n2n Layer 2 VPN Tunnel mit Verschlüsselung

Installation

Installation unter Debian

Beschreibung:

die installation ist so einfach, wie sie für einen Tunnel noch nie da gewesen ist.
Einfach paket installieren, fertig.

Das automatische starten des Tunnels wird mit post-up befehlen in der Netzwerkkonfig realisert..
keine unnötigen startscripte etc.
Und das beste auch noch einfach für ansible bau bar.

Installation:

Das paket n2n und falls man später benötigt das Paket bridge-utils

```
apt install n2n bridge-utils
```

Das Routing sollte in der sysctl auch aktiviert sein, wenn ip routing und nicht Layer 2 benötigt wird

```
nano /etc/sysctl.conf
```

Inhalt am Ende anfügen

```
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
```

nun anwenden

```
sysctl -p
```

Das wars schon.

Inbetriebnahme

Server starten

n2n besteht aus einem Server Dienst der nur listener ist, möchte man das der Server also da wo der Listener ist auch ein VPN Teilnehmer wird, **braucht er auch eine Verbindung (edge Client)!**
Dieser listener wird supernode genannt

Server starten, mit nohup bringen wir das ding in den Hintergrund dann den Pfad zu supernode -l gibt den port an. Dann geben wir das ganze auch noch in eine logfile, damit wir darin wenn nötig debuggen können

```
nohup /usr/bin/supernode -l {{ supernode_port }} > /var/log/supernode.log 2>&1 &


Beispiel:


nohup /usr/bin/supernode -l 5555 > /var/log/supernode.log 2>&1 &
```

Damit läuft der Listenener auch schon.
um den Server zu beenden den Prozess einfach killen mit pkill

```
pkill supernode
```

In einer /etc/network/interfaces würde das ganze so aussehen. Sobald das interface up ist, startet der Server.
Wird das interface beendet wird der server gekillt

```
auto enp6s18
iface enp6s18 inet static
    address 192.168.178.138
    netmask 255.255.255.0
    gateway 192.168.178.1
    dns-nameservers 8.8.8.8 8.8.4.4
    post-up nohup /usr/bin/supernode -l 5555 > /var/log/supernode.log 2>&1 &
    pre-down /usr/bin/pkill supernode
```

Verbindungen aufbauen

mit dem prgramm edge, hier die Parameter

```
  /usr/sbin/edge -r -d <netzwerkdevicename> -c "layer2" -k "meinultrageheimespasswort>" -a vpn-seite-ip -l "supernodehost:supernodeport" -f > /var/log/n2n_edge

  Parameter:
  -r das routen automatisch erstellt werden sollen
  -d wie das netzwerkdevice heißen soll, dies wird mit ip a angezeigt. wir nennen es einfach n2n0
  -c community namen muss bei allen teilnehmern gleich sein, die in eine gruppe sollen
  -k das super sicher hash passwort um so länger um so besser, muss auch bei allen gleich sein, die in eine gruppe sollen
  -a die eigen interne ip des vpn adapters
  -l dieip/hostname des supernode:denport des supernodes
  -f soll im fordergrund laufen
  >  /var/log/n2n_edge  ausgabe in log file

  Hier der ganze Befehl
  /usr/sbin/edge -r -d n2n0 -c "layer2" -k "supergeheimespasswort" -a 10.10.2.1 -l "167.235.xxx.xxx:5555" -f > /var/log/n2n_edge

  Und schon baut er eine Verbindung zum supernode auf
  am zweiten client der gleiche befehl nur die ip ist anderns, beispiel
  /usr/sbin/edge -r -d n2n0 -c "layer2" -k "supergeheimespasswort" -a 10.10.2.2 -l "167.235.xxx.xxx:5555" -f > /var/log/n2n_edge
```

mit ip a hier vom zeiten client sieht man die netzwerkkonfig:

```
 n2n0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc fq_codel master br0 state UNKNOWN group default qlen 1000
    link/ether 4a:30:8d:45:50:db brd ff:ff:ff:ff:ff:ff
    inet 10.10.2.2/24 brd 10.10.2.255 scope global n2n0
       valid_lft forever preferred_lft forever
    inet6 fe80::4830:8dff:fe45:50db/64 scope link
       valid_lft forever preferred_lft forever
```

Nun sollten wir in der lage sein, 10.10.2.1 anzupingen.
Mit dem Befehl

```
  /usr/bin/pkill edge
```

Können wir die Verbindung wieder beenden

Verbindung über die interfaces mittels post-up Sektion starten

```
auto enp6s18
iface enp6s18 inet static
    address 192.168.178.138
    netmask 255.255.255.0
    gateway 192.168.178.1
    dns-nameservers 8.8.8.8 8.8.4.4
    post-up /usr/sbin/edge -r -d n2n0 -c "layer2" -k "meingeheimespasswort" -a 10.10.2.2 -l
"167.235.xxx.xxx:5555" -f > /var/log/n2n_edge
    pre-down /usr/bin/pkill edge
```

Damit ist die Grundlegende installation abgeschlossen

N2N Interface an eine Brücke kleben

Beschreibung:

Der sinn eines layer 2 Tunnels ist das der Ganze Netzwerkverkehr geroutet wird, also kein ip sondern ethernet frame.
Damit aber unser vpn interface auch irgendwo raus kann und nicht nur im tunnel gefangen ist, erstellen wir eine Brücke mit einem normalen interface, also das was irgendwo raus kann und unser n2n0 interface mit dran.
Das auf beiden seiten. und schon haben wir Sinnbildlich ein Netzwerkkabel von A nach B gelegt, übers Internet natürlich.

Inbetriebnahme

Das Paket bridge-utils haben wir schon installiert.

Auch hier packen wir das wieder in unsere interfaces Datei.
Wir erstellen eine neue bridge und packen an diese bridge unseren verbindungsaufbau und wenn das unser Server ist auch den supernode.
Denn erst wenn die Bridge fertig ist erstellen wir den tunnel und assignen ihn. Wir können den n2n0 Tunnel nicht sofort in der config mit dran packen, denn er existiert noch nicht, aber die Netzwerkkarte schon.

Beispiel interfaces:

Erläuterung:
Ein bridge ist wie eine Netzwerkkarte.
hier bekommt sie unsere Adresse die die Netzwerkkarte sonst hätte samt gateway netmask dns
Unter bridged ports wer hätte es gedacht stehen nachher die devices. Unsere Netzwerkkarte und das n2n0 Device
Wir tragen aber nur die echte Netzwerkkarte ein.
Die anderen Befehle kennen wir schon, ist das device up dann supernode erstellen, sollte es der supernode server sein.

(hier ist es der Fall)
im pre-down den supernode wieder killen

im post-up die edge verbindung herstellen
und nun kommt das neue, nachdem die edge verbindung hergestellt wurde, mittels brctl die n2n0 device zur bridge hinzufügen
im predown das interface von der brücke wieder entfernen.
Dann zu guter letzt edge killen.
Schon haben wir unsere n2n0 device an eine brücke gepackt mit einem anderen netzwerkinterface

```
auto eth0
iface eth0 inet manual

auto br0
iface br0 inet static

    address 167.235.xxx.xxx
    netmask 255.255.255.255
    gateway 172.31.1.1
    dns-nameservers 8.8.8.8 8.8.4.4

    bridge_ports eth0
    bridge_stp off
    bridge_fd 0
    bridge_maxwait 0

    post-up nohup /usr/bin/supernode -l 5555 > /var/log/supernode.log 2>&1 &
    pre-down /usr/bin/pkill supernode

    post-up /usr/sbin/edge -r -d n2n0 -c "layer2" -k "7473535ghbfdsAq!" -a 10.10.2.1 -l "167.235.xxx.xxx:5555" -f
> /va>
    post-up /bin/sh -c "brctl addif br0 n2n0"
    pre-down /bin/sh -c "brctl delif br0 n2n0"
    pre-down /usr/bin/pkill edge
```

Hinweis: Wenn das n2n0 interface an eine Brücke hängt, ist das pingen auf den internen adressen nicht mehr möglich, außer man hängt dei internen Adressen an die bridge.

bei einem layer 2 tunnel interessieren uns die ips auch nicht.
Denn eijn layer 2 tunnel kann gleichgestellt werden wie ein switch.
Die netzwerkports die mit der Brücke verbunden sind , sind die buchsen vom switch.
Würde man eine dritte verbindung noch von irgendwo anders aufbauen, können alle 3 Standorte miteinander kommunizieren als wären sie an einem switch.

Fertig.

Wekzeuge zum Überprüfen des status:

arping:

Augabe:

Unicast reply from 78.47.xxx.xxx [FA:B6:CE:7E:64:D5] 21.255ms
Unicast reply from 78.47.xxx.xxx [FA:B6:CE:7E:64:D5] 20.561ms
Unicast reply from 78.47.xxx.xxx [FA:B6:CE:7E:64:D5] 3.325ms
Unicast reply from 78.47.xxx.xxx [FA:B6:CE:7E:64:D5] 4.039ms
Unicast reply from 78.47.xxx.xxx [FA:B6:CE:7E:64:D5] 2.805ms
Unicast reply from 78.47.xxx.xxx [FA:B6:CE:7E:64:D5] 4.669ms

wie wir sehen ist das die MAC Adresse des ersten Adapters, aber wir haben das doch eigentlich auf enps19 gelegt,
denn enp6s19 ist doch mit dem netzwerkadpter des wir nennen es mal vswitch PC verbunden.
jep aber das arp kommt vom vswitch enp6s20 an das enp6s19 im server an, da aber alle interfaces doert erreichbar sind, gibt es die mac des ersten wieder.
Aber wir sehen das arp findet zur richtigen maschine.
Würden wir aber zum beispiel ein vlan auf de enp6s19 packen,
würde das arp immer noch funktionieren, allerdings die Pakete kommen nicht mehr an enp6s20 an.

brctl show

Ausgabe vom vswitch

bridge name□bridge id□□STP enabled□interfaces
br0□□8000.3e9283b89594□no□□enp6s20
□□□□□□□   n2n0

Hiermit können wir überprüfen ob auch beide Netzwerkadapter an die Brücke geheftet sind

trace route

Ausgabe:

traceroute to 78.47.xxx.xxx (78.47.xxx.xxx), 30 hops max, 60 byte packets
1 fritz.box (192.168.178.1) 18.739 ms 18.697 ms 24.687 ms
2 85.16.121.38 (85.16.121.38) 26.205 ms 27.366 ms 27.396 ms
3 bbrt-ol-0-90730207-ae26.ewe-ip-backbone.de (85.16.251.206) 27.341 ms 27.333 ms 27.324 ms
4 bbrt-ol-0-1-90730201-ae4.ewe-ip-backbone.de (80.228.90.49) 27.317 ms 27.310 ms 27.303 ms
5 bbrt-ffm-0-23730205-ae11.ewe-ip-backbone.de (212.6.114.38) 35.071 ms 35.064 ms 35.057 ms
6 decix-gw.hetzner.com (80.81.192.164) 35.048 ms 10.851 ms 10.819 ms
7 core12.nbg1.hetzner.com (213.239.252.26) 19.413 ms 19.395 ms 28.227 ms
8 * * *
9 spine2.cloud1.nbg1.hetzner.com (213.133.112.198) 19.348 ms spine1.cloud1.nbg1.hetzner.com
(213.133.112.194) 20.421 ms spine2.cloud1.nbg1.hetzner.com (213.133.112.198) 20.412 ms
10 * * *
11 12205.your-cloud.host (116.203.161.48) 17.632 ms 22.230 ms 22.212 ms
12 static.115.118.47.78.clients.your-server.de (78.47.xxx.xxx) 35.191 ms 33.650 ms 33.603 ms

Installation

N2N bridge Setup via Ansible

Beschreibung:

Hier ein script mit dem man relativ einfach ein setup aufbauen kann.
Einfach nur die hosts.ini richtig ausfüllen.
Diese kann um unendliche clients erweitert werden.
Wichtig ist nur das für jeden client eine eigene Gruppe erstellt wird, weil Gruppen Variablen nutzen um so flexibler zu sein.

Installation:

Es muss auf den hosts zugriff ber Schlüssel von nöten sein.
Die Netzwerkinterfaces ip-adressen gateways etc müssen bekannt sein und dem entsprechen in der hosts ini abgelegt sein.

Die host.ini

```
[client1]
peer1 ansible_host=167.235.xxx.xxx
[client2]
peer2 ansible_host=192.168.178.138

[all:vars]
community=layer2
password=ultrageheimeskennwort
supernode_port=5555
supernode_ip=167.235.xxx.xxx
ansible_user=root
debug=false

[client1:vars]
delete_cloud_init=true
bridgeaddress=167.235.xx.xxx
netmask=255.255.255.255
gateway=172.31.1.1
dns1=8.8.8.8
dns2=8.8.4.4
```

```
main_bridge_interface=eth0
n2n_ip=10.10.2.1
supernode=true
# Optional: Für die zweite Netzwerkkarte
secondary_interface=
secondary_ip=
secondary_netmask=
secondary_gateway=

[client2:vars]
delete_cloud_init=false
bridgeaddress=
netmask=
gateway=
dns1=8.8.8.8
dns2=8.8.4.4
main_bridge_interface=enp6s20
n2n_ip=10.10.2.2
supernode=false
# Optional: Für die zweite Netzwerkkarte

secondary_interface=enp6s18
secondary_ip=192.168.178.138
secondary_netmask=255.255.255.0
secondary_gateway=192.168.178.1
```

Kurze Erläuterung:

Die Variable delete_cloud_init besagt wenn die true ist, wird die cloud init Netzwerkkonfig gelöscht und deaktiviert.
Hier ist sie auf Hetzner VPS ausgelegt, ob die Konfig bei anderen VPS anbietern auch so ist, weiß ich nicht

Das main_bridge_interface sagt aus welche pysikalisch (bei ner vm die Netzwerkkarte) an die Bridge dran soll.
Ist eine zweite Netzwerkkarte ausgefüllt, so bekommt die bridge keine ip selbst wenn dieses ausgefüllt ist.
Soll keine zweite Karte genutzt werden, dann die variablen leer lassen.
Auch für die zweite Karte werden die DNS server verwendet die oben angeben sind, also ausfüllen bitte.

Die n2n_ip ist die interne ip, selbst wenn bei brücken keine interne ip von nöten ist, ist sie ein Pflichtfeld.

Möchten wir einen dritten client haben einfach einen client:vars abschnitt kopieren und einfügen und dann eine 3 draus machen.

in der Regel möchte man bei den anderen Clients immer eine separate Netzwerkkarte haben. Nur beim VPS-Server hat man in der regel nur eine.

Beispiel:

```
[client3:vars]
delete_cloud_init=false
bridgeaddress=
netmask=
gateway=
dns1=8.8.8.8
dns2=8.8.4.4
main_bridge_interface=enp6s20
n2n_ip=10.10.2.3
supernode=false
# Optional: Für die zweite Netzwerkkarte

secondary_interface=enp6s18
secondary_ip=192.168.178.139
secondary_netmask=255.255.255.0
secondary_gateway=192.168.178.1
```

Nun das template file

interfaces.j2

```
auto {{ main_bridge_interface }}
iface {{ main_bridge_interface }} inet manual

auto br0
iface br0 inet {% if secondary_interface %}manual{% else %}static{% endif %}

{% if not secondary_interface %}

    address {{ bridgeaddress }}
```

```
    netmask {{ netmask }}
    gateway {{ gateway }}
    dns-nameservers {{ dns1 }} {{ dns2 }}
{% endif %}

    bridge_ports {{ main_bridge_interface }}
    bridge_stp off
    bridge_fd 0
    bridge_maxwait 0
    {% if supernode %}

    post-up nohup /usr/bin/supernode -l {{ supernode_port }} > /var/log/supernode.log 2>&1 &
    pre-down /usr/bin/pkill supernode
    {% endif %}

    post-up /usr/sbin/edge -r -d n2n0 -c "{{ community }}" -k "{{ password }}" -a {{ n2n_ip }} -l "{{
supernode_ip }}:{{ supernode_port }}" -f > /var/log/n2n_edge
    post-up /bin/sh -c "brctl addif br0 n2n0"
    pre-down /bin/sh -c "brctl delif br0 n2n0"
    pre-down /usr/bin/pkill edge

{% if secondary_interface %}

auto {{ secondary_interface }}
iface {{ secondary_interface }} inet static
    address {{ secondary_ip }}
    netmask {{ secondary_netmask }}
    gateway {{ secondary_gateway }}
    dns-nameservers {{ dns1 }} {{ dns2 }}
{% endif %}
```

Und zum schluss das Playbook

```
---
- hosts: all
  become: yes
  tasks:
    - name: Zeige alle Variablen
      debug:
        var: hostvars[inventory_hostname]
```

```
    when: debug | default(false) | bool

  - name: Aktualisiere APT-Repository
    apt:
      update_cache: yes

  - name: Installiere n2n Paket
    apt:
      name: n2n
      state: present

  - name: Installiere bridge-utils Paket
    apt:
      name: bridge-utils
      state: present

  - name: Entferne cloud-init Netzwerkkonfiguration auf Debian
    file:
      path: /etc/network/interfaces.d/50-cloud-init.cfg
      state: absent
    when: delete_cloud_init | bool

  - name: Erstelle 99-cloud-init-disable-Datei
    copy:
      dest: /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
      content: "network: {config: disabled}\n"
    when: delete_cloud_init | bool



  - name: Template the interfaces file
    template:
      src: interfaces.j2
      dest: /etc/network/interfaces
    notify:
      - restart network

handlers:
  - name: restart network
    service:
      name: networking
```

```
    state: restarted
```

Fertig.
nun kann mittels, das playbook gestartet werden und ausgerollt werden

```
ansible-playbook -i hosts.ini setup_n2n.yml
```

Hier Dateien zum download:

interfaces.j2

setup_n2n.yml

Die Host.ini oben aus dem text kopieren.

Nützliches / Tests

Layer2 Paket senden

Beschreibung:

Kleines python script mit dem man ein layer2 Datenpaket senden kann.

Bedienung:

```
sendframe -h
usage: sendframe.py [-h] -d DEVICE -m {senden,empfangen}



-d device = eth0 eth1 enp6s0 n2n0 etc
-m senden oder empfangen

Beispiel Ziel Computer im empfangsmodus zuerst starten :
./sendframe.py -d n2n0 -m empfangen
Dann auf dem Quellcomputer paket senden
./sendframe.py -d n2n0 -m senden
```

Das Script auch zum sendframe.py

```
#!/usr/bin/python3

import socket
import sys
import argparse
import signal



def signal_handler(sig, frame):
    print('Programm wurde mit STRG+C beendet.')
```

```
    sys.exit(0)

def send_frame(iface):
    # Erstellt einen RAW-Socket
    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW)
    # Bindet den Socket an die angegebene Schnittstelle
    s.bind((iface, 0))

    # Beispiel eines Ethernet-Frames mit einer leeren Nutzlast
    # Ethernet Header: Ziel MAC, Quelle MAC, Ethertype (0x0800 für IPv4)
    dst_mac = b'\xff\xff\xff\xff\xff\xff'  # Broadcast MAC-Adresse
    src_mac = s.getsockname()[4]
    ethertype = b'\x08\x00'
    payload = b'Hello, network!'
    frame = dst_mac + src_mac + ethertype + payload

    # Sendet den Frame
    s.send(frame)
    print(f'Paket gesendet auf {iface}')

def receive_frame(iface):
    # Erstellt einen RAW-Socket
    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003))
    # Bindet den Socket an die angegebene Schnittstelle
    s.bind((iface, 0))

    print(f'Warte auf Pakete auf {iface}...')
    while True:
        # Empfängt den Frame
        frame, addr = s.recvfrom(65535)
        src_mac = frame[6:12]
        print(f'Paket empfangen von MAC: {":".join(format(x, "02x") for x in src_mac)}')

def main():
    parser = argparse.ArgumentParser(description='Einfaches Skript zum Senden und Empfangen von Ethernet-
Frames')
    parser.add_argument('-d', '--device', required=True, help='Netzwerkgerät (z.B. eth0)')
    parser.add_argument('-m', '--mode', required=True, choices=['senden', 'empfangen'], help='Modus: senden
oder empfangen')
```

```
    args = parser.parse_args()

    # Registriert das STRG+C Interrupt-Signal
    signal.signal(signal.SIGINT, signal_handler)

    if args.mode == 'senden':
        send_frame(args.device)
    elif args.mode == 'empfangen':
        receive_frame(args.device)

if __name__ == '__main__':
    main()
```

N2N Neigboarhood

Beschreibung:

Das Programm LLDPd (Link Layer Discovery Protocol daemon) ist ein Open-Source-Daemon, der das Link Layer Discovery Protocol (LLDP) implementiert. LLDP ist ein standardisiertes Netzwerkprotokoll, das von Netzwerkgeräten verwendet wird, um ihre Identität, Fähigkeiten und Nachbarschaftsinformationen auf dem Netzwerk weiterzugeben. Hier sind die Hauptfunktionen und Merkmale von LLDPd:

1. **Geräteidentifikation**: LLDPd ermöglicht es Netzwerkgeräten, Informationen wie den Gerätenamen, die Portbezeichnung und die Systembeschreibung an andere Geräte im Netzwerk weiterzugeben. Dies erleichtert die Verwaltung und das Troubleshooting von Netzwerken.
2. **Topologieerkennung**: Mit LLDPd können Netzwerkadministratoren eine detaillierte Übersicht über die Netzwerkstruktur erhalten. Geräte, die LLDP unterstützen, senden regelmäßig LLDP-Informationen aus, die von anderen Geräten empfangen und angezeigt werden können.
3. **Kapazitätenaustausch**: Geräte können über LLDPd Informationen über ihre Fähigkeiten und Ressourcen austauschen, wie z.B. die unterstützten Geschwindigkeiten, die Duplex-Konfiguration und andere technische Spezifikationen.
4. **Energieverwaltung**: LLDPd unterstützt auch LLDP-MED (Media Endpoint Discovery), eine Erweiterung von LLDP, die zusätzliche Funktionen wie die Verwaltung von Energieeinstellungen und die Priorisierung von Netzwerkverkehr für Voice-over-IP (VoIP) bietet.
5. **Integration mit Netzwerkmanagement-Tools**: LLDPd kann in verschiedene Netzwerkmanagement-Tools integriert werden, um eine zentrale Verwaltung und Überwachung von Netzwerken zu ermöglichen.
6. **Konfigurationsdateien und Plugins**: LLDPd verwendet konfigurierbare Dateien und unterstützt Plugins, die zusätzliche Funktionen hinzufügen oder das Verhalten des Daemons anpassen können.
7. **Multiplattformunterstützung**: LLDPd ist für verschiedene Betriebssysteme verfügbar, einschließlich Linux, FreeBSD und anderen Unix-ähnlichen Systemen.

Zusammengefasst dient LLDPd dazu, die Verwaltung und Überwachung von Netzwerken zu erleichtern, indem es detaillierte Informationen über die Netzwerkgeräte und ihre Verbindungen bereitstellt.

Wir nutzen es um zu schauen ob der n2n Tunnel funkltioniert

Installation:

```
apt-get install lldpd


#Dieses dann an der Brücke anwenden
lldpd -I br0


#Nun die Ausgabe mit:
lldpcli show neighbors
```

Ausgabe (Dieser Befehl wurde auf n2n Node A ausgefüjhrt und zeigt den neigbour n2n node B im lokalen netz:

```
-------------------------------------------------------------------------------
LLDP neighbors:
-------------------------------------------------------------------------------
Interface:    n2n0, via: LLDP, RID: 2, Time: 12 days, 14:36:25
  Chassis:
    ChassisID:    mac 26:da:23:37:41:42
    SysName:      pubxxxxxxx.local
    SysDescr:     Debian GNU/Linux 11 (bullseye) Linux 5.10.0-23-amd64 #1 SMP Debian 5.10.179-1 (2023-05-12)
x86_64
    MgmtIP:       192.168.178.xxx
    MgmtIface:    3
    MgmtIP:       fd00::949e:xxxx:fe27:xxxx
    MgmtIface:    3
    Capability:   Bridge, on
    Capability:   Router, off
    Capability:   Wlan, off
    Capability:   Station, off
  Port:
    PortID:       mac 4a:30:8d:xx:xx:xx
    PortDescr:    n2n0
    TTL:          120
-------------------------------------------------------------------------------
```