# ALT - Einrichtung eines Gateway-Servers

Wir verwendet seit geraumer Zeit Ansible zur Server Konfiguration und der Inhalt dieser Seite wurde nicht weiter gepflegt. Details findest du unter https://github.com/FreiFunkMuenster/Ansible-Freifunk-Gateway. Sprich uns bei Fragen gerne im Forum an.

In diesem Artikel wird beschrieben, wie du einen Server für Freifunk Münsterland als Gateway-Server mit fastd einrichten kannst.

Für den Zugang zum Internet benötigen die meisten Freifunk-Netze mindestens zwei Serverdienste:

- Mindestens einen Zielserver für die VPN-Tunnel der Freifunk-Knoten (Oft Super-Node genannt)
- Mindestens ein Gateway um die Daten ins Internet zu leiten

Beim Freifunk Münsterland werden aktuell beide Dienste auf einem Server installiert. Daher ist der genutzte Begriff "Gateway-Server" technisch gesehen nicht ganz richtig.

# Installation

## Die Wahl der Linux-Distribution

Wir verwenden bisher auf unseren Gateways Debian Wheezy, haben daher auch die meiste Erfahrung damit, wie ein Gateway unter dieser Distribution zu betreiben ist. Es sollte aber auch möglich sein, nach dieser Anleitung ein Gateway unter Ubuntu einzurichten. Bei anderen Distributionen ist deutlich mehr eigene Recherche nötig.

## Paketquellen

Wir verwenden folgende zusätzliche Paketquellen auf unseren Servern:

```
deb http://repo.universe-factory.net/debian/ sid main
```

```
deb http://download.opensuse.org/repositories/home:/fusselkater:/ffms/Debian_7.0/ /
```

Zusätzlich benötigen wir die wheezy-Backports für libjson-c2 (Abhängigkeit von fastd >= 15)

```
deb http://http.debian.net/debian wheezy-backports main
```

Um die nötigen Schlüssel hinzuzufügen, mache folgendes:

```
gpg --keyserver pgpkeys.mit.edu --recv-key 16EF3F64CB201D9C
gpg -a --export 16EF3F64CB201D9C | apt-key add -
```

```
wget http://download.opensuse.org/repositories/home:fusselkater:ffms/Debian_7.0/Release.key
apt-key add - < Release.key
```

Nun sind die benötigten Repos eingetragen und du kannst mit

```
aptitude update
```

die Paketquellen aktualisieren.

## Notwendige Pakete

Folgende Pakete müssen nun auf deinem Gateway installiert werden:

| Paket | Beschreibung |
|---|---|
| sudo | Ausführung von Befehlen mit Root-Rechten |

| | |
|---|---|
| bridge-utils | Verwaltung von Netzwerkbrücken |
| batctl=2013.4.0-1 | B.A.T.M.A.N. Verwaltungstools |
| bird | BGP Routing |
| haveged | Entropie |
| fastd | VPN für Verbindung zu den Knoten |
| radvd | IPv6 Router Advertisements |
| isc-dhcp-server | DHCP Server |
| bind9 | DNS Server |
| git | Versionsverwaltungssystem |
| alfred | A.L.F.R.E.D. Datenübertragung |
| alfred-json | A.L.F.R.E.D. Datenausgabe |
| batman-adv-dkms | Batman-Kernelmodul in Version 2013.04 |
| nagios-nrpe-server | Statusüberwachung |
| ntp | Zeitsynchronisation |
| tinc | Inter-City VPN |
| iptables | Paketfilter Userland |

Oder als ein Befehl:

```
aptitude install sudo bridge-utils batctl=2013.4.0-1 bird haveged fastd radvd isc-dhcp-server bind9 git alfred
alfred-json batman-adv-dkms nagios-nrpe-server ntp
```

Tinc wird in der Version 1.11~pre11 benötigt.
Eine für Debian Wheezy geeignete version muss entsprechend `https://gist.github.com/mweinelt/efff4fb7eba1ee41ef2d` manuell Kompiliert werden.
Die Installation erfolgt ebenfalls manuell mittels `dpkg -i <dateiname>.deb`

## IP Forwarding aktivieren

Konfigurationsdatei `/etc/sysctl.d/forwarding.conf`
<file - forwarding.conf>

1. IPv4 Forwarding

net.ipv4.ip_forward=1

1. IPv6 Forwarding

net.ipv6.conf.all.forwarding = 1
</file>

Jetzt solltest du noch einmal rebooten, um diese Änderung zu aktivieren.

## batman-adv Kernelmodul automatisch laden

In die Datei `/etc/modules` müssen die folgenden Zeilen ergänzt werden um das batman-adv Kernelmodul beim Start des Servers automatisch zu laden:

```
batman-adv
```

## batman-adv Version Prüfen

Nach dem Reboot solltest du die batman-adv Version prüfen. Diese muss derzeit `2013.4.0` sein.

```
modinfo batman-adv
```

# Netzwerkkonfiguration

Das Loopback Interface muss durch die öffentliche IP des Gateway Servers ergänzt werden

<file - interfaces>

1. The loopback network interface

auto lo
iface lo inet loopback

```
        up ip address add <öffentliche ipv4 die wir vom FF Rheinland erhalten>/32 dev lo
```

</file>

Als nächstes brauchen wir eine Netzwerkbrücke, als Schnittstelle zwischen dem Mesh-Netz und dem Internet-Uplink.

Dazu füge in die Konfigurationsdatei `/etc/network/interfaces` folgendes hinzu:
<file - interfaces>

1. Netwerkbruecke fuer Freifunk
2. - Hier laeuft der Traffic von den einzelnen Routern und dem externen VPN zusammen
3. - Unter der hier konfigurierten IP ist der Server selber im Freifunk Netz erreichbar
4. - bridge_ports none sorgt dafuer, dass die Bruecke auch ohne Interface erstellt wird

```

```

auto br0

iface br0 inet static

```
        address 10.43.0.z
        netmask 255.255.0.0
        bridge_ports none
```

iface br0 inet6 static

```
        address 2a03:2260:115::z
        netmask 48
```

</file>

Das z ist hier gegen die Zahl zu ersetzen, die wir dem neuen Gateway geben möchten.

Als nächstes muss das bat0-Interface konfiguriert werden. Dazu bearbeitest du wieder die `/etc/network/interfaces`
<file - interfaces>

1. Batman Interface
2. - Erstellt das virtuelle Inteface fuer das Batman-Modul und bindet dieses an die Netzwerkbruecke
3. - Die unten angelegte Routing-Tabelle wird spaeter fuer das Routing innerhalb von Freifunk (Router/VPN) verwendet

allow-hotplug bat0

iface bat0 inet6 manual

```
        pre-up modprobe batman-adv
        post-up ip link set dev bat0 up
        post-up brctl addif br0 bat0
        post-up batctl it 10000
        post-up ip rule add from all fwmark 0x1 table 42
```

</file>

Zur Erklärung:
Bevor das Netzwerkinterface gestartet wird, wird zur Sicherheit nochmal batman-adv geladen.

Nachdem das Interface gestartet ist, wird eine IP-Regel angelegt, die besagt, dass alle Pakete, die über das bat0-Interface eingehen, und mit 0x1 markiert sind, über die Routing-Tabelle 42 geleitet werden. Dies ist wichtig, damit die Pakete aus dem Mesh-VPN wirklich über das MULLVAD-VPN wieder raus gehen, und nicht direkt über deinen Server.

Am Ende wird noch alfred und batadv-vis gestartet. Diese dienen zur Sammlung der Daten für die Knotenkarte.

Als nächstes müssen noch die GRE Tunnel zu den Servern des Rheinland Backbone erstellt werden.
Die zugewiesenen Daten stehen hier: intern:backbone_rheinland
Entsprechend der zugewiesenen Endpunkte werden pro Server zwei Tunnel erstellt.

<file - interfaces>

1. GRE Tunnel zum Rheinland Backbone
2. - Die Konfigurationsdaten werden vom Rheinland Backbone vergeben und zugewiesen

auto tun-ffrl-???
iface tun-ffrl-??? inet static

```
        address <lokale IPv4 Adresse im Tunnel>
        netmask 255.255.255.254
        pre-up ip tunnel add $IFACE mode gre local <IPv4 Adresse dieses Servers> remote <IPv4 Adresse der
Gegenstelle> ttl 255
        post-up ip link set $IFACE mtu 1400
        post-down ip tunnel del $IFACE
```

iface tun-ffrl-??? inet6 static

```
        address <lokale IPv6 Adresse im Tunnel>/64
        netmask 64
```

</file>

Um diese Änderung zu aktivieren, startest du das Netzwerk einmal per `service networking restart` neu.

## IPtables-Regeln

Nun brauchst du noch IPtables-Regeln. Lege dazu die Konfigurationsdatei `/etc/iptables.up.rules` an, und trage folgendes ein.

Damit werden alle Pakete, die über die Bridge rein kommen, mit dem 0x1-Flag markiert, und damit über Routing-Tabelle 42 geschickt. Das bedeutet, diese gehen nicht über die standard Routing-Tabelle. Außerdem gibt es noch 2 Regeln für DNS, dass auch DNS-Pakete (Port 53 TCP/UDP) über die Tabelle 42 geschickt werden. Ansonsten würden DNS-Anfragen über deine normale Internetverbindung raus gehen.

<file - iptables.up.rules>
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
COMMIT

1.

Regeln zum markieren eingehender Pakete

```
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -p tcp --tcp-flags SYN,RST SYN -o tun+ -j TCPMSS --set-mss 1280
-A PREROUTING -i br0 -j MARK --set-xmark 0x1/0xffffffff
-A OUTPUT -o eth0 -p udp --dport 53 -j MARK --set-xmark 0x1/0xffffffff
-A OUTPUT -o eth0 -p tcp --dport 53 -j MARK --set-xmark 0x1/0xffffffff
COMMIT
```

2.

Route nach extern per nat.
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
~~A POSTROUTING -o tun+~~ -j SNAT --to-source <öffentliche IPv4 des Gateway>
COMMIT
</file>

Diese Regel sorgt dafür, dass dein Router auch das NAT übernimmt.

Als letztes musst du dafür sorgen, dass die IPtables-Regeln auch geladen werden. dazu erzeuge die Datei `/etc/network/if-pre-up.d/iptables` mit folgendem Inhalt:
<file bash iptables>
#!/bin/sh
/sbin/iptables-restore < /etc/iptables.up.rules
</file>

Und mache sie danach ausführbar:

```
chmod +x /etc/network/if-pre-up.d/iptables
```

Jetzt lädst du noch fix einmal die IPtables-Regeln via

```
iptables-restore < /etc/iptables.up.rules
```

## Einrichtung des BGP Daemon

Der Austausch von Routen mit dem Rheinland Backbone erfolgt über BGP.
Hierfür wird der Routing Daemon bird verwendet.

Es müssen die folgenden Konfigurationsdateien erstellt werden:

<file - /etc/bird.conf>
log syslog all;
router id <Freifunk Interne IPv4 des Servers, z.B. 10.43.0.5>;

protocol direct {

```
        interface "*";
```

};

protocol kernel {

```
        device routes;
        import all;
        export all;
        kernel table 42;
```

};

protocol device {

```
        scan time 8;
```

};

function is_default() {

```
        return (net ~ [0.0.0.0/0]);
```

};

1. own network

function is_self_net() {

```
    return (net ~ [ 10.43.0.0/16+ ]);
```

}

1. freifunk ip ranges in general

function is_freifunk() {

```
  return net ~ [ 10.0.0.0/8+,
    104.0.0.0/8+
  ];
```

}

filter hostroute {

```
        if net ~ <Öffentliche IPv4 Adresse vom Rheinland Backbone, 185.66.193.48>/32 then accept;
        reject;
```

};

1. ibgp zwischen den gateways

template bgp internal {

```
        local as <Unsere AS Nummer, 65251>;
        import filter {
                preference = 99;
                accept;
        };
        export where source = RTS_BGP;
        gateway direct;
        next hop self;
```

};

1. hier jeweils ein Eintrag zu jedem der anderen Gateways

protocol bgp gw_??? from internal {

```
        neighbor <IP-Adresse innerhalb des FF netztes, z.B. 10.43.0.4> as <Unsere AS Nummer, 65251>;
```

};

1. Uplink über ff Rheinland

template bgp uplink {

```
        local as <Unsere AS Nummer, 65251>;
        import where is_default();
        export filter hostroute;
        next hop self;
        multihop 64;
        default bgp_local_pref 200;
```

};

protocol bgp ffrl_??? from uplink {

```
        source address <Unsere IP in GRE Tunnel, 100.64.0.109>;
        neighbor <IP der Gegenstelle im GRE Tunnel, z.B. 100.64.0.108> as 201701;
```

};

protocol bgp ffrl_??? from uplink {

```
        source address <Unsere IP in GRE Tunnel, 100.64.0.109>;
        neighbor <IP der Gegenstelle im GRE Tunnel, z.B. 100.64.0.108> as 201701;
```

};

1. template for icvpn gateways of other cities

template bgp icvpn {

```
   local as 65251;
   # ignore routes for our own network
   import where (is_freifunk() && !is_self_net());
   export where is_freifunk();
   route limit 10000;
```

};

1. generated icvpn config

include "/var/tmp/bird-icvpn.conf";

</file>

<file - /etc/bird6.conf>
/*

- This is an example configuration file.
- /

1. Yes, even shell-like comments work...

1. Configure logging

log syslog { debug, trace, info, remote, warning, error, auth, fatal, bug };
#log stderr all;
#log "tmp" all;

1. Override router ID

router id <Freifunk Interne IPv4 des Servers, z.B. 10.43.0.5>;

protocol direct {

```
        interface "*";  # Restrict network interfaces it works with
```

}

protocol kernel {

```
        device routes;
        import all;
        export all;             # Default is export none
        kernel table 42;                # Kernel table to synchronize with (default: main)
```

}

protocol device {

```
        scan time 10;           # Scan interfaces every 10 seconds
```

}

function is_default() {

```
        return (net ~ [::/0]);
```

}

1. own networks

function is_self_net() {
return net ~ [ fd68:e2ea:a53::/48+ ];
}

1. freifunk ip ranges in general

function is_freifunk() {
return net ~ [ fc00::/7{48,64},
2001:bf7::/32+];
}

filter hostroute {

```
        if net ~ 2a03:2260:115::/48 then accept;
        reject;
```

}

1. ibgp zwischen den gateways

template bgp internal {

```
        local as <Unsere AS Nummer, 65251>;
        import filter {
                preference = 99;
                accept;
        };
        export where source = RTS_BGP;
        gateway direct;
        next hop self;
```

};

1. pro anderem gateway ein eintrag

protocol bgp gw_??? from internal {

```
        neighbor <Interne IPv6 Adresse des anderen Gateways, z.B. 2a03:2260:115::6> as <Unsere AS-Nummer,
65251>;
```

};

1. Uplink zum FF Rheinland

template bgp uplink {

```
        local as <Unsere AS Nummer, 65251>;
        import where is_default();
        export filter hostroute;
        gateway recursive;
```

}

protocol bgp ffrl_??? from uplink {

```
        description "Rheinland Backbone";
        source address <Unsere IPv6 Adresse im GRE Tunnel, z.B. 2a03:2260:0:3e::2>;
        neighbor <IPv6 Adresse der Gegenstelle im GRE Tunnel, z.B. 2a03:2260:0:3e::1> as 201701;
```

}

protocol bgp ffrl_??? from uplink {

```
        description "Rheinland Backbone";
        source address <Unsere IPv6 Adresse im GRE Tunnel, z.B. 2a03:2260:0:3f::2>;
        neighbor <IPv6 Adresse der Gegenstelle im GRE Tunnel, z.B. 2a03:2260:0:3f::1> as 201701;
```

}

1. template for icvpn gateways of other cities

template bgp icvpn {

```
        local as 65251;
        # ignore routes for our own network
        import where is_freifunk() && !is_self_net();
        export where is_freifunk() || (source = RTS_BGP);
        route limit 10000;
```

};

1. aus ICVPN Meta erzeugte konfiguration

include "/var/tmp/bird6-icvpn.conf";

</file>

Die verwendeten IC-VPN Konfigurationsdaten werden später erzeugt und jetzt erst einmal leer angelegt.

```
touch /var/tmp/bird-icvpn.conf
touch /var/tmp/bird6-icvpn.conf
```

Anschließend kann bird mit `service bird start` und `service bird6 start` gestartet werden.
Die Befehle `birdc show protocols` und `birdc6 show protocols` zeigen den zustand der Verbindungen an.

## Einrichtung des Mesh-VPNs

Für das Mesh-VPN wird fastd eingesetzt. Hierzu erzeugst du als erstes das Konfigurationsverzeichnis:

```
mkdir -p /etc/fastd/vpn/peers
```

Als nächstes erzeugst du die Schlüssel für deinen Server. Dazu führst du folgendes aus:

```
fastd --generate-key
```

dieser Befehl zeigt dir die Schlüssel lediglich an. Heißt du musst sie dir aus dem Terminal kopieren, um sie im nächsten Schritt verwenden zu können.

Jetzt legst du eine Konfigurationsdatei `/etc/fastd/vpn/fastd.conf` mit folgendem Inhalt an:
<file - fastd.conf>

1. Bind to a fixed address and port, IPv4 and IPv6

```
bind EXTERNE-IPv4-ADRESSE:14242 interface "eth0";
bind [EXTERNE-IPv6-ADRESSE]:14242 interface "eth0";
```

1. Set the user, fastd will work as

```
user "nobody";
```

1. Set the interface name

```
interface "mesh-vpn";
```

1. Set the mode, the interface will work as

```
mode tap;
```

1. Set the mtu of the interface (salsa2012 with ipv6 will need 1406)

```
mtu 1406;
```

1. Set the methods (aes128-gcm preferred, salsa2012+umac preferred for nodes)

```
method "aes128-gcm";
method "salsa2012+umac";
method "salsa2012+gmac";
```

1. Secret key generated by `fastd --generate-key`

```
secret "SERVER-SECRET-KEY";
```

1. Log everything to syslog

```
log to syslog level debug;
```

1. Include peers from our git-repos

include peers from "/var/gateway-ffms/backbone/";

1. Status Socket

status socket "/tmp/fastd-status";

1. Configure a shell command that is run on connection attempts by unknown peers (true means, all attempts are accepted)
2. on verify "true";

on verify "

```
  /bin/bash /var/gateway-ffms/fastd/verify.sh $PEER_KEY
```

";

1. Configure a shell command that is run when fastd comes up

on up "

```
  chmod ugo+rw /tmp/fastd-status
  ip link set dev $INTERFACE address de:ad:be:ef:43:XX
  ip link set dev $INTERFACE up
  ifup bat0
  batctl if add $INTERFACE
  batctl gw server DOWNSTREAM/UPSTREAM
  batctl vm server
  ip rule add from ÖFFENTLICHE-IPv4-ADRESSE/METZMASKE lookup 42
  ip -6 rule add from ÖFFENTLICHE-IPv6-ADRESSE/METZMASKE lookup 42
```

";
</file>

Folgendes muss ersetzt werden:

- `EXTERNE-IPv4-ADRESSE` gegen die externe IPv4-Adresse des Gateways
- `EXTERNE-IPv6-ADRESSE` gegen die externe IPv6-Adresse des Gateways
- `SERVER-SECRET-KEY` gegen deinen geheimen Schlüssel
- `XX` gegen die Mac-Adresse, die wir dem Client geben möchten
- `DOWNSTREAM` gegen die Bandbreite deines Downstreams (z.B. `1024Mbit`)
- `UPSTREAM` gegen die Bandbreite deines Upstreams (z.B. `1024Mbit`)
- `ÖFFENTLICHE-IPv4-ADRESSE/METZMASKE` gegen die öffentliche IPv4 Adresse die wir vom Rheinland Backbone erhalten (inkl Netzmaske)
- `ÖFFENTLICHE-IPv6-ADRESSE/METZMASKE` gegen die öffentliche IPv6 Adresse die wir vom Rheinland Backbone erhalten (inkl Netzmaske)

Nun brauchst du die Peers aus unserem GIT-Repository. Wechsle dazu in das Verzeichnis /var und führe folgendes aus:

```
git clone https://github.com/FreiFunkMuenster/gateway-ffms.git
```

Zum testen der fastd-Konfiguration führe folgendes aus:

```
fastd -c /etc/fastd/vpn/fastd.conf
```

Anschließend kannst du fastd mit

```
service fastd start
```

starten.

Den Öffentlichen Schlüssel deines Servers schickst du uns, damit wir ihn einpflegen können.

## Einrichtung des InterCity VPNs

Das InterCity-VPN wird über Tinc aufgebaut.
Dokumentation im Freifunk Wiki

Die Konfiguration der Verbindungspartner ist in dem Git Repo hinterlegt.
Hier muss der neue Server eingetragen werden.

Anschließend wird das Repo https://github.com/freifunk/icvpn auf den lokalen Server geklont.

```
cd /var
git clone https://github.com/freifunk/icvpn
```

Anschließend muss das Verzeichnis `/etc/tinc/icvpn` erstellt werden.

```
mkdir /etc/tinc/icvpn
ln /var/icvpn/hosts /etc/tinc/icvpn/hosts -s
```

Als nächstes werden die Schlüssel für den Server erzeugt.
Die Dateien müssen in dem Verzeichnis `/etc/tinc/icvpn` gespeichert werden.

```
tinc generate-keys
```

Jetzt folgt die Konfigurationsdatei `/etc/tinc/icvpn/tinc.conf`

```
Name = muensterland<Kürzel des Gateways>
PrivateKeyFile = /etc/tinc/icvpn/rsa_key.priv
Mode = Switch
PingTimeout = 30
Port = 656
Hostnames = yes
```

Im Gleichen Verzeichnis muss auch ein Script `/etc/tinc/icvpn/tinc-up` angelegt werden.
Die IP Adresse innerhalb des VPNs muss für den Server muss in der Liste https://wiki.freifunk.net/IC-VPN eingetragen werden.

```
#!/bin/sh
/sbin/ip link set dev $INTERFACE up
/sbin/ip addr add dev $INTERFACE 10.207.X.Y/16 broadcast 10.207.255.255 scope link
/sbin/ip -6 addr add dev $INTERFACE fec0::a:cf:X:Y/96 preferred_lft 0
```

In der Datei `/etc/tinc/nets.boot` wird anschließend folgende Zeile ergänzt damit Tinc das Netzwerk automatisch startet.

```
icvpn
```

TODO: Update Mechanismus für IC-VPN Hosts ??

## IPv6 Router Advertisements einrichten

Alle Gateway-Server teilen das selbe IPv6-Prefix durch Router Advertisements aus. Router Advertisements können maximal ein /64 großes Netz vergeben, weshalb dies hier getan wird.
Lege die Konfigurationsdatei `/etc/radvd.conf` mit folgendem Inhalt an:
<file - radvd.conf>
interface br0
{

```
    AdvSendAdvert on;
    IgnoreIfMissing on;
```

AdvManagedFlag off;

```
    AdvOtherConfigFlag on;
    AdvLinkMTU 1280;
```

```
    prefix 2a03:2260:115::z/64
    {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr on;
```

};

```
    RDNSS 2a03:2260:115::z {
    };
```

};
</file>

Hierbei muss die RDNSS-Adresse auf deinen Server zeigen.

Als letztes kannst du radvd via

```
service radvd restart
```

neustarten.

## DHCPv4 Server einrichten

Dein Server muss als nächstes IPv4-Adressen vergeben. Erzeuge dafür die Konfigurationsdatei `/etc/dhcp/dhcpd.conf` mit folgendem Inhalt:
<file - dhcpd.conf>
default-lease-time 240;
max-lease-time 1200;

authoritative;

log-facility local7;

subnet 10.43.0.0 netmask 255.255.0.0 {

```
    range 10.43.zz.1 10.43.zz.254;
```

option routers 10.43.0.x;

```
    option domain-name-servers 10.43.0.x;
    option interface-mtu 1280;
```

}
</file>

Jeder Gateway-Server erhält einen Teil des IP-Bereiches um Adressen zu vergeben. Diesen Bereich erhälst du bei uns.
Die Optionen Router und Domain-Name-Servers enthalten jeweils die IP deines Servers.

Jetzt muss der DHCP-Server noch auf das Bridge-Interface festgelegt werden. Dazu bearbeitest du die Datei `/etc/default/isc-dhcp-server` und setzt folgende Option:
<file bash isc-dhcp-server>

1. On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
2. Separate multiple interfaces with spaces, e.g. "eth0 eth1".

INTERFACES="br0"
</file>

Nun testest du den DHCP-Server via:

```
dhcpd -f -d
```

und anschließend startest du den DHCP-Server via:

```
service isc-dhcp-server restart
```

## DHCPv6 Server einrichten

Da Windows die RDNSS-Option aus den Router Advertisements nicht kennt, ist es außerdem nötig, einen DHCP6-Server laufen zu lassen. Da Debian derzeit keine Konfigurationsdateien hierfür anbietet, erstellen wir diese selbst.
Lege folgende Dateien an:
`/etc/init.d/isc-dhcp6-server`
<file bash isc-dhcp6-server>
#!/bin/sh
#
#

1. a. i. BEGIN INIT INFO
2. Provides: isc-dhcp6-server
3. Required-Start: $remote_fs $network $syslog
4. Required-Stop: $remote_fs $network $syslog
5. Should-Start: $local_fs slapd $named
6. Should-Stop: $local_fs slapd
7. Default-Start: 2 3 4 5
8. Default-Stop: 0 1 6
9. Short-Description: DHCP6 server
10. Description: Dynamic Host Configuration Protocol Server V6
    a. i. END INIT INFO

PATH=/sbin:/bin:/usr/sbin:/usr/bin

test -f /usr/sbin/dhcpd ::UWCTOKENCOLSPANS:2::| exit 0

DHCPD_DEFAULT="${DHCPD_DEFAULT:-/etc/default/isc-dhcp6-server}"

1. It is not safe to start if we don't have a default configuration...

if [ ! -f "$DHCPD_DEFAULT" ]; then

```
        echo "$DHCPD_DEFAULT does not exist! - Aborting..."
        if [ "$DHCPD_DEFAULT" = "/etc/default/isc-dhcp6-server" ]; then
                echo "Run 'dpkg-reconfigure isc-dhcp-server' to fix the problem."
        fi
        exit 0
```

fi

. /lib/lsb/init-functions

1. Read init script configuration

[ -f "$DHCPD_DEFAULT" ] && . "$DHCPD_DEFAULT"

NAME=dhcpd
DESC="ISC DHCP6 server"

1. fallback to default config file

DHCPD_CONF=${DHCPD_CONF:-/etc/dhcp/dhcpd6.conf}

1. try to read pid file name from config file, with fallback to /var/run/dhcpd.pid

if [ -z "$DHCPD_PID" ]; then

```
        DHCPD_PID=$(sed -n -e 's/^[ \t]*pid-file-name[ \t]*"(.*)"[ \t]*;.*$/\1/p' < "$DHCPD_CONF" 2>/dev/null |
head -n 1)
```

fi
DHCPD_PID="${DHCPD_PID:-/var/run/dhcpd6.pid}"

test_config()
{

```
        if ! /usr/sbin/dhcpd -6 -t $OPTIONS -q -cf "$DHCPD_CONF" > /dev/null 2>&1; then
                echo "dhcpd self-test failed. Please fix $DHCPD_CONF."
                echo "The error was: "
                /usr/sbin/dhcpd -6 -t $OPTIONS -cf "$DHCPD_CONF"
                exit 1
        fi
```

}

1. single arg is -v for messages, -q for none

check_status()
{

```
    if [ ! -r "$DHCPD_PID" ]; then
        test "$1" != -v || echo "$NAME is not running."
        return 3
    fi
    if read pid < "$DHCPD_PID" && ps -p "$pid" > /dev/null 2>&1; then
        test "$1" != -v || echo "$NAME is running."
        return 0
    else
        test "$1" != -v || echo "$NAME is not running but $DHCPD_PID exists."
        return 1
    fi
```

}

case "$1" in

```
        start)
                test_config
                log_daemon_msg "Starting $DESC" "$NAME"
                start-stop-daemon --start --quiet --pidfile "$DHCPD_PID" \
```

1. -exec /usr/sbin/dhcpd – \
2. 6 -q $OPTIONS -cf "$DHCPD_CONF" -pf "$DHCPD_PID" $INTERFACES

```
                sleep 2
```

if check_status -q; then

```
                        log_end_msg 0
                else
                        log_failure_msg "check syslog for diagnostics."
                        log_end_msg 1
                        exit 1
                fi
                ;;
        stop)
                log_daemon_msg "Stopping $DESC" "$NAME"
                start-stop-daemon --stop --quiet --pidfile "$DHCPD_PID"
                log_end_msg $?
                rm -f "$DHCPD_PID"
                ;;
        restart | force-reload)
                test_config
                $0 stop
                sleep 2
                $0 start
                if [ "$?" != "0" ]; then
                        exit 1
                fi
                ;;
        status)
                echo -n "Status of $DESC: "
                check_status -v
                exit "$?"
                ;;
```

- )

```
                echo "Usage: $0 {start|stop|restart|force-reload|status}"
                exit 1
```

esac

exit 0
</file>

Danach mache das Script ausführbar und lege es in den Autostart via:

```
chmod +x /etc/init.d/isc-dhcp6-server
update-rc.d isc-dhcp6-server defaults
```

`/etc/default/isc-dhcp6-server`
<file bash isc-dhcp6-server>

1. Defaults for isc-dhcp-server initscript
2. sourced by /etc/init.d/isc-dhcp-server
3. installed at /etc/default/isc-dhcp-server by the maintainer scripts

#

1. This is a POSIX shell fragment
   #

1. Path to dhcpd's config file (default: /etc/dhcp/dhcpd6.conf).
   #DHCPD_CONF=/etc/dhcp/dhcpd6.conf

1. Path to dhcpd's PID file (default: /var/run/dhcpd6.pid).
   #DHCPD_PID=/var/run/dhcpd6.pid

1. Additional options to start dhcpd with.
2. Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
   #OPTIONS=""

1. On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
2. Separate multiple interfaces with spaces, e.g. "eth0 eth1".

INTERFACES="br0"
</file>

`/etc/dhcp/dhcpd6.conf`
<file - dhcpd6.conf>

1. Enable RFC 5007 support (same than for DHCPv4)

allow leasequery;

1. Global definitions for name server address(es)

option dhcp6.name-servers 2a03:2260:115::z;

subnet6 2a03:2260:115::/64 {
}
</file>

In `option dhcp6.name-servers` gehört die IPv6-Adresse deines Servers.

Als letztes erzeuge die `dhcpd6.leases` Datei via:

```
touch /var/lib/dhcp/dhcpd6.leases
```

Nun kannst du den DHCPv6-Server starten:

```
service isc-dhcp6-server start
```

## InterCity Routen und DNS

Für die nächsten Schritte werden die Repositories `https://github.com/freifunk/icvpn-meta` und `https://github.com/freifunk/icvpn-scripts` benötigt.
Beide Repositories werden erst einmal mit git in `/var` geklont.

```
cd /var
git clone https://github.com/freifunk/icvpn-meta
git clone https://github.com/freifunk/icvpn-scripts
```

Anschließend werden die BGP-Konfigurationen erzeugt und in bird geladen.

```
/var/icvpn-scripts/mkbgp -x muenster -p icvpn_ -s /var/icvpn-meta/ -f bird  -d icvpn -4 > /var/tmp/bird-icvpn.
conf
birdc configure
/var/icvpn-scripts/mkbgp -x muenster -p icvpn_ -s /var/icvpn-meta/ -f bird  -d icvpn -6 > /var/tmp/bird6-icvpn.
conf
birdc6 configure
```

Als nächstes wird eine DNS Konfigurationsdatei erzeugt

```
/var/icvpn-scripts/mkdns -x muenster -x chaosvpn -x dn42  -s /var/icvpn-meta/ -f bind > /var/tmp/named.conf.
icvpn
```

# DNS-Server einrichten

Wir verwenden als DNS-Server bind9. Lege folgende 2 Konfigurationsdateien an:
`/etc/bind/named.conf.options`
<file - named.conf.options>
options {

```
        directory "/var/cache/bind";
```

_ If there is a firewall between you and nameservers you want

```
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113
```

_ If your ISP provided one or more IP addresses for stable

```
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.
```

_ forwarders {

```
        //      0.0.0.0;
        // };
```

_

```
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //=====================================================================
        dnssec-validation auto;

        recursion yes;
        allow-recursion { localnets; localhost; };
```

allow-notify { 10.43.0.10; };

```
        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };
```

};
</file>

`/etc/bind/named.conf.local`
<file - named.conf.local>

_
_ Do any local configuration here
_

_ Consider adding the 1918 zones here, if they are not used in your
_ organization
include "/etc/bind/zones.rfc1918";

1. inter city dns

include "/var/tmp/named.conf.icvpn"

zone "ffms" {

```
        type slave;
        masters { 10.43.0.10; };
        file "/var/tmp/db.ffms.bak";
```

};

zone "nodes.ffms" {

```
        type slave;
        masters { 10.43.0.10; };
        file "/var/tmp/db.nodes.ffms.bak";
```

};

zone "gw.freifunk-muenster.de" {

```
        type slave;
        masters { 10.43.0.10; };
        file "/var/tmp/db.gw.freifunk-muenster.de.bak";
```

};

</file>

Nun kannst du bind9 starten:

```
service bind9 restart
```

## NRPE Daemon einrichten

Die Gateways werden von einem zentralen Nagios Server überwacht.

Nach der Installation müssen in der Konfigurationsdatei `/etc/nagios/nrpe.cfg` die folgenden Zeilen angepasst werden:

```
allowed_hosts=10.43.0.13
```

```
include_dir=/var/gateway-ffms/nrpe/
```

Zusätzlich benötigt der NRPE-Daemon weitere Berechtigungen die in der `/etc/sudoers` vergeben werden müssen:

```
nagios  ALL=NOPASSWD: /usr/sbin/batctl
nagios  ALL=NOPASSWD: /usr/sbin/birdc
nagios  ALL=NOPASSWD: /usr/sbin/birdc6
```

## Statistiken via Munin

Die Statistiken werden auf den Gateways via Munin erzeugt.

Zuerst muss Munin noch installiert werden, diese Installtion sollte erst jetzt, nachdem das Gateway komplett konfiguriert ist erfolgen, da die Interface-Konfigurationen für Munin während der Installation automatisch generiert werden, und man sich arbeit spart, wenn die Netzwerkinterfaces bei der Installation alle bereits existieren.

```
aptitude install munin-node
```

Dazu bearbeite die Konfigurationsdatei `/etc/munin/munin-node.conf` wie folgt:
<file - munin-node.conf>
log_level 4
log_file /var/log/munin/munin-node.log
pid_file /var/run/munin/munin-node.pid

background 1
setsid 1

user root
group root

1. This is the timeout for the whole transaction.
2. Units are in sec. Default is 15 min
   #
3. global_timeout 900

1. This is the timeout for each plugin.
2. Units are in sec. Default is 1 min
   #
3. timeout 60

1. Regexps for files to ignore

```
ignore_file [#~]$
ignore_file DEADJOE$
ignore_file \.bak$
ignore_file %$
ignore_file \.dpkg-(tmp|new|old|dist)$
ignore_file \.rpm(save|new)$
ignore_file \.pod$
```

1. Set this if the client doesn't report the correct hostname when
2. telnetting to localhost, port 4949
   #
   host_name freifunk-gateway-zz

1. A list of addresses that are allowed to connect. This must be a
2. regular expression, since Net::Server does not understand CIDR-style
3. network notation unless the perl module Net::CIDR is installed. You
4. may repeat the allow line as many times as you'd like

```
allow ||127\.0\.0\.1$
allow ||::1$
allow ||2a03:2260:115::10$
```

1. If you have installed the Net::CIDR perl module, you can use one or more
2. cidr_allow and cidr_deny address/mask patterns. A connecting client must
3. match any cidr_allow, and not match any cidr_deny. Note that a netmask
4. **must** be provided, even if it's /32
   #

5. Example:
   #
6. cidr_allow 127.0.0.1/32
7. cidr_allow 192.0.2.0/24
8. cidr_deny 192.0.2.42/32

1. Which address to bind to;
   #host *
2. host 127.0.0.1

host [2a03:2260:115::z]

1. And which port

port 4949
</file>
Hierbei ist der Eintrag `host` gegen die IPv6-Adresse deines Servers im Freifunknetz (also des br0-Interfaces) zu ersetzen, und host_name gegen den gewünschten Hostnamen.

Nun wechselst du in den Ordner `/etc/munin/plugins` und löscht dort alle Plugins, dessen Statistiken du nicht sammeln möchstest.

Zum Schluss startest du munin-node via

```
service munin-node restart
```

neu. Die Statistiken sollten bei dem nächsten Update von Munin auf unserer Seite https:_freifunk-muenster.de/stats/ erscheinen.

## Abschluss

Dein Server ist nun für den Betrieb als Gateway für Freifunk Münster eingerichtet. Am besten startest du den Server einmal neu, und kontrolliere, dass alles korrekt hoch kommt.

## Logging abschalten

"Freifunk steht unter Anderem für Netzneutralität und ist in keinster Weise an irgendwelchen Nutzer-, Meta-, Irgendwasdaten interessiert. Aus diesem Grund muss den sonst so redseligen Linux Daemonen das Logging abgewöhnt werden." (https://gluon-gateway-doku.readthedocs.org/de/latest/configuration/cleanup.html#logging)

Als erstes erzeuge ein "Schwarzes Loch" im im rsyslogd
Suche in `/etc/rsyslog.conf` folgende Zeile:
<file - rsyslog.conf>
**.**;auth,authpriv.none -/var/log/syslog
</file>
und ersetze sie durch
<file - rsyslog.conf>
**.**;auth,authpriv.none;local6.none -/var/log/syslog
</file>
Dann suche den Block
<file - rsyslog.conf>
**.=info;**.=notice;*.=warn;\

```
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
```

</file>
und ändere ihn so, daß er wie folgt aussieht
<file - rsyslog.conf>
**.=info;**.=notice;*.=warn;\

```
        auth,authpriv.none;local6.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
```

</file>
dann füge noch das Folgende am Ende ein:
<file - rsyslog.conf>
#

1. BlackHole
   #
   local6.* /dev/null
   </file>
   Jetzt den Logdämon restarten mit

   ```
   service rsyslog restart
   ```

Als nächstes bringen wir dem dhcpd bei, seine unerwünschte Ausgabe ins schwarze Loch zu schieben.
Dazu editieren wir `/etc/dhcp/dhcpd.conf` und `/etc/dhcp/dhcpd6.conf`. In der Datei dhcpd.conf ändern wir die Log-Facility auf `local6` und und der dhcpd6.conf fügen wir die Log-Facility hinzu.
<file - dhcpd.conf>
log-facility local6;
</file>
<file - dhcpd6.conf>
log-facility local6;
</file>
Zum Aktivieren der Änderung die DHCP neu starten:

```
service isc-dhcp-server restart
service isc-dhcp6-server restart
```

fastd bringen wir jetzt auch zum Schweigen. Editiere `/etc/fastd/vpn/fastd.conf` und lösche die Zeile
<file - fastd.conf>
log to syslog level debug;
</file>
dafür fügst du folgendes ein
<file - fastd.conf>
log level warn;
hide ip addresses yes;
hide mac addresses yes;
</file>

```
service fastd restart
```

abschicken und schon ist da auch Ruhe drin.

Jetzt fehlt nur noch der Nameserver.
Erstelle die Datei `/etc/bind/named.conf.logging` mit folgendem Inhalt:
<file - named.conf.logging>
logging {

```
    channel null { null; };
    category default { null; };
```

};
</file>
und füge in `/etc/bind/named.conf` diese Zeile ein
<file - named.conf>
include "/etc/bind/named.conf.logging";
</file>
abspeichern und

```
service bind9 restart
```

abschicken

Jetzt könnten die Daten auch nicht mehr weitergegeben werden, weil sie gar nicht vorhanden sind.

# Bekannte Fehlerbilder

## Kernelpanik /-oops - Reboot automatisieren

B.A.T.M.A.N. bereitet häufiger Probleme mit Kernelpanic /-oops. Um das Gateway möglichst schnell wieder verfügbar zu machen, habe ich folgendes eingerichtet:

In die `/etc/sysctl.conf` einfügen:

<file - sysctl.conf>
net.ipv4.conf.default.rp_filter=0
net.ipv4.conf.all.rp_filter=0
net.ipv4.tcp_syncookies=1
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
net.ipv4.conf.all.accept_redirects = 1
net.ipv6.conf.all.accept_redirects = 1
net.ipv4.conf.all.secure_redirects = 1
net.ipv4.conf.all.send_redirects = 1
net.ipv4.conf.all.accept_source_route = 1
net.ipv6.conf.all.accept_source_route = 1
net.ipv4.conf.all.log_martians = 0
net.bridge.bridge-nf-call-arptables = 0
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.ipv6.conf.all.autoconf = 0
net.ipv6.conf.default.autoconf = 0
net.ipv6.conf.eth0.autoconf = 0
net.ipv6.conf.all.accept_ra = 0
net.ipv6.conf.default.accept_ra = 0
net.ipv6.conf.eth0.accept_ra = 0
kernel.panic_on_oops = 1
kernel.panic = 1
</file>

Nicht vergessen! `sysctl -p` absetzen, um es auch sofort zu aktivieren.