Hardware

Installing a VPN server on a router with OpenWrt (WireGuard)

7 March 2020, 12:01 GMT 17 August 2024, 09:39 BST 12 min.

Contents

1. WireGuard and OpenWrt (Server)
2. WireGuard and OpenWrt (Client)
    a. Adding another client (peer)
3. WireGuard and IPv6 Tunnel

If you have your own router with OpenWrt software, you have probably thought more than once about how to set up a VPN server on it so that you can connect to your local network from outside your home/office and have access to local data, such as a network drive or printer.

In the case of OpenWrt, if you start looking for information about VPN, you will immediately come across information about **OpenVPN**.

Initially I thought about creating a post about **installing OpenVPN on a router with OpenWrt**, however, the further into the forest, the more trees. The configuration can and starts off simple. You can even add an interface from the browser, however, everything is far from simple and clear. Being halfway through my post, I realized that further creation of it will introduce more complications than I initially assumed. At this point I remembered about **WireGuard**.

Both **OpenVPN** and **WireGuard** are not available by default on operating systems (Windows, macOS, as well as Android or iOS). In both cases, you must use a dedicated application to connect.

WireGuard won over **OpenVPN** for me because it is much easier to set up and manage. It is also much faster, offers a higher level of security and, in the case of mobile devices, does not eat up our battery like OpenVPN does.

> VPN Ranks ↗ has put together a comprehensive comparison of both solutions, so if you're interested, I refer you to it.

While in the case of OpenVPN, you had to execute a series of commands related to setting up the server from the SSH terminal, in the case of WireGuard, we can set up our server with its minimal use and do the rest via a web browser. All in a user-friendly way.

So let's get started…

WireGuard and OpenWrt (Server)

First, we need to install a few packages for WireGuard itself and those that add the ability to manage it from a web browser.

From the SSH connection to our router, we issue the following commands.

To update package information:

Copy

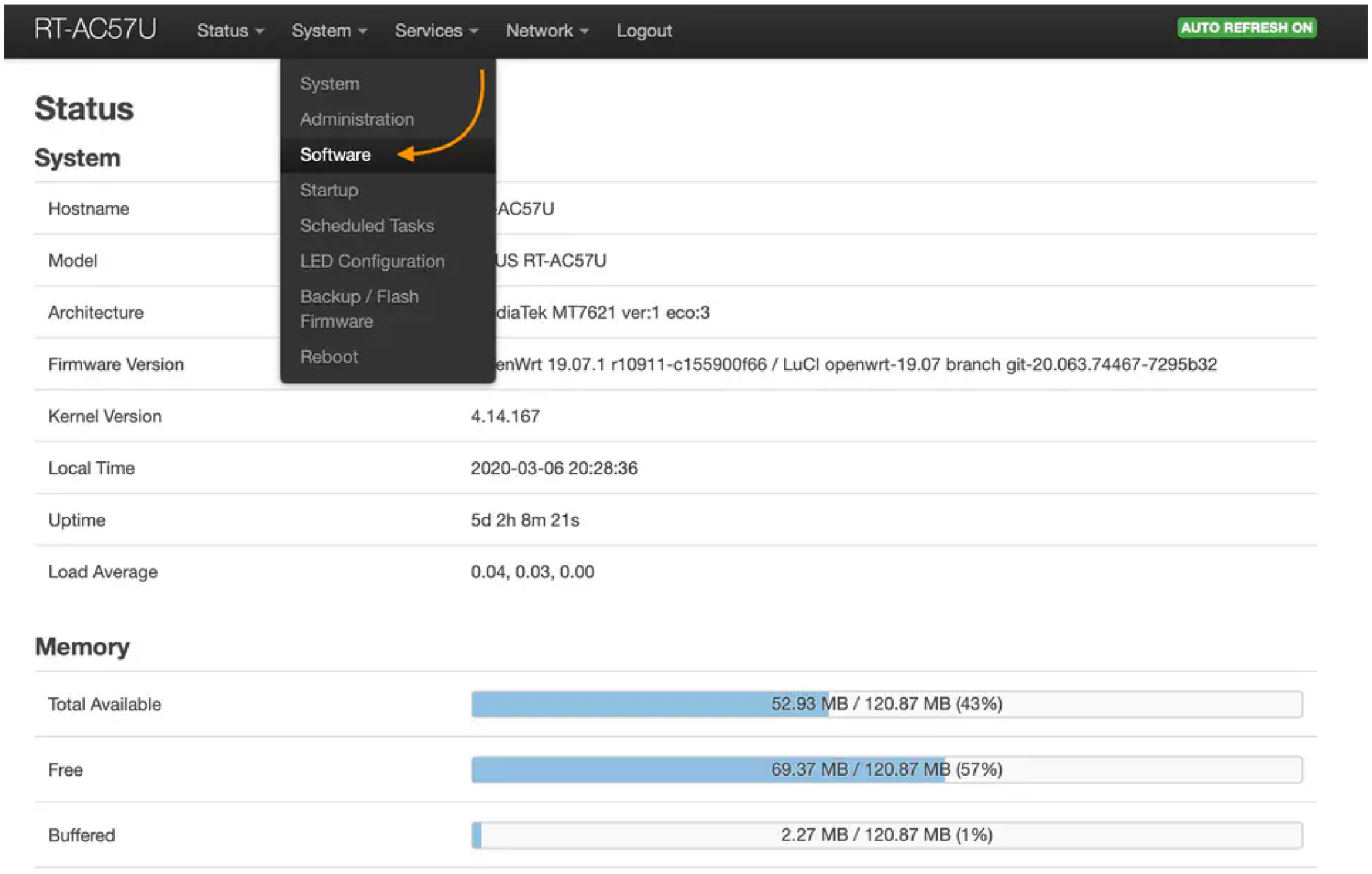
```
opkg update
```

And to install the packages we execute the following command:

Copy

```
opkg install luci-app-wireguard luci-proto-wireguard kmod-wireguard wireguard-tools
```

We can also do this from a web browser by logging into our router and going to **System > Software**



Status

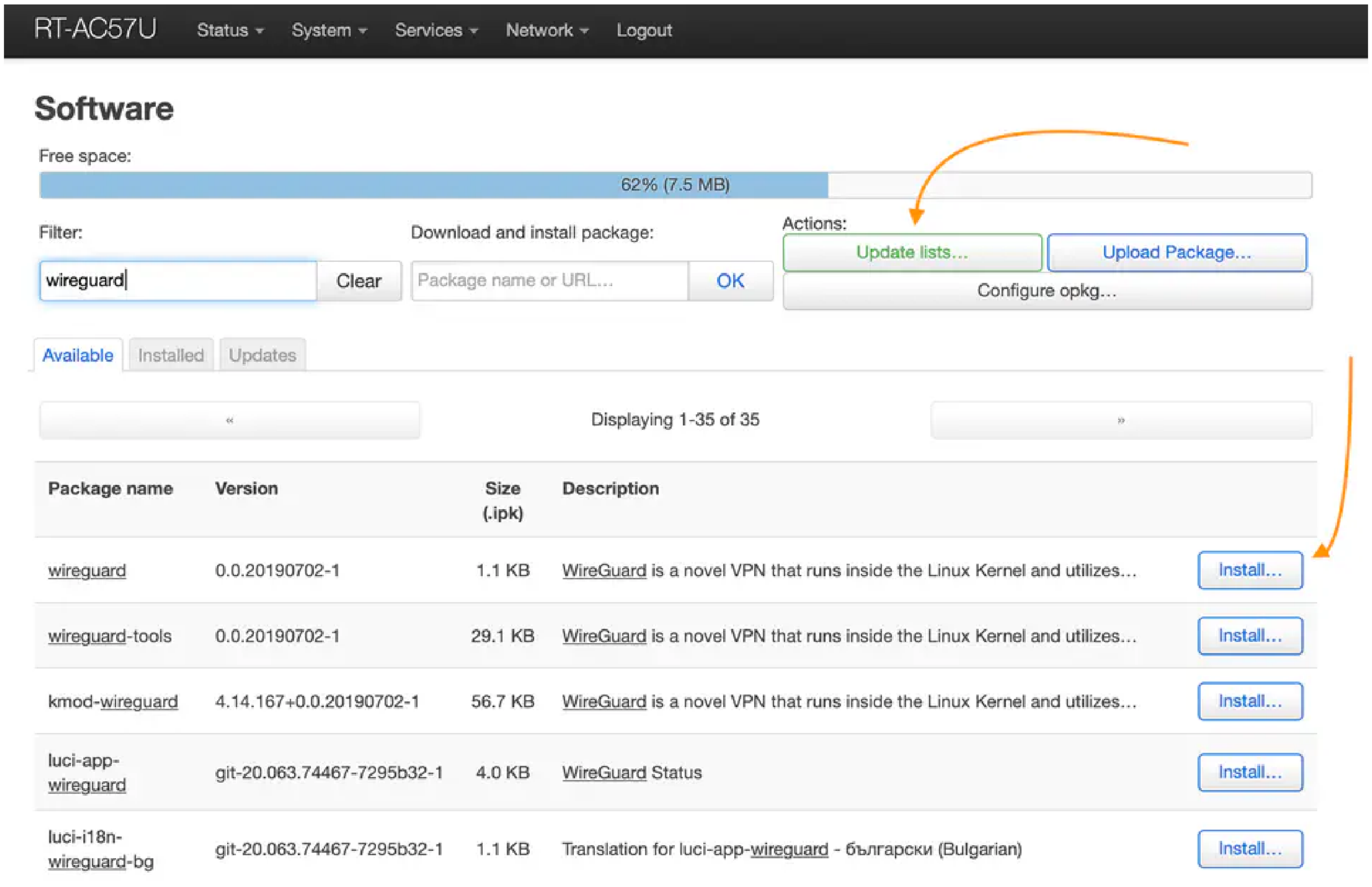
System

Hostname	AC57U
Model	US RT-AC57U
Architecture	diaTek MT7621 ver:1 eco:3
Firmware Version	enWrt 19.07.1 r10911-c155900f66 / LuCI openwrt-19.07 branch git-20.063.74467-7295b32
Kernel Version	4.14.167
Local Time	2020-03-06 20:28:36
Uptime	5d 2h 8m 21s
Load Average	0.04, 0.03, 0.00

Memory

Total Available	52.93 MB / 120.87 MB (43%)
Free	69.37 MB / 120.87 MB (57%)
Buffered	2.27 MB / 120.87 MB (1%)

And so for the `luci-app-wireguard luci-proto-wireguard kmod-wireguard wireguard-tools` packages we click on the **Install** buttons.



Package name	Version	Size (.ipk)	Description	
wireguard	0.0.20190702-1	1.1 KB	WireGuard is a novel VPN that runs inside the Linux Kernel and utilizes...	Install...
wireguard-tools	0.0.20190702-1	29.1 KB	WireGuard is a novel VPN that runs inside the Linux Kernel and utilizes...	Install...
kmod-wireguard	4.14.167+0.0.20190702-1	56.7 KB	WireGuard is a novel VPN that runs inside the Linux Kernel and utilizes...	Install...
luci-app-wireguard	git-20.063.74467-7295b32-1	4.0 KB	WireGuard Status	Install...
luci-i18n-wireguard-bg	git-20.063.74467-7295b32-1	1.1 KB	Translation for luci-app-wireguard - български (Bulgarian)	Install...

If you do not see the appropriate packages after entering **wireguard** in the search bar, click the **Update list** button to update the information on available packages (required after every router startup).

Since WireGuard uses kernel-level elements for its functionality, we need to restart our router for everything to load correctly at this stage

Copy

```
reboot
```

> Do not proceed if you have not restarted your router after installing the packages.

Before we go any further, we need to create a unique key for our server, which will be necessary to establish a connection, and a key for the first client (called **peer**).

We create a folder to store the server key

Copy

```
mkdir -p /etc/wireguard
```

We generate the server key

Copy

```
wg genkey | tee /etc/wireguard/server-privatekey | wg pubkey > /etc/wireguard/server-publickey
```

We generate the first client key

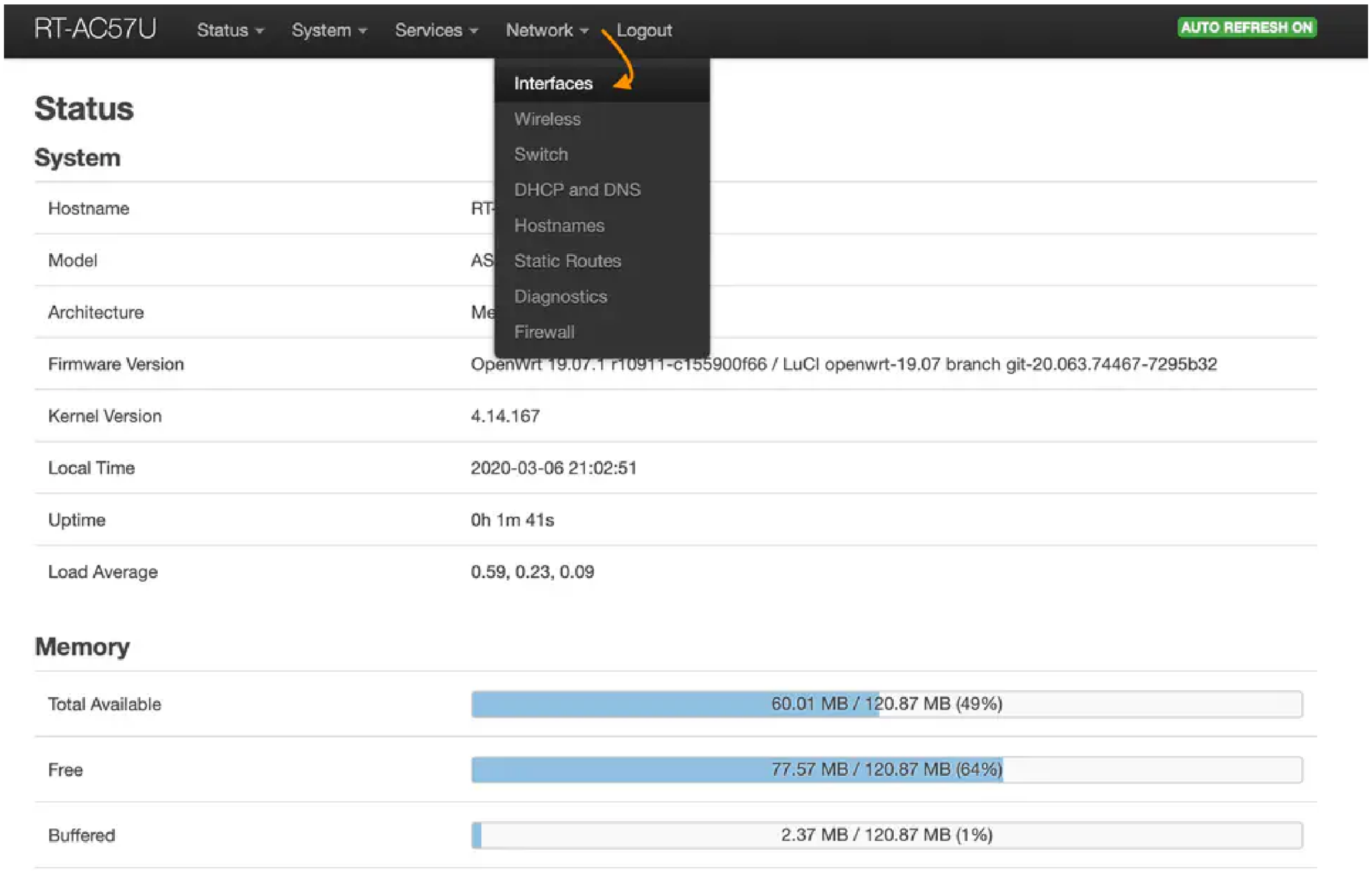
Copy

```
wg genkey | tee client1-privatekey | wg pubkey > client1-publickey
```

Now we can move on.

The next step is to create a WireGuard interface from the browser, similarly to how we set up a **lan0**, **wan0** connection or other router network settings.

We go to **Network > Interfaces** and add a new interface using the **Add new interface** button



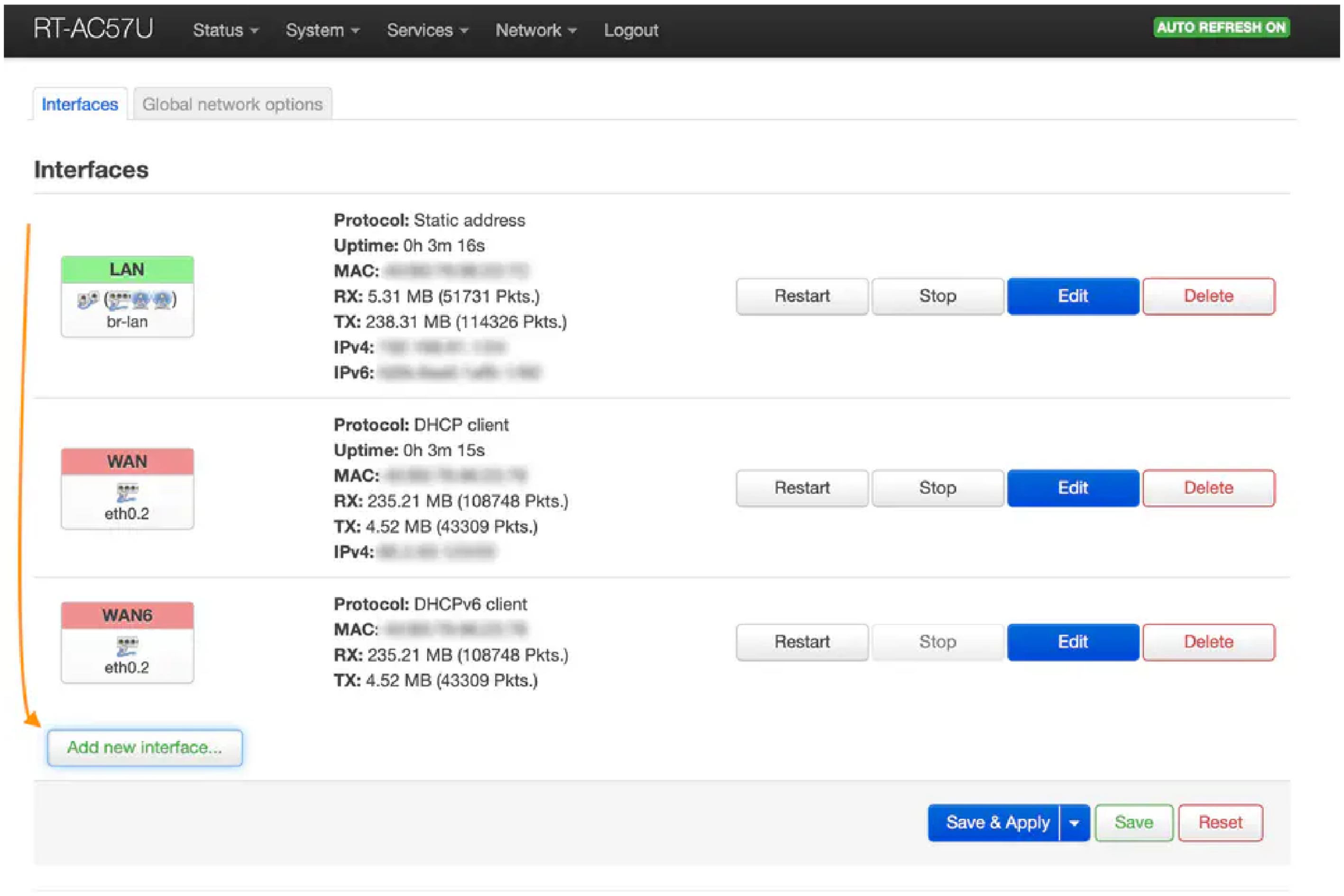
Status

System

Hostname	RT
Model	AS
Architecture	Me
Firmware Version	OpenWrt 19.07.1 r10911-c155900f66 / LuCI openwrt-19.07 branch git-20.063.74467-7295b32
Kernel Version	4.14.167
Local Time	2020-03-06 21:02:51
Uptime	0h 1m 41s
Load Average	0.59, 0.23, 0.09

Memory

Total Available	60.01 MB / 120.87 MB (49%)
Free	77.57 MB / 120.87 MB (64%)
Buffered	2.37 MB / 120.87 MB (1%)



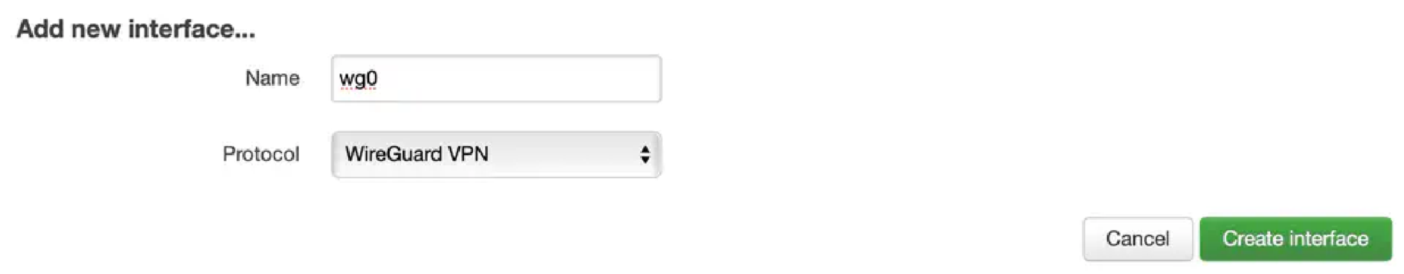
Interfaces

LAN br-lan
Protocol: Static address
Uptime: 0h 3m 16s
MAC: [redacted]
RX: 5.31 MB (51731 Pkts.)
TX: 238.31 MB (114326 Pkts.)
IPv4: [redacted]
IPv6: [redacted]

WAN eth0.2
Protocol: DHCP client
Uptime: 0h 3m 15s
MAC: [redacted]
RX: 235.21 MB (108748 Pkts.)
TX: 4.52 MB (43309 Pkts.)
IPv4: [redacted]

WAN6 eth0.2
Protocol: DHCPv6 client
MAC: [redacted]
RX: 235.21 MB (108748 Pkts.)
TX: 4.52 MB (43309 Pkts.)

We name our interface e.g. **wg0** and as the protocol we choose **WireGuard VPN**. Click on **Create interface**.

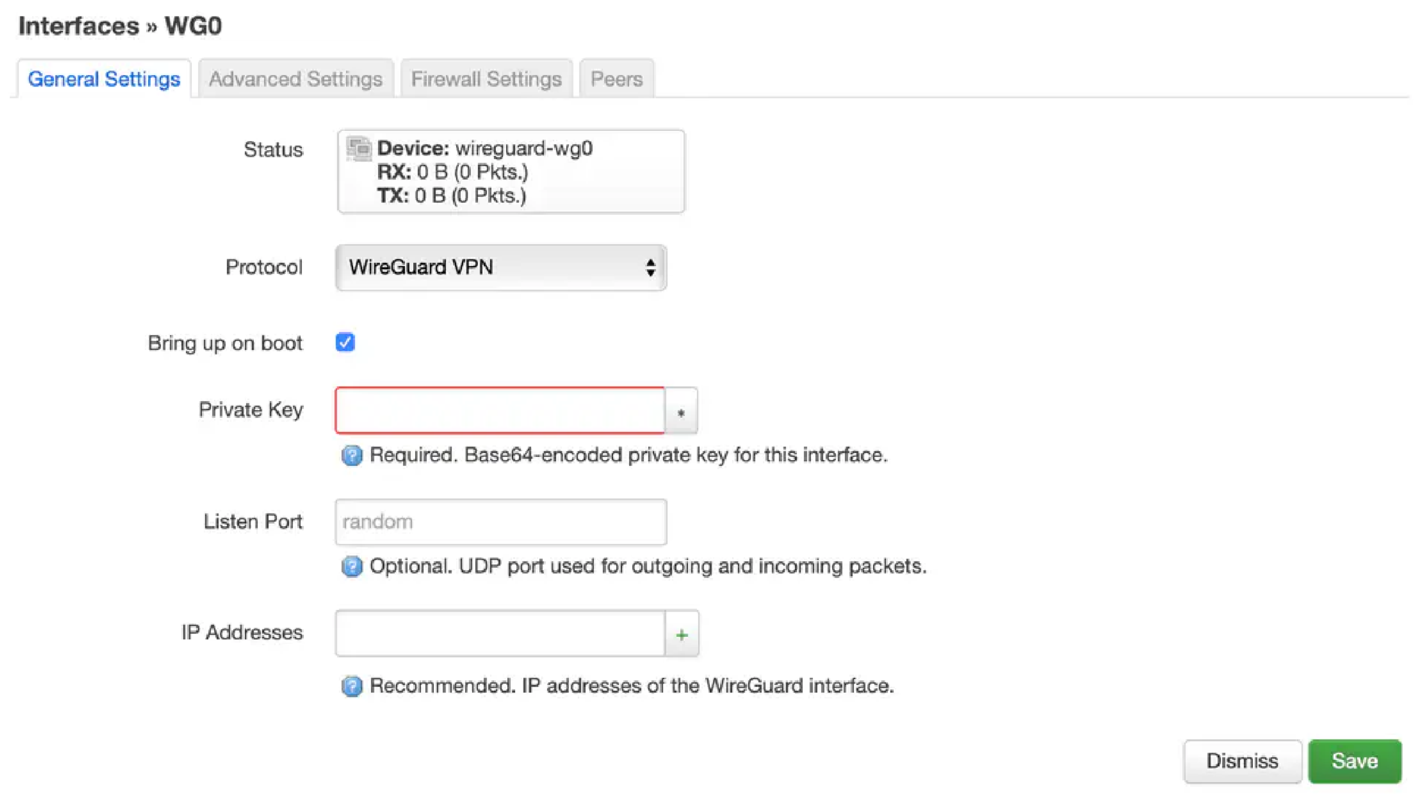

Add new interface...

Name: wg0

Protocol: WireGuard VPN

In the next step we will need some information.

First, we will see a red highlighted field with the missing key.


Interfaces » WG0

Status: **Device:** wireguard-wg0
RX: 0 B (0 Pkts.)
TX: 0 B (0 Pkts.)

Protocol: WireGuard VPN

Bring up on boot: ✓

Private Key:

Required. Base64-encoded private key for this interface.

Listen Port: random

Optional. UDP port used for outgoing and incoming packets.

IP Addresses:

Recommended. IP addresses of the WireGuard interface.

From the terminal we read the **private key** of our server:

Copy

```
tail /etc/wireguard/server-privatekey
```

We copy the key and paste it on the browser side.

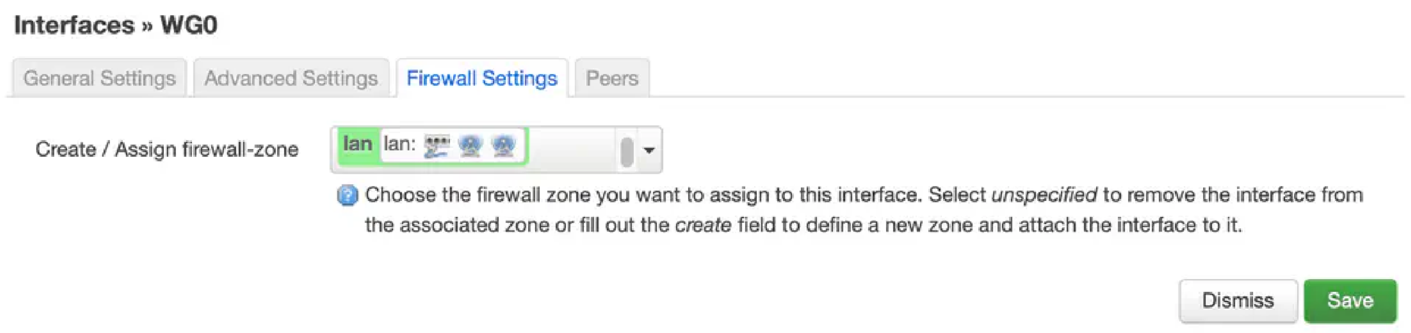
The port (**Listen port**) on which WireGuard will listen is set according to our preferences, e.g. **1234**. Otherwise, the program will select a random port every time our router starts, and this may later have a different effect on the configurations in the devices from which we will connect.

Next, we enter the local IP address of our server. Here we don't have to worry about this address being the same as our local network. At a later stage, we will set it up so that we can connect to local addresses, which by default will be in the range of **192.168.0.x**

In my case I chose to enter **10.0.0.1/24**.

Then go to the **Firewall Settings** tab.

From the **Create / Assign firewall-zone** item we select our **LAN** network, thanks to which we will be able to see and connect to our devices in the local network.



Create / Assign firewall-zone: lan lan:

Choose the firewall zone you want to assign to this interface. Select *unspecified* to remove the interface from the associated zone or fill out the *create* field to define a new zone and attach the interface to it.

In the last tab **Peers** we set our clients that can connect to the server. Here we will need the client key that we generated earlier.

So in the name field, we enter e.g. **client1**.

In the **Public Key** field we paste the client key, which we can read from the terminal using the command:

Copy

```
tail client1-publickey
```

Next, **Allowed IPs** we enter a static address that will be used by our client, which is analogous to the IP address of our server (10.0.0.1/24). In my case it will be **10.0.0.2/32**.

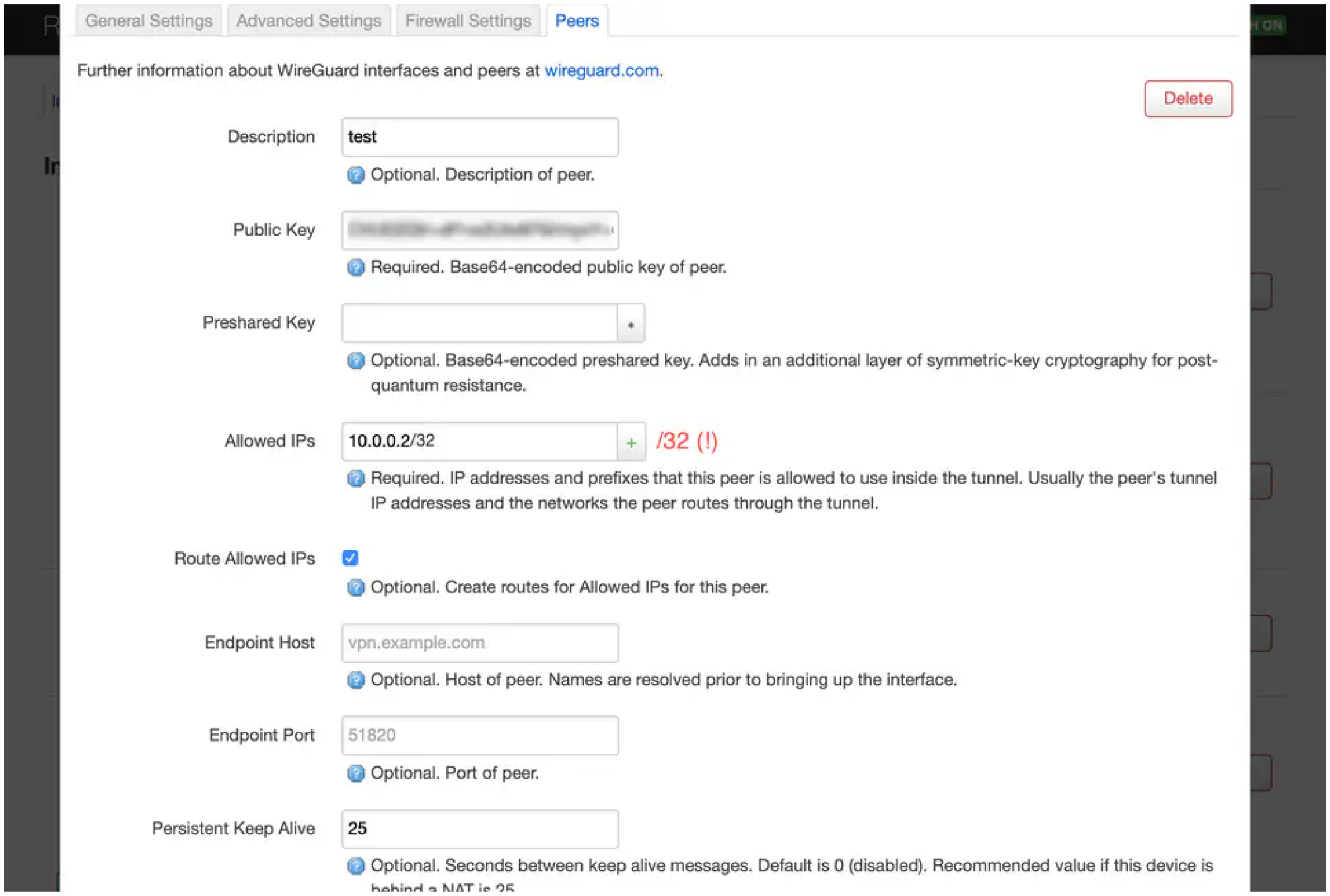
> Notice that on the router side, our client's address is in the /32 mask.

We also check **Route Allowed IPs**.

> If we have more than one internet connection set up using **mwan3** (as described in post: [Adding a second internet connection to a router with OpenWrt]) leave Route

Allowed IPs **NOT checked**.

And in **Persistent Keep Alive** we enter the value **25**.



Further information about WireGuard interfaces and peers at wireguard.com.

Description: test
Optional. Description of peer.

Public Key: [redacted]
Required. Base64-encoded public key of peer.

Preshared Key:
Optional. Base64-encoded preshared key. Adds in an additional layer of symmetric-key cryptography for post-quantum resistance.

Allowed IPs: 10.0.0.2/32 /32 (!)
Required. IP addresses and prefixes that this peer is allowed to use inside the tunnel. Usually the peer's tunnel IP addresses and the networks the peer routes through the tunnel.

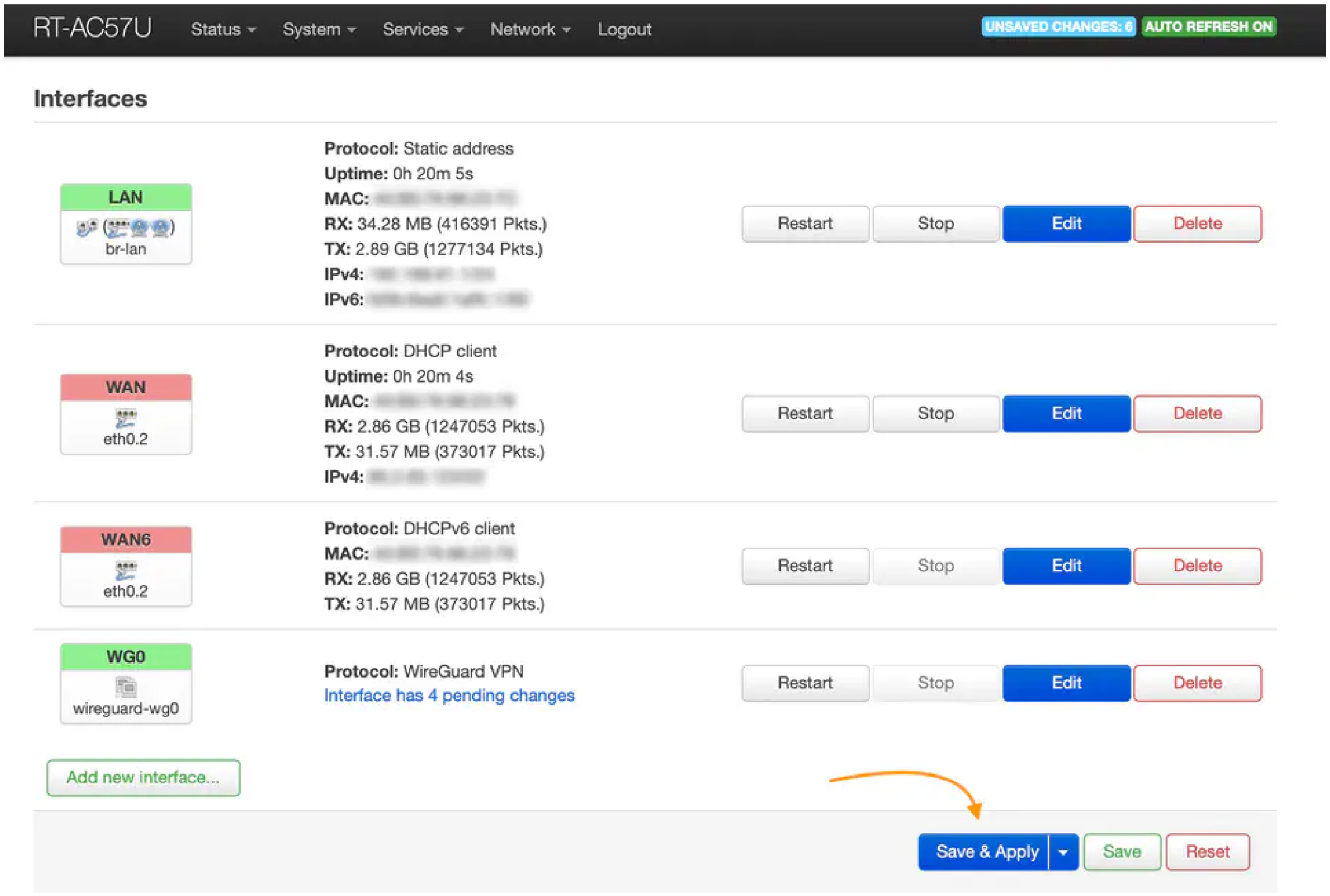
Route Allowed IPs: ✓
Optional. Create routes for Allowed IPs for this peer.

Endpoint Host: vpn.example.com
Optional. Host of peer. Names are resolved prior to bringing up the interface.

Endpoint Port: 51820
Optional. Port of peer.

Persistent Keep Alive: 25
Optional. Seconds between keep alive messages. Default is 0 (disabled). Recommended value if this device is behind a NAT is 25

Select **Save** and then **Save & Apply** again.



Interfaces

LAN
br-lan

Protocol: Static address
Uptime: 0h 20m 5s
MAC: [redacted]
RX: 34.28 MB (416391 Pkts.)
TX: 2.89 GB (1277134 Pkts.)
IPv4: [redacted]
IPv6: [redacted]

WAN
eth0.2

Protocol: DHCP client
Uptime: 0h 20m 4s
MAC: [redacted]
RX: 2.86 GB (1247053 Pkts.)
TX: 31.57 MB (373017 Pkts.)
IPv4: [redacted]

WAN6
eth0.2

Protocol: DHCPv6 client
MAC: [redacted]
RX: 2.86 GB (1247053 Pkts.)
TX: 31.57 MB (373017 Pkts.)

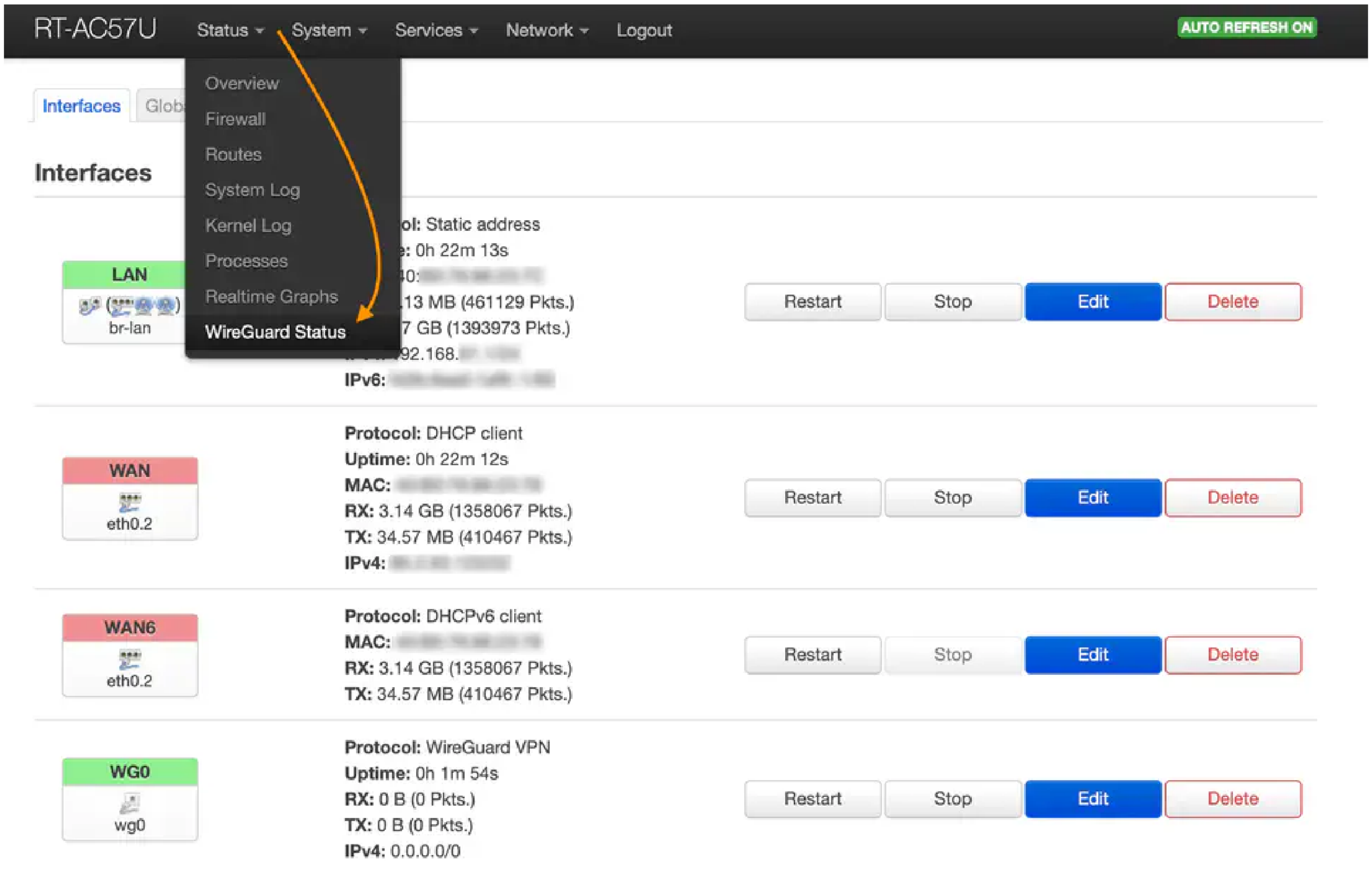
WG0
wireguard-wg0

Protocol: WireGuard VPN
Interface has 4 pending changes

And so we have a WireGuard-based VPN server running.

If at a later stage we will add new clients **(Peers)**, we must also remember to restart the WireGuard server using the **Restart** button.

To check the status of our server, go from the browser to **Status > Wireguard**.



Interfaces

LAN
br-lan

ol: Static address
e: 0h 22m 13s
40:[redacted]
.13 MB (461129 Pkts.)
7 GB (1393973 Pkts.)
92.168.[redacted]
IPv6: [redacted]

WAN
eth0.2

Protocol: DHCP client
Uptime: 0h 22m 12s
MAC: [redacted]
RX: 3.14 GB (1358067 Pkts.)
TX: 34.57 MB (410467 Pkts.)
IPv4: [redacted]

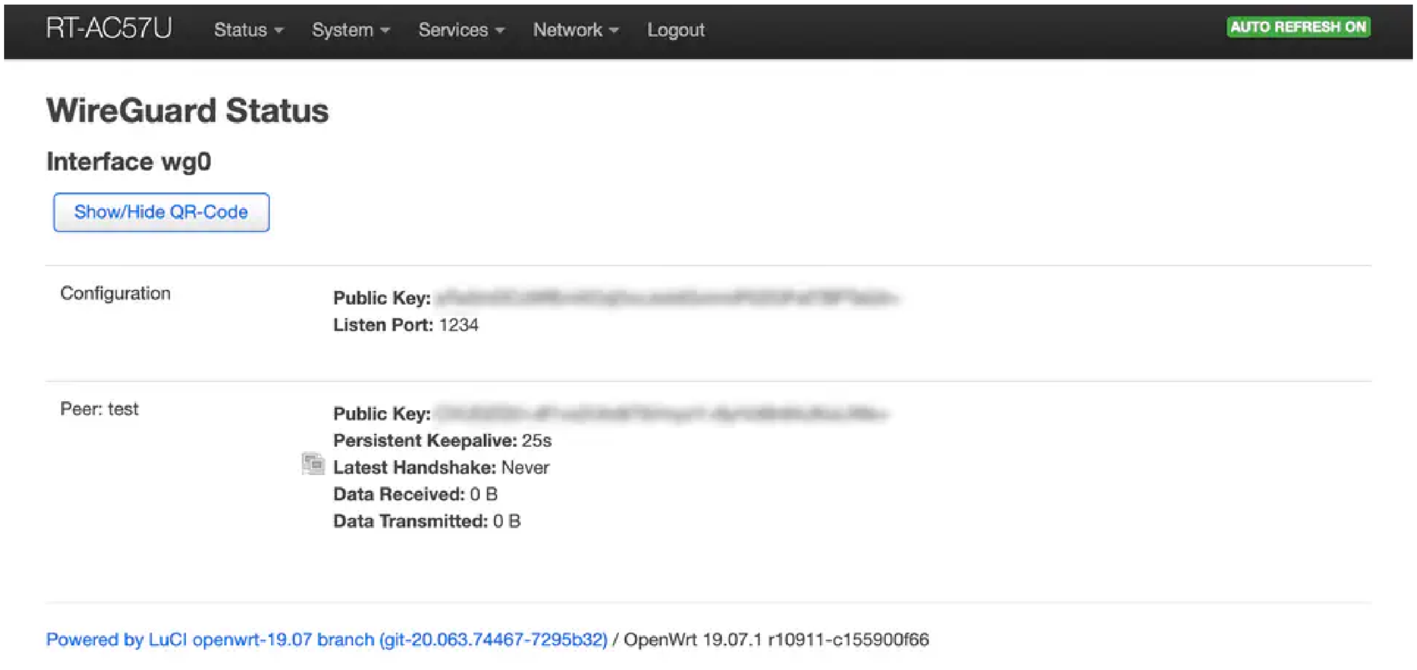
WAN6
eth0.2

Protocol: DHCPv6 client
MAC: [redacted]
RX: 3.14 GB (1358067 Pkts.)
TX: 34.57 MB (410467 Pkts.)

WG0
wg0

Protocol: WireGuard VPN
Uptime: 0h 1m 54s
RX: 0 B (0 Pkts.)
TX: 0 B (0 Pkts.)
IPv4: 0.0.0.0/0

On this page we will see our **public key**, which we will need in the next step, and the **port (Listen Port)** on which the service is running.



WireGuard Status

Interface wg0

Configuration

Public Key: [redacted]
Listen Port: 1234

Peer: test

Public Key: [redacted]
Persistent Keepalive: 25s
Latest Handshake: Never
Data Received: 0 B
Data Transmitted: 0 B

Powered by LuCI openwrt-19.07 branch (git-20.063.74467-7295b32) / OpenWrt 19.07.1 r10911-c155900f66

As you can see here, our router supports configuration exchange (client setup) using QR codes. If you want to play with this, you need to install a package on your router:

Copy

```
opkg install qrencode
```

From what I've noticed, the QR code doesn't pass on all the necessary data to the configuration, so we have to change it a bit anyway, so in this case I left it alone.

In order to connect to our router via port **1234/udp**, we need to allow it from the **firewall** level. To do this, we add the following commands from the terminal:

Copy

```
uci add firewall rule
uci set firewall.@rule[-1].src="*"
uci set firewall.@rule[-1].target="ACCEPT"
uci set firewall.@rule[-1].proto="udp"
uci set firewall.@rule[-1].dest_port="1234"
uci set firewall.@rule[-1].name="Allow-Wireguard-Inbound"
uci commit firewall
```

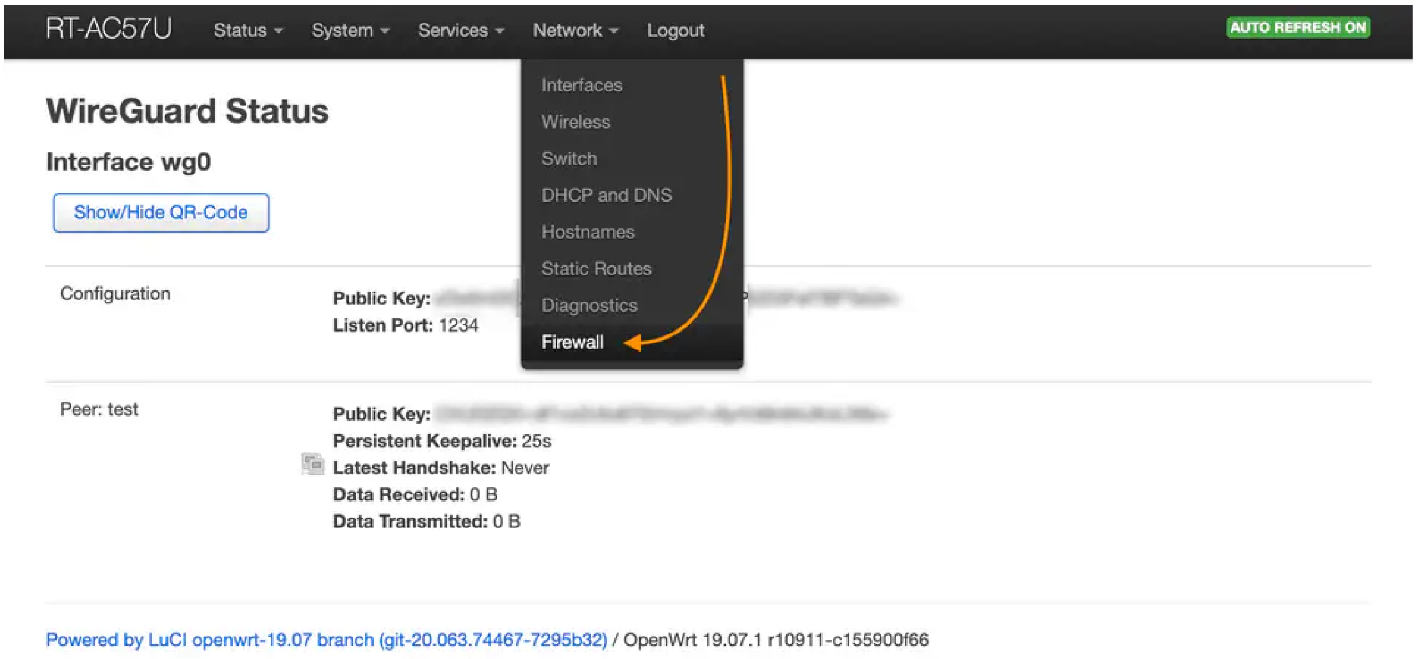
Of course, port 1234 is as we selected earlier.

Then we restart the **firewall**

Copy

```
/etc/init.d/firewall restart
```

We can also do this from the browser, then go to **Network > Firewall**



WireGuard Status

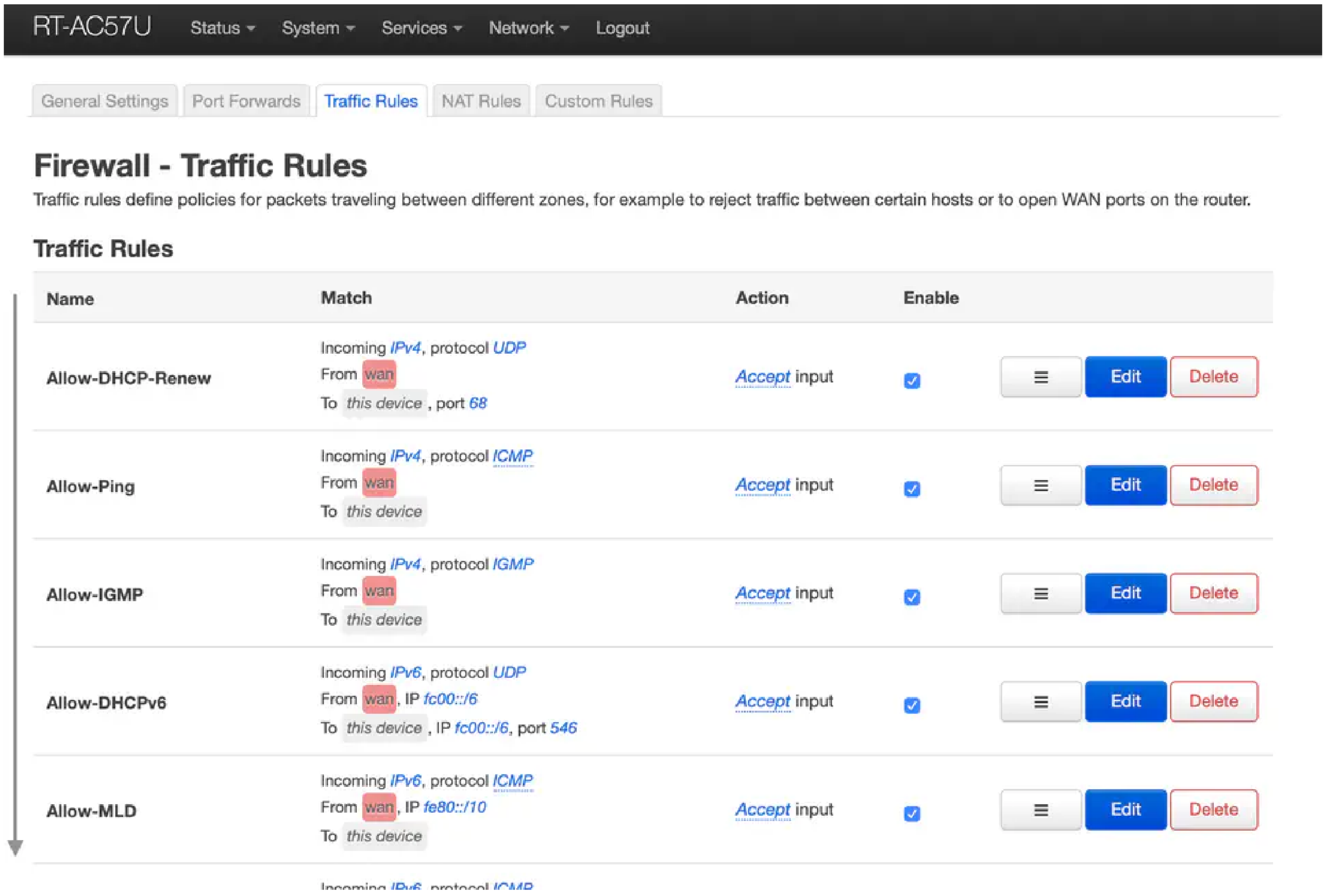
Interface wg0

Configuration
Public Key: [redacted]
Listen Port: 1234

Peer: test
Public Key: [redacted]
Persistent Keepalive: 25s
Latest Handshake: Never
Data Received: 0 B
Data Transmitted: 0 B

Powered by LuCI openwrt-19.07 branch (git-20.063.74467-7295b32) / OpenWrt 19.07.1 r10911-c155900f66

In the **Traffic Rules** tab, we add a new rule using the **Add** button at the bottom of the page. We add our rule similarly to the screenshots below.

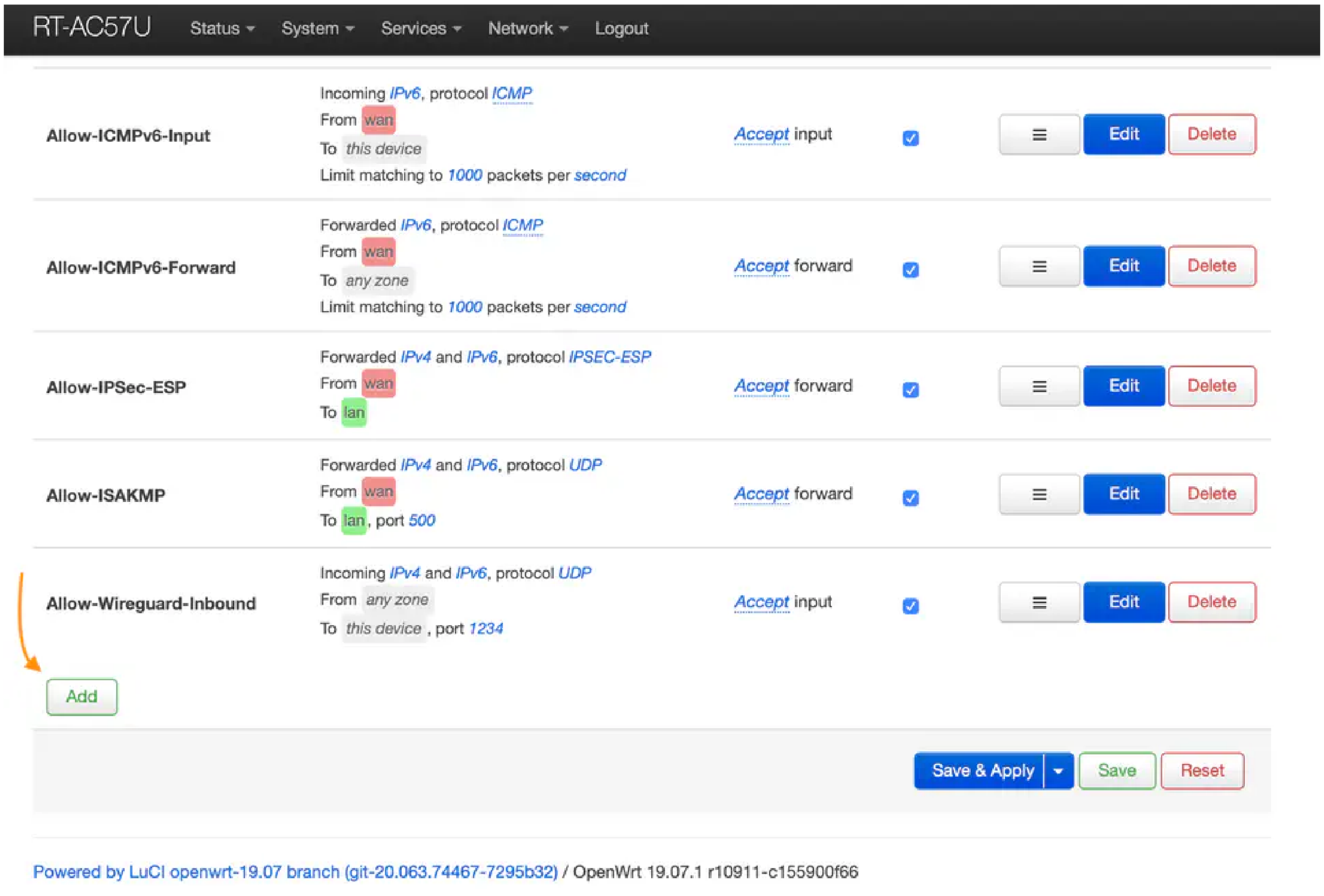


Firewall - Traffic Rules

Traffic rules define policies for packets traveling between different zones, for example to reject traffic between certain hosts or to open WAN ports on the router.

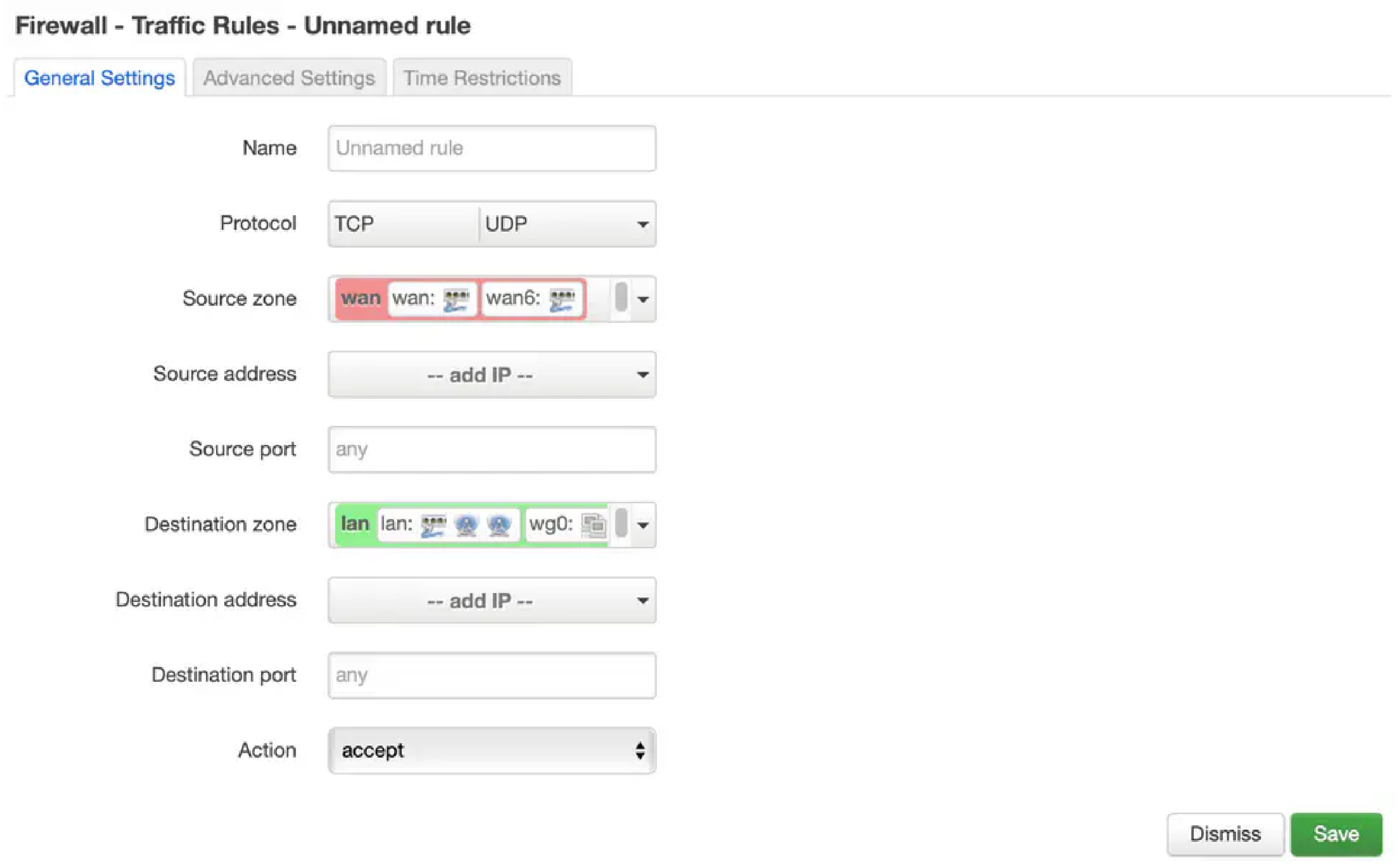
Traffic Rules

Name	Match	Action	Enable
Allow-DHCP-Renew	Incoming *IPv4*, protocol *UDP* From wan To *this device* , port *68*	*Accept* input	✓
Allow-Ping	Incoming *IPv4*, protocol *ICMP* From wan To *this device*	*Accept* input	✓
Allow-IGMP	Incoming *IPv4*, protocol *IGMP* From wan To *this device*	*Accept* input	✓
Allow-DHCPv6	Incoming *IPv6*, protocol *UDP* From wan, IP *fc00::/6* To *this device* , IP *fc00::/6*, port *546*	*Accept* input	✓
Allow-MLD	Incoming *IPv6*, protocol *ICMP* From wan, IP *fe80::/10* To *this device*	*Accept* input	✓



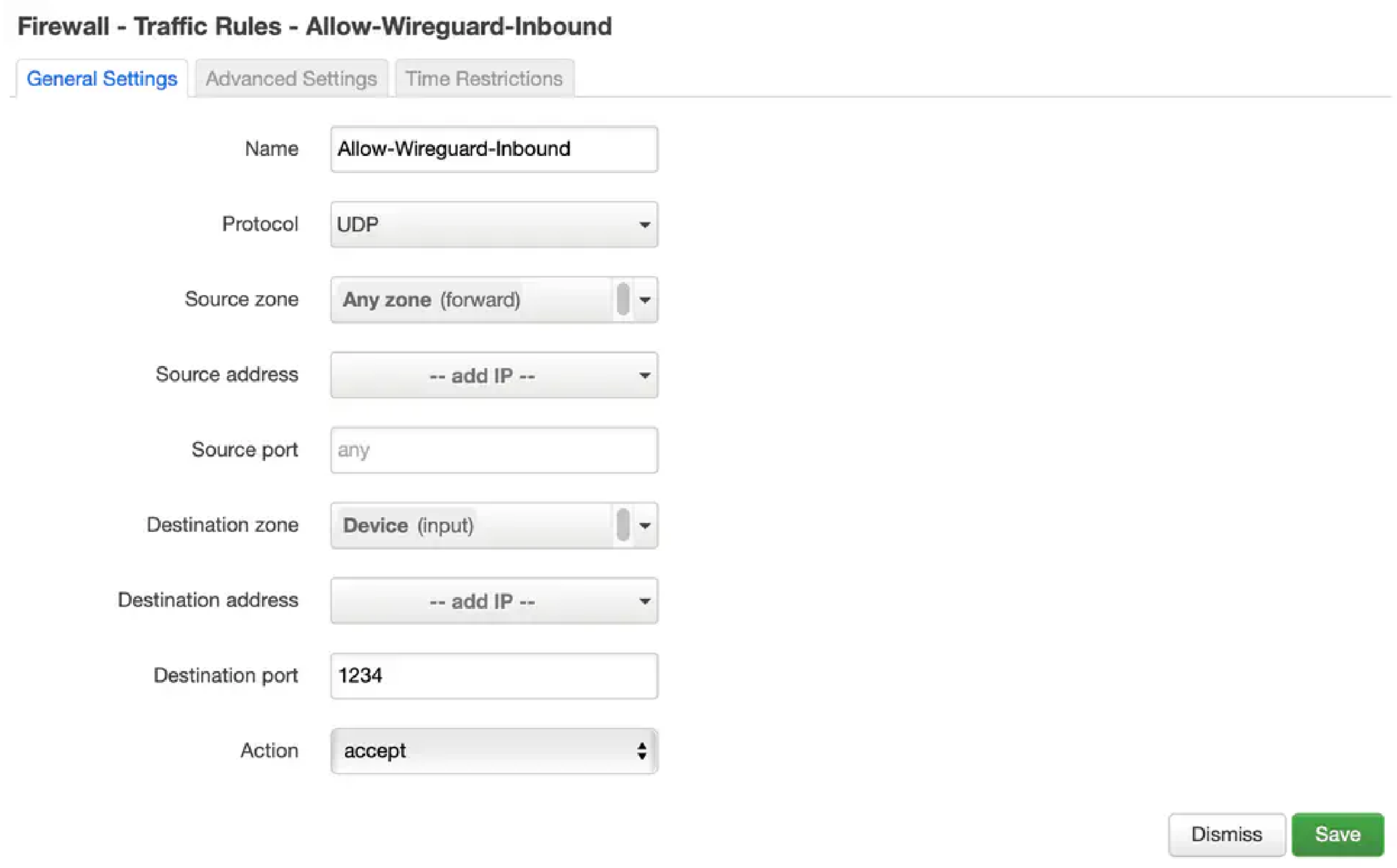
Name	Match	Action	Enable
Allow-ICMPv6-Input	Incoming *IPv6*, protocol *ICMP* From wan To *this device* Limit matching to *1000* packets per *second*	*Accept* input	✓
Allow-ICMPv6-Forward	Forwarded *IPv6*, protocol *ICMP* From wan To *any zone* Limit matching to *1000* packets per *second*	*Accept* forward	✓
Allow-IPSec-ESP	Forwarded *IPv4* and *IPv6*, protocol *IPSEC-ESP* From wan To lan	*Accept* forward	✓
Allow-ISAKMP	Forwarded *IPv4* and *IPv6*, protocol *UDP* From wan To lan, port *500*	*Accept* forward	✓
Allow-Wireguard-Inbound	Incoming *IPv4* and *IPv6*, protocol *UDP* From *any zone* To *this device* , port *1234*	*Accept* input	✓

Powered by LuCI openwrt-19.07 branch (git-20.063.74467-7295b32) / OpenWrt 19.07.1 r10911-c155900f66



Firewall - Traffic Rules - Unnamed rule

Name	Unnamed rule
Protocol	TCP UDP
Source zone	wan wan: wan6:
Source address	-- add IP --
Source port	any
Destination zone	lan lan: wg0:
Destination address	-- add IP --
Destination port	any
Action	accept



Firewall - Traffic Rules - Allow-Wireguard-Inbound

Name	Allow-Wireguard-Inbound
Protocol	UDP
Source zone	Any zone (forward)
Source address	-- add IP --
Source port	any
Destination zone	Device (input)
Destination address	-- add IP --
Destination port	1234
Action	accept

If we want to add IPv6 support on the server and client side, in addition to the server address **10.0.0.1/24** we also add **fd86::1/64** at the appropriate stage of the configuration.

Similarly, for client **(Peer)** we add at **10.0.0.2/32** also **fd86::2/128**.

> Notice that on the router side, our client's address is in the mask /32 (for IPv4) and /128 (for IPv6).

WireGuard and OpenWrt (Client)

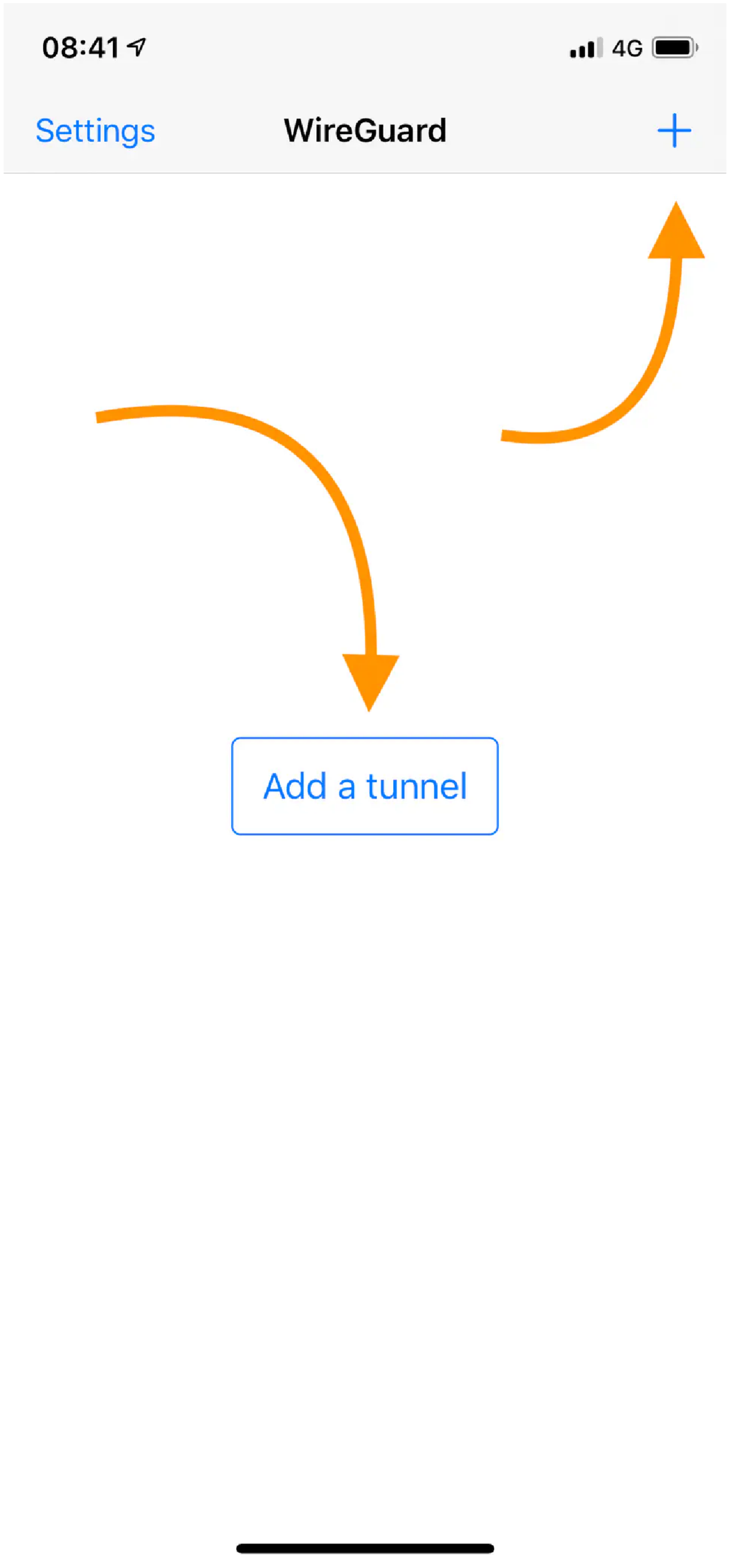
It's time to set up the client **(peer)**, i.e. the device that will connect to our server.

I will describe it using the iPhone (iOS) app as an example. In other operating systems the setting is analogous.

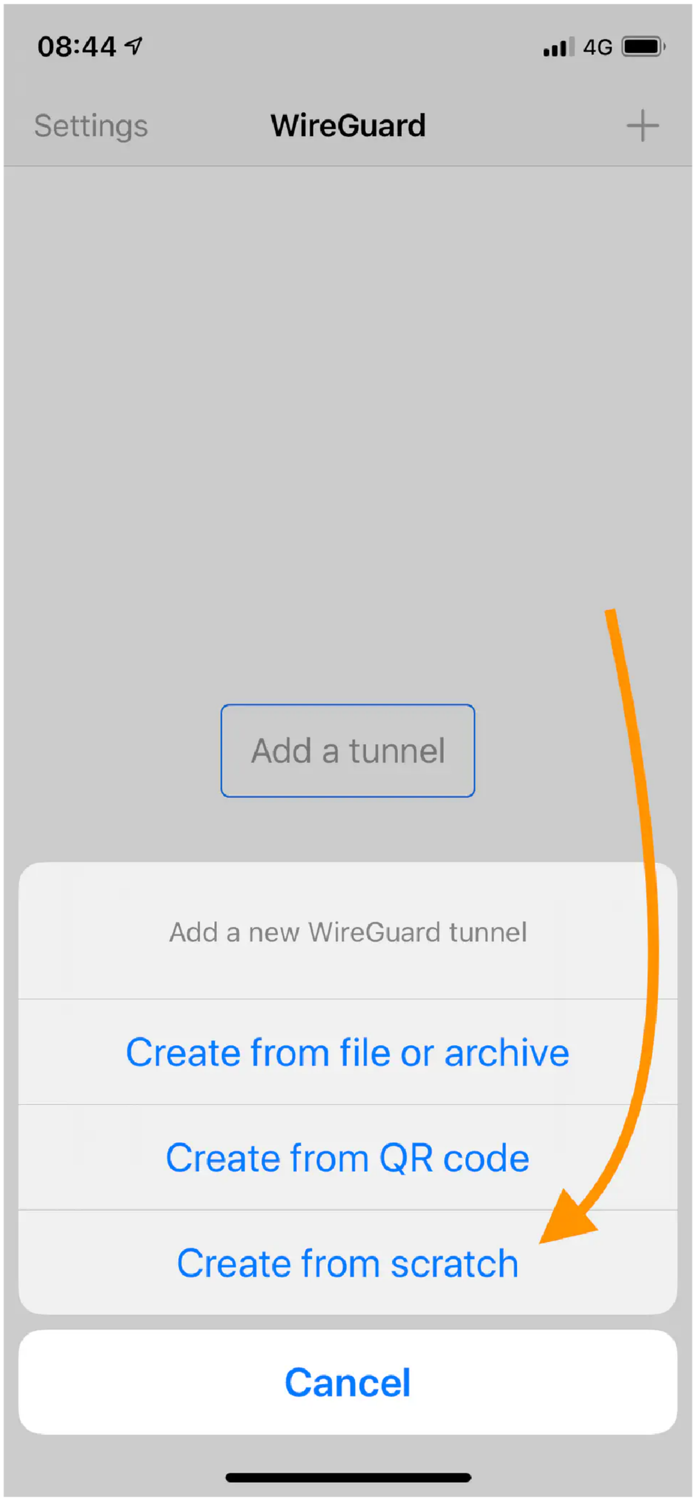
Download the **WireGuard** application from **AppStore** ↗.

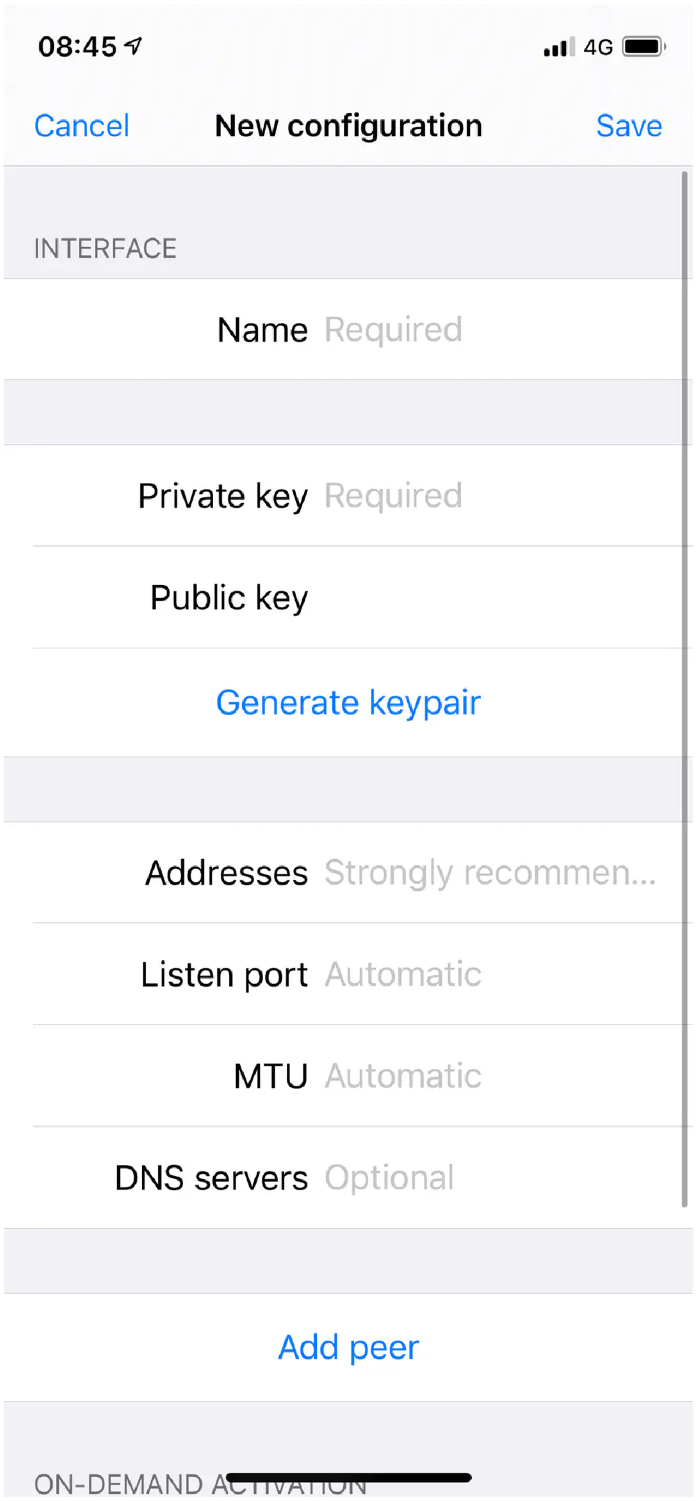
Similarly, when it comes to **Play Store** ↗.

After starting the application, click on the **Add a tunnel** button.



Since we are starting from scratch, we choose **Create from scratch**





And so we begin entering our configuration, starting with:

Name: test

Private key: {paste}

This is our *client1-privatekey* generated at the beginning:

Copy

```
tail client1-privatekey
```

As you will notice, the **Public key** field will be automatically filled in. We can verify this with what we generated at the beginning:

Copy

```
tail client1-publickey
```

Addresses: 10.0.0.2/24

Here we enter the client IP address, which we also set on our server side. If we added IPv6 support, we enter **10.0.0.2/24, fd86::2/64**

DNS servers: 192.168.1.1

Where 192.168.1.1 is the local IP address of our router.

> If you want your device to send DNS requests from your internet connection rather than through your router, you can skip this option.

So we've added the client keys. Now we need to add information about the server we'll be connecting to.

Next, click on **Add peer** to add information about our server.

In the **public key** field we enter the key of our server, which we can read from the terminal:

Copy

```
tail /etc/wireguard/server-publickey
```

Endpoint: [externalIP]:[port]

If our router has a permanent, external IP address, we enter it here and specify the port (previously set) through which the connection will take place. For example: 123.34.45.56:1234, where 123.34.45.56 is the IP address, 1234 is our port.

In the **Allowed IPs** field: 0.0.0.0/0

This will allow us to see devices on the local network.

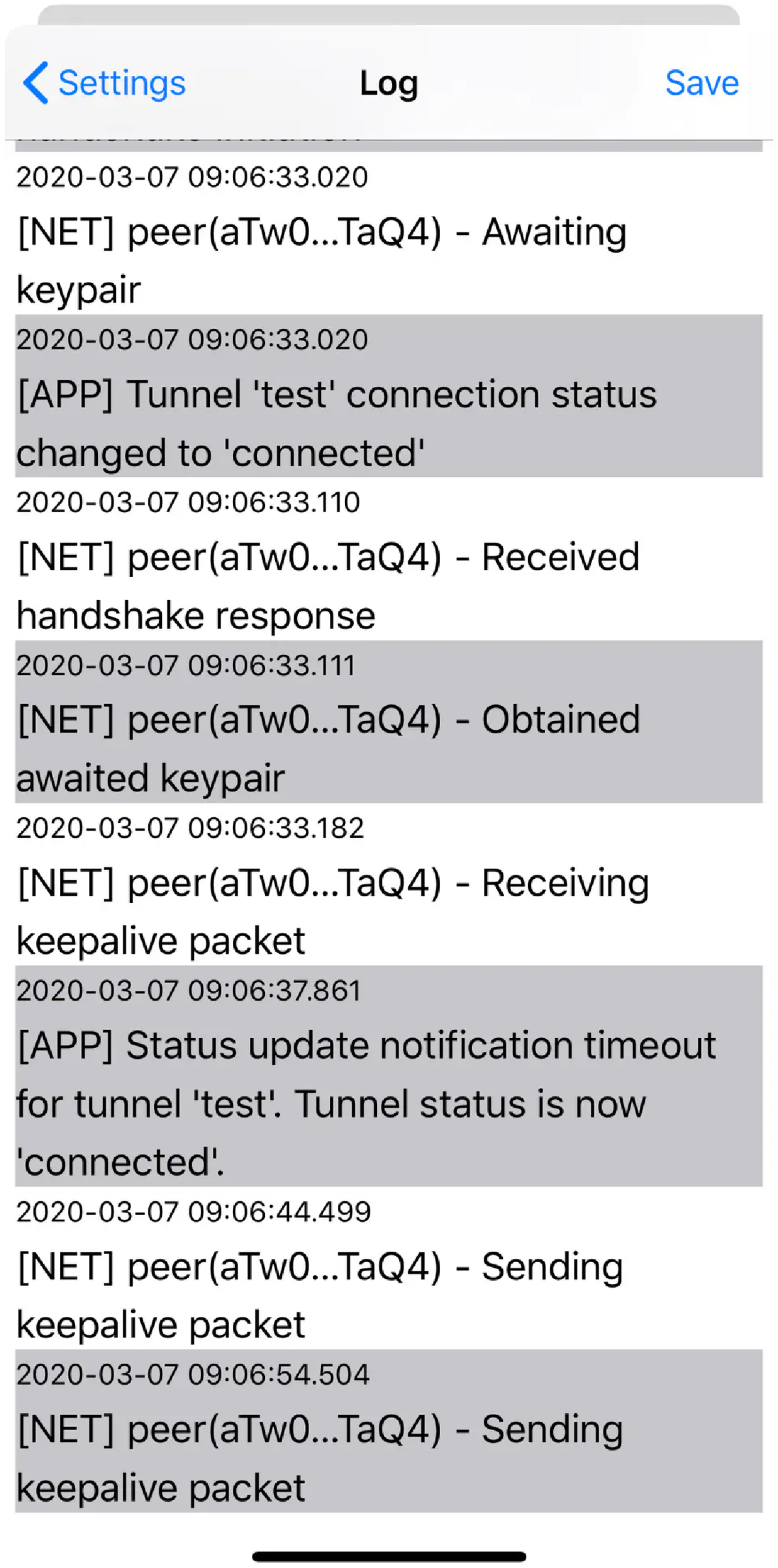
If we added IPv6 support, we enter **0.0.0.0/0, ::/0**

Persistent keepalive: 25

Then we **save** our configuration.

At this stage, our phone will ask if it is possible to add the VPN configuration to the settings, which we of course allow.

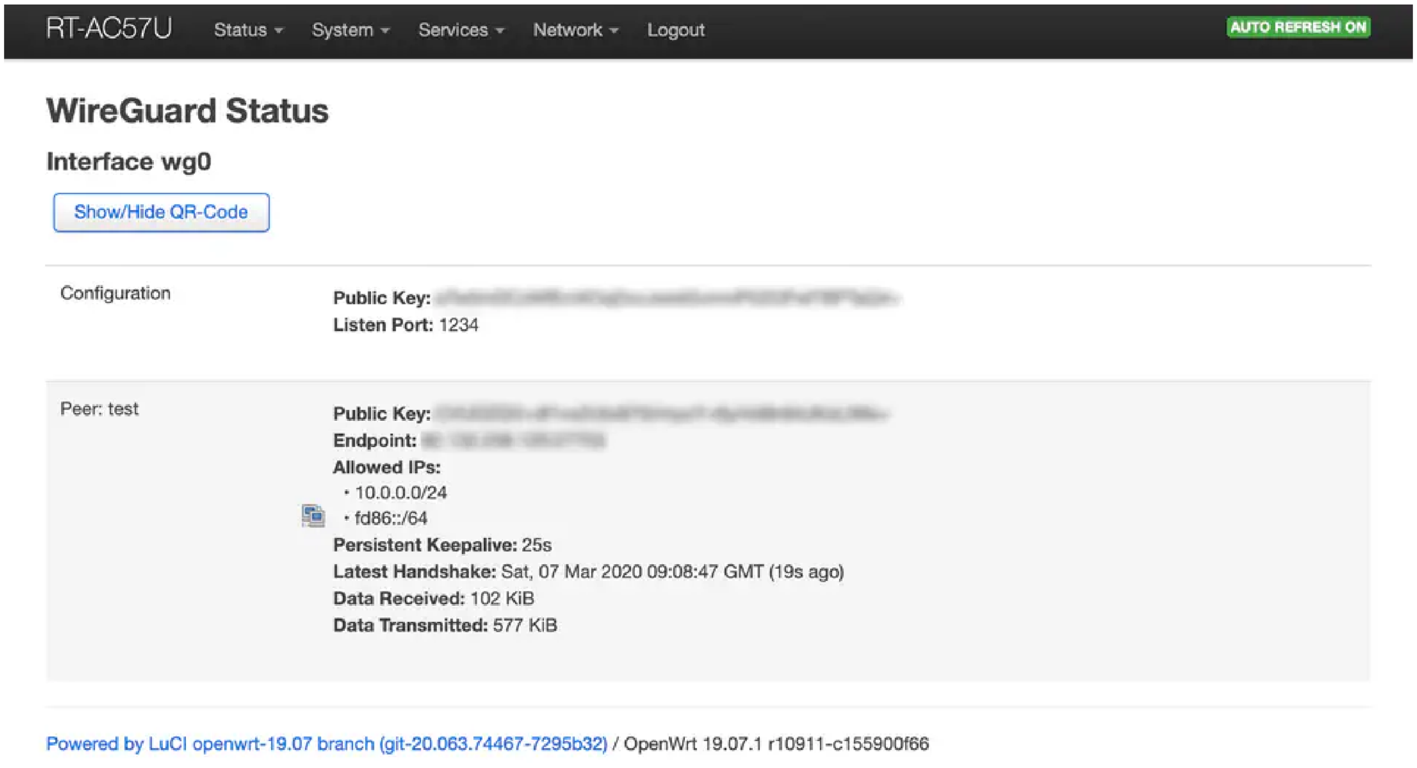
If we did everything correctly, all that’s left is to make the connection and everything should work. To check this, go to **Settings** of the WireGuard program on our device and click on the **View log** option, where we should see if everything works.



```
2020-03-07 09:06:33.020
[NET] peer(aTw0...TaQ4) - Awaiting keypair
2020-03-07 09:06:33.020
[APP] Tunnel 'test' connection status changed to 'connected'
2020-03-07 09:06:33.110
[NET] peer(aTw0...TaQ4) - Received handshake response
2020-03-07 09:06:33.111
[NET] peer(aTw0...TaQ4) - Obtained awaited keypair
2020-03-07 09:06:33.182
[NET] peer(aTw0...TaQ4) - Receiving keepalive packet
2020-03-07 09:06:37.861
[APP] Status update notification timeout for tunnel 'test'. Tunnel status is now 'connected'.
2020-03-07 09:06:44.499
[NET] peer(aTw0...TaQ4) - Sending keepalive packet
2020-03-07 09:06:54.504
[NET] peer(aTw0...TaQ4) - Sending keepalive packet
```

We also open our browser and enter **myip** in Google and check if it returns the same as when we are connected to our local network.

On our router side, from the browser level we can also see in **WireGuard Status** that our connection is working and data is traveling between devices.



WireGuard Status

Interface wg0

Configuration

Public Key: [redacted]
Listen Port: 1234

Peer: test

Public Key: [redacted]
Endpoint: [redacted]
Allowed IPs:
- 10.0.0.0/24
- fd86::/64

Persistent Keepalive: 25s
Latest Handshake: Sat, 07 Mar 2020 09:08:47 GMT (19s ago)
Data Received: 102 KiB
Data Transmitted: 577 KiB

Powered by LuCI openwrt-19.07 branch (git-20.063.74467-7295b32) / OpenWrt 19.07.1 r10911-c155900f66

If something is not working, we need to check whether our **Firewall** on the router is configured correctly (see above) and whether we have entered the correct keys everywhere.

In order not to have to enter everything from scratch every time we set up our device to connect to WireGuard, we can save our configuration to a file from the **Settings** level. When reconfiguring, we can import these settings.

Adding additional **clients (Peers)** is done analogously, by generating keys:

Copy

```
wg genkey | tee client2-privatekey | wg pubkey > client2-publickey
```

Then, in the **wg0** interface settings we add another **Peer**.

Let’s not forget to save our settings and reset our interface.

And that would be all.

Before we go any further, it is a good idea to save **server-privatekey** and **server-publickey** somewhere safe, along with the client keys **client1-privatekey** and **client1-publickey**.

Adding another client (peer)

Although adding another client is analogous, I have received questions on how to do it, which is somewhat understandable. A small error in keys and IP addresses can cause one client (peer) to work and another not.

When adding more clients, remember that the private key (Private Key) and public key (Public Key) for the server have already been generated and set in our virtual interface (**wg0**), so **we don't have to generate it again**. The entire method involves generating keys for a new user, adding them to the WireGuard interface (wg0) on the router side, and configuring the client software accordingly.

Here's how, with the first client (peer) set up, I added another one.

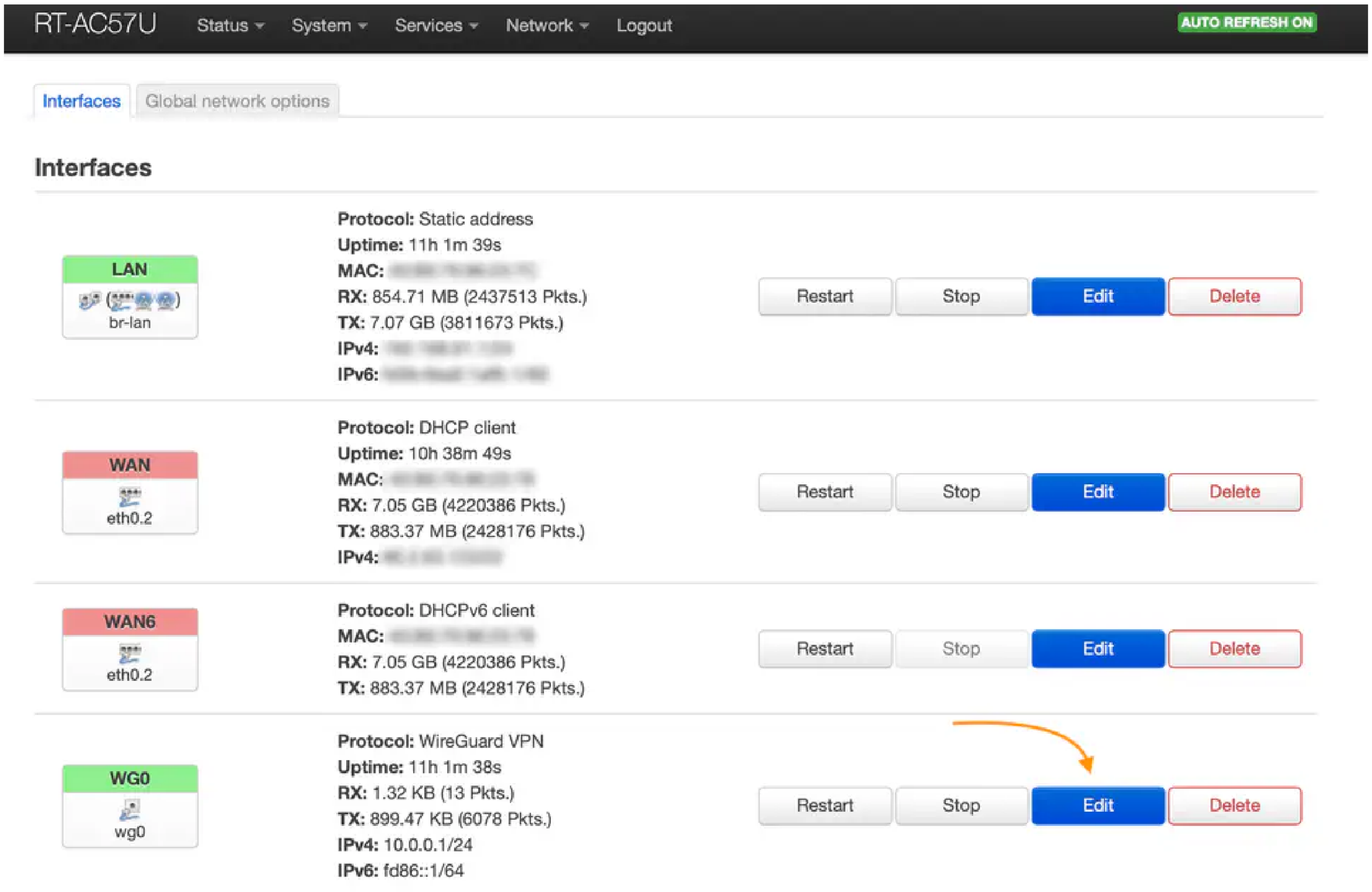
We start by generating keys for the new client from the SSH level of our router.

Copy

```
wg genkey | tee client2-privatekey | wg pubkey > client2-publickey
```

We add the client (peer) and keys to the router configuration.

We go to **Network > Interfaces** and edit our **wg0** interface.



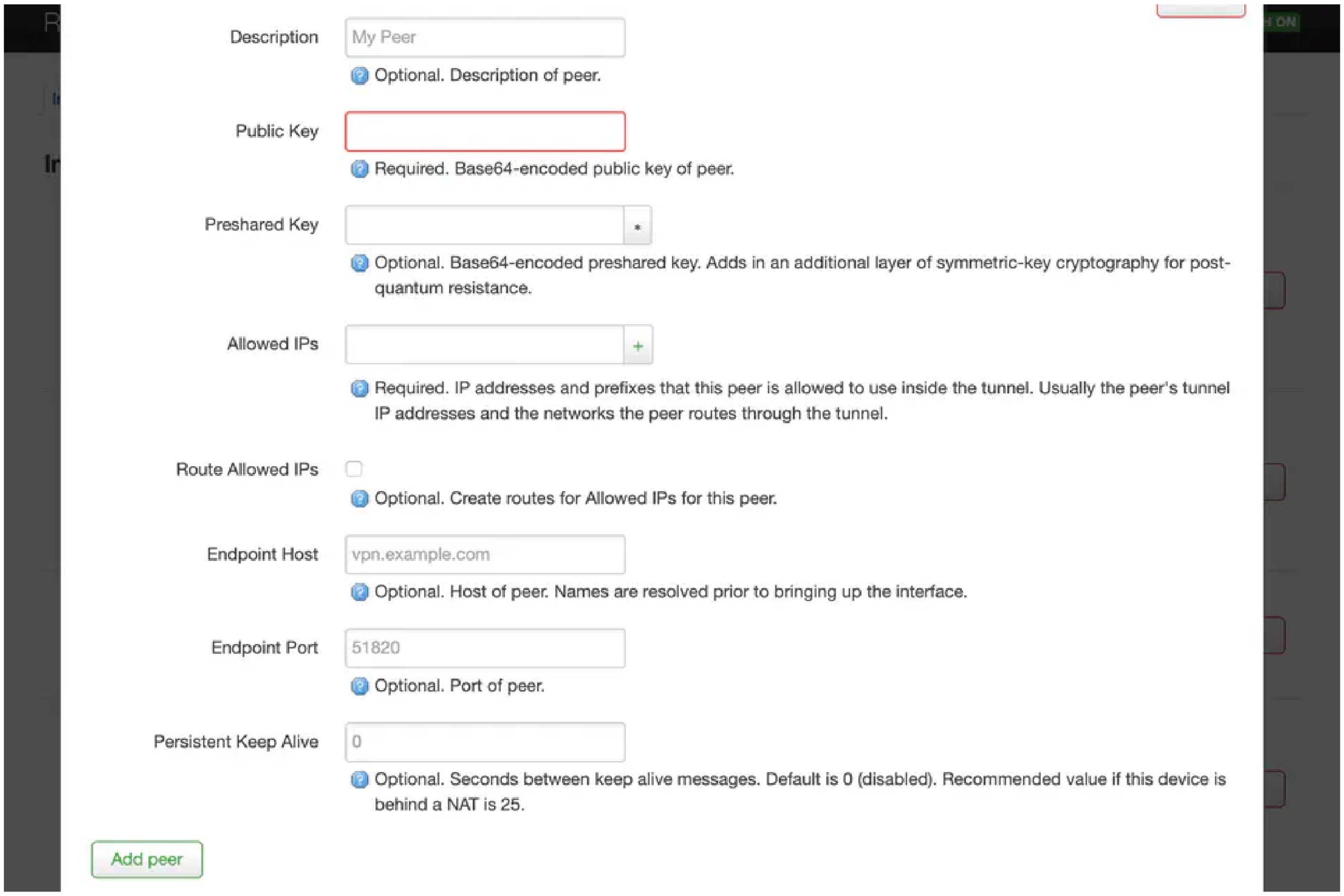
Protocol: Static address
Uptime: 11h 1m 39s
MAC:
RX: 854.71 MB (2437513 Pkts.)
TX: 7.07 GB (3811673 Pkts.)
IPv4:
IPv6:

Protocol: DHCP client
Uptime: 10h 38m 49s
MAC:
RX: 7.05 GB (4220386 Pkts.)
TX: 883.37 MB (2428176 Pkts.)
IPv4:

Protocol: DHCPv6 client
MAC:
RX: 7.05 GB (4220386 Pkts.)
TX: 883.37 MB (2428176 Pkts.)

Protocol: WireGuard VPN
Uptime: 11h 1m 38s
RX: 1.32 KB (13 Pkts.)
TX: 899.47 KB (6078 Pkts.)
IPv4: 10.0.0.1/24
IPv6: fd86::1/64

In the next step, we go to the **Peers** tab. Since the first client is already set up, we scroll down the page and click on the **Add peer** button.



Description: My Peer
Optional. Description of peer.

Public Key:
Required. Base64-encoded public key of peer.

Preshared Key:
Optional. Base64-encoded preshared key. Adds in an additional layer of symmetric-key cryptography for post-quantum resistance.

Allowed IPs:
Required. IP addresses and prefixes that this peer is allowed to use inside the tunnel. Usually the peer's tunnel IP addresses and the networks the peer routes through the tunnel.

Route Allowed IPs: ☐
Optional. Create routes for Allowed IPs for this peer.

Endpoint Host: vpn.example.com
Optional. Host of peer. Names are resolved prior to bringing up the interface.

Endpoint Port: 51820
Optional. Port of peer.

Persistent Keep Alive: 0
Optional. Seconds between keep alive messages. Default is 0 (disabled). Recommended value if this device is behind a NAT is 25.

Similarly as before, we set **Description** (client2) and in the **Public Key** field we paste the newly generated key, which we read with the command:

Copy

```
tail client2-publickey
```

In the **Allowed IPs** field enter the next free IP address:

Copy

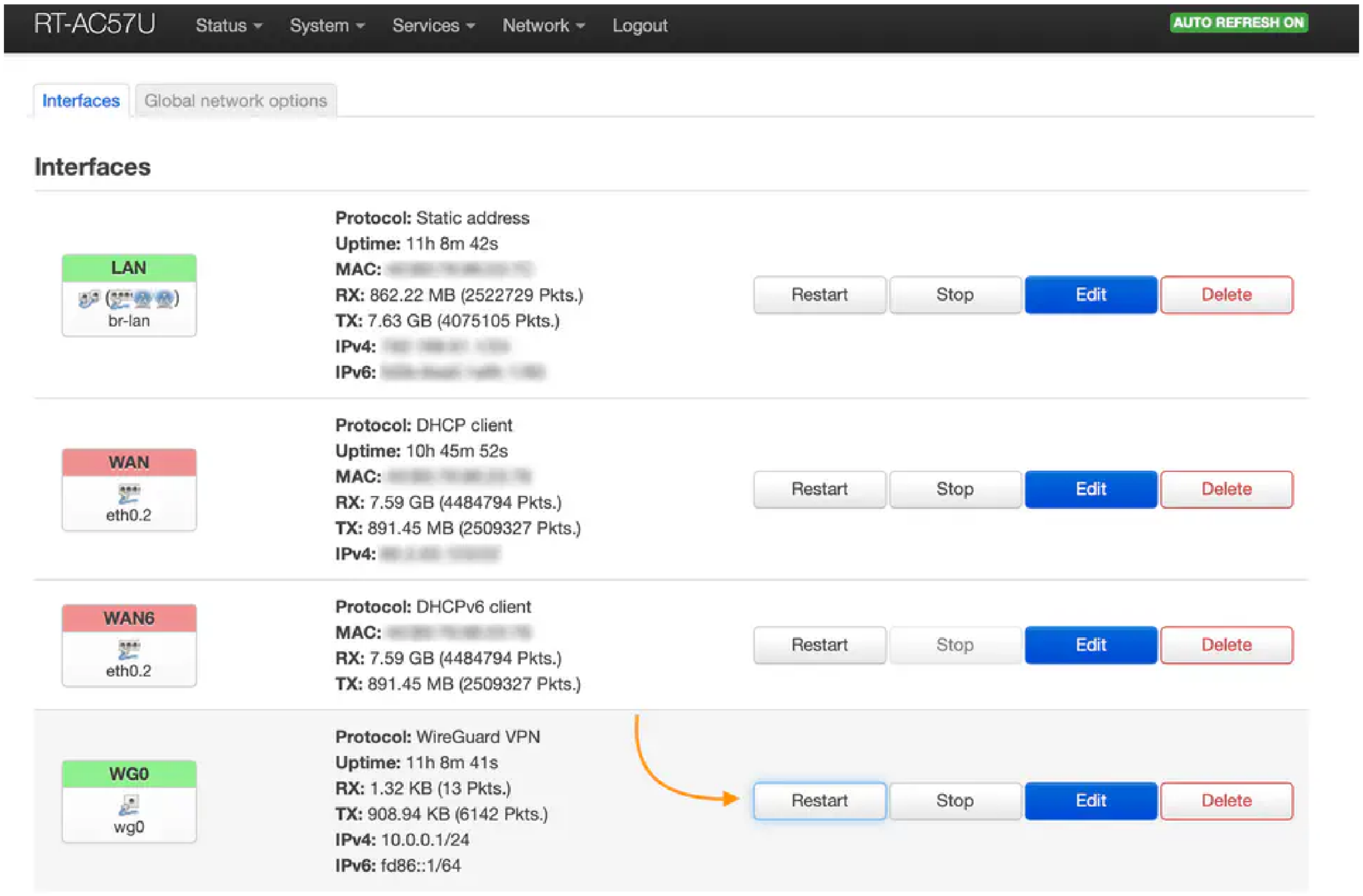
```
10.0.0.3/32
fd86::3/128 (if we also add IPv6)
```

> Notice that on the router side, our client's address is in the mask /32 (for IPv4) and /128 (for IPv6).

Let's check **Route Allowed IPs** (if we only have one Internet connection) and enter the value **25** in **Persistent Keep Alive**.

Then we **save** our settings and at the next stage we confirm the changes made to the configuration by clicking on **Save & Apply**.

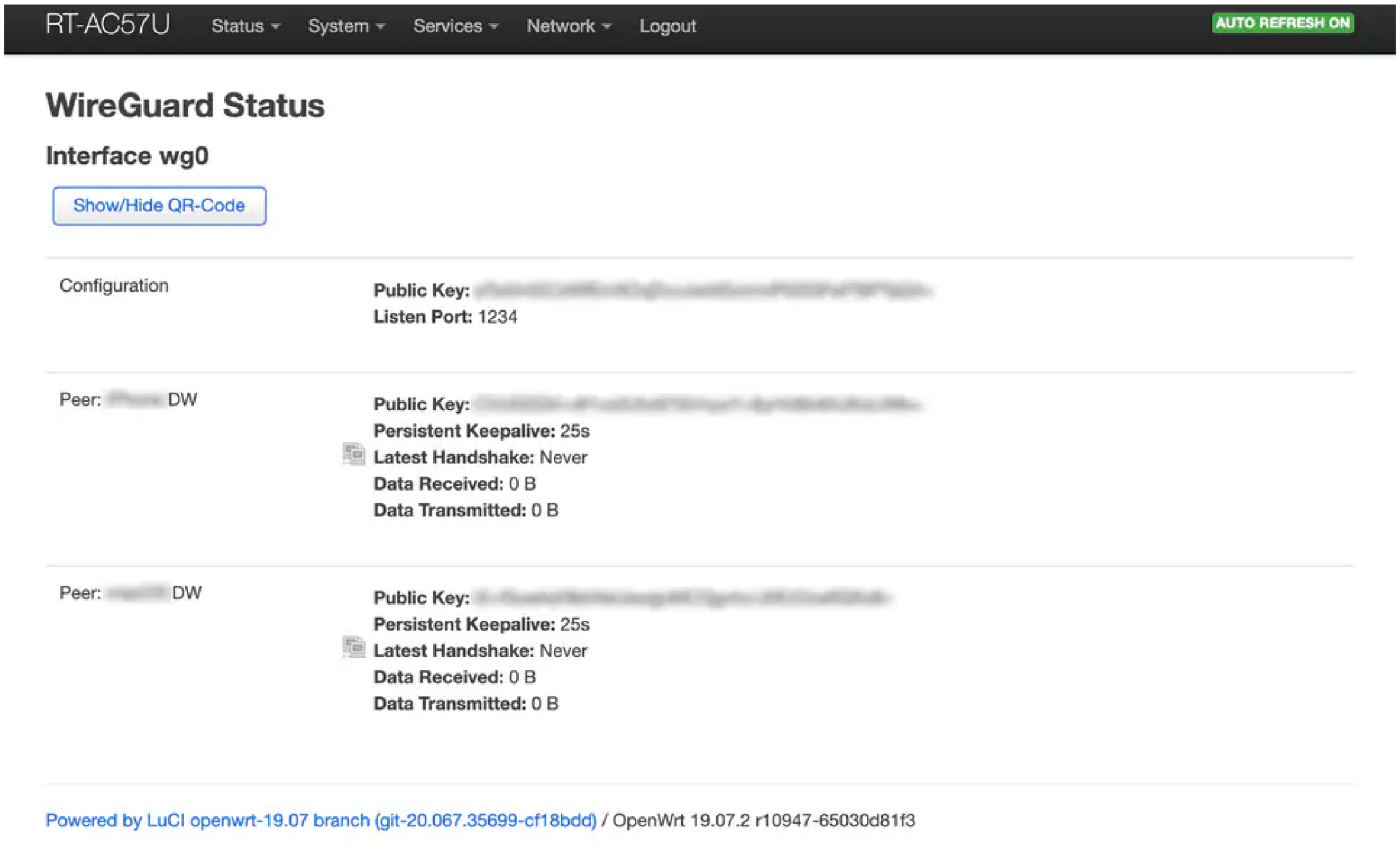
Now we need to restart the **wg0** interface using the **Restart** button.



Interfaces

LAN br-lan	**Protocol:** Static address **Uptime:** 11h 8m 42s **MAC:** [redacted] **RX:** 862.22 MB (2522729 Pkts.) **TX:** 7.63 GB (4075105 Pkts.) **IPv4:** [redacted] **IPv6:** [redacted]	Restart Stop Edit Delete
WAN eth0.2	**Protocol:** DHCP client **Uptime:** 10h 45m 52s **MAC:** [redacted] **RX:** 7.59 GB (4484794 Pkts.) **TX:** 891.45 MB (2509327 Pkts.) **IPv4:** [redacted]	Restart Stop Edit Delete
WAN6 eth0.2	**Protocol:** DHCPv6 client **MAC:** [redacted] **RX:** 7.59 GB (4484794 Pkts.) **TX:** 891.45 MB (2509327 Pkts.)	Restart Stop Edit Delete
WG0 wg0	**Protocol:** WireGuard VPN **Uptime:** 11h 8m 41s **RX:** 1.32 KB (13 Pkts.) **TX:** 908.94 KB (6142 Pkts.) **IPv4:** 10.0.0.1/24 **IPv6:** fd86::1/64	Restart Stop Edit Delete

We can check if everything is working and if the new client (peer) has been added correctly by going to **Status > WireGuard Status**. We should see our server's public key there (in the **Configuration** item) and both peers.



WireGuard Status

Interface wg0

Configuration

Public Key: [redacted]
Listen Port: 1234

Peer: [redacted] DW

Public Key: [redacted]
Persistent Keepalive: 25s
Latest Handshake: Never
Data Received: 0 B
Data Transmitted: 0 B

Peer: [redacted] DW

Public Key: [redacted]
Persistent Keepalive: 25s
Latest Handshake: Never
Data Received: 0 B
Data Transmitted: 0 B

Powered by LuCI openwrt-19.07 branch (git-20.067.35699-cf18bdd) / OpenWrt 19.07.2 r10947-65030d81f3

Now we move on to setting the client in the application.

For iOS applications, we select **Create from scratch**, similarly to the first example above.

We enter **name** as **client2** and paste **Private key** which we read from the terminal:

Copy

```
tail client2-privatekey
```

Public key, similarly to the first example, should be automatically completed and should be the same as:

Copy

```
tail client2-publickey
```

In the **Addresses** field we put

Copy

```
10.0.0.3/24
```

or with IPv6

Copy

```
10.0.0.3/24, fd86::3/64
```

DNS servers: 192.168.1.1

Next, click on **Add peer** to add information about our server.

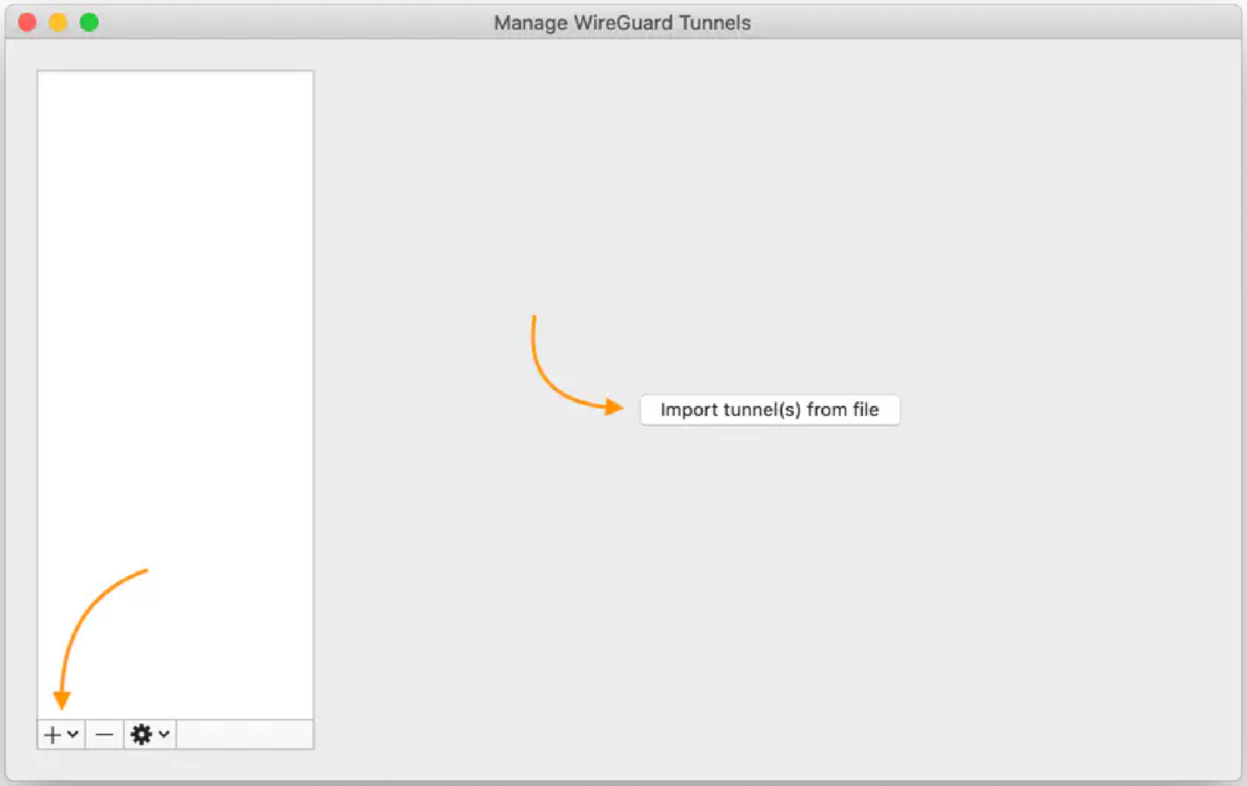
Here we repeat everything as in the first example.

Copy

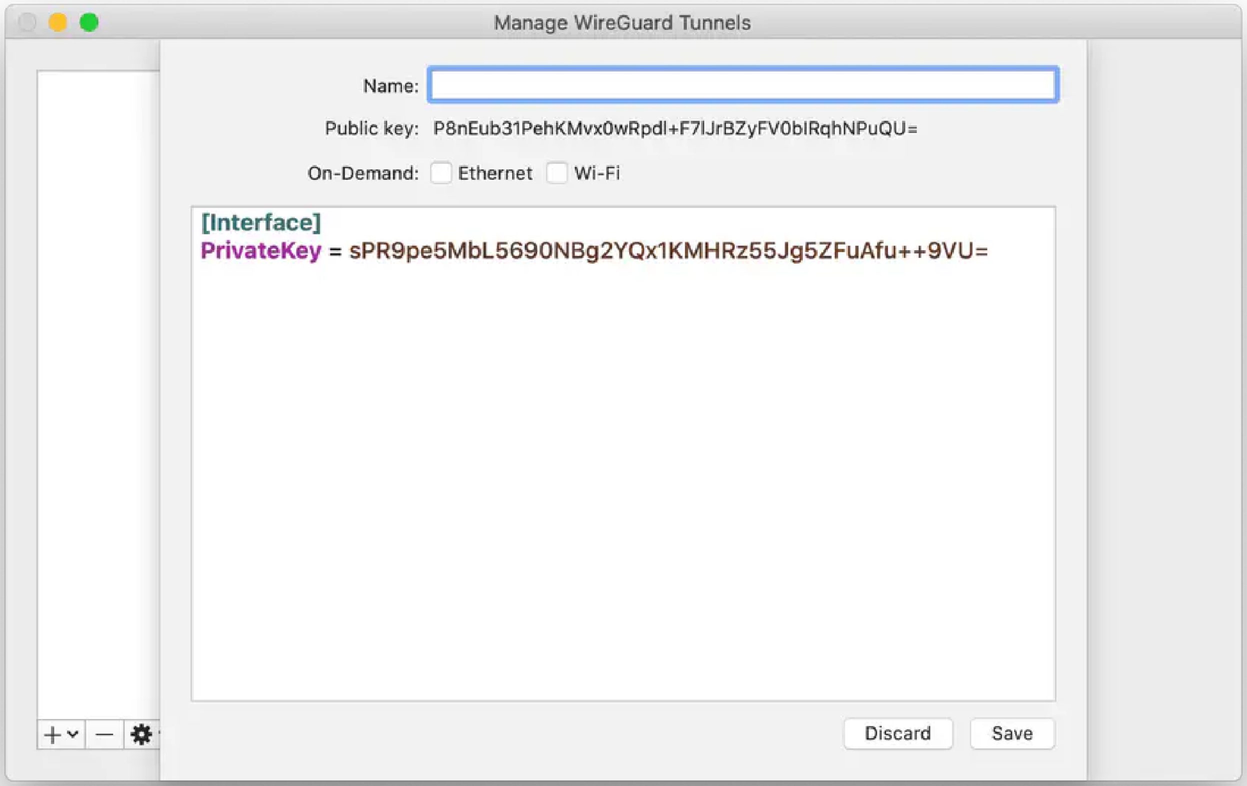
```
tail /etc/wireguard/server-publickey
```

Same **Endpoint**, **Allowed IPs** and **Persistent keepalive**.

However, as I have noticed, not every app has the ability to set everything up from scratch. In the case of WireGuard on macOS ↗, the number of options is limited to importing settings or adding an empty tunnel **Add empty tunnel**.



After clicking **Add empty tunnel** we will see a configuration file containing the generated **public key** and **private key**.



Public key: P8nEub31PehKMvx0wRpdl+F7lJrBZyFV0blRqhNPuQU=

```
[Interface]
PrivateKey = sPR9pe5MbL5690NBg2YQx1KMHRz55Jg5ZFuAfu++9VU=
```

We can use these keys by adding them to our interface in the **Peers** tab or by changing the **Private key** to a previously generated key:

Copy

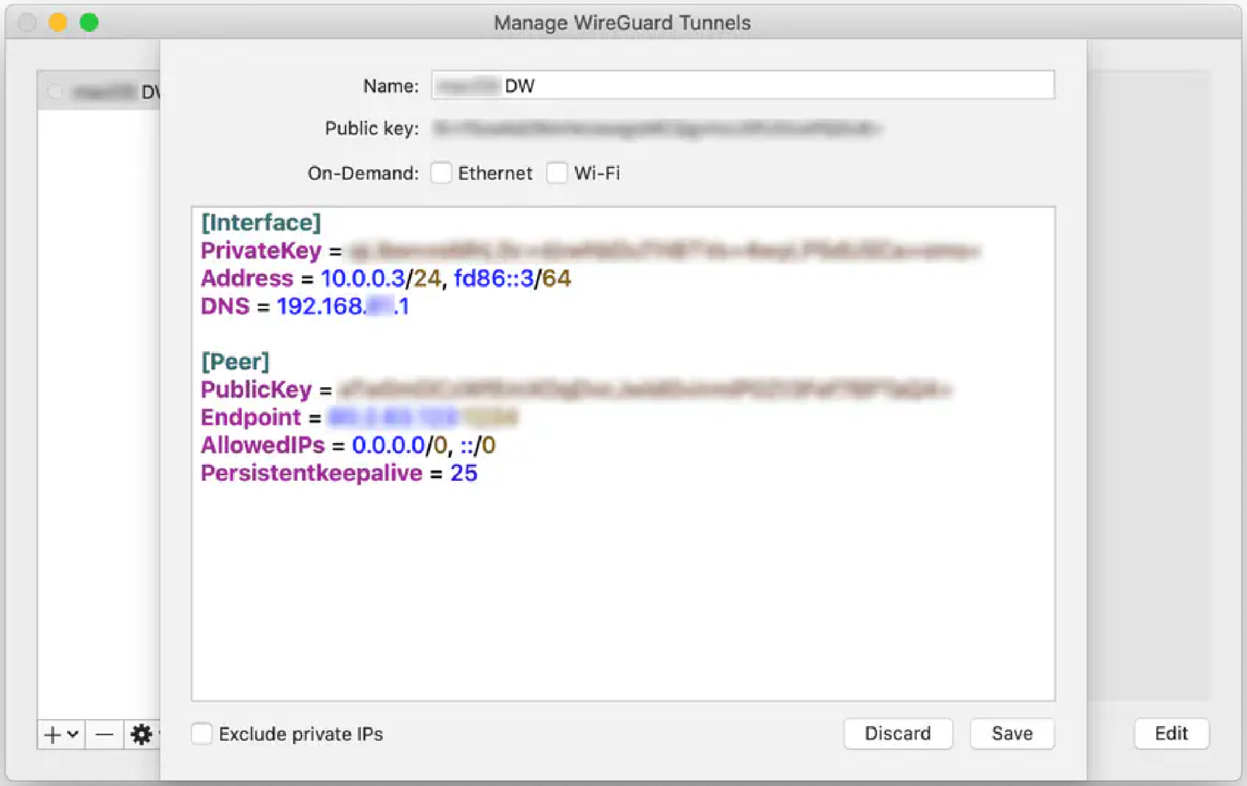
```
tail client2-privatekey
```

We will see that **Public key** will also change.

That's not all!

For the **iOS** app, we have also introduced our server's **Public key**, client IP address **(Addresses)**, name server **(DNS servers)**, **Endpoint**, **Allowed IPs** and **Persistent keepalive**.

Here we need to do it more manually. Our configuration will look like this:



Name: [redacted] DW

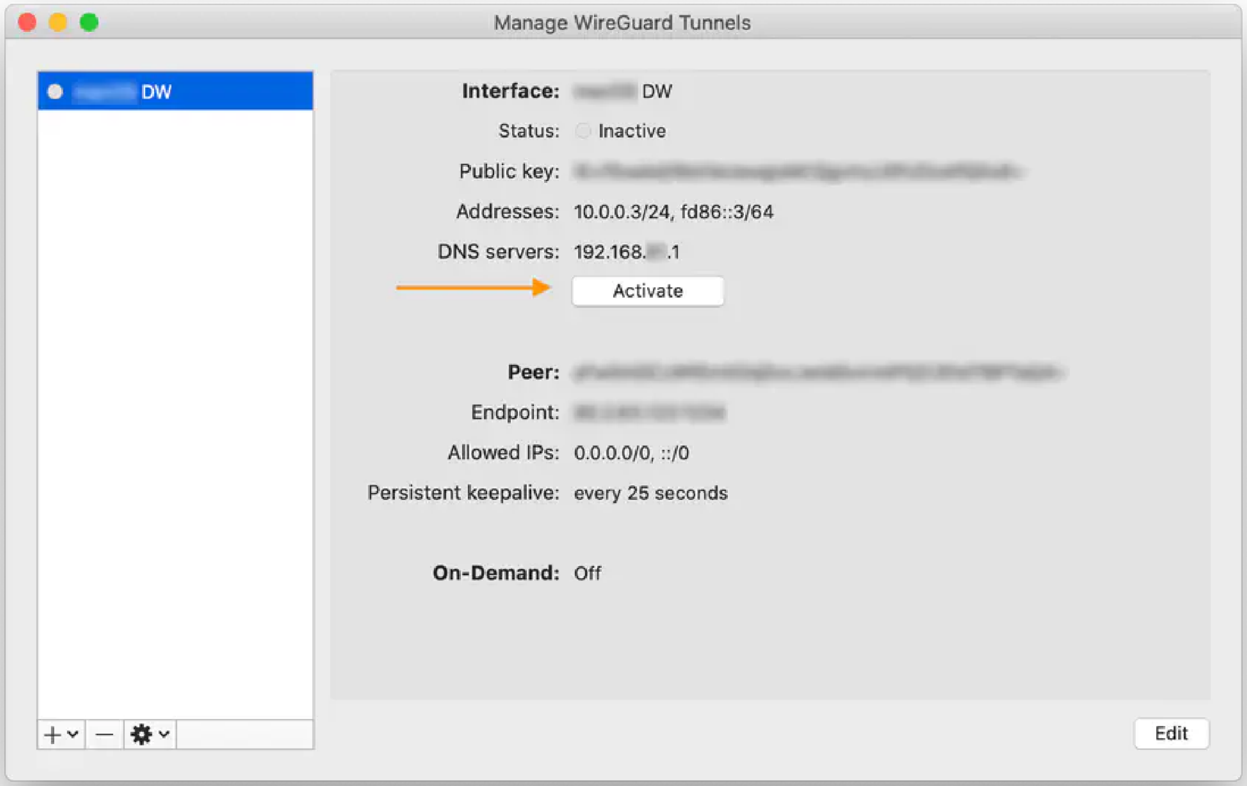
Public key: [redacted]

```
[Interface]
PrivateKey = [redacted]
Address = 10.0.0.3/24, fd86::3/64
DNS = 192.168.[redacted].1

[Peer]
PublicKey = [redacted]
Endpoint = [redacted]
AllowedIPs = 0.0.0.0/0, ::/0
Persistentkeepalive = 25
```

After pressing the **Save** button, macOS will ask you to add the appropriate VPN profile to the configuration.

And so, all we have to do is test whether everything works by clicking the **Activate** button.



Interface: DW
Status: Inactive
Public key:
Addresses: 10.0.0.3/24, fd86::3/64
DNS servers: 192.168.[redacted].1

Peer:
Endpoint:
Allowed IPs: 0.0.0.0/0, ::/0
Persistent keepalive: every 25 seconds

On-Demand: Off

If we did everything correctly, we should connect without any problems and everything should start working.

I hope this helps you when adding additional clients **(peers)**.

WireGuard and IPv6 Tunnel

If you've added an IPv6 connection to your Internet connection (which doesn't natively support it) via a tunnel using **Hurricane Electric Free IPv6 Tunnel Broker**, as I did in the post Adding an IPv6 tunnel to a router with OpenWrt, you're probably wondering how to configure the WireGuard server to support this solution.

The above description adds support for local IPv6, which works within the router, but does not support outgoing traffic (to the Internet), even when IPv6 is available. All traffic is therefore carried out exclusively using IPv4.

You can check this by visiting https://test-ipv6.com/ ↗ from your local network after connecting to WireGuard.

Yes, there is a possibility of adding IPv6 support! Which of course we will do!

The advantage of this solution is that not only will we be seen on the Internet using the router's external IP address (IPv4), but we can also assign an external, individual IPv6 address directly to our device connected to the VPN!

Although each WireGuard client on the Internet will be seen under one IP in version 4 (IPv4), in version 6 (IPv6) each device will have an individual address! All the magic of IPv6 (no need for port forwarding, etc.).

First we need to change the IPv6 settings we have entered, so where we have **fd86::** we need to replace it so that it matches our WAN6 interface prefix. We need to do this both on the router side, in the WireGuard server configuration and in the Peer settings, as well as on the client side (configuration file).

In the case of the server (including client settings - Peers - on the server side), we make changes via a web browser and then restart our **wg0** interface.

In the case of clients (Peers), we make changes in the configuration file and then import it to the application, or we edit each profile separately on each device.

To maintain consistency with the current IP address configuration, i.e. 10.0.0.1 for the server and subsequent numbers for clients, we will build the IPv6 address on the same principle.

Using IPv4 to IPv6 calculator ↗ for 10.0.0.1 we get for example:

Copy

```
0:0:0:0:0:ffff:a00:1
```

We will change the first 4 zeros (**0:0:0:0:**) to the network prefix we received for our tunnel ↗.

For example, if we received in **Routed IPv6 Prefixes** something like this:

Copy

```
2001:470:____:5cb::/64
```

Start - The prefix **2001:470:____:5cb:** will be constant, and each local address will get **0:ffff:a00:1**, with the last digit changing. 1 for the server, 2 for the first client, and so on.

And so, we change the server settings **fd86::1/64** to **2001:470:____:5cb:0:ffff:a00:1/64**

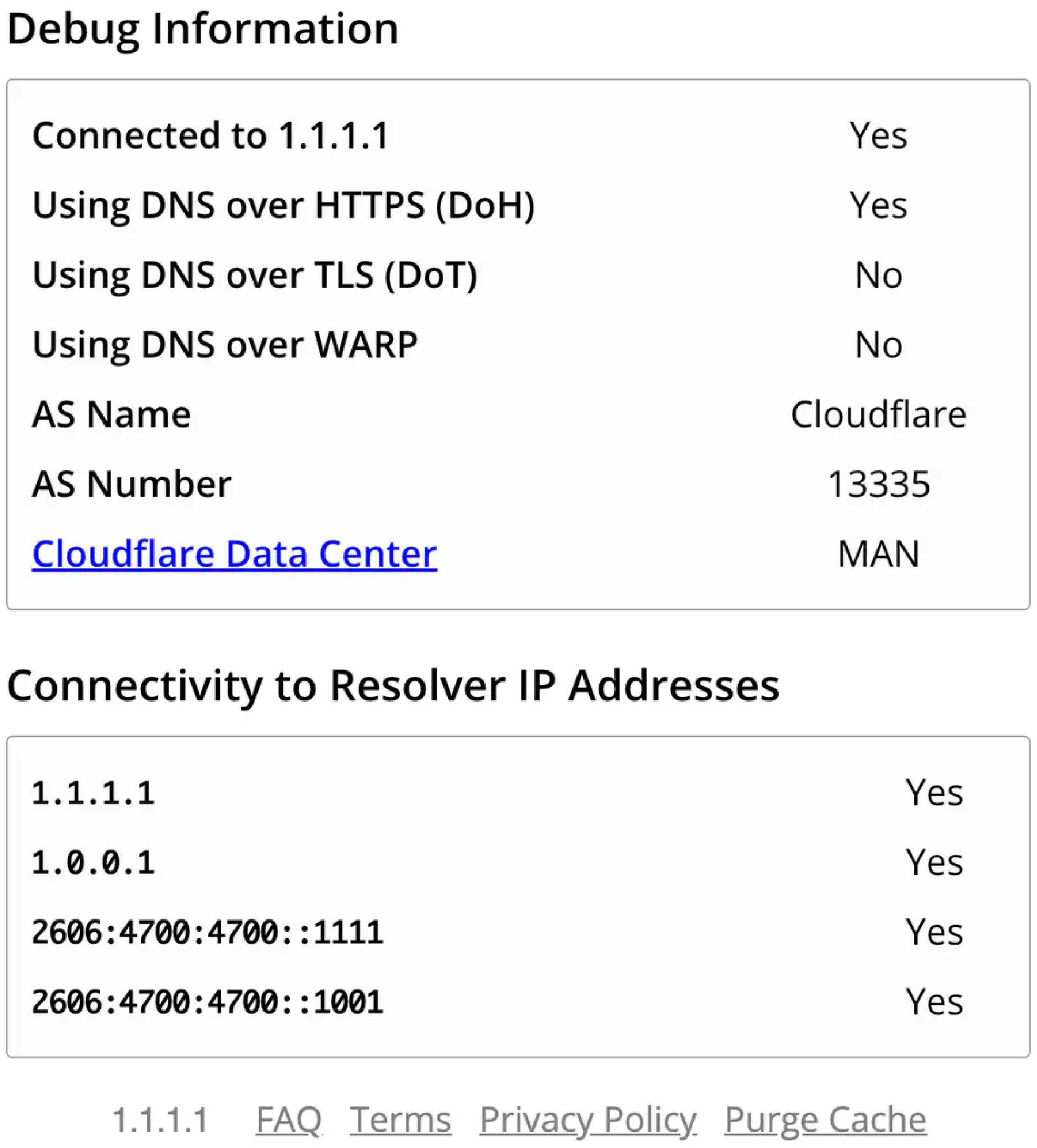
For the first Peer **fd86::2/128** to **2001:470:____:5cb:0:ffff:a00:2/128**

On the client side, in the WireGuard application, for the first client where we have: **10.0.0.2/24, fd86::2/64** change to **10.0.0.2/24, 2001:470:____:5cb:0:ffff:a00:2/64**

> Item **:____:** has been censored of its full (current) prefix.
>
> **0:ffff:a00:1** is an example address and I recommend not using it, but choosing your own unique one. If you are not sure whether the address you have chosen is correct, validate it along with the prefix using, for example, this tool ↗.

If we did everything correctly, then **after connecting to the VPN** and going to the website https://test-ipv6.com ↗ we will get a result of **10/10**.

Additionally, if we have **Secure DNS** set, the https://1.1.1.1/help ↗ page will also return in **Connectivity to Resolver IP Addresses** the availability of DNS addresses from the IPv6 level.



Debug Information

Connected to 1.1.1.1	Yes
Using DNS over HTTPS (DoH)	Yes
Using DNS over TLS (DoT)	No
Using DNS over WARP	No
AS Name	Cloudflare
AS Number	13335
Cloudflare Data Center	MAN

Connectivity to Resolver IP Addresses

1.1.1.1	Yes
1.0.0.1	Yes
2606:4700:4700::1111	Yes
2606:4700:4700::1001	Yes

Regards.

Credits: **Birkhoff Lee**

#hardware #asus rt-ac57u #openvpn #openwrt #peer #private key #public key #tunel ipv6 #ipv6 #vpn #wireguard #wireguard openwrt

Share

▶ Comments & Reactions

Categories

apple 20 blog 73 chromeos 3 hardware 23 hugo 12 internet 10
keep it short 3 mac 5 recipes 3 software 17 technology 5 webdev 28
windows 18 writing 1