TailScale & NAT Traversal

References:

- https://laisky.notion.site/How-NAT-traversal-works-Tailscale-3ace6e2373574e5184f7975902ce04fe?pvs=4

VPN & TailScale

What is VPN

VPN: Virtual Private Network

Network: 连接不同设备的网络 Private: 私有，安全的信道 Virtual: 虚拟，不是真实的物理网络

Why need VPN

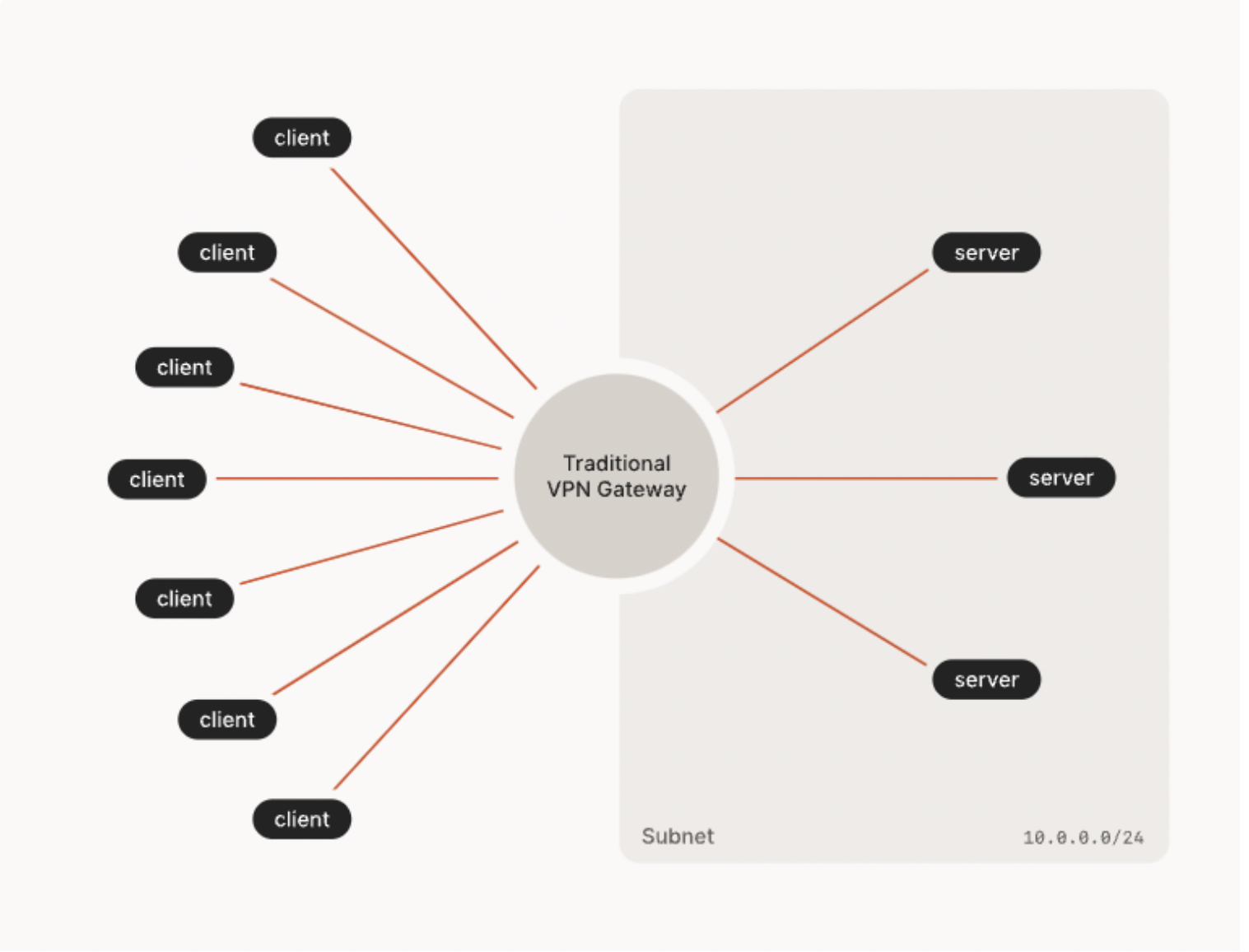
VPN 可以用来做什么？

1. 在不安全的网络环境中创建加密信道
2. 在公网环境中接入某个内网
3. 突破地域限制

对公场景 2 用处比较多，比如远程办公时接入办公环境。或者打通多个隔离的子网。

对私场景 1、3 最常见，比如抗审查、突破地域限制等。

传统的 VPN 拓扑

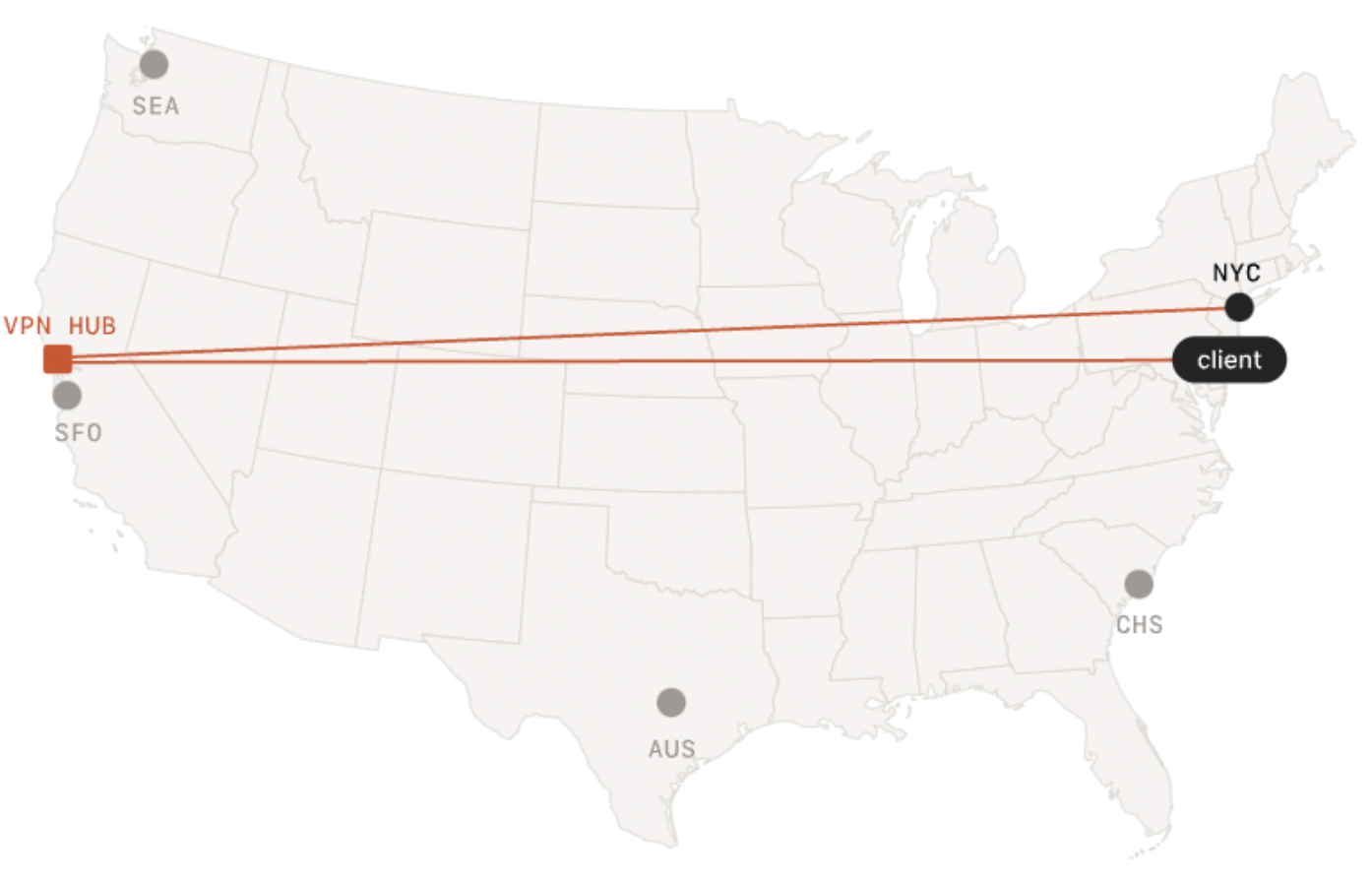


一台中心服务器，扮演网关和转发服务器（relay）的角色。

所有的客户端都接入这台中心网关，所有的数据也流经这个中心网关。

数据平米和控制平面都在这台中心网关上。

最显著的缺点就是，两台客户端可能离得非常进，但是流量缺需要通过遥远的中心网关来转发



这一点我有一段切身之痛的经历。

之前我在一家网络环境控制的很变态的公司。办公区域的网线和 wifi 处于不同的隔离域。

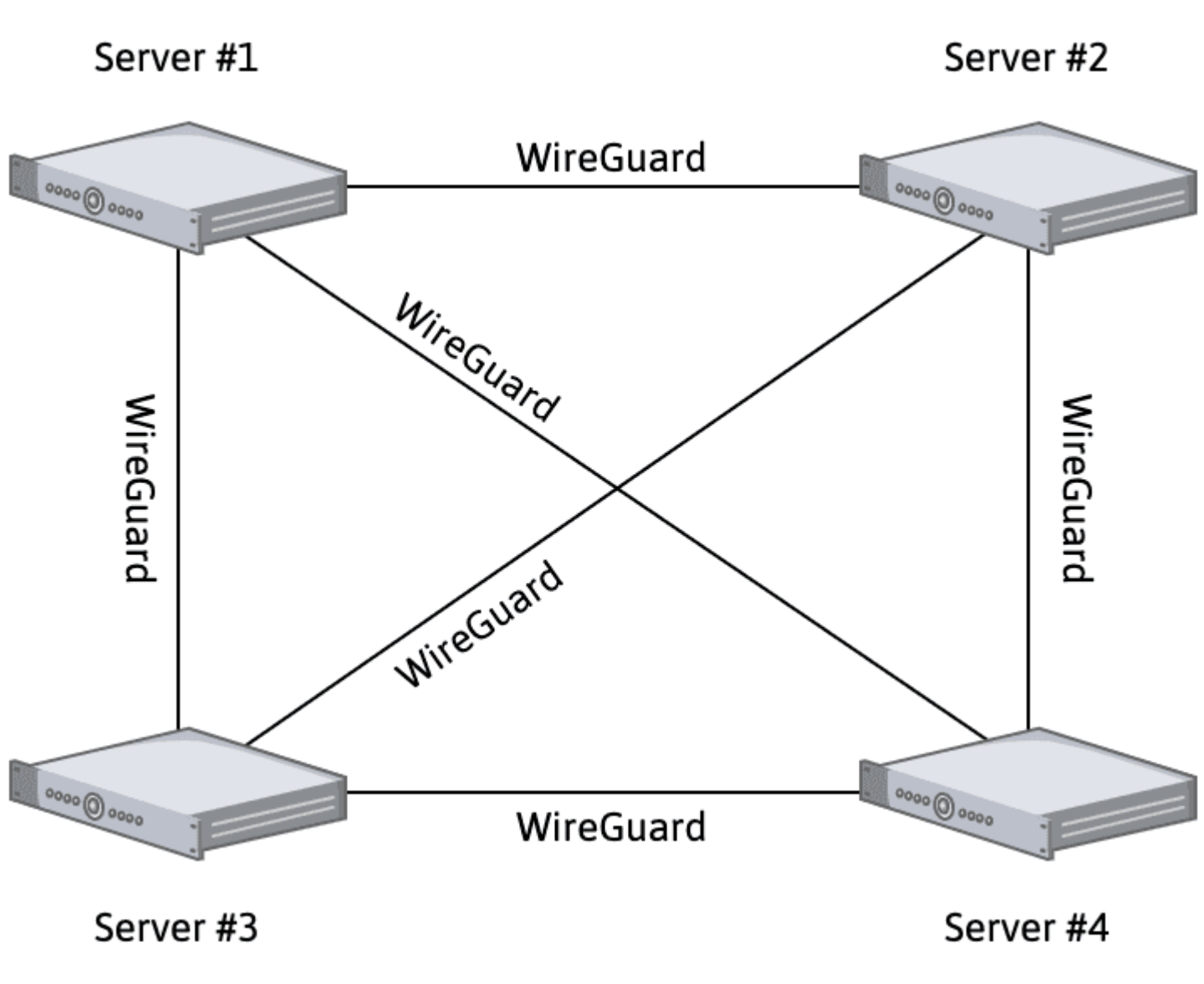
我喜欢远程开发，有一台台式机作为开发机，还有一台笔记本作为 UI。都插着网线的时候还好， 但是一旦遇到开会等场景，笔记本接入 wifi 了，就和台式机断开了。 后来我被逼无奈，只能在两台设备上都拨入 VPN。

但是公司的 VPN 网关在北京，我的两台设备相距不超过 1m，然而每一个比特都需要去北京跑一个来回。

Mesh VPN

那能不能引入 P2P 的思维，让距离比较近的客户端之间直接通信呢？

这个思想就是 Mesh VPN 了，其中的代表就是 wireguard。



wireguard 虽然非常强大，但是也很难用，首当其冲的就是配置极其繁琐。

从上图也能看出，每一个结点都需要能够感知到其他结点的存在，这完全依赖于手动配置结点信息和证书文件。

我们很懒，能不能开箱即用？于是有了 tailscale。

tailscale

tailscale 的底层依赖于 wireguard，但是在此基础上，分离了控制平面和数据平面。

引入中心化的 coordiantor 作为控制平面，管理每个租户名下的所有结点， 每个结点都会以 long polling（comet）的方式挂载到 coordinator，随时拉取最新的配置更新。

但是 coordiantor 只负责下发配置，不负责转发数据流量，所以不会成为单点瓶颈。

数据平面则以底层的 wireguard 为基础，扩充实现了多种 NAT 穿透协议和基于 UDP/TCP 的 relay 转发机制。

在强大的 NAT 穿透之外，以 relay 转发为最终的 fallback 保底方案。实现了强大的网络连通性。

当然，tailscale 发展了这么多年，其功能已经远不止打洞这么简单了。

其核心功能还包括：tunnel 外网 HTTPS 暴露、ACL 管理、taildrop 文件传送、tailssh 远程登录、taillock 零信任结点锁等等

NAT

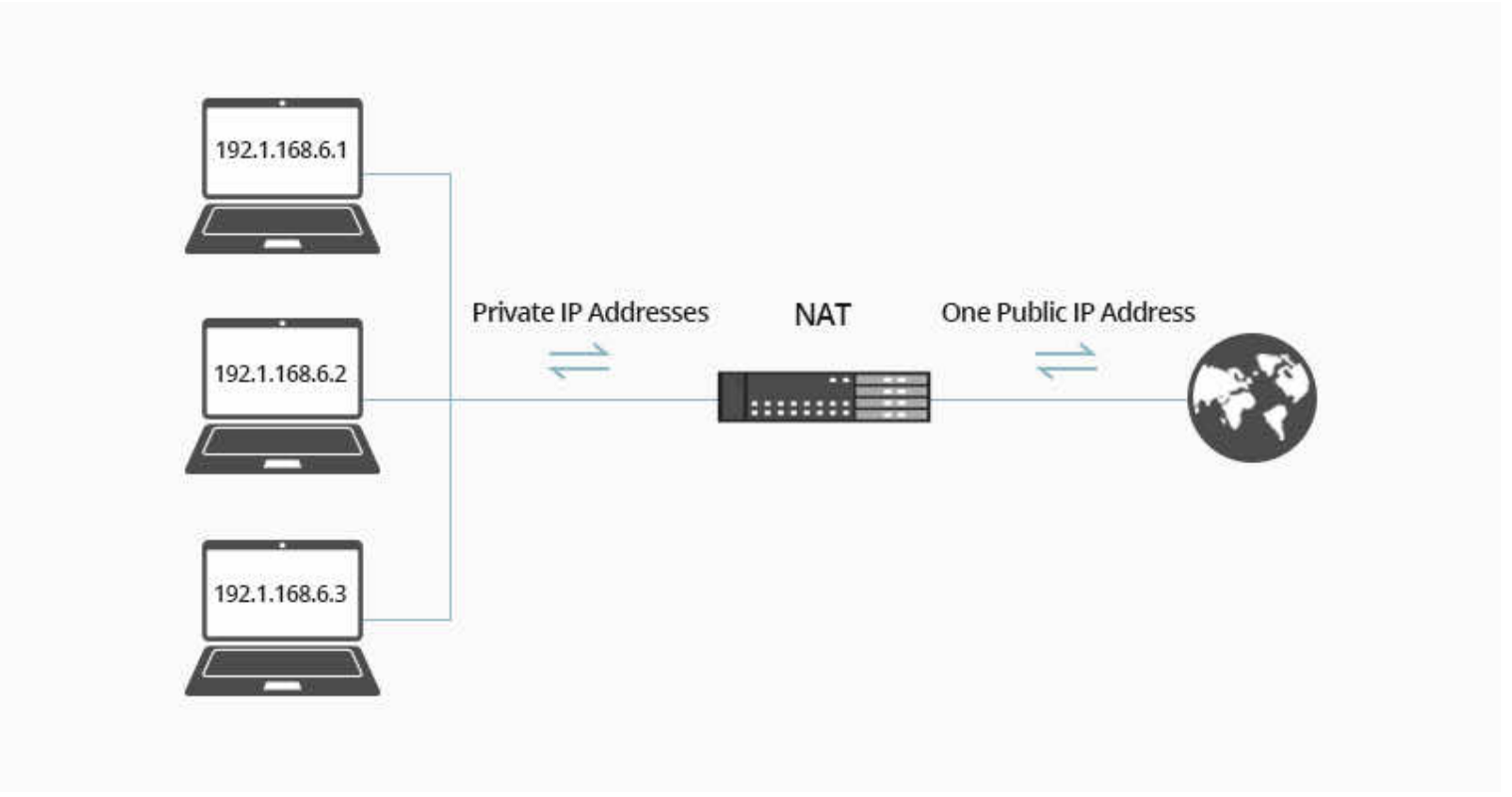
- 什么是 NAT
- 为什么需要 NAT 穿透
- 如何穿透

什么是 NAT

IPv4 一共有 43 亿个地址，其中还有 6 亿的地址是不可用于公网的保留地址。

而世界上的人口已经超过了 70 亿，IPv4 的地址事实上是不够用的。为了解决这个问题，人们引入了 NAT。

NAT 位于公网和私网的交界处，它可以将多个私网的 `ip:port` 转换为公网 IP 上的一个端口，以便私网的主机能够访问公网。

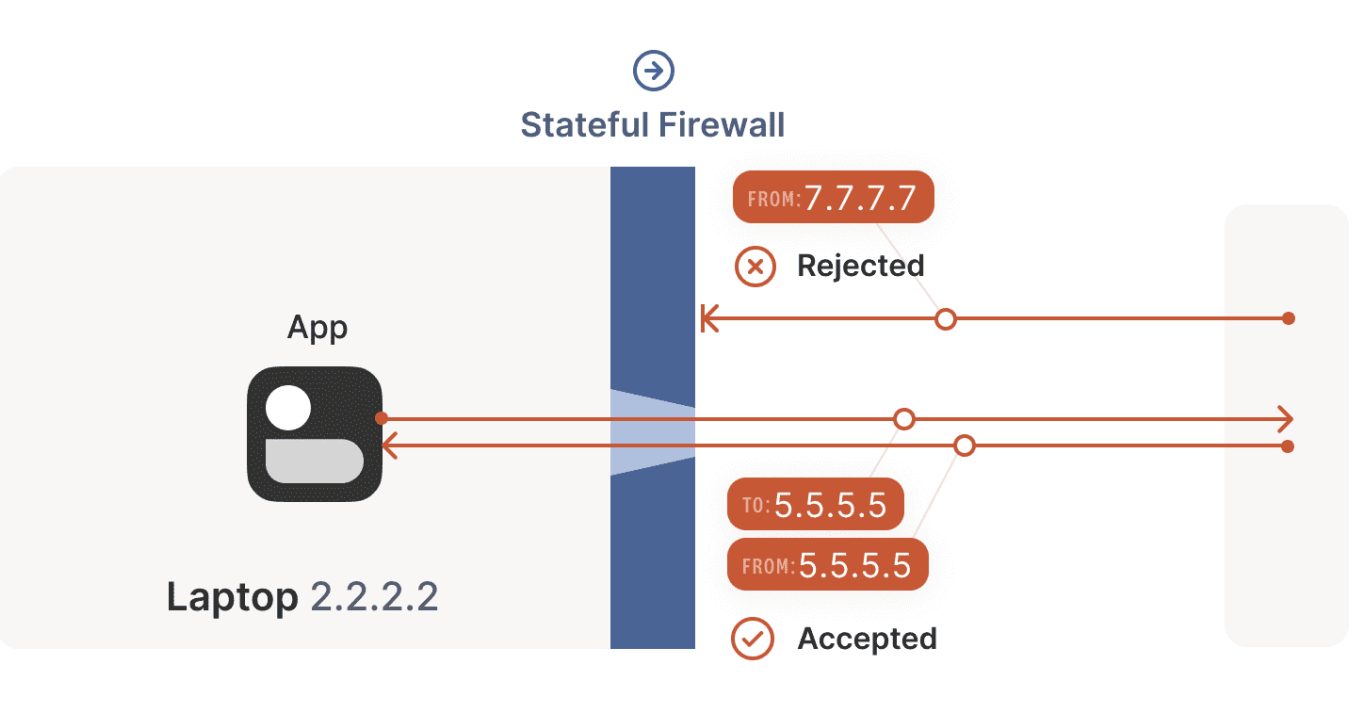


What is stateful firewall

最简单粗暴的防火墙策略就是放行出站，同时阻断一切入站。

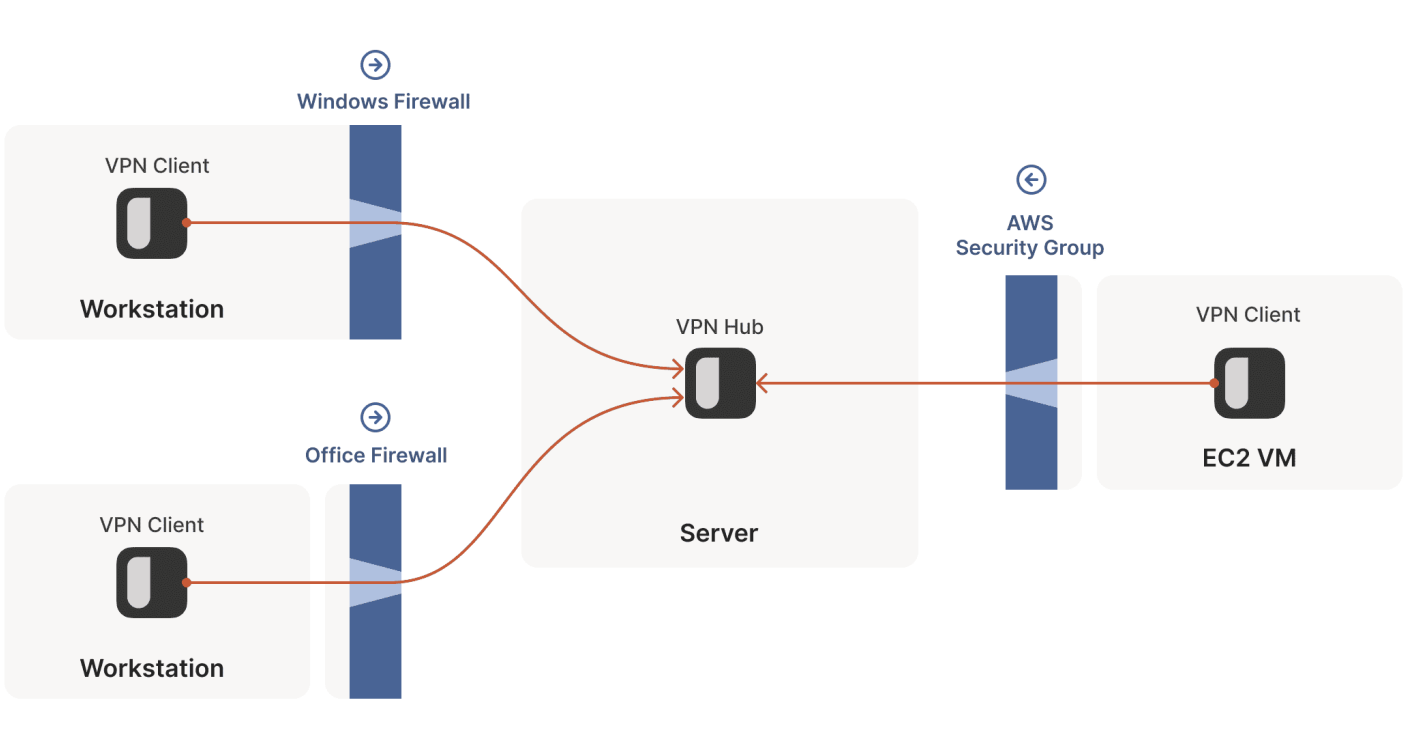
但是为了实现双向交流，防火墙一般都带有 `stateful` 的功能，简而言之：

防火墙可以记录最近的出站流量，并放行来自同一个 destination 的入站流量。



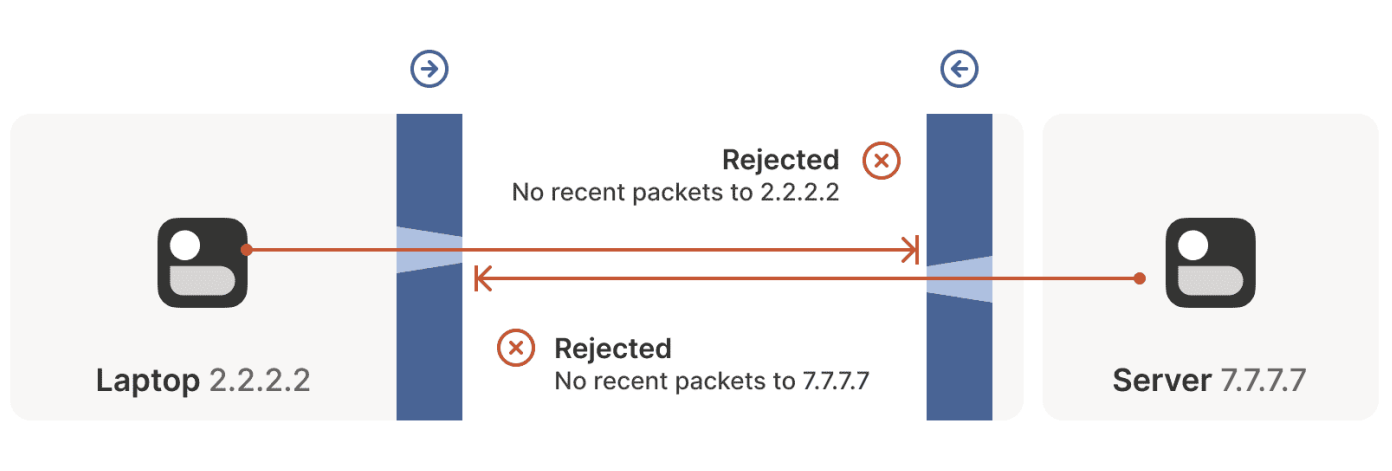
为什么需要穿透

在传统的中心化 VPN 服务器的时代，NAT 并不是一个问题，因为 VPN 服务器是暴露在公网的， 所以 VPN 客户端可以直接连接到 VPN 服务器，而不需要考虑 NAT 的问题。



但是在 mesh VPN 时代，为了让结点能够互联，NAT 就成了一个很严重的问题。

1. 虽然结点可以从 coordiantor 处获知自己和对方的公网 IP，但是无法知道对方端口，不知道往哪发
2. stateful 防火墙还会阻断对方发来的请求



NAT 穿透

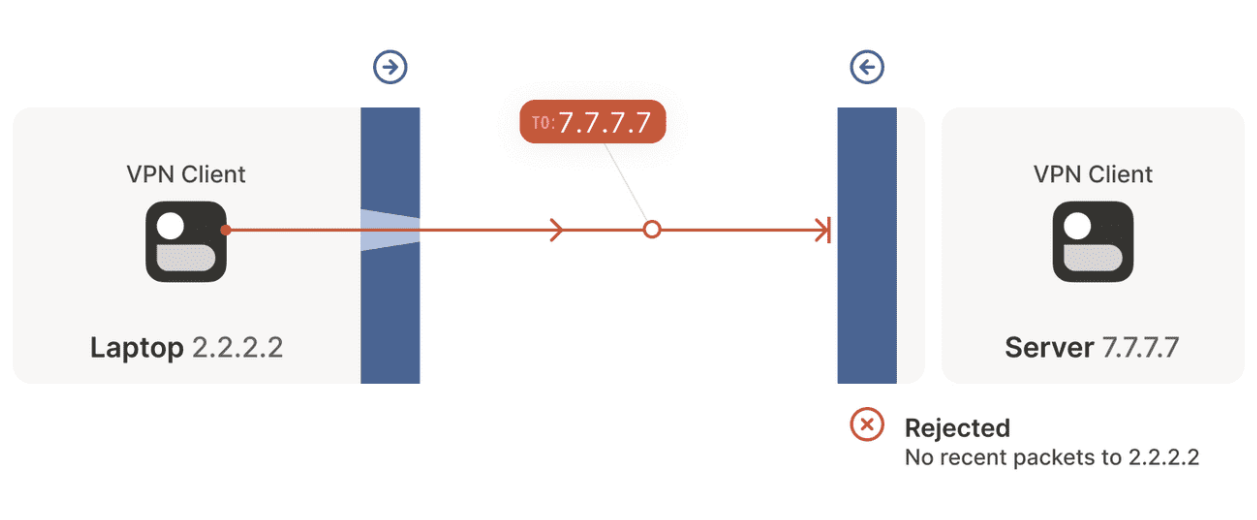
- 基于 UDP 的穿透基础
- STUN
- TURN/relay
- Hard NAT（Symmetric NAT）
- uPnP/NAT-PMP/PCP
- NAT hairpinning
- ICE

UDP 与打洞的基础原理

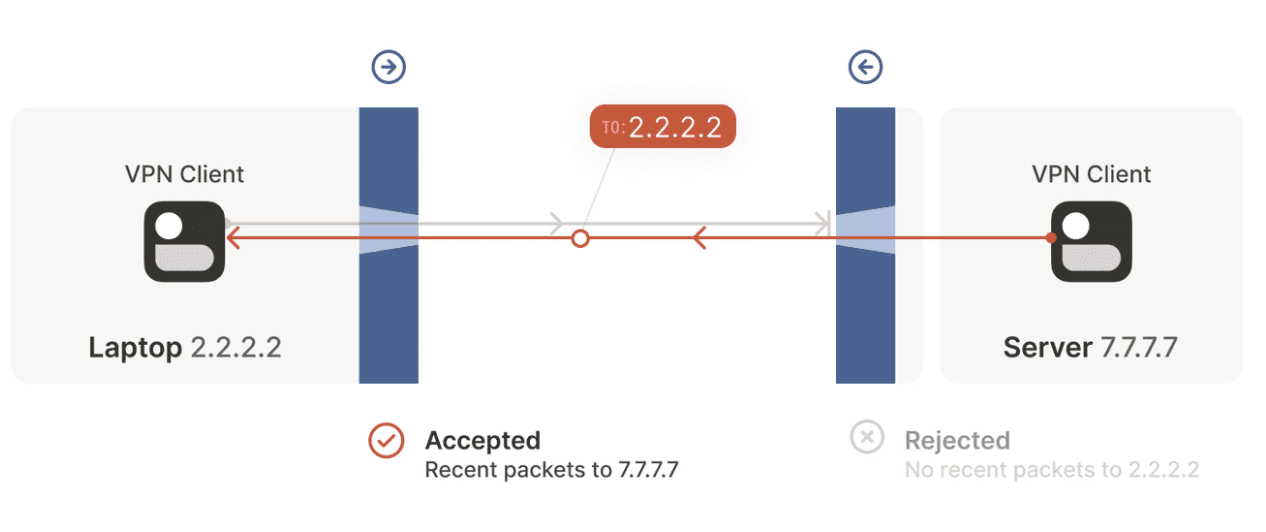
已知：

1. UDP 是一种无连接的协议，可以在不需要建立连接的情况下直接发送数据包。
2. stateful firewall 会记录出站流量，并放行同 dst 的入站流量
3. NAT 会为出站流量创建一个公网端口

综上，Alice 先向 Bob 发送一个有去无回的 UDP 包，就可以在自己的 NAT & firewall 上打一个洞，然后 Bob 就可以通过这个洞向 Alice 发送数据包了。



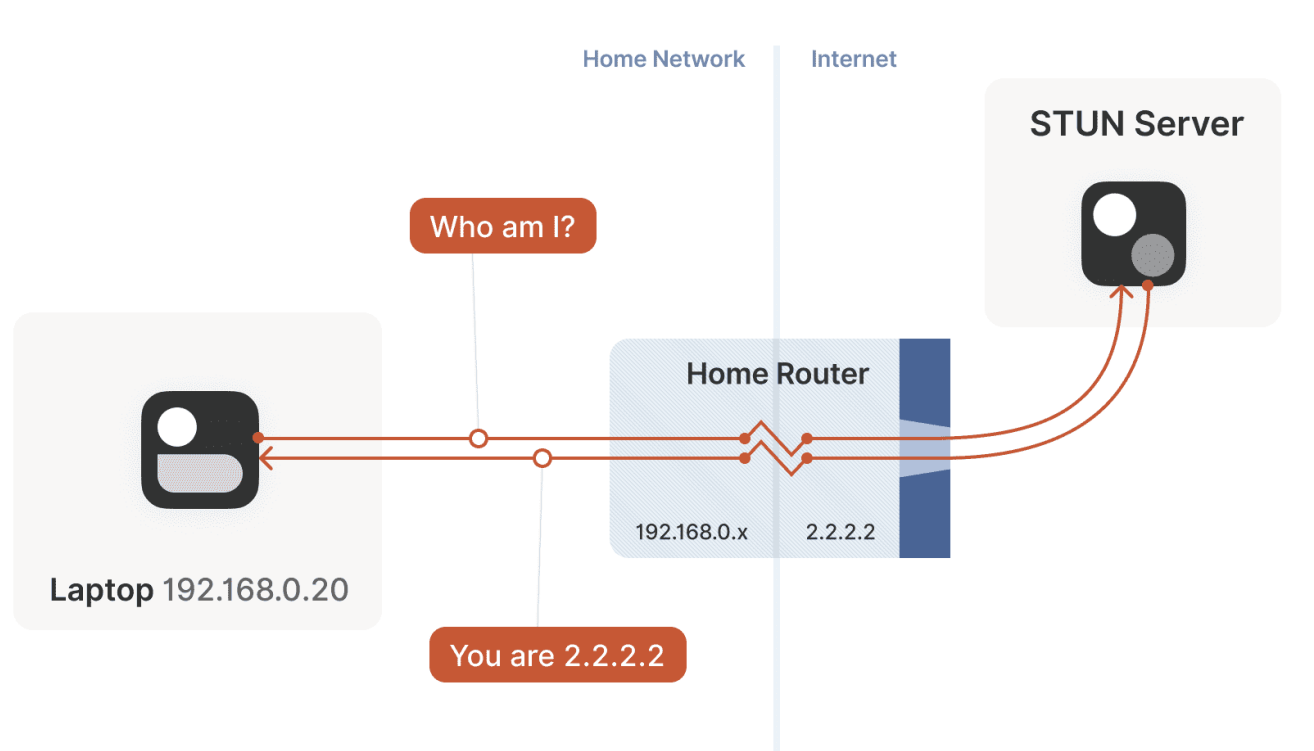
> The laptop's first packet, from `2.2.2.2:1234` to `7.7.7.7:5678`, goes through the Windows Defender firewall and out to the internet. The corporate firewall on the other end blocks the packet, since it has no record of `7.7.7.7:5678` ever talking to `2.2.2.2:1234`. However, Windows Defender now remembers that it should expect and allow responses from `7.7.7.7:5678` to `2.2.2.2:1234`.



STUN

Alice 和 Bob 在相互握手前，需要先预知对方在公网 NAT 上的 `ip:port`，而 STUN 协议正好就是用来干这事儿的。

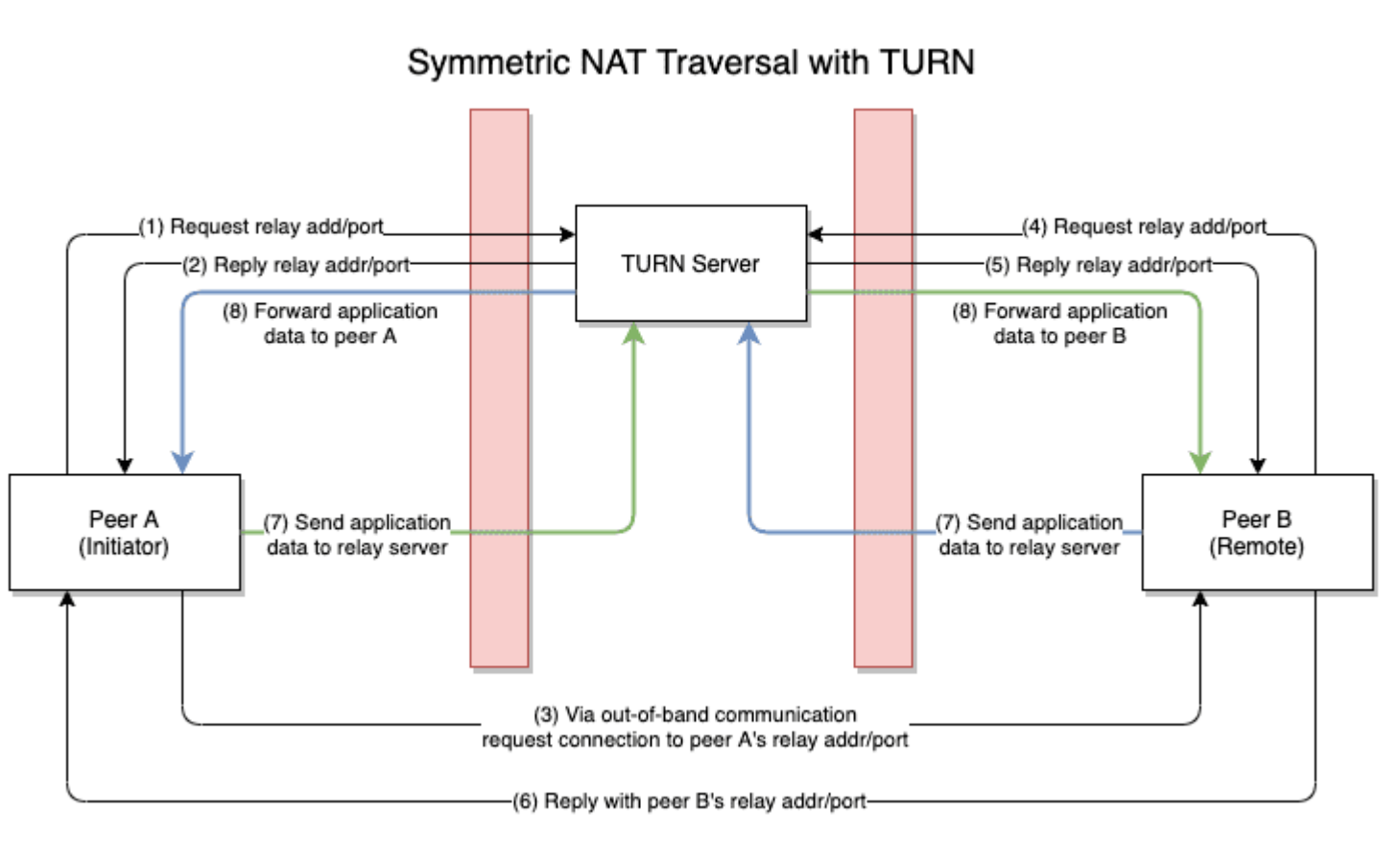
STUN 是一个暴露在公网上的服务协议，客户端向 STUN 发起请求，STUN 就会返回客户端的公网 `ip:port`， 这个地址信息实际上就是客户端在自己的 NAT 上开的洞。



TURN

TURN 是个经典的 relay 协议。一个结点去 TURN 上注册一个公网 `ip:port`，然后告诉对端可以通过 TURN 这个中转和自己通信。

tailscale 实现了自己的 relay 协议，称为 derper。这个协议最大的特点是基于 tcp， 在国内这种 UDP 干扰特别严重的网络环境下，可以提供相对更好的体验。



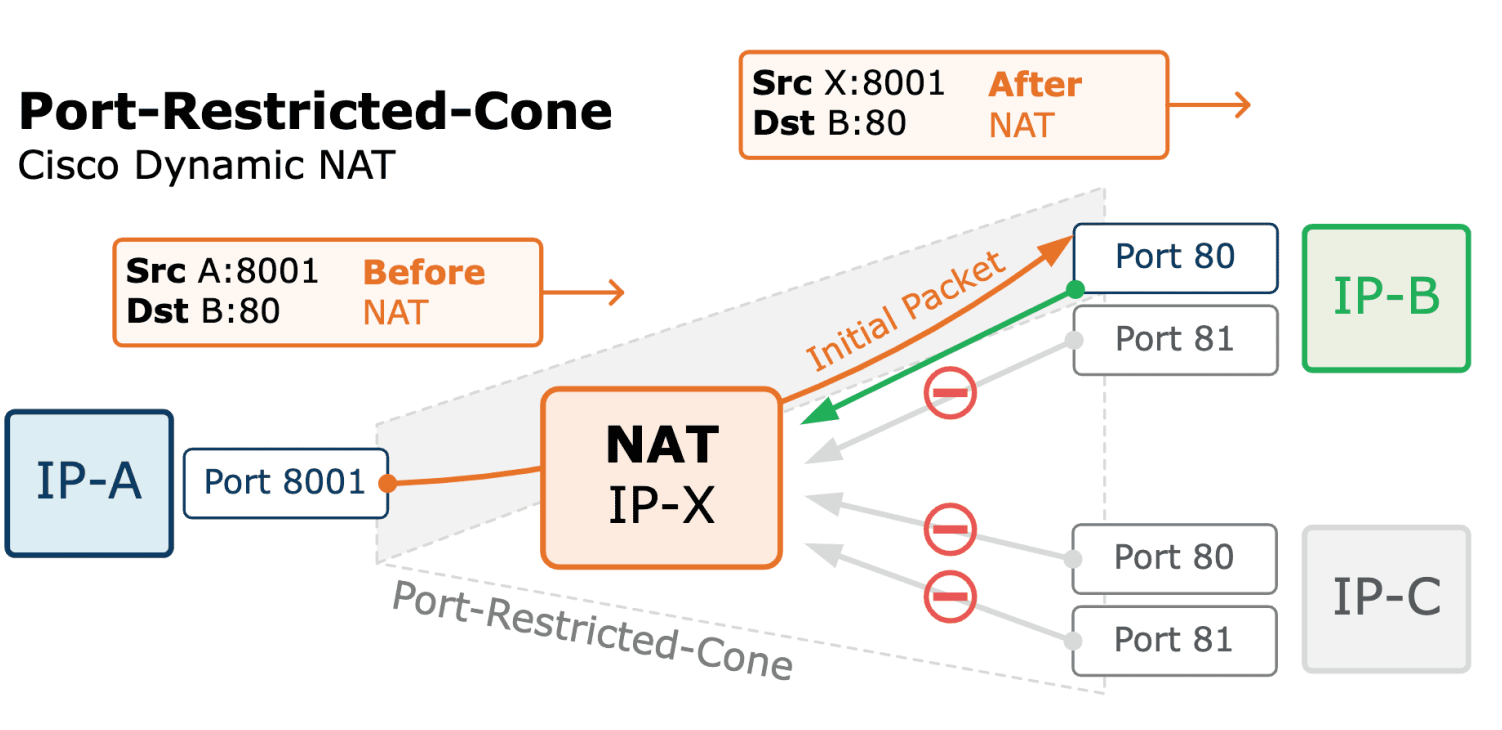
Hard NAT

前文提到，NAT 后的结点可以通过 STUN 查询自己在公网的 `ip:port`，从而远端可以使用这个地址和 NAT 后的结点通信。

但是，NAT 是分为很多种类的：Full-Cone、Restricted-Cone、Port-Restricted-Cone、Symmetric 等。

简而言之，我们前文讨论的是最宽松的 NAT（full-cone），就是一个 `src ip:port` 始终会被映射到同一个公网 `ip:port`。

但是，还存在一种最严苛的 NAT（symmetric），就是每一个 `(src ip:port, dst ip:port)` 都会被映射到不同的公网 `ip:port`。



我们按照 NAT 是否会根据 `dst ip:port` 进行映射，将其简单粗暴的分为 Easy/Hard 两类

映射方式	无视 `dst ip:port`	依赖 `dst ip:port`
分类	Easy	Hard

注：默认都启用了 stateful firewall。

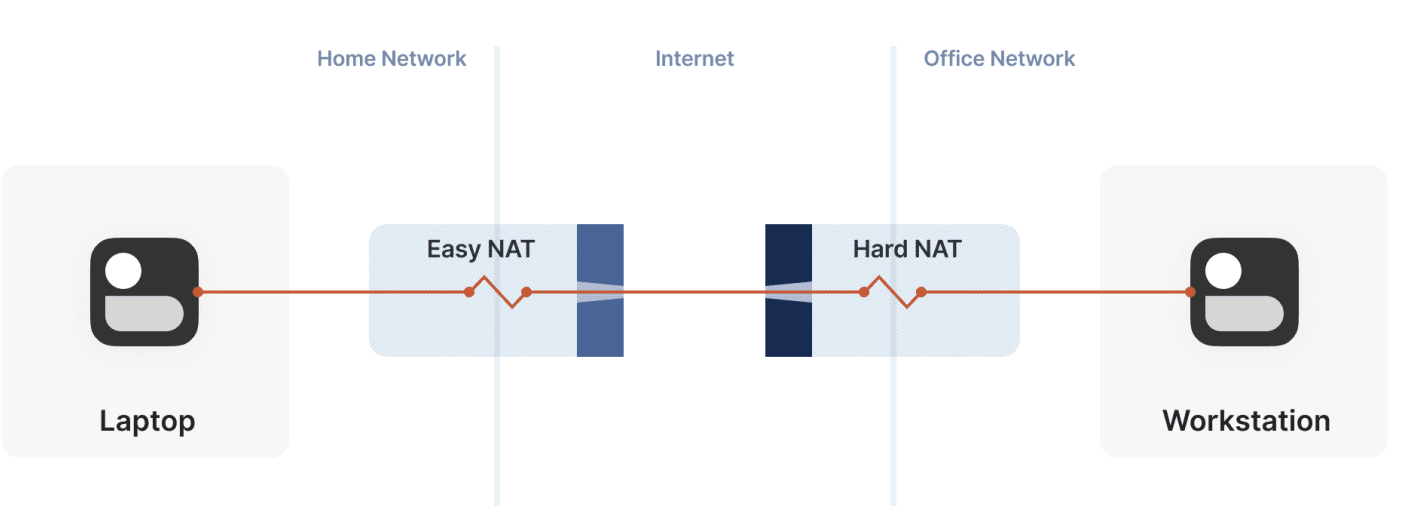
Hard NAT 导致的最直接的后果就是，STUN 没用了。

任何结点去请求 STUN 拿到的公网 `ip:port` 都没法交给其他结点使用，因为 NAT 只会放行 `dst ip:port` 严格匹配的入站流量。

如果通信双方都是 hard NAT，那么无可奈何，别指望能建立连接了，只能依赖 TURN/derper 进行中转转发。

Hard And Easy NAT

但是，如果通信的双方，仅有一方位于 Hard NAT 后，而另一方位于 Easy NAT 后，那么事情还有转机。



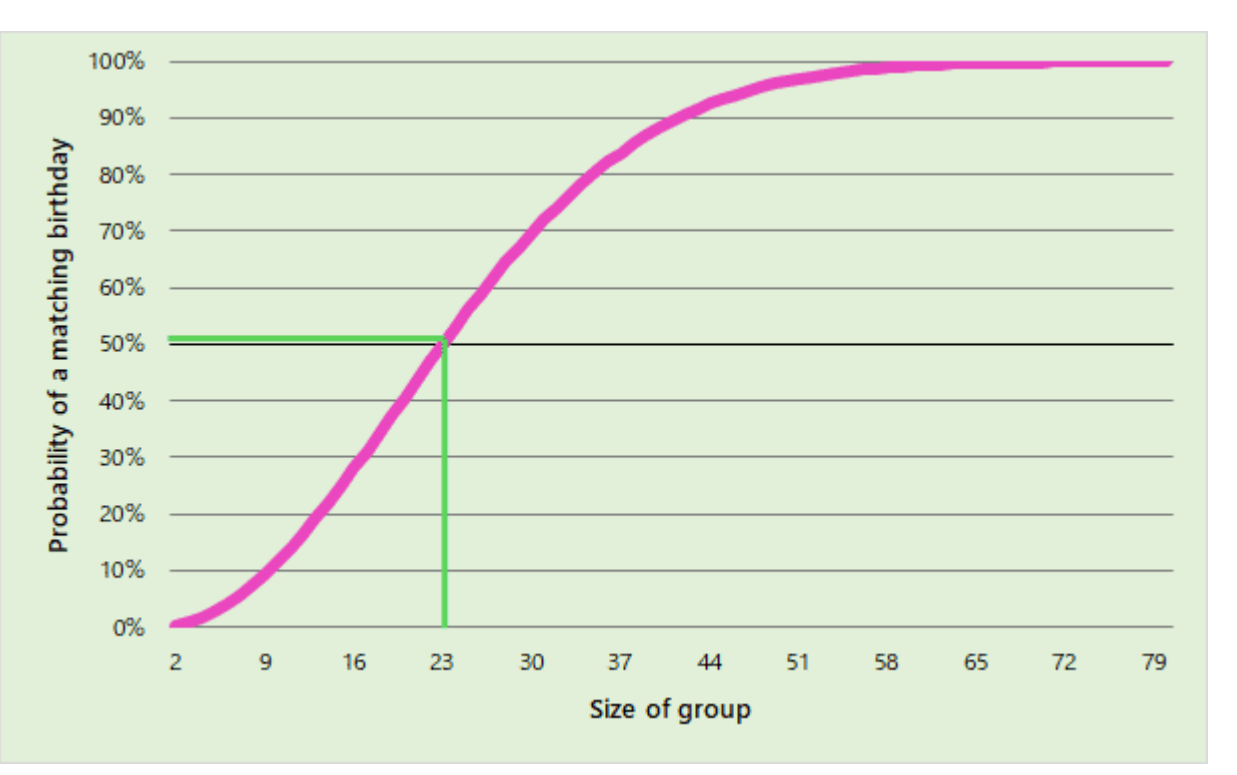
每个结点实际上面临多重限制：

1. firewall 打洞：必须要先往 `dest ip:port` 发送包，这个 dest 才能回包
2. easy NAT 打洞：必须要先发送任何包，NAT 上才会绑定公网 `ip:port`
3. hard NAT 打洞：必须要先往 `dest ip:port` 发包，NAT 上才会绑定公网 `ip:port`，并接受特定 `dest ip:port` 的回包

通过 STUN，easy 可以获取自己的 `ip:port`，hard 可以获取自己的 `ip`。

先介绍下 **Birthday Paradox**

N 个人在一起，其中至少有两个人是同一天生日的概率。当 N > 23 时，概率就超过了 50%。



Birthday Paradox 在随机碰撞中，是一个非常强大的理论。

对方在 M 的范围内提供了 N 个可选项，你在 M 内随机选择，撞上至少一个 N 的概率随着尝试次数的增多是显著增长的。

发起 birthday attack：

1. hard 往 easy 的 `ip:port` 建立 256 个连接，在 NAT 和防火墙上开了 256 个洞
2. easy 开始往 hard 的 `ip` 遍历发包，每次都使用随机不重复的 `port`。

如果恰好随机生成的 `port` 就是 hard 打过洞的 `port`，那么双方就能成功握手。 根据概率论，256 次尝试就能有 64% 成功率，1024 次尝试可达到 98% 成功率。

假设 100 packets/sec，平均 2.5 秒，最坏 10 秒，就能建立连接。

但是如果双方都是 Hard NAT，碰撞成功率就会下降到约等于零。

所以这个方法只能用于 Hard & Easy 的组合。

uPnP

Birthday Paradox 虽然，但是看起来实在是有点想端口扫描，可能会在 IT 层面引起不必要的麻烦。

更好的方式是，如果 NAT 设备允许后方的结点主动注册一个公网端口，就不需要费尽心力地去打洞了。

类似的协议就被称为 uPnP，苹果后来推出的 NAT-PMP，以及业界升级后的 PCP 等。

但是 uPnP 非常复杂，老设备的实现往往存在缺陷，所以管理员倾向于默认禁用 uPnP。

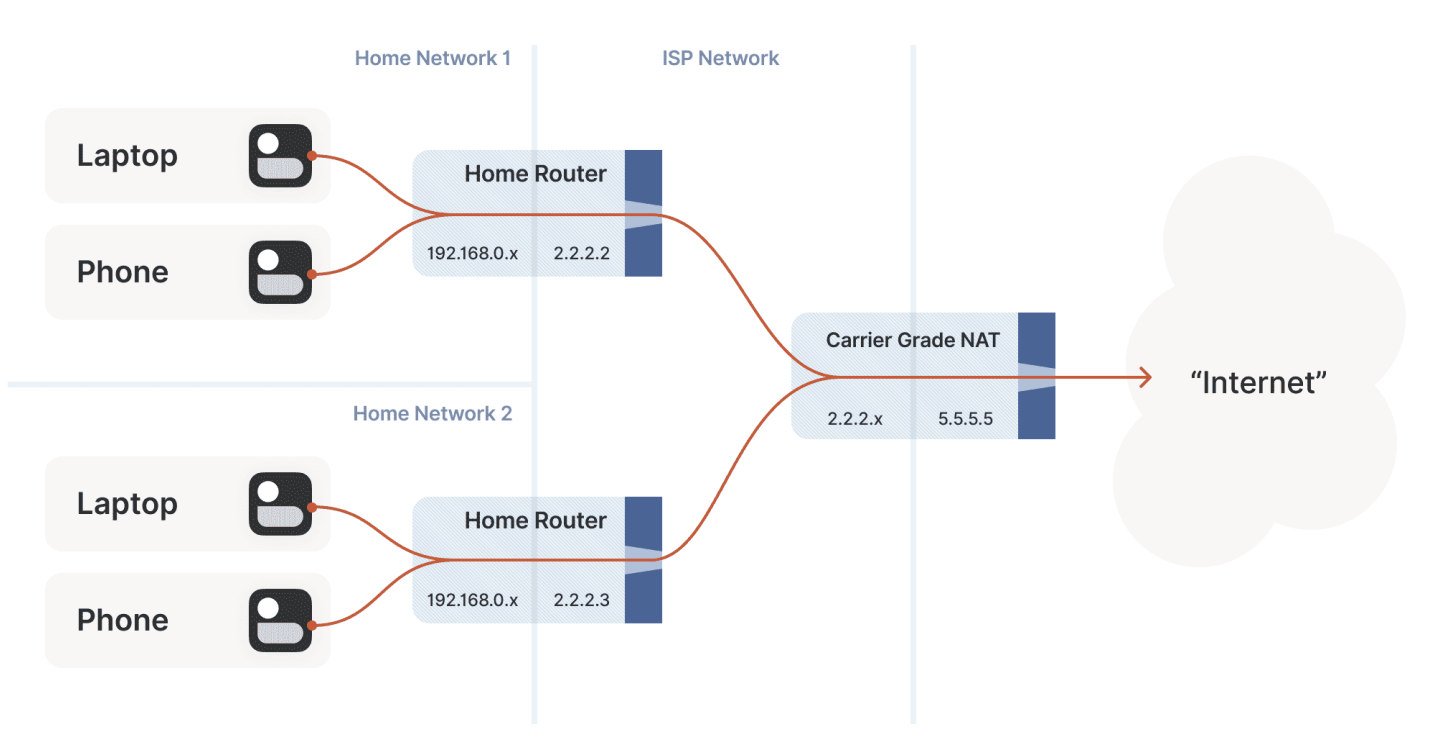
而很多设备使用同一个开关来控制 uPnP、NAT-PMP、PCP，所以往往导致所有的这三个协议都会被默认禁用。

CGNAT & Haripinning

这一节和 NAT 穿透并没有太大关系，使用的都是前文所论及的旧技术。

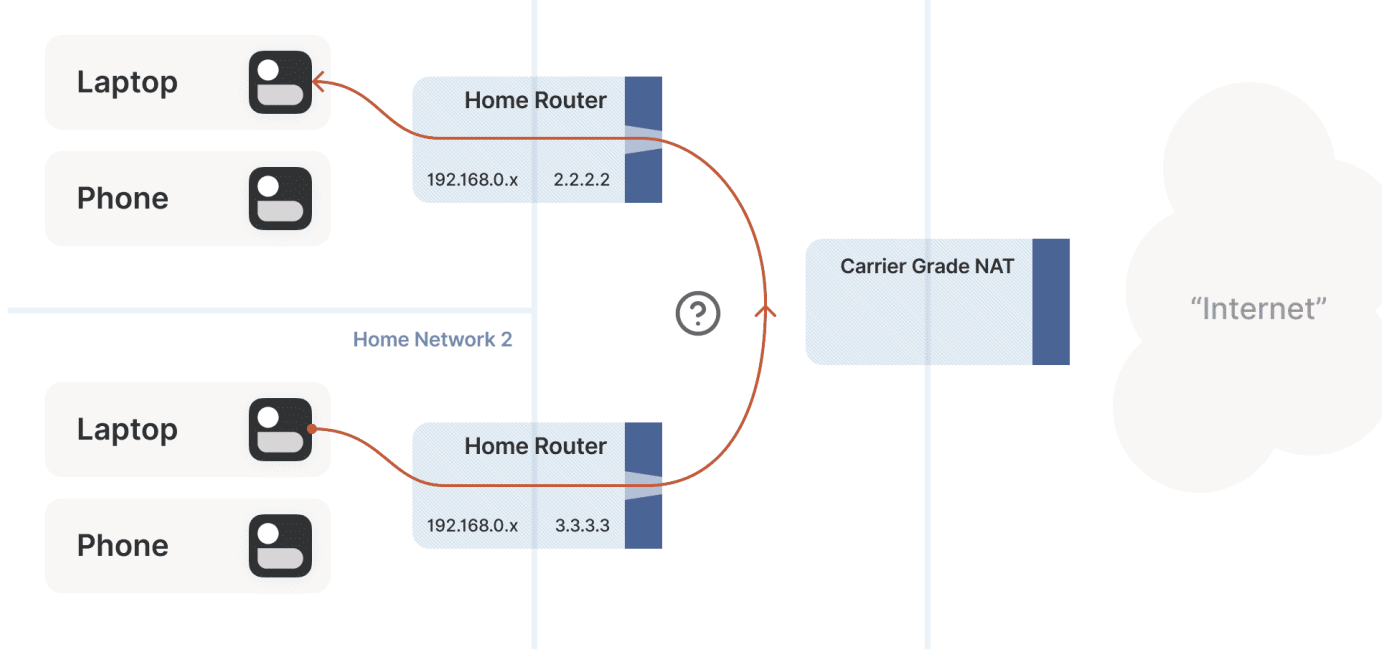
在 ISP 运营商场景下，运营商出于成本，很可能不会给每个用户都分配公网 IP。 而是利用 NAT，给用户分配私网地址，共享一个公网 IP。

这种 NAT 称为 CGNAT，Carrier Grade NAT，运营商级 NAT。



这种情况下，两个处于同一个 NAT 内部的结点要想实现直连，就需要依赖 hairpinning 技术。

支持该技术的 NAT，发现一个数据包的 dst 指向的是自己的公网地址时，就会改写数据包，将其转发给在同一个私网内的对端结点。



ICE

Interactive Connectivity Establishment (ICE)，用 tailscale 的话来说就是：

> try everything at once, and pick the best thing that works.

简而言之就是，首先使用 relay 保证联通，然后在后台遍历尝试所有的直连方式，如果有直连方式成功了，就切换到直连方式。

Thank you