

Paket VPN

Version 4.0.0-trunk-x86-r60803

Frank Meyer Das fli4l-Team
E-Mail: frank@fli4l.de E-Mail: team@fli4l.de

28. Dezember 2022

Inhaltsverzeichnis

1	Dokumentation des Paketes VPN	3
1.1	VPN - Unterstützung virtueller privater Netzwerke	3
1.1.1	PPTP-Tunnel	3
1.1.2	fastd-Tunnel	9
1.1.3	WireGuard	10
	Abbildungsverzeichnis	26
	Tabellenverzeichnis	27
	Index	28

1 Dokumentation des Paketes VPN

1.1 VPN - Unterstützung virtueller privater Netzwerke

Dieses Paket erlaubt es, gesicherte Verbindungen zwischen privaten Netzwerken über öffentliche, aber unsichere Netzwerke aufzubauen.

1.1.1 PPTP-Tunnel

PPTP¹ bietet eine Möglichkeit, einen privaten Kanal über ein öffentliches Netzwerk aufzubauen. Dabei wird der Tunnelaufbau und -abbau über ein spezielles TCP/IP-Kontrollprotokoll gesteuert. Die eigentlichen Nutzdaten werden in PPP-Paketen verpackt, die über einen GRE-Tunnel² geschickt werden.

In Österreich (und anderen europäischen Ländern) wird PPTP zusätzlich als Protokoll zwischen Router und DSL-Modem verwendet. Im Gegensatz zu PPPoE und PPPoA, die beide noch unterhalb der IP-Ebene arbeiten (Sicherheitsschicht, "Link Layer"), gibt es bei PPTP wie oben beschrieben zwei Datenströme. Somit benötigt man für DSL über PPTP im Gegensatz zu anderen DSL-Zugangsmethoden auch für die für PPTP reservierte Ethernet-Karte eine IP-Adresse. Die ist je nach Provider entweder fest vorgegeben oder muss via DHCPv4 konfiguriert werden. Mehr dazu steht in der Beschreibung der Variable `CIRC_x_PPP_PPTP_PEER`.

Ist man auf Grund eines DSL-Anschlusses oder einer veralteten VPN-Gegenstelle nicht gezwungen, PPTP zu nutzen, so wird davon abgeraten, PPTP als VPN-Lösung zu verwenden. Die Verschlüsselung in PPTP gilt als geknackt³, so dass man über PPTP-Tunnel nicht sicherheitskritische Daten verschicken sollte. Für einen solchen Einsatzzweck bilden Tunnel auf der Basis von OpenVPN sicherlich die bessere Wahl.

Ausgehende PPTP-Verbindungen

Generell werden ausgehende PPTP-Verbindungen als PPP-Circuits konfiguriert (siehe Circuits vom Typ "ppp" (Seite ??)), d. h. es gilt:

```
CIRC_x_TYPE='ppp'
```

Zusätzlich muss das `OPT_PPP_PPTP` aktiviert werden:

OPT_PPP_PPTP Diese Variable aktiviert die Unterstützung für PPTP. Damit auch tatsächlich eine PPTP-Verbindung genutzt werden kann, muss mindestens ein PPP-Circuit den Typ "pptp" besitzen, d. h. es muss zusätzlich gelten

¹"Point-to-Point Tunneling Protocol", siehe RFC 2637

²"Generic Routing Encapsulation", ein Protokoll, welches beliebige andere Protokolle kapseln und über IP-Netzwerke transportieren kann, siehe RFC 2784

³siehe <http://heise.de/-1701365>

```
CIRC_x_TYPE='ppp'
CIRC_x_PPP_TYPE='pptp'
```

(wobei “x” einen gültigen Circuit-Index darstellt).

Standard-Einstellung: OPT_PPP_PPTP='no '

Beispiel: OPT_PPP_PPTP='yes '

CIRC_x_PPP_PPTP_TYPE Diese Variable definiert, wer die PPP-Datenpakete in GRE-Paketen kapselt und über die Leitung schickt. Dies kann durch ein Benutzerprogramm (**pptp**) oder durch den Kern erfolgen. Mittels **CIRC_x_PPP_PPTP_TYPE** wird die Art und Weise der GRE-Paketerzeugung definiert, siehe hierzu Tabelle 1.1.

Wert	Beschreibung
kernel	Die PPP-Pakete werden direkt an den Linux-Kern gereicht, der daraus GRE-Pakete macht. Dadurch entfällt die Kommunikation mit einem zweiten Prozess und damit eine Menge Kopier- und Scheduling-Aufwand, was wiederum zu geringerer Prozessorlast führt.
daemon	Die Pakete werden durch den pptp -Dämon erzeugt; die Kommunikation zwischen pppd und pptp erfolgt asynchron. Das bedeutet, dass der Datenstrom mit Anfang- und Ende-Markern versehen wird, damit der pptp -Dämon die einzelnen Pakete auseinanderhalten kann. Aufgrund des zweiten Prozesses und der zusätzlichen Markierungen ist diese Methode aufwändiger als die Methode “kernel”.

Tabelle 1.1: Arten der GRE-Paketerzeugung

Momentan ist “daemon” die einzige unterstützte Methode (und somit auch Standard, falls der Typ nicht angegeben wird). Eine Erweiterung des Pakets um das Nutzen des entsprechenden **pptp**-Kernel-Moduls steht noch aus.

Standard-Einstellung: **CIRC_x_PPP_PPTP_TYPE**='daemon '

CIRC_x_PPP_PPTP_PEER Hier wird die IP-Adresse der PPTP-Gegenstelle eingetragen.

Nutzt man PPTP für einen DSL-Internet-Zugang, muss dazu passend die IP-Adresse der fli4l-PPTP-Ethernet-Karte gewählt werden. In Tabelle 1.2 sind bekannte Konfigurationsvarianten aufgelistet.

Provider	lokale IP-Adresse (IP_NET_2)	entfernte IP-Adresse (CIRC_x_PPP_PPTP_PEER)
Telekom Austria (Österreich)	10.0.0.140/29	10.0.0.138
mxstream (Niederlande, Dänemark)	10.0.0.140/29	10.0.0.138
Inode xDSL (Österreich)	via DHCPv4	10.0.0.138

Tabelle 1.2: Einstellungen für Provider, die PPTP nutzen

Für den Fall, dass DHCP zur Konfiguration der lokalen Netzwerk-Karte benötigt wird, ist das **dhcpcd**-Paket zu installieren und ein entsprechender DHCPv4-Circuit für die

jeweilige Ethernet-Karte (in der Regel eth1) einzurichten. Ein Beispiel für Inode xDSL findet sich am Ende des Abschnitts.

CIRC_x_PPP_PPTP_REORDER_TIMEOUT Der PPTP-Client muss unter Umständen Pakete zwischenspuffern und umordnen. Normalerweise wartet er 0,3 Sekunden auf ein ausstehendes Paket. Mit dieser Variable kann man den Timeout zwischen 0.00 (gar nicht puffern) und 10.00 (max. 10 Sekunden warten) variieren. Die Zeiten müssen immer mit Punkt und zwei Nachkommstellen angegeben werden.

Standard-Einstellung: CIRC_x_PPP_PPTP_REORDER_TIMEOUT='0.30'

Beispiel: CIRC_1_PPP_PPTP_REORDER_TIMEOUT='1.00'

CIRC_x_PPP_PPTP_LOGLEVEL Mit dieser Variable kann konfiguriert werden, wieviel Ausgaben der PPTP-Client produziert. Möglich sind 0 (wenig), 1 (mittel) und 2 (viel).

Standard-Einstellung: CIRC_x_PPP_PPTP_LOGLEVEL='1'

Beispiel: CIRC_1_PPP_PPTP_LOGLEVEL='2'

Eingehende PPTP-Verbindungen

Der fli4l kann auch konfiguriert werden, *eingehende* PPTP-Verbindungen anzunehmen, also als ein Server zu fungieren. Solche PPTP-Verbindungen werden ebenfalls als PPP-Circuit konfiguriert (siehe Circuits vom Typ “ppp” (Seite ??)), d. h. es gilt:

```
CIRC_x_TYPE='ppp'
```

Zusätzlich muss das OPT_PPP_PPTP_SERVER aktiviert werden:

OPT_PPP_PPTP_SERVER Mit dieser Variable wird die Unterstützung für eingehende PPTP-Verbindungen aktiviert. Damit auch tatsächlich PPTP-Verbindungen angenommen werden können, muss mindestens ein PPP-Circuit den Typ “pptp-server” besitzen, d. h. es muss zusätzlich gelten

```
CIRC_x_TYPE='ppp'  
CIRC_x_PPP_TYPE='pptp-server'
```

(wobei “x” einen gültigen Circuit-Index darstellt).

Standard-Einstellung: OPT_PPP_PPTP_SERVER='no'

Beispiel: OPT_PPP_PPTP_SERVER='yes'

Zu den allgemeinen Circuit-Variablen kommen die folgenden, für PPP-Circuits des Typs “pptp-server” spezifischen Variablen hinzu:

CIRC_x_PPP_PPTP_SERVER_LISTEN Mit dieser Variable kann die IPv4-Adresse festgelegt werden, an welcher der PPTP-Server horcht. Wird diese Variable weggelassen, horcht der PPTP-Server an *allen* Schnittstellen des Routers.⁴

⁴Dies entspricht der IPv4-Adresse 0.0.0.0.

Mit Hilfe der hier konfigurierten Adresse wird im Falle von `PF_INPUT_ACCEPT_DEF='yes'` (Seite ??) bzw. `PF_OUTPUT_ACCEPT_DEF='yes'` (Seite ??) die Firewall in den INPUT- und OUTPUT-Ketten sowohl für das PPTP-Kontrollprotokoll auf TCP-Port 1723 als auch für die GRE-Pakete geöffnet. Fehlt diese Variable, wird die Firewall so konfiguriert, dass die Kontroll- und Daten-Pakete an *jeder* Adresse des PPTP-Servers empfangen werden können.

Beispiel: `CIRC_1_PPP_PPTP_SERVER_LISTEN='IP_NET_1_ADDR'`

CIRC_x_PPP_PPTP_SERVER_ALLOW_FROM_y Dieses Array enthält eine Liste von IPv4-Netzadressen, für die ein Zugriff auf den PPTP-Server in der Firewall erlaubt wird. Mit Hilfe der hier konfigurierten Adressen wird im Falle von `PF_INPUT_ACCEPT_DEF='yes'` (Seite ??) bzw. `PF_OUTPUT_ACCEPT_DEF='yes'` (Seite ??) die Firewall in den INPUT- und OUTPUT-Ketten sowohl für das PPTP-Kontrollprotokoll auf TCP-Port 1723 als auch für die GRE-Pakete geöffnet. Fehlt dieses Array, wird die Firewall so konfiguriert, dass die Kontroll- und Daten-Pakete von bzw. nach *überall* akzeptiert werden.

Beispiel:

```
CIRC_1_PPP_PPTP_SERVER_ALLOW_FROM_N='3'
CIRC_1_PPP_PPTP_SERVER_ALLOW_FROM_1='IP_NET_1'
CIRC_1_PPP_PPTP_SERVER_ALLOW_FROM_2='10.1.2.0/24'
CIRC_1_PPP_PPTP_SERVER_ALLOW_FROM_3='{Labor}'
```

CIRC_x_PPP_PPTP_SERVER_SESSIONS Diese Variable enthält die Anzahl der Verbindungen, die dieser PPTP-Server maximal gleichzeitig verwalten kann. Maximal werden 255 Tunnel unterstützt. ⁵

Standard-Einstellung: `CIRC_x_PPP_PPTP_SERVER_SESSIONS='100'`

Beispiel: `CIRC_1_PPP_PPTP_SERVER_SESSIONS='200'`

Beispiele

Beispiel 1 (Internet-Zugang über PPTP mit fester lokaler Adresse):

```
IP_NET_N='2'                # (mindestens) zwei Netze (LAN + PPTP)
IP_NET_1='192.168.6.0/24'    # lokales Netz, wie benötigt konfigurieren
IP_NET_1_DEV='eth0'          # lokales Netz hängt an erster Karte
IP_NET_2='10.0.0.140/29'     # unsere Adresse im PPTP-Netz
IP_NET_2_DEV='eth1'          # Internet-Modem hängt an zweiter Karte
#
OPT_PPP='yes'                # PPP-Circuits aktivieren
OPT_PPP_PPTP='yes'           # PPTP-Client-Circuits aktivieren
#
CIRC_N='1'
CIRC_1_NAME='DSL-mxstream'    # beliebig, aber eindeutig
CIRC_1_TYPE='ppp'             # das ist ein PPP-Circuit
```

⁵Diese Beschränkung resultiert daher, dass der verwendete PPTP-Dämon einen Adressbereich nur in einer Komponente einer IPv4-Adresse erlaubt, also z.B. "192.168.222.0-254". Da eine Komponente nur die Werte 0-255 annehmen kann und der Wert 255 für die Broadcast-Adresse reserviert sind, resultiert daraus die genannte Beschränkung.

```

CIRC_1_ENABLED='yes'
CIRC_1_NETS_IPV4_N='1'
CIRC_1_NETS_IPV4_1='0.0.0.0/0'      # Default-Route ins Internet
CIRC_1_CLASS_N='1'
CIRC_1_CLASS_1='internet'          # Klasse für Internet-Anbindung
CIRC_1_UP='yes'                     # beim Booten aktivieren
CIRC_1_TIMES='Mo-Su:00-24:0.0:Y'
CIRC_1_USEPEERDNS='yes'             # DNS-Server des Providers nutzen
CIRC_1_PPP_TYPE='pptp'              # PPTP-Client
CIRC_1_PPP_USERID='anonymer'        # Benutzername zur Authentifizierung
CIRC_1_PPP_PASSWORD='surfer'        # Passwort zur Authentifizierung
CIRC_1_PPP_PPTP_PEER='10.0.0.138'   # Adresse des Internet-Modems im PPTP-Netz
#
CIRC_CLASS_N='1'
CIRC_CLASS_1='internet'             # Klasse aller Internet-Circuits

```

Beispiel 2 (Internet-Zugang über PPTP mit dynamisch zugewiesener lokaler Adresse):

```

IP_NET_N='2'                        # (mindestens) zwei Netze (LAN + PPTP)
IP_NET_1='192.168.6.0/24'           # lokales Netz, wie benötigt konfigurieren
IP_NET_1_DEV='eth0'                 # lokales Netz hängt an erster Karte
IP_NET_2='{DHCP-Inode}'             # PPTP-Netz, via DHCP konfiguriert
IP_NET_2_DEV='eth1'                 # Internet-Modem hängt an zweiter Karte
#
OPT_DHCP_CLIENT='yes'               # DHCP-Circuits aktivieren
OPT_PPP='yes'                       # PPP-Client-Circuits aktivieren
OPT_PPP_PPTP='yes'                 # PPTP-Client-Circuits aktivieren
#
CIRC_N='2'                          # zwei Circuits: DHCP und PPTP
#
CIRC_1_NAME='DHCP-Inode'            # beliebig, aber eindeutig
CIRC_1_TYPE='dhcp'                 # das ist ein DHCP-Circuit
CIRC_1_ENABLED='yes'
CIRC_1_NETS_IPV4_N='1'              # hierüber soll die PPTP-Gegenstelle
CIRC_1_NETS_IPV4_1='10.0.0.138/32' # (= Internet-Modem) erreichbar sein
CIRC_1_DHCP_DEV='IP_NET_2_DEV'      # die PPTP-Ethernet-Karte
CIRC_1_UP='yes'                     # beim Booten aktivieren
#
CIRC_2_NAME='PPTP-Inode'            # beliebig, aber eindeutig
CIRC_2_TYPE='ppp'                  # das ist ein PPP-Circuit
CIRC_2_ENABLED='yes'
CIRC_2_PPP_TYPE='pptp'             # PPTP-Client
CIRC_2_PPP_USER='anonymer'         # Benutzername zur Authentifizierung
CIRC_2_PPP_PASS='surfer'           # Passwort zur Authentifizierung
CIRC_2_PPP_FILTER='yes'            # Datenverkehr-Filter aktivieren
CIRC_2_PPP_PPTP_PEER='10.0.0.138'  # Adresse des Internet-Modems im PPTP-Netz
CIRC_2_NETS_IPV4_N='1'
CIRC_2_NETS_IPV4_1='0.0.0.0/0'     # Default-Route ins Internet
CIRC_2_USEPEERDNS='yes'            # DNS-Server des Providers nutzen
CIRC_2_HUP_TIMEOUT='600'           # nach 10 Minuten Inaktivität auflegen
CIRC_2_UP='yes'                     # beim Booten aktivieren
CIRC_2_DEPS='DHCP-Inode'           # PPTP benötigt DHCP-Konfiguration

```

Beispiel 3 (VPN-Client):

1 Dokumentation des Paketes VPN

```
IP_NET_N='1' # (mindestens) ein (lokales) Netz
IP_NET_1='192.168.6.0/24' # lokales Netz, wie benötigt konfigurieren
IP_NET_1_DEV='eth0' # lokales Netz hängt an erster Karte
#
OPT_PPP='yes' # PPP-Circuits aktivieren
OPT_PPP_ETHERNET='yes' # PPPoE-Client-Circuits aktivieren (DSL)
OPT_PPP_PPTP='yes' # PPTP-Client-Circuits aktivieren (VPN)
#
CIRC_N='2' # zwei Circuits: PPPoE (Internet) und PPTP
#
CIRC_1_NAME='DSL-Telekom' # beliebig, aber eindeutig
CIRC_1_TYPE='ppp' # das ist ein PPP-Circuit
CIRC_1_ENABLED='yes'
CIRC_1_NETS_IPV4_N='1'
CIRC_1_NETS_IPV4_1='0.0.0.0/0' # Default-Route ins Internet
CIRC_1_CLASS_N='1'
CIRC_1_CLASS_1='internet' # Klasse für Internet-Anbindung
CIRC_1_UP='yes' # beim Booten aktivieren
CIRC_1_TIMES='Mo-Su:00-24:0.0:Y'
CIRC_1_USEPEERDNS='yes' # DNS-Server des Providers nutzen
CIRC_1_PPP_TYPE='ethernet' # PPPoE-Client
CIRC_1_PPP_USERID='anonymer' # Benutzername zur Authentifizierung
CIRC_1_PPP_PASSWORD='surfer' # Passwort zur Authentifizierung
CIRC_1_PPP_ETHERNET_TYPE='kernel' # Kernel soll PPPoE-Pakete packen
CIRC_1_PPP_ETHERNET_DEV='eth1' # DSL-Modem hängt an zweiter Karte
#
CIRC_2_NAME='VPN-Firma' # beliebig, aber eindeutig
CIRC_2_TYPE='ppp' # das ist ein PPP-Circuit
CIRC_2_ENABLED='yes'
CIRC_2_NETS_IPV4_N='1'
CIRC_2_NETS_IPV4_1='10.11.12.0/24' # Firmennetz
CIRC_2_DEPS='internet/ipv4' # Verbindung zur Firma benötigt
# IPv4-Internet
CIRC_2_UP='yes' # beim Booten aktivieren
CIRC_2_TIMES='Mo-Su:00-24:0.0:Y'
CIRC_2_PPP_TYPE='pptp' # PPTP-Client
CIRC_2_PPP_USERID='mustermann' # Benutzername zur Authentifizierung
CIRC_2_PPP_PASSWORD='geheim' # Passwort zur Authentifizierung
CIRC_2_PPP_PPTP_PEER='192.0.2.1' # Adresse des PPTP-Servers der Firma
#
CIRC_CLASS_N='1'
CIRC_CLASS_1='internet' # Klasse aller Internet-Circuits
```

Beispiel 4 (VPN-Server):

```
OPT_PPP='yes' # PPP-Circuits aktivieren
OPT_PPP_PPTP_SERVER='yes' # PPTP-Server-Circuits aktivieren
OPT_PPP_PEERS='yes' # zum Speichern der Anmeldedaten
PPP_PEER_N='1' # 1x Anmeldedaten hinterlegen
PPP_PEER_1_USERID='user' # Benutzername vom Client
PPP_PEER_1_PASSWORD='pass' # Passwort vom Client
PPP_PEER_1_CIRCUITS='pptp-eth1' # Anmeldedaten gelten für PPTP-Circuit
#
```



```
CIRC_N='1'
CIRC_1_NAME='pptp-eth1'           # beliebig, aber eindeutig
CIRC_1_TYPE='ppp'                 # das ist ein PPP-Circuit
CIRC_1_ENABLED='yes'
CIRC_1_UP='yes'                   # beim Booten aktivieren
CIRC_1_TIMES='Mo-Su:00-24:0.0:Y'
CIRC_1_PROTOCOLS='ipv4'          # IPv4 soll über die Verbindung laufen
CIRC_1_PPP_TYPE='pptp-server'     # PPTP-Server
CIRC_1_PPP_PEER_AUTH='yes'        # Client-Authentifizierung ist Pflicht
CIRC_1_PPP_COMP_MPPE='yes'        # benutze Verschlüsselung
CIRC_1_PPP_LOCALIP='192.168.222.1' # IP-Adresse des Servers
CIRC_1_PPP_REMOTEIP='192.168.222.2' # Start-IP-Adresse der Clients
CIRC_1_PPP_PPTP_SERVER_SESSIONS='10' # max. 10 Tunnel
```

1.1.2 fastd-Tunnel

Mit fastd⁶ steht eine schnelle und kleine Alternative zu bekannten VPN-Lösungen bereit. Ziele bei der Entwicklung waren folgende Punkte:

- sehr kleine Binärdatei (um unter OpenWRT auf Geräten mit sehr wenig Speicher eingesetzt werden zu können)
- austauschbare Krypto-Methoden
- Nutzung von UDP, um auch hinter NAT eingesetzt werden zu können
- Unterstützung von 1:1 und 1:n Szenarios

In vielen Freifunk-Projekten ist fastd Bestandteil der auf Gluon basierenden Firmware. Während dort in der Regel „klar“ ist, was Client und Server ist, wird dies nicht durch das verwendete Protokoll bestimmt, sondern nur durch die Nutzung. Für fastd werden daher neben der Serverinstanz bei jedem beteiligten Kommunikationspartner ein oder mehrere sogenannte Peers definiert. Für die Konfiguration der jeweiligen Gegenstelle kehren sich damit die Rollen um: was auf der einen Seite Serverdaemon ist, muss auf der anderen Seite als Peer konfiguriert werden und umgekehrt.

Generelle Einstellungen

OPT_FASTD Diese Variable aktiviert den fastd. Um tatsächlich Verbindungen aufzubauen, müssen diese natürlich noch entsprechend konfiguriert werden.

Standard-Einstellung: OPT_FASTD='no'

Beispiel: OPT_FASTD='yes'

FASTD_SECRET Setzt den geheimen Schlüssel dieser fastd-Instanz. Ein Schlüsselpaar kann beispielsweise mit dem Programm fastd erzeugt werden, welches auf dem Router oder auf einem beliebigen anderen Linux-Rechner laufen kann:

⁶Fast and Secure Tunnelling Daemon

```
# fastd --generate-key
2016-01-06 18:54:13 +0100 --- Info: Reading 32 bytes from /dev/random...
Secret: d8dec7f65c89a39561ecde12623e8051d1d7c4286253b119f155e3fe169cd161
Public: e89e34e53c35bd6d750558a5e747fa72f25fbc922c86625b705ef1aa1865e32a
```

Wird diese Variable leer gelassen, erzeugt das Startscript ein Schlüsselpaar und legt den Inhalt in der Datei `/tmp/fastd.secret` ab, die man sich dann vom Router runter kopieren kann. Der fastd Service wird dann nicht gestartet!

Standard-Einstellung: `FASTD_SECRET=''`

Beispiel: `FASTD_SECRET='d8dec7f65c89a39561ecde12623e8051d1d7c4286253b119f155e3fe169cd161'`

1.1.3 WireGuard

WireGuard⁷ ist eine recht neue Alternative zu bekannten VPN-Lösungen wie OpenVPN oder IPSec. Ziel ist ein modernes, schnelles, einfaches VPN, das state-of-the-Art Verschlüsselung einsetzt. IPv4, IPv6 und auch Dual-Stack sind mit WireGuard kein Problem.

Ziele bei der Entwicklung waren unter anderem

- Einfachheit
- Geschwindigkeit
- state-of-the-art Kryptographie
- schneller, einfacher, schlanker als IPsec
- performanter als OpenVPN
- genauso einfach zu konfigurieren wie SSH

Diese Einfachheit zeigt sich auch in der Konfiguration unter `fl4l`.

Generelle Einstellungen

OPT_WIREGUARD Standard-Einstellung: `OPT_WIREGUARD='no'`

Diese Variable aktiviert das Paket WireGuard. Zusätzlich müssen die einzelnen Verbindungen oder Clients (peers) konfiguriert werden.

Beispiel: `OPT_WIREGUARD='yes'`

WireGuard Server

WireGuard VPNs sind kein Client-Server-Modell im eigentlichen Sinne. Vielmehr werden VPN-Verbindungen immer zwischen zwei Peers aufgebaut. Dies bringt den Vorteil mit sich, dass die IP-Adressen beider Peers floating sein können. Beide Peers werden Änderungen der IP-Adresse automatisch übernehmen ohne weitere Konfiguration. Entscheidend ist nur, dass die ankommenden UDP-Pakete richtig signiert und verschlüsselt sind (siehe Private/Public Key unten). Weder im `fl4l` noch bei den Peers ist hierfür spezielle Konfiguration erforderlich.

⁷<https://www.wireguard.com/>

Der Einfachheit halber sprechen wir im Weiteren teils dennoch von Server (damit ist der fli4l gemeint) und Client (damit ist dann die Gegenstelle wie z.B. das Handy, der PC oder auch ein anderer fli4l gemeint)

Achtung: Zusätzlich zur Konfiguration von WireGuard in `vpn.txt` muss der Paketfilter entsprechend konfiguriert werden. Dies geschieht über die normalen Wege in `base.txt`. Wichtig sind v. a.:

Port öffnen In der Standardeinstellung werden die WireGuard Ports automatisch im Paketfilter geöffnet. Sollte dies nicht gewünscht sein, kann `WIREGUARD_DEFAULT_OPEN_PORT='no'` gesetzt werden. In diesem Fall müssen die Ports dann manuell geöffnet werden z. B.

```
PF_INPUT[]='prot:udp 50002 ACCEPT' # in base.txt
```

Paketfilter-Regeln entsprechende FORWARD- und MASQ-Regeln im Paketfilter

Erforderliche Parameter Server

WIREGUARD_N Legt fest wie viele WireGuard Konfigurationen ('Server') wir haben wollen.

Beispiel: `WIREGUARD_N='1'`

WIREGUARD_x_NAME Der Name dieser WireGuard-Instanz

WIREGUARD_x_LOCAL_IP4 Lokale IP-Adresse (bei \32) oder Netzwerk (z. B. \24) dieser WireGuard-Instanz

WIREGUARD_x_LISTEN_PORT Port, auf dem WireGuard lauschen soll. Der Port muss zusätzlich in `base.txt` geöffnet werden, z. B.:

```
WIREGUARD_1_LISTEN_PORT='50002' # in vpn.txt
PF_INPUT[]='prot:udp 50002 ACCEPT' # in base.txt
```

WIREGUARD_x_PRIVATE_KEY Standardwert: `WIREGUARD_x_PRIVATE_KEY='auto'`

Der private Schlüssel dieser WireGuard-Instanz.

Es gibt zwei Möglichkeiten:

auto Private und zugehöriger Public Key werden automatisch erstellt

some key ein gültiger WireGuard Private Key

Der Key kann manuell auf dem fli4l per

```
wg genkey > privatekey
```

erstellt werden. Einfacher ist es aber sicherlich, beim ersten Start hier 'auto' zu konfigurieren und den erstellten Private Key dann nach Start aus dem Webinterface des fli4l zu übernehmen. Anders als die Private Keys der Peers muss der Private Key des Servers auf dem fli4l verbleiben.

Optionale Parameter Server

WIREGUARD_DEFAULT_OPEN_PORT Standardwert: `WIREGUARD_DEFAULT_OPEN_PORT='yes'`

Optional: Legt fest ob im Paketfilter automatisch Regeln angelegt werden sollen, um eingehende WireGuard-Pakete zu akzeptieren. Damit werden alle in `WIREGUARD_x_LISTEN_PORT` definierten UDP-Ports automatisch geöffnet.

WIREGUARD_x_DEFAULT_ALLOWED_IPS_N Optional: Anzahl der IPs bzw. Netzwerke, die den Peers vorgegeben werden sollen, diese durch den VPN-Tunnel zu routen.

Standardwert: `WIREGUARD_x_DEFAULT_ALLOWED_IPS_N='0'`

WIREGUARD_x_DEFAULT_ALLOWED_IPS Optional: IPs bzw. Netzwerke, die für alle Peers durch den VPN-Tunnel geroutet werden sollen. Hier angegebene Netze werden in der Peer-Config, die im Webinterface als QR-Code oder Datei einsehbar ist, entsprechend übernommen. Bei manueller Konfiguration der Peers wären diese Werte in `AllowedIPs` anzugeben.

Es können sowohl IPv4- als auch IPv6-Netzwerke angegeben werden. Interne Identifier wie `IP_NET_x` und `IP6_NET_x` sind ebenfalls möglich.

Beispiel:

```
WIREGUARD_1_DEFAULT_ALLOWED_IPS_N='2'
WIREGUARD_1_DEFAULT_ALLOWED_IPS_1='IP_NET_1'
WIREGUARD_1_DEFAULT_ALLOWED_IPS_2='10.0.0.2/24'
```

WIREGUARD_x_KEEP_ALIVE Standard-Einstellung: `WIREGUARD_x_KEEP_ALIVE='0'`

Optional: Intervall in Sekunden, in dem WireGuard UDP-Pakete versendet, um die Verbindung aufrecht zu erhalten. Da UDP an sich zustandslos ist, ist dies im Normalfall nicht notwendig.

WIREGUARD_x_LOCAL_HOST Standard-Einstellung: `WIREGUARD_x_LOCAL_HOST=""`

Optional: Der DNS-Name, unter dem flail im Internet erreichbar ist. Für die Funktion von WireGuard nicht erforderlich. Allerdings bietet es sich an, die Variable auf einen Hostname zu setzen, der z. B. von `OPT_DYNDNS` aktualisiert wird. Die Peer-Konfiguration für die Gegenstellen im WebInterface (QR-Code bzw. Download der Config-Datei) enthält dann diesen Wert als Gegenstelle.

```
WIREGUARD_x_LOCAL_HOST='some.domain.de'
```

WIREGUARD_x_LOCAL_IP6 Optional: IPv6-Adresse des Servers.

WIREGUARD_x_PUBLIC_KEY Standard-Einstellung: `WIREGUARD_x_PUBLIC_KEY=""`

Optional: Public Key der lokalen WireGuard-Instanz (Server).

Da der Public Key aus dem Private Key errechnet werden kann, ist der Public Key optional.

Die allermeisten Nutzer werden diese Variable nicht benötigen und stattdessen den Private Key beim ersten Start automatisch erstellen lassen und danach den Private Key aus dem Webinterface in die Konfiguration übernehmen. Der Public Key wird dabei automatisch aus dem Private Key abgeleitet.

```
WIREGUARD_x_PRIVATE_KEY='auto'
```

WIREGUARD_x_PUSH_DNS_N Optional: Anzahl der DNS-Server, die dem Peer in der Konfiguration mitgegeben werden. **Achtung:** WireGuard unterstützt kein Push von

Netzwerkparametern beim Verbindungsaufbau. Die DNS-Server werden in der Peer-Config, die im Webinterface als QR-Code und Download verfügbar ist, hinterlegt.

Standardwert: `WIREGUARD_x_PUSH_DNS_N='0'`

WIREGUARD_x_PUSH_DNS_x Optional: IPs von DNS-Servern, die in der Peer-Konfiguration hinterlegt werden. Achtung: WireGuard unterstützt kein Push von Netzwerkparametern beim Verbindungsaufbau. Die DNS-Server werden in der Peer-Config, die im Webinterface als QR-Code und Download verfügbar ist, hinterlegt und somit erst angewendet, wenn der Peer diese Konfiguration neu bekommt.

Es können sowohl IPv4- als auch IPv6-Adressen angegeben werden.

Beispiel:

```
WIREGUARD_1_PUSH_DNS_N='2'  
WIREGUARD_1_PUSH_DNS_1='10.0.0.1'  
WIREGUARD_1_PUSH_DNS_2='2001:4860:4860::8888'
```

Erforderliche Parameter Client (Peer)

WIREGUARD_x_PEER_N Standardwert: `WIREGUARD_x_PEER_N='0'`

Legt fest wie viele Peers wir konfigurieren wollen.

Beispiel: `WIREGUARD_1_PEER_N='3'`

WIREGUARD_x_PEER_x_NAME Der Name des WireGuard Peers. Dient nur zur Identifikation z. B. im Webinterface.

WIREGUARD_x_PEER_x_LOCAL_IP4 IP-Adresse (bei \32) des WireGuard Peers bzw. Netzwerk (z. B. \24), das sich hinter dem Peer verbirgt (z. B. bei Site-2-Site VPN). Die Adresse muss im selben Netzwerk liegen, wie die IP-Adresse des zugehörigen WireGuard Servers `WIREGUARD_x_LOCAL_IP4`

Beispiel:

```
WIREGUARD_1_LOCAL_IP4='10.0.0.1/24'  
WIREGUARD_1_PEER_1_LOCAL_IP4='10.0.0.2/32'  
WIREGUARD_1_PEER_2_LOCAL_IP4='10.0.0.3/32'
```

WIREGUARD_x_PEER_x_PUBLIC_KEY Der Public Key des WireGuard Peers.

Zum Verbindungsaufbau benötigt WireGuard zwar den Private Key der lokalen Instanz, vom Peer ist jedoch natürlich nur der Public Key erforderlich.

Es gibt zwei Möglichkeiten:

auto es werden Private und zugehöriger Public Key des Peers erstellt. Dies bietet sich an wenn man die Konfiguration später einfach per QR-Code oder Download der Config-Datei zum Peer übertragen möchte

some key ein gültiger WireGuard Public Key des Peers

Optionale Parameter Client (Peer)

WIREGUARD_x_PEER_x_ALLOWED_IPS_N Optional: Anzahl der IPs bzw. Netzwerke, die diesem einzelnen Peer vorgegeben werden sollen, diese durch den VPN-Tunnel zu routen.

Standardwert: `WIREGUARD_x_PEER_x_ALLOWED_IPS_N='0'`

WIREGUARD_x_PEER_x_ALLOWED_IPS_x Optional: IPs bzw. Netzwerke, die der jeweilige Peer durch den VPN-Tunnel routen soll. Hier angegebene Netze werden in der Peer-Config, die im Webinterface als QR-Code oder Datei einsehbar ist, entsprechend übernommen. Bei manueller Konfiguration der Peers wären diese Werte in `AllowedIPs` anzugeben. Die hier für jeden einzelnen Peer konfigurierten Netzwerke werden mit den Netzen aus `WIREGUARD_x_DEFAULT_ALLOWED_IPS` zusammengeführt und als `AllowedIPs` in der jeweiligen Peer-Config hinterlegt.

Es können sowohl IPv4- als auch IPv6-Netzwerke angegeben werden. Interne Identifier wie `IP_NET_x` und `IP6_NET_x` sind ebenfalls möglich.

Beispiel:

```
WIREGUARD_1_PEER_1_ALLOWED_IPS_N='2'
WIREGUARD_1_PEER_1_ALLOWED_IPS_1='IP_NET_2'
WIREGUARD_1_PEER_2_ALLOWED_IPS_2='192.168.1.0/24'
```

WIREGUARD_x_PEER_x_LOCAL_IP6 Optional: IPv6-Adresse des Servers. Die Adresse muss im selben Netzwerk liegen wie die Adresse des zugehörigen WireGuard Servers `WIREGUARD_x_LOCAL_IP6`.

WIREGUARD_x_PEER_x_PRIVATE_KEY Optional: Der Private Key des Peers. Im Gegensatz zum Public Key ist dieser nicht erforderlich. Falls man die Konfiguration allerdings per QR-Code oder Download der Config-Datei im Webinterface auf den übernehmen möchte, ist es sinnvoll, den Private Key des Peers hier anzugeben.

```
WIREGUARD_x_PEER_x_PUBLIC_KEY='auto'
```

So erzeugte Private (und evtl. Public Keys des Peers sollten statisch in die Konfigurationsdatei übernommen werden, da sonst bei jedem Neustart neue Keys erzeugt werden

und der Client sich erst wieder verbinden kann, wenn diese Keys entsprechend übertragen wurden.

Bei Private Keys der Peers sollte man sich überlegen, ob diese wirklich dauerhaft auf dem fli4l verbleiben sollen. Für die VPN-Verbindung sind nur die Public Keys des Peers notwendig (sowie Private Key des Servers). Man kann also durchaus die Keys der Peers automatisch erstellen lassen, dann aber nur die jeweiligen Public Keys in die statische Konfiguration übernehmen. Selbst wenn Angreifer dann Zugriff auf den fli4l hätten, könnten sie keine VPN-Verbindung erstellen, da der Private Key des Peers fehlt.

Allerdings leidet darunter etwas die Einfachheit: Wenn der Private Key des Peers nicht auf dem fli4l bekannt ist, ist die Peer-Konfiguration, die per QR-Code und Download angeboten wird, unvollständig und kann je nach Betriebssystem nicht eingelesen werden.

Man muss sich also entscheiden:

Sicherheit Private Keys der Peers nach automatischer Erstellung nicht in vpn.txt hinterlegen und höhere Sicherheit gegen eventuelle Eindringlinge.

Einfachheit Private Keys der Peers in vpn.txt hinterlegen und vollständige Peer-Konfigurationen per QR-Code und Download im Webinterface verfügbar haben.

WIREGUARD_x_PEER_x_PRESHARED_KEY Optional: mittels eines zusätzlichem Preshared Keys zwischen Server und Client kann die Sicherheit noch weiter erhöht werden.

Es gibt zwei Möglichkeiten:

auto Preshared Key wird bei (jedem!) Start automatisch erstellt

some key ein gültiger WireGuard Key

Wie auch die übrigen automatisch erstellten Keys, sollte auch der Preshared Key nach Erstellung beim ersten Start statisch in die Konfiguration übernommen werden, da ansonsten nach dem nächsten Reboot (und damit verbundener automatischer Neuerstellung der Keys) keine Verbindung mehr zum fli4l möglich ist.

WIREGUARD_x_PEER_x_REMOTE_HOST Optional: DNS-Name oder IP-Adresse des Peers. Der WireGuard Server nutzt dies, um die Verbindung zu initiieren. Falls nicht gesetzt, wartet der WireGuard Server auf eingehende gültige (verschlüsselte und signierte) Pakete von Peers und antwortet an die IP-Adresse, von der das Paket kam.

WIREGUARD_x_PEER_x_REMOTE_PORT Optional: Der Port, auf dem der jeweilige Peer lauscht. Der WireGuard Server nutzt dies zum initialen Verbindungsaufbau.

Standard-Einstellung: `WIREGUARD_x_PEER_x_REMOTE_PORT='51820'`

WIREGUARD_x_PEER_x_ROUTE_TO_N Optional: Die Anzahl der Netzwerke, die über diese VPN-Verbindung zum Peer geroutet werden sollen.

Standardwert: `WIREGUARD_x_PEER_x_ROUTE_TO_N='0'`

WIREGUARD_x_PEER_x_ROUTE_TO_x Optional: Die einzelnen Netzwerke, die über diesen Peer geroutet werden sollen.

Beispiel: Hinter Peer 1 sind die beiden Netzwerke 192.168.1.1/24 und 192.168.188.1/24 erreichbar:

```
WIREGUARD_1_PEER_1_ROUTE_TO_N='2'  
WIREGUARD_1_PEER_1_ROUTE_TO_1='192.168.1.1/24'  
WIREGUARD_1_PEER_1_ROUTE_TO_2='192.168.188.1/24'
```

Bitte die entsprechenden Forward- und Post-Routing-Regeln in `base.txt` nicht vergessen (ggf. auch auf der Gegenseite).

Werkzeug wg-tool

Zur Verwaltung einiger Grundfunktionen bringt Wireguard auf fli4l das Kommandozeilenwerkzeug wg-tool mit:

```
fli4l-rgb 4.0.0-r59812M # wg-tool  
This is a helper script to start and stop WireGuard connections  
Usage:  
  wg-tool <wg interface> <up|down>  
  wg-tool <wg interface> reresolve <peerName>  
      where <peerName> is the name defined in WIREGUARD_x_PEER_x_NAME  
  
example: wg-tool wg1 down  
example: wg-tool wg1 reresolve peer5Name
```

Im Normalbetrieb ist die Nutzung des Tools nicht erforderlich. Es erlaubt bei Bedarf jedoch auf einfache Art das Starten und Stoppen konfigurierter WireGuard-Instanzen.

Beispiel:

```
wg-tool wg1 down
```

`wg interface` ergibt sich aus der Konfiguration in `vpn.txt` wobei die Zählung der Schnittstellen mit 0 beginnt. `WIREGUARD_1` resultiert also in `wg0`.

Zusätzlich erlaubt wg-tool die erneute Auflösung des DNS-Namens eines Peers ohne den Server neu starten zu müssen, also ohne Unterbrechung der Verbindungen eventueller weiterer Peers, die mit der selben Server-Instanz verbunden sind. Im Normalbetrieb sollte auch dies nicht erforderlich sein.

Eine bekannte Ausnahme ist, wenn der DNS-Name des Peers zum Boot-Zeitpunkt des fli4l nicht aufgelöst werden kann. In diesem Fall wird zwar der WireGuard Server gestartet, die Gegenstelle dieses einen Peers jedoch nicht gesetzt. Falls der Peer von sich aus die Verbindung initiiert, ist auch dies kein Problem, da bei WireGuard ein Verbindungsaufbau grundsätzlich von beiden Seiten (Peers) gleichermaßen initiiert werden kann.

Ansonsten bietet es sich an, entweder wg-tool manuell aufzurufen oder - meist einfacher - über Paket easycron zu automatisieren.

Die Parameter ergeben sich wieder aus der Konfiguration in `vpn.txt`:

wg interface ergibt sich aus `WIREGUARD_x`. Die Zählung der Schnittstellen beginnt dabei mit 0. `WIREGUARD_x` resultiert also in `wg0`.

peerName ist der Name des Peers in `WIREGUARD_x_PEER_x_NAME`.

Beispiel:


```
WIREGUARD_2_PEER_3_NAME='peerTestName'
WIREGUARD_2_PEER_3_REMOTE_HOST='somehost.dyndns.de'

wg-tool wg1 reresolve peerTestName
```

Oder alternativ per cron-Job über Paket easycron alle 6 Stunden:

```
EASYCRON_x_COMMAND='wg-tool wg1 reresolve peerTestName'
EASYCRON_x_TIME='* */6 * * *'
```

WireGuard Beispiele

Beispiel: minimal notwendige Konfiguration

```
OPT_WIREGUARD='yes'

WIREGUARD_N='1'
WIREGUARD_1_NAME='RoadWarriors'
WIREGUARD_1_LOCAL_IP4='10.0.0.1/24'
WIREGUARD_1_LISTEN_PORT='50002'

WIREGUARD_1_PEER_N='1'
WIREGUARD_1_PEER_1_NAME='peer1'
WIREGUARD_1_PEER_1_LOCAL_IP4='10.0.0.2/32'
```

Diese Konfiguration erzeugt einen lokalen WireGuard-Endpunkt auf dem fl4l mit Namen *RoadWarriors* sowie einen Peer mit Namen *peer1*.

Alle notwendigen Krypto-Schlüssel (jeweils private und public key für Server und Peer) werden automatisch erstellt und können später im Webinterface eingesehen und in die Konfiguration übernommen werden.

Zusätzlich wird die Peer-Konfiguration im Webinterface

- als QR-Code für Android- und iPhone-App
- als Download für Windows-/Linux-Clients

bereitgestellt.

Die automatisch erstellten private Keys sollten nach dem ersten Start statisch in die Konfigurationsdatei `vpn.txt` übernommen werden. Ansonsten werden sie bei jedem Neustart des Routers neu erzeugt und die WireGuard-Verbindung kann erst wieder aufgebaut werden, wenn die entsprechenden Parameter an den Peer neu übertragen wurden.

Eine vollständige, minimale Konfiguration, die auch Neustarts problemlos übersteht sieht wie folgt aus:

```
OPT_WIREGUARD='yes'

WIREGUARD_N='1'
WIREGUARD_1_NAME='RoadWarriors'
WIREGUARD_1_LOCAL_IP4='10.0.0.1/24'
WIREGUARD_1_LISTEN_PORT='50002'
WIREGUARD_1_PRIVATE_KEY='+PxRP6JmwDyM1R+KdeN+vIL2UY2WB+eG/AHLJ/OdsHo='
```

```
WIREGUARD_1_PEER_N='1'  
WIREGUARD_1_PEER_1_NAME='peer1'  
WIREGUARD_1_PEER_1_LOCAL_IP4='10.0.0.2/32'  
WIREGUARD_1_PEER_1_PUBLIC_KEY='gP+moyiQtfe007UH3ASGAy1tX0sLhivLIXGNP0IKbG8='
```

Beispiel: RoadWarriors - von unterwegs ins Heimnetzwerk

Die folgende Konfiguration zeigt ein typisches Roadwarrior-Beispiel um von unterwegs jederzeit Zugriff aufs heimische Netz haben zu können. Sicherer Zugriff auf die Heimautomatisierung, den Fileserver zu Hause den digitalen Videorekorder, etc. können so umgesetzt werden.

Die notwendigen Private und Public Keys werden automatisch beim Booten erstellt und sollten nach dem ersten Start aus dem WireGuard Webinterface in die Konfigurationsdatei übernommen werden. Sonst werden beim nächsten Start neue Keys erzeugt und man kommt mit den vorherigen Keys von extern nicht mehr ins VPN.

Folgende Ziele wollen wir mit der Beispiel-Konfiguration erreichen

Dual-Stack Wir wollen sowohl IPv4- als auch IPv6-Adressen nutzen können.

DNS DNS-Namen aus dem internen Netz sollen auch unterwegs aufgelöst werden, wir wollen uns nicht die IP-Adressen der internen Hosts merken müssen.

kein Default-Gateway Es soll nur Traffic ins interne Netz zu Hause durch den Tunnel geroutet werden. Traffic des mobilen Geräts ins Internet soll weiterhin direkt über die LTE- oder fremde WLAN-Verbindung gehen.

MASQ Wir wollen kein Masquerading der Peer-Adressen sondern direktes Routing in das interne Netz.

Network Prefix Der Einfachheit halber wollen wir direkt Netzwerk-Präfixe nutzen.

Folgende Annahmen zu internen Netzen, IPs, etc. wurden getroffen

internes Netz IP_NET_1 ist das interne Netz, das per WireGuard angebunden erreicht werden soll

DynDNS Der fli4l ist unter router.dyndns.name erreichbar

IPv4 der WireGuard-Tunnel soll das IPv4-Netz 10.0.0.1/24 nutzen.

IPv6 der WireGuard-Tunnel soll das IPv6-Netz 'fd00:cfcf:abcd::/48' nutzen.

Um diese Konfiguration umzusetzen, müssen folgende Dateien angepasst werden

vpn.txt Für die Konfiguration von WireGuard-Server und Peers

base.txt zur Konfiguration des Network-Prefixes sowie der notwendigen Regeln für den Paketfilter

Konfiguration Wireguard Server und Peer in **vpn.txt**:

```
OPT_WIREGUARD='yes'

WIREGUARD[] {
    NAME='RoadWarriors'
    LOCAL_IP4='{wg-RoadWarriors}+0.0.0.1/24'
    LOCAL_IP6='{wg-RoadWarriors}+::1/64'
    PRIVATE_KEY='auto'
    LISTEN_PORT = '50002'
    LOCAL_HOST='router.dyndns.name'
    DEFAULT_ALLOWED_IPS[]='IP_NET_1'
    PUSH_DNS[]='10.0.0.1'
    PUSH_DNS[]='fd00:cfcf:abcd::1'

    PEER[] {
        NAME='Peer1'
        LOCAL_IP4='{wg-RoadWarriors}+0.0.0.2/32'
        LOCAL_IP6='{wg-RoadWarriors}+::2/128'
        PRIVATE_KEY='auto'
    }
}
```

Konfiguration der IPv4- und IPv6-Netze mittels Network Prefixes in `base.txt`:

```
OPT_NET_PREFIX='yes'
NET_PREFIX
{
    []
    {
        NAME='wg-RoadWarriors' # Netzwerk-Präfix für den WireGuard-Tunnel
        TYPE='static'
        STATIC_IPV4='10.0.0.0/24'
        STATIC_IPV6='fd00:cfcf:abcd::/48'
    }
}
```

Paketfilter-Regeln für IPv4 in `base.txt`:

```
# IPv4 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF_INPUT[]='{wg-RoadWarriors.prefix} ACCEPT'
{
    COMMENT='Zugriff WireGuard RoadWarriors auf fli4l'
}

# IPv4 Forward-Regel, um Zugriff auf das Netz hinter dem fli4l zu erlauben
PF_FORWARD[]='{wg-RoadWarriors.prefix} IP_NET_1 ACCEPT'
{
    COMMENT='Weiterleitung WireGuard RoadWarriors in IP_NET_1 zulassen'
}

# IPv4 Postrouting-Regel, um Netz hinter fli4l ohne Masquerading zu erreichen
PF_POSTROUTING[]='{wg-RoadWarriors.prefix} IP_NET_1 ACCEPT'
{
    COMMENT='Zugriff WireGuard RoadWarriors auf IP_NET_1 ohne Masquerading'
```

```
}
```

Paketfilter-Regeln für IPv6 in `base.txt`:

```
# IPv6 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF6_INPUT[]='{rgb-RoadWarriors.prefix} ACCEPT'
{
    COMMENT='Zugriff WireGuard RoadWarriors auf fli4l'
}

# IPv6 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF6_FORWARD[]='{rgb-RoadWarriors.prefix} IPV6_NET_1 ACCEPT'
{
    COMMENT='Zugriff WireGuard RoadWarriors auf IPV6_NET_1'
}

# IPv6 Postrouting-Regel, um Netz hinter fli4l ohne Masquerading zu erreichen
PF6_POSTROUTING[]='{rgb-RoadWarriors.prefix} IPV6_NET_1 ACCEPT'
{
    COMMENT='Zugriff WireGuard RoadWarriors auf IPV6_NET_1 ohne Masquerading'
}
```

Beispiel: RoadWarriors - kompletter Traffic durch den Tunnel (default route)

Falls der komplette Traffic des mobilen Geräts immer durch den WireGuard-Tunnel gehen soll (Default-Route durch den VPN-Tunnel), lässt sich dies mit wenigen Anpassungen entsprechend ändern.

Konfiguration der Default-Routen des Peers. Die so erstellte Konfiguration muss per Download oder erneutem Einlesen des QR-Codes erneut auf das mobile Gerät übertragen werden
Neuer Peer in `vpn.txt`

```
PEER[] {
    NAME='Peer2'
    LOCAL_IP4='{wg-RoadWarriors}+0.0.0.3/32'
    LOCAL_IP6='{wg-RoadWarriors}+::3/128'
    PRIVATE_KEY='auto'
    ALLOWED_IPS[] = '0.0.0.0/0'
    ALLOWED_IPS[] = '::/0'
}
```

Ergänzung der Paketfilter-Regeln für IPv4 und IPv6 in `base.txt`:

```
# IPv4-Regel, um Zugriff auf Netze und Internet hinter dem fli4l zu erlauben
PF_FORWARD[]='{wg-RoadWarriors.prefix} ACCEPT'
{
    COMMENT='Weiterleitung WireGuard RoadWarriors in alle Netze des fli4l'
}

# IPv4-Regel, um Pakete ins Internet zu maskieren
PF_POSTROUTING[]='{wg-RoadWarriors.prefix} {DHCPv4-cable} MASQUERADE'
{
    COMMENT='MASQ WireGuard RoadWarriors to Internet'
```

```
}

# IPv6-Regel, um Zugriff auf Netze und Internet hinter dem fli4l zu erlauben
PF6_FORWARD[]='{wg-RoadWarriors.prefix} ACCEPT'
{
    COMMENT='Weiterleitung WireGuard RoadWarriors in alle Netze des fli4l'
}

# IPv6-Regel, um Pakete ins Internet zu maskieren
PF6_POSTROUTING[]='{wg-RoadWarriors.prefix} {DHCPv6-cable} MASQUERADE'
{
    COMMENT='MASQ WireGuard RoadWarriors to Internet'
}
```

Wir gehen dabei davon aus, dass in `circuits.txt`:

DHCPv4-cable der Name für den IPv4-Circuit ist

DHCPv6-cable der Name für den IPv6-Circuit ist

Die Beispiel-Konfiguration geht von einem fli4l hinter einer FritzBox aus wie im Wiki⁸ beschrieben

Beispiel: Site2Site-VPN

Die folgende Konfiguration zeigt ein typisches Beispiel, um zwei Standorte per WireGuard VPN-Tunnel zu verbinden. An beiden Standorten kümmert sich ein fli4l um Routing und verwaltet jeweils weitere interne Netze.

Folgende Ziele wollen wir mit der Beispiel-Konfiguration erreichen

Dual-Stack Wir wollen sowohl IPv4- als auch IPv6-Adressen nutzen können.

DNS DNS-Namen aus dem jeweils anderen Netz sollen aufgelöst werden.

MASQ Wir wollen kein Masquerading zwischen den Netzen sondern direktes Routing in die jeweiligen interne Netz hinter den beiden Standorte.

VPN-Netz Das VPN-Netz und die jeweiligen IPs dienen jeweils nur als Transportnetz zwischen den beiden fli4l. Sie tauchen nicht in den lokalen Netzen auf.

Network Prefix Der Einfachheit halber wollen wir direkt Netzwerk-Präfixe nutzen.

Folgende Annahmen zu internen Netzen, IPs, etc. wurden getroffen
Standort A:

- Ein internes IPv4-Netz 192.168.1.0/24
- Ein internes IPv6-Netz fd5e:aaaa:aaaa:aaaa::/64
- Der fli4l ist unter Dyndns fli4lA.dyndns.name erreichbar
- Lokale Domain ist standortA.home.somedomain.net (siehe DOMAIN_NAME in `base.txt`)

⁸siehe <https://web.networks.org/wiki/pages/viewpage.action?pageId=35651587>

Standort B:

- Ein internes IPv4-Netz 192.168.2.0/24
- Ein internes IPv6-Netz fd5e:bbbb:bbbb:bbbb::/64
- Der fli4l ist unter Dyndns fli4lb.somedomain.org erreichbar
- Lokale Domain ist standortb.domain.de

Um diese Konfiguration umzusetzen, müssen auf beiden Seiten folgende Dateien angepasst werden

vpn.txt Für die Konfiguration von WireGuard-Server und Peers

base.txt zur Konfiguration des Network-Prefixes sowie der notwendigen Regeln für den Paketfilter

dns_dhcp.txt um den jeweils anderen fli4l als DNS-Server für lokale Namen zu konfigurieren

auf beiden fli4ls an beiden Standorten identisch

Netze in `base.txt`:

```
OPT_NET_PREFIX='yes'
NET_PREFIX
{
    []
    {
        NAME='fli4a-intern'
        TYPE='static'
        STATIC_IPV4='192.168.1.0/24'
        STATIC_IPV6='fd5e:aaaa:aaaa::/48'
    }
    []
    {
        NAME='fli4b-intern'
        TYPE='static'
        STATIC_IPV4='192.168.2.0/24'
        STATIC_IPV6='fd5e:bbbb:bbbb::/48'
    }
    []
    {
        NAME='wg-Site2Site'
        TYPE='static'
        STATIC_IPV4='10.0.0.0/24'
        STATIC_IPV6='fd5e:cccc:cccc::/48'
    }
}
```

Konfiguration fli4l an Standort A (neues Config-Format)

WireGuard VPN-Tunnel Standort A in `vpn.txt`:

```
OPT_WIREGUARD='yes'

WIREGUARD[] {
    NAME='Site2Site-VPN'
    LOCAL_IP4='{wg-Site2Site}+0.0.0.1/24'
    LOCAL_IP6='{wg-Site2Site}+::1/64'
    PRIVATE_KEY='iFwKnsJo/N00WWiuMPxugdP0q9jqYhIP46uQi1AZj3E=' # unbedingt ändern
    LISTEN_PORT='50002'
    DEFAULT_ALLOWED_IPS[]='{fli4a-intern.prefix}' # von allen Peers Pakete
                                                    # für dieses Netz erlauben

    PEER[] {
        NAME='fli4l-B'
        LOCAL_IP4='{wg-Site2Site}+0.0.0.2/32'
        LOCAL_IP6='{wg-Site2Site}+::2/128'
        REMOTE_HOST='fli4lb.somedomain.org'
        REMOTE_PORT='50000'
        PUBLIC_KEY='eYyUIGxafeJwLC+BbFTJZ45CtKvCU3TlZODFCx9MlD0='
        ROUTE_TO[]='{fli4b-intern.prefix}' # Pakete in dieses Netz durch den
                                           # Tunnel routen
    }
}
```

Paketfilter-Regeln Standort A für IPv4 und IPv6 in `base.txt`:

```
# IPv4 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF_INPUT[]='{fli4b-intern.prefix} ACCEPT'
{
    COMMENT='IPv4 Zugriff von Standort B auf fli4l'
}

# IPv4 Forward-Regel, um Zugriff auf das Netz hinter dem fli4l zu erlauben
PF_FORWARD[]='{fli4b-intern.prefix} IP_NET_1 ACCEPT'
{
    COMMENT='IPv4 Weiterleitung von Standort B auf IP_NET_1 zulassen'
}

# IPv4 Postrouting-Regel, um Netz hinter fli4l ohne Masquerading zu erreichen
PF_POSTROUTING[]='{fli4b-intern.prefix} IP_NET_1 ACCEPT'
{
    COMMENT='IPv4 Zugriff von Standort B auf IP_NET_1 ohne Masquerading'
}

# IPv6 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF6_INPUT[]='{fli4b-intern.prefix} ACCEPT'
{
    COMMENT='IPv6 Zugriff von Standort B auf fli4l'
}
```

```
# IPv6 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF6_FORWARD[]='{fli4b-intern.prefix} IPV6_NET_1 ACCEPT'
{
    COMMENT='IPv6 Weiterleitung von Standort B auf IPV6_NET_1 zulassen'
}

# IPv6 Postrouting-Regel, um Netz hinter fli4l ohne Masquerading zu erreichen
PF6_POSTROUTING[]='{fli4b-intern.prefix} IPV6_NET_1 ACCEPT'
{
    COMMENT='IPv6 Zugriff Zugriff Standort B auf IPV6_NET_1 ohne Masquerading'
}
```

DNS Zone Delegation an Standort A in `dns_dhcp.txt`, um auch DNS-Namen des Netzes hinter fli4l-B aufzulösen:

```
OPT_DNS='yes'
DNS_ZONE_DELEGATION_N='1'
DNS_ZONE_DELEGATION_1_UPSTREAM_SERVER_N='1'
DNS_ZONE_DELEGATION_1_UPSTREAM_SERVER_1_IP='192.168.2.1'    # an fli4l-B delegieren
DNS_ZONE_DELEGATION_1_UPSTREAM_SERVER_1_QUERYSOURCEIP='IP_NET_1_IPADDR'

DNS_ZONE_DELEGATION_1_DOMAIN_N='1'
DNS_ZONE_DELEGATION_1_DOMAIN_1='standortb.domain.de'
DNS_ZONE_DELEGATION_1_NETWORK_N='1'
DNS_ZONE_DELEGATION_1_NETWORK_1='192.168.2.0/24'
```

Konfiguration fli4l an Standort B (altes Config-Format)

WireGuard VPN-Tunnel Standort B in `vpn.txt`:

```
OPT_WIREGUARD='yes'

WIREGUARD_N='1'
WIREGUARD_NAME='Site2Site-VPN'
WIREGUARD_LOCAL_IP4='{wg-Site2Site}+0.0.0.2/24'
WIREGUARD_LOCAL_IP6='{wg-Site2Site}+:::2/64'
WIREGUARD_PRIVATE_KEY='eJ8s8+oyItWPUSs/5TDBroGWleEV7W/4p98qUY2xD2I='
WIREGUARD_LISTEN_PORT='50002'

WIREGUARD_1_PEER_N='1'
WIREGUARD_1_PEER_1_NAME='fli4l-A'
WIREGUARD_1_PEER_1_LOCAL_IP4='{wg-Site2Site}+0.0.0.1/32'
WIREGUARD_1_PEER_1_LOCAL_IP6='{wg-Site2Site}+:::1/128'
WIREGUARD_1_PEER_1_REMOTE_HOST='fli4lA.dyndns.name'
WIREGUARD_1_PEER_1_REMOTE_PORT='50002'
WIREGUARD_1_PEER_1_PUBLIC_KEY='S9GsipIvFl36HkcXaET8v0G+UuAw6onDiC+22jJJjVs='
WIREGUARD_1_PEER_1_ALLOWED_IPS_N='1'
WIREGUARD_1_PEER_1_ALLOWED_IPS_1='{fli4a-intern.prefix}'
WIREGUARD_1_PEER_1_ROUTE_TO_N='1'
WIREGUARD_1_PEER_1_ROUTE_TO_1='{fli4a-intern.prefix}'
```

Paketfilter-Regeln Standort B für IPv4 und IPv6 in `base.txt`:


```
# IPv4 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF_INPUT[]={fli4a-intern.prefix} ACCEPT'
{
    COMMENT='IPv4 Zugriff von Standort A auf fli4l'
}

# IPv4 Forward-Regel, um Zugriff auf das Netz hinter dem fli4l zu erlauben
PF_FORWARD[]={fli4a-intern.prefix} IP_NET_1 ACCEPT'
{
    COMMENT='IPv4 Weiterleitung von Standort A auf IP_NET_1 zulassen'
}

# IPv4 Postrouting-Regel, um Netz hinter fli4l ohne Masquerading zu erreichen
PF_POSTROUTING[]={fli4a-intern.prefix} IP_NET_1 ACCEPT'
{
    COMMENT='IPv4 Zugriff von Standort A auf IP_NET_1 ohne Masquerading'
}

# IPv6 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF6_INPUT[]={fli4a-intern.prefix} ACCEPT'
{
    COMMENT='IPv6 Zugriff von Standort A auf fli4l'
}

# IPv6 Input-Regel, um Zugriff auf fli4l-Router zu erlauben
PF6_FORWARD[]={fli4a-intern.prefix} IPV6_NET_1 ACCEPT'
{
    COMMENT='IPv6 Weiterleitung von Standort A auf IPV6_NET_1 zulassen'
}

# IPv6 Postrouting-Regel, um Netz hinter fli4l ohne Masquerading zu erreichen
PF6_POSTROUTING[]={fli4a-intern.prefix} IPV6_NET_1 ACCEPT'
{
    COMMENT='IPv6 Zugriff Zugriff von Standort A auf IPV6_NET_1 ohne Masquerading'
}
```

DNS Zone Delegation an Standort B in `dns_dhcp.txt`, um auch DNS-Namen des Netzes hinter fli4l-A aufzulösen:

```
OPT_DNS='yes'
DNS_ZONE_DELEGATION_N='1'
DNS_ZONE_DELEGATION_1_UPSTREAM_SERVER_N='1'
DNS_ZONE_DELEGATION_1_UPSTREAM_SERVER_1_IP='192.168.1.1'    # an fli4l-A delegieren
DNS_ZONE_DELEGATION_1_UPSTREAM_SERVER_1_QUERYSOURCEIP='IP_NET_1_IPADDR'

DNS_ZONE_DELEGATION_1_DOMAIN_N='1'
DNS_ZONE_DELEGATION_1_DOMAIN_1='standortA.home.somedomain.net'
DNS_ZONE_DELEGATION_1_NETWORK_N='1'
DNS_ZONE_DELEGATION_1_NETWORK_1='{fli4a-intern.prefix}'
```

Abbildungsverzeichnis

Tabellenverzeichnis

1.1	Arten der GRE-Paketerzeugung	4
1.2	Einstellungen für Provider, die PPTP nutzen	4

Index

CIRC_x_PPP_PPTP_LOGLEVEL, [5](#)
CIRC_x_PPP_PPTP_PEER, [4](#)
CIRC_x_PPP_PPTP_REORDER_TIMEOUT, [5](#)
CIRC_x_PPP_PPTP_SERVER_ALLOW_-
FROM_N, [6](#)
CIRC_x_PPP_PPTP_SERVER_ALLOW_-
FROM_y, [6](#)
CIRC_x_PPP_PPTP_SERVER_LISTEN,
[5](#)
CIRC_x_PPP_PPTP_SERVER_SESSIONS,
[6](#)
CIRC_x_PPP_PPTP_TYPE, [4](#)
FASTD_SECRET, [9](#)
OPT_FASTD, [9](#)
OPT_PPP_PPTP, [3](#)
OPT_PPP_PPTP_SERVER, [5](#)
OPT_WIREGUARD, [10](#)
WIREGUARD_DEFAULT_OPEN_PORT,
[12](#)
WIREGUARD_N, [11](#)
WIREGUARD_x_DEFAULT_ALLOWED_-
IPS, [12](#)
WIREGUARD_x_DEFAULT_ALLOWED_-
IPS_N, [12](#)
WIREGUARD_x_KEEP_ALIVE, [12](#)
WIREGUARD_x_LISTEN_PORT, [11](#)
WIREGUARD_x_LOCAL_HOST, [12](#)
WIREGUARD_x_LOCAL_IP4, [11](#)
WIREGUARD_x_LOCAL_IP6, [12](#)
WIREGUARD_x_NAME, [11](#)
WIREGUARD_x_PEER_N, [13](#)
WIREGUARD_x_PEER_x_ALLOWED_-
IPS_N, [14](#)
WIREGUARD_x_PEER_x_ALLOWED_-
IPS_x, [14](#)
WIREGUARD_x_PEER_x_LOCAL_IP4,
[13](#)
WIREGUARD_x_PEER_x_LOCAL_IP6,
[14](#)
WIREGUARD_x_PEER_x_NAME, [13](#)
WIREGUARD_x_PEER_x_PRESHARED_-
KEY, [15](#)
WIREGUARD_x_PEER_x_PRIVATE_-
KEY, [14](#)
WIREGUARD_x_PEER_x_PUBLIC_KEY,
[13](#)
WIREGUARD_x_PEER_x_REMOTE_-
HOST, [15](#)
WIREGUARD_x_PEER_x_REMOTE_-
PORT, [15](#)
WIREGUARD_x_PEER_x_ROUTE_TO_-
N, [15](#)
WIREGUARD_x_PEER_x_ROUTE_TO_-
x, [15](#)
WIREGUARD_x_PRIVATE_KEY, [11](#)
WIREGUARD_x_PUBLIC_KEY, [12](#)
WIREGUARD_x_PUSH_DNS_N, [13](#)
WIREGUARD_x_PUSH_DNS_x, [13](#)