Freifunk S"u+00f"dholstein Documentation

Release 1.5

Grotax, ffsh

27.12.2021

Inhalte

KAPITEL 1

Infrastruktur

1.1 Netzwerk

Network IPv4: `10.144.0.0/16`

Network IPv6: `fddf:0bf7:80::/48`

1.2 Gateways

Name	ULA	IPv6	RFC1918	DHCP	ICVPN-Transit	Mesh MAC(s)	B.A.T.M.A.N. adv. MAC(s)	Standort	Betreuer	Exit/VPN-Dienst	Status
Barnitz	fddf:0bf7:801:148::	2a03:2260:2010:148::	10.144.48.1	10.144.48.2-10.144.63.254	n/a	00:5b:27:80:00:48	02:00:27:81:00:48	Hetzner (nbg)	ul	VPN	online
Beste	fddf:0bf7:801:164::	2a03:2260:2010:164::	10.144.64.1	10.144.64.2-10.144.79.254	n/a	00:5b:27:80:00:64	02:00:27:81:00:64	Hetzner (fsn)	ul	VPN	online
Bille	fddf:0bf7:801:180::	2a03:2260:2010:180::	10.144.80.1	10.144.80.2-10.144.95.254	n/a	00:5b:27:80:00:80	02:00:27:81:00:80	Nbg-cup	fr	VPN	online
Brunsbach	fddf:0bf7:801:196::	2a03:2260:2010:196::	10.144.96.1	10.144.96.2-10.144.111.254	n/a	00:5b:27:80:00:96	02:00:27:81:00:96	Nbg-cup	be	VPN	online
Heilsau	fddf:0bf7:801:1a2::	2a03:2260:2010:1a2::	10.144.112.1	10.144.112.2-10.144.127.254	n/a	00:5b:27:80:01:12	02:00:27:81:01:12	Hetzner (hel)	ul	VPN	online
Hopfenbach	fddf:0bf7:801:128::	2a03:2260:2010:128::	10.144.128.1	10.144.128.2-10.144.143.254	n/a	00:5b:27:80:01:28	02:00:27:81:01:28				
Krummbach	fddf:0bf7:801:144::	2a03:2260:2010:144::	10.144.144.1	10.144.144.2-10.144.159.254	n/a	00:5b:27:80:01:44	02:00:27:81:01:44	Hetzner(fsn)	ks	VPN	online
Piepenbek	fddf:0bf7:801:160::	2a03:2260:2010:160::	10.144.160.1	10.144.160.2-10.144.175.254	n/a	00:5b:27:80:01:60	02:00:27:81:01:60				
Strusbek	fddf:0bf7:801:176::	2a03:2260:2010:176::	10.144.176.1	10.144.176.2-10.144.191.254	n/a	00:5b:27:80:01:76	02:00:27:81:01:76				
Sylsbek	fddf:0bf7:801:192::	2a03:2260:2010:192::	10.144.192.1	10.144.192.2-10.144.207.254	n/a	00:5b:27:80:01:92	02:00:27:81:01:92	Nbg-cup	ul	VPN	online
Trave	fddf:0bf7:801:208::	2a03:2260:2010:208::	10.144.208.1	10.144.208.2-10.144.223.254	n/a	00:5b:27:80:02:08	02:00:27:81:02:08	Hetzner (fsn)	ul	VPN	online
Viehbach	fddf:0bf7:801:224::	2a03:2260:2010:224::	10.144.224.1	10.144.224.2-10.144.239.254	n/a	00:5b:27:80:02:24	02:00:27:81:02:24				

1.3 Karte

Die Karte kann unter https://map.freifunk-suedholstein.de erreicht werden. Die Karte wird auf dem Gateway Hopfenbach betrieben. Sie basiert auf dem meshviewer von Freifunk Regensburg. Unsere Konfiguration findet man in unserem fork auf GitHub.

Unsere Grafana instanz ist unter https://map.freifunk-suedholstein.de/grafana erreichbar.

```
# meshviewer.json
https://map.freifunk-suedholstein.de/data/meshviewer.json
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# nodes.json (v2)
https://map.freifunk-suedholstein.de/data/nodes.json
# nodelist.json
https://map.freifunk-suedholstein.de/data/nodelist.json
```

1.4 GitHub

Fast alle Daten, welche für den Betrieb notwendig sind, werden unter https://github.com/ffsh gespeichert.

KAPITEL 2

Gateway Konfiguration

2.1 Allgemeines

Warnung: Diese Anleitung ist nicht mehr aktuell, Freifunk Südholstein ist 2021 auf Ansible umgestiegen. Daher wird diese Anleitung nicht mehr aktuallisiert und ist daher mit Vorsicht zu lesen.

Diese Anleitung ist auf Debian 10 ausgerichtet

Es wird neben dem OS nicht viel Plattenplatz auf dem System benötigt. In Vorbereitung zur Installation wird in dieser Anleitung von folgendem ausgegangen:

- Einen Hardware-Server oder ein virtualisiertes System auf dem man Kernel-Module kompilieren und laden kann (z.b. kvm)
- default-Installation von Debian 10
- Einen User auf dem Server mit sudo-Rechten

Alle Befehle werden als der User ausgeführt wenn dies nicht explizit abweichend angegeben ist. In den Beispielen benutzen wir den als Editor `nano`. Es funktioniert natürlich auch jeder andere Texteditor.

2.2 Hinweise zur Nutzung

Diese Dokumentation soll (auch) eine vollständige Anleitung sein um ein Gateway aufzusetzen. Daher

- Ist das gesamte Dokument in der Reihenfolge aufgebaut, in der die einzelnen Elemente benötigt werden
- Es sollten sich alle Befehle per Copy&Paste aus diesem Dokument in die Shell übernehmen lassen, wobei natürlich die individuellen Anpassungen noch von Hand vorgenommen werden müssen. Sollte das der Fall sein wird jeweils auf den Aufruf des Editors hingewiesen.

Um den zweiten Schritt zu ermöglichen wurde ein kleiner „Hack“ angewandt. Da unser Anliegen natürlich immer auch Erkenntnisgewinn ist wollen wir das hier näher erläutern. Grundsätzlich haben alle Kommandos auf der Bash

eine Standard-Eingabe und eine Standard-Ausgabe, die man aber auch manipulieren kann. Wir nutzen das hier mit dem Befehl `cat`. cat macht erst mal nichts anderes als eine Datei zeilenweise zu lesen und auf die Standard-Ausgabe auszugeben. Diese Standard-Ausgabe ist der Bildschirm.

```
cat README.txt
```

Macht also nicht anderes, als die Datei README.txt auf den Bildschirm auszugeben. Diese Ausgabe kann man über > umlenken.

```
cat README.txt > WRITEME.txt
```

Liest den Inhalt aus README.txt und schreibt ihn nach WRITEME.txt. Wichtig ist: Der Befehl überschreibt das Ziel ohne Rückfrage. Wenn es WRITEME.txt also vorher gegeben haben wollte wäre der Inhalt in dem Beispiel komplett überschrieben. Wenn man was an eine bestehende Datei anhängen will nutzt man >>.

```
cat README.txt >> WRITEME.txt
```

Hängt also den Inhalt von README.txt an WRITEME.txt an. Achtung! Wir nutzen hier beide Möglichkeiten, das ist also Absicht ob da 1 oder 2 > verwendet werden.

Ebenso kann man über < auch den Standard-Input verbiegen. Das ist für unseren Anwendungsfall wichtig da wir ja in den meisten Fällen auf dem System gar keine Dateien haben die wir nutzen wollen sondern die ja erst aus dieser Dokumentation erzeugen wollen.

```
cat < README.txt
```

Macht im Prinzip das Selbe wie „cat README.txt“. Im Beispiel ohne „<“ wird der Dateiname als Parameter an des Kommando übergeben. Der Parameter besagt „Lies den Standard-Input aus der Datei mit diesem Namen“ in der Variante mit „<“ erhält der Befehl keinen Parameter, allerdings verbiegt „<“ den Standard-Input von der Tastatur auf die Datei README.txt. Wer das verifizieren möchte kann mal `cat` ohne jeden Parameter ausführen. Dann wird cat starten und Daten vom Standard-Input (=Tastatur) erwarten und an den Standard-Output (=Bildschirm) weiterreichen. Ctrl-C beendet den Befehl.

Dazu nutzen wir eine Konstruktion namens „Heredoc“ bzw. „Herestring“. Dies erlaubt es uns, über die Shell auch mehrzeiligen Text an ein Programm zu übergeben. Kurz gefasst: Hier wird ein Ende-Zeichen angegeben und aller Text bis zu diesem Ende-Zeichen wird an das Programm übergeben. Das sieht dann so aus:

```
cat << EOF > WRITEME.txt
Dies wird die erste Zeile
Dies wird die zweite Zeile
EOF
```

Hier wird „cat“ auf der Standard-Eingabe der folgende Text übergeben und als Standard-Ausgabe wird die Datei WRITEME.txt angegeben.

Einen ziemlich guten Guide, was man mit Stanard-Eingabe und Standard-Ausgabe auf der Shell anstellen kann findet man z.B. unter http://mywiki.wooledge.org/BashGuide/InputAndOutput

Wer sich in der Anleitung nur für den Inhalt der jeweiligen Datei interessiert kann also einfach alles zwischen der „cat-Zeile und dem „EOF“ nehmen und z.B. per Copy&Paste in seinen Lieblings-Editor übertragen.

2.3 Installation der Debain-Pakete

Zunächst installieren wir mal alle Pakete die generell benötigt werden

- eine Build-Umgebung (build-essentials)

- Git (git)
- https-Support für apt (apt-transport-https)
- Zeitsynchronisation (ntp)
- Netzwerk-Tools (bridge-utils, net-tools)

```
sudo apt install build-essential git apt-transport-https ntp bridge-utils ntp net-
↪tools
```

2.4 Netzwerk Konfiguration

2.4.1 Interfaces Konfigurieren

Nun kommt das eigentlich wichtigste. Das Netzwerk muss eingerichtet werden, so das die einzelnen Schnittstelle bereitstehen und eine Art Brücke vom Freifunknetz in das Internet aufbauen.

Hinweis: diese Konfiguration ist allgemeingültig für unser Netz. Daher ist das jeweilige Gateway in den IP-Adressen mit `[GW Netz]` geschrieben. Diese Nummer muss natürlich durchgängig gleich sein, da sonst nichts funktionieren wird!

Interfaces werden in `/etc/network/interfaces` definiert. Es gibt aber die Möglichkeit, Konfigurationen aus eigenen Dateien zu sourcen. Um hier die Konfigurationen sauber zu halten lassen wir die defaults unangetastet. In :code: */etc/network/interfaces* sind unter Debian 10 keine Interfaces mehr angegeben, sie werden aus Dateien in /etc/network/interfaces.d/ gelesen. Hier sollte es eine Datei geben, in der z.B. loopback und eth0 definiert sind. Die fassen wir nicht an, wir erzeugen eine eigene Datei `/etc/network/interfaces.d/60-ffsh-init.cfg`

```
sudo cat << EOF > /etc/network/interfaces.d/60-ffsh-init.cfg
# Netwerkbruecke fuer Freifunk
# - Hier laeuft der Traffic von den einzelnen Routern und dem externen VPN zusammen
# - Unter der hier konfigurierten IP ist der Server selber im Freifunk Netz erreichbar
# - bridge_ports none sorgt dafuer, dass die Bruecke auch ohne Interface erstellt wird

auto br-ffsh
iface br-ffsh inet static
    address 10.144.[GW Netz].1
    netmask 255.255.0.0
    bridge_ports none

iface br-ffsh inet6 static
    address fddf:0bf7:80::[GW Netz]:1
    netmask 64


    post-up /sbin/ip rule add iif br-ffsh table 42
    pre-down /sbin/ip rule del iif br-ffsh table 42

# B.A.T.M.A.NAdvanced Advanced Interface
# - Erstellt das virtuelle Inteface fuer das B.A.T.M.A.N Advanced-Modul und bindet␣
↪dieses an die Netzwerkbruecke
# - Die unten angelegte Routing-Tabelle wird spaeter fuer das Routing innerhalb von␣
↪Freifunk (Router/VPN) verwendet
#
# Nachdem das Interface gestartet ist, wird eine IP-Regel angelegt, die besagt, dass␣
↪alle Pakete, die über das bat0-Interface eingehen,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# und mit 0x1 markiert sind, über die Routing-Tabelle 42 geleitet werden.
# Dies ist wichtig, damit die Pakete aus dem Mesh wirklich über das VPN raus gehen.
#

allow-hotplug bat0
iface bat0 inet6 manual
    pre-up batctl if add ffsh-mesh
    post-up ip link set address 00:5b:27:81:0:[GW Netz] dev bat0
    post-up ip link set dev bat0 up
    post-up brctl addif br-ffsh bat0
    post-up batctl it 10000
    post-up batctl gw server 100mbit/100mbit

    post-up ip rule add from all fwmark 0x1 table 42

    pre-down brctl delif br-ffsh bat0 || true
    down ip link set dev bat0 down
EOF
```

Und hinterher nicht vergessen, die Platzhalter zu ersetzen

```
sudo nano /etc/network/interfaces.d/60-ffsh-init.cfg
```

Die `/etc/hosts` mit Folgenden Zeilen ergänzen:

```
sudo cat << EOF >> /etc/hosts

[externe IP]                [GW Name].freifunk-suedholstein.de    [GW Name]
10.144.[GW Netz].1          ffsh
fddf:0bf7:80::[GW Netz]:1   ffsh
```

Und hinterher nicht vergessen, die Platzhalter zu ersetzen

```
sudo nano /etc/hosts
```

2.4.2 IP Forwarding

In der Konfigurationsdatei `/etc/sysctl.d/forwarding.conf` bitte die folgenden Zeilen eintragen, damit das IP-Forwarding für IPv4 und IPv6 laufen:

```
sudo cat << EOF > /etc/sysctl.d/forwarding.conf
# IPv4 Forwarding
net.ipv4.ip_forward=1

# IPv6 Forwarding
net.ipv6.conf.all.forwarding = 1
EOF
```

2.4.3 IP Tables

Lege die Konfigurationsdatei `/etc/iptables.up.rules` an mit Folgendem:

Damit werden alle Pakete, die über die Bridge rein kommen, mit dem 0x1-Flag markiert, und damit über Routing-Tabelle 42 geschickt. Es gibt noch 2 Regeln für DNS, dass auch DNS-Pakete (Port 53 TCP/UDP) über die Tabelle 42

geschickt werden.

```
sudo cat << EOF > /etc/iptables.up.rules
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
COMMIT
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
COMMIT
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
COMMIT
EOF
```

Nun müssen die IP-Tables geladen werden. Bitte erstellt die Datei `/etc/network/if-pre-up.d/iptables` mit folgenden Zeilen:

```
sudo cat << EOF > /etc/network/if-pre-up.d/iptables
#!/bin/sh
/sbin/iptables-restore < /etc/iptables.up.rules
EOF
```

Bitte nun noch eine Datei `/etc/fastd/ffsh/iptables\_ffsh.sh` erstellen, die alle Routing iptables Vorgaben enthält:

```
sudo cat << EOF > /etc/fastd/ffsh/iptables\_ffsh.sh
#!/bin/sh
/sbin/ip route add 10.144.0.0/16 dev br-ffsh src 10.144.[GW Netz].1 table 42
/sbin/iptables -t nat -I POSTROUTING -s 0/0 -d 0/0 -j MASQUERADE
# IGMP/MLD segmentation
echo 2 > /sys/class/net/bat0/brport/multicast_router
EOF
```

Jetzt müssen die für Linux ausführbar werden.

```
sudo chmod +x /etc/network/if-pre-up.d/iptables
sudo chmod +x /etc/fastd/ffsh/iptables_ffsh.sh

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

2.5 B.A.T.M.A.N Advanced und Fastd

B.A.T.M.A.N Advanced ist das in Südholstein verwendete Routing Verfahren. B.A.T.M.A.N Advanced benötigt ein Kernel Modul und batclt.

Damit B.A.T.M.A.N Advanced bei einem Kernel Update nicht verschwindet oder durch die alte OS-Version ersetzt wird, richten wir das Modul mit `dkms` ein.

2.5.1 B.A.T.M.A.N Advanced Kernel Modul und batctl

Zur Sicherheit prüfen wir erst mal ob es ein geladenes B.A.T.M.A.N Advanced-Modul gibt:

```
lsmod | grep batman
```

Sollte das der Fall sein einmel entladen, wir wollen unser eigened laden:

```
modprobe -rf batman_adv
```

Dann können wir die Pakete installieren die wir für das Bauen des Kernel-Moduls benötigen:

```
sudo apt install linux-headers-amd64 libnl-3-dev libnl-genl-3-dev libcap-dev pkg-
↪config dkms
```

Und nun widmen wir uns dem eigentlichen B.A.T.M.A.N Advanced - Modul.

Erst mal die notwendigen Sourcen runterladen und entpacken:

```
sudo su -
cd /usr/src
wget https://downloads.open-mesh.org/batman/releases/batman-adv-2021.1/batman-adv-
↪2021.1.tar.gz
tar xzfv batman-adv-2021.1.tar.gz

wget https://downloads.open-mesh.org/batman/releases/batman-adv-2021.1/batctl-2021.1.
↪tar.gz
tar xzvf batctl-2021.1.tar.gz

exit
```

Nun wird das B.A.T.M.A.N Advanced-Modul gebaut. Dafür müssen wir uns erst mal eine :code: *dkms.conf* zusammenbauen

```
sudo cat << EOF > /usr/src/batman-adv-2021.1/dkms.conf
PACKAGE_NAME=batman-adv
PACKAGE_VERSION=2021.1

DEST_MODULE_LOCATION=/extra
BUILT_MODULE_NAME=batman-adv
BUILT_MODULE_LOCATION=net/batman-adv

MAKE="'make' CONFIG_BATMAN_ADV_BATMAN_V=n"
CLEAN="'make' clean"

AUTOINSTALL="yes"
EOF
```

Und nun können die Module gebaut und werden:

```
sudo dkms add -m batman-adv -v 2021.1
sudo dkms build -m batman-adv -v 2021.1
sudo dkms install -m batman-adv -v 2021.1
```

Und das dazugehoerende Control- und Management-Tool:

```
cd /usr/src/batctl-2021.1/
make
sudo make install
```

Damit B.A.T.M.A.N Advanced bei jedem Neustart auch geladen wird muss er nun noch in /etc/modules hinterlegt werden:

```
sudo cat << EOF >> /etc/modules
batman_adv
EOF
```

2.6 fastd

fastd v18 ist in Debian 9 und 10 bereits in den Repositories enthalten. Unter Debian 8 findet man es in den jessie-backports.

```
sudo apt install fastd
```

2.6.1 fastd-Konfiguration

Wir brauchen für den neuen Server die Schlüssel für fastd. Diese sind in Südholstein für 12 Gateways bereits in der Firmware eingetragen und den privaten Schlüssel gibt es über das NOC-Team (noc@freifunk-suedholstein.de).

Im Folgenden wird der sichere private Schlüssel als [SERVER-SECRET-KEY] aufgeführt und müssen durch die erzeugten Schlüssel sinnvoll ersetzt werden!

Wir benötigen einen System-User, als der fastd ausgeführt werden kann. Dafür versorgen wir :code: *adduser* mit folgenden Parametern:

–system	Es handelt sich um einen Systemuser
–no-create-home	Es soll kein Homeverzeichnis angelegt werden
–disabled-password	Der User hat kein Passwort
–disabled-login	Anmelden ist nicht möglich
–home /nonexistent	Das Home-Verzeichnis gibt es wirklich nicht

```
sudo adduser --system --no-create-home --disabled-password --disabled-login --home /
↪nonexistent ffsh
```

Es ist eine Konfigurationsdatei für fastd notwendig. In der folgenden Konfiguration bitte die `[EXTERNE-IPv4]` durch die echte IP vom Server ersetzen. Wenn es auch eine IPv6 gibt, kann die entsprechende Zeile aktiviert werden und benötigt die echte IPv6 `[EXTERNE-IPv6]`.

```
sudo cat << EOF > /etc/fastd/ffsh/fastd.conf
# Bind to a fixed address and port, IPv4 and IPv6 at Port 1234
bind any:10000 interface "eth0";
# bind [EXTERNE-IPv6]:1234 interface "eth0";

# Set the user, fastd will work as
user "ffsh";

# Set the interface name
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
interface "ffsh-mesh";

# Set the mode, the interface will work as
mode tap;

# Set the mtu of the interface (salsa2012 with ipv6 will need 1406)
mtu 1426;

# Set the methods (aes128-gcm preferred, salsa2012+umac preferred for nodes)
method "null";
method "salsa2012+umac";

#hide ip addresses yes;
#hide mac addresses yes:

# Secret key generated by `fastd --generate-key`
secret "[SERVER-SECRET-KEY]";

# Log everything to syslog
log to syslog level debug;

# Include peers from our git-repos
#include peers from "peers/"; #optional eigene peers anlegen zb den eigenen toaster␣
↪mit fastd oder so
include peers from "gateways/"; #git repo klonen in /etc/fastd/ffsh/ git clone ␣
↪https://github.com/ffsh/gateways.git

# Configure a shell command that is run on connection attempts by unknown peers (true␣
↪means, all attempts are accepted)
on verify "true";
# on verify "/etc/fastd/fastd-blacklist.sh $PEER_KEY";

# Configure a shell command that is run when fastd comes up
on up "
 ip link set dev $INTERFACE address 00:5b:27:80:0X:XX           # X für das GW Netz,␣
↪zB 2:24 für 10.144.224.0/20
 ip link set dev $INTERFACE up
 ifup bat0
 sh /etc/fastd/ffsh/iptables_ffsh.sh
";

on down "
 ifdown bat0
";
EOF
```

Und noch die Variablen ersetzen:

```
sudo nano /etc/fastd/ffsh/fastd.conf
```

Das Beste ist, wenn man nun die fastd-Konfiguration mal überprüft. Vorher muss der Server neugestartet werden, damit die vorher durchgeführten Anpassungen auch Wirkung zeigen :-)

```
sudo reboot
```

Dann auf der Konsole mit folgender Zeile die fastd Einstellungen prüfen:

```
sudo fastd -c /etc/fastd/ffsh/fastd.conf
```

Bei der Installation von :code: *GW_Bille* ist fastd nicht automatisch gestartet. Er muss als Dienst in systemd hinterlegt werden. Mangels Wissen (und es war schon spät) haben wir uns mit einem Workaround beholfen, sobald wir den korrekten Weg gefunden haben werden wir das auch hier korrigieren

```
sudo nano /etc/default/fastd
```

Dort gibt es einen Parameter :code: *AUTOSTART="none"*, der muss auskommentiert werden. Damit greift der default `all`:

```
# This is the configuration file for /etc/init.d/fastd

#
# This configuration file is DEPRECATED! Please set autostart to "none" in
# this file and use the instanced systemd unit fastd@.service
#

#
# Start only these VPNs automatically via init script.
# Allowed values are "all", "none" or space separated list of
# names of the VPNs. If empty, "all" is assumed.
#
#AUTOSTART="none"
```

Wenn das erfolgreich war, kann nun fastd eingeschaltet und gestartet werden:

```
sudo systemctl enable fastd
sudo systemctl start fastd
```

2.6.2 Zusätzliche Sicherheit - derzeit nicht genutzt !

Wichtig: In der Konfiguration wird jeder Router reingelassen. Das mag nicht jeder, aber es vereinfacht die Integration der Router und damit auch die Verteilung. Wenn man das nicht möchte, müsste jeder Router separat mit seinem öffentlichen Schlüssel unter `.../peers/` hinterlegt werden. Auskommentiert ist eine Zeile bei `on verify` die eine Blacklist führt. Damit kann man unliebsame Genossen aussperren.

```
sudo cat << EOF > /etc/fastd/fastd-blacklist.sh
#!/bin/bash
PEER_KEY=$1
if /bin/grep -Fq $PEER_KEY /etc/fastd/fastd-blacklist.json; then
    exit 1
else
    exit 0
fi
EOF
```

Und da es sich um ein Skript handelt müssen wir es auch ausführbar machen:

```
sudo chmod +x /etc/fastd/fastd-blacklist.sh
```

Wie die weiteren Dateien mit der Blacklist aussehen, findet man unter diesem Link <*https://github.com/ffruhr/fastdbl*>

2.7 DHCP

Jetzt brauchen wir noch den dhcp innerhalb des FFSH-Netzes

```
sudo apt install radvd isc-dhcp-server
```

2.7.1 DHCP radvd IPv6

Es wird für IPv6 die Konfigurationsdatei `/etc/radvd.conf`

```
sudo cat << EOF > /etc/radvd.conf
interface br-ffsh {
    AdvSendAdvert on;
    IgnoreIfMissing on;
    AdvManagedFlag off;
    AdvOtherConfigFlag on;
    MaxRtrAdvInterval 200;
    AdvLinkMTU 1280;
    prefix fddf:0bf7:80::/64 {
        AdvOnLink on;
        AdvAutonomous on;
        AdvRouterAddr on;
    };

    RDNSS fddf:0bf7:80::[GW Netz]:1 {
    };
};
EOF
```

Und die Variablen ersetzen

```
sudo nano /etc/radvd.conf
```

Jetzt kann radvd gestartet werden:

```
sudo service radvd restart
```

2.7.2 DHCP isc-dhcp-server IPv4 und IPv6

Die Konfigurationsdatei `/etc/dhcp/dhcpd.conf` wird für IPv4 mit folgenden Zeilen benötigt:

```
sudo cat << EOF > /etc/dhcp/dhcpd.conf
ddns-update-style none;
option domain-name ".ffsh";

# möglichst kurze Leasetime
default-lease-time 120;
max-lease-time 600;

log-facility local7;

subnet 10.144.0.0 netmask 255.255.0.0 {
    authoritative;
    range 10.144.[GW Netz].2 10.144.[GW Netz + 15].254;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    option routers 10.144.[GW Netz].1;

    option domain-name-servers 10.144.[GW Netz].1; # für die eigenen DNS-Einträge
    # option domain-name-servers 85.214.20.141; # weitere anonyme DNS
    # option domain-name-servers 213.73.91.35;
}

include "/etc/dhcp/static.conf";
```

Und wieder die Variablen anpassen:

```
sudo nano /etc/dhcp/dhcpd.conf
```

Bitte eine leere Datei `/etc/dhcp/static.conf` erzeugen.

```
sudo echo > /etc/dhcp/static.conf
```

Und den User dhcpstatic anlegen

```
sudo useradd -m -s /bin/bash dhcpstatic
```

Und als der User die Static aus dem Git laden

```
sudo su - dhcpstatic

git clone https://github.com/ffsh/dhcp-static.git

chmod +x dhcp-static/updateStatics.sh

exit
```

Und einmal mit dem auch aus dem Git gezogenen Befehl den dhcpd aktualisieren

```
sudo /home/dhcpstatic/dhcp-static/updateStatics.sh
```

Und über cron dafür sorgen dass das regelmäßig passiert.

```
sudo cat << EOF > /etc/cron.d/ffsh_dhcpstatic
*/5 * * * * root /home/dhcpstatic/dhcp-static/updateStatics.sh > /dev/null 2>&1
EOF
```

Auf dem DHCP-Server muss noch das Bridge-Interface für IPv4 festgelegt werden. Bitte die Datei `/etc/default/isc-dhcp-server` mit folgender Option ergänzen:

```
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="br-ffsh"
INTERFACESv6=""
```

Am Besten wird der DHCP-Server vor dem Start und Betrieb noch mal geprüft. Bitte vorher den Server rebooten

```
sudo reboot
```

und dann auf der Konsole folgende Zeile ausführen:

```
sudo dhcpd -f -d
```

War das erfolgreich, so kann der DHCP-Server als root gestartet werden:

```
sudo systemctl restart isc-dhcp-server
```

2.8 DNS-Server (BIND)

```
apt install bind9
```

Für das interne Freifunknetz ist nun noch der DNS-Server bind9 mit den Konfigurationsdateien wie folgt zu konfigurieren:

Erstmal diese Datei `/etc/bind/named.conf.options`

```
sudo cat << EOF > /etc/bind/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 {
        8.8.8.8;
        8.8.4.4;
    };
    //========================================================================
    // 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-enable yes;
    // dnssec-validation yes;
    dnssec-validation no;
    // dnssec-lookaside auto;
    // recursion yes;
    // allow-recursion { localnets; localhost; };
    auth-nxdomain no;    # conform to RFC1035
    listen-on-v6 { any; };
};
EOF
```

Dann in der Datei `/etc/bind/named.conf.local` folgendes am Ende ergänzen:

```
sudo cat << EOF >> /etc/bind/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";

zone "stormarn.freifunk.net" {
       type master;
       file "/etc/bind/db.net.freifunk.stormarn";
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  };

zone "freifunk-stormarn.de" {
       type master;
       file "/etc/bind/db.de.freifunk-stormarn";
  };

zone "lauenburg.freifunk.net" {
       type master;
       file "/etc/bind/db.net.freifunk.lauenburg";
  };

zone "freifunk-lauenburg.de" {
       type master;
       file "/etc/bind/db.de.freifunk-lauenburg";
  };

zone "freifunk-suedholstein.de" {
        type master;
        file "/etc/bind/db.de.freifunk-suedholstein";
};

zone "ffshev.de" {
     type master;
     file "/etc/bind/db.de.ffshev";
};
EOF
```

Die zugehörigen Zone Dateien werden in einem Repository verwaltet. Diese sollen automatisch aktualisiert werden.

Als erstes legen wir einen neuen Benutzer an.

```
sudo useradd -m -s /bin/bash dnsbind
```

Dann wechseln wir zu diesem Nutzer.

```
sudo su - dnsbind
```

Und klonen das Repository

```
git clone https://github.com/ffsh/bind.git
```

Danach verlassen wir den Nutzer.

```
exit
```

Und legen einige Cron jobs an.

```
sudo cat << EOF > /etc/cron.d/ffsh_dnsbind
*/15 * * * * root /home/dnsbind/bind/updatestofrei.sh > /dev/null 2>&1
*/15 * * * * root /home/dnsbind/bind/updatelauen.sh > /dev/null 2>&1
*/15 * * * * root /home/dnsbind/bind/updateffsh.sh > /dev/null 2>&1
EOF
```

Zum Schluss starten wir bind neu.

```
sudo systemctl restart bind9
```

2.9 Mesh Announce

Um als Gateway, Server oder alles was kein Freifunk Router ist auf der Karte zu erscheinen kann mesh-announce installiert werden.

Dafür müssen folgende Dinge vorhanden sein: lsb_release, ethtool, python3 (>= 3.3)

```
sudo apt install ethtool python3
```

Mesh Announce kann auch im alfred Stil Daten broadcasten das wollen wir aber nicht.

```
sudo git clone https://github.com/ffnord/mesh-announce /opt/mesh-announce
sudo cp /opt/mesh-announce/respondd.service /etc/systemd/system/respondd.service
nano /etc/systemd/system/respondd.service
```

Den Systemd Service passen wir jetzt an unser Netzwerk und Gateway an. Erstmal das Konzept. Wir starten respondd.py mit einigen argumenten:

```
respondd.py -d /opt/mesh-announce/providers -i <your-clientbridge-if> -i <your-mesh-
↪vpn-if> -b <your-batman-if> -m <mesh ipv4 address>
```

```
your-clientbridge-if - br-ffsh
your-mesh-vpn-if     - ffsh-mesh
your-batman-if       - bat0
mesh ipv4 address    - GW-IPV4
```

Im folgenden Beispiel ist Hopfenbach das Gateway dort sind die Interfaces so wie in der Anleitung benannt und die IP ist `10.144.128.1`.

```
[Unit]
Description=Respondd
After=network.target

[Service]
ExecStart=/opt/mesh-announce/respondd.py -d /opt/mesh-announce/providers -i br-ffsh -
↪i ffsh-mesh -b bat0 -m 10.144.128.1
Restart=always
Environment=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

[Install]
WantedBy=multi-user.target
```

Wir erzeugen die Datei `/etc/systemd/system/respondd.service`

```
sudo echo << EOF > /etc/systemd/system/respondd.service
[Unit]
Description=Respondd
After=network.target

[Service]
ExecStart=/opt/mesh-announce/respondd.py -d /opt/mesh-announce/providers -i br-ffsh -
↪i ffsh-mesh -b bat0 -m 10.144.[GW Netz].1
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
Restart=always
Environment=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

[Install]
WantedBy=multi-user.target
```

Und passen das GW-Netz an:

```
sudo nano /etc/systemd/system/respondd.service
```

Dann mit `hostname` prüfen ob der erwünschte Gateway-Name eingetragen ist. Wenn dem so ist kann der nächste Schritt übersprungen werden. Andernfalls (oder zur Sicherheit) hinterlegen wir es an passender Stelle:

```
sudo echo << EOF > /opt/mesh-announce/providers/nodeinfo/hostname.py
import providers
import socket
class Source(providers.DataSource):
    def call(self):
        return "[GW hostname]"
EOF
```

Und den gewünschten Hostname eintragen

```
sudo nano /opt/mesh-announce/providers/nodeinfo/hostname.py
```

Dann den Service aktivieren

```
sudo systemctl daemon-reload
sudo systemctl start respondd
sudo systemctl enable respondd
```

Das System sollte in kürze auf der Karte auftauchen, je nachdem wie der Kartenserver konfiguriert ist.

2.10 Munin

Damit das Gateway auch in den Statistiken unter http://stats.freifunk-suedholstein.de/ auftauch den Munin Node installieren

```
sudo apt install munin-node
```

und die :code: */etc/munin/munin-node.conf* anpassen.

```
cat << EOF > `/etc/munin/munin-node.conf
#
# Example config-file for munin-node
#

log_level 4
log_file /var/log/munin/munin-node.log
pid_file /var/run/munin/munin-node.pid

background 1
setsid 1

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
user root
group root

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

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

# 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$

# Set this if the client doesn't report the correct hostname when
# telnetting to localhost, port 4949
#
#host_name localhost.localdomain

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

allow ^127\.0\.0\.1$
allow ^::1$
allow ^176\.9\.83\.60$
allow ^159\.69\.191\.196$
allow ^2a01:4f8:1c17:44d1::1$


# If you have installed the Net::CIDR perl module, you can use one or more
# cidr_allow and cidr_deny address/mask patterns.  A connecting client must
# match any cidr_allow, and not match any cidr_deny.  Note that a netmask
# *must* be provided, even if it's /32
#
# Example:
#
# cidr_allow 127.0.0.1/32
# cidr_allow 192.0.2.0/24
# cidr_deny  192.0.2.42/32

# Which address to bind to;
# host *
# host 127.0.0.1
host 5.181.50.231
host 2a03:4000:3f:4db:7824:4eff:fe98:638

# And which port
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
port 4949
EOF
```

Node restarten

```
systemctl restart munin-node
```

2.11 Extras

2.11.1 VPN (Mullvad)

Wenn der Übergang in das Internet nicht auf diesem Knoten liegen soll kann man ein VPN nutzen. Im Folgenden ist die Konfiguration beschrieben. Die meisten Gateways nutzen das derzeit nicht.

Achtung: Kopiere bitte nicht die Konfigurationsdateien von einem Gateway auf andere Gateways!

Für das VPN werden diese Dateien benötigt, die alle nach `/etc/openvpn/` müssen:

```
ca.crt
crl.pem
mullvad.crt
mullvad.key
mullvad_linux.conf
```

Die Datei `mullvad\_linux.conf` muss noch um folgende Zeilen am Ende ergänzt werden:

```
#custom
route-noexec
up /etc/openvpn/mullvad_up.sh
up /etc/fastd/ffsh/iptables_ffsh.sh
```

Mullvad hat an seinen Konfigurationen seit mehreren Sicherheitslücken bei OpenVPN und Snowden/NSA geändert. Es kann sein, dass ein Fehler zur Cipher-Liste angezeigt wird. Dann muss in der `mullvad\_linux.conf` die Zeile zur TLS-Verschlüsselung beginnend tls-cipher auskommentiert werden. Wenn kein IPv6 am Server ins Internet möglich ist, kann auch tun-ipv6 auskommentiert werden.

Die Datei `/etc/openvpn/mullvad\_up.sh` gibt es noch nicht.Also bitte die Datei mit folgenden Zeilen anlegen:

```
#!/bin/sh
ip route replace 0.0.0.0/1 via $5 table 42
ip route replace 128.0.0.0/1 via $5 table 42

service dnsmaq restart
exit 0
```

Diese Datei muss nun auch als `root` ausführbar gemacht werden:

```
chmod +x /etc/openvpn/mullvad\_up.sh
```

Damit Linux auch diese VPN-Schnittstelle kennt, muss tun in der Datei `/etc/modules` bekannt gemacht werden. OpenVPN benötigt ein tun-Interface. Trage einfach in eine eigene neue Zeile dies ein

```
tun
```

Bitte nun als `root` über die Konsole tun aktivieren und den VPN starten mit:

```
modprobe tun
service openvpn start
```

2.11.2 VPN-Connect regelmäßig überprüfen

Es ist sinnvoll regelmäßig zu prüfen, ob die VPN Verbindung noch aktiv ist. Dazu wird ein Script auf dem Server abgelegt, dass dann über den CRON immer neu den VPN-Connect prüft.

`/ffsh/check-vpn.sh`

```
#!/bin/bash

# Test gateway is connected to VPN
test=$(ping -q -I tun0 8.8.8.8 -c 4 -i 1 -W 5 | grep 100 )

if [ "$test" != "" ]
    then
    echo "VPN nicht da - Neustart!"
    service openvpn restart      # Fehler - VPN nicht da - Neustart
else
    echo "alles gut"
fi
```

Dann noch das Script ausführbar machen:

```
chmod ug+x /ffsh/check-vpn.sh
```

Danach in die Datei `/etc/crontab` das Skript alle 10 Minute auszuführen und damit regelmäßig der VPN-Status geprüft wird.

```
# Check VPN via openvpn is running, if not service restart
*/10 * * * * root /ffsh/check-vpn.sh > /dev/null
```

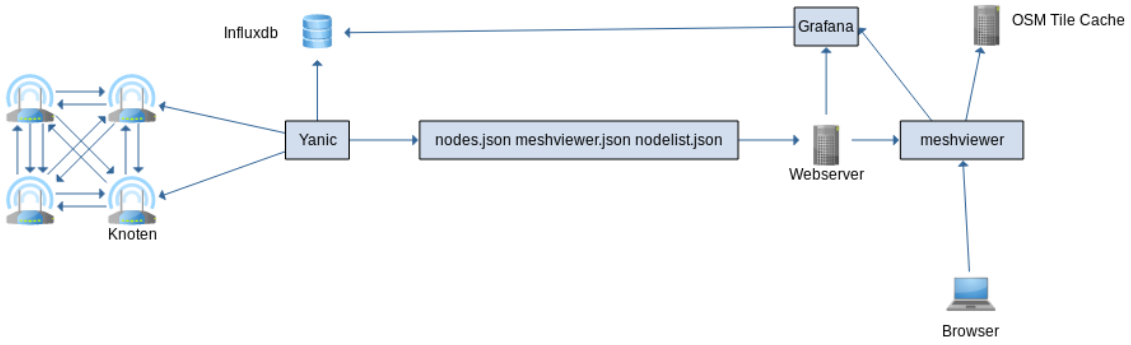
Die Änderungen übernehmen durch einen Neustart des Cron-Dämonen:

```
service cron restart
```

KAPITEL 3

map

Diese Dokumentation behandelt das Aufsetzen von https://map.freifunk-suedholstein.de



3.1 Vorraussetzungen

Wie für all unsere Service Server verwenden wir Ubuntu als Basis. Als Hardware verwenden für den gesammten Stack eine VM von Hetzner (CPX21) 2vCores 4GB RAM 40GB disk. Für unsere Community reicht dieses Setup bisher aus.

3.2 yanic

yanic sammelt von den Knoten Daten, welche dann auf einer Karte angezeigt werden können, früher wurde hierfür Alfred benutzt. yanic ist in go geschrieben also installieren wir eine neue Version von go. golang

```
wget https://dl.google.com/go/go1.17.5.linux-amd64.tar.gz
# Bitte sha256 vergleichen https://golang.org/dl/
tar -C /usr/local -xzf go1.17.5.linux-amd64.tar.gz
rm go1.17.5.linux-amd64.tar.gz
```

In `~/.bashrc`

```
GOPATH=/opt/go
PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
```

Hier musst du dich einmal abmelden und neu anmelden damit die Variablen auch gesetzt werden.

Nach dem Anmelden kann man prüfen ob die Variablen korrekt gesetzt wurden.

```
echo $GOPATH
/opt/go
```

Mit `whereis` go prüfen ob go gefunden wird:

```
go: /usr/local/go /usr/local/go/bin/go
```

Dann wird yanic installiert.

```
go get -v -u github.com/FreifunkBremen/yanic
```

Die Konfiguration von Yanic wird in `/etc/yanic.conf` angelegt:

```
[respondd]
enable           = true
synchronize      = "1m"
collect_interval = "1m"

[respondd.sites.ffsh]
domains            = ["ffsh", "ffod", "ffrz"]

[respondd.sites.ffod]
domains            = []

[respondd.sites.ffrz]
domains            = []

[[respondd.interfaces]]
ifname          = "bat0"

[[respondd.interfaces]]
ifname            = "bat0"
multicast_address = "ff02::2:1001"

[webserver]
enable  = false
bind    = "127.0.0.1:8080"
webroot = "/var/www/html/meshviewer"

[nodes]
state_path    = "/var/lib/yanic/state.json"
prune_after   = "14d"
save_interval = "2m"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
offline_after = "10m"

[[nodes.output.geojson]]
enable   = true
path = "/var/www/map/data/nodes.geojson"

[[nodes.output.meshviewer-ffrgb]]
enable   = true
path = "/var/www/map/data/meshviewer.json"

[nodes.output.meshviewer-ffrgb.filter]
no_owner = true
blacklist = ["f4f26d4a2ecc"]

[[nodes.output.meshviewer]]
enable = true
version    = 2
nodes_path = "/var/www/map/data/nodes.json"
graph_path = "/var/www/map/data/graph.json"

[nodes.output.meshviewer.filter]
no_owner = true
blacklist = ["f4f26d4a2ecc"]

[[nodes.output.nodelist]]
enable   = true
path = "/var/www/map/data/nodelist.json"

[nodes.output.nodelist.filter]
no_owner = true
blacklist = ["f4f26d4a2ecc"]

[database]
delete_after    = "30d"
delete_interval = "1h"

[[database.connection.influxdb]]
enabled  = true
address  = "http://localhost:8086"
database = "ffsh"
username = ""
password = ""

[database.connection.influxdb.tags]

[[database.connection.graphite]]
enable   = false
address  = "localhost:2003"
prefix   = "freifunk"

[[database.connection.respondd]]
enable   = false
type     = "udp6"
address  = "stats.bremen.freifunk.net:11001"

[[database.connection.logging]]
enable   = false
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
path      = "/var/log/yanic.log"
```

Wir können testen ob yanic funktioniert in dem wir eine manuelle Anfrage stellen hier an das Gateway Hopfenbach:

```
yanic query --wait 5 br-ffsh "fddf:bf7:80::48:1"
```

Damit yanic auch als Deamon läuft legen wir noch einen Service an.

```
sudo cp /opt/go/src/github.com/FreifunkBremen/yanic/contrib/init/linux-systemd/yanic.
↪service /lib/systemd/system/yanic.service
sudo systemctl daemon-reload
```

3.3 influxdb

Influxdb dient als Datenbank für `yanic`

Achtung hier wird Influxdb 1.x (aktuell 1.8.10) installiert, die aktuelle version ist 2.0 (diese wird aktuell nicht von yanic unterstützt).

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/lsb-release
echo "deb https://repos.influxdata.com/${DISTRIB_ID,,} ${DISTRIB_CODENAME} stable" |␣
↪sudo tee /etc/apt/sources.list.d/influxdb.list
```

```
sudo apt install influxdb influxdb-client
```

Nun sichern wir die influxdb ab `/etc/influxdb/influxdb.conf`

Hier werden nur die empfohlenen Anpassungen beschrieben: Noch vor der `[meta]` Sektion setzen wir, sonst wäre der port 8088 überall offen.

```
bind-address = "localhost:8088"
```

Weiter unten bei `[admin]` das gleiche:

```
bind-address = "localhost:8083"
```

kurz danach in `[http]`

```
bind-address = "localhost:8086"
```

```
systemctl restart influxdb
```

Nun sollte influxdb nur noch auf localhost erreichbar sein, prüfen kann man dies mit `netstat -tlpn`

3.4 Grafana

Grafana kann Graphen erstellen welche im meshviewer eingebunden werden können. Hier wird Grafana über eine Repository installiert.

```
deb https://packagecloud.io/grafana/stable/debian/ stable main
curl https://packagecloud.io/gpg.key | sudo apt-key add -
sudo apt-get update
sudo apt-get install grafana
```

Da Grafana bei uns hinter einem Proxy laufen soll, setzen wir auch hier alle IPs auf `localhost`. Am besten einmal am Ende prüfen ob alles richtig konfiguriert ist mit `netstat -tlpn`.

Ein wichtiger Punkt ist der öffentliche Zugang, damit die Statistiken auch von Besuchern abgerufen werden können.

```
#################################### Anonymous Auth ##########################
[auth.anonymous]
# enable anonymous access
enabled = true

# specify organization name that should be used for unauthenticated users
org_name = Freifunk Südholstein

# specify role for unauthenticated users
org_role = Viewer
```

Die Organisation kann man als Admin in Grafana anlegen.

3.5 meshviewer

Für den Meshviewer installieren wir als erstes nodejs und yarn

3.5.1 nodejs

Wir brauchen ein aktuelles nodejs das finden wir auf nodejs.org Wir benutzen die LTS Variante 16.x

```
curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash -
sudo apt-get install -y nodejs
```

3.5.2 yarn

Dann installieren wir yarn folge einfach den Anweisungen der Dokumentation.

3.5.3 meshviewer-rgb

Nun installieren wir den meshviewer selbst. Im web Verzeichnis `/var/www/`

```
git clone https://github.com/ffsh/meshviewer.git
cd meshviewer
yarn
```

Nun muss die Konfiguration in `meshviewer/config.js` eventuell noch angepasst werden.

Danach `yarn run gulp` Nun muss nur noch ein Webserver meshviewer/build ausliefern.

3.6 Tile-cache mit nginx

Für den Meshviewer benötigt man einen Tile-Server, der die Karte als einzelne Kacheln ausliefert. Wir verwenden dabei das kostenlose und freie Angebot von OpenStreetMap. Damit die Server von OpenStreetMap weniger stark belastet werden verwenden wir einen Tile-Cache. Bei einer Anfrage für eine Karten-Kachel fragt der Browser den Cache, hat dieser die Datei bereits, so liefert er sie direkt aus. Hat er sie nicht, so fragt er bei den OpenStreetMap Servern und speichert die Datei in seinem Cache. Für die einfache Umsetzung haben ein paar Freifunker an einer Konfiguration für nginx gearbeitet, welche genau das umsetzt.

Voraussetzungen: - nginx erreichbar unter der entsprechenden Domain - TLS mit gültigem Zertifikat (Let's Encrypt) - ein wenig Speicherplatz

```
#
# Nginx >= 1.9.15 - 1.10.1 recommended
#
# Thanks to https://github.com/cbricart

proxy_cache_path /var/www/tilecache/osm
    levels=1:2 inactive=7d
    keys_zone=tilecache:64m
    max_size=500M;

upstream osm_tiles {
    server a.tile.openstreetmap.org;
    server b.tile.openstreetmap.org;
    server c.tile.openstreetmap.org;
    keepalive 16;
}

server {
    listen 80;
    listen [::]:80;

    server_name tiles.freifunk-suedholstein.de;

    # Redirect all HTTP requests to HTTPS with a 301 Moved Permanently response.
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_session_tickets off;

    ssl_dhparam /etc/ssl/dhparam.pem;

    server_name tiles.freifunk-suedholstein.de;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-
↪ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-
↪SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-
↪SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
↪SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-
↪ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-
↪RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-
↪SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-
↪SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    ssl_prefer_server_ciphers on;

    add_header Strict-Transport-Security max-age=15768000;

    ssl_stapling on;
    ssl_stapling_verify on;

    ssl_certificate /etc/letsencrypt/live/tiles.freifunk-suedholstein.de/fullchain.
↪pem;
    ssl_certificate_key /etc/letsencrypt/live/tiles.freifunk-suedholstein.de/privkey.
↪pem;

    root /var/www/tilecache/html;

    location / {
        try_files $uri @osm;
    }

    location @osm {
        proxy_pass http://osm_tiles;
        proxy_http_version      1.1;
        proxy_set_header    Connection "";
        proxy_set_header    Accept-Encoding "";
        proxy_set_header    User-Agent "Mozilla/5.0 (compatible; OSMTileCache/1.0;␣
↪+mailto:noc@freifunk-suedhoslтein.de; +https://map.freifunk-suedholstein.de/)";
        proxy_set_header    Host tile.openstreetmap.org;

        proxy_temp_path     /var/www/tilecache/temp;
        proxy_cache         tilecache;
        proxy_store         off;
        proxy_cache_key     $uri$is_args$args;
        proxy_ignore_headers Expires Cache-Control;
        proxy_cache_valid   200 301 302 7d;
        proxy_cache_valid   404 1m;
        proxy_cache_valid   any 1m;
        proxy_next_upstream   error timeout invalid_header http_500 http_502 http_503␣
↪http_504 http_403 http_404;
        proxy_cache_use_stale error timeout updating invalid_header http_500 http_502␣
↪http_503 http_504 http_403 http_404;

        proxy_hide_header   Via;
        proxy_hide_header   X-Cache;
        proxy_hide_header   X-Cache-Lookup;

        expires 7d;
        }
}
```

3.7 Grafna cache mit nginx

Da Grafana ab Version 7.0 das rendern der images, welche wir auf der Karte einbetten, anders rendert als früher mussten wir einen Cache einrichten. Siehe https://github.com/ffrgb/meshviewer/issues/304

Basierend auf den Kommentaren haben wir auch eine Konfiguration zusammengestellt.

```
proxy_cache_path /var/cache/nginx/grafana
        keys_zone=grafana:10m
        max_size=128m;

server {
    listen 80;
    listen [::]:80;

    server_name grafana.freifunk-suedholstein.de;

    # Redirect all HTTP requests to HTTPS with a 301 Moved Permanently response.
    return 301 https://$host$request_uri;
}
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_session_tickets off;

    ssl_dhparam /etc/ssl/dhparam.pem;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-
↪ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-
↪SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-
↪SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
↪SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-
↪ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-
↪RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-
↪SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-
↪SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';
    ssl_prefer_server_ciphers on;

    add_header Strict-Transport-Security max-age=15768000;

    ssl_stapling on;
    ssl_stapling_verify on;

    ssl_certificate /etc/letsencrypt/live/grafana.freifunk-suedholstein.de/fullchain.
↪pem;
    ssl_certificate_key /etc/letsencrypt/live/grafana.freifunk-suedholstein.de/
↪privkey.pem;

    server_name grafana.freifunk-suedholstein.de;

    location / {
        proxy_pass http://127.0.0.1:3000;
    }

    location /render/ {
        proxy_http_version 1.1;
        proxy_pass http://127.0.0.1:3000/render/;

        proxy_cache grafana;
        proxy_cache_valid 300s;
        proxy_cache_lock on;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        proxy_cache_lock_age 60s;
        proxy_cache_lock_timeout 60s;

        proxy_hide_header Cache-Control;
        proxy_hide_header Expires;
        proxy_hide_header Pragma;

        proxy_ignore_headers Cache-Control Expires;

        expires 150s;
    }

}
```

KAPITEL 4

Firmware

Die Firmware von Freifunk Südholstein basiert auf Gluon, wir verwenden dabei in der Regel keine Community-Packages, dadurch ist unsere Firmware sehr stabil und wir können stets auf dem neusten Stand bleiben.

Das site Repository findet man auf GitHub. Freifunk Südholstein hat drei update Kanäle, die regelmäßig bedient werden.

- testing
- rc
- stable

Eine neue Firmware wird stets zuerst im testing Kanal verbreitet, wir haben nur wenige Knoten, welche den testing oder rc Kanal benutzen. Wir hoffen jedoch, dadurch zumindest grobe Probleme rechtzeitig zu entdecken.

4.1 Firmware Releases

Ein neuer Release beginnt in der Regel mit einem neuen Release von Gluon, dabei können Änderungen an der site Konfiguration nötig werden. Gelegentlich gibt es auch neue Features, ob diese per Update ausgerollt werden sollen wird dabei mit einer Folgen-Nutzen abschätzung Abgewegt. Sind die Risiken für Fehlfunktionen zu groß, so wird die Funktion zunächst nicht eingeführt.

Sobald die site config angepasst wurde, wird ein neuer git tag angelegt, die Version setzt sich dabei aus der Gluon-Version sowie einer extra Stelle für Änderungen von FFSH zusammen.

- Gluon Release = 2020.2.3
- FFSH Release = 0
- FFSH+Gluon Release 2020.2.3.0

Durch die extra Stelle kann FFSH beliebig viele Releases auf Basis derselben Gluon-Version erstellen. Nachdem ein Release erfolgreich gebaut wurde, wird der Release signiert und per Webserver an die Knoten verteilt.

Dabei gilt die Reihenfolge testing, rc, stable. Nachdem die Knoten eines Kanals aktuallisiert wurden wird in der Regel etwa eine Woche gewartet bis die Firmware auf den folgenden Kanal ausgerollt wird. Aufgrund der geringen Anzahl an Knoten werden testing und rc gelegentlich parallel ausgerollt.

4.2 Firmware Pipeline

Für die Firmware Pipeline benutzt Freifunk Südholstein GitHub Actions, sowie abgewandelte Skripte vom Gluon Projekt.

Im site Repository gibt es ein actions Verzeichnis, dort liegen ein paar scripte.

- `generate-actions.py` ist ein leicht abgewandeltes script vom Gluon Projekt, welches den GitHub Workflow generiert.
- `install-dependencies.sh` installiert zusätzliche Pakete in dem Cointainer von GitHub.
- `run-build-local.sh` kann zum bauen auf der lokalen maschine verwendet werden, je nach Bedaf anpassen, für die Fehlersuche gedacht.
- `run-build.sh` ist das eigentliche build Skript, bei einem neuen release wird hier die Version angepasst.

Die vom `generate-actions.py` generierte `build-gluon.yml` Datei liegt in `.github/workflows`, das Skript muss in der Regel nur dann neu ausgeführt werden, wenn ein neuer Gluon Release ein *target* hinzufügt oder entfernt. Das Script muss angepasst werden, wenn es einen neuen major Release gibt z.B. 2020->2021, dann muss der tags liste ein neuer regex Ausdruck hinzugefügt werden.

Experimentell ist bisher der Workflow im `release.yml`, das Ziel ist es nach dem erfolgreichen Bauen der Firmware manuell einen Release auf GitHub anzulegen. Der Workflow soll dann über API ein Firmware Release-Archiv zusammenstellen und als Anhang an den Release beifügen.

Das Bauen der Firmware dauert aktuell etwas mehr als eine Stunde, wenn GitHub durchschnittlich ausgelastet ist. Sollte GitHub sich eines Tages dazu entscheiden, die Ressourcen zu verringern, gibt es die Möglichkeit einen self-hosted runner z.b. in der Hetzner Cloud zu installieren.

4.2.1 Lokal Bauen

Die Firmware kann auch lokal gebaut werden, dies ist vor allem nützlich, wenn man einen speziellen release erstellen möchte, oder zur Fehlersuche.

Dafür können natürlich ganz klassisch alle Abhängigkeiten installiert werden, Infos gibt es in der Gluon Dokumentation. Allerdings kann es dabei leicht zu Problemen kommen, alternativ liegt im site Repository ein Dockerfile.

```
docker build . --tag gluon
docker run --name gluon --mount type=bind,source=$(pwd),target=/gluon gluon
```

4.3 Site Repository

Das site Repository wird auf GitHub gehostet. Es gibt einen main branch, auf diesem Branch findet die hauptsächliche Entwicklung statt. Daneben kann es feature branches geben, die ausgehend vom main branch eine experimentelle oder abweichende Funktionalität mitbringen.

Es gibt kein Changelog im Repository, auf freifunk-suedholstein.de sind alle Beiträge zur Firmware getagt und man kann einen schnellen Überblick über die Veränderungen bekommen.

KAPITEL 5

Indices and tables

- genindex
- modindex
- search