controller. This is a piece of hardware (or virtual hardware), which is responsible for sending and receiving frames . NIC has a MAC-address - a unique address usually embedded in a hardware or generated by a virtualization system.

Of course, a machine can have multiple NIC's. Let's look at the interfaces using the `ip` command:

```
[root@localhost ~]$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAU
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    link/ether 52:54:00:05:36:e6 brd ff:ff:ff:ff:ff:ff
```

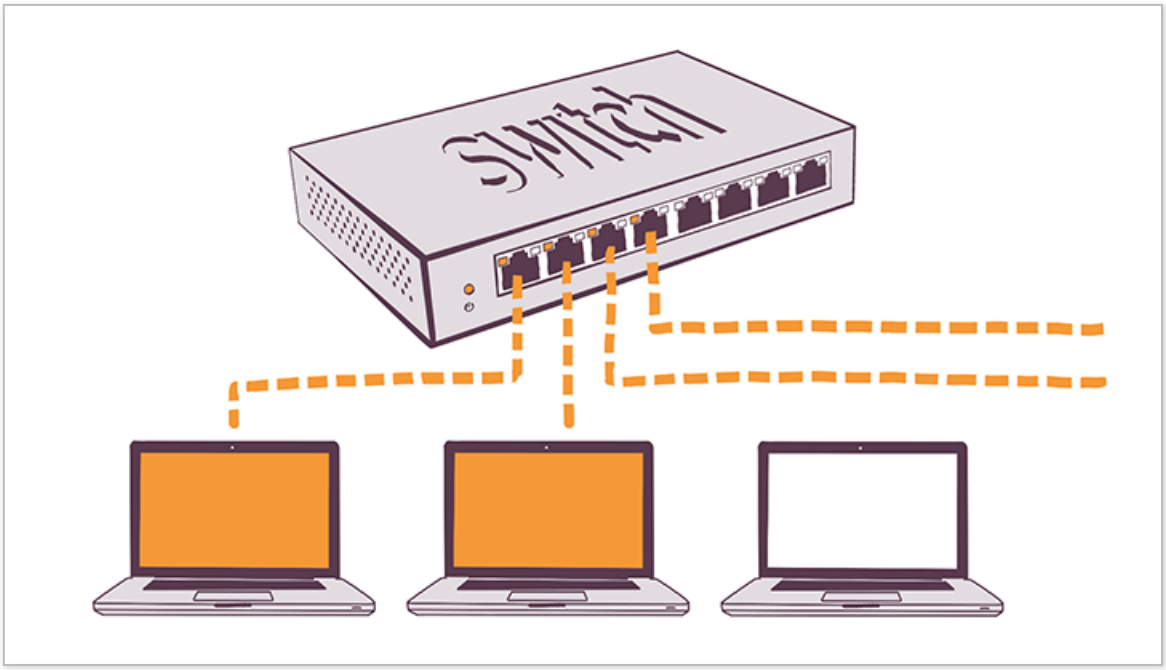
In this case, an interface used for communicating with the world via network is eth0, which has the MAC-address *52:54:00:05:36:e6*. But what is `lo` ?

`lo` is a loopback device, a specific virtual interface, which system uses to communicate with itself. Thanks to `lo` , local applications can communicate with each other even without a network connection.

You have already noticed that your computer has billions of cables connected directly to all computers in the world. A network needs additional devices for its organization.

For example, *switch*.

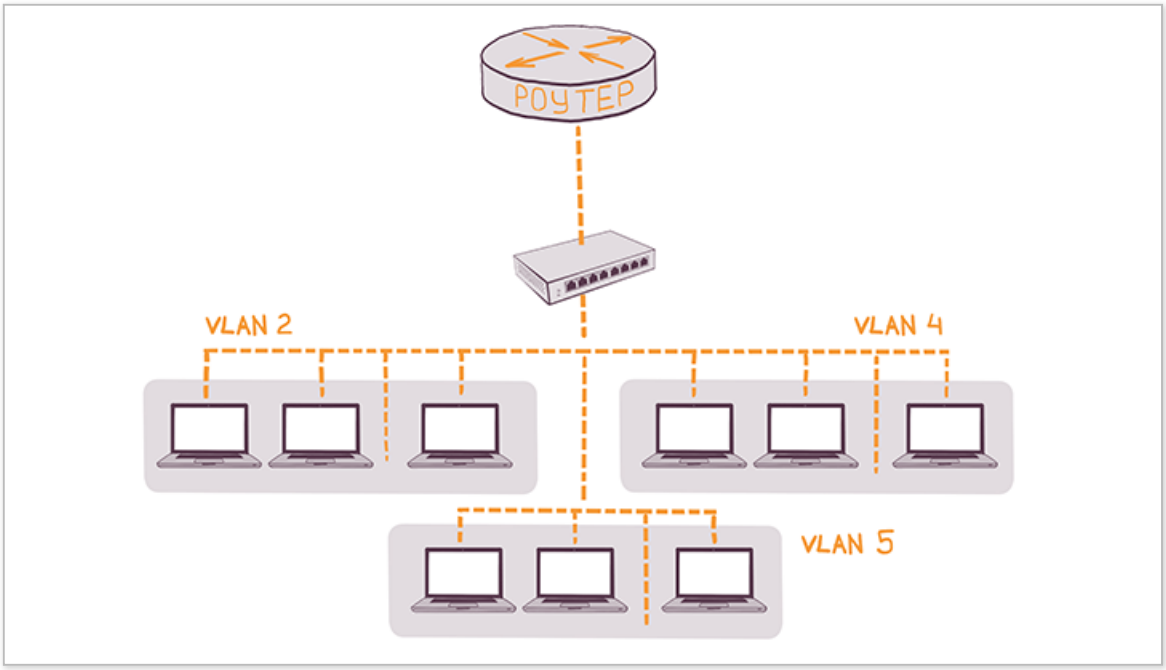
A switch is a device which builds up the network and which all our machines are connected to via ports. The task of L2 switch (there are more advanced ones, concerning L3 and even L7) - to forward frames from MAC sender to MAC receiver. A lot of devices connected to one switch form a local area network(LAN) .



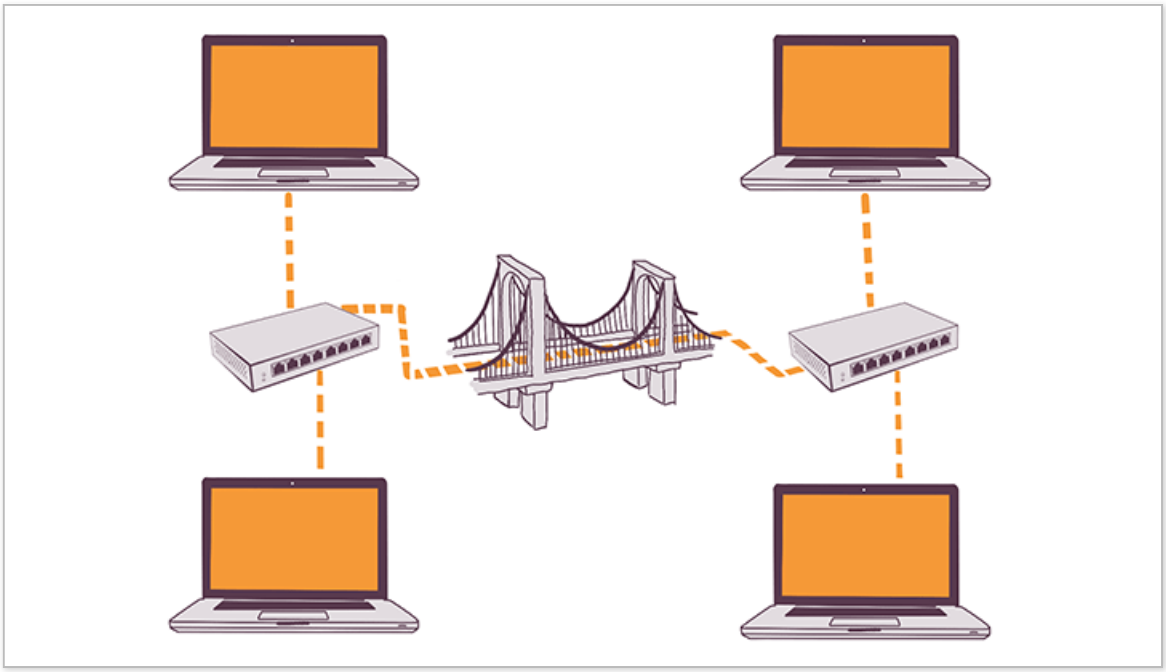
Of course, a bunch of servers connected to one switch is quite an obvious way to create a network. But what if we want to establish a network of servers located physically in different places? Or, say, we want to logically separate servers connected to one switch in one location into different networks?

For such cases a VLAN (virtual local area network) is created. You can implement it, say, using a switch. It works pretty simple:

an additional header with a VLAN-tag is added to the frame, and it determines which network the frame belongs to.



Another device is a bridge. An L2 bridge is used to connect two networks, formed using switches, together like this:



Both switches and bridges (and also hubs, read about them yourself) help to connect multiple devices together into one network. There are also routers which connect networks, they work on L3. For example, your Wi-Fi router connects your local

area network (where there is your laptop, mobile phone and tablet) with the Internet.

> Besides LAN, there are some other network types: For example, WAN . You can count Internet as WAN except that Internet completely erases the geographic boundaries of a network.

> As I have already mentioned, there are also L3 switches, which can not only forward frames from one device to another, but also have some more advanced specialties, like routing. So, what is the difference between a router and an L3 switch, you may ask. It's all pretty difficult (and boring), but if you are interested, read this article Layer 3 Switches compared to Routers

---

Network layer

On the third, network layer, IP-addresses are used instead of MAC-addresses. Let's look at our device's IP using the same command `ip` :

```
[root@localhost ~]$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    link/ether 52:54:00:05:36:e6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.212/24 brd 192.168.122.255 scope global dynamic eth0
       valid_lft 2930sec preferred_lft 2930sec
    inet6 fe80::5054:ff:fe05:36e6/64 scope link
       valid_lft forever preferred_lft forever
```

The *192.168.122.212/24* address is assigned to the `eth0` interface.

But what is `/24`? And why does the loopback interface have `/8`? You have probably already heard that there are 4 294 967 296 IPv4 addresses. Internet is not one large network, but many little networks. Moreover, separate blocks of IP addresses are reserved for different kinds of networks (for example, private networks, which are not accessible from outside).

> There are a lot more IPv6 addresses. But the full transition to IPv6 have not yet happened :-)

CIDR is a method for allocating IP addresses for different kinds of networks. And the CIDR-notation is a way to write this block in a format `192.168.122.212/24`, where the number `/24`, called a mask, makes it possible to understand how many addresses there are in this block.

IPv4 is a simple number with the length of 32 bits, which can be represented in binary code. In binary code IP addresses go from `00000000000000000000000000000000` to `11111111111111111111111111111111`. For convenience, let's split this number up into 4 pieces, each one having 8 digits: `11111111.11111111.11111111.11111111`. In decimal system we are used to, this address looks like this: `255.255.255.255`.

The mask `/24` can be represented as `255.255.255.0`, or, in binary notation, `11111111.11111111.11111111.00000000`. In order to find the first and the last addresses of a network, we can use one of the addresses and a network mask and apply a bitwise AND to their binary notation:

```
11000000.10101000.01111010.11010100
&
11111111.11111111.11111111.00000000
=
11000000.10101000.01111010.00000000
```

Let's translate the result in a human-readable representation: `192.168.122.0` is the starting address of our network. In order to count the amount of all accessible addresses, we need to count the amount of zeros in the mask. In our case, there are 8 zeros, or positions. Each of them can possess the value of 1 or 0, that's why in the whole we get 2^8 degree, or 256 addresses. It means that the last address will be `192.168.122.255` .

> You don't have to count all this manually, you can use a calculator .

ARP

We already know that L2 uses MAC-addresses and L3 - IP-addresses. There has to be some mechanism, which associates a MAC-address with its IP-address. This mechanism is called ARP (Address Resolution Protocol) .

Linux has a command of the same name `arp` , which allows us to look at the table of MAC-addresses the device knows and IP-addresses mapped to them.

```
[root@localhost]# arp -n
Address HWtype HWaddress Flags Mask Iface
192.168.178.1 ether 5c:49:79:99:f3:23 C wlp3s0
```

In this case, 192.168.178.1 is the IP-address of my Wi-Fi router, which my laptop is connected to via wlp3s0 interface.

> The `arp` command is considered to be deprecated, and it is strongly recommended to use `ip neigh` instead of it.

One of the cyber-attacks kinds is connected to ARP and is called ARP spoofing . The goal of such an attack is to replace a MAC-address, associated with a certain IP-address, with a hacker's device address. Life is a scary thing, indeed.

DHCP

But how exactly a network interface is assigned an IP-address? One of the options - manually. The disadvantage: handwork. If you're no good with your hands, you can configure duplicate addresses and get a conflict :)

Another option: Dynamic Host Configuration Protocol (DHCP), a protocol used for setting different configuration, including IP-addresses, automatically.

> Refer to RFC documentation for more details on DHCP: https://www.ietf.org/rfc/rfc2131.txt

For DHCP to work, you need a DHCP-server, which will assign IP-addresses, and a DHCP-client on your device, which will request for an address. At home, the DHCP-server is usually located in router.

In order to understand how exactly DHCP works, you need to focus on "broadcasting" . This is a process, in which our server transfers a message to *all* servers in the network, as it doesn't know where exactly the information it needs is located. Such a broadcast communication is close to a radio broadcasting.

In case of DHCP, it happens like this:

- A DHCP-client sends a broadcast message with a request "I need an IP-address"
- A DHCP-server catches it and sends back also a broadcast message "I have an IP-address x.x.x.x, do you want it?"
- The DHCP-client receives the message and sends another one: "Yes, I want the address x.x.x.x"
- The DHCP-server answers "Ok, then x.x.x.x belongs to you"

> In this video the whole process is shown more clearly:
> https://www.youtube.com/watch?v=RUZohsAxPxQ

And where are the connection settings stored?

The connection settings are stored in `/etc/sysconfig/network-scripts`. That's where you can edit such things as the way an IP-address is assigned (automatic or static), whether to start connection automatically when the system loads or not, etc. For example, that's how my Wi-Fi-connection config looks like:

```
[root@localhost network-scripts]# cat ifcfg-FRITZ-Box_7490
HWADDR=4C:34:88:54:C1:2B
ESSID="FRITZ!Box 7490"
MODE=Managed
KEY_MGMT=WPA-PSK
TYPE=Wireless
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
NAME="FRITZ!Box 7490"
UUID=55ba9218-1d2f-407d-af13-51502d542edb
ONBOOT=yes
```

```
SECURITYMODE=open
PEERDNS=yes
PEERROUTES=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
```

Pay attention to `BOOTPROTO=dhcp` - this option means that my computer will use a DHCP-server, for receiving an IP-address as well. As a comparison, the connection config for a loopback device:

```
[root@localhost network-scripts]# cat ifcfg-lo
DEVICE=lo
IPADDR=127.0.0.1
NETMASK=255.0.0.0
NETWORK=127.0.0.0
# If you're having problems with gated making 127.0.0.0/8 a martian,
# you can change this to something else (255.255.255.255, for example)
BROADCAST=127.255.255.255
ONBOOT=yes
NAME=loopback
```

The static address is specified here: `IPADDR=127.0.0.1` . At home, you can use the tool `nmcli` or install the package `Networkmanager-tui` , which will provide a user-friendly text interface right in your console, instead of editing configs manually. in field conditions, on servers, you better not do this and use the system of configuration (Puppet, Chef, Salt) instead.

One more important part of the configuration: routing. How to understand, where the traffic will flow? Everything is pretty simple: it's enough to look at the local routing table using the `ip_r` command. At the time of writing, I am sitting in a coffee shop with a laptop, which uses a mobile phone as a router. That's what `ip_r` displays:

```
default via 172.20.10.1 dev wlp3s0 proto static metric 600
172.20.10.0/28 dev wlp3s0 proto kernel scope link src 172.20.10.3 metric 600
```

```
192.168.100.0/24 dev virbr2 proto kernel scope link src 192.168.100.1
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1
```

As you can see, all the traffic goes by default to the machine with the address `172.20.10.1`. And if I run `ip addr show`, I will see that the network interface on my laptop also has an IP address from this network:

```
4: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
    link/ether 4c:34:88:54:c1:2b brd ff:ff:ff:ff:ff:ff
    inet 172.20.10.3/28 brd 172.20.10.15 scope global dynamic wlp3s0
       valid_lft 83892sec preferred_lft 83892sec
    inet6 fe80::4e34:88ff:fe54:c12b/64 scope link
       valid_lft forever preferred_lft forever
```

You can add new paths using the `ip r add` command, and delete them using the `ip r del` command.

DNS

You have probably already heard about DNS. The idea is simple: to request server not by its IP-address (it's hard to remember for people), but by its normal name.

The oldest and the most popular DNS-server (the one that stores information about addresses and responds to requests) is BIND. There a lot of alternatives, but it is BIND that you are recommended to deploy locally first of all.

The material from Cisco DNS Best Practices, Network Protections, and Attack Identification must be on your reading list - there you will learn not only DNS basics but also a lot of useful recommendations on creating a safe and sustainable DNS-server.

It is possible to update records in DNS-server automatically. You can read about nsupdate. You will find a link to a great guide on

configuring, including safe records updating, below. One of the interesting usages is service discovery. Look at the Internet what it is about or wait for the article about it on mkdev :-)

Before DNS, all that we had was a file /etc/hosts. It's often used even now.

> A section "Viruses for Dummies"! Open /etc/hosts file on a friend's computer and add there a line `52.28.20.212 facebook.com` . No more sitting on facebook, it's better he studies development!

There is one more interesting file /etc/nsswitch.conf. This is where it's defined in what order and where to look for different information, including where to look for hosts. By default, they are looked for in /etc/hosts, and only after that a request to a DNS-server is sent.

A server used for resolving DNS-names is defined in /etc/resolv.conf, by the way.

It's better to debug DNS problems using commands dig and nslookup. For example, in order to request information from nameserver 8.8.8.8 about mkdev.me, all you need to do is:

```
# dig mkdev.me @8.8.8.8

; <<>> DiG 9.10.3-P4-RedHat-9.10.3-12.P4.fc23 <<>> mkdev.me @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3320
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;mkdev.me. IN A
```

```
;; ANSWER SECTION:
mkdev.me. 299 IN A 52.28.20.212

;; Query time: 355 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Fri May 27 12:51:04 CEST 2016
;; MSG SIZE rcvd: 53
```

Virtual machines

Before this, all the examples were made on a local machine. Of course, it's useful for your perception, but it's not that interesting. That's why we will solidify everything we've read using virtual machines and libvirt, and also get acquainted with a couple of terms.

First of all, let's create a VM using virt-install:

```
sudo virt-install --name mkdev-networking-basics-1 \
--location ~/Downloads/CentOS-7-x86_64-Minimal-1511.iso \
--initrd-inject /path/to/ks.cfg \
--extra-args ks=file:/ks.cfg \
--memory=1024 --vcpus=1 --disk size=8
```

By default, libvirt creates one network:

```
[root@localhost]# virsh net-list
 Name State Autostart Persistent
----------------------------------------------------------
 default active yes yes
```

A block `192.168.0.0/16` is allocated for private networks. libvirt has allocated a block `192.168.122.212/24` for its network, that means all the addresses from `192.168.122.0` to `192.168.122.255`.

To look at the detailed information about a certain network, you can use either `virsh net-info` or `virsh net-dumpxml`. The second command will return a lot more details, that's why let's use it:

```
[root@CentOS-72-64-minimal ~]# virsh net-dumpxml default
<network connections='1'>
  <name>default</name>
  <uuid>f2ee9249-6bed-451f-a248-9cd223a80702</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:83:b4:74'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

`connections` shows the amount of machines connected to this network. You can read a detailed description of all the possible options of this XML-file in the libvirt documentation . But right now we are interested in two words: bridge and dhcp.

bridge, or a `virbr0` device, or a virtual network switch, is a special device which all the VMs in this network are connected to. All the requests from one VM to another within one network go through this virtual switch. Libvirt creates one virtual switch for each network, and every switch is recognized as a separate device on host machine:

```
[root@localhost]# ip link show
8: virbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP r
    link/ether 52:54:00:a8:02:f2 brd ff:ff:ff:ff:ff:ff
```

Creating a network in libvirt is by default equated with creating a virtual switch which all the VMs are connected to, thus creating a local area network, LAN.

The virbr0 switch is implemented using Linux Bridge - a technology originally intended exactly for creating virtual local area networks. You can see a list of all the switches executing the `brctl show` command on the host machine.

> Linux Bridge is "slightly" different from a typical hardware L2 switch. During the years of its existence a lot of features were added to it, like traffic filtering and firewall. The most proper way to call it is L3 switch, but here your obedient servant isn't completely sure.

Now let's pay attention to the following section:

```
<ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
```

Here a block of addresses used for virtual machines in this network is declared. `192.168.122.1` is the IP address of a host-machine within this virtual network.

If we run `ip r` in VM, we'll see:

```
[vagrant@localhost ~]$ ip r
default via 192.168.122.1 dev eth0 proto static metric 100
192.168.122.0/24 dev eth0 proto kernel scope link src 192.168.122.209 metric 1
```

By default, the traffic from VM goes outside through a host-machine. As an entertainment, you can set a configuration for a traffic to go to one virtual machine through another.

As we already know, the DHCP service is responsible for assigning IP-addresses. Libvirt uses dnsmaq for DHCP and DNS

and runs one dnsmasq instance for each network.

```
`[root@CentOS-72-64-minimal ~]# ps aux | grep dns
nobody 10600 0.0 0.0 15548 856 ? S Apr01 0:02 /sbin/dnsmasq --conf-file=/var/
root 10601 0.0 0.0 15520 312 ? S Apr01 0:00 /sbin/dnsmasq --conf-file=/var/lib
```

Now we can look at the DHCP table, which will show us the assigned addresses:

```
[root@loclahost]# virsh net-dhcp-leases default
 Expiry Time MAC address Protocol IP address Hostname Client ID or DUID
-------------------------------------------------------------------------------
 2016-04-29 16:31:19 52:54:00:05:36:e6 ipv4 192.168.122.212/24 - -
```

Pay attention that *52:54:00:05:36:e6* is our VMs MAC-address of the eth0 interface.

NAT

When you were reading about CIDR, there was something that might get your attention: even if we divide the network into many blocks, the total amount of IP-addresses isn't going to increase. Actually, a combination of private and public addresses is always used. Usually, one public address hides a lot of machines, each one having its own private address.

This is also true for our VMs. Each one has the private IP-address from the block `192.168.122.0/24`, and all of them are hidden behind the public address of the host machine.

The host machine, if we continue to use our private laptop at home as it, is hidden behind our Wi-Fi router and also doesn't have a public address.

At first glance, the fact that VMs have an access to Internet seems evident. But the VM only has a private address, which is not accessible outside of the host machine. A public server VM

requests to needs to forward a response somewhere, but it just won't be able to find VM's private IP-address (because it is private).

NAT(Network Address Translation) will solve this problem. It is a mechanism of resolving IP-addresses in network packages. Usually, sender's and receiver's IP-addresses are included in a package. NAT makes it possible to change these addresses dynamically and save the table of changed addresses.

There is also SNAT (source NAT), which is the one that is used by our VMs to get access to Internet. When a package is sent, its source address is replaced with the host machine address. When a response from the target server goes back, the address is changed from the host machine address to the VM address. It is the router that changes the address.

DNAT (destination NAT) does pretty much the same, but vice versa: this is when you request to some public address which hides private, local addresses.

NAT is the default way of VM's communicating with the world. But libvirt is a flexible thing. For example, you can connect VMs directly to a host's physical interface instead of a virtual switch. Actually, there are a lot of ways to create a network .

Libvirt uses iptables for NAT. In short, this is a tool responsible for filtering network packages. iptables are configured with the help of special rules, which combine in chains. By means of adding such rules, libvirt gives our VMs access to Internet, using NAT. We will return to iptables, when we speak about safety in general.

> Also, the ip*forward option must be enabled in core settings in order for package redirecting to work on the host. It is very easy to enable it: `echo 1 > /proc/sys/net/ipv4/ip*forward`

tcpdump

Perhaps, the most essential tool for network problems, or, to be more specific, traffic going through our machine debugging, is tcpdump. It is highly important to know how to use it. Let's look, for example, what is happening on our virbr0 when restarting a VM.

Let's open a console on the host machine and run `tcpdump -i virbr0` .

Open the separate window and run `virsh reboot #{number_of_VM}` .

Look at the result in the first window and see where which requests came from.

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on virbr0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:57:31.339135 IP6 :: > ff02::16: HBH ICMP6, multicast listener report v2, 1
12:57:31.397937 IP 0.0.0.0.bootpc > 255.255.255.255.bootps: BOOTP/DHCP, Reques
12:57:31.398182 IP linux.fritz.box.bootps > 192.168.122.209.bootpc: BOOTP/DHCP
12:57:31.590332 ARP, Request who-has linux.fritz.box tell 192.168.122.209, len
12:57:31.590373 ARP, Reply linux.fritz.box is-at 52:54:00:7e:33:23 (oui Unknow
12:57:31.590409 IP 192.168.122.209.38438 > linux.fritz.box.domain: 61342+ A? (
12:57:31.590458 IP 192.168.122.209.38438 > linux.fritz.box.domain: 25671+ AAAA
12:57:31.590618 IP linux.fritz.box.domain > 192.168.122.209.38438: 25671 0/0/0
### And so on
```

Here is, for example, a broadcast from the VM: `12:57:31.397937 IP 0.0.0.0.bootpc > 255.255.255.255.bootps: BOOTP/DHCP, Request from 52:54:00:e0:06:54 (oui Unknown), length 300` .

Also, let's look at the ARP table:

```
Address HWtype HWaddress Flags Mask Iface
# ...
192.168.122.209 ether 52:54:00:e0:06:54 C virbr0
# ...
```

VPN

Sometimes (to tell the truth, very often) it is necessary to make it look like both the client and the server are in one private network. For example, when all the company services are in private network, which is accessible only from the office, but you need to give a remote access to the company workers. Or when a company has several offices or data centers, which need to be connected with each other in such a way that all the network still would not be accessible to all Internet.

VPN in itself is putting one tcp/ip package into another and encrypting the content. As a result, there is a virtual network working inside of a real network. Virtual network devices (tun/tap) are created for virtual networks. They have virtual IP-addresses which are accessible only within our virtual encrypted network.

I will leave VPN configuring out of this article. It is on reader's conscience to try to do it on his own using OpenVPN or strongSwan .

We will return to the subject of safety later on, but you can already read about IPsec — this is the protocol used by strongSwan.

For-self-study

We have just covered the most basics of networks, but, of course, there is almost a dozen of technologies, which are worth to look at. Google VXLAN on your own, learn TCP and UDP (and

figure out which one when to use), look at IMCP. You will encounter new terms constantly, but, as always, the most important thing is to learn the fundamentals.

We haven't reached the higher OSI layers, haven't looked at the different protocols web-applications work with: HTTP(S), FTP, SSH, NTP and many others.

> Don't forget to look into RFC . This is the first stop when searching information on networks you need.

I think we will return to these subjects in the next articles. As for me, the most difficult part was to understand how exactly all this works on the layers below application layer. all these nets, subnets, inexplicable problems with one server getting access to another etc.

I hope that now you know a little bit more about the basic components involved in a network communication, and that in future you will be able to solve arousing problems faster - because you know in advance what and where might go wrong.

What's next?

I know that my dear reader can't wait for me to start speaking about Chef, Puppet, Ansible and many other cool stuff. But it's too early. I have at least one more article on this subject, in which we will consider all the possible ways to authenticate and authorize users and servers, and this way dig into the subject of safety in general more.

Further reading

As I have already said, the subject of networks is complicated, deep and touches upon many different subjects. You must be

having a little mess in your head. It's ok! The following links will help you to learn everything you need to know about networks more deeply.

- Virtual Networking
- Intro to networking
- What is the loopback device and how do I use it
- Reserved IP addresses
- NAT
- DNS best practices
- http://illumium.org/node/33
- Linux Bridge
- Libvirt networking handbook
- Using nsswitch.conf to find Linux system information
- Hubs, switches, routers
- Guide to network scripts