System Optimization

Taras Pudiak - 2021-09-21 - 0 Comments - in System Settings

Due to a very wide list of supported hardware, VyOS cannot be optimized to any of it "out of the box". So, it is always a good idea to check some values and make fine-tuning, according to your network requirements. Here is the list of some things, which can require your attention for optimization:

1. Network card and driver optimization.

Many modern network interfaces support fine-tuning. We recommend to start from the ring queue buffer size - this buffer is used to save pointers into data, which was received from the NIC or should be transferred via it. By default, drivers set some "optimal" value, but it not always can fit requirements. In two words:

- high size of ring buffer leads to less CPU usage due to lowing frequency of network-related interrupts. But, due to this, make latency a bit higher. Preferred for high loads systems;

- low size of ring buffer leads to high CPU usage and even packet drops, in case if packets rate is too high. But makes latency lower. Preferred for latency-sensitive systems.

To check your current values of ring buffers, you can use the next command:

```
sudo ethtool -g eth0
```

where eth0 is the name of your network interface. You should see something like this:

```
vyos@test-05:~$ sudo ethtool -g eth3
Ring parameters for eth3:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
Current hardware settings:
```

```
RX:             128
RX Mini:        0
RX Jumbo:       0
TX:             128
```

As you can see, our network card support values up to 4096, but the current setting is 128.

To change this value you need to add the next lines to the "/config/scripts/vyos-postconfig-bootup.script" file:

```
ethtool -G eth0 tx 4096 rx 4096
```

This command will change RX and TX values to 4096 at every boot.

2. Optimizing for virtual environments.

2.1. Network interfaces.

Using VyOS as guest machine supported in many hypervisors, and this feature is very often actively used by our customers. Routing high levels of traffic via virtual environments meets no so often, but still required in some cases.

Delivering traffic to VM could be done via three types of network interfaces:

- physical (SRV-IO, VFIO, PCI passthrough);

- fully emulated (e1000, rtl8139);

- para-virtualized (virtio, vmxnet3).

Physical interfaces provide the best performance, almost at the level of bare-metal, but require support in hardware and additional settings. Paravirtualized interfaces is a good compromise between performance, flexibility, and software support.

Fully emulated interfaces we recommend to use only in case if any other option is not available on your platform.

As an addition, modern para-virtualized network interfaces support the multi-queue option, which allows balancing traffic between multiple virtual queues. If you want to make your instance fully optimized, we recommend setting queues count equal to count of physical CPU cores, available for a virtual machine. This will give the ability to use all CPU cores for routing traffic. Typically, to reach a maximum efficiency (not performance!) of resource usage, a count of available CPU cores should be multiple of total queues count.

2.2. RAM allocation.

If your hypervisor supports RAM preallocation or reservation, you should use this feature for VyOS VM. As VyOS does not use swapping, overprovisioning of RAM by hypervisor could easily trigger out-of-memory for VyOS VM and leads to unexpected and unpredictable problems with traffic processing. So, do not use features like memory ballooning or memory sharing.

2.3. CPU allocation.

A count of vCPU cores allocated to VyOS is a topic for discussion on each specific case, there is no magic formula for counting this value. But we have some general recommendations, which could be used as a start point:

- do not use only one vCPU core, except cases, when traffic level is really low (maximum several tens of Mbps), and system latency and responsiveness are not critical;

- do not allocate count of vCPU cores more than a count of physical CPU cores on the node, which will handle this VM;

- if you have a NUMA system, the router will use more effective CPU cores, which belongs to the same node as used RAM. But, if resources of the single node are not enough to handle all the traffic, of course, you will need to add CPU cores from the other ones.

And a special note: never use Hyper-Threading for systems, which handle routing instances - with this option system will always have less performance, than without.

2.4. High availability in virtual environments.

If you want to provide reliable services, you may want to use integrated into your hypervisor solutions for this, like live VM migration between nodes, VM re-balancing between physical hosts, and so on. But, in most situations, we strongly recommend to not do this with VMs which route traffic, like VyOS. As traffic flows are strongly reliant on physical paths and connections, a transparent moving router's VM to another physical node can easily break down these flows and confuse hosts and routers in your network.

So, we strongly recommend to bind VyOS VM to a single physical node, and if you want to reach high availability with routing, use solutions like VRRP.

3. Optimization for NAT and stateful firewall.

If NAT or stateful firewall is used in your network for providing access for many clients to the Internet, for translating connections to some high-load internal services or in any other way, which prognosticate usage of many thousands of connections via NAT at the same time, you may meet the problem with concurrent connections limit. Each connection between the hosts via NAT require storing at least one record in connections states table. If that information will be unavailable, any new packets in established connection via NAT will be not able to reach it's destination correctly. And if that table will be full, there will be no

possibility to establish a new one connections. As a result, you may see many different issues with traffic, like:

- you need to refresh the page in browser multiple times to load some website;

- access to some websites or resources works well, but you cannot reach others;

- sometimes, when you run a ping to some host behind the NAT you will get all answers without any problems, but if you stop and run ping again there will be no answers.

To avoid such problems, you need to be sure that your router always has free space in connections tracking table. To get the information about the current state of this table you can use the next commands:

Get the count of the current connections in table:

```
conntrack -C
```

or

```
sysctl net.netfilter.nf_conntrack_count
```

Get the maximum predefined size of connections tracking table:

```
sysctl net.netfilter.nf_conntrack_max
```

If you see that current count is close to table size, this indicates that you need to make this table bigger. To change this table size you can use the next command:

```
set system conntrack table-size XXXXX
```

where XXXXX is the new size of the table. Each connection use 320 Bytes of memory, so the exact table size for your system must be lower than TotalRAM / 320Bytes. In general cases, to avoid calculations and leave some reserved memory for other services, you can think that you need 512 MB of RAM for each million of connections.

Another one system variable that you may need to tune is conntrack bucket size. Depending on this value, you may get better firewall performance, especially for the first packets in connections. We recommend to set its value twice time less than a table-size value. This option can be changed via:

```
set system sysctl custom net.netfilter.nf_conntrack_buckets
```

According to this, an example for the system with 2GB RAM will look like:

```
set system conntrack table-size 4194304
set system sysctl custom net.netfilter.nf_conntrack_buckets value
2097152
```

Some services can generate a lot of connections with short time, which are not closing correctly. If your router serves such services, you may additionally want to tune connections timeout, to make their life shorter and avoid connection table overload with "outdated" connections. To change timeout values, you may use the next command:

```
set system conntrack timeout
```

Some additional information about exact values, you can read here: https://www.kernel.org/doc/Documentation/networking/nf_conntrack-sysctl.txt

4. SMP Affinity.

SMP Affinity lets you assign IRQs to specific processors. By default VyOS configures SMP affinity automatically distributing NICs across CPU. But you can also configure it manually for specific needs.

For optimal performance a dedicated CPU core could be assigned to each network interface. You can manually see the IRQ for each interface in /proc/interrupts, e.g.

```
vyos@vyos:~$ cat /proc/interrupts | grep eth
  16:        930   IO-APIC-fasteoi   eth1
  19:       2445   IO-APIC-fasteoi   eth0
```

The smp_affinity value for an interface is a bitmask of available CPU cores in hexidecimal. Example for a 4-core system:

```
            Binary       Hex
    CPU 0    0001         1
```

```
    CPU 1    0010         2
    CPU 2    0100         4
    CPU 3    1000         8
```

The configuration would look something like:

```
set interfaces ethernet eth0 smp_affinity 1
set interfaces ethernet eth1 smp_affinity 2
```

Manually, this would be:

```
echo 1 > /proc/irq/16/smp_affinity
echo 2 > /proc/irq/19/smp_affinity
```

Note that newer systems may have multiple IRQs for each interface, e.g.

```
 30:    3176819    3181843   PCI-MSI-edge      eth0-rx-0
 31:        393        343   PCI-MSI-edge      eth0-tx-0
 32:          5          5   PCI-MSI-edge      eth0
 33: 3696151744 3709942356   PCI-MSI-edge      eth1-rx-0
 34: 3110055844 3108916060   PCI-MSI-edge      eth1-tx-0
 35:    1674002    1673645   PCI-MSI-edge      eth1
 36:  695750170  650735677   PCI-MSI-edge      eth2-rx-0
 37: 2838722531 2838813824   PCI-MSI-edge      eth2-tx-0
 38:     879793     879761   PCI-MSI-edge      eth2
 39:   51381461   50904827   PCI-MSI-edge      eth3-rx-0
 40:    9658376    9570304   PCI-MSI-edge      eth3-tx-0
 41:          5          7   PCI-MSI-edge      eth3
```

Here, for each interface seperate IRQs are used for RX, TX, and control. VyOS will assign the same affinity to each of the queues associated with a specific network interface.

If **smp-affinity auto** is configured, VyOS will try to distribute NICs across CPU cores automatically.