

Some Unofficial WireGuard Documentation

API reference guide for WireGuard including Setup, Configuration, and Usage, with examples.

All credit goes to the WireGuard project, zx2c4 (<https://www.zx2c4.com/>) and the open source contributors (<https://github.com/WireGuard/WireGuard/graphs/contributors>) for the original software, this is my solo unofficial attempt at providing more comprehensive documentation, API references, and examples.

Source for these docs, example code, and issue tracker: <https://github.com/pirate/wireguard-docs>
(<https://github.com/pirate/wireguard-docs>)

Nicer HTML page version: <https://docs.sweeting.me/s/wireguard> (<https://docs.sweeting.me/s/wireguard>)

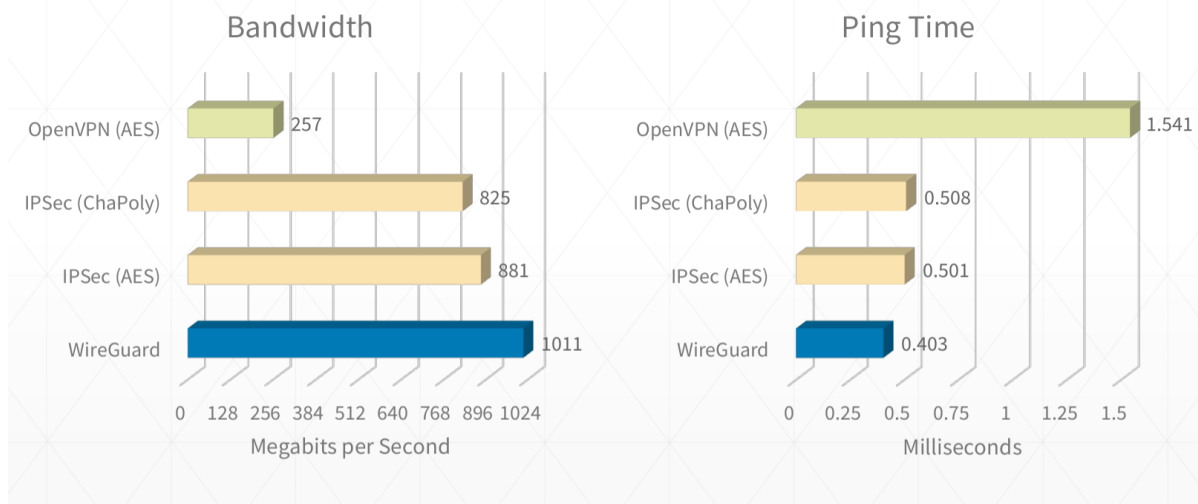
WireGuard (<https://www.wireguard.com/>) is an open-source VPN solution written in C by Jason Donenfeld (<https://www.jasondonenfeld.com>) and others (<https://github.com/WireGuard/WireGuard/graphs/contributors>), aiming to fix many of the problems that have plagued other modern server-to-server VPN offerings like IPSec/IKEv2, OpenVPN, or L2TP. It shares some similarities with other modern VPN offerings like Tinc (<https://www.tinc-vpn.org/>) and MeshBird (<https://github.com/meshbird/meshbird>), namely good cipher suites and minimal config. As of 2020-01 it's been merged into the 5.6 version of the Linux kernel (<https://arstechnica.com/gadgets/2020/01/linus-torvalds-pulled-wireguard-vpn-into-the-5-6-kernel-source-tree/>), meaning it will ship with most Linux systems out-of-the-box.

Official Links

- Homepage: <https://www.wireguard.com> (<https://www.wireguard.com>)
- Install: <https://www.wireguard.com/install/> (<https://www.wireguard.com/install/>)
- QuickStart: <https://www.wireguard.com/quickstart/> (<https://www.wireguard.com/quickstart/>)
- Manpages: `wg` (<https://manpages.debian.org/testing/wireguard-tools/wg.8.en.html>), `wg-quick` (<https://manpages.debian.org/unstable/wireguard-tools/wg-quick.8.en.html>)
- Main Git repo: <https://git.zx2c4.com/WireGuard/> (<https://git.zx2c4.com/WireGuard/>)
- GitHub Mirror: <https://github.com/WireGuard/WireGuard> (<https://github.com/WireGuard/WireGuard>)
- Mailing List: <https://lists.zx2c4.com/mailman/listinfo/wireguard> (<https://lists.zx2c4.com/mailman/listinfo/wireguard>)

WireGuard Goals

- strong, modern security by default
- minimal config and key management
- fast, both low-latency and high-bandwidth
- simple internals and small protocol surface area
- simple CLI and seamless integration with system networking



(<https://www.ckn.io>)

[/blog/2017/11/14/wireguard-vpn-typical-setup/](https://www.ckn.io/blog/2017/11/14/wireguard-vpn-typical-setup/))

It's also *fast as hell*. I routinely get sub 0.5ms pings and 900mbps+ on good connections.

(See <https://www.ckn.io/blog/2017/11/14/wireguard-vpn-typical-setup/>)

Table of Contents

See <https://github.com/pirate/wireguard-docs> (<https://github.com/pirate/wireguard-docs>) for example code and documentation source.

- Table of Contents
- Intro
 - WireGuard Overview
 - List of Other VPN Solutions
 - Mesh VPN Solutions
 - VPN Setup Tools
- WireGuard Documentation
 - Glossary
 - Example Strings
 - Peer/Node/Device
 - Bounce Server
 - Subnet
 - CIDR Notation
 - NAT
 - Public Endpoint
 - Private key
 - Public key
 - DNS
 - How WireGuard Works
 - How Public Relay Servers Work
 - How WireGuard Routes Packets
 - What WireGuard Traffic Looks Like
 - WireGuard Performance
 - WireGuard Security Model
 - How WireGuard Manages Keys
 - Usage
 - QuickStart
 - Setup
 - Config Creation
 - Key Generation
 - Start / Stop
 - Inspect
 - Testing
 - Config Reference
 - Overview
 - [Interface]
 - [Peer]
 - Advanced Topics
 - IPv6
 - Forwarding All Traffic
 - NAT To NAT Connections

- Dynamic IP Allocation
- Other WireGuard Implementations
- WireGuard Setup Tools
- Config Shortcuts
- Containerization
- Further Reading
 - Reference Docs
 - Tutorials
 - Papers, Articles, and Talks
 - Related Projects
 - Docker
 - Other
 - Discussions

Intro

Whether living behind the Great Wall of China or just trying to form a network between your servers, WireGuard is a great option and serves as a “lego block” for building networks (much in the same way that ZFS is a lego block for building filesystems).

WireGuard Overview

- minimal config, low tunable surface area and sane defaults
- minimal key management work needed, just 1 public & 1 private key per host
- behaves like a normal ethernet interface, behaves well with standard kernel packet routing rules
- ability to easily create a LAN like 192.0.2.0/24 between all servers, or more complex networks using custom routes
- ability to some traffic or all traffic to/through arbitrary hosts on the VPN LAN
- robust automatic reconnects after reboots / network downtime / NAT connection table drops
- fast (low latency and line-rate bandwidth)
- modern encryption, secure by default with forward secrecy & resilience to downgrade attacks
- ideally support for any type of Level 2 and control traffic, e.g. ARP/DHCP/ICMP (or ideally raw ethernet frames), not just TCP/HTTP
- ability to join the VPN from Ubuntu, FreeBSD, iOS, MacOS, Windows, Android (via open-source apps or natively)
- supports both running on the host routing traffic for docker or running in a docker container routing for the host

Things WireGuard does not do:

- form a self-healing mesh network where nodes automatically gossip with neighbors
- break through double NATs with a signalling server (WebRTC-style)
- handle automatically distributing & revoking keys through a central authority
- allow sending raw layer-2 ethernet frames (it's at the IP layer)

But you can write your own solutions for these problems using WireGuard under the hood (like Tailscale (<https://github.com/tailscale/tailscale>) or AltheaNet (<https://althea.net/>)).

List of Other VPN Solutions

- WireGuard (<https://www.wireguard.com/>)
- IPSec (IKEv2) (<https://github.com/jawj/IKEv2-setup>)/strongSwan: in my experience, there was lots of brittle config that was different for each OS, the NAT busting setup is very manual and involves updating the central server and starting all the others in the correct order, it wasn't great at becoming stable again after network downtime, had to be manually restarted often. your mileage may vary.
- OpenVPN (<https://openvpn.net/vpn-server-resources/site-to-site-routing-explained-in-detail/>): can work over UDP or be disguised as HTTPS traffic over TCP
- StealthVPN: haven't tried it, should I?
- DsVPN (<https://github.com/jedisct1/dsvpn>): I think it does TCP-over-TCP which usually doesn't end well...
- SoftEther (<https://www.softether.org/>) (SSTP (https://en.wikipedia.org/wiki/Secure_Socket_Tunneling_Protocol)): haven't tried it yet, should I? (also does TCP-over-TCP?)
- L2TP: somewhat outdated
- PPTP: ancient, inflexible, insecure, doesn't solve all the requirements
- SOCKS/SSH: good for proxying single-port traffic, not a full networking tunnel or VPN

Mesh VPN Solutions

- TINC (<https://www.tinc-vpn.org/>): haven't tried it yet, but it doesn't work on iOS, worst case scenario I could live
- VPNCloud (<https://github.com/dswd/vpncloud>): similar properties to WireGuard, with more auto-mesh features
- cjdns (<https://github.com/cjdelisle/cjdns>): haven't tried it yet, should I?
- ZeroTier (<https://www.zerotier.com>): haven't tried it yet, should I
- MeshBird (<https://github.com/meshbird/meshbird>): "Cloud native" VPN/networking layer
- Yggdrasil Network (<https://yggdrasil-network.github.io/>): Yggdrasil is a self-arranging IPv4/IPv6 mesh VPN (haven't tried it yet)

VPN Setup Tools

- Algo (<https://github.com/trailofbits/algo>) WireGuard setup tool
- Striesand (<https://github.com/StreisandEffect/striesand>) Multi-protocol setup tool
- IKEv2-setup (<https://github.com/jawj/IKEv2-setup>) IKEv2 server setup script

- WireGuard-Manager (<https://github.com/complexorganizations/wireguard-manager>) WireGuard setup tool, all in one

WireGuard Documentation

Glossary

Example Strings

These are demo hostnames, domain names, IP addresses, and ranges used in the documentation and example configs.

Replace them with your preferred values when doing your own setup.

- Example domain: `example-vpn.dev` can be replaced with any publicly accessible domain you control
- Example hostnames: `public-server1`, `public-server2`, `home-server`, `laptop`, `phone` can be changed to your device hostnames
- IP addresses & ranges: `192.0.2.1/24`, `192.0.2.3`, `192.0.2.3/32`, `2001:DB8::/64` can be replaced with your preferred subnets and addresses (e.g. `192.168.5.1/24`)

Wherever you see these strings below, they're just being used as placeholder values to illustrate an example and have no special meaning.

Make sure to change the IP addresses in your configs! The blocks used in these docs are reserved for example purposes by the IETF and should never be used in real network setups.

- `192.0.2.0/24` (TEST-NET-1) IPv4 example range RFC5737 (<https://tools.ietf.org/html/rfc5737>)
- `2001:DB8::/32` IPv6 example range RFC3849 (<https://tools.ietf.org/html/rfc3849>)

You can use any private range you want for your own setups, e.g. `10.0.44.0/24`, just make sure they don't conflict with any of the LAN subnet ranges your peers are on.

Peer/Node/Device

A host that connects to the VPN and registers a VPN subnet address such as `192.0.2.3` for itself. It can also optionally route traffic for more than its own address(es) by specifying subnet ranges in comma-separated CIDR notation.

Bounce Server

A publicly reachable peer/node that serves as a fallback to relay traffic for other VPN peers behind NATs. A bounce server is not a special type of server, it's a normal peer just like all the others, the only difference is that it has a public IP and has kernel-level IP forwarding turned on which allows it to bounce traffic back down the VPN to other clients.

See more: <https://tailscale.com/blog/how-nat-traversal-works/> (<https://tailscale.com/blog/how-nat-traversal-works/>) (Tailscale uses Wireguard under the hood)

Subnet

A group of IPs separate from the public internet, e.g. 192.0.2.1-255 or 192.168.1.1/24. Generally behind a NAT provided by a router, e.g. in office internet LAN or a home Wi-Fi network.

CIDR Notation

A way of defining a subnet and its size with a “mask”, a smaller mask = more address bits usable by the subnet & more IPs in the range. Most common ones:

- 192.0.2.1/32 (a single IP address, 192.0.2.1) netmask = 255.255.255.255
- 192.0.2.1/24 (255 IPs from 192.0.2.0 - 192.0.2.255) netmask = 255.255.255.0
- 192.0.2.1/16 (65,536 IPs from 192.0.0.0 - 192.0.255.255) netmask = 255.255.0.0
- 192.0.2.1/8 (16,777,216 IPs from 192.0.0.0 - 192.255.255.255) netmask = 255.0.0.0
- 0.0.0.1/0 (4,294,967,296 IPs from 0.0.0.0 - 255.255.255.255) netmask = 0.0.0.0
- IPv6 CIDR notation is also supported e.g. 2001:DB8::/64

https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing (https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing)

To people just getting started 192.0.2.1/32 may seem like a weird and confusing way to refer to a single IP. This design is nice though because it allows peers to expose multiple IPs if needed without needing multiple notations. Just know that anywhere you see something like 192.0.2.3/32 , it really just means 192.0.2.3 .

NAT

A subnet with private IPs provided by a router standing in front of them doing Network Address Translation, individual nodes are not publicly accessible from the internet, instead the router keeps track of outgoing connections and forwards responses to the correct internal IP (e.g. standard office networks, home Wi-Fi networks, free public Wi-Fi networks, etc)

Public Endpoint

The publicly accessible address:port for a node, e.g. 123.124.125.126:1234 or some.domain.tld:1234 (must be accessible via the public internet, generally can't be a private IP like 192.0.2.1 or 192.168.1.1 unless it's directly accessible using that address by other peers on the same subnet).

Private key

A WireGuard private key for a single node, generated with:

```
wg genkey > example.key
```

(never leaves the node it's generated on)

Public key

A WireGuard public key for a single node, generated with:

```
wg pubkey < example.key > example.key.pub
```

(shared with other peers)

DNS

Domain Name Server, used to resolve hostnames to IPs for VPN clients, instead of allowing DNS requests to leak outside the VPN and reveal traffic. Leaks are testable with <http://dnsleak.com>

(<http://dnsleak.com>).

How WireGuard Works

How Public Relay Servers Work

Public relays are just normal VPN peers that are able to act as an intermediate relay server between any VPN clients behind NATs, they can forward any VPN subnet traffic they receive to the correct peer at the system level (WireGuard doesn't care how this happens, it's handled by the kernel `net.ipv4.ip_forward = 1` and the iptables routing rules).

If all peers are publicly accessible, you don't have to worry about special treatment to make one of them a relay server, it's only needed if you have any peers connecting from behind a NAT.

Each client only needs to define the publicly accessible servers/peers in its config, any traffic bound to other peers behind NATs will go to the catchall VPN subnet (e.g. `192.0.2.1/24`) in the public relays `AllowedIPs` route and will be forwarded accordingly once it hits the relay server.

In summary: only direct connections between clients should be configured, any connections that need to be bounced should not be defined as peers, as they should head to the bounce server first and be routed from there back down the vpn to the correct client.

How WireGuard Routes Packets

More complex topologies are definitely achievable, but these are the basic routing methods used in typical WireGuard setups:

- **Direct node-to-node**

In the simplest case, the nodes will either be on the same LAN or both be publicly accessible. Define directly accessible nodes with hardcoded `Endpoint` addresses and ports so that WireGuard can connect straight to the open port and route UDP packets without intermediate hops.

- **Node behind local NAT to public node**

When 1 of the 2 parties is behind remote NAT (e.g. when a laptop behind NAT connects to `public-server2`), define the publicly accessible node with a hardcoded `Endpoint` and the NAT-ed node without. The connection will be opened from NAT client -> public client, then

traffic will route directly between them in both directions as long as the connection is kept alive by outgoing `PersistentKeepalive` pings from the NAT-ed client.

- **Node behind local NAT to node behind remote NAT (via relay)**

Most of the time when both parties are behind NATs, the NATs do source port randomization making direct connections infeasible, so they will both have to open a connection to `public-server1`, and traffic will forward through the intermediary bounce server as long as the connections are kept alive.

- **Node behind local NAT to node behind remote NAT (via UDP NAT hole-punching)**

While sometimes possible, it's generally infeasible to do direct NAT-to-NAT connections on modern networks, because most NAT routers are quite strict about randomizing the source port, making it impossible to coordinate an open port for both sides ahead of time. Instead, a signaling server (STUN) must be used that stands in the middle and communicates which random source ports are assigned to the other side. Both clients make an initial connection to the public signaling server, then it records the random source ports and sends them back to the clients. This is how WebRTC works in modern P2P web apps. Even with a signalling server and known source ports for both ends, sometimes direct connections are not possible because the NAT routers are strict about only accepting traffic from the original destination address (the signalling server), and will require a new random source port to be opened to accept traffic from other IPs (e.g. the other client attempting to use the originally communicated source port). This is especially true for "carrier-grade NATs" like cellular networks and some enterprise networks, which are designed specifically to prevent this sort of hole-punching connection. See the full section below on **NAT to NAT Connections** for more information.

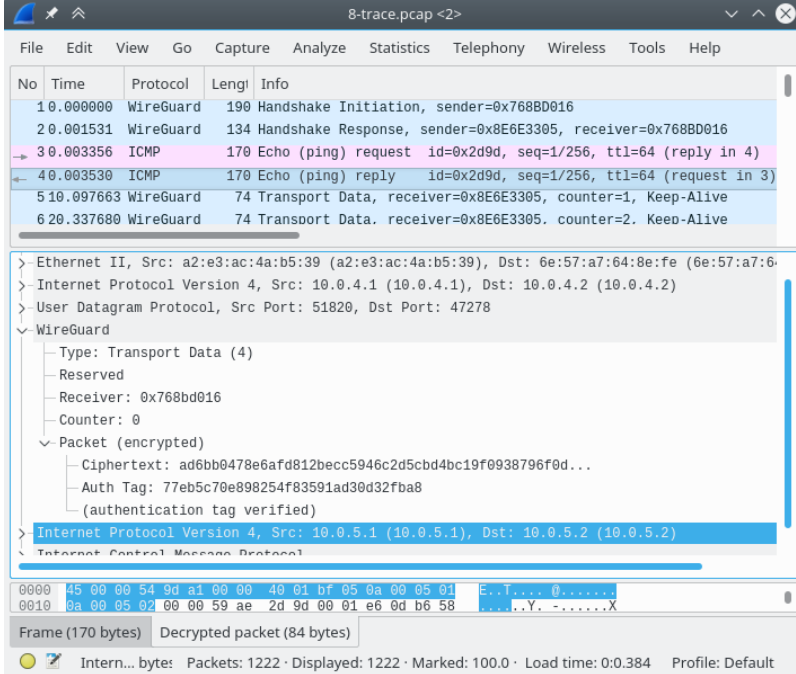
More specific (also usually more direct) routes provided by other peers will take precedence when available, otherwise traffic will fall back to the least specific route and use the `192.0.2.1/24` catchall to forward traffic to the bounce server, where it will in turn be routed by the relay server's system routing table (`net.ipv4.ip_forward = 1`) back down the VPN to the specific peer that's accepting routes for that traffic. WireGuard does not automatically find the fastest route or attempt to form direct connections between peers if not already defined, it just goes from the most specific route in `[Peers]` to least specific.

You can figure out which routing method WireGuard is using for a given address by measuring the ping times to figure out the unique length of each hop, and by inspecting the output of:

```
wg show wg0
```

What WireGuard Traffic Looks Like

WireGuard uses encrypted UDP packets for all traffic, it does not provide guarantees around packet delivery or ordering, as that is handled by TCP connections within the encrypted tunnel.



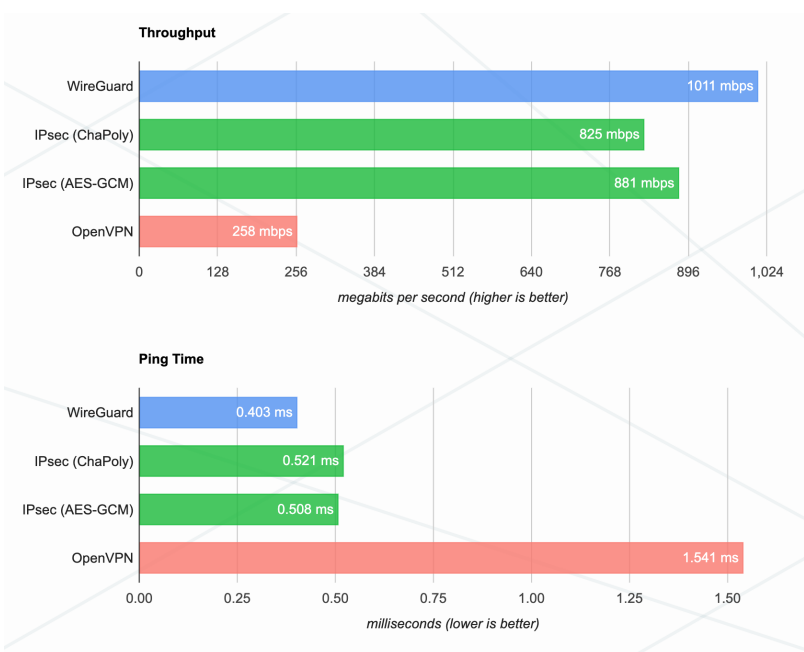
Further reading:

- <https://www.wireshark.org/docs/dfref/w/wg.html> (<https://www.wireshark.org/docs/dfref/w/wg.html>)
- <https://github.com/Lekensteyn/wireguard-dissector> (<https://github.com/Lekensteyn/wireguard-dissector>)
- <https://nbsoftsolutions.com/blog/viewing-wireguard-traffic-with-tcpdump> (<https://nbsoftsolutions.com/blog/viewing-wireguard-traffic-with-tcpdump>)

WireGuard Performance

WireGuard claims faster performance than most other competing VPN solutions, though the exact numbers are sometimes debated and may depend on whether hardware-level acceleration is available for certain cryptographic ciphers.

WireGuard’s performance gains are achieved by handling routing at the kernel level, and by using modern cipher suites running on all cores to encrypt traffic. WireGuard also gains a significant advantage by using UDP with no delivery/ordering guarantees (compared to VPNs that run over TCP or implement their own guaranteed delivery mechanisms).



Further reading:

- <https://www.wireguard.com/performance/> (<https://www.wireguard.com/performance/>)
- https://www.reddit.com/r/linux/comments/9bnowo/wireguard_benchmark_between_two_servers_with_10/ (https://www.reddit.com/r/linux/comments/9bnowo/wireguard_benchmark_between_two_servers_with_10/)
- <https://restoreprivacy.com/openssl-ipsec-wireguard-l2tp-ikev2-protocols/> (<https://restoreprivacy.com/openssl-ipsec-wireguard-l2tp-ikev2-protocols/>)

WireGuard Security Model

WireGuard uses the following protocols and primitives to secure traffic:

- ChaCha20 for symmetric encryption, authenticated with Poly1305, using RFC7539's AEAD construction
- Curve25519 for ECDH
- BLAKE2s for hashing and keyed hashing, described in RFC7693
- SipHash24 for hashtable keys
- HKDF for key derivation, as described in RFC5869

WireGuard's cryptography is essentially an instantiation of Trevor Perrin's Noise framework. It's modern and, again, simple. Every other VPN option is a mess of negotiation and handshaking and complicated state machines. WireGuard is like the Signal/Axolotl of VPNs, except it's much simpler and easier to reason about (cryptographically, in this case) than double ratchet messaging protocols.

It is basically the gmail of VPN software.

And it's ~4000 lines of code. It is plural orders of magnitude smaller than its competitors.

<https://news.ycombinator.com/item?id=14599834> (<https://news.ycombinator.com/item?id=14599834>)

Further reading:

- <https://www.wireguard.com/papers/wireguard.pdf> (<https://www.wireguard.com/papers/wireguard.pdf>)
- <https://eprint.iacr.org/2018/080.pdf> (<https://eprint.iacr.org/2018/080.pdf>)
- <https://courses.csail.mit.edu/6.857/2018/project/He-Xu-Xu-WireGuard.pdf> (<https://courses.csail.mit.edu/6.857/2018/project/He-Xu-Xu-WireGuard.pdf>)
- <https://www.wireguard.com/talks/blackhat2018-slides.pdf> (<https://www.wireguard.com/talks/blackhat2018-slides.pdf>)
- <https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen/> (<https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen/>)

How WireGuard Manages Keys

Authentication in both directions is achieved with a simple public/private key pair for each peer.

Each peer generates these keys during the setup phase, and shares only the public key with other peers.

No other certificates or pre-shared keys are needed beyond the public/private keys for each node.

Key generation, distribution, and revocation can be handled in larger deployments using a separate service like Ansible or Kubernetes Secrets.

Some services that help with key distribution and deployment:

- <https://pypi.org/project/wireguard-p2p/> (<https://pypi.org/project/wireguard-p2p/>)
- <https://github.com/trailofbits/algo> (<https://github.com/trailofbits/algo>)
- <https://github.com/StreisandEffect/streisand> (<https://github.com/StreisandEffect/streisand>)
- <https://github.com/its0x08/wg-install> (<https://github.com/its0x08/wg-install>)
- https://github.com/brittson/wireguard_config_maker (https://github.com/brittson/wireguard_config_maker)
- <https://www.wireguardconfig.com> (<https://www.wireguardconfig.com>)
- <https://wirt.network> (<https://wirt.network>)

You can also read in keys from a file or via command if you don't want to hardcode them in `wg0.conf`, this makes managing keys via 3rd party service much easier:

```
[Interface]
...
PostUp = wg set %i private-key /etc/wireguard/wg0.key <(cat /some/path/%i/privkey)
```

Technically, multiple servers can share the same private key as long as clients aren't connected to two servers with the same key simultaneously.

An example of a scenario where this is a reasonable setup is if you're using round-robin DNS to load-balance connections between two servers that are pretending to be a single server.

Most of the time however, every peer should have its own public/private keypair so that peers can't read each other's traffic and can be individually revoked.

Usage

QuickStart

Overview of the general process:

1. Install `apt install wireguard` or `pkg/brew install wireguard-tools` on each node
2. Generate public and private keys locally on each node `wg genkey + wg pubkey`
3. Create a `wg0.conf` WireGuard config file on the main relay server
 - `[Interface]` Make sure to specify a CIDR range for the entire VPN subnet when defining the address the server accepts routes for `Address = 192.0.2.1/24`

- [Peer] Create a peer section for every client joining the VPN, using their corresponding remote public keys
4. Create a `wg0.conf` on each client node
 - [Interface] Make sure to specify only a single IP for client peers that don't relay traffic `Address = 192.0.2.3/32` .
 - [Peer] Create a peer section for each public peer not behind a NAT, make sure to specify a CIDR range for the entire VPN subnet when defining the remote peer acting as the bounce server `AllowedIPs = 192.0.2.1/24` . Make sure to specify individual IPs for remote peers that don't relay traffic and only act as simple clients `AllowedIPs = 192.0.2.3/32` .
 5. Start WireGuard on the main relay server with `wg-quick up /full/path/to/wg0.conf`
 6. Start WireGuard on all the client peers with `wg-quick up /full/path/to/wg0.conf`
 7. Traffic is routed from peer to peer using most specific route first over the WireGuard interface, e.g. `ping 192.0.2.3` checks for a direct route to a peer with `AllowedIPs = 192.0.2.3/32` first, then falls back to a relay server that's accepting IPs in the whole subnet

Setup

```
# install on Ubuntu
sudo add-apt-repository ppa:wireguard/wireguard
apt install wireguard

# install on macOS
brew install wireguard-tools

# install on FreeBSD
pkg install wireguard

# install on iOS/Android using Apple App Store/Google Play Store
# install on other systems using https://www.wireguard.com/install/#installation

# to enable kernel relaying/forwarding ability on bounce servers
echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
echo "net.ipv4.conf.all.proxy_arp = 1" >> /etc/sysctl.conf
sudo sysctl -p /etc/sysctl.conf

# to add iptables forwarding rules on bounce servers
iptables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i wg0 -o wg0 -m conntrack --ctstate NEW -j ACCEPT
iptables -t nat -A POSTROUTING -s 192.0.2.0/24 -o eth0 -j MASQUERADE
```

Config Creation

```
nano wg0.conf # can be placed anywhere, must be referred to using absolute path (usi
```

Key Generation

```
# generate private key
wg genkey > example.key

# generate public key
wg pubkey < example.key > example.key.pub
```

Start / Stop

```
wg-quick up /full/path/to/wg0.conf
wg-quick down /full/path/to/wg0.conf
# Note: you must specify the absolute path to wg0.conf, relative paths won't work

# start/stop VPN network interface
ip link set wg0 up
ip link set wg0 down

# register/unregister VPN network interface
ip link add dev wg0 type wireguard
ip link delete dev wg0

# register/unregister local VPN address
ip address add dev wg0 192.0.2.3/32
ip address delete dev wg0 192.0.2.3/32

# register/unregister VPN route
ip route add 192.0.2.3/32 dev wg0
ip route delete 192.0.2.3/32 dev wg0
```

Inspect

Interfaces

```
# show system LAN and WAN network interfaces
ifconfig
ip address show

# show system VPN network interfaces
ifconfig wg0
ip link show wg0

# show WireGuard VPN interfaces
wg show all
wg show wg0
```

Addresses

```
# show public IP address
ifconfig eth0
ip address show eth0
dig -4 +short myip.opendns.com @resolver1.opendns.com

# show VPN IP address
ip address show wg0
```

Routes

```
# show WireGuard routing table and peer connections
wg show
wg show wg0 allowed-ips

# show system routing table
ip route show table main
ip route show table local

# show system route to specific address
ip route get 192.0.2.3
```

Logs

To enable additional logging run:

```
modprobe wireguard
echo module wireguard +p > /sys/kernel/debug/dynamic_debug/control
```

To follow logs:

```
dmesg -wH
```

Systems with modern kernel and Safe Boot might require disabling Secure Boot DKMS Signature Verification to allow access to kernel logs.

```
mokutil --disable-verification
reboot
```

Testing

Ping Speed

```
# check that main relay server is accessible directly via public internet
ping public-server1.example-vpn.dev

# check that the main relay server is available via VPN
ping 192.0.2.1

# check that public peers are available via VPN
ping 192.0.2.2

# check that remote NAT-ed peers are available via VPN
ping 192.0.2.3

# check that NAT-ed peers in your local LAN are available via VPN
ping 192.0.2.4
```

Bandwidth

```
# install iperf using your preferred package manager
apt/brew/pkg install iperf

# check bandwidth over public internet to relay server
iperf -s # on public relay server
iperf -c public-server1.example-vpn.dev # on local client

# check bandwidth over VPN to relay server
iperf -s # on public relay server
iperf -c 192.0.2.1 # on local client

# check bandwidth over VPN to remote public peer
iperf -s # on remote public peer
iperf -c 192.0.2.2 # on local client

# check bandwidth over VPN to remote NAT-ed peer
iperf -s # on remote NAT-ed peer
iperf -c 192.0.2.3 # on local client

# check bandwidth over VPN to local NAT-ed peer (on same LAN)
iperf -s # on local NAT-ed peer
iperf -c 192.0.2.4 # on local client
```

DNS

Check for DNS leaks using <http://dnsleak.com> (<http://dnsleak.com>), or by checking the resolver on a lookup:

```
dig example.com A
```

Config Reference

Overview

WireGuard config is in INI syntax (https://en.wikipedia.org/wiki/INI_file), defined in a file usually called `wg0.conf`. It can be placed anywhere on the system, but is often placed in `/etc/wireguard/wg0.conf`.

The config path is specified as an argument when running any `wg-quick` command, e.g:
`wg-quick up /etc/wireguard/wg0.conf` (always specify the full, absolute path)

The config file name must be in the format `${name of the new WireGuard interface}.conf`. WireGuard interface names are typically prefixed with `wg` and numbered starting at `0`, but you can use any name that matches the regex `^[a-zA-Z0-9_+=.-]{1,15}$`.

Config files can opt to use the limited set of `wg` config options, or the more extended `wg-quick` options, depending on what command is preferred to start WireGuard. These docs recommend sticking to `wg-quick` as it provides a more powerful and user-friendly config experience.

Jump to definition:

```
[Interface]
# Name = node1.example.tld
Address = 192.0.2.3/32
ListenPort = 51820
PrivateKey = localPrivateKeyAbcAbcAbc=
DNS = 1.1.1.1,8.8.8.8
Table = 12345
MTU = 1500
PreUp = /bin/example arg1 arg2 %i
PostUp = /bin/example arg1 arg2 %i
PreDown = /bin/example arg1 arg2 %i
PostDown = /bin/example arg1 arg2 %i

[Peer]
# Name = node2-node.example.tld
AllowedIPs = 192.0.2.1/24
Endpoint = node1.example.tld:51820
PublicKey = remotePublicKeyAbcAbcAbc=
PersistentKeepalive = 25
```

[Interface]

Defines the VPN settings for the local node.

Examples

- Node is a client that only routes traffic for itself and only exposes one IP

```
[Interface]
# Name = phone.example-vpn.dev
Address = 192.0.2.5/32
PrivateKey = <private key for phone.example-vpn.dev>
```

- Node is a public bounce server that can relay traffic to other peers and exposes route for entire VPN subnet

```
[Interface]
# Name = public-server1.example-vpn.tld
Address = 192.0.2.1/24
ListenPort = 51820
PrivateKey = <private key for public-server1.example-vpn.tld>
DNS = 1.1.1.1
```

Name

This is just a standard comment in INI syntax used to help keep track of which config section belongs to which node, it's completely ignored by WireGuard and has no effect on VPN behavior.

Address

Defines what address range the local node should route traffic for. Depending on whether the node is a simple client joining the VPN subnet, or a bounce server that's relaying traffic between multiple clients, this can be set to a single IP of the node itself (specified with CIDR notation), e.g. 192.0.2.3/32, or a range of IPv4/IPv6 subnets that the node can route traffic for.

Examples

- Node is a client that only routes traffic for itself
Address = 192.0.2.3/32
- Node is a public bounce server that can relay traffic to other peers
When the node is acting as the public bounce server, it should set this to be the entire subnet that it can route traffic, not just a single IP for itself.

```
Address = 192.0.2.1/24
```

- You can also specify multiple subnets or IPv6 subnets like so:
Address = 192.0.2.1/24,2001:DB8::/64

ListenPort

When the node is acting as a public bounce server, it should hardcode a port to listen for incoming VPN connections from the public internet. Clients not acting as relays should not set this value.

Examples

- Using default WireGuard port

```
ListenPort = 51820
```

- Using custom WireGuard port

```
ListenPort = 7000
```

PrivateKey

This is the private key for the local node, never shared with other servers.

All nodes must have a private key set, regardless of whether they are public bounce servers relaying traffic, or simple clients joining the VPN.

This key can be generated with `wg genkey > example.key`

Examples

```
PrivateKey = somePrivateKeyAbcdAbcdAbcdAbcd=
```

DNS

The DNS server(s) to announce to VPN clients via DHCP, most clients will use this server for DNS requests over the VPN, but clients can also override this value locally on their nodes

Examples

- The value can be left unconfigured to use system default DNS servers
- A single DNS server can be provided

```
DNS = 1.1.1.1
```

- or multiple DNS servers can be provided

```
DNS = 1.1.1.1,8.8.8.8
```

Table

Optionally defines which routing table to use for the WireGuard routes, not necessary to configure for most setups.

There are two special values: 'off' disables the creation of routes altogether, and 'auto' (the default) adds routes to the default table and enables special handling of default routes.

<https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8> (<https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8>)

Examples

```
Table = 1234
```

MTU

Optionally defines the maximum transmission unit (MTU, aka packet/frame size) to use when connecting to the peer, not necessary to configure for most setups.

The MTU is automatically determined from the endpoint addresses or the system default route, which is usually a sane choice.

<https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8> (<https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8>)

Examples

```
MTU = 1500
```

PreUp

Optionally run a command before the interface is brought up.

This option can be specified multiple times, with commands executed in the order they appear in the file.

Examples

- Add an IP route

```
PreUp = ip rule add ipproto tcp dport 22 table 1234
```

PostUp

Optionally run a command after the interface is brought up.

This option can appear multiple times, as with PreUp

Examples

- Read in a config value from a file or some command's output

```
PostUp = wg set %i private-key /etc/wireguard/wg0.key <(some command here)
```

- Log a line to a file

```
PostUp = echo "$(date +%s) WireGuard Started" >> /var/log/wireguard.log
```

- Hit a webhook on another server

```
PostUp = curl https://events.example.dev/wireguard/started/?key=abcdefg
```

- Add a route to the system routing table

```
PostUp = ip rule add ipproto tcp dport 22 table 1234
```

- Add an iptables rule to enable packet forwarding on the WireGuard interface

```
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j  
ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

- Force WireGuard to re-resolve IP address for peer domain

```
PostUp = resolvectl domain %i "~."; resolvectl dns %i 192.0.2.1; resolvectl  
dnssec %i yes
```

PreDown

Optionally run a command before the interface is brought down.

This option can appear multiple times, as with PreUp

Examples

- Log a line to a file

```
PostDown = echo "$(date +%s) WireGuard Going Down" >> /var/log/wireguard.log
```

- Hit a webhook on another server

```
PostDown = curl https://events.example.dev/wireguard/stopping/?key=abcdefg
```

PostDown

Optionally run a command after the interface is brought down.

This option can appear multiple times, as with PreUp

Examples

- Log a line to a file

```
PostDown = echo "$(date +%s) WireGuard Stopped" >> /var/log/wireguard.log
```

- Hit a webhook on another server

```
PostDown = curl https://events.example.dev/wireguard/stopped/?key=abcdefg
```

- Remove the iptables rule that forwards packets on the WireGuard interface

```
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j  
ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
```

[Peer]

Defines the VPN settings for a remote peer capable of routing traffic for one or more addresses (itself and/or other peers). Peers can be either a public bounce server that relays traffic to other peers, or a directly accessible client via LAN/internet that is not behind a NAT and only routes traffic for itself.

All clients must be defined as peers on the public bounce server. Simple clients that only route traffic for themselves, only need to define peers for the public relay, and any other nodes directly accessible. Nodes that are behind separate NATs should *not* be defined as peers outside of the public server config, as no direct route is available between separate NATs. Instead, nodes behind NATs should only define the public relay servers and other public clients as their peers, and should specify `AllowedIPs = 192.0.2.1/24` on the public server that accept routes and bounce traffic for the VPN subnet to the remote NAT-ed peers.

In summary, all nodes must be defined on the main bounce server. On client servers, only peers that are directly accessible from a node should be defined as peers of that node, any peers that must be relayed by a bounce server should be left out and will be handled by the relay server's catchall route.

In the configuration outlined in the docs below, a single server `public-server1` acts as the relay bounce server for a mix of publicly accessible and NAT-ed clients, and peers are configured on

each node accordingly:

- **in public-server1 wg0.conf (bounce server)**
[peer] list: public-server2 , home-server , laptop , phone
- **in public-server2 wg0.conf (simple public client)**
[peer] list: public-server1
- **in home-server wg0.conf (simple client behind NAT)**
[peer] list: public-server1 , public-server2
- **in laptop wg0.conf (simple client behind NAT)**
[peer] list: public-server1 , public-server2
- **in phone wg0.conf (simple client behind NAT)**
[peer] list: public-server1 , public-server2

Examples

- Peer is a simple public client that only routes traffic for itself

```
[Peer]
# Name = public-server2.example-vpn.dev
Endpoint = public-server2.example-vpn.dev:51820
PublicKey = <public key for public-server2.example-vpn.dev>
AllowedIPs = 192.0.2.2/32
```

- Peer is a simple client behind a NAT that only routes traffic for itself

```
[Peer]
# Name = home-server.example-vpn.dev
Endpoint = home-server.example-vpn.dev:51820
PublicKey = <public key for home-server.example-vpn.dev>
AllowedIPs = 192.0.2.3/32
```

- Peer is a public bounce server that can relay traffic to other peers

```
[Peer]
# Name = public-server1.example-vpn.tld
Endpoint = public-server1.example-vpn.tld:51820
PublicKey = <public key for public-server1.example-vpn.tld>
# routes traffic to itself and entire subnet of peers as bounce server
AllowedIPs = 192.0.2.1/24
PersistentKeepalive = 25
```

Name

This is just a standard comment in INI syntax used to help keep track of which config section belongs to which node, it's completely ignored by WireGuard and has no effect on VPN behavior.

Endpoint

Defines the publicly accessible address for a remote peer. This should be left out for peers behind a NAT or peers that don't have a stable publicly accessible IP:PORT pair. Typically, this only needs to be defined on the main bounce server, but it can also be defined on other public nodes with stable IPs like `public-server2` in the example config below.

Examples

- Endpoint is an IP address
`Endpoint = 123.124.125.126:51820` (IPv6 is also supported)
- Endpoint is a hostname/FQDN
`Endpoint = public-server1.example-vpn.tld:51820`

AllowedIPs

This defines the IP ranges for which a peer will route traffic. On simple clients, this is usually a single address (the VPN address of the simple client itself). For bounce servers this will be a range of the IPs or subnets that the relay server is capable of routing traffic for. Multiple IPs and subnets may be specified using comma-separated IPv4 or IPv6 CIDR notation (from a single /32 or /128 address, all the way up to `0.0.0.0/0` and `::/0` to indicate a default route to send all internet and VPN traffic through that peer). This option may be specified multiple times.

When deciding how to route a packet, the system chooses the most specific route first, and falls back to broader routes. So for a packet destined to `192.0.2.3`, the system would first look for a peer advertising `192.0.2.3/32` specifically, and would fall back to a peer advertising `192.0.2.1/24` or a larger range like `0.0.0.0/0` as a last resort.

Examples

- peer is a simple client that only accepts traffic to/from itself
`AllowedIPs = 192.0.2.3/32`
- peer is a relay server that can bounce VPN traffic to all other peers
`AllowedIPs = 192.0.2.1/24`
- peer is a relay server that bounces all internet & VPN traffic (like a proxy), including IPv6
`AllowedIPs = 0.0.0.0/0, ::/0`
- peer is a relay server that routes to itself and only one other peer
`AllowedIPs = 192.0.2.3/32, 192.0.2.4/32`
- peer is a relay server that routes to itself and all nodes on its local LAN
`AllowedIPs = 192.0.2.3/32, 192.168.1.1/24`

PublicKey

This is the public key for the remote node, shareable with all peers.

All nodes must have a public key set, regardless of whether they are public bounce servers

relaying traffic, or simple clients joining the VPN.

This key can be generated with `wg pubkey < example.key > example.key.pub` .
(see above for how to generate the private key `example.key`)

Examples

```
PublicKey = somePublicKeyAbcdAbcdAbcdAbcd=
```

PersistentKeepalive

If the connection is going from a NAT-ed peer to a public peer, the node behind the NAT must regularly send an outgoing ping in order to keep the bidirectional connection alive in the NAT router's connection table.

Examples

- local public node to remote public node
This value should be left undefined as persistent pings are not needed.
- local public node to remote NAT-ed node
This value should be left undefined as it's the client's responsibility to keep the connection alive because the server cannot reopen a dead connection to the client if it times out.
- local NAT-ed node to remote public node
`PersistentKeepalive = 25` this will send a ping to every 25 seconds keeping the connection open in the local NAT router's connection table.

Advanced Topics

IPv6

The examples in these docs primarily use IPv4, but WireGuard natively supports IPv6 CIDR notation and addresses everywhere that it supports IPv4, simply add them as you would any other subnet range or address.

Example

```
[Interface]
AllowedIps = 192.0.2.3/24, 2001:DB8::/64

[Peer]
...
AllowedIPs = 0.0.0.0/0, ::/0
```

Forwarding All Traffic

If you want to forward *all* internet traffic through the VPN, and not just use it as a server-to-

server subnet, you can add `0.0.0.0/0, ::/0` to the `AllowedIPs` definition of the peer you want to pipe your traffic through.

Make sure to also specify an IPv6 catchall even when only forwarding IPv4 traffic in order to avoid leaking IPv6 packets outside the VPN, see:

https://www.reddit.com/r/WireGuard/comments/b0m5g2/ipv6_leaks_psa_for_anyone_here_using_wireguard_to/ (https://www.reddit.com/r/WireGuard/comments/b0m5g2/ipv6_leaks_psa_for_anyone_here_using_wireguard_to/)

Example

```
[Interface]
# Name = phone.example-vpn.dev
Address = 192.0.2.3/32
PrivateKey = <private key for phone.example-vpn.dev>

[Peer]
# Name = public-server1.example-vpn.dev
PublicKey = <public key for public-server1.example-vpn.dev>
Endpoint = public-server1.example-vpn.dev:51820
AllowedIPs = 0.0.0.0/0, ::/0
```

NAT To NAT Connections

WireGuard can sometimes natively make connections between two clients behind NATs without the need for a public relay server, but in most cases this is not possible. NAT-to-NAT connections are only possible if at least one host has a stable, publicly-accessible IP address:port pair that can be hardcoded ahead of time, whether that's using a FQDN updated with Dynamic DNS, or a static public IP with a non-randomized NAT port opened by outgoing packets, anything works as long as all peers can communicate it beforehand and it doesn't change once the connection is initiated.

A known port and address need to be configured ahead of time because WireGuard doesn't have a signalling layer or public STUN servers that can be used to search for other hosts dynamically. WebRTC is an example of a protocol that can dynamically configure a connection between two NATs, but it does this by using an out-of-band signaling server to detect the IP:port combo of each host. WireGuard doesn't have this, so it only works with a hardcoded `Endpoint + ListenPort` (and `PersistentKeepalive` so it doesn't drop after inactivity).

Learn more from Tailscale's bible of NAT traversal: <https://tailscale.com/blog/how-nat-traversal-works/> (<https://tailscale.com/blog/how-nat-traversal-works/>)

Requirements for NAT-to-NAT setups

- At least one peer has to have a hardcoded, directly-accessible `Endpoint` defined. If they're both behind NATs without stable IP addresses, then you'll need to use Dynamic DNS or another solution to have a stable, publicly accessible domain/IP for at least one peer
- At least one peer has to have a hardcoded `UDP ListenPort` defined, and it's NAT router must not do UDP source port randomization, otherwise return packets will be sent to the

hardcoded `ListenPort` and dropped by the router, instead of using the random port assigned by the NAT on the outgoing packet

- All NAT'ed peers must have `PersistentKeepalive` enabled on all other peers, so that they continually send outgoing pings to keep connections persisted in their NAT's routing table

The hole-punching connection process

1. Peer1 sends a UDP packet to Peer2, it's rejected Peer2's NAT router immediately, but that's ok, the only purpose was to get Peer1's NAT to start forwarding any expected UDP responses back to Peer1 behind its NAT
2. Peer2 sends a UDP packet to Peer1, it's accepted and forwarded to Peer1 as Peer1's NAT server is already expecting responses from Peer2 because of the initial outgoing packet
3. Peer1 sends a UDP response to Peer2's packet, it's accepted and forwarded by Peer2's NAT server as it's also expecting responses because of the initial outgoing packet

This process of sending an initial packet that gets rejected, then using the fact that the router has now created a forwarding rule to accept responses is called "UDP hole-punching".

When you send a UDP packet out, the router (usually) creates a temporary rule mapping your source address and port to the destination address and port, and vice versa. UDP packets returning from the destination address and port (and no other) are passed through to the original source address and port (and no other). This is how most UDP applications function behind NATs (e.g. BitTorrent, Skype, etc). This rule will timeout after some minutes of inactivity, so the client behind the NAT must send regular outgoing packets to keep it open (see `PersistentKeepalive`).

Getting this to work when both end-points are behind NATs or firewalls requires that both end-points send packets to each-other at about the same time. This means that both sides need to know each-other's public IP addresses and port numbers ahead of time, in WireGuard's case this is achieved by hard-coding pre-defined ports for both sides in `wg0.conf` .

Drawbacks and limitations

As of 2019, many of the old hole-punching methods used that used to work are no longer effective. One example was a novel method pioneered by `pwnat` (<https://github.com/samyk/pwnat>) that faked an ICMP Time Exceeded response from outside the NAT to get a packet back through to a NAT'ed peer, thereby leaking its own source port. Hardcoding UDP ports and public IPs for both sides of a NAT-to-NAT connection (as described above) still works on a small percentage of networks. Generally the more "enterprisey" a network is, the less likely you'll be able to hole punch public UDP ports (commercial public Wi-Fi and cell data NATs often don't work for example).

Source port randomization

NAT-to-NAT connections are not possible if all endpoints are behind NAT's with strict UDP source port randomization (e.g. most cellular data networks). Since neither side is able to hardcode a `ListenPort` and guarantee that their NAT will accept traffic on that port after the outgoing ping, you cannot coordinate a port for the initial hole-punch between peers and

connections will fail. For this reason, you generally cannot do phone-to-phone connections on LTE/3g networks, but you might be able to do phone-to-office or phone-to-home where the office or home has a stable public IP and doesn't do source port randomization.

Using a signaling server

NAT-to-NAT connections from behind NATs with strict source-port randomization is possible, you just need a signaling server to tell each side the other's IP:port tuple. Here are a few implementations that achieve this with WireGuard:

- <https://github.com/takutakahashi/wg-connect> (<https://github.com/takutakahashi/wg-connect>)
- <https://git.zx2c4.com/wireguard-tools/tree/contrib/nat-hole-punching/> (<https://git.zx2c4.com/wireguard-tools/tree/contrib/nat-hole-punching/>)
- <https://github.com/jwhited/wgsd> (<https://github.com/jwhited/wgsd>)

Dynamic IP addresses

Many users report having to restart WireGuard whenever a dynamic IP changes, as it only resolves hostnames on startup. To force WireGuard to re-resolve dynamic DNS `Endpoint` hostnames more often, you may want to use a `Postup` hook to restart WireGuard every few minutes or hours.

Testing it out

You can see if a hole-punching setup is feasible by using netcat on the client and server to see what ports and connection order work to get a bidirectional connection open: run `nc -v -u -p 51820 <address of peer2> 51820` (on peer1) and `nc -v -u -l 0.0.0.0 51820` (on peer2), then type in both windows to see if you can get bidirectional traffic going. If it doesn't work regardless of which peer sends the initial packet, then WireGuard won't be able to work between the peers without a public relay server.

NAT-to-NAT connections are often more unstable and have other limitations, which is why having a fallback public relay server is still advised.

Further reading

- <https://github.com/samyk/pwnat> (<https://github.com/samyk/pwnat>)
- https://en.wikipedia.org/wiki/UDP_hole_punching (https://en.wikipedia.org/wiki/UDP_hole_punching)
- <https://stackoverflow.com/questions/8892142/udp-hole-punching-algorithm> (<https://stackoverflow.com/questions/8892142/udp-hole-punching-algorithm>)
- <https://stackoverflow.com/questions/12359502/udp-hole-punching-not-going-through-on-3g> (<https://stackoverflow.com/questions/12359502/udp-hole-punching-not-going-through-on-3g>)
- <https://stackoverflow.com/questions/11819349/udp-hole-punching-not-possible-with-mobile-provider> (<https://stackoverflow.com/questions/11819349/udp-hole-punching-not-possible-with-mobile-provider>)
- <https://github.com/WireGuard/WireGuard/tree/master/contrib/examples/nat-hole-punching> (<https://github.com/WireGuard/WireGuard/tree/master/contrib/examples/nat-hole-punching>)
- <https://staalraad.github.io/2017/04/17/nat-to-nat-with-wireguard/> (<https://staalraad.github.io/2017/04/17/nat-to-nat-with-wireguard/>)

/2017/04/17/nat-to-nat-with-wireguard/)

- <https://golb.hplar.ch/2019/01/expose-server-vpn.html> (<https://golb.hplar.ch/2019/01/expose-server-vpn.html>)
- <https://www.jordanwhited.com/posts/wireguard-endpoint-discovery-nat-traversal/> (<https://www.jordanwhited.com/posts/wireguard-endpoint-discovery-nat-traversal/>)

Example

Peer1:

```
[Interface]
...
ListenPort 12000

[Peer]
...
Endpoint = peer2.example-vpn.dev:12000
PersistentKeepalive = 25
```

Peer2:

```
[Interface]
...
ListenPort 12000

[Peer]
...
Endpoint = peer1.example-vpn.dev:12000
PersistentKeepalive = 25
```

Dynamic IP Allocation

Note: this section is about dynamic peer IPs within the VPN subnet, not dynamic public Endpoint addresses.

Dynamic allocation of peer IPs (instead of only having fixed peers) is being developed, the WIP implementation is available here:

<https://github.com/WireGuard/wg-dynamic> (<https://github.com/WireGuard/wg-dynamic>)

You can also build a dynamic allocation system yourself by reading in IP values from files at runtime by using `PostUp` (see below).

Example

```
[Interface]
...
PostUp = wg set %i allowed-ips /etc/wireguard/wg0.key <(some command)
```

Other WireGuard Implementations

- <https://git.zx2c4.com/wireguard-go/about/> (<https://git.zx2c4.com/wireguard-go/about/>)
A compliant userland WireGuard implementation written in Go.
- <https://git.zx2c4.com/wireguard-rs/about/> (<https://git.zx2c4.com/wireguard-rs/about/>)
An incomplete, insecure userspace implementation of WireGuard written in Rust (not ready for the public).
- <https://git.zx2c4.com/wireguard-hs/about/> (<https://git.zx2c4.com/wireguard-hs/about/>)
An incomplete, insecure userspace implementation of WireGuard written in Haskell (not ready for the public).
- <https://github.com/cloudflare/boringtun> (<https://github.com/cloudflare/boringtun>)
A non-compliant, independent WireGuard implementation written in Rust (a separate fork written by CloudFlare).
See <https://blog.cloudflare.com/boringtun-userspace-wireguard-rust/> (<https://blog.cloudflare.com/boringtun-userspace-wireguard-rust/>)
- Platform-specific WireGuard apps
<https://git.zx2c4.com/wireguard-ios/about/> (<https://git.zx2c4.com/wireguard-ios/about/>)
<https://git.zx2c4.com/wireguard-android/about/> (<https://git.zx2c4.com/wireguard-android/about/>)
<https://git.zx2c4.com/wireguard-windows/about/> (<https://git.zx2c4.com/wireguard-windows/about/>)

All of the userspace implementations are slower than the native C version that runs in kernel-land, but provide other benefits by running in userland (e.g. easier containerization, compatibility, etc.).

WireGuard Setup Tools

These are some GUI and CLI tools that wrap WireGuard to assist with config, deployment, key management, and connection.

- <https://github.com/seashell/drago> (<https://github.com/seashell/drago>)
- <https://github.com/vx3r/wg-gen-web> (<https://github.com/vx3r/wg-gen-web>)
- <https://github.com/subspacecloud/subspace> (<https://github.com/subspacecloud/subspace>)
- <https://github.com/corrad1nho/qomui> (<https://github.com/corrad1nho/qomui>)
- <https://github.com/max-moser/network-manager-wireguard> (<https://github.com/max-moser/network-manager-wireguard>)
- <https://github.com/psychomb/wireguard-tools> (<https://github.com/psychomb/wireguard-tools>)
- <https://github.com/its0x08/wg-install> (<https://github.com/its0x08/wg-install>)
- <https://github.com/sowbug/mkwgconf> (<https://github.com/sowbug/mkwgconf>)
- https://github.com/brittson/wireguard_config_maker (https://github.com/brittson/wireguard_config_maker)
- <https://github.com/SirTofski/WireGuard-Ligase/> (<https://github.com/SirTofski/WireGuard-Ligase/>)
- <https://pypi.org/project/wireguard-p2p/> (<https://pypi.org/project/wireguard-p2p/>)
- <https://github.com/trailofbits/algo> (<https://github.com/trailofbits/algo>)

- <https://github.com/StreisandEffect/streisand> (<https://github.com/StreisandEffect/streisand>)
- <https://www.veeam.com/blog/veeam-pn-v2-wireguard.html> (<https://www.veeam.com/blog/veeam-pn-v2-wireguard.html>)
- <https://github.com/wg-dashboard/wg-dashboard> (<https://github.com/wg-dashboard/wg-dashboard>)
- <https://www.wireguardconfig.com> (<https://www.wireguardconfig.com>)
- <https://github.com/complexorganizations/wireguard-manager> (<https://github.com/complexorganizations/wireguard-manager>)
- <https://github.com/influxdata/wirey> (<https://github.com/influxdata/wirey>)
- <https://github.com/apognu/wgctl> (<https://github.com/apognu/wgctl>)
- <https://github.com/naggie/dsnet> (<https://github.com/naggie/dsnet>)
- <https://github.com/perara/wg-manager> (<https://github.com/perara/wg-manager>)
- <https://github.com/pivpn/pivpn> (<https://github.com/pivpn/pivpn>)
- <https://github.com/BrunIF/wg-CCG> (<https://github.com/BrunIF/wg-CCG>)

Config Shortcuts

Credit for these shortcuts goes to:

<https://www.ericlight.com/new-things-i-didnt-know-about-wireguard.html> (<https://www.ericlight.com/new-things-i-didnt-know-about-wireguard.html>)

Sharing a single peers.conf file

WireGuard will ignore a peer whose public key matches the interface's private key. So you can distribute a single list of peers everywhere, and only define the `[Interface]` separately on each server.

See: <https://lists.zx2c4.com/pipermail/wireguard/2018-December/003703.html>

(<https://lists.zx2c4.com/pipermail/wireguard/2018-December/003703.html>)

You can combine this with `wg addconf` like this:

- Each peer has its own `/etc/wireguard/wg0.conf` file, which only contains its `[Interface]` section.
- Each peer also has a shared `/etc/wireguard/peers.conf` file, which contains all the peers.
- The `wg0.conf` file also has a `PostUp` hook: `PostUp = wg addconf /etc/wireguard/peers.conf`.

It's up to you to decide how you want to share the `peers.conf`, be it via a proper orchestration platform, something much more pedestrian like Dropbox, or something kinda wild like Ceph. I dunno, but it's pretty great that you can just wildly fling a peer section around, without worrying whether it's the same as the interface.

Setting config values from files or command outputs

You can set config values from arbitrary commands or by reading in values from files, this makes key management and deployment much easier as you can read in keys at runtime from a 3rd

party service like Kubernetes Secrets or AWS KMS.

See: <https://lists.zx2c4.com/pipermail/wireguard/2018-December/003702.html>

(<https://lists.zx2c4.com/pipermail/wireguard/2018-December/003702.html>)

Example

You can read in a file as the `PrivateKey` by doing something like:

```
PostUp = wg set %i private-key /etc/wireguard/wg0.key <(some command)
```

Containerization

WireGuard can be run in Docker with varying degrees of ease. In the simplest case,

`--privileged` and `--cap-add=all` arguments can be added to the docker commands to enable the loading of the kernel module.

Setups can get somewhat complex and are highly dependent on what you're trying to achieve. You can have WireGuard itself run in a container and expose a network interface to the host, or you can have WireGuard running on the host exposing an interface to specific containers.

See below for an example of a Docker container `vpn_test` routing all its traffic through a WireGuard relay server.

Example Relay Server Setup

```
version: '3'

services:
  wireguard:
    image: linuxserver/wireguard
    ports:
      - 51820:51820/udp
    cap_add:
      - NET_ADMIN
      - SYS_MODULE
    volumes:
      - /lib/modules:/lib/modules
      - ./wg0.conf:/config/wg0.conf:ro
```

wg0.conf :

```
[Interface]
# Name = relay1.wg.example.com
Address = 192.0.2.1/24
ListenPort = 51820
PrivateKey = oJpRt20q27vIB5/UVb7BRqCwad2YMReQgH5tlxz8YmI=
DNS = 1.1.1.1,8.8.8.8
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0

[Peer]
# Name = peer1.wg.example.com
PublicKey = I+hXRAJOG/UE2IQvIHSou2zTgkUyPve2pzvHTnd/2Gg=
AllowedIPs = 192.0.2.2/32
```

Example Client Container Setup

In this example *all* the traffic from inside the `speedtest` container will go through the wireguard VPN.

To only route some traffic, replace `0.0.0.0/0` in `wg0.conf` below with the subnet ranges you want to route via the VPN.

docker-compose.yml :

```
version: '3'

services:
  wireguard:
    image: linuxserver/wireguard
    cap_add:
      - NET_ADMIN
      - SYS_MODULE
    volumes:
      - /lib/modules:/lib/modules
      - ./wg0.conf:/config/wg0.conf:ro

  vpn_test:
    image: curlimages/curl
    entrypoint: curl -s http://whatismyip.akamai.com/
    network_mode: 'service:wireguard'
```

wg0.conf :

[Interface]

Name = peer1.wg.example.com

Address = 192.0.2.2/32

PrivateKey = YCW76edD4W7nZrPbWZxPZhcS32CsBLIi1sEhsV/sgk8=

DNS = 1.1.1.1,8.8.8.8

[Peer]

Name = relay1.wg.example.com

Endpoint = relay1.wg.example.com:51820

PublicKey = zJNKewtL3gcHdG62V3GaBkErFtapJWsAx+2um0c0B1s=

AllowedIPs = 192.0.2.1/24,0.0.0.0/0

PersistentKeepalive = 21

For more details see the Further Reading: Docker section below.

Further Reading

Reference Docs

- <https://www.wireguard.com/install/#installation> (<https://www.wireguard.com/install/#installation>)
- <https://git.zx2c4.com/WireGuard/about/src/tools/man/wg.8> (<https://git.zx2c4.com/WireGuard/about/src/tools/man/wg.8>)
- <https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8> (<https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8>)
- <https://wiki.archlinux.org/index.php/WireGuard> (<https://wiki.archlinux.org/index.php/WireGuard>)
- <https://wiki.debian.org/Wireguard#Configuration> (<https://wiki.debian.org/Wireguard#Configuration>)

Tutorials

- <https://www.wireguard.com/quickstart/> (<https://www.wireguard.com/quickstart/>)
- <https://www.stavros.io/posts/how-to-configure-wireguard/> (<https://www.stavros.io/posts/how-to-configure-wireguard/>)
- <https://nbsoftsolutions.com/blog/wireguard-vpn-walkthrough> (<https://nbsoftsolutions.com/blog/wireguard-vpn-walkthrough>)
- <https://networkhobo.com/building-a-wireguard-router/> (<https://networkhobo.com/building-a-wireguard-router/>)
- <https://proprivacy.com/guides/wireguard-hands-on-guide> (<https://proprivacy.com/guides/wireguard-hands-on-guide>)
- <https://angristan.xyz/how-to-setup-vpn-server-wireguard-nat-ipv6/> (<https://angristan.xyz/how-to-setup-vpn-server-wireguard-nat-ipv6/>)
- <https://medium.com/@headquartershq/setting-up-wireguard-on-a-mac-8a121bfe9d86> (<https://medium.com/@headquartershq/setting-up-wireguard-on-a-mac-8a121bfe9d86>)
- <https://grh.am/2018/wireguard-setup-guide-for-ios/> (<https://grh.am/2018/wireguard-setup-guide-for-ios/>)
- <https://techcrunch.com/2018/07/28/how-i-made-my-own-wireguard-vpn-server/>

- (<https://techcrunch.com/2018/07/28/how-i-made-my-own-wireguard-vpn-server/>)
- <https://www.ckn.io/blog/2017/11/14/wireguard-vpn-typical-setup/> (<https://www.ckn.io/blog/2017/11/14/wireguard-vpn-typical-setup/>)
- <https://jrs-s.net/2018/08/05/routing-between-wg-interfaces-with-wireguard/> (<https://jrs-s.net/2018/08/05/routing-between-wg-interfaces-with-wireguard/>)
- <https://www.stavros.io/posts/how-to-configure-wireguard/> (<https://www.stavros.io/posts/how-to-configure-wireguard/>)
- <https://vincent.bernat.ch/en/blog/2018-route-based-vpn-wireguard> (<https://vincent.bernat.ch/en/blog/2018-route-based-vpn-wireguard>)
- <https://staaldrad.github.io/2017/04/17/nat-to-nat-with-wireguard> (<https://staaldrad.github.io/2017/04/17/nat-to-nat-with-wireguard>)
- <https://try.popho.be/wg.html> (<https://try.popho.be/wg.html>)
- <https://docs.artemix.org/sysadmin/wireguard-management/> (<https://docs.artemix.org/sysadmin/wireguard-management/>)
- <https://github.com/adrianmihalko/raspberrypiwireguard> (<https://github.com/adrianmihalko/raspberrypiwireguard>)
- <https://www.ericlight.com/wireguard-part-one-installation.html> (<https://www.ericlight.com/wireguard-part-one-installation.html>)
- <https://www.ericlight.com/wireguard-part-two-vpn-routing.html> (<https://www.ericlight.com/wireguard-part-two-vpn-routing.html>)
- <https://www.ericlight.com/wireguard-part-three-troubleshooting.html> (<https://www.ericlight.com/wireguard-part-three-troubleshooting.html>)
- https://wiki.dd-wrt.com/wiki/index.php/The_Easiest_Tunnel_Ever (https://wiki.dd-wrt.com/wiki/index.php/The_Easiest_Tunnel_Ever)
- https://www.reddit.com/r/pihole/comments/bnihyz/guide_how_to_install_wireguard_on_a_raspberry_pi/ (https://www.reddit.com/r/pihole/comments/bnihyz/guide_how_to_install_wireguard_on_a_raspberry_pi/)
- <https://jwillmer.de/blog/tutorial/wireguard-proxy-configuration> (<https://jwillmer.de/blog/tutorial/wireguard-proxy-configuration>)
- <https://www.maths.tcd.ie/~fionn/misc/wireguard.php> (<https://www.maths.tcd.ie/~fionn/misc/wireguard.php>)
- <https://www.linode.com/docs/networking/vpn/set-up-wireguard-vpn-on-debian/> (<https://www.linode.com/docs/networking/vpn/set-up-wireguard-vpn-on-debian/>)
- <https://golb.hplar.ch/2019/01/expose-server-vpn.html> (<https://golb.hplar.ch/2019/01/expose-server-vpn.html>)
- <https://medium.com/@jmarhee/configuring-and-managing-routes-between-multiple-networks-with-wireguard-61ad995c887c> (<https://medium.com/@jmarhee/configuring-and-managing-routes-between-multiple-networks-with-wireguard-61ad995c887c>)
- <https://stanislas.blog/2019/01/how-to-setup-vpn-server-wireguard-nat-ipv6/> (<https://stanislas.blog/2019/01/how-to-setup-vpn-server-wireguard-nat-ipv6/>)

Papers, Articles, and Talks

- <https://www.wireguard.com/papers/wireguard.pdf> (<https://www.wireguard.com/papers/wireguard.pdf>)
- <https://www.wireguard.com/presentations/> (<https://www.wireguard.com/presentations/>)
- <https://eprint.iacr.org/2018/080.pdf> (<https://eprint.iacr.org/2018/080.pdf>)
- <https://courses.csail.mit.edu/6.857/2018/project/He-Xu-Xu-WireGuard.pdf>
(<https://courses.csail.mit.edu/6.857/2018/project/He-Xu-Xu-WireGuard.pdf>)
- <https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen/> (<https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen/>)
- <https://www.wireguard.com/talks/blackhat2018-slides.pdf> (<https://www.wireguard.com/talks/blackhat2018-slides.pdf>)

Related Projects

- <https://github.com/complexorganizations/wireguard-manager> (<https://github.com/complexorganizations/wireguard-manager>)
- <https://github.com/subspacecloud/subspace> (<https://github.com/subspacecloud/subspace>)
- <https://github.com/trailofbits/algo> (<https://github.com/trailofbits/algo>)
- <https://github.com/StreisandEffect/streisand> (<https://github.com/StreisandEffect/streisand>)
- <https://github.com/its0x08/wg-install> (<https://github.com/its0x08/wg-install>)
- <https://github.com/sowbug/mkwgconf> (<https://github.com/sowbug/mkwgconf>)
- https://github.com/brittson/wireguard_config_maker (https://github.com/brittson/wireguard_config_maker)
- <https://github.com/SirTofski/WireGuard-Ligase/> (<https://github.com/SirTofski/WireGuard-Ligase/>)
- <https://pypi.org/project/wireguard-p2p/> (<https://pypi.org/project/wireguard-p2p/>)
- <https://github.com/cloudflare/boringtun> (<https://github.com/cloudflare/boringtun>)
- <https://git.zx2c4.com/wireguard-go/about/> (<https://git.zx2c4.com/wireguard-go/about/>)
- <https://github.com/WireGuard/wg-dynamic> (<https://github.com/WireGuard/wg-dynamic>)
- <https://github.com/WireGuard/wireguard-ios> (<https://github.com/WireGuard/wireguard-ios>)
- <https://github.com/WireGuard/wireguard-windows> (<https://github.com/WireGuard/wireguard-windows>)
- <https://github.com/WireGuard/wireguard-rs> (<https://github.com/WireGuard/wireguard-rs>)
- <https://github.com/WireGuard/wireguard-go> (<https://github.com/WireGuard/wireguard-go>)
- <https://www.veeam.com/blog/veeam-pn-v2-wireguard.html> (<https://www.veeam.com/blog/veeam-pn-v2-wireguard.html>)
- <https://github.com/wg-dashboard/wg-dashboard> (<https://github.com/wg-dashboard/wg-dashboard>)
- <https://wirtbot.com> (<https://wirtbot.com>)
- <https://github.com/seashell/drago> (<https://github.com/seashell/drago>)
- <https://www.wireguardconfig.com> (<https://www.wireguardconfig.com>)
- <https://github.com/angristan/wireguard-install> (<https://github.com/angristan/wireguard-install>)
- <https://github.com/complexorganizations/wireguard-manager> (<https://github.com/complexorganizations/wireguard-manager>)
- <https://github.com/influxdata/wirey> (<https://github.com/influxdata/wirey>)
- <https://github.com/apognu/wgctl> (<https://github.com/apognu/wgctl>)

- <https://github.com/tailscale/tailscale> (<https://github.com/tailscale/tailscale>)
- <https://github.com/pivpn/pivpn> (<https://github.com/pivpn/pivpn>)
- <https://github.com/jwhited/wgsd> (<https://github.com/jwhited/wgsd>)

Docker

- <https://blog.jessfraz.com/post/installing-and-using-wireguard/> (<https://blog.jessfraz.com/post/installing-and-using-wireguard/>)
- <https://codeopolis.com/posts/installing-wireguard-in-docker/> (<https://codeopolis.com/posts/installing-wireguard-in-docker/>)
- <http://tiven.wang/articles/wireguard-setup-server-in-docker/> (<http://tiven.wang/articles/wireguard-setup-server-in-docker/>)
- <https://github.com/activeeos/wireguard-docker> (<https://github.com/activeeos/wireguard-docker>)
- <https://github.com/cmulk/wireguard-docker> (<https://github.com/cmulk/wireguard-docker>)
- <https://github.com/ironhalik/docker-wireguard> (<https://github.com/ironhalik/docker-wireguard>)
- <https://github.com/linuxserver/docker-wireguard> (<https://github.com/linuxserver/docker-wireguard>)
- <https://github.com/squat/kilo> (<https://github.com/squat/kilo>)
- <https://github.com/gravitational/wormhole> (<https://github.com/gravitational/wormhole>)
- <https://medium.com/@mdp/securing-docker-with-wireguard-82ad45004f4d> (<https://medium.com/@mdp/securing-docker-with-wireguard-82ad45004f4d>)
- <https://nbsoftsolutions.com/blog/leaning-on-algo-to-route-docker-traffic-through-wireguard> (<https://nbsoftsolutions.com/blog/leaning-on-algo-to-route-docker-traffic-through-wireguard>)
- <https://nbsoftsolutions.com/blog/routing-select-docker-containers-through-wireguard-vpn> (<https://nbsoftsolutions.com/blog/routing-select-docker-containers-through-wireguard-vpn>)
- <https://www.net.in.tum.de/fileadmin/bibtex/publications/theses/2018-pudelko-vpn-performance.pdf> (<https://www.net.in.tum.de/fileadmin/bibtex/publications/theses/2018-pudelko-vpn-performance.pdf>)
- <https://www.wireguard.com/#ready-for-containers> (<https://www.wireguard.com/#ready-for-containers>)
- <https://discuss.linuxcontainers.org/t/solved-wireguard-in-macvlan-container-on-ubuntu-18-04/4445> (<https://discuss.linuxcontainers.org/t/solved-wireguard-in-macvlan-container-on-ubuntu-18-04/4445>)
- https://www.reddit.com/r/WireGuard/comments/gdhcej/trouble_tunneling_docker_containers_through_a/ (https://www.reddit.com/r/WireGuard/comments/gdhcej/trouble_tunneling_docker_containers_through_a/)
- <https://forums.unraid.net/topic/91367-partially-working-wireguard-docker/> (<https://forums.unraid.net/topic/91367-partially-working-wireguard-docker/>)
- <https://saasbootstrap.com/how-to-setup-a-vpn-with-wireguard-that-only-routes-traffic-from-a-specific-docker-container-or-specific-ip/> (<https://saasbootstrap.com/how-to-setup-a-vpn-with-wireguard-that-only-routes-traffic-from-a-specific-docker-container-or-specific-ip/>)

Other

- <https://blog.cloudflare.com/boringtun-userspace-wireguard-rust/> (<https://blog.cloudflare.com/boringtun-userspace-wireguard-rust/>)
- <https://jrs-s.net/category/open-source/wireguard/> (<https://jrs-s.net/category/open-source/wireguard/>)

- <https://restoreprivacy.com/openssl-ipsec-wireguard-l2tp-ikev2-protocols/>
(<https://restoreprivacy.com/openssl-ipsec-wireguard-l2tp-ikev2-protocols/>)
- <https://restoreprivacy.com/wireguard/> (<https://restoreprivacy.com/wireguard/>)
- <https://www.ericlight.com/new-things-i-didnt-know-about-wireguard.html>
(<https://www.ericlight.com/new-things-i-didnt-know-about-wireguard.html>)
- <https://www.ericlight.com/tag/wireguard.html> (<https://www.ericlight.com/tag/wireguard.html>)
- <https://www.linode.com/docs/networking/vpn/set-up-wireguard-vpn-on-ubuntu/>
(<https://www.linode.com/docs/networking/vpn/set-up-wireguard-vpn-on-ubuntu/>)
- https://www.reddit.com/r/linux/comments/9bnowo/wireguard_benchmark_between_two_servers_with_10/
(https://www.reddit.com/r/linux/comments/9bnowo/wireguard_benchmark_between_two_servers_with_10/)
- <https://www.wireguard.com/netns/> (<https://www.wireguard.com/netns/>)
- <https://www.wireguard.com/performance/> (<https://www.wireguard.com/performance/>)
- <https://blogs.gnome.org/thaller/2019/03/15/wireguard-in-networkmanager/>
(<https://blogs.gnome.org/thaller/2019/03/15/wireguard-in-networkmanager/>)
- <https://github.com/max-moser/network-manager-wireguard> (<https://github.com/max-moser/network-manager-wireguard>)
- <https://blog.linuxserver.io/2019/11/24/connect-an-ubuntu-client-to-opnsense-wireguard-tunnel-with-a-gui-toggle-in-gnome/>
(<https://blog.linuxserver.io/2019/11/24/connect-an-ubuntu-client-to-opnsense-wireguard-tunnel-with-a-gui-toggle-in-gnome/>)

Discussions

- <https://www.reddit.com/r/WireGuard> (<https://www.reddit.com/r/WireGuard>)
- <https://lists.zx2c4.com/mailman/listinfo/wireguard> (<https://lists.zx2c4.com/mailman/listinfo/wireguard>)
- https://www.reddit.com/r/VPN/comments/a914mr/can_you_explain_the_difference_between_openvpn/
(https://www.reddit.com/r/VPN/comments/a914mr/can_you_explain_the_difference_between_openvpn/)
- https://www.reddit.com/r/WireGuard/comments/b0m5g2/ipv6_leaks_psa_for_anyone_here_using_wireguard_to/?utm_source=reddit&utm_medium=usertext&utm_name=WireGuard&utm_content=t1_ep8tv0o
(https://www.reddit.com/r/WireGuard/comments/b0m5g2/ipv6_leaks_psa_for_anyone_here_using_wireguard_to/?utm_source=reddit&utm_medium=usertext&utm_name=WireGuard&utm_content=t1_ep8tv0o)
- https://www.reddit.com/r/VPN/comments/au4owb/how_secure_is_wireguard_vpn_protocol/
(https://www.reddit.com/r/VPN/comments/au4owb/how_secure_is_wireguard_vpn_protocol/)
- https://www.reddit.com/r/WireGuard/comments/ap33df/wireguard_what_is_so_special_about_it_and_why/
(https://www.reddit.com/r/WireGuard/comments/ap33df/wireguard_what_is_so_special_about_it_and_why/)
- https://www.reddit.com/r/VPN/comments/9hgs2x/what_is_the_difference_between_wireguard_openvpn/
(https://www.reddit.com/r/VPN/comments/9hgs2x/what_is_the_difference_between_wireguard_openvpn/)
- https://www.reddit.com/r/WireGuard/comments/d3thxp/port_forwarding_on_the_router_with_wireguard_is/
(https://www.reddit.com/r/WireGuard/comments/d3thxp/port_forwarding_on_the_router_with_wireguard_is/)

/d3thxp/port_forwarding_on_the_router_with_wireguard_is/)

- https://www.reddit.com/r/privacytoolsIO/comments/8l0vxt/what_do_you_think_guys_of_wireguard/ (https://www.reddit.com/r/privacytoolsIO/comments/8l0vxt/what_do_you_think_guys_of_wireguard/)
- <https://community.ui.com/questions/Edgerouter-with-remote-Wireguard-access-issue/03e4f2e2-3871-437f-8632-3c5c7fb1c7a4> (<https://community.ui.com/questions/Edgerouter-with-remote-Wireguard-access-issue/03e4f2e2-3871-437f-8632-3c5c7fb1c7a4>)
- <https://discuss.linuxcontainers.org/t/solved-wireguard-in-macvlan-container-on-ubuntu-18-04/4445> (<https://discuss.linuxcontainers.org/t/solved-wireguard-in-macvlan-container-on-ubuntu-18-04/4445>)
- <https://news.ycombinator.com/item?id=20036194> (<https://news.ycombinator.com/item?id=20036194>)
- <https://news.ycombinator.com/item?id=17659983> (<https://news.ycombinator.com/item?id=17659983>)
- <https://news.ycombinator.com/item?id=17846387> (<https://news.ycombinator.com/item?id=17846387>)

For more detailed instructions, see the QuickStart guide and API reference above. You can also download the complete example setup here: <https://github.com/pirate/wireguard-example> (<https://github.com/pirate/wireguard-example>).

Suggest changes: <https://github.com/pirate/wireguard-docs/issues> (<https://github.com/pirate/wireguard-docs/issues>)