


[< Go back](#)

# How Tailscale works

23 mins    March 20 2020     Avery Pennarun

People often ask us for an overview of how Tailscale works. We've been putting off answering that, because we kept changing it! But now things have started to settle down.

Let's go through the entire Tailscale system from bottom to top, the same way we built it (but skipping some zigzags we took along the way). With this information, you should be able to build your own Tailscale replacement... except you don't have to, since our node software is open source and we have a flexible free plan.

## The data plane: WireGuard®

Our base layer is the increasingly popular and excellent open source WireGuard package (specifically the userspace Go variant, wireguard-go). WireGuard creates a set of extremely lightweight encrypted tunnels between your computer, VM, or container (which WireGuard calls an "endpoint" and we'll call a "node"), and any other nodes in your network.

Let's clarify that: most of the time, VPN users, including WireGuard users, build a "hub and spoke" architecture, where each client device (say, your laptop) connects to a central "concentrator" or VPN gateway.

## Hub-and-spoke networks

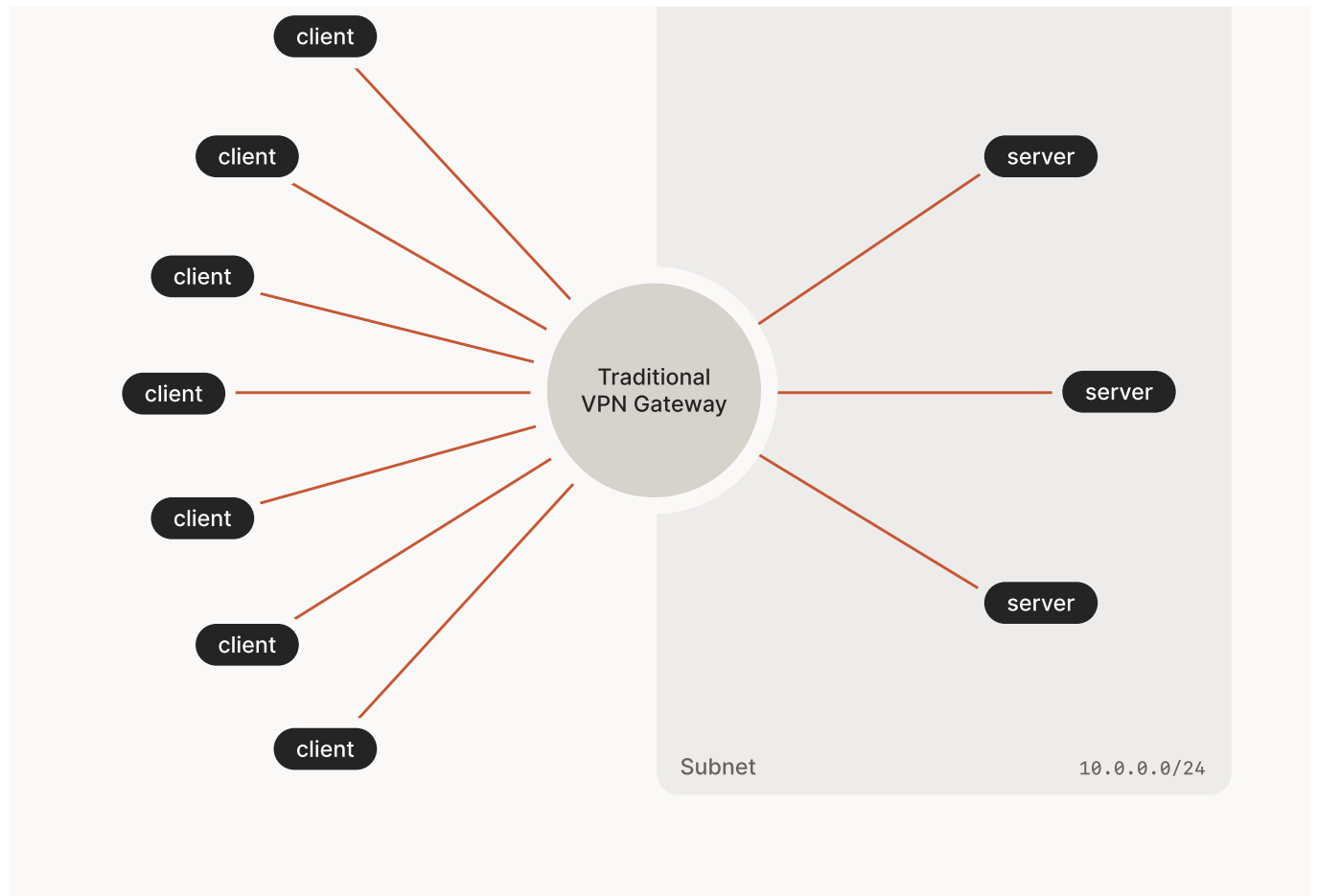


Figure 1. A traditional hub-and-spoke VPN.

This is the easiest way to set up WireGuard, because each node in the network needs to know the public key, public IP address, and port number of each other node it wants to connect directly to. If you wanted to fully connect 10 nodes, then that would be 9 peer nodes that each node has to know about, or 90 separate tunnel endpoints. In a hub-and-spoke, you just have one central hub node and 9 outliers (with “spokes” connecting them), which is much simpler. The hub node usually has a static IP address and a hole poked in its firewall so it’s easy for everyone to find. Then it can accept incoming connections from nodes at other IP addresses, even if those nodes are themselves behind a firewall, in the usual way that client-server Internet protocols do.

Hub-and-spoke works well, but it has some downsides. First of all, most modern companies don’t have a single place they want to designate as a hub. They have multiple offices, multiple cloud datacenters or regions or VPCs, and so on. In traditional VPN setups, companies configure a single VPN concentrator, and then set up secondary tunnels (often using IPsec) between locations. So remote users arrive at the VPN concentrator in one place, then have their traffic forwarded to its final destination in another place.

This traditional setup can be hard to scale. First of all, remote users might or might not be close to the VPN concentrator; if they’re far away, then they incur high latency connecting to it. Secondly, the datacenter they want to reach might not be close to the VPN concentrator either; if it’s far away, they incur high latency again. Imagine this case, with a worker in New York trying to reach a

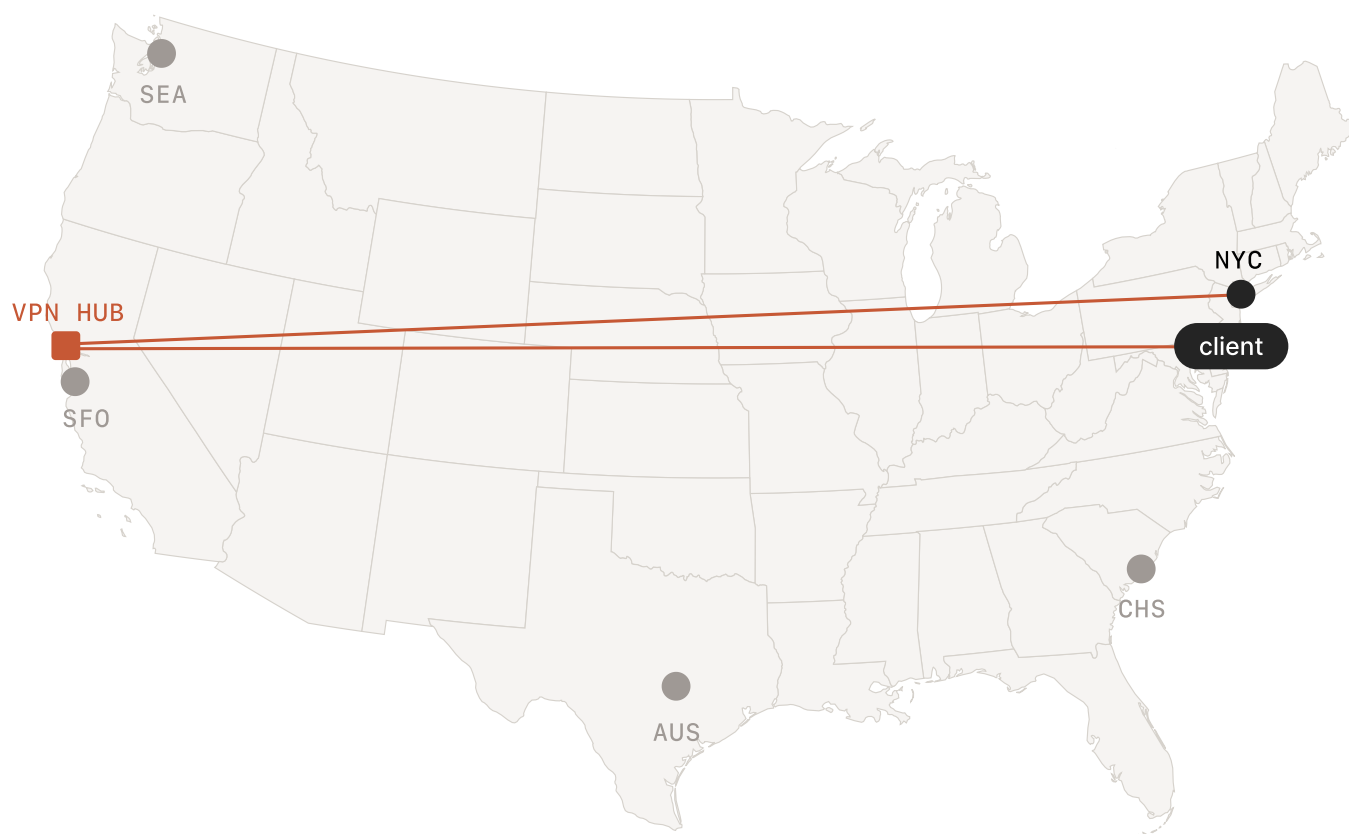


Figure 2(a). Inefficient routing through a traditional VPN concentrator.

Luckily, WireGuard is different. Remember how we said it creates a “set of extremely lightweight tunnels” above? The tunnels are light enough that we can create a multi-hub setup without too much trouble:

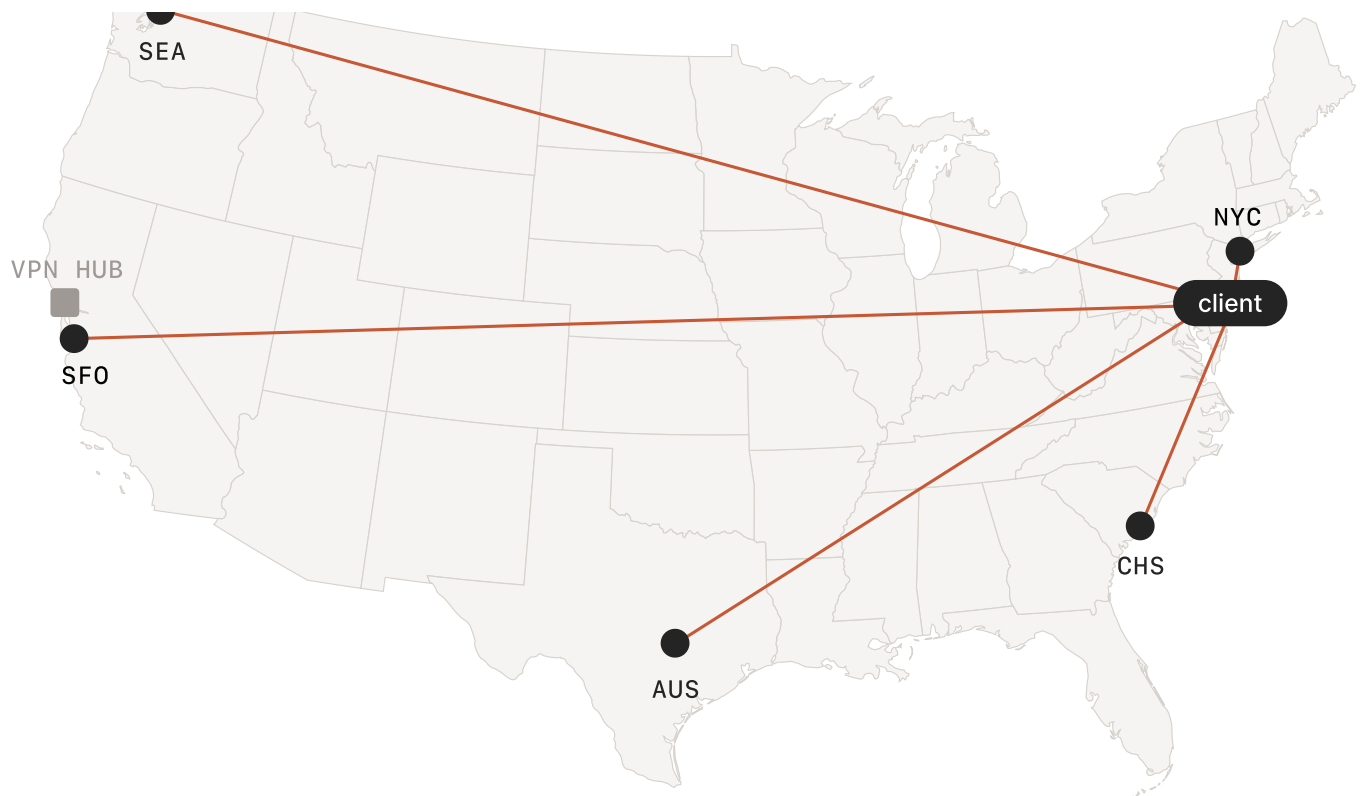


Figure 2(b). A WireGuard multipoint VPN routes traffic more efficiently.

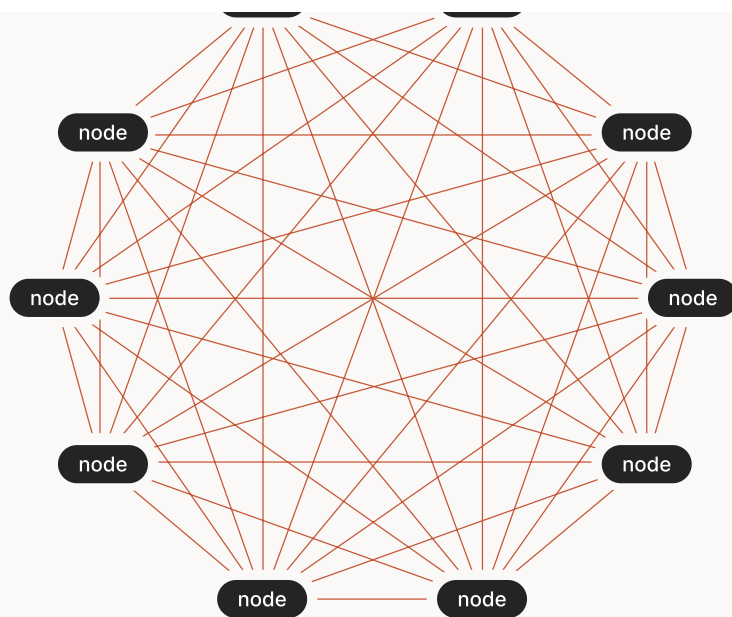
The only catch is that now each of the datacenters needs a static IP address, an open firewall port, and a set of WireGuard keys. When we add a new user, we'll have to distribute the new key to all five servers. When we add a new server, we'll have to distribute its key to every user. But we can do it, right? It's only about 5 times as much work as one hub, which is not that much work.

## Mesh networks

So that's hub-and-spoke networks. They're not too hard to set up with WireGuard, although a bit tedious, and we haven't really talked about safe practices for key management yet (see below).

Still, there's something fundamentally awkward about hub-and-spoke: they don't let your individual nodes talk to each other. If you're old like me, you remember when computers could just exchange files directly without going to the cloud and back. That was how the Internet all used to work, believe it or not! Sadly, developers have stopped building peer-to-peer apps because the modern Internet's architecture has evolved, almost by accident, entirely into this kind of hub-and-spoke design, usually with the major cloud providers in the center charging rent.

Remember those "extremely lightweight" WireGuard tunnels? Wouldn't it be nice if you could directly connect all the nodes to all the other nodes? That's called a mesh network:



$$n(n-1) = 90 \text{ WireGuard endpoints (for 45 connections)}$$

Figure 3. A Tailscale point-to-point mesh network minimizes latency.

That would be very elegant—at least, it would let you design elegant peer-to-peer apps—but it would be tricky. As mentioned above, a 10-node network would require  $10 \times 9 = 90$  WireGuard tunnel endpoint configurations; every node would need to know its own key plus 9 more, and each node would have to be updated every time you rotate a key or add/remove a user.

And the nodes would all have to find each other somehow—user devices rarely have static IP addresses—and reconnect whenever one of them moves around.

Plus, you can't easily open a firewall port for every single node to allow incoming connections in, say, a cafe, hotel, or airport.

And then, after you've done all that, if your company has compliance requirements, you need to be able to somehow block and audit traffic between all the nodes, even though it's no longer all going through a central location that you can clamp down.

Tailscale makes all that work too! Let's talk about how. This is where things get a bit hairy.

## The control plane: key exchange and coordination

Okay, we've made it this far. We got WireGuard connecting. We got it connecting to multiple things at a time. We've marvelled at its unparalleled reliability and efficiency. Great! Now we want to build a mesh network—everything connected to everything else.

## tailscale

How do we get all the WireGuard encryption keys (a simplified and more secure form of “certificates”) onto every device?

For that, we use the open source Tailscale node software. It talks to what we call a “coordination server” (in our case, login.tailscale.com)—essentially, a shared drop box for public keys.

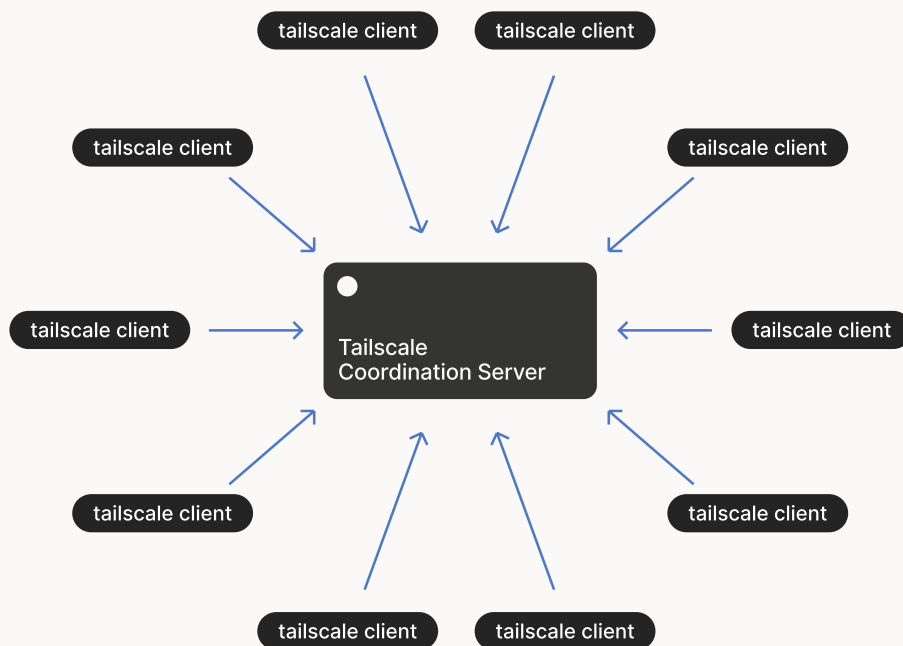


Figure 4. Tailscale public keys and metadata are shared through a centralized coordination server.

Hold on, are we back to hub-and-spoke again? Not exactly. The so-called “control plane” is hub and spoke, but that doesn’t matter because it carries virtually no traffic. It just exchanges a few tiny encryption keys and sets policies. The data plane is a mesh.

(We like this hybrid centralized-distributed model. Companies and teams usually want to have central control, but they don’t want a central bottleneck for their data. Traditional VPN concentrators centralize both, limiting performance, especially under stress. Conversely, Tailscale’s data processing power scales up with the number of nodes.)

Here’s what happens:

- 01 Each node generates a random public/private keypair for itself, and associates the public key with its identity (see login, below).

## tailscale

- 03 The node downloads a list of public keys and addresses in its domain, which have been left on the coordination server by other nodes.
- 04 The node configures its WireGuard instance with the appropriate set of public keys.

Note that the private key never, ever leaves its node. This is important because the private key is the only thing that could potentially be used to impersonate that node when negotiating a WireGuard session. As a result, only that node can encrypt packets addressed from itself, or decrypt packets addressed to itself. It's important to keep that in mind: Tailscale node connections are end-to-end encrypted (a concept called "zero trust networking").

Unlike a hub-and-spoke network, unencrypted packets never need to be sent over a wire, and no intermediary can inspect them. (The exception is if you use subnet routes, which are useful for incremental deployment, which Tailscale supports. You can create a hybrid network that combines new-style mesh connections with zero or more old-style "hubs" that decrypt packets, then forward them to legacy physical networks.)

## Login and 2-factor auth (2FA)

But we skipped over a detail. How does the coordination server know which public keys should be sent to which nodes? Public keys are just that—public—so they are harmless to leak to anyone, or even post on a public web site. This is exactly the same situation as an ssh server with an `authorized_keys` file; you don't have to keep your public ssh key secret, but you still have to be careful which public keys you put in `authorized_keys`.

There are many ways to make the authentication decision. An obvious way would be to build a username+password system, also known as PSK (pre-shared keys). To set up your node, connect to the server, enter your username and password, then upload your public key and download other public keys posted by either your account or other accounts in your domain. If you want to get fancy, you can add two-factor authentication (2FA, also known as MFA) such as SMS, Google Authenticator, Microsoft Authenticator, and so on.

A system administrator could also set up your machine with a "machine certificate" — a key that belongs permanently (or semi-permanently) to the device rather than to the user account. It could use this to ensure that, even with the right username and password, an untrusted device can never publish new keys to the coordination server.

Tailscale operates a coordination server based around these concepts. However, we don't handle user authentication ourselves. Instead, we always outsource authentication to an OAuth2, OIDC (OpenID Connect), or SAML provider. Popular ones include Gmail, GSuite, and Office365.

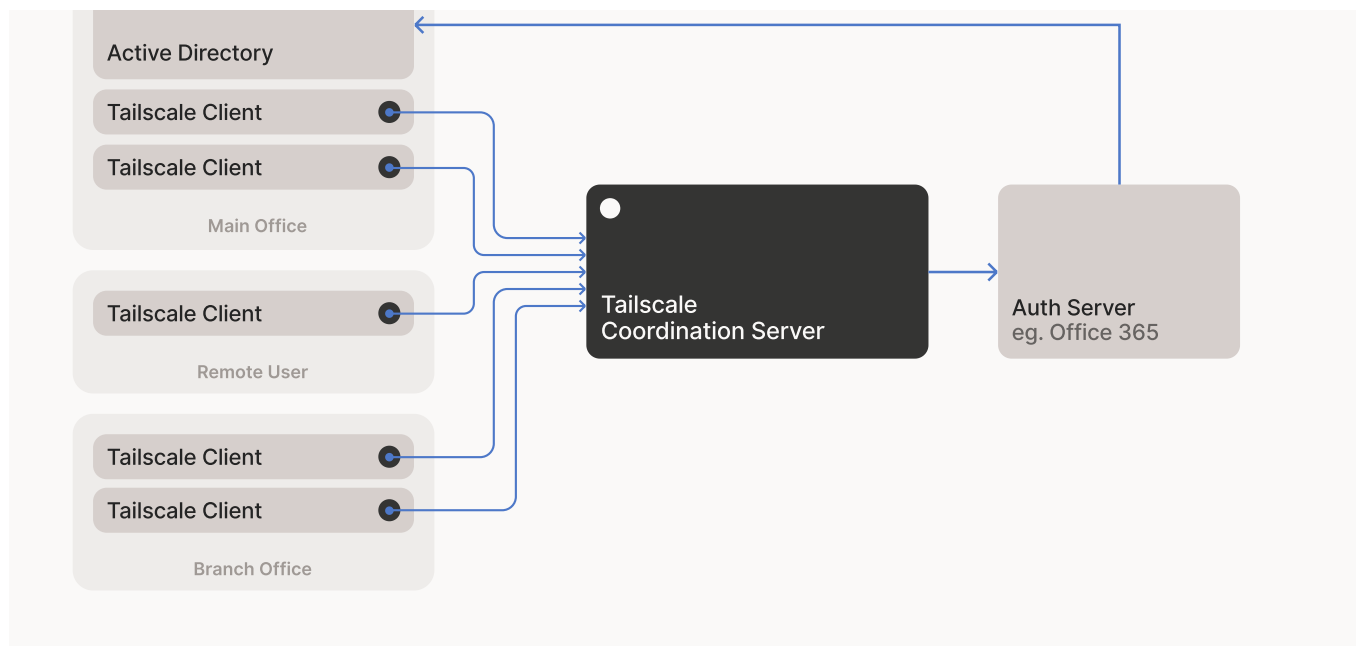


Figure 5. Tailscale 2FA authentication flow in the control plane.

The identity provider maintains a list of users in your domain, passwords, 2FA settings, and so on. This avoids the need to maintain a separate set of user accounts or certificates for your VPN — you can use the authentication you already have set up for use with Google Docs, Office 365, and other web apps. Also, because all your private user account and login data is hosted on another service, Tailscale is able to operate a highly reliable central coordination service while holding a minimum of your users' personally identifiable information (PII). (Read our privacy policy for more.)

You also don't have to change how you do single sign-on, account creation and removal, password recovery, and 2FA setup. It's all exactly like what you already have.

And because of all that, Tailscale domains can activate instantly the moment you login. If you download our macOS or iOS app from the App Store, for example, and login to your account, this immediately creates a secure key drop box for your domain and lets you exchange public keys with other devices in that account or domain, like a Windows or Linux server.

And then, as we just saw, the public keys get downloaded to each node, each node configures WireGuard with a super-lightweight tunnel to each other node, and ta da! A mesh network in two minutes!

## NAT traversal

...but we're not quite done yet. Recall that up above, we decided to pretend that every node has a static IP address and an open firewall port for incoming WireGuard traffic. In real life, that's not too likely. In real life, some of your nodes are in cafes or on airplanes or LTE on the highway with one bar of signal, battery drained, fleeing desperately from the cops across state lines... oh, sorry, wrong movie.



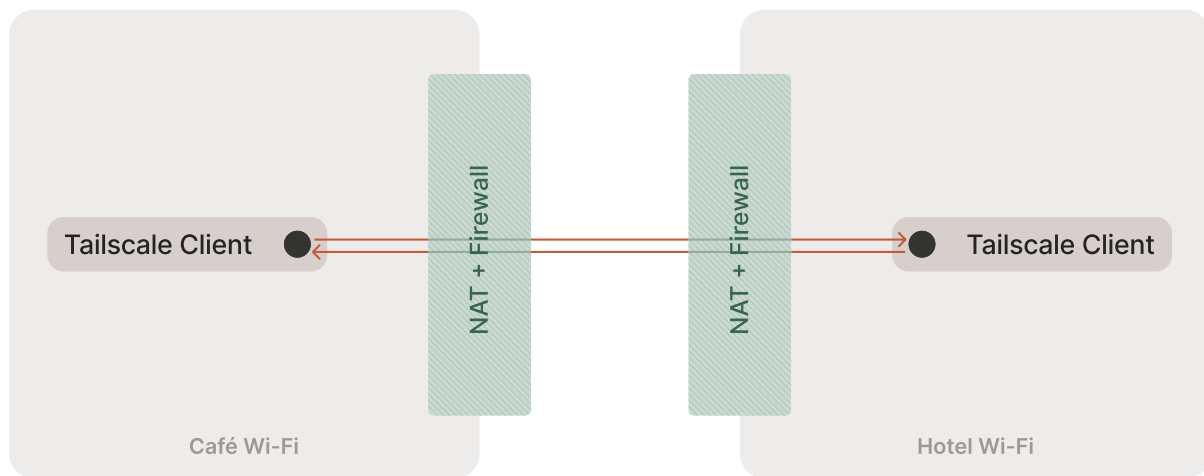


Figure 6. Tailscale can connect even when both nodes are behind separate NAT firewalls.

That's two NATs, no open ports. Historically, people would ask you to enable uPnP on your firewall, but that rarely works and even when it does work, it usually works dangerously well until administrators turn it off.

For now, suffice it to say that Tailscale uses several very advanced techniques, based on the Internet STUN and ICE standards, to make these connections work even though you wouldn't think it should be possible. This avoids the need for firewall configurations or any public-facing open ports, and thus greatly reduces the potential for human error.

For all the gory details, see my teammate Dave Anderson's post: [How NAT traversal works](#).

## Encrypted TCP relays (DERP)

Just one more thing! Some especially cruel networks block UDP entirely, or are otherwise so strict that they simply cannot be traversed using STUN and ICE. For those situations, Tailscale provides a network of so-called DERP (Designated Encrypted Relay for Packets) servers. These fill the same role as TURN servers in the ICE standard, except they use HTTPS streams and WireGuard keys instead of the obsolete TURN recommendations.

Relaying through DERP looks like this:

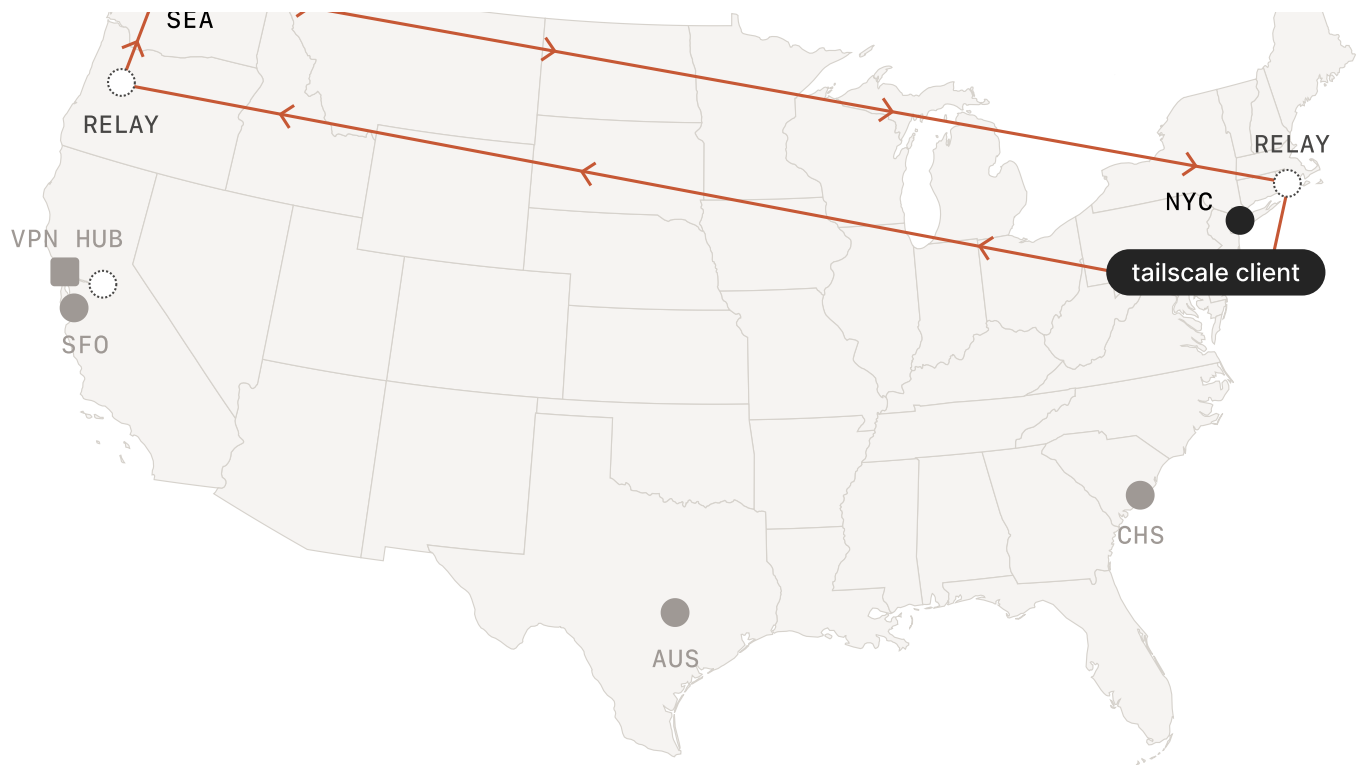


Figure 7. Tailscale asymmetrically routes traffic through the DERP nearest to each recipient.

(Despite how it might look above, we really do have a globally distributed network of relay servers. It's not just the United States. In the words of our intrepid designer, "I was hoping to include London in there, but finding two SVGs with compatible non-Mercator projections was a disaster." This is how much we care about technical accuracy. Also, stop using Mercator projections. They're super misleading.)

Remember that Tailscale private keys never leave the node where they were generated; that means there is never a way for a DERP server to decrypt your traffic. It just blindly forwards already-encrypted traffic from one node to another. This is like any other router on the Internet, albeit using a slightly fancier protocol to prevent abuse.

Many people on the Internet are curious about whether WireGuard supports TCP, like OpenVPN does. Currently this is not built into WireGuard itself, but the open source Tailscale node software includes DERP support, which adds this feature. Over time, it's possible the code will be refactored to include this feature in WireGuard itself. (Tailscale has already contributed several fixes and improvements to WireGuard-Go.) We have also open sourced the (quite simple) DERP relay server code so you can see exactly how it works.

## Bonus: ACLs and security policies

People often think of VPNs as "security" software. This categorization seems obvious because VPNs are filled with cryptography and public/private keys and certificates. But VPNs really fit in



As a result, VPN concentrators (remember the hub-and-spoke model from up above) are usually coupled with firewalls; sometimes they're even sold together. All the traffic comes from individual client devices, flows through the VPN concentrator, then into the firewall, which applies access controls based on IP addresses, and finally into the network itself.

That model works, but it can become a pain. First of all, firewall rules are usually based on IP addresses, not users or roles, so they can be very awkward to configure safely. As a result, you end up having to add more layers of authentication, at the transport or application layers. Why do you need ssh or HTTPS? Because the network layer is too insecure to be trusted.

Second, firewalls are often scattered around the organization and need to be configured individually. If you have a multi-hub network (for example, with different VPN concentrators in different geographic locations), you have to make sure to configure the firewall correctly on each one, or risk a security breach.

Finally, it's usually a pain to configure some particular vendor's VPN/firewall device to authenticate VPN connections against some other vendor's identity system. This is why some identity system vendors will try to sell you a VPN, and some VPN vendors will try to sell you an identity system.

When you switch to a mesh network, this problem seems to get even worse — there is no central point at all, so where do you even put the firewall rules? In Tailscale, the answer is: in every single node. Each node is responsible for blocking incoming connections that should not be allowed, at decryption time.

To make that easier, your company's security policy is stored on the Tailscale coordination server, all in one place, and automatically distributed to each node. This way you have central control over policy, but efficient, distributed enforcement.

## tailscale

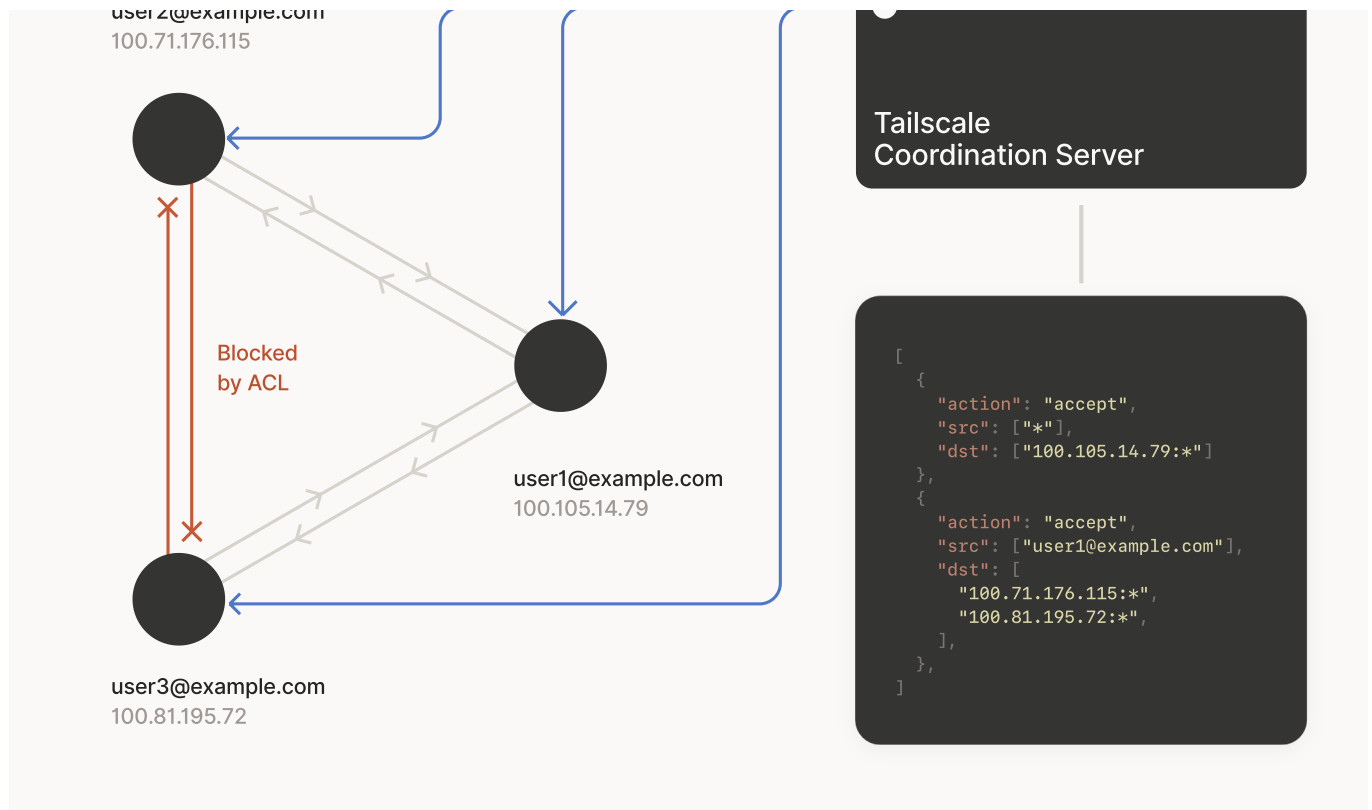


Figure 8. Central ACL policies are enforced by each Tailscale node's incoming packet filter. If an 'accept' rule doesn't exist, the traffic is rejected.

At a less granular level, the coordination server (key drop box) protects nodes by giving each node the public keys of only the nodes that are supposed to connect to it. Other Internet computers are unable to even request a connection, because without the right public key in the list, their encrypted packets cannot be decoded. It's like the unauthorized machines don't even exist. This is a very powerful protection model; it prevents virtually any kind of protocol-level attack. As a result, Tailscale is especially good at protecting legacy, non-web based services that are no longer maintained or receiving updates.

## Bonus: Audit logs

Another concern of companies with tight compliance requirements is audit trails. Modern firewalls don't just block traffic — they log it. Since Tailscale doesn't have a central traffic funnel, what can you do?

The answer is: we allow you to log all internal network connections from each node, asynchronously, to a central logging service. An interesting side effect of this design is that every connection is logged twice: on the source machine and the destination machine. As a result, log tampering is easy to detect, because it would require simultaneous tampering on two different nodes.

Because logs are streamed in real time from each node, rather than batched, the window for local log tampering on a node is extremely short, in the order of dozens of milliseconds, making even a



metadata about how your internal mesh is established, not Internet usage or personal information. (Read our privacy policy for more.)

Rather than providing a complete logs and metrics pipeline, the Tailscale logging service is intended as a real-time streaming data collector that can then stream data out into other systems for further analysis. (You can read my early blog post that evolved into Tailscale's logs collector architecture.)

## Bonus: Incremental deployment

There is one last question that comes up a lot: given that Tailscale creates a mesh “overlay” network (a VPN that parallels a company's internal physical network), does a company have to switch to it all at once? Many BeyondCorp and zero-trust style products work that way. Or can it be deployed incrementally, starting with a small proof of concept?

Tailscale is uniquely suited to incremental deployments. Since you don't need to install any hardware or any servers at all, you can get started in two minutes: just install the Tailscale node software onto two devices (Linux, Windows, macOS, iOS), login to both devices with the same user account or auth domain, and that's it! They're securely connected, no matter how the devices move around. Tailscale runs on top of your existing network, so you can safely deploy it without disrupting your existing infrastructure and security settings.

You can then extend the network by adding subnet routes to one or more offices or datacenters, building up a traditional hub-and-spoke or multi-hub VPN.

As you gain confidence, you can install Tailscale on more servers or containers, which allows point-to-point fully encrypted connections (no unencrypted traffic carried over any LAN). This final configuration is called “zero trust networking,” which is gaining fame lately, but so far has been hard to attain.

With Tailscale, you can build up to “zero trust,” one employee device and one server at a time. After the incremental deployment is done, you can safely shut down your legacy or unencrypted access methods.

Thanks to Ross Zurowski, our intrepid designer, for the illustrations :)

Share Article





## Subscribe to Tailscale's blog

We have a deep commitment to keeping your data safe.

Enter your email

Submit

Too much email?

RSS

X

## More articles

Apr 11, 2024

Remotely access Home Assistant via Tailscale for free



Alex Kretzschmar



## About the tailscale.com outage on March 7, 2024



Parker Higgins

Mar 22, 2024

## Tailscale SSH is now Generally Available



Sam Linville

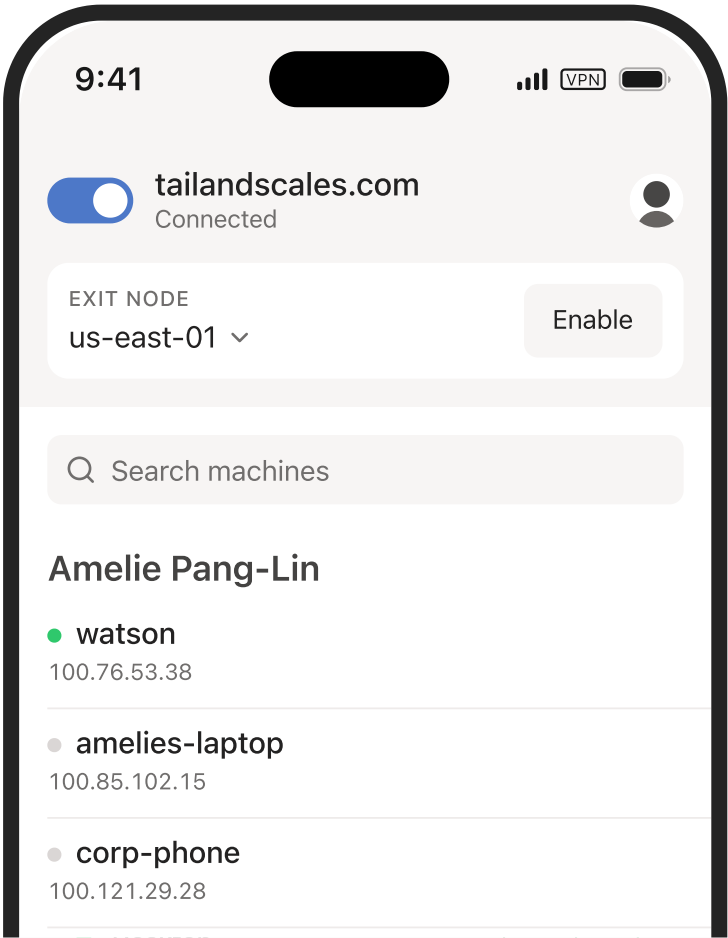
## Try Tailscale for free

Get started →



Schedule a demo

Contact sales



MERCURY



Product

[How it works](#)

[Pricing](#)

[Integrations](#)

[Features](#)

Use Cases

[Business VPN](#)

[Remote Access](#)

[Site-to-Site Networking](#)

[Homelab](#)

Resources

[Blog](#)

[Events & Webinars](#)





[Company](#)

[Careers](#)

[Press](#)

[Support](#)

[Sales](#)

[Security](#)

[Legal](#)

[Open Source](#)

[Changelog](#)

[SSH keys](#)

[Docker SSH](#)

[DevSecOps](#)

[Multicloud](#)

[NAT Traversal](#)

[MagicDNS](#)

[PAM](#)

[PoLP](#)

[All articles](#)



[Terms of Service](#)

[Privacy Policy](#)

WireGuard is a registered trademark of Jason A. Donenfeld.



© 2024 Tailscale Inc. All rights reserved. Tailscale is a registered trademark of Tailscale Inc.