Complex internship: computer networks

By Hannes Blümel, Philipp Herold, Aleksey Ermoshin

Motivation

This internship aims to investigate the potential benefits for replacing VyOS, an open-source network operating system, with a NixOS system based on one configuration file to build it. Background for this is that building VyOS is like fighting an uphill battle. The requirered packages and dependencies to build VyOS are endless and need a specialist to solve. NixOS on the other side has it advantages in its reproducibility of packages, which means that packages are isolated from another and have no dependencies on each other. The question to ask now is: can we use the advantages of NixOS and reproduce the functionality of VyOS with it?

To resolve this we are going to analyse the networking functionality of VyOS. Then we derive 3 real world scenarios which include sub-functions of VyOS. These scenarios will then be build by using NixOS. To complete our proof-of-concept we will write a translator which converts a VyOS configuration, with the functionality of the 3 scenarios, to the nixos configuration, so that afterwards both systems have the same scope of functions.

Background

VyOS

VyOS, a versatile network operating system and offers a wide range of use cases we will list some of them here.

- VyOS supports various routing protocols such as OSPF, BGP, RIP
- It supports VPN technologies including IPsec, OpenVPN, L2TP, and PPTP, enabling secure remote access and site-to-site connectivity
- VyOS offers firewall functionality, allowing administrators to control traffic flow and enforce security policies
- It supports Network Address Translation (NAT)
- VyOS can function as a DHCP server, providing automatic IP addresses to client devices on the network.
- The system supports Virtual LAN (VLAN) functionality, enabling network segmentation
- The operating system offers high availability features such as VRRP (Virtual Router Redundancy Protocol), ensuring network resilience and redundancy
- VyOS can be deployed on various hardware platforms, including physical servers, virtual machines, and cloud environments, providing flexibility and scalability
- It supports Multi-WAN load balancing and failover, distributing network traffic across multiple Internet connections for improved performance and reliability
- VyOS offers a command-line interface (CLI) and a web-based management interface, providing multiple options for configuration and administration

The full list of configuration options for VyOS can be found in the documentation:
https://docs.vyos.io/en/latest/configuration/index.html#configuration-guide

While VyOS offers several advantages, it comes at a price that goes beyond monetary considerations. Although it is an open-source solution, utilizing VyOS effectively can be quite expensive in terms of knowledge and expertise. Building VyOS from source code is a cumbersome process due to its reliance on various packages, not to mention the complexities involved in customizing the system to suit specific needs. This requires a significant investment of time and effort to gain a comprehensive understanding of VyOS.

NixOS

NixOS is a Linux distribution known for its unique package management and system configuration approach. It is based on the Nix package manager, offering a declarative way to manage software and system settings. One of its key advantages is the ability to manage packages independently, meaning each package and its dependencies are isolated and self-contained. This approach avoids conflicts and ensures that upgrading or removing a package does not impact other parts of the system. A unique feature of this approach is to simultaneosly install and switch between different versions of one package. Upgrades in NixOS are designed to be performed as a whole, ensuring the entire upgrade process is reliable and can be easily reversed if needed. With NixOS, the entire operating system is defined by a single configuration file, allowing for reproducible setups. To conclude NixOS is famous due to its reproducibility, reliability, flexibility, and the independent nature of its packages.

Task

To demonstrate the advantages of NixOS and show its ability to provide similar functionality as VyOS, we will replicate three specific use cases in both VyOS and NixOS configurations. Additionally, we will develop a generic transformer that takes a VyOS configuration as input and generates a corresponding NixOS configuration.

The transformer will automate the process of migrating from VyOS to NixOS for the specified use cases, ensuring a smooth transition and replication of functionality.

While the initial transformer implementation will focus on the specified use cases, it can be extended by additional mapping files and configurations to cover more services. This flexibility allows for customization and adaptation of the transformer to suit specific needs beyond the initial use cases. For further information see Transformer.

By successfully replicating the use cases and providing a tool to transform VyOS configurations into NixOS configurations, we aim to highlight NixOS's ability to offer comparable functionality to VyOS while using its unique features such as package isolation and reproducibility, using one config.

Requirement analysis

This section lists all requirements to complete this internship.

- R1: Analyse the functionality of VyOS and NixOS
- R2: Derive three use cases of VyOS and build the scenarios using NixOS
- R3: Develop a generic transformer to convert VyOS configurations to NixOS configurations
- R4: Assess the advantages and disadvantages of replacing VyOS with NixOS

Use cases

For our proof of concept we choose some realistic use cases, where we provide the VyOS functionality by NixOS. While defining our goal we divided our use cases by increasing komplexity.

The first one rebuilds a DHCP server with three clients that automatically receive an ip-addresses. Additionally to the DHCP we configured a bonding interface between two NixOS systems.

In the second scenario we configure a wireguard VPN which simulates a remote worker to be connected to an onsite network.

The third scenario addresses the routing functionality of a VyOS device. Therefore we configure three AS boder-routers exchanging network topology with their peers.

Further description and explanation about the scenarios can be found in the according folders in this repository. DHCP scenario Bonding scenario VPN scenario BGP scenario

DHCP scenario

Usecase

- receive a network configuration via DHCP server in the local network
- 3 actors: two clients and one DHCP Server
- each actor is based on NixOS
- client and actor differ in their configuration file (more details below)

Implementation

DHCP Server configuration

Interface configuration

- add interface / adapter to internal network
- disable that interface obtains network configuration via DHCP:

```
useDHCP = false;
```

- assign static IP-Address to new interface

```
ipv4.addresses = [{
  address = "10.1.1.1";
  prefixLength = 24;
}];
```

DHCP server configuration

- enable DHCPv4 Server

```
enable = true;
```

- configure that DHCP server should listen on new created interface

```
interface = ["enp0s8"];
```

- configure subnet, subnet-mask, broadcast-address, interface (must be selected because their could be other interface which also use DHCP and serve other subnetworks

```
extraConfig = ''
  option subnet-mask 255.255.255.0;
  subnet 10.1.1.0 netmask 255.255.255.0 {
    option broadcast-address 10.1.1.255;
    option routers 10.1.1.1;
    interface enp0s8;
    range 10.1.1.2 10.1.1.254;
  }
'';
```

Complete configuration snippet

```
networking = {
  hostName = "nixos-router"; # Define your hostname.
  defaultGateway = "";
  interfaces.enp0s8 = {
    useDHCP = false;
    ipv4.addresses = [{
      address = "10.1.1.1";
      prefixLength = 24;
    }];
  };
};

services.dhcpd4 = {
  enable = true;
  interfaces = ["enp0s8"];
  extraConfig = ''
    option subnet-mask 255.255.255.0;
    subnet 10.1.1.0 netmask 255.255.255.0 {
      option broadcast-address 10.1.1.255;
      option routers 10.1.1.1;
      interface enp0s8;
      range 10.1.1.2 10.1.1.254;
    }
  '';
};
```

Client configuration

Interface configuration

- add interface / adapter to internal network
- enable that interface obtain network configuration via DHCP

```
useDHCP = true;
```

Complete configuration snippet

```
networking = {
  hostName = "nixos-client-0"; # Define your hostname.
  defaultGateway = "";
  interfaces.enp0s8 = {
    useDHCP = true;
  };
};
```

Testing

The testing in this scenario is pretty simple. We just have to start the DHCP server and the two clients. After that we can check if the clients got a valid IP address from the DHCP server. We can do this by running the following command on the clients:

```
ip a
```

Problems

One mystery we encountered was, that the clients automatically received ip-addresses in a network space we did not know. Trough further research we found out that Virtualbox already got a DHCP server running which distributed our machines a basic network configuration for the internal network.

To speed up the process of configuring NixOS in the virtual machines and copy the configuraitions file to the host system, we used the shared folder functionality of Virtualbox. This allowed us to edit the configuration file on the host system and deploy the changes to the virtual machines while saving them in the git repository.

Bonding scenario

Usecase

This example aims at rebuilding the bonding/link aggregation functionality of VyOS. Bonding is a technology used to increase fault tolerance and load balancing by conbining multiple network interface cards (nics) into one logical.

The setup is build by two virtual maschines using VirtualBox, which are directly connected to two interfaces of your host system. To prove the increased avaiability of this setup we deactivate one link and still keep the connection to the other OS alive.

Configuration

Adding virtual interfaces

To create interfaces on the host maschine over which the virtual maschines (VMs) can connect to each other you need to execute the following:

```
ip tuntap add p1 mode tap
ip tuntap add p2 mode tap
```

This adds virtual interfaces to your host which operate at layer 2 and can be used to bridge network traffic.

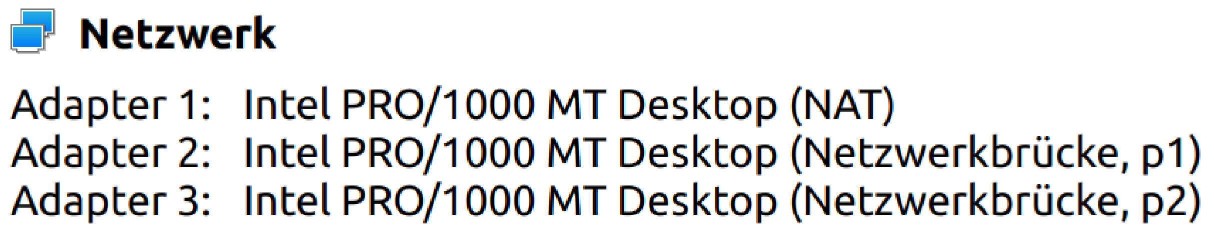
After you initialized the interfaces you need to activate them.

```
ip link set dev p1 up
ip link set dev p2 up
```

Settings in VirtualBox

Connecting the VMs to each other you have to add two additional nics to each VM and configure them as shown in the picture.

The first nic is used for internet connection and the 2nd and 3rd will be used for the bonding.



Netzwerk

Adapter 1: Intel PRO/1000 MT Desktop (NAT)
Adapter 2: Intel PRO/1000 MT Desktop (Netzwerkbrücke, p1)
Adapter 3: Intel PRO/1000 MT Desktop (Netzwerkbrücke, p2)

Configuration of NixOS

Configuring bonding with the NixOS configuration file can be accomplished in three different ways:

- using systemd.network
- install netplan and configure .yaml
- use NixOS internal networking setction

We chose the third option because it's the simplest and most straightforward approach.

The following network configuration configures the bond interface. First you have to set a hostname and disable the DHCP client for the two interfaces (in this case 'enp0s8' and 'enp0s9'). The third interfaces section configures the 'bond0' interface by setting the IP-address and the subnet and deactivating the DHCP client aswell. The bonds section creates actual bonds with their specific settings. The interfaces option allows you to choose the interfaces that schould be used as slaves by the bond. Within the driverOptions you can configure all the attributes explained in the following manpage. https://www.kernel.org/doc/Documentation/networking/bonding.txt

To get everything up and running you need to add this lines to the file at "/etc/nixos/configuration.nix". For the second VM you need to change the hostName and the IP-address of the bond interface (e.g. "192.169.100.3").

```
    networking = {
      hostName = "host-1";
      defaultGateway = "";
      interfaces.enp0s8 = {
        useDHCP = false;
      };
      interfaces.enp0s9 = {
        useDHCP = false;
      };
      interfaces.bond0 = {
        useDHCP = false;
        ipv4.addresses = [{
          address = "192.168.100.2";
          prefixLength = 24;
        }];
      };
      bonds = {
        bond0 = {
          interfaces = [ "enp0s8" "enp0s9" ];
          driverOptions = {
            miimon = "100";
            mode = "active-backup";
            primary = "enp0s8";
            fail_over_mac = "active";
          };
        };
      };
    };
```

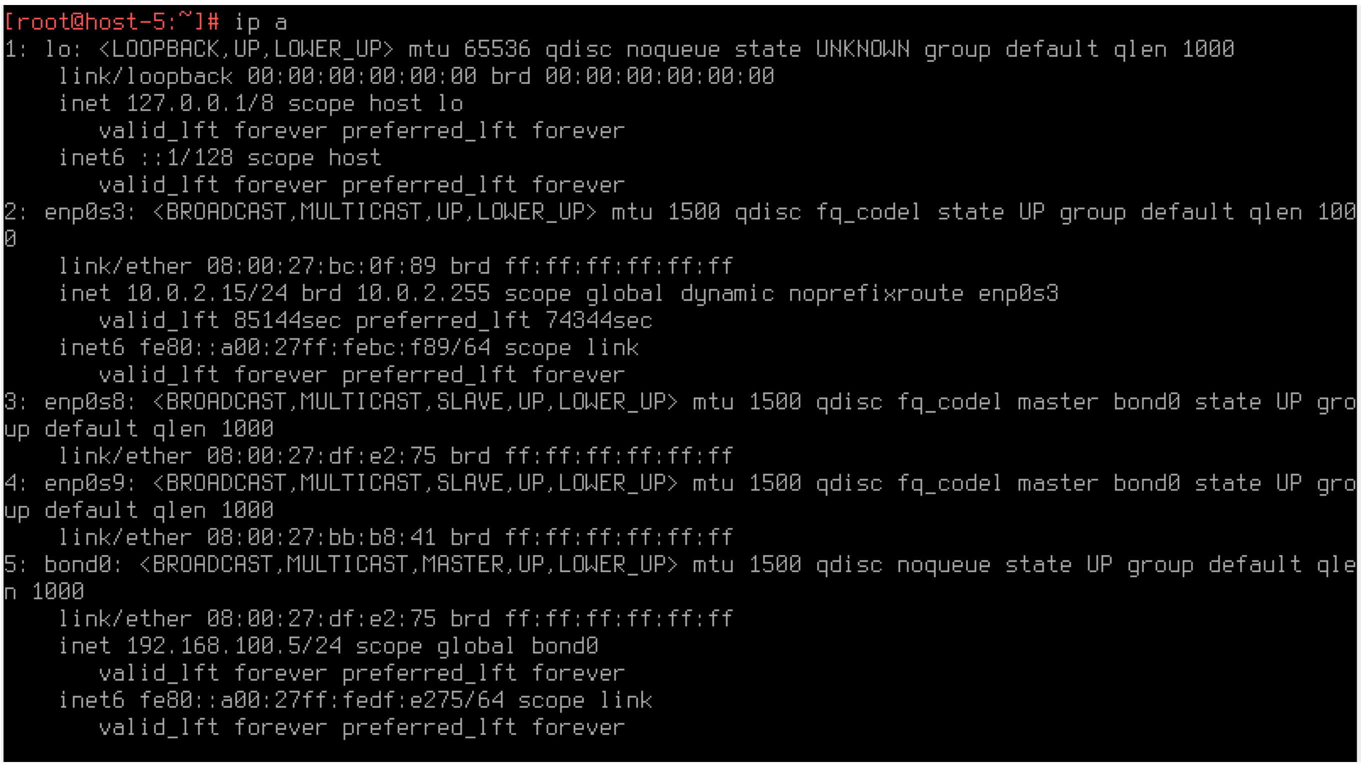
After you added this to your configuration.nix file you can run the command to rebuild the system and apply all changes.

```
sudo nixos-rebuild switch
```

Testing

To be sure that the bond interfaces are workung correctly we can look up the current state of the nics with the command:

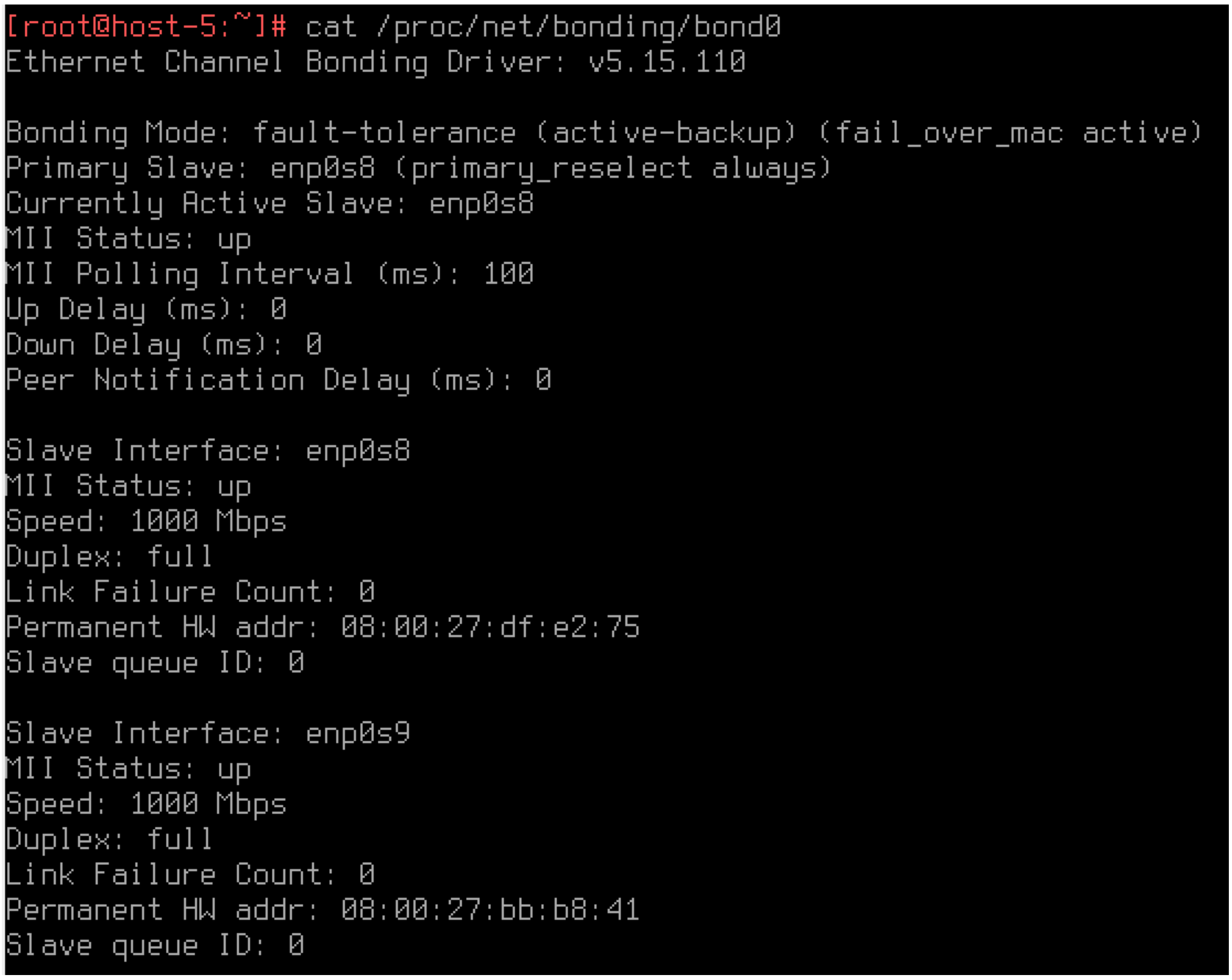
```
ip a
```



```
[root@host-5:~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
0
    link/ether 08:00:27:bc:0f:89 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 85144sec preferred_lft 74344sec
    inet6 fe80::a00:27ff:febc:f89/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bond0 state UP gro
up default qlen 1000
    link/ether 08:00:27:df:e2:75 brd ff:ff:ff:ff:ff:ff
4: enp0s9: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bond0 state UP gro
up default qlen 1000
    link/ether 08:00:27:bb:b8:41 brd ff:ff:ff:ff:ff:ff
5: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qle
n 1000
    link/ether 08:00:27:df:e2:75 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 scope global bond0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fedf:e275/64 scope link
       valid_lft forever preferred_lft forever
```

If we want to get a more concrete setting of the bond interface we can read the following file by:

```
cat /proc/net/bonding/bond0
```



```
[root@host-5:~]# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.15.110

Bonding Mode: fault-tolerance (active-backup) (fail_over_mac active)
Primary Slave: enp0s8 (primary_reselect always)
Currently Active Slave: enp0s8
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: enp0s8
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:df:e2:75
Slave queue ID: 0

Slave Interface: enp0s9
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:bb:b8:41
Slave queue ID: 0
```

The fault-tolerance improvement can be tested and simulated by deactivating the active slave interface while pinging the other VM and see that there is no disturbance. So we start by pinging the VM. Then we use the following command to deactivate the interface on the other VM which was set as 'Currently Active Slave' in the bond configuration.

```
ip link set dev enp0s8 down
```

If we then look at the bond interface configuration we see that the 'Currently Active Slave' has changed but the connection is still active.

Problems

In the process of building this setup there where mainly two obsacles. The first was about NixOS which did not apply the configuration changes after using 'nixos-rebuild switch' therefore I used a new clone from my initial instance.

The other obstacle is to read the manual of the bonding configuration carefully to be sure that your config is valid. You find that at: https://www.kernel.org/doc/Documentation/networking/bonding.txt

VyOS configuration of bonding

Create bond ad set mode

```
set interfaces bonding bond0 mode 802.3ad
```

Give address to bond

```
set interfaces bonding bond0 address 10.1.1.1/24
```

add interface to the bond-group (vyos 1.3)

```
set interfaces bonding bond0 member interface eth0
set interfaces bonding bond0 member interface eth1
```

set primary bond interface

```
set interfaces bonding bond0 primary eth0
```

See further commands here

Site-To-Site Wireguard VPN

Usecase

By using Wireshark to analyze network traffic between the server and the client, we have verified that the data transmission between the devices is encrypted using Wireguard. This enhances the security and integrity of the transmitted data, ensuring that confidential information remains protected from unauthorized access.

The usecase aims to provide assurance that confidential information is being transmitted safely and that the implemented security measures are functioning as intended.

Setup

- A server and a client (VMs) are set up in the same network.
- Wireguard is configured and enabled on both devices.
- Wireshark is installed on the analysis device(Host).

Implementation

Adding virtual interfaces on the host

We need to add a virtual interface (p1) to the host which operate at layer 2 and can be used to bridge network traffic. We will specify this added port in the network settings of virtual machines and we will select it for analysis with wireshark.

This adds virtual interfaces to your host which operate at layer 2 and can be used to bridge network traffic.

Then we need to install wireguard on both virtual machines

```
nix-env -iA nixos.wireguard-tools
```

and generate public and private keys.

```
umask 077
mkdir ~/wireguard-keys
wg genkey > ~/wireguard-keys/private
wg pubkey < ~/wireguard-keys/private > ~/wireguard-keys/public
```

Server configuration

```
  networking = {
    hostName = "host1";
    defaultGateway = "";
    interfaces.enp0s8 = {
      useDHCP = false;
      ipv4.addresses = [{
        address = "10.1.1.1";
        prefixLength = 24;
      }];
    };
  };
```

Using this setting, we assigned a static IP address 10.1.1.1 to the server. The client was assigned IP 10.1.1.2. Then we need to configure the wg0 interface. The wg0 interface is a virtual network interface created by Wireguard. wg0 handles the encapsulation and encryption of network traffic, ensuring that data sent between peers remains confidential, authenticated, and tamper-proof. It serves as the entry point for Wireguard connections.

```
 # enable NAT
 networking.nat.enable = true;
 networking.nat.externalInterface = "enp0s8";
 networking.nat.internalInterfaces = [ "wg0" ];
 networking.firewall = {
   allowedUDPPorts = [ 51820 ];
 };

 networking.wireguard.interfaces = {
   # "wg0" is the network interface name.
```

```
    wg0 = {
      # Determines the IP address and subnet of the server's end of the
tunnel interface.
      ips = [ "10.100.0.1/24" ];
      # The port that WireGuard listens to.
      listenPort = 51820;
      # Path to the private key file.
      privateKeyFile = "/root/wireguard-keys/private";
      # List of allowed peers.
      peers = [
        {
          # Public key of the peer
          publicKey = "+OJZvRhvhX3Bnlvk0uQXALaVDZk0v6PZ9LQMFUe94hI=";
          # List of IPs assigned to this peer within the tunnel subnet. Used
to configure routing.
          allowedIPs = [ "10.100.0.2/32" ];
        }
      ];
    };
  };
```

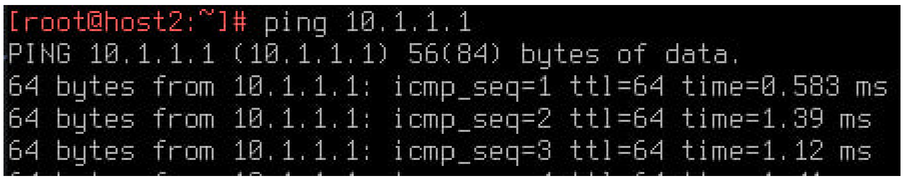
Client configuration

```
  networking.firewall = {
    allowedUDPPorts = [ 51820 ]; # Clients and peers can use the same port
  };

  networking.wireguard.interfaces = {
    wg0 = {
      # Determines the IP address and subnet of the client's end of the
tunnel interface.
      ips = [ "10.100.0.2/24" ];
      listenPort = 51820;
      # Path to the private key file.
      privateKeyFile = "/root/wireguard-keys/private";

      peers = [
        {
          # Public key of the server
          publicKey = "C3XoNS2rRNPW4Dtph7O/cZnURrYv/X1ZMZinHUuhBxc=";
          # Forward only particular subnets
          allowedIPs = [ "10.100.0.1" ];
          endpoint = "10.1.1.1:51820";
          # Send keepalives every 25 seconds
          persistentKeepalive = 25;
        }
      ];
    };
  };
```

After all the settings, we need to execute a command

```
sudo nixos-rebuild switch
```

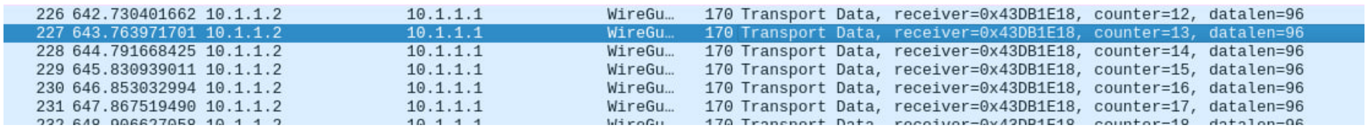
that "rebuilds the system" and if everything went well, now we can ping the configured wg0 interface.



```
[root@host2:~]# ping 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=64 time=0.583 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=64 time=1.39 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=64 time=1.12 ms
```

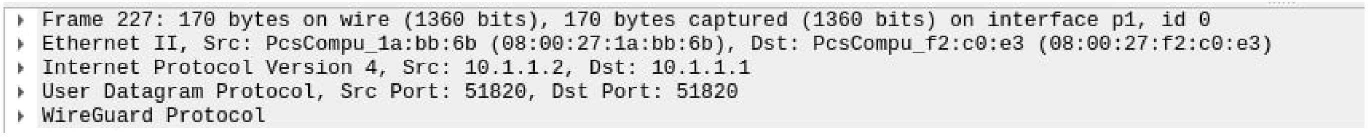
Testing with WireShark

After installing wireshark on the host, we need to select the interface created in the previous step and execute the ping command on any of the VMs. We will see the following results:

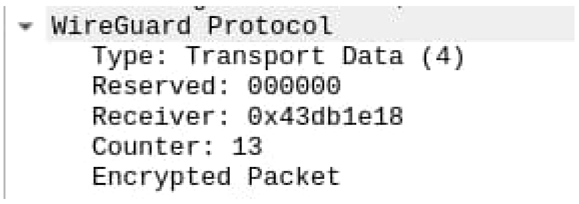


```
226 642.730401662 10.1.1.2    10.1.1.1    WireGu…  170 Transport Data, receiver=0x43DB1E18, counter=12, datalen=96
227 643.763971701 10.1.1.2    10.1.1.1    WireGu…  170 Transport Data, receiver=0x43DB1E18, counter=13, datalen=96
228 644.791668425 10.1.1.2    10.1.1.1    WireGu…  170 Transport Data, receiver=0x43DB1E18, counter=14, datalen=96
229 645.830939011 10.1.1.2    10.1.1.1    WireGu…  170 Transport Data, receiver=0x43DB1E18, counter=15, datalen=96
230 646.853032994 10.1.1.2    10.1.1.1    WireGu…  170 Transport Data, receiver=0x43DB1E18, counter=16, datalen=96
231 647.867519490 10.1.1.2    10.1.1.1    WireGu…  170 Transport Data, receiver=0x43DB1E18, counter=17, datalen=96
```

How can we see the packets are transmitted in encrypted format.



```
▸ Frame 227: 170 bytes on wire (1360 bits), 170 bytes captured (1360 bits) on interface p1, id 0
▸ Ethernet II, Src: PcsCompu_1a:bb:6b (08:00:27:1a:bb:6b), Dst: PcsCompu_f2:c0:e3 (08:00:27:f2:c0:e3)
▸ Internet Protocol Version 4, Src: 10.1.1.2, Dst: 10.1.1.1
▸ User Datagram Protocol, Src Port: 51820, Dst Port: 51820
▸ WireGuard Protocol
```



```
▾ WireGuard Protocol
    Type: Transport Data (4)
    Reserved: 000000
    Receiver: 0x43db1e18
    Counter: 13
    Encrypted Packet
```

If we ping another interface, the situation will be different. There, the packets are transmitted unencrypted, which allows a third party to take over the data.

Problems

Since IP addresses were dynamic by default, it was necessary to change this configuration to static for correct operation.

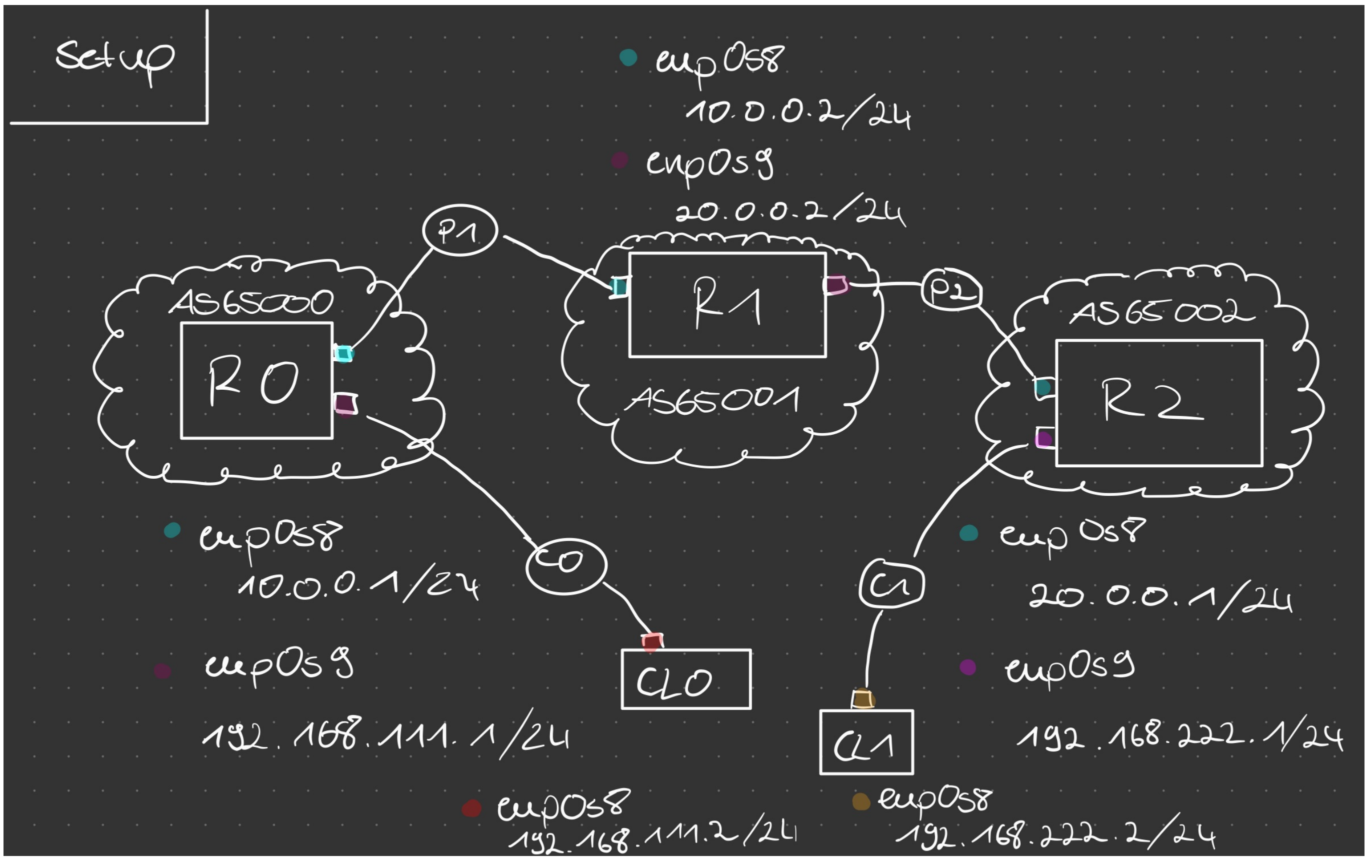
Also when configuring wireguard there were some bugs inside the nixos system. Two machines with exactly the same config produced different results. After a hard reset and a new setup, everything was fixed.

BGP Router Setup

Usecase

This example aims at rebuilding the routing functionality of VyOS, especially the BGP protocol. BGP, or Border Gateway Protocol, is the core routing protocol of the Internet that allows autonomous systems to exchange routing information and make decisions about the best paths for data traffic to flow between networks.

The setup is build by five virtual maschines using VirtualBox, these are connected in a line and share a seperate network at each connection. The separation in different networks should virtualize the real world of autonomous systems which are not a complete graph and have to exchange routing information over multiple hops. The architecture of this example is shown in the following image. The ports (p1,p2,c0,c1) are virtual interfaces on the host maschiene where the VMs can connect to each other and we can use wireshark on them to analyse the traffic.



Configuration

Adding virtual interfaces

To create interfaces on the host maschine over which the virtual maschines (VMs) can connect to each other you need to execute the following:

```
ip tuntap add p1 mode tap
ip tuntap add p2 mode tap
ip tuntap add c0 mode tap
ip tuntap add c1 mode tap
```

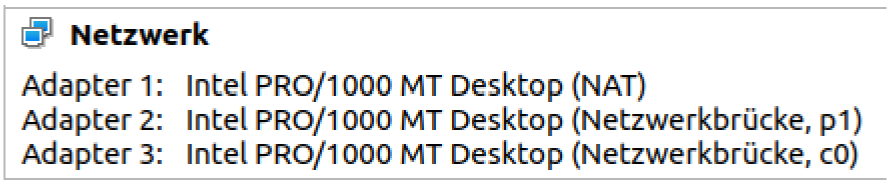
This adds virtual interfaces to your host which operate at layer 2 and can be used to bridge network traffic.

After you initialized the interfaces you need to activate them.

```
ip link set dev p1 up
ip link set dev p2 up
ip link set dev c0 up
ip link set dev c1 up
```

Settings in VirtualBox

The 2nd step is to connect the VMs according to the network achitecture. Therefore we can choose the option "networkbridge" and select the right port. In the image you see the network configuration of the R0 router.


Netzwerk

Adapter 1:	Intel PRO/1000 MT Desktop (NAT)
Adapter 2:	Intel PRO/1000 MT Desktop (Netzwerkbrücke, p1)
Adapter 3:	Intel PRO/1000 MT Desktop (Netzwerkbrücke, c0)

Configuration of NixOS

For the application of BGP there are two packages supported by NIXOS, which can add this functionality

- BIRD
- FRR

We choose the FRR daemon, but you can also use the BIRD daemon the only difference is their configuration syntax.

You can search for packages that are avaiable for nix on the following website: NixOS packages

Before we can define any configuration for FRR we have to add it to the packages we want to install system wide. in the following example you see that vim and wget are installed togeather with FRR.

```
  environment.systemPackages = with pkgs; [
    vim
    wget
    frr
  ];
```

Another preparation step is to configure the network interfaces at each VM. The following snippet is an example for the R0 router. Therefore we add the ip addresses to each interface. To allow incomming connections on the port 179 (BGP) we decided to deactivate the firewall completely in our lab environment. Of cause this must be configured differently on a working environment.

```
  networking = {
    hostName = "bgp-router-0";
    defaultGateway = "";
    interfaces.enp0s8 = {
      useDHCP = false;
      ipv4 = {
        addresses = [{
          address = "10.0.0.1";
          prefixLength = 24;
        }];
      };
    };
    interfaces.enp0s9 = {
      useDHCP = false;
      ipv4.addresses = [{
        address = "192.168.111.1";
        prefixLength = 24;
      }];
    };
    firewall.enable = false;
  };
```

After the network setup we have to add the configuration for the bgp daemon. In the following code block you see an example for the R0 router. First we activate the daemon. Secondly we give it the configuration. In the code are comments to explain each line

```
  services.frr.bgp = {
    enable = true;
    config = ''
      router bgp 65000                  % Configuring BGP process with
autonomous system number 65000
        bgp router-id 10.0.0.1          % Assigning the BGP router ID as
10.0.0.1
        no bgp ebgp-requires-policy     % Disabling the requirement for
eBGP peering to have a policy applied
      network 192.168.111.0/24          % Advertising the network
192.168.1.0/24 into BGP routing table
      neighbor 10.0.0.2 remote-as 65001 % Establishing a BGP peering with
the neighbor at IP address 10.0.0.2, which belongs to AS 65001
      redistribute connected            % Redistributing connected routes
into BGP
    '';
  };
```

After you added all the above to your configuration.nix file at '/etc/nixos/' you can run the command to rebuild the system and apply all changes.

```
sudo nixos-rebuild switch
```

We applay the same configuration with exchanged ip addresses and AS numbers to the R1 and R2. For the CL0 and CL1 (clients) we just have to configure the network connection and set the default gatway to the connected bgp router. The following code block shows you how we did that.

```
  networking = {
    hostName = "bgp-client-0";
    defaultGateway = "192.168.111.1";
    interfaces.enp0s8 = {
      useDHCP = false;
      ipv4 = {
        addresses = [{
          address = "192.168.111.2";
          prefixLength = 24;
        }];
      };
    };
  };
```

Testing

To to make sure the confiruation was successful we have to send a package from CL0 to CL1 and get an answer. We do this by using the following command on the CL0.

```
ping 192.168.222.2
```

If the ping succesfully returns everything went well. In the other case we have some options to narrow down the error.

First we can look at the log messages of the bgp daemon to see if they are running on each router and resolve any errors there. We do this, as all the folloing options, for all routing VM or the network they are connectet to seperately.

```
systemctl --status bgpd.service
```

For further information about the frr configuration you can run:

```
vtysh
```

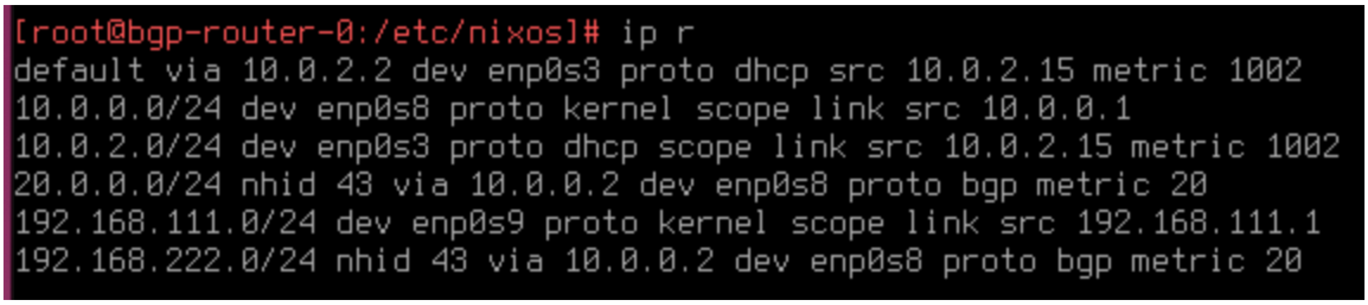
This opens the commandline for the routing daemon. If you run

```
show bgp summary
```

you get information about the confirued neighbors and their status which can give you another hint what maybe is wrong.

When looking at the routing tables of the bgp routers all of the networkspaces shown in the following image should be available (The 10.0.2.0 is from VirtualBox and needet for an interent connection but not necessary for our use case.). Notice that they can be routed over different network interfaces on the 3 routers. So it is good as long as the networks have a routing table entry. You can look at them with:

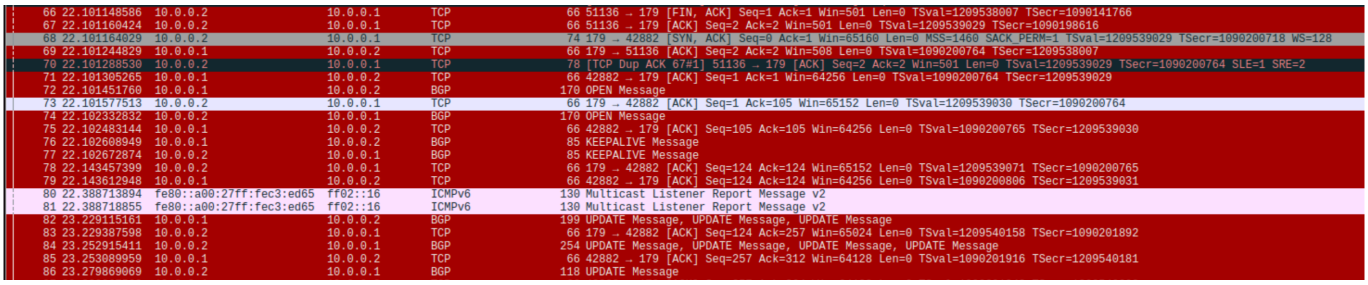
```
ip r
```



```
[root@bgp-router-0:/etc/nixos]# ip r
default via 10.0.2.2 dev enp0s3 proto dhcp src 10.0.2.15 metric 1002
10.0.0.0/24 dev enp0s8 proto kernel scope link src 10.0.0.1
10.0.2.0/24 dev enp0s3 proto dhcp scope link src 10.0.2.15 metric 1002
20.0.0.0/24 nhid 43 via 10.0.0.2 dev enp0s8 proto bgp metric 20
192.168.111.0/24 dev enp0s9 proto kernel scope link src 192.168.111.1
192.168.222.0/24 nhid 43 via 10.0.0.2 dev enp0s8 proto bgp metric 20
```

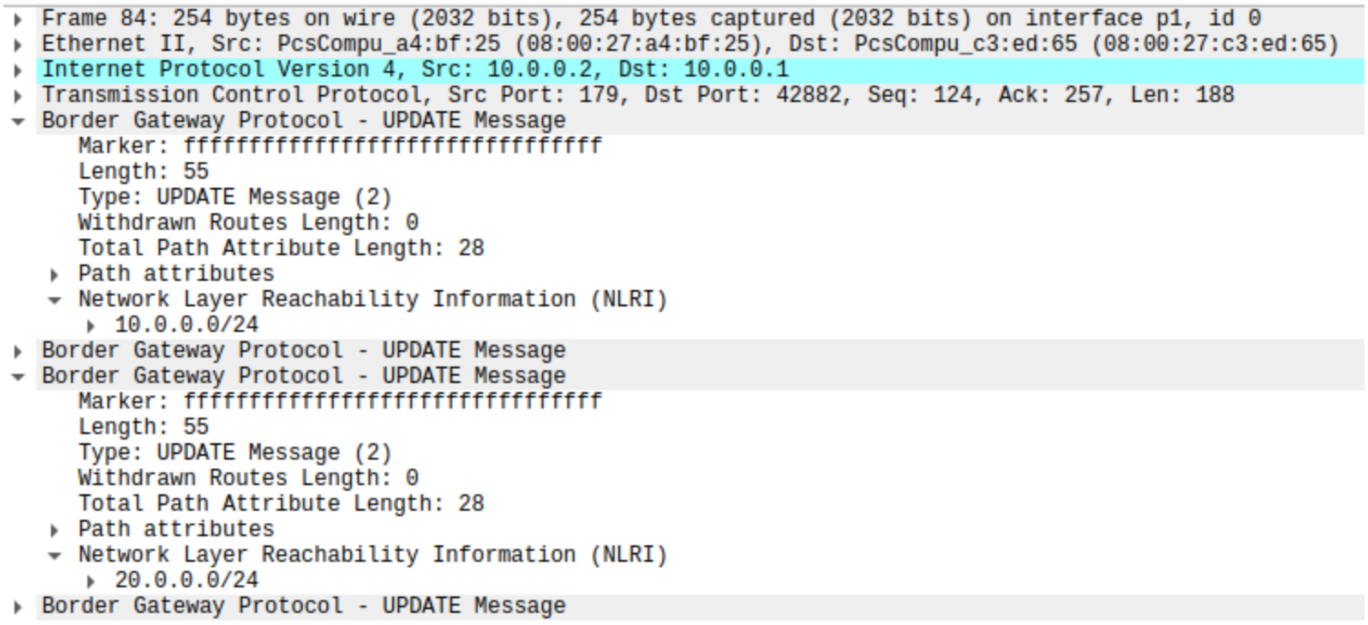
The last step to narrow down the error is to sniff on the network links and look at the bgp connection individually. Therefore we start wireshark on the host system and listen on the links between the routers.

A successful connection starts with a TCP handshake, followed by the BGP Open message, some Keepalive messages and the update messages. The following picture shows a succesfull connection.



```
66 22.101148586  10.0.0.2                   10.0.0.1   TCP      66 51136 → 179 [FIN, ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=1209538007 TSecr=1090141766
67 22.101160424  10.0.0.2                   10.0.0.1   TCP      66 51136 → 179 [ACK] Seq=2 Ack=2 Win=501 Len=0 TSval=1209539029 TSecr=1090198616
68 22.101164029  10.0.0.2                   10.0.0.1   TCP      74 179 → 42882 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1209539029 TSecr=1090200718 WS=128
69 22.101244829  10.0.0.1                   10.0.0.2   TCP      66 179 → 51136 [ACK] Seq=2 Ack=2 Win=508 Len=0 TSval=1090200764 TSecr=1209538007
70 22.101288530  10.0.0.2                   10.0.0.1   TCP      78 [TCP Dup ACK 67#1] 51136 → 179 [ACK] Seq=2 Ack=2 Win=501 Len=0 TSval=1209539029 TSecr=1090200764 SLE=1 SRE=2
71 22.101305265  10.0.0.1                   10.0.0.2   TCP      66 42882 → 179 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1090200764 TSecr=1209539029
72 22.101451760  10.0.0.1                   10.0.0.2   BGP     170 OPEN Message
73 22.101577513  10.0.0.2                   10.0.0.1   TCP      66 179 → 42882 [ACK] Seq=1 Ack=105 Win=65152 Len=0 TSval=1209539030 TSecr=1090200764
74 22.102332832  10.0.0.2                   10.0.0.1   BGP     170 OPEN Message
75 22.102483144  10.0.0.1                   10.0.0.2   TCP      66 42882 → 179 [ACK] Seq=105 Ack=105 Win=64256 Len=0 TSval=1090200765 TSecr=1209539030
76 22.102608949  10.0.0.1                   10.0.0.2   BGP      85 KEEPALIVE Message
77 22.102672874  10.0.0.2                   10.0.0.1   BGP      85 KEEPALIVE Message
78 22.143457399  10.0.0.2                   10.0.0.1   TCP      66 179 → 42882 [ACK] Seq=124 Ack=124 Win=65152 Len=0 TSval=1209539071 TSecr=1090200765
79 22.143612948  10.0.0.1                   10.0.0.2   TCP      66 42882 → 179 [ACK] Seq=124 Ack=124 Win=64256 Len=0 TSval=1090200806 TSecr=1209539031
80 22.388713894  fe80::a00:27ff:fec3:ed65  ff02::16   ICMPv6  130 Multicast Listener Report Message v2
81 22.388718855  fe80::a00:27ff:fec3:ed65  ff02::16   ICMPv6  130 Multicast Listener Report Message v2
82 23.229115161  10.0.0.1                   10.0.0.2   BGP     199 UPDATE Message, UPDATE Message, UPDATE Message
83 23.229387598  10.0.0.2                   10.0.0.1   TCP      66 179 → 42882 [ACK] Seq=124 Ack=257 Win=65024 Len=0 TSval=1209540158 TSecr=1090201892
84 23.252915411  10.0.0.2                   10.0.0.1   BGP     254 UPDATE Message, UPDATE Message, UPDATE Message, UPDATE Message
85 23.253089959  10.0.0.1                   10.0.0.2   TCP      66 42882 → 179 [ACK] Seq=257 Ack=312 Win=64128 Len=0 TSval=1090201916 TSecr=1209540181
86 23.279869069  10.0.0.2                   10.0.0.1   BGP     118 UPDATE Message
```

If you open the update messages and dig into them you should find the anouncement of the other networks. This information is listed unter network layer reachability information.



```
▸ Frame 84: 254 bytes on wire (2032 bits), 254 bytes captured (2032 bits) on interface p1, id 0
▸ Ethernet II, Src: PcsCompu_a4:bf:25 (08:00:27:a4:bf:25), Dst: PcsCompu_c3:ed:65 (08:00:27:c3:ed:65)
▸ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
▸ Transmission Control Protocol, Src Port: 179, Dst Port: 42882, Seq: 124, Ack: 257, Len: 188
▾ Border Gateway Protocol - UPDATE Message
     Marker: ffffffffffffffffffffffffffffffff
     Length: 55
     Type: UPDATE Message (2)
     Withdrawn Routes Length: 0
     Total Path Attribute Length: 28
   ▸ Path attributes
   ▾ Network Layer Reachability Information (NLRI)
     ▸ 10.0.0.0/24
▸ Border Gateway Protocol - UPDATE Message
▾ Border Gateway Protocol - UPDATE Message
     Marker: ffffffffffffffffffffffffffffffff
     Length: 55
     Type: UPDATE Message (2)
     Withdrawn Routes Length: 0
     Total Path Attribute Length: 28
   ▸ Path attributes
   ▾ Network Layer Reachability Information (NLRI)
     ▸ 20.0.0.0/24
▸ Border Gateway Protocol - UPDATE Message
```

After you combined all the information of the options before you should be able to resolve the error.

Problems

While implemnting this szenario we had the problem that sometimes wireshark did not show any packages. To solve that problem you need to restart every VM and wireshark. This did not happend if you started wireshark first.

VyOS Configuration

Router 0

```
set interfaces ethernet eth3 address 10.0.0.1/24
```

Assigns IP address 10.0.0.1/24 to Ethernet interface eth3.

```
set interfaces ethernet eth1 address 192.168.111.1/24
```

Assigns IP address 192.168.111.1/24 to Ethernet interface eth1.

```
set protocols bgp system-as 65000
```

Sets local BGP router's AS number to 65000.

```
set protocols bgp neighbor 10.0.0.2 ebgp-multihop '2'
```

Enables eBGP multihop with a maximum hop count of 2 for neighbor 10.0.0.2.

```
set protocols bgp neighbor 10.0.0.2 remote-as '65001'
```

Sets remote BGP neighbor's AS number to 65001 for neighbor 10.0.0.2.

```
set protocols bgp neighbor 10.0.0.2 update-source '10.0.0.1'
```

Sets the update source for neighbor 10.0.0.2 to IP address 10.0.0.1.

```
set protocols bgp neighbor 10.0.0.2 address-family ipv4-unicast
```

Configures the BGP neighbor 10.0.0.2 for the IPv4 unicast address family.

```
set protocols bgp address-family ipv4-unicast network '192.168.111.0/24'
```

Advertises network 192.168.111.0/24 in the BGP IPv4 unicast address family.

```
set protocols bgp parameters router-id '10.0.0.1'
```

Sets the router ID for the BGP routing process to 10.0.0.1.

Router 1

```
set interfaces ethernet eth0 address 10.0.0.2/24
```

Assigns IP address 10.0.0.2/24 to Ethernet interface eth0.

```
set interfaces ethernet eth1 address 20.0.0.2/24
```

Assigns IP address 20.0.0.2/24 to Ethernet interface eth1.

```
set protocols bgp system-as 65001
```

Sets local BGP router's AS number to 65001.

```
set protocols bgp neighbor 10.0.0.1 ebgp-multihop '2'
```

Enables eBGP multihop with a maximum hop count of 2 for neighbor 10.0.0.1.

```
set protocols bgp neighbor 20.0.0.1 ebgp-multihop '2'
```

Enables eBGP multihop with a maximum hop count of 2 for neighbor 20.0.0.1.

```
set protocols bgp neighbor 10.0.0.1 remote-as '65000'
```

Sets remote BGP neighbor's AS number to 65000 for neighbor 10.0.0.1.

```
set protocols bgp neighbor 20.0.0.1 remote-as '65002'
```

Sets remote BGP neighbor's AS number to 65002 for neighbor 20.0.0.1.

```
set protocols bgp neighbor 10.0.0.1 update-source '10.0.0.2'
```

Sets the update source for neighbor 10.0.0.1 to IP address 10.0.0.2.

```
set protocols bgp neighbor 20.0.0.1 update-source '20.0.0.2'
```

Sets the update source for neighbor 20.0.0.1 to IP address 20.0.0.2.

```
set protocols bgp neighbor 10.0.0.1 address-family ipv4-unicast
```

Configures the BGP neighbor 10.0.0.1 for the IPv4 unicast address family.

```
set protocols bgp neighbor 20.0.0.1 address-family ipv4-unicast
```

Configures the BGP neighbor 20.0.0.1 for the IPv4 unicast address family.

```
set protocols bgp parameters router-id '10.0.0.2'
```

Sets the router ID for the BGP routing process to 10.0.0.2.

Router 2

```
set interfaces ethernet eth0 address 20.0.0.1/24
```

Assigns IP address 20.0.0.1/24 to Ethernet interface eth0.

```
set interfaces ethernet eth1 address 192.168.222.1/24
```

Assigns IP address 192.168.222.1/24 to Ethernet interface eth1.

```
set protocols bgp system-as 65002
```

Sets local BGP router's AS number to 65002.

```
set protocols bgp neighbor 20.0.0.2 ebgp-multihop '2'
```

Enables eBGP multihop with a maximum hop count of 2 for neighbor 20.0.0.2.

```
set protocols bgp neighbor 20.0.0.2 remote-as '65001'
```

Sets remote BGP neighbor's AS number to 65001 for neighbor 20.0.0.2.

```
set protocols bgp neighbor 20.0.0.2 update-source '20.0.0.1'
```

Sets the update source for neighbor 20.0.0.2 to IP address 20.0.0.1.

```
set protocols bgp neighbor 20.0.0.2 address-family ipv4-unicast
```

Configures the BGP neighbor 20.0.0.2 for the IPv4 unicast address family.

```
set protocols bgp address-family ipv4-unicast network '192.168.222.0/24'
```

Advertises network 192.168.222.0/24 in the BGP IPv4 unicast address family.

```
set protocols bgp parameters router-id '10.0.0.3'
```

Sets the router ID for the BGP routing process to 10.0.0.3.

Client 0

```
set interfaces ethernet eth0 address 192.168.111.2/24
```

Assigns IP address 192.168.111.2/24 to Ethernet interface eth0.

```
set protocols static route 0.0.0.0/0 next-hop 192.168.111.1
```

Configures a static route where any traffic with a destination of 0.0.0.0/0 (default route) will be forwarded to the next hop IP address 192.168.111.1.

Client 1

```
set interfaces ethernet eth0 address 192.168.222.2/24
```

Assigns IP address 192.168.222.2/24 to Ethernet interface eth0.

```
set protocols static route 0.0.0.0/0 next-hop 192.168.222.1
```

Configures a static route where any traffic with a destination of 0.0.0.0/0 (default route) will be forwarded to the next hop IP address 192.168.222.1.

Approach

NixOS installation

To set up the scenarios, we choose VirtualBox as our virtualization tool. VirtualBox is a free and open-source software that enables the creation and operation of multiple virtual machines on a single physical computer. This choice helps minimize research costs since no specialized equipment is required.

To begin, we downloaded the minimal ISO of NixOS from the official website: https://nixos.org/download.html#nixos-iso

We selected the minimal ISO because it offers lighter system requirements, which is beneficial when running multiple virtual machines simultaneously. There is also no the need for graphical software in our scenarios. Because of that, the command-line interface suits our purposes.

Next, we installed the ISO onto a newly created virtual machine, following the instructions provided on the NixOS website: https://nixos.org/manual/nixos/stable/index.html#sec-installation-manual-partitioning.

Once the virtual machine was operational, we proceeded to create virtual duplicates of it. These duplicates served as the starting point for our scenarios.

VyOS to NixOS transformer

What it does?

To complete our proof of concept we designed a translator which takes a VyOS configruation as an input and gives you a NixOS configuration as an output. This is possible for our use cases:

1. DHCP server
2. Bonding
3. Wireguard configuration
4. BGP router

How to use it?

1. To obtain the VyOS configuration of a running system you can utilize one of the following commands:

```
show configuration > config.json
show configuration json pretty >config.json # only from version 1.3
available
```

(Before proceeding, ensure that Python 3 is installed on your NixOS system.)

2. Next, transfer the 'config.json' file along with the '/transformer/*' directory from this repository to your active NixOS system. Ensure that the 'config.json' file is located next to the 'transformer.py' file.

3. Proceed by running the 'transformer.py' script. After completion, it will generate a 'configuration.nix' file.

4. Copy the 'configuration.nix' file to '/etc/nixos/configuration.nix' and apply it to the system by executing:

```
nixos-rebuild switch
```

How does it work?

The algorithm converts VyOS configuration files in JSON format to Nix configuration files in Nix format. It accomplishes this by taking into account the VyOS configuration and mappings, which are created for each transformation scenario. The VyOS configuration is extracted from an active system, allowing the algorithm to work directly with real-world data. On the other hand, the mappings are manually designed and stored in the "/transformer/mappings/" directory. These mappings follow a specific syntax, explained in the Mapping syntax section. The mapping files can be expanded to cover additional services or functionalities, providing flexibility for future extensions.

The transformer itself is divided into three parts.

The preprocessor is responsible for collecting the data and processing the representation of it. Notably, it analyses the network interface names of the NixOS system to ensure that they are configured with the appropriate name in the nix configuration.

The mainprocessor is responsible for applying the specified rules of the mapping files to the extracted information from vyos and translate it to nixos config. This involves an iterative algorithm that scans for matches from the mapping file within the VyOS data. The algorithm supports two types of placeholders. The first type, $X (where X is any positive integer starting from 0), stores the value of the VyOS configuration at that place in a list indexed correspondingly, which later supplement the Nix path. The second type involves the use of regular expressions. Here, the algorithm compares the value from the VyOS configuration at that place with the regex. If a match is found, the algorithm continues comparing the paths from VyOS with the mapping path until it verifies a full match. The use of regular expressions also allows for group matches with round brackets. These groups are added separately to the list of stored values and can be utilized within the NixOS configuration path of the mapping. For each match, the configuration is immediately translated and stored in the NixOS configuration.

The postprocessor, is designed to handle special configuration files for the use cases DHCP and BGP. When the postprocessor detects a configuration syntax for any of the services, it initiates a special treatment. These configurations, despite included in the configuration.nix file, follow a different syntax and require special considerations during creation. The mapping for these transformers is integrated into the code and relies on templates equipped with placeholders. These templates are stored in the following dictionaries in this repository: '/bgpTemplates' and '/dhcpTemplates'. When information is detected, it is immediately injected into the template. To guarantee a functioning syntax for the services, all lines lacking values are removed at the end of the process. Similar to the main transformer, these templates require a functional VyOS configuration.

The reason why we splitted these cases from the mainprocessor are the different mapping language requirements for these configurations. For instance, unlike pure Nix config syntax, DHCP and BGP syntax can contain repeated sections requiring unique mapping treatment and programming logic like loops in a mapping language. Furthermore the needed values for DHCP need to change the representation style of the subnet mask from '/24' to '255.255.255.0' or calculate a broadcast address from the given network address and the subnet mask.

After all the processing stages, the main file combines all the configurations into a single string and exports it to the 'configuration.nix'.

Mapping syntax

This JSON structure defines a mapping file that helps to translate or convert VyOS configurations into NixOS configurations. The mapping syntax is designed to be as simple as possible while still providing the necessary functionality. To acomplish this we decided to use a JSON format.

```
{
    "interfaces": {
        "bonding": {
            "$0": {
                "member": {
                    "interface": {
                        "vyosValue": "$1",
                        "nixosPath": "networking.bonds.$0.interfaces=$1"
                    }
                }
            }
        }
    }
}
```

For each interface type like 'bonding', it specifies how values are mapped from VyOS to NixOS. The "$X", where X is any positive number starting from 0, are placeholders for actual values that will be replaced during the conversion process. The "vyosValue" key represents the value from the VyOS configuration, while the "nixosPath" key represents the path to the corresponding value in the NixOS configuration.

```
{
    "interfaces": {
        "ethernet": {
            "$0": {
                "address": {
                "vyosValue": "^dhcp$",
                "nixosPath": "networking.interfaces.$0.useDHCP=true"
                }
            }
        }
    }
}
```

One special feature is the usage of regular expressions. In the example above, the VyOS value is checked against the regex "^dhcp$". If the value matches the regex, the corresponding NixOS path is applied. This feature allows to implement if statements in the mapping process.

```
{
    "interfaces": {
        "wireguard": {
            "$0": {
                "peer": {
                    "client": {
                        "address": {
                        "additionalVyOSPath":
["interfaces#wireguard#$2#port=$3"],
                        "vyosValue": "$1",
                        "nixosPath":
"networking.wireguard.interfaces.$0.peers.endpoint=$1:$3"
                        }
                    }
                }
            }
        }
    }
}
```

The last feature is the usage of additional paths. In the example above, the VyOS confiuration needs to have two matches to apply the NixOS path. First, the regular path according to the hierarchical structure of the VyOS configuration. Secondly all the paths that are given in the "additionalVyOSPath" value. This is a list of strings, where each stage is divided by a '#' and the "vyosValue" is seperated by a '='. Within that string you can continue using the '$X' placeholders as well as regular expressions. This feature allows to extract values from different positions in the VyOS configuration and save them into one NixOS path.

Comparison

This section will discuss the benefits of using NixOS for VyOS functionality.

Project ideas rise from problems. One problem of using VyOS as a router software is to create your own executable. Despite VyOS is open source it is very hard to build your own version of it. The reason for this is that VyOS has a lot of dependencies and packages which are not easy to install. The VyOS documentation provides a guide to build your own version of VyOS. This guide is very long and requires a lot of knowledge about the VyOS system. Building your own version of VyOS basically has two main adavnatages for network administrators. First you can customize the VyOS system to your needs. Second and more important you can audit the code for security vulnerabilities and be sure that no changes have been made to the code before compiling.

Based on this motivation we now want to conclude our internship with a comparison of VyOS and NixOS. We will compare the two systems by the following criteria:

Work effort to build nixos and vyos from source code

As described above building VyOS is cumbersome. NixOS on the other side rebuilds itself every time you apply the configuration. In NixOS it is even possible to build an ISO with your own configuration, which can then be used for later installations. This is a clear advantage for NixOS.

Usability of a running system and learning curve

This section will discuss the usability while using each system once they are running. This includes the learning curve to get familiar with the syntax and the complexity of the syntax.

Starting with NixOS the syntax for the Nix intern configuration is easy to understand. The complexity starts at the point when you want to add aditional services that are not directly included and need special configurations for the deamons, like dhcp and bgp. These configurations follow their own syntax. So for each service you need to learn a new one. VyOS on the other side has a unified and simplified syntax for all services. This makes it easy to learn and understand. Setting up the scenarios for this project was three times faster for VyOS than for NixOS. Comparing the lerning curves of both systems, they both start with the same rise but at some point VyOS leaves NixOS behind beause the syntax stays the same. Applying a new configuration is easy and fast for NixOS and VyOS. Therefore you just need to use one/two commands. Looking at the functionality of both systems, VyOS probably has the same range of functions as NixOS. But using all functions in NixOS requires far more knowledge in different services and deamons. This is because VyOS is specialized on networking and NixOS is a general purpose operating system. To conclude this section, VyOS has a clear advantage in usability and learning curve.

Maintainance cost and security aspects

As discussed above NixOS has it advantages in the building process and VyOS in the usability. But there are also other aspects to consider. One of them is the maintainance cost. This includes the effort to update the system and the security aspects. Updating NixOS is automatically done every time you rebuild the system with the new configuration. VyOS on the other side requires a manual update. This is a slightly advantage for NixOS. Becuase both distributions are open source the security risks of malicius code are very low. But the risk of security vulnerabilities is higher for NixOS. This is because NixOS is a general purpose operating system and therefore has more attack vectors. VyOS on the other side is specialized on networking and therefore has less attack vectors. In this comparison they are both almost equal.

Summary

Our internship project entailed an analysis and comparison of NixOS and VyOS, two distinct Linux distributions, in an attempt to optimize router software deployment. Each system carries its own distinct strengths and weaknesses.

The complexity of building VyOS from source is a clear disadvantage. On the other hand, NixOS offers a more streamlined and automated building process, rebuilding itself every time a configuration is applied. Additionally, NixOS can generate an ISO with user-specific configurations. In terms of usability and the learning curve, VyOS takes the lead. Despite the simplicity of NixOS's internal configuration syntax, its complexity spikes when incorporating additional services. In contrast, VyOS maintains a unified and simple syntax across all services, enabling quicker set up and reducing the learning curve.

However, the versatility of NixOS and VyOS might be restricted in certain contexts. NixOS is a general-purpose operating system that can handle various tasks but may require more extensive knowledge and time to configure specific functionalities. VyOS, on the other hand, being a specialized networking OS, is more straightforward in its networking functions but might not be as flexible. In the end, NixOS and VyOS both leverage the Linux kernel, so they have similar capabilities and can run on the same machines with similar drivers.

Given the specialized nature of VyOS, we do not recommend replacing it with NixOS, despite NixOS's more manageable building process. The time potentially saved during environment setup is outweighed by the ease of configuration and modification during operation offered by VyOS. We, therefore, value usability and functional range over the trustworthiness gained by self-building the system with NixOS. For individuals prioritizing security and trustworthiness, the choice between using NixOS or attempting to build VyOS themselves should be determined by project size. The latter option is particularly recommended for projects requiring diverse services and the potential for future network capability expansion.

Future research in this field could take two directions. The first involves creating a NixOS configuration capable of building the VyOS ISO from source code, effectively bypassing the cumbersome manual building process of VyOS while reaping the benefits of both operating systems.

Alternatively, researchers could consider the utilization of a JSON processor like "https://jqlang.github.io/jq/" or an XSLT transformer "https://www.w3.org/TR/xslt-30/#json" to engineer a generic transformer for all configuration files.