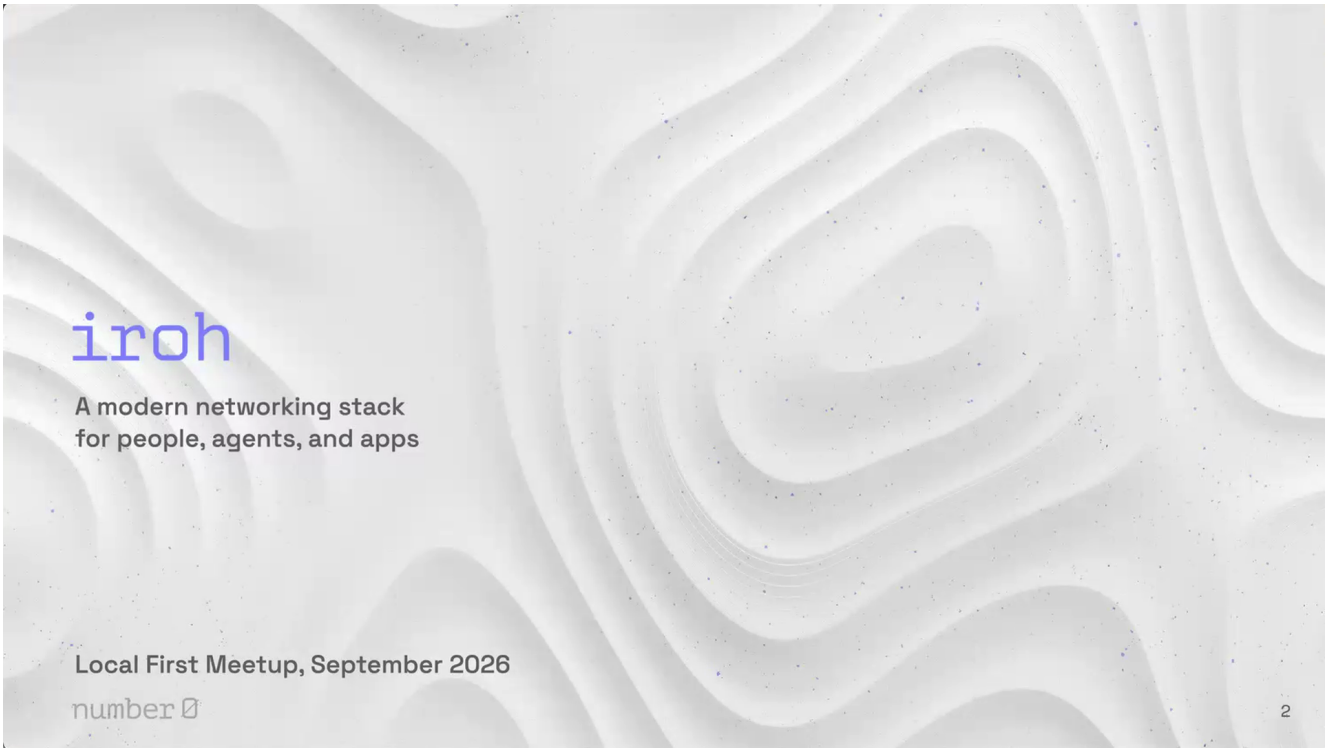
iroh 协议：网络韧性的基础设施

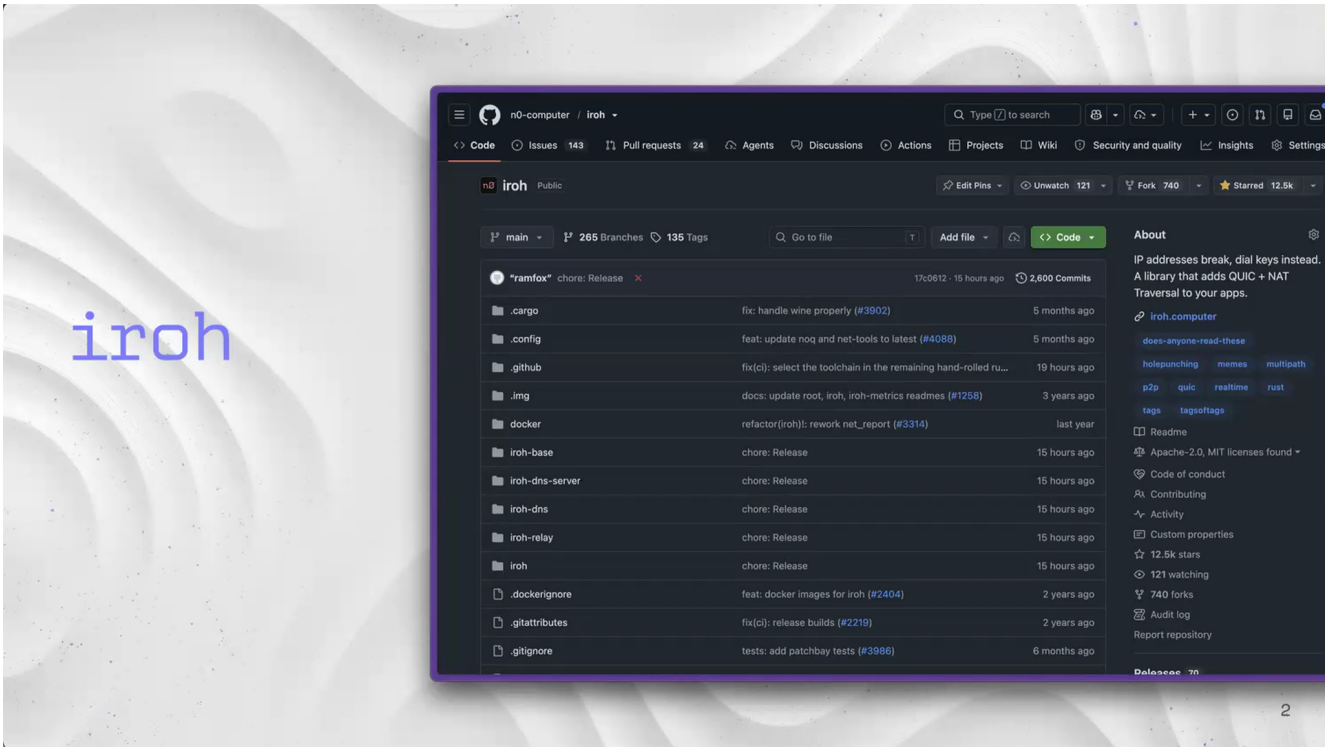
用公钥拨号，而不是 IP 地址

第一次本地优先会议 · 2026 年 9 月 22 日

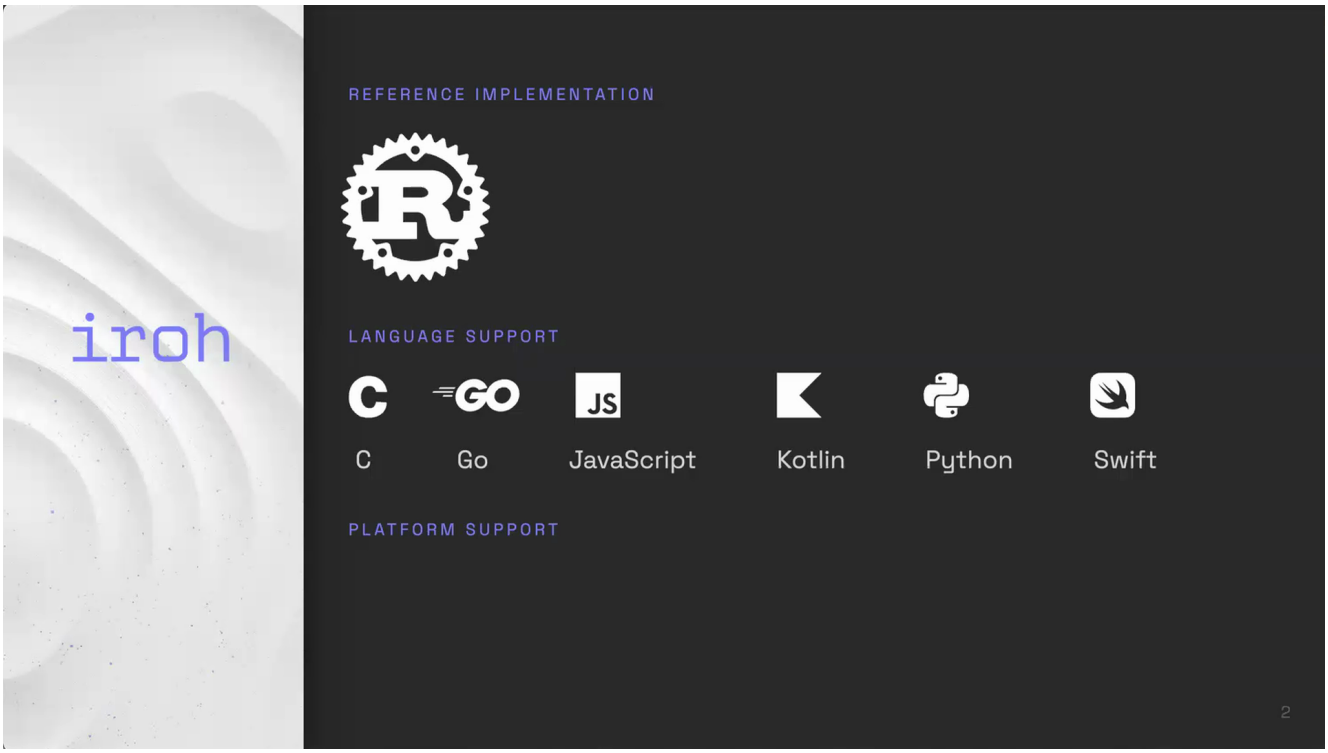
分享者：Brendan O'Brien（iroh / number 0）



大家好。很高兴能和各位聊聊，今天我来讲 iroh。有些人以前可能听说过 iroh，但我会从头讲起，所以如果你完全不了解它是什么、它和 local-first（本地优先）有什么关系，希望今天你能把这些都弄明白。



iroh 的内核是一个网络栈（networking stack），是一份开源软件。iroh 是一个库，现在就挂在 GitHub 上，你可以直接拉下来用。你对 iroh 做的事情，就是把它嵌进你自己的应用里——用法和你引入一个 HTTP 栈或者别的什么库是一样的。它想做的事，是让你可以对自己的网络提出更高的要求。这句话今天会反复出现。



iroh 是用 Rust 写的，从一开始整个项目就是 Rust，参考实现就在那里。项目已经发展了一段时间，让我们很高兴的是，iroh 已经有了第一个独立实现，是用 Go 写的，而且是照着 iroh 协议本身从零实现的一遍。我们也支持其他语言，目前已提供 C、JavaScript、Kotlin、Python 等多种语言的绑

定；如果还缺什么，我们可以再加。这些大多是通过 binding 做的，也就是到 Rust 内核的 Cbinding。这么做的好处是：不管你用哪种语言，你的网络栈底下都是一套很快的 Rust 基座。

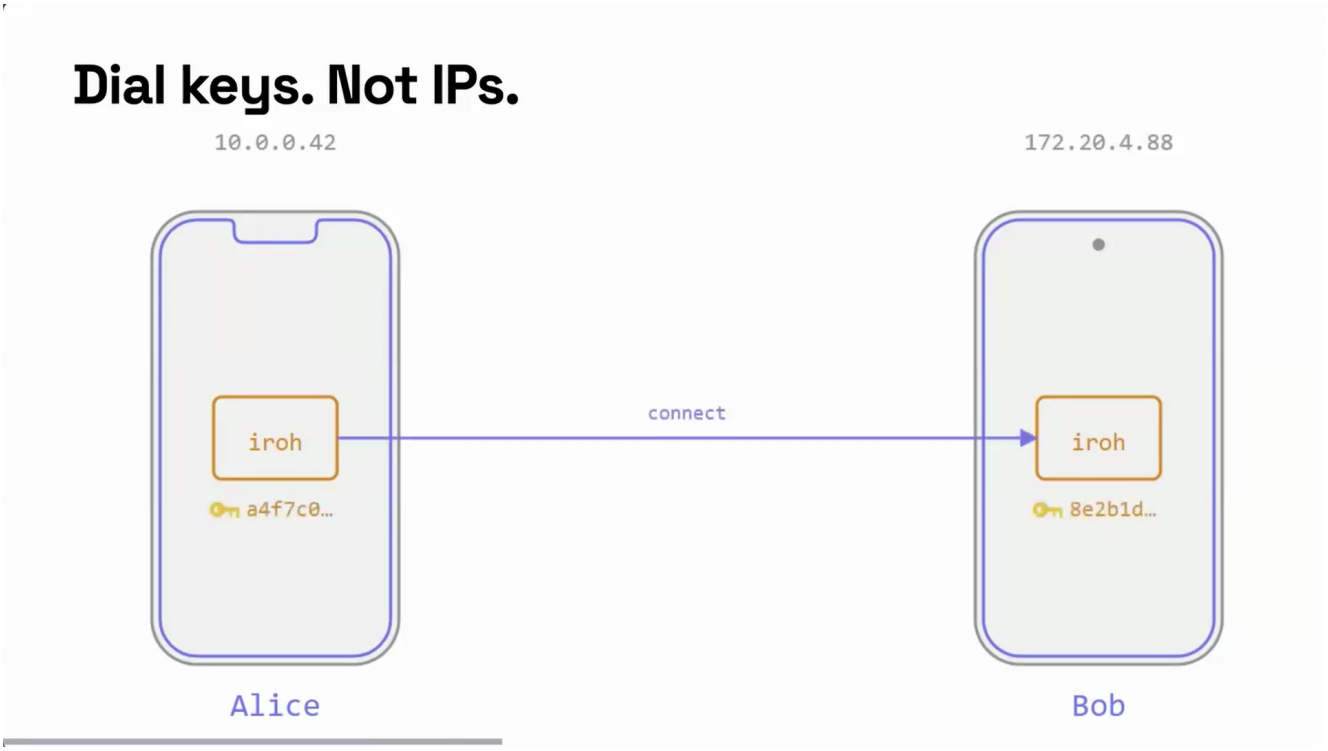
平台方面：iroh 可以编译成 WebAssembly，在浏览器里跑；也可以跑在 ESP32 这种非常低功耗的设备上——现在我们能做到的最小二进制大约 1.5 MB，而运行时内存可以低于 2 MB，所以整套系统可以跑在极小的设备上。你能想到的主流平台我们都能跑：Windows、Linux、FreeBSD、macOS，全都可以，而且我们在生产环境里持续测这些。



那么，为什么要再写一个网络栈？核心想法是什么？我们的核心想法是：所有人都应该去拨号一对密钥（dialing cryptographic key pairs），而不是拨号一个 IP 地址。

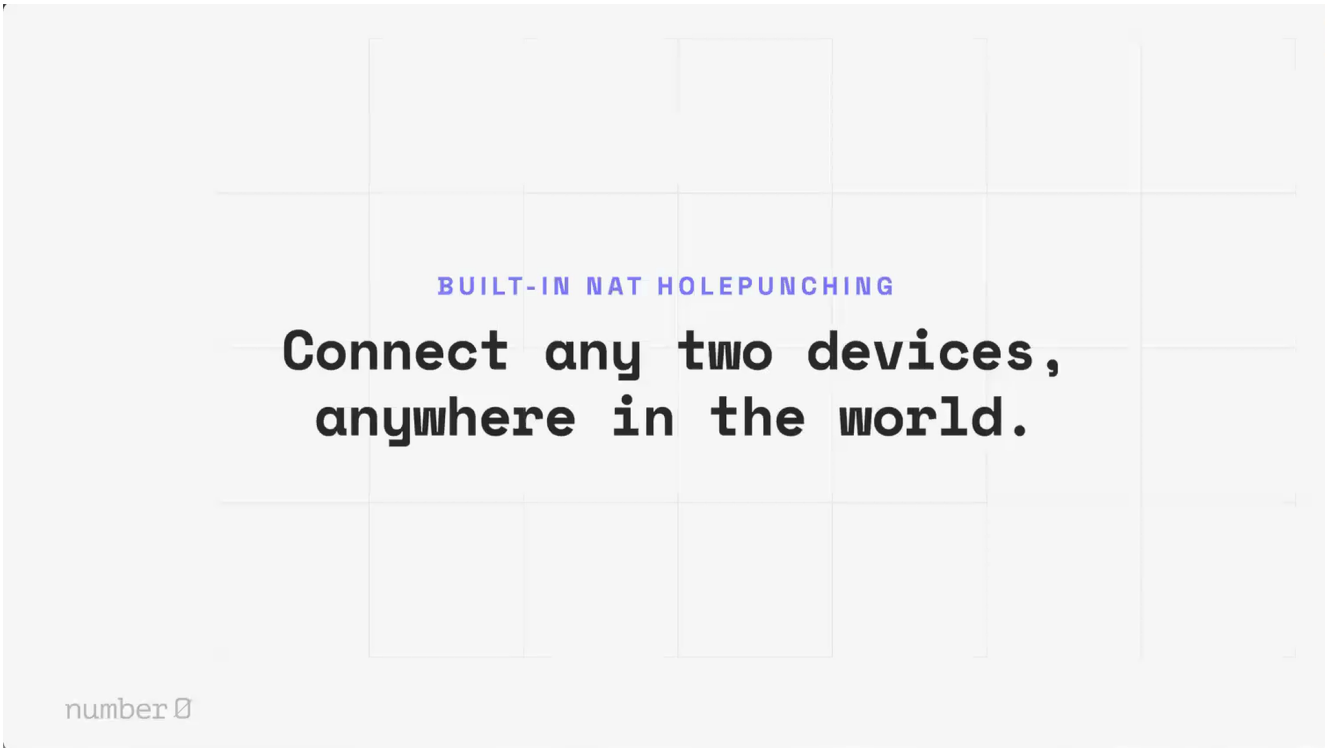
我想请你先设想一下整个互联网，然后设想：原本填 IP 地址的每一个地方，当你要和另一台设备建立连接时，你填进去的是一个公钥（public key）。在 iroh 里，我们不按 IP 地址拨号，而是按公钥拨号。当然我们改不了互联网的运行方式，所以底层有一大堆东西负责把这些公钥翻译成 IP 地址。但作为开发者，你打交道的核心原语（primitive）就是公钥。我们用这些公钥给你设备之间的端到端加密连接。所以你永远是这样：我要拨这个公钥——当你和它通话时，你拿到的是一条由那把密钥保护的 TLS 连接。这里很有意思的一点是，在你通信的过程中，每一个包都由你正在对话的那把密钥签名。这就是一个非常令人兴奋的新地基。

在 iroh 里我们还把它和打洞（hole punching）结合起来，所以 iroh 能穿透 NAT，这意味着你可以拨通地球上任意两台设备。无所谓在哪。你的手机和你的笔记本可以互相通话；两台在不同云上的服务器也可以互相通话——只要你别在网络策略上做特别离谱的事情把它堵死。所以 iroh 不是彻底的魔法，但它相当神奇。

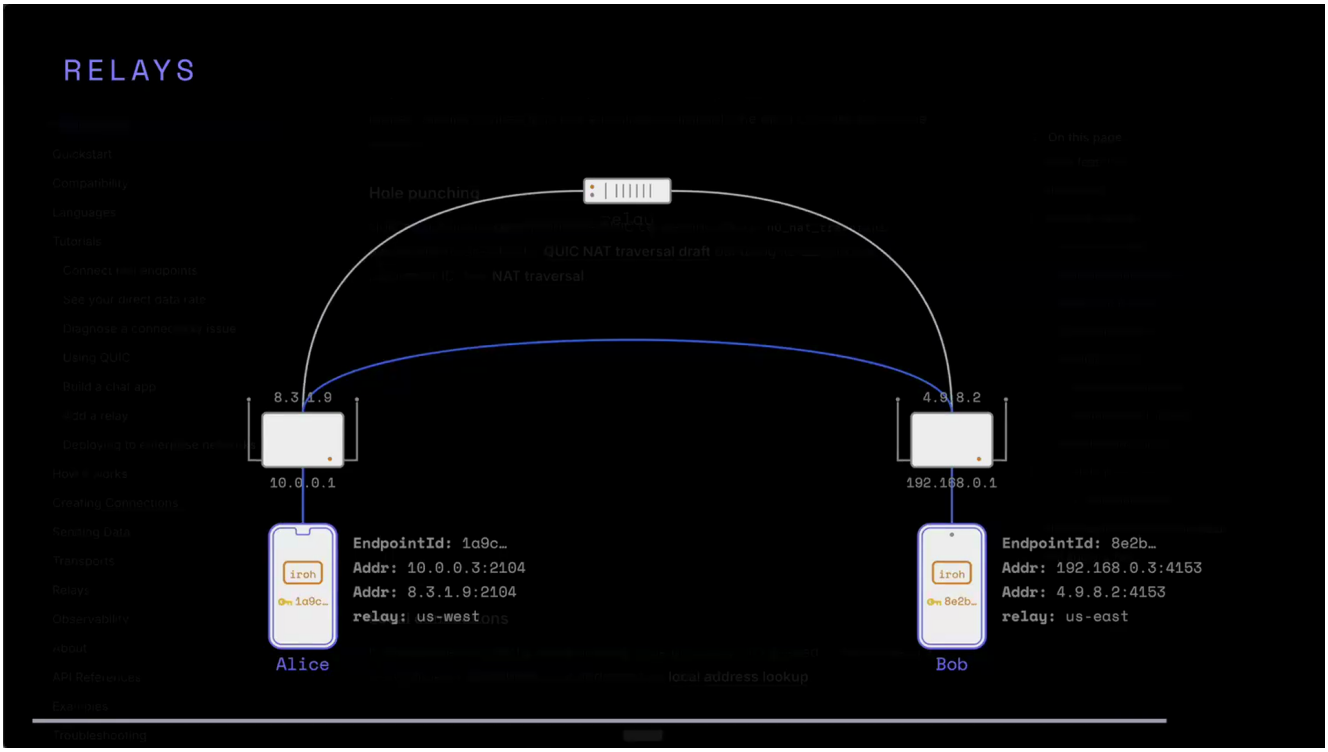


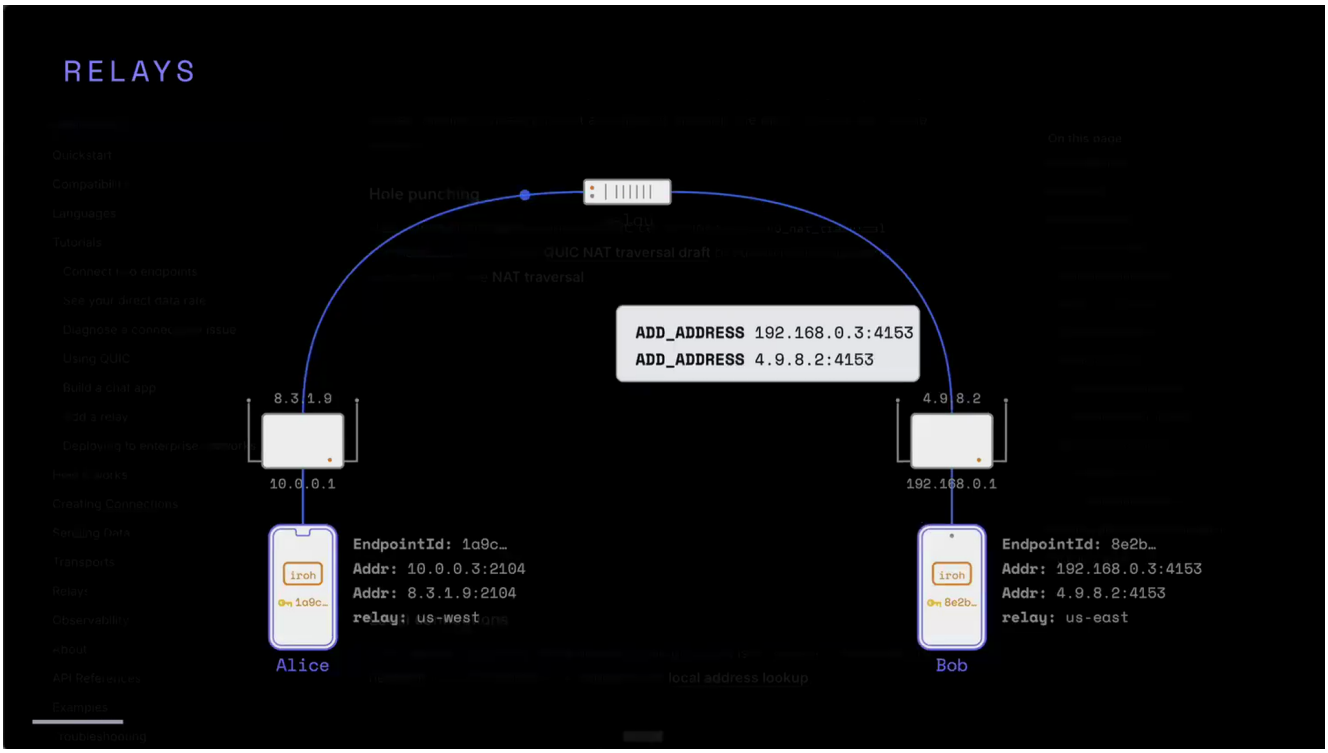
底层大致是这样运作的：IP 地址是会随时间变的，所以当我们尝试连接时，实际上是在两把密钥之间做建连。这里要记住的一点是，每把密钥对应一个进程（per process）。你可以设想：你的应用，以及你应用的某一个用户，拿到这一把密钥，于是你就有了一个不随时间变化的标识。

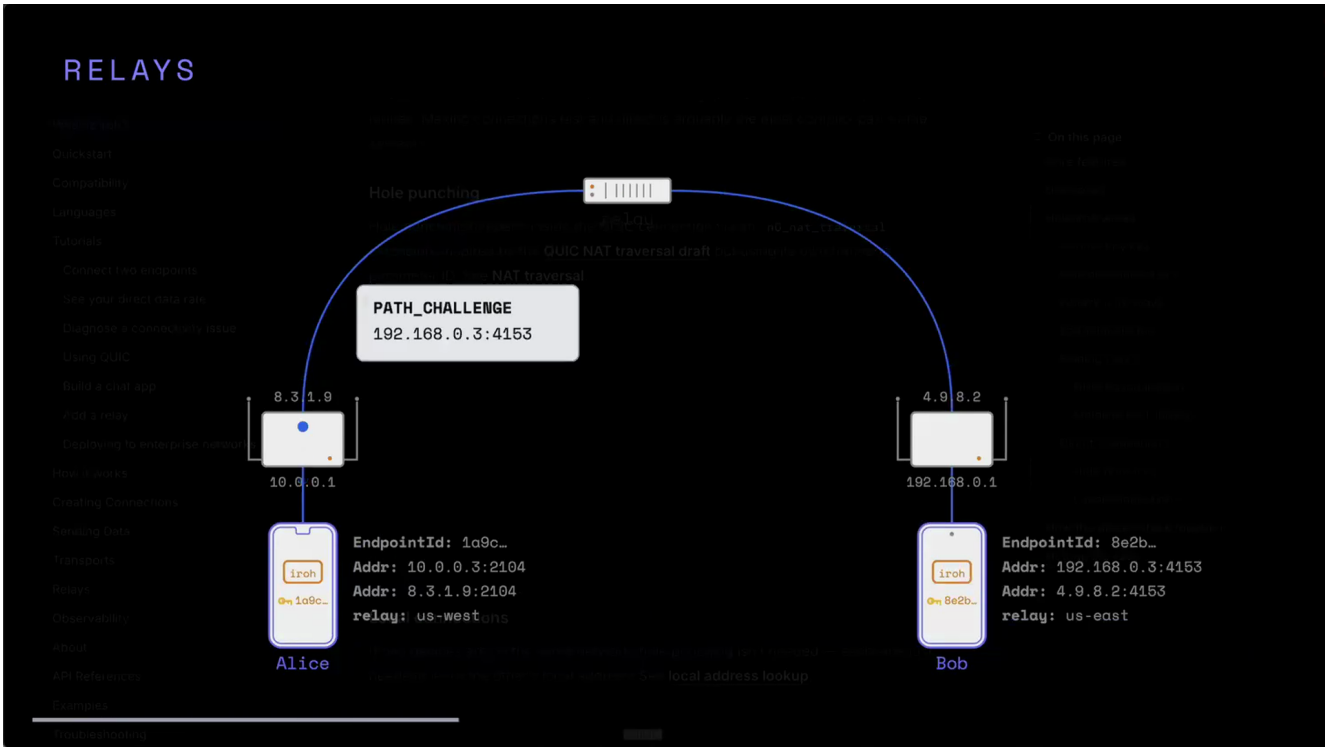
这就是相对于以 IP 地址为地基的最大优势。用 IP 地址的话，你的路由器随时可以决定你的 IP 是什么，并且替你改掉。在 iroh 里不是这样。我们可以说：这就是我希望别人拨过来的那把密钥。你在世界上的哪个位置都无所谓。你走来走去，在 Wi-Fi 和蜂窝网络之间切换，或者插上一根网线，这条连接都会保持住，并且认的是你要对话的那把密钥。这是按这种方式思考所带来的核心优势之一。

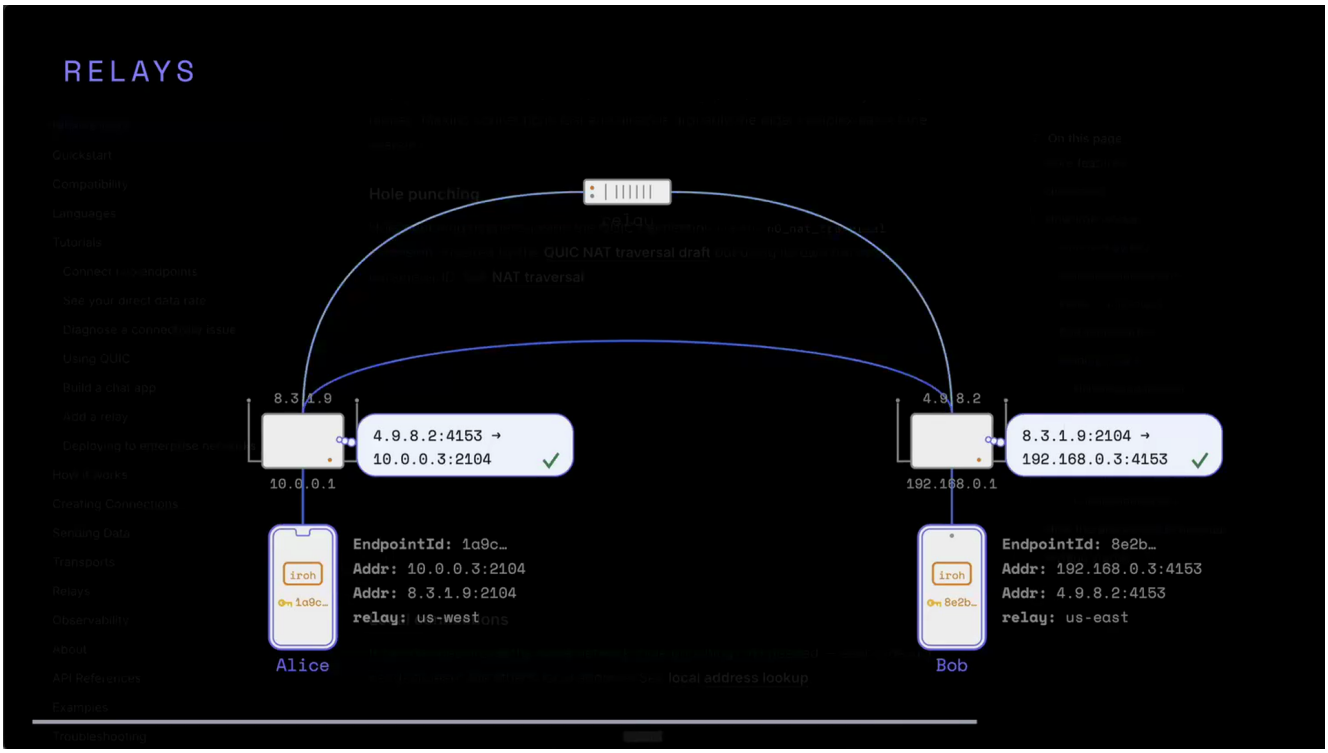


所以整个想法就是：我们把地球上任意两台设备连起来。









为此我们写了一套叫中继（relay）的技术。iroh 确实要依赖中继服务器来建立这些连接，因此每一个端点（endpoint）在任一时刻都有一个我们称为 home relay 的归属中继。当一个 iroh 端点启动时，它会去 ping 预先给定的那份列表里它知道的中继，然后据此确定：我希望别人以这种方式联系到我。

接下来，当两个端点想要互相通话时，它们就开始执行这套连接流程：其中一个查出对方的 home relay 在哪，然后说：我想连你。中继把这条消息送到那台设备，与此同时，我们并行地开始协商一条直连。所以 iroh 一直在切换，找当下可能的最佳连接。

一般来说，你在这张图里看到的这个过程，发生在第一个包之内。通常我们会看到一个加密的数据包流经中继。中继是完全盲的——它看不到设备之间的任何通信内容。但第一个包倾向于走中继，为的是拿到很快的首字节时间（time to first byte）。在那之后，按惯例——或者说在很多情况下——我们那时已经通过你看到的这段动画里的路径挑战（path challenge）流程，在两台设备之间建立起了直连。一旦建好，它们就切过去。之后如果你的连接在任何时刻发生变化，我们随时可以退回中继，再重新接回直连。

所以在底层，这就是 iroh 给你的抽象。这些你都不用操心，它替你做完了。你只要说：给我一条能用的连接，给我我们手上最快的那条。如果因为某种原因我无法与那台设备建立直连、必须退回中继，它照样能用，而且不会牺牲这条连接的安全性和加密。

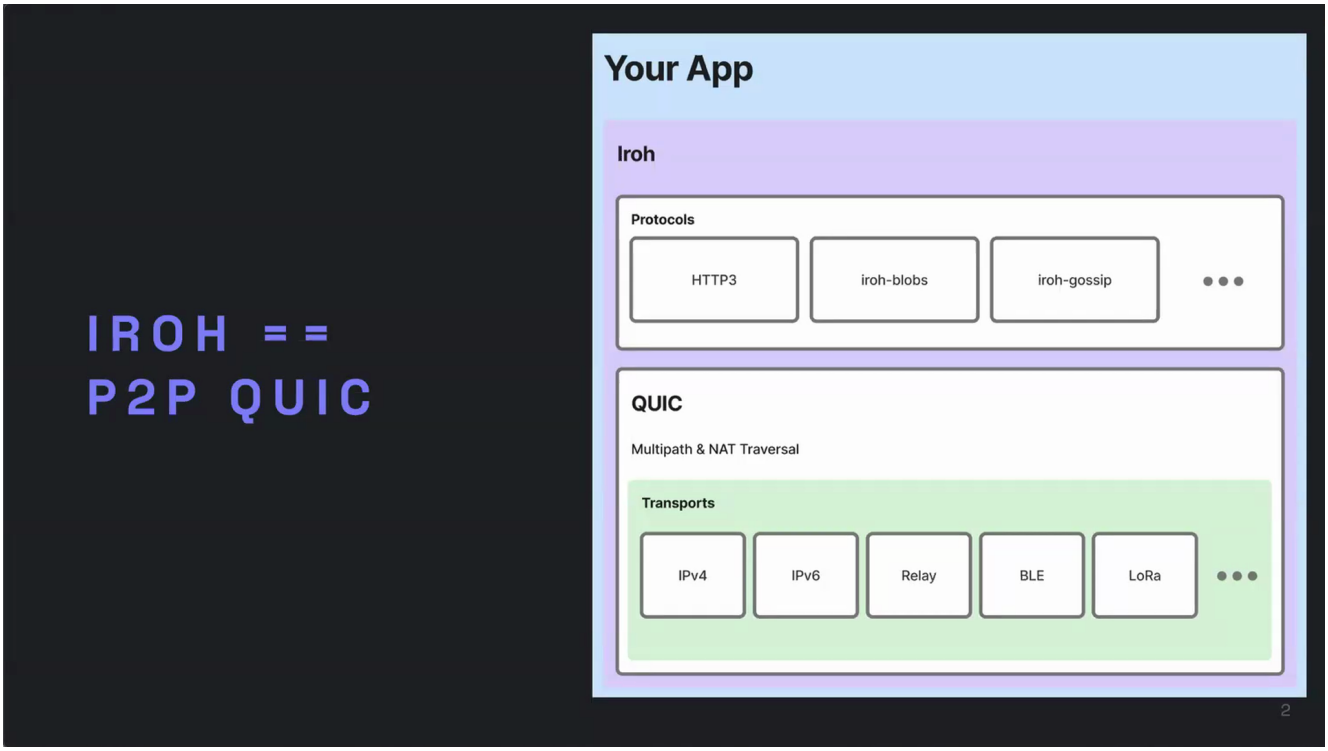
```rust
use anyhow::Result;
use iroh::endpoint::{Endpoint, presets::N0};

#[tokio::main]
async fn main() -> Result<()> {
    let endpoint: Endpoint = Endpoint::bind(preset: N0).await?;
    println!("my public key: {:?}", endpoint.id());

    tokio::signal::ctrl_c().await?;
    Ok(())
}
```

我给大家看一小段 Rust 代码。在很多别的语言里也是这个样子，我只是想让你对它长什么样有个感觉。你不需要写 Rust 才能用 iroh，但大体上就是这个样子。

流程是这样：我们引入 iroh 库，然后声明一个端点——我要建一个 endpoint，我要用我们称为 preset 的这个东西来 bind 它。而 number 0——也就是我们公司——运行着一组公共中继，你可以直接拿来连，开箱即用。所以这段代码原样就能跑。不用账号，没有任何前置要求。到这一步，这个函数返回时，你就拿到了一个带标识、可以被拨号的端点。那一行代码，做完了我们前面看到的全部过程。再往下我们只是在等退出信号。所以这只是一个开着、在等着、其实什么也没干的端点，但它能让你看出这个库有多强——这是一个非常小、非常快的抽象。



这就是一个端点的 bind，我们引入了那个 preset。

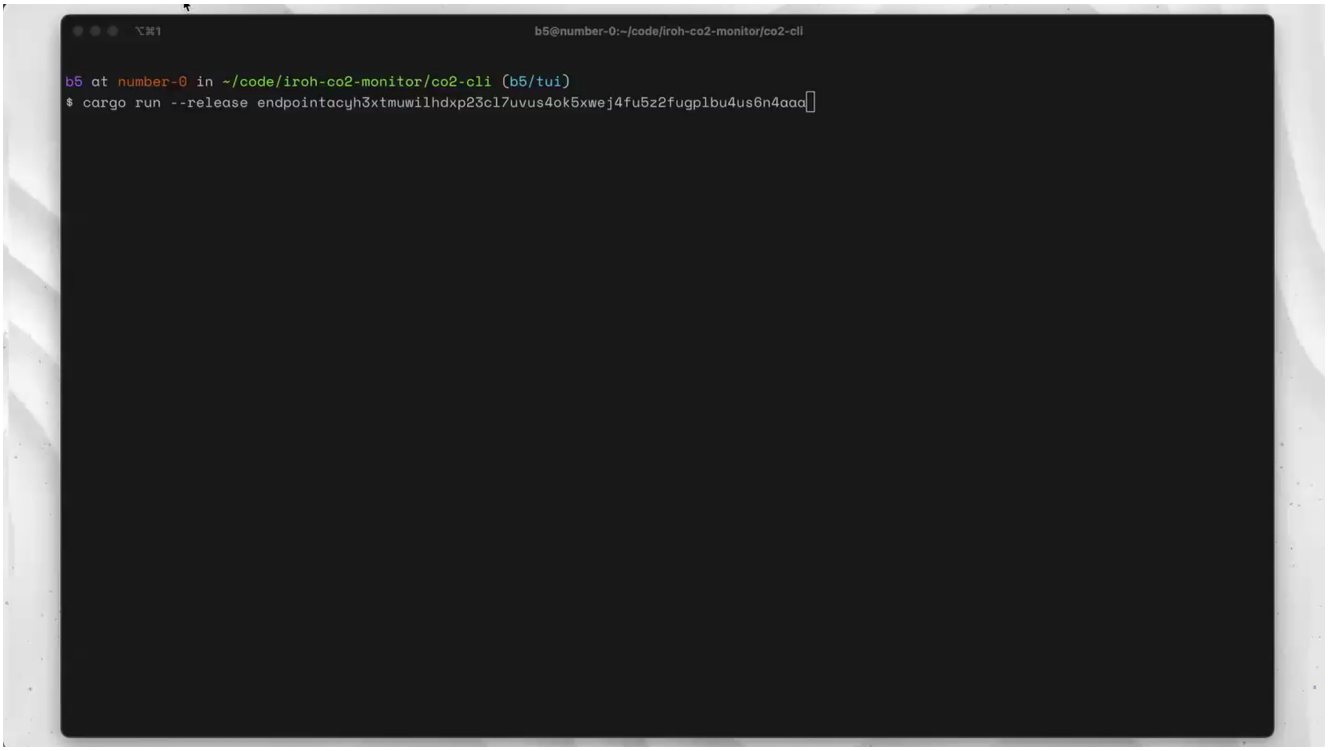
我们倾向于把 iroh 理解成：架在 QUIC 协议之上的点对点（peer-to-peer）。不知道你有没有听说过 QUIC，它是由若干大公司共同发展出来的技术，目前 HTTP/3 就是基于它定的规范。今天互联网上有相当大比例的流量跑在 QUIC 上。QUIC 的目标是取代 TCP，Google、Amazon 这样的大公司都在给 QUIC 规范做贡献。

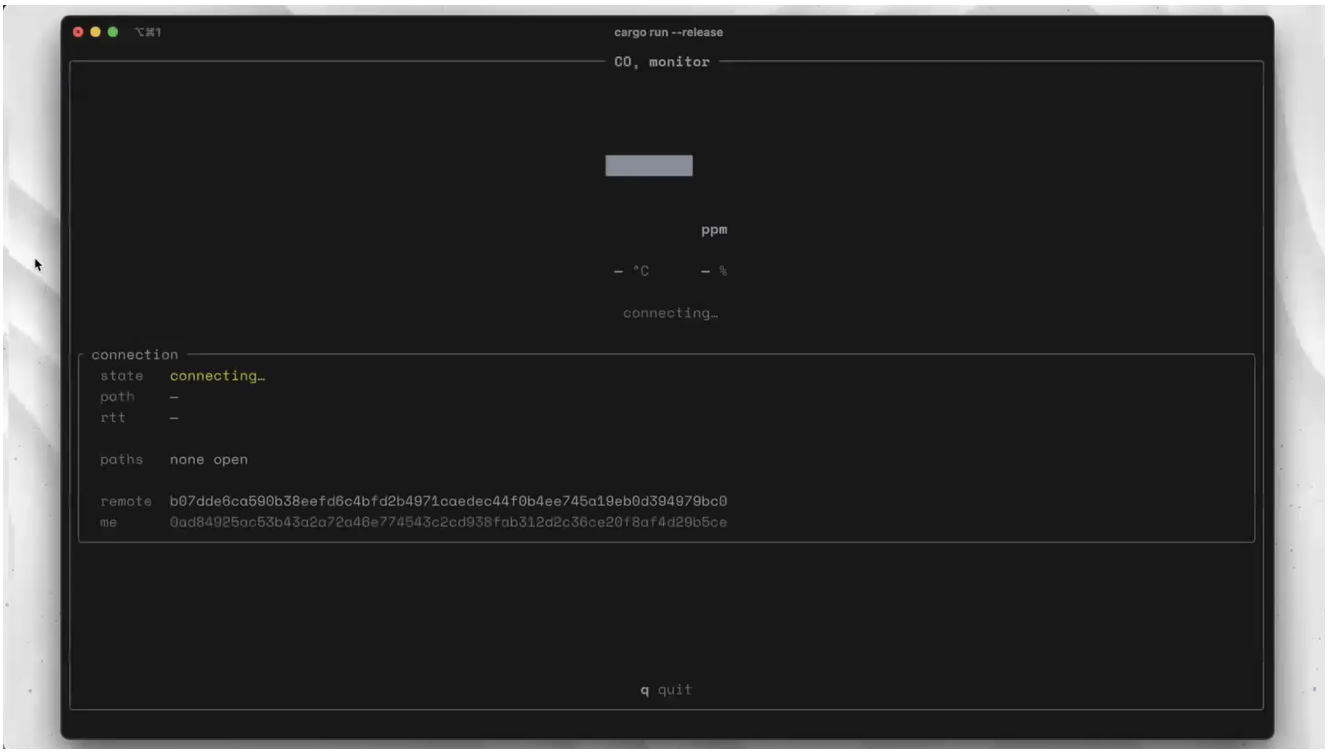
我们喜欢把 iroh 想成这么一个形状：我们在 QUIC 之上给你一批协议，其中之一是 HTTP/3，但也有社区做出来的各种很有意思的协议。iroh-blobs 是其中之一，用来传 blob 数据——如果你想搬文件，这有点像 iroh 版的 S3。iroh-gossip 很适合广播消息——如果你想搭一群端点组成的集群，从任何一个位置把一条消息广播给所有人，那就用 iroh-gossip。

再往里看，在 QUIC 内部我们还有一层对传输（transport）的抽象。iroh 支持可插拔传输（pluggable transports）的想法。开箱自带的是 IPv4、IPv6 和中继连接；你还可以加上低功耗蓝牙（Bluetooth Low Energy），可以加上远距离 Wi-Fi。这些都是你可以插进 iroh 的扩展，插进来之后它就会把这些纳入传输切换的过程，在任何时刻挑最好的那一条。

这一点对 local-first 非常重要——这也是 local-first 第一次真正进入画面的地方。iroh 和常规互联网栈的区别在于，它从设计之初就不要求广域网连接。你在这里看到的这一切，我们可以用 mDNS 发现在本地网络里找到新的对端，然后在两台设备之间直接建连，全程完全不需要中继。整个流程从根上就是围绕这个场景建的：我们可能没有广域互联网，但我们仍然想互相通信。

而且这个过程是动态的：如果中途你失去了广域网，你可以用已经建好的直连把对话继续下去；而中途如果网又回来了，你会像变魔术一样重新能和更广互联网上的对端通话。它会随着网络状况的变化自然地来来去去。我们认为这是一个非常重要的特性。

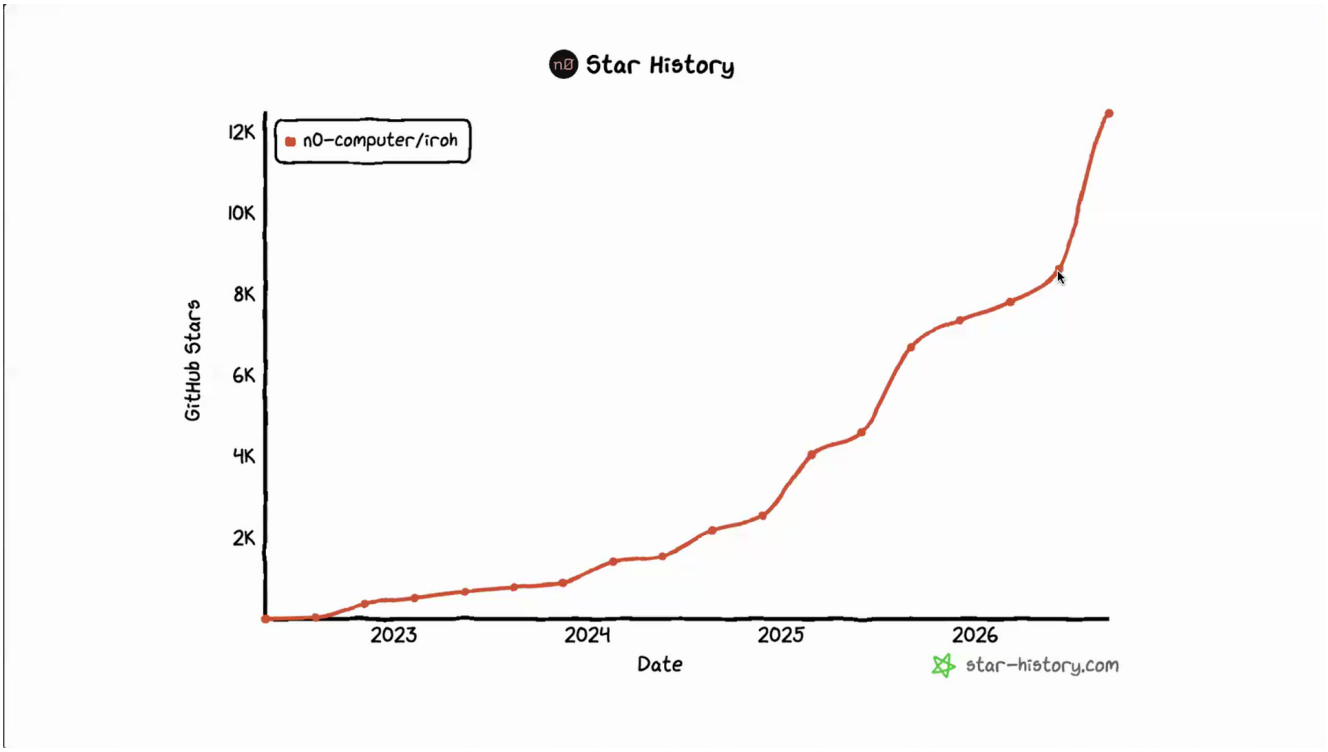




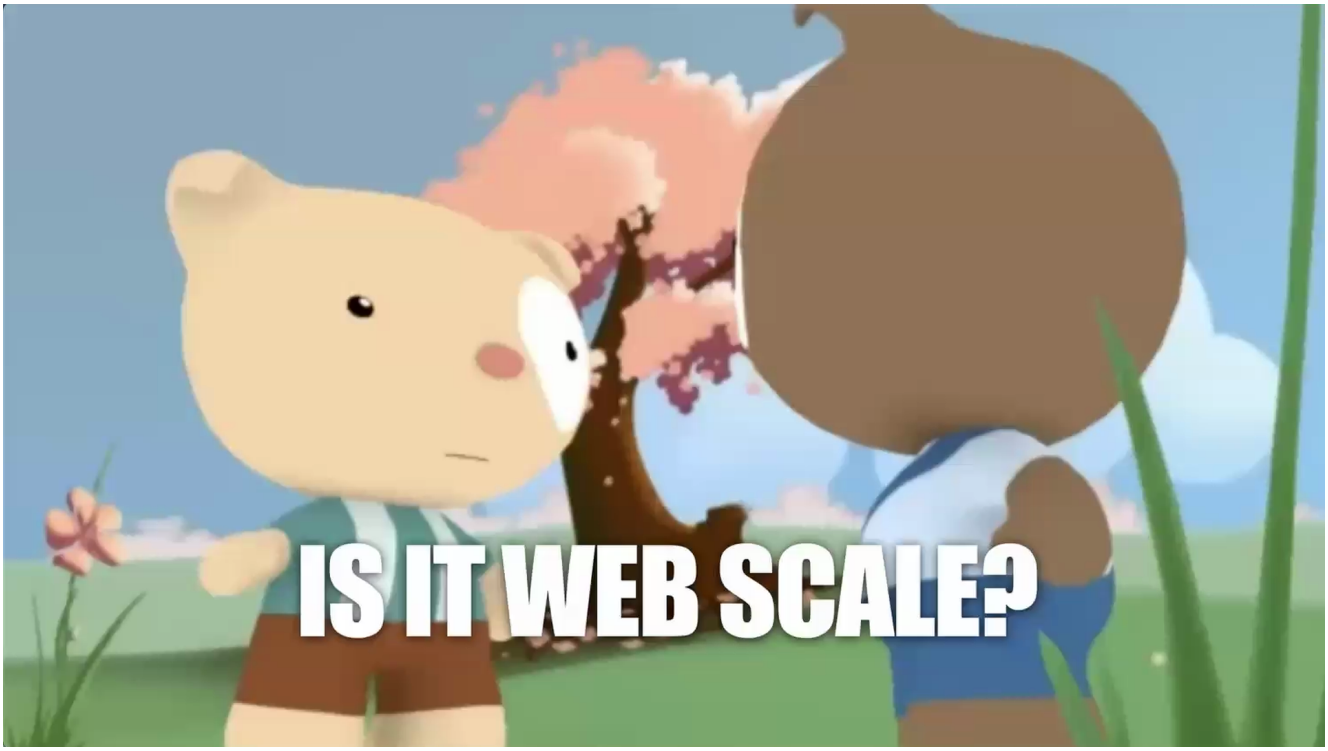
本来我想在这里演示一下，不过我换个东西给大家看：这是连一台 CO2 监测器的例子，那其实是一块 ESP32，每两分钟才采样一次。

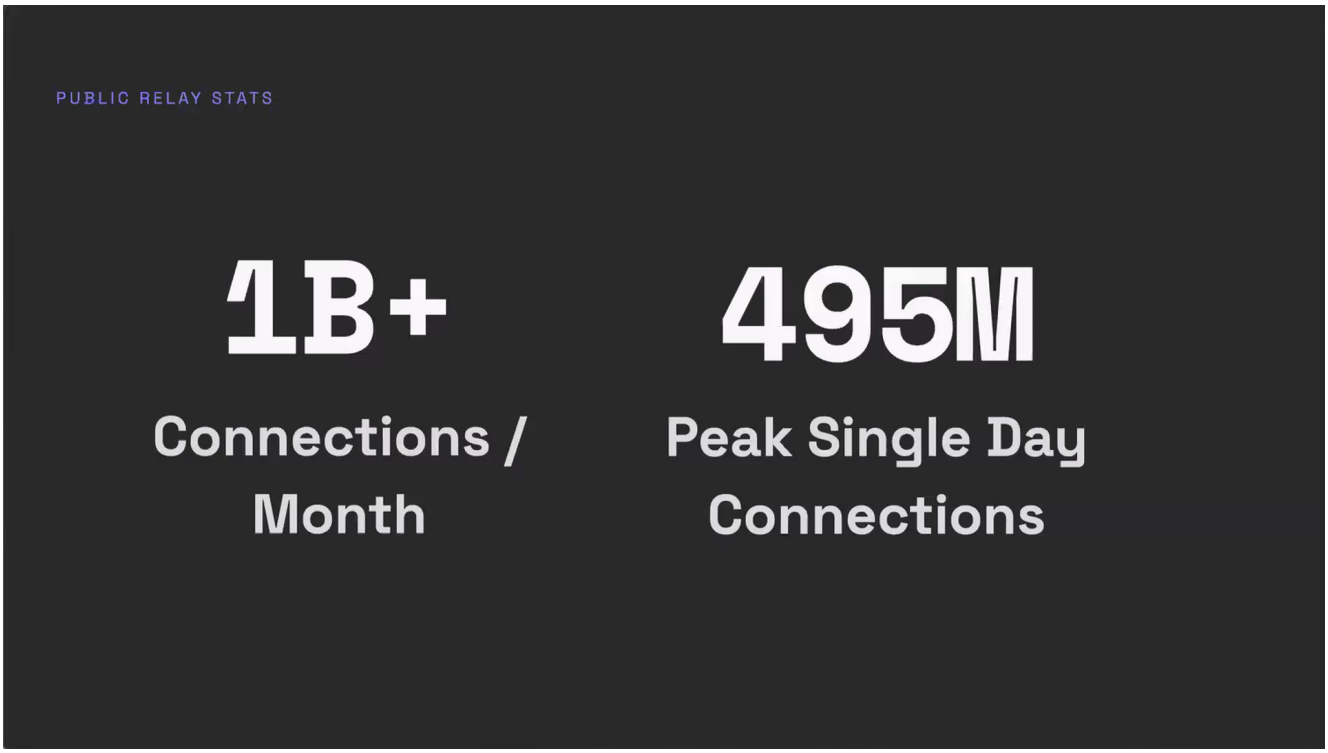


当你写 CLI 和二进制程序，让 iroh 和别的东西直接连起来时，大概就是这个样子。我们还有一些更有用的例子。



这个项目已经发展了很长时间。这个位置就是 1.0 发布的时刻，之后热度有一波不错的上扬。





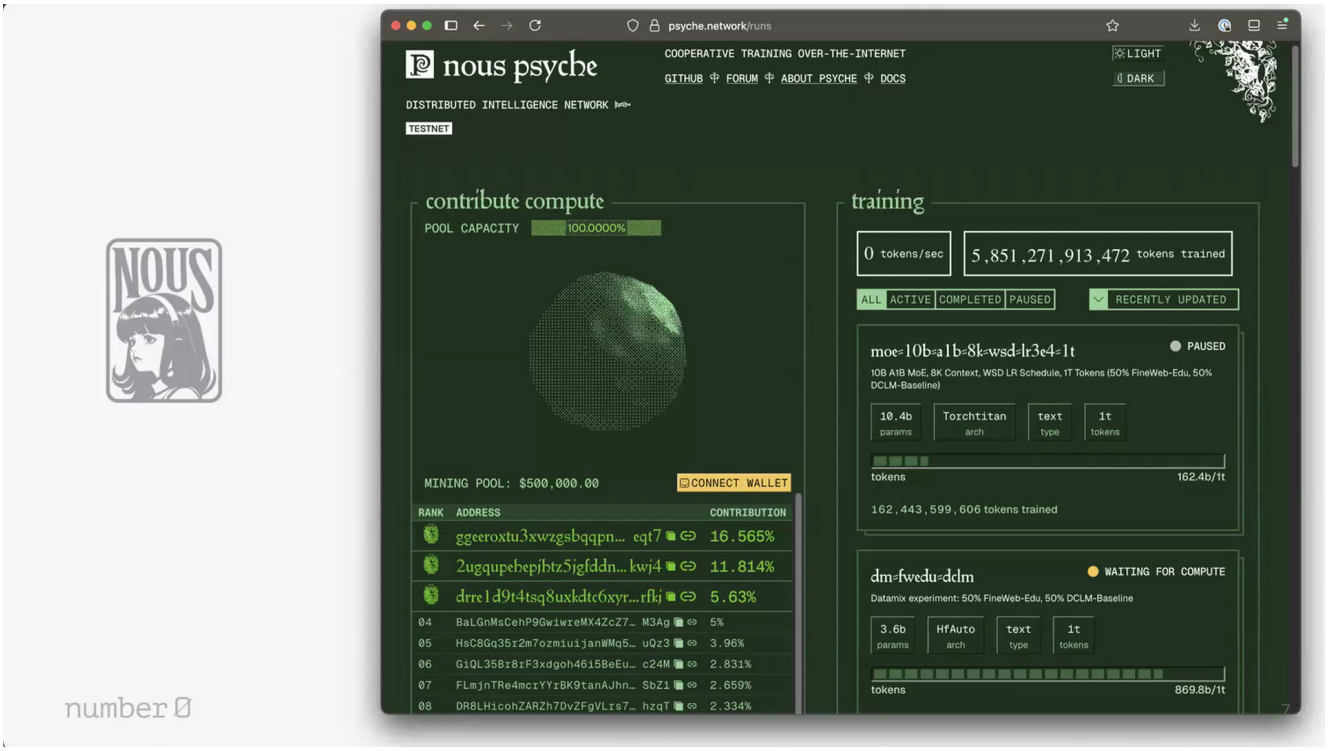
给大家一个规模感：我们的量级是每月十亿次连接；某些日子里，单日连接数接近五亿。所以从我们能看到的中继来看，这套系统本身承载了很多。再强调一次，我们看不到任何流量内容，但能感受到使用规模。到今天，这个项目部署得相当广。



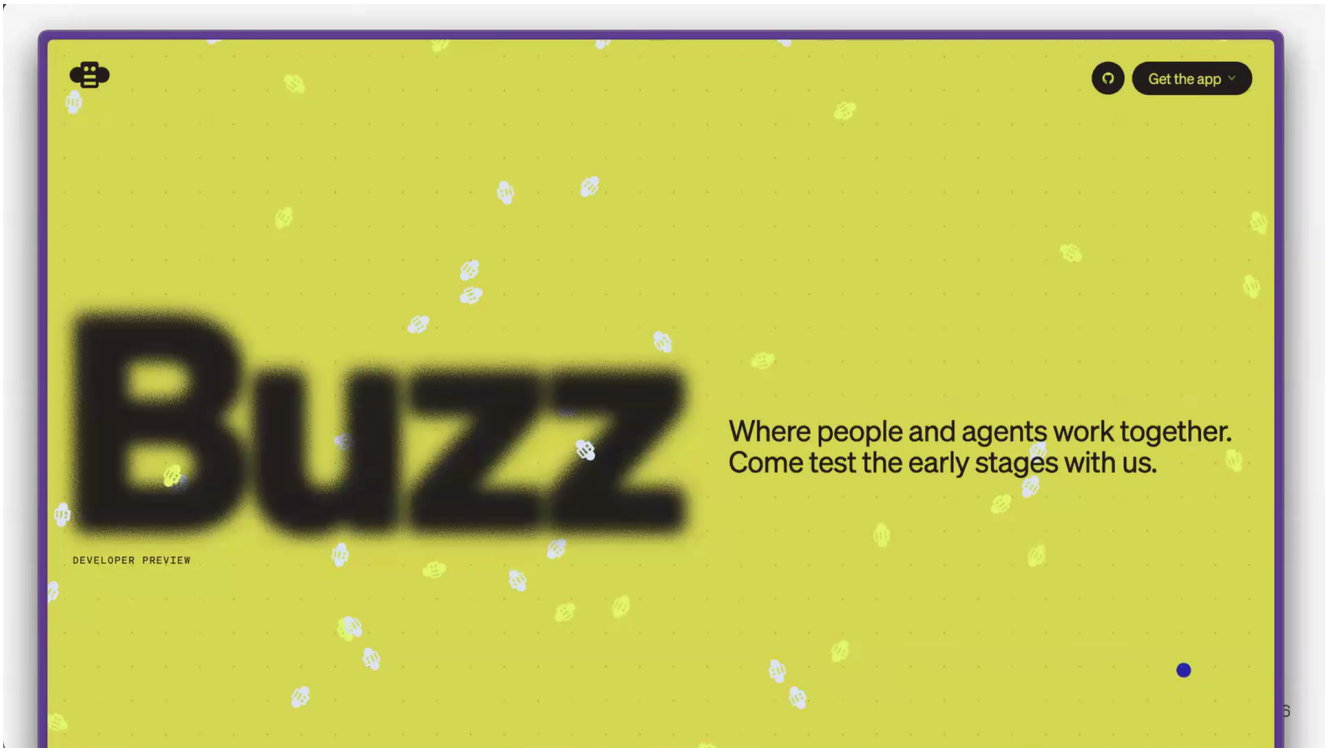
就库本身的使用而言，crates.io 上大约有 200 万次下载；其他几个包管理器里，JavaScript 和 Python 生态我记得是几十万的量级。另外至少有 700 个项目建在它之上。所以有很多人给它提 bug，有很多人在用它。



它被用在各种非常有意思的地方。



我们和 Nous Research 合作过。他们用这套东西在开放互联网上训练了一个基座模型（base model，LLM）。他们把一批 H100 串了起来做基础模型训练，让算力集群可以弹性地加入和退出，其中每一个端点都是一个真实的 H100 GPU 集群，在和别的 GPU 集群通话。这是我们做过的算力最密集的项目之一。



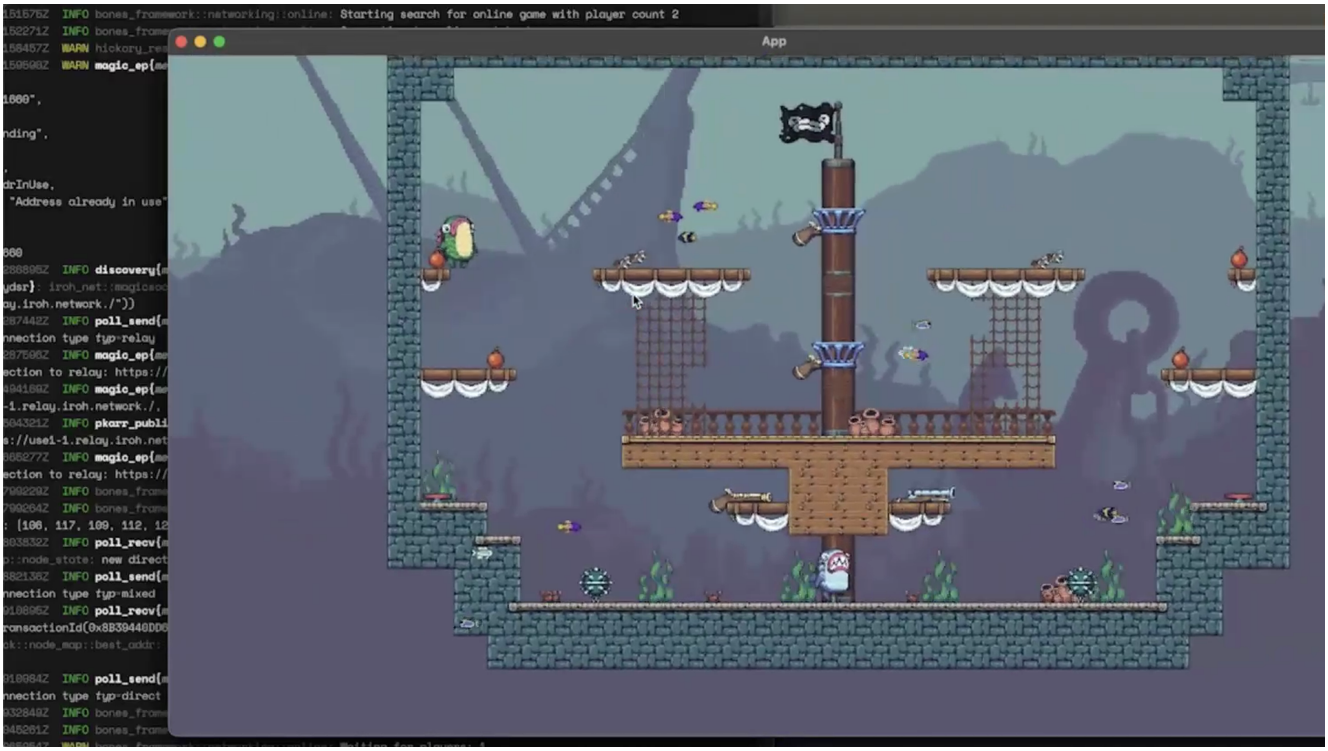
还有 Buzz，一个 agentic 的交互框架——或者说一个聊天应用——建在 iroh 之上，用它来做设备之间以及房间内部的通信。

这个例子是在玩一款电子游戏：游戏的实际渲染发生在笔记本上，手柄输入来自手机。我们能做到 60 FPS、50 毫秒以内的延迟，而应用本身是用 C++ 写的。所以是 C++ 往下调 Rust，把数据发到线上再通信。我们还演示过相距 300 公里仍然可用。



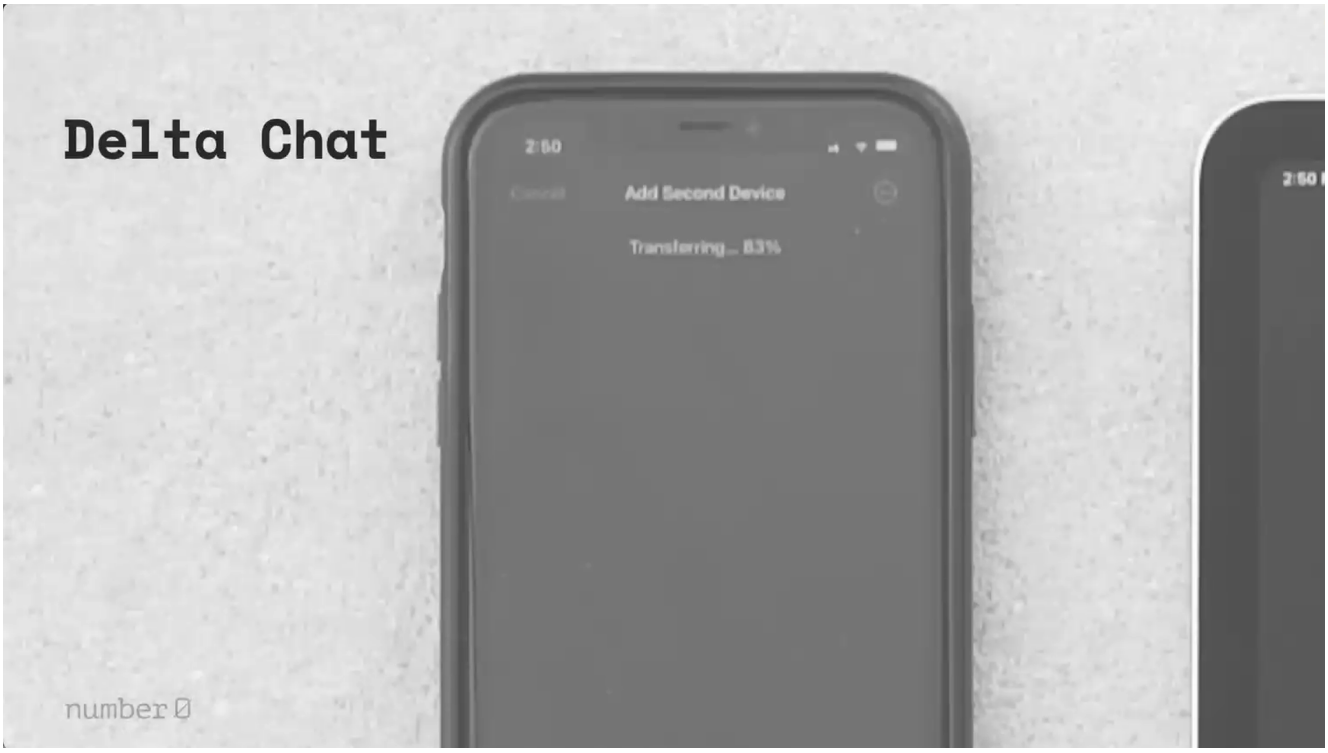
所以这条路是走得通的。你可以用 iroh 做非常快、非常高性能的完整音视频流，这是很常见的用法。

我们也可以做游戏引擎。这里有一个例子，整个网络栈完全由 iroh 驱动。如果你来自 local-first 的世界，这里最酷的一点是：你可以做一款根本不需要存在游戏服务器的游戏。于是即使开发者不再运行服务器，这款游戏也还能继续玩。如果你玩游戏，你一定知道那种悲剧——有人把服务器关了，你就再也玩不到这款游戏了。



这种事不再是必然了。你的局域网联机（LAN party）可以回来了。

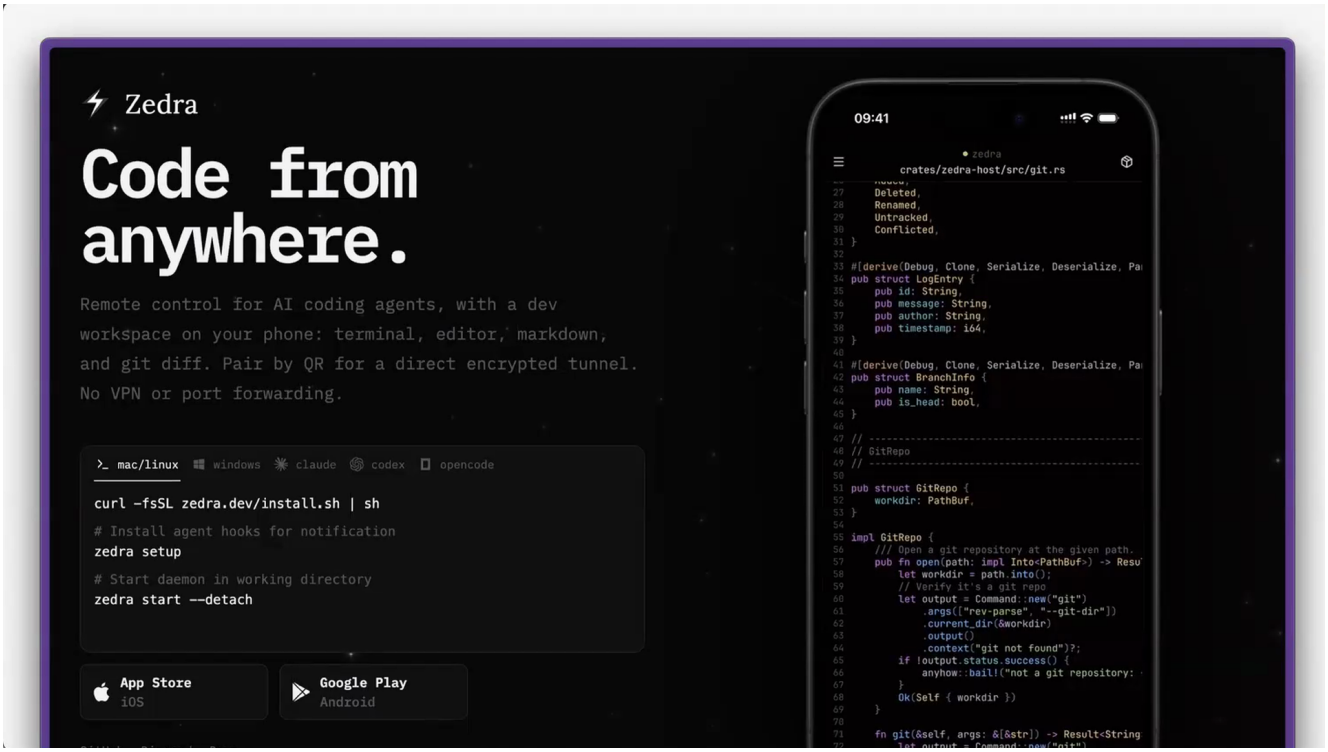
我们也被大量用在设备备份上。我们最早的用例之一就是：我怎么把加密的消息从一台设备搬到另一台？



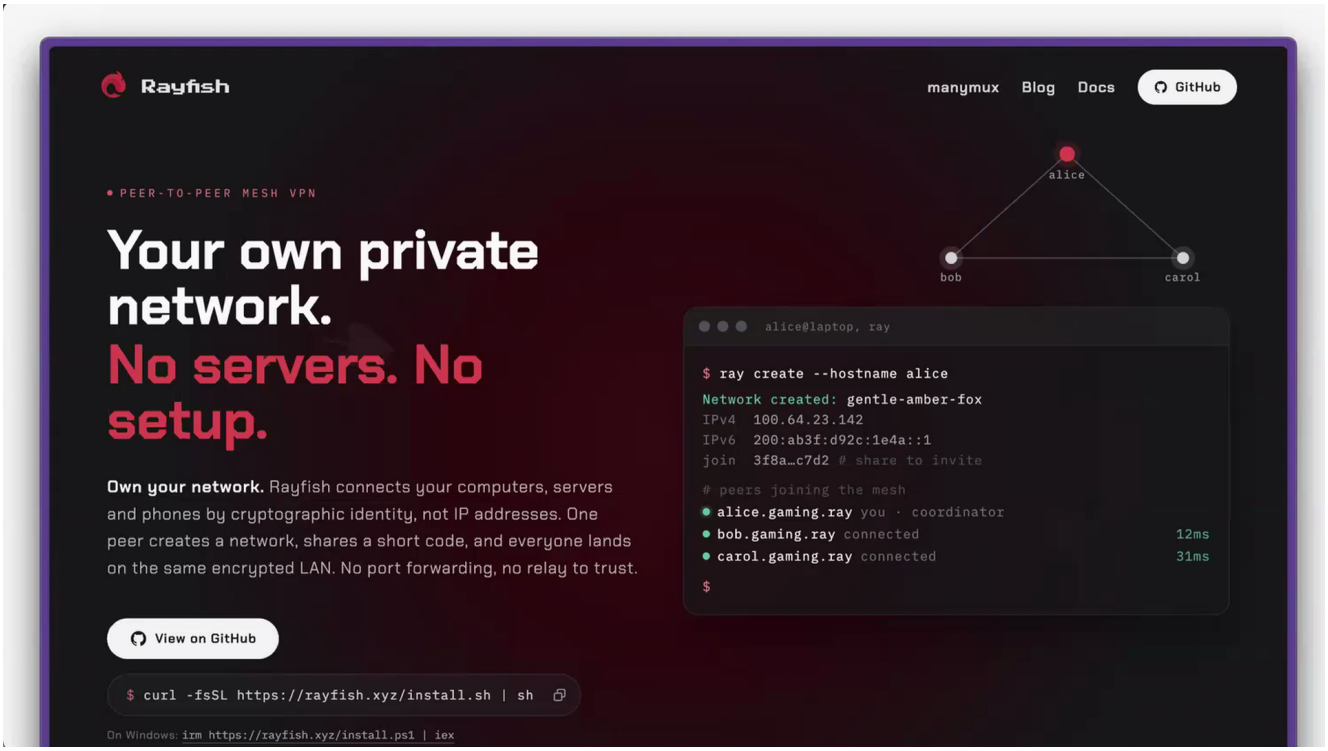
我刚买了一台新 iPad，我得把 iPhone 上的东西全搬过去。



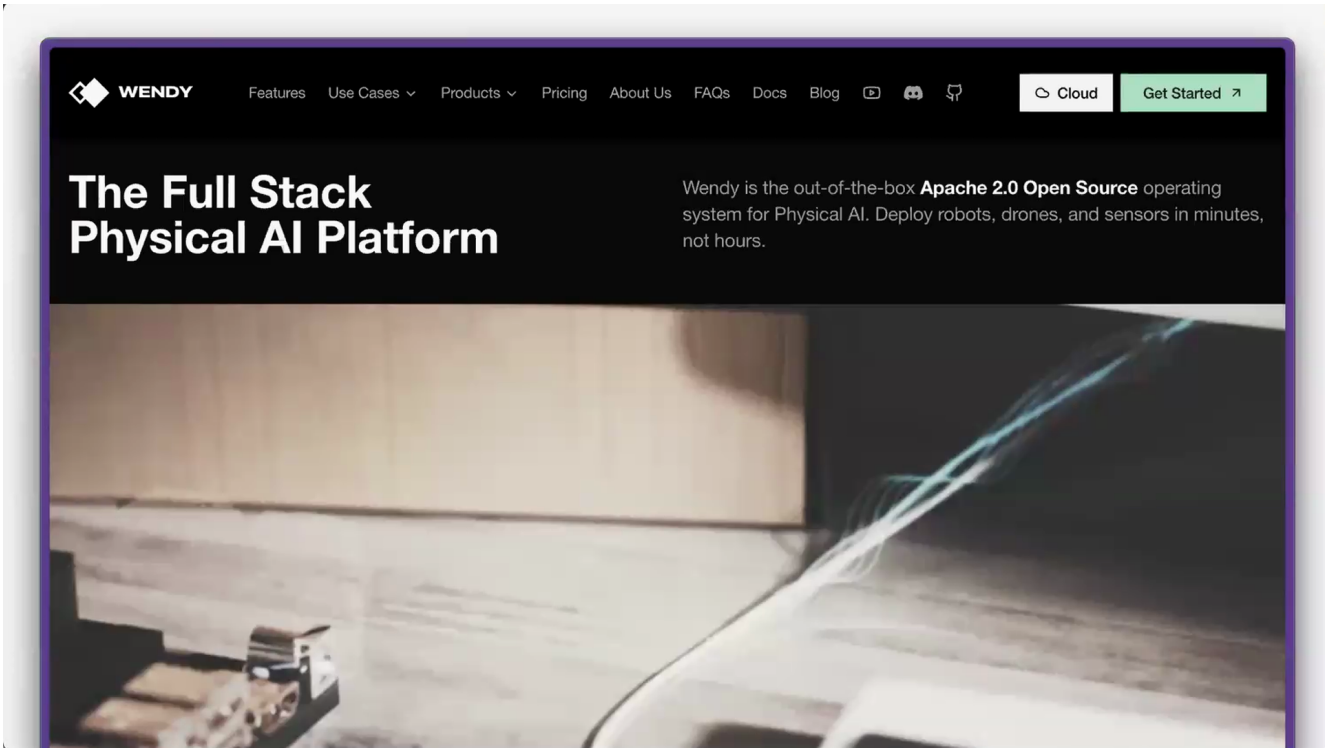
在这件事上，它是一种相当神奇的办法，能在两端之间拿到直接的加密连接。



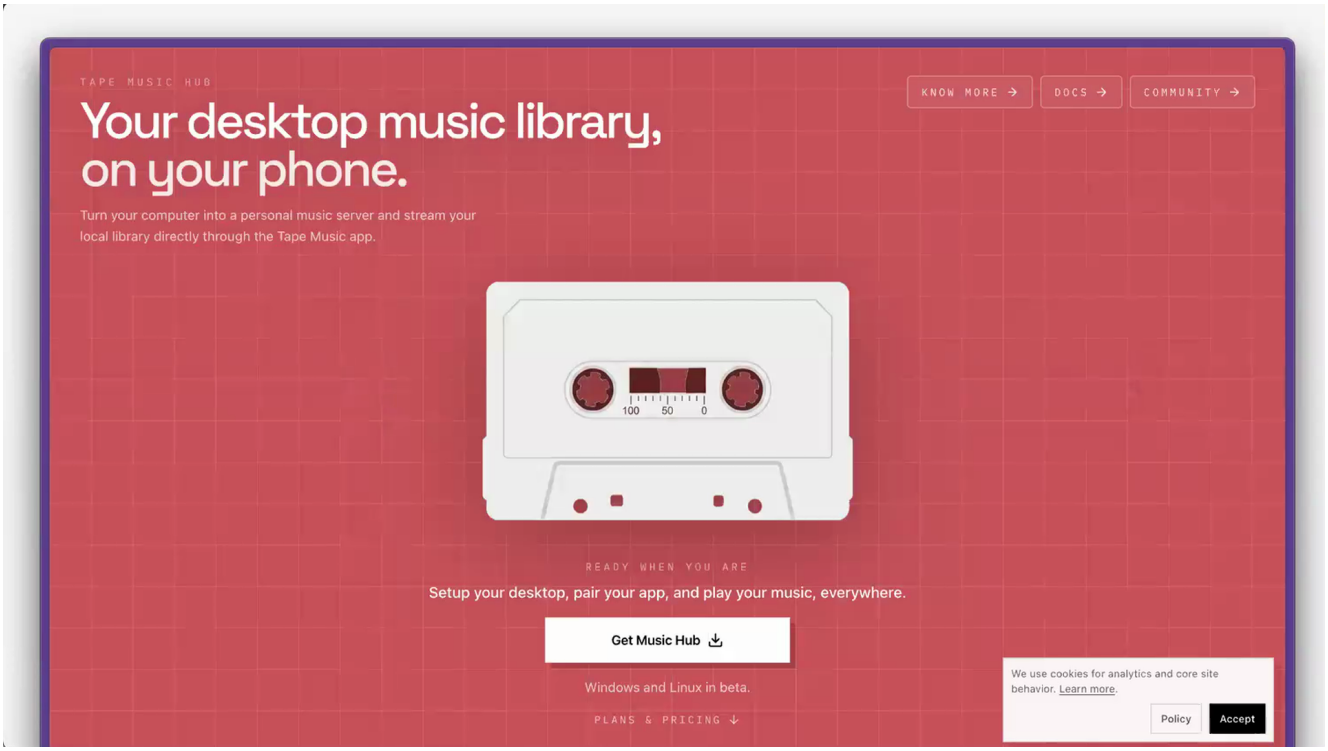
代码编辑器。这个方向最近真的起来了：我想在手机上编辑跑在我台式机上的代码——你有一台机器插在某个地方，你想能够和它对话。这是一个非常有力、增长很快的做法。



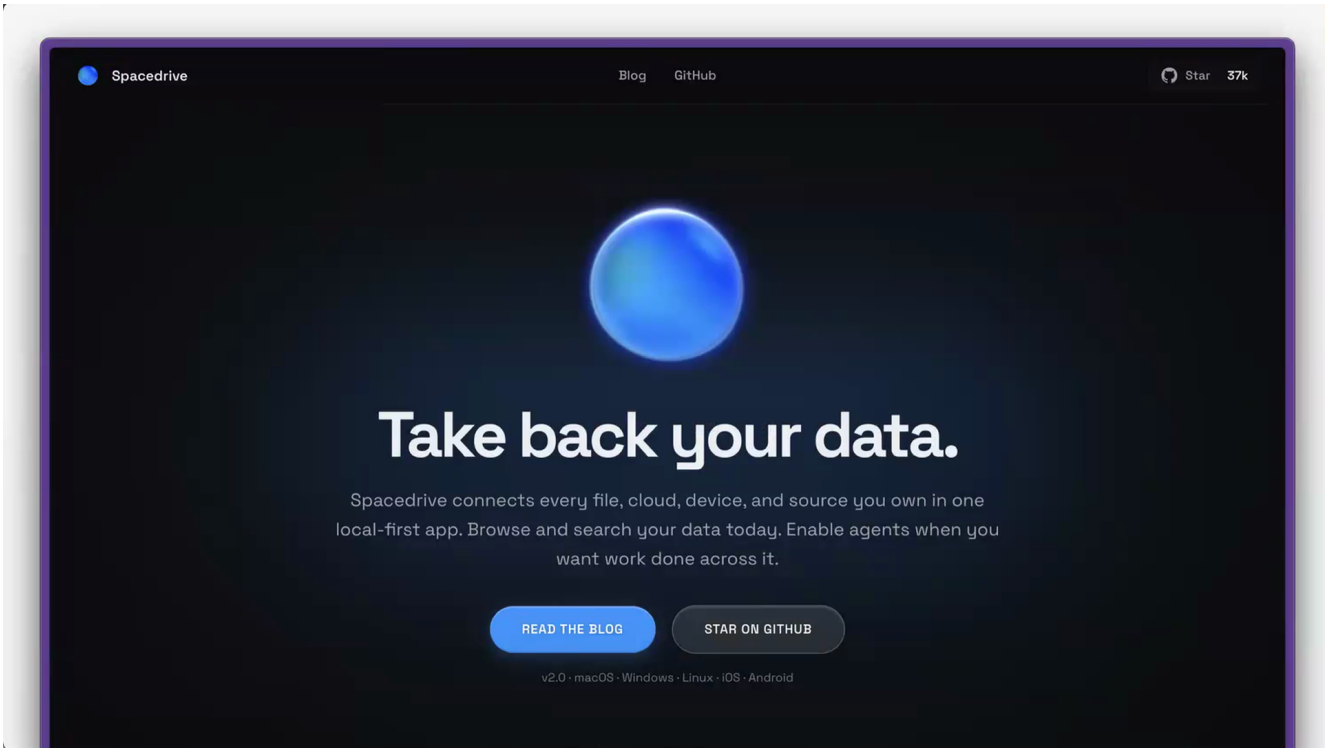
VPN。我们看到很多人拿它做 VPN：我想在一堆设备之间组一张 mesh 网络，我想把所有端口都转发过去，并且能长期管起来。



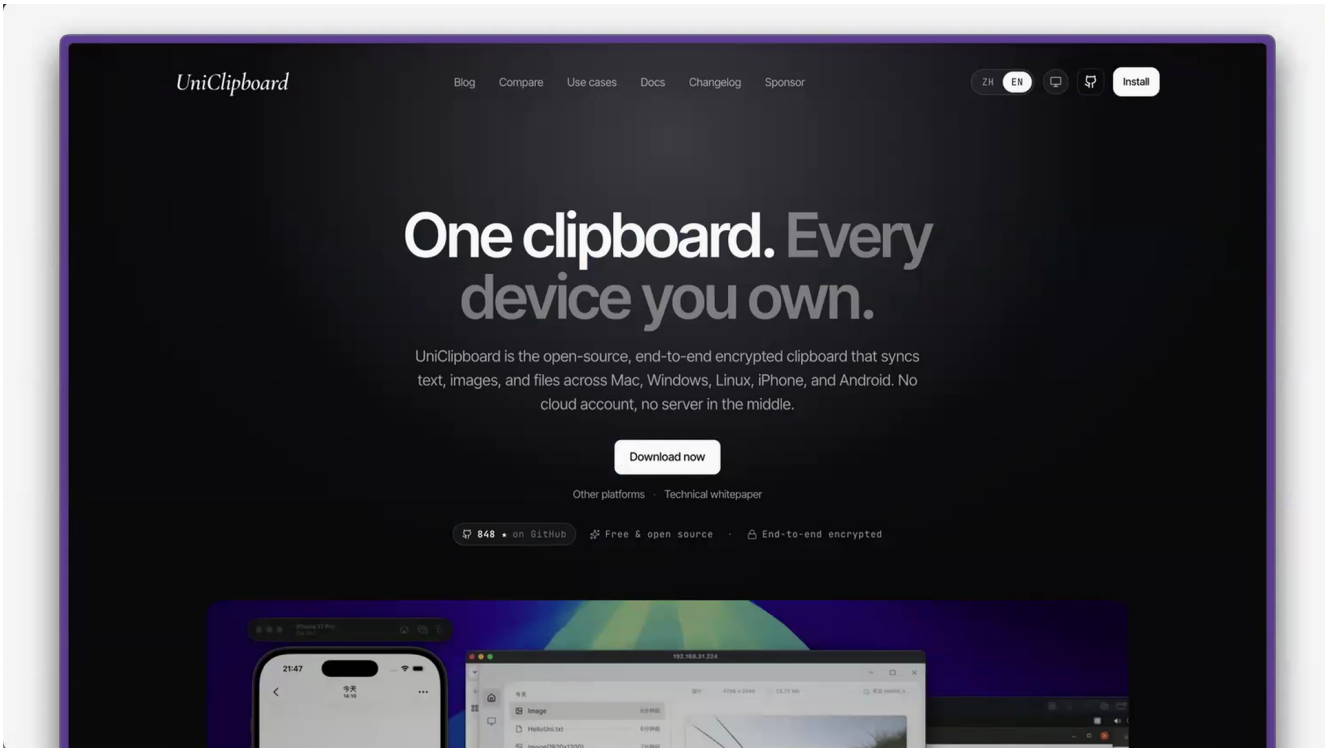
还有大量的物理计算（physical compute）——我们看到很多机器人在用 iroh，人们希望家里的设备能被自动化、能以安全加密的方式对话。



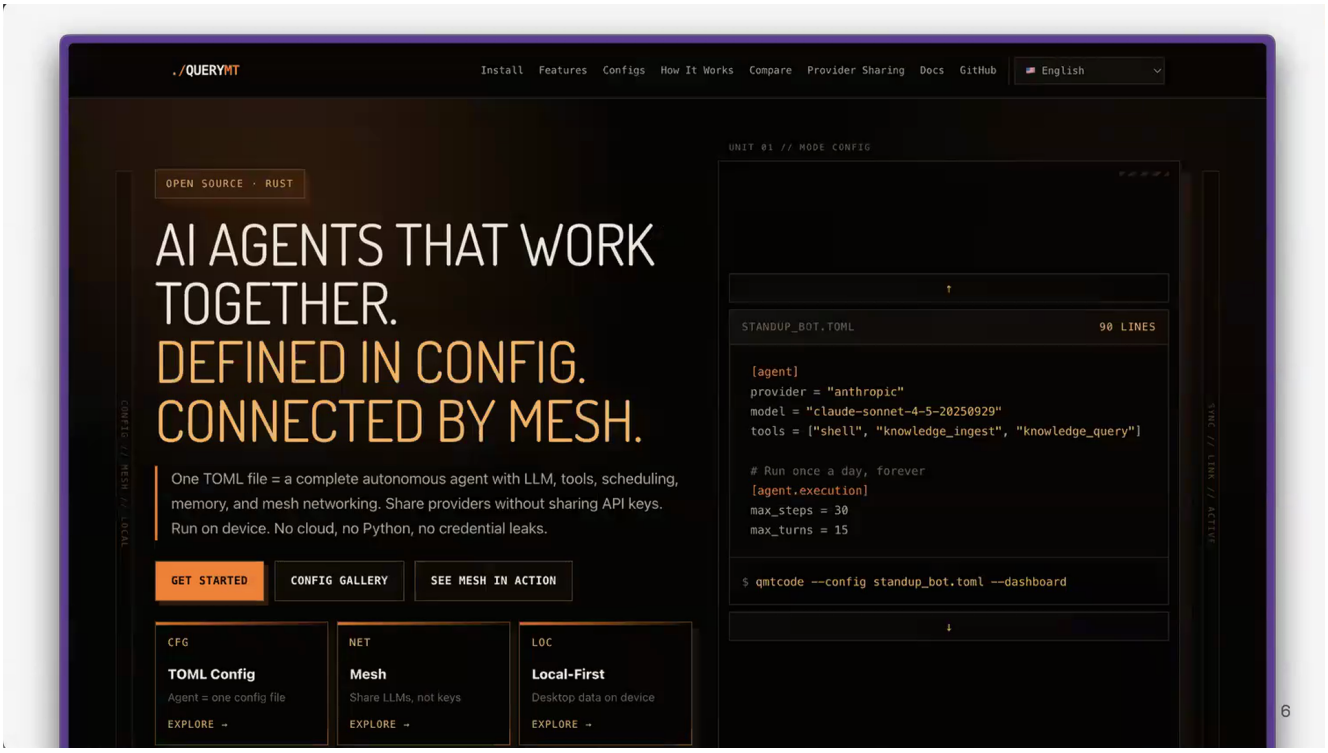
这个我个人特别喜欢：我的台式机上一直存着一大堆音乐，能把它取出来、在任何地方都能访问，这感觉很好。



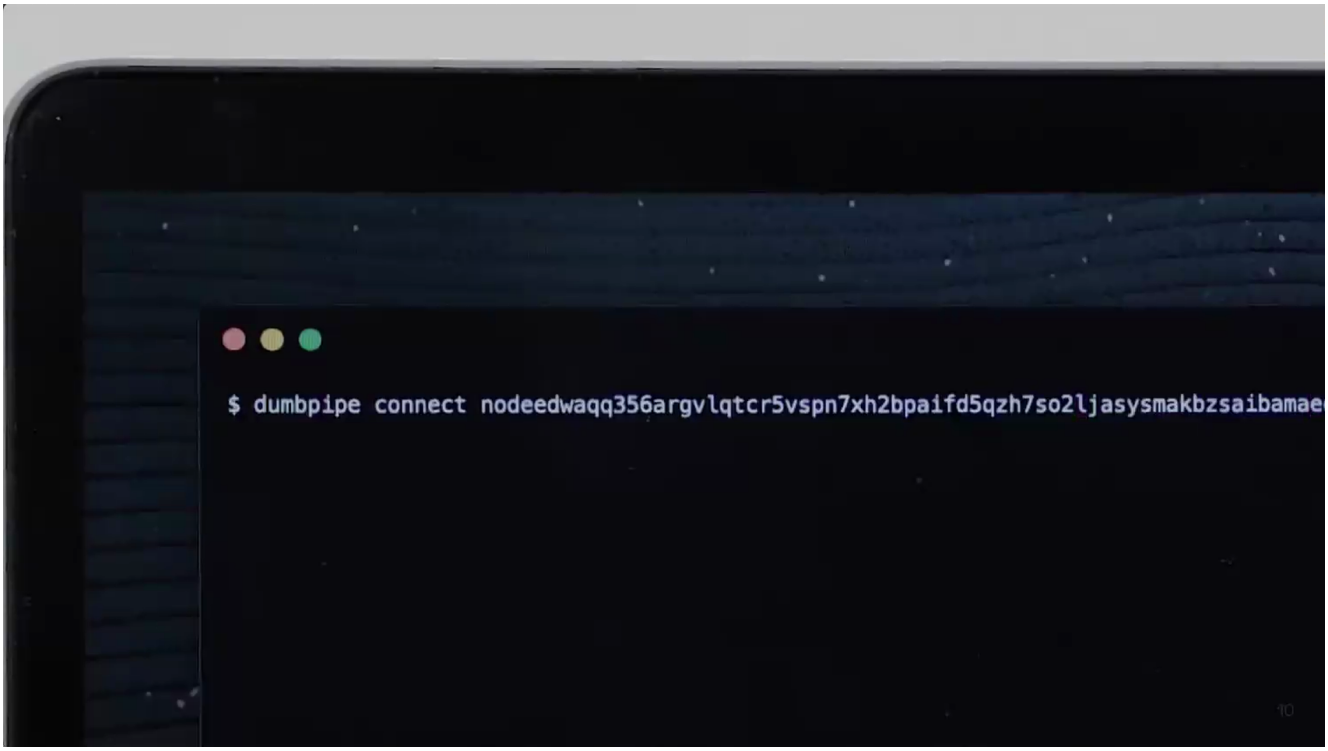
再往后，就有很多东西开始非常接近 local-first 宣言的那个取向了：我们要把数据的所有权收回来。这里我们就直接接进去了。如果你在原则上采取 local-first 的路线，重要的是你能够说：不，数据就是要诞生在我的设备上。而现在有了 iroh，我就有能力按我自己可控的方式把它联网，因为我们可以把全部联网能力作为库的一部分，直接发到设备本身上。这就是整个想法——我们认为你应该有能力推动这样的事情。



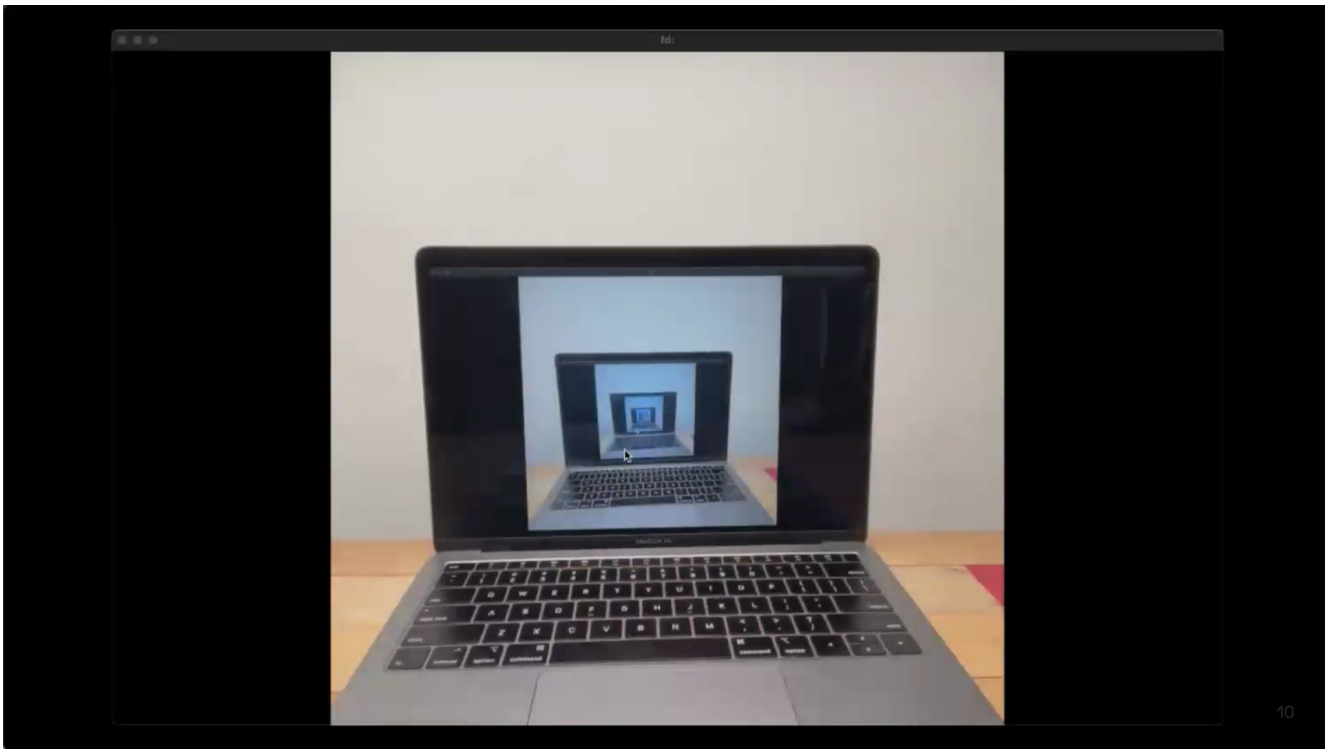
我们还有一些很棒的项目，比如跨平台的统一剪贴板。于是你终于可以在 Mac 上 Ctrl-C，在 Android 设备上粘贴，而且它真的能用。



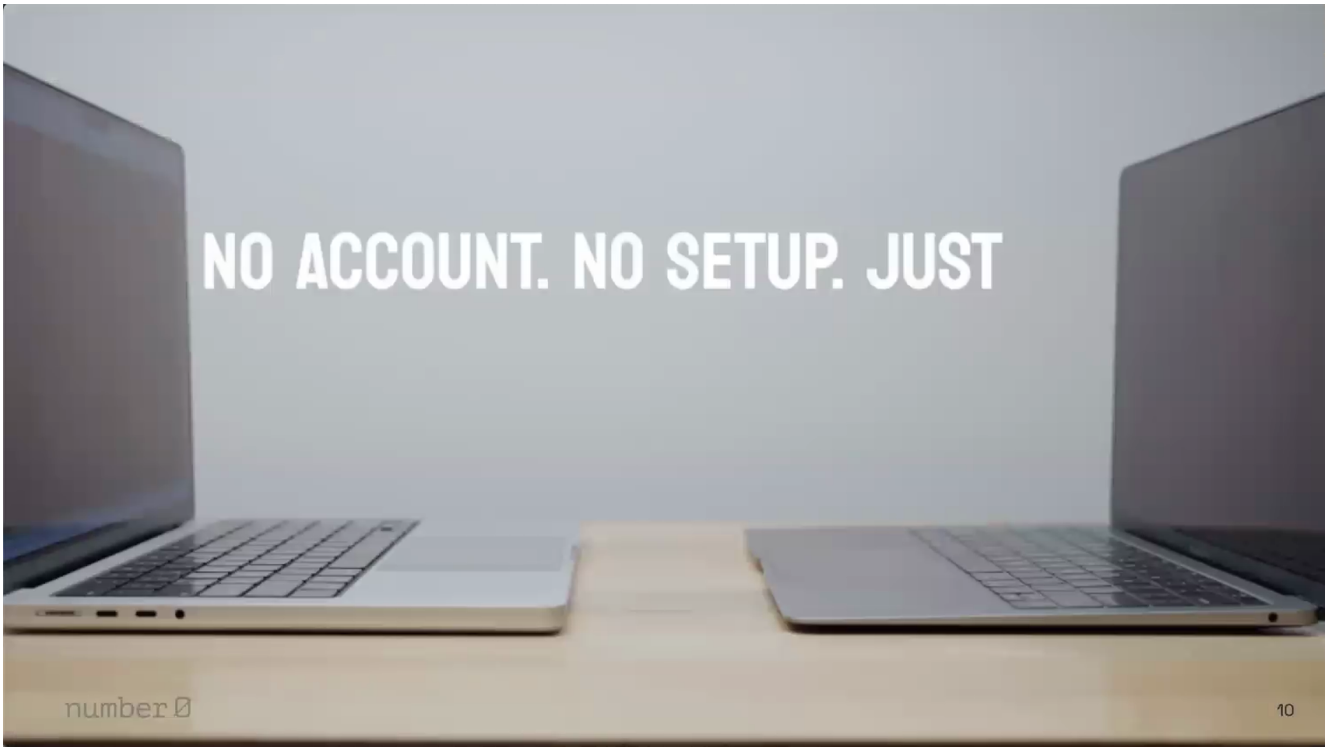
还有大量多智能体（multi-agent）配置和 agentic 编排的场景，你可以为 agent 做 mesh 组网，让它们互相调用。

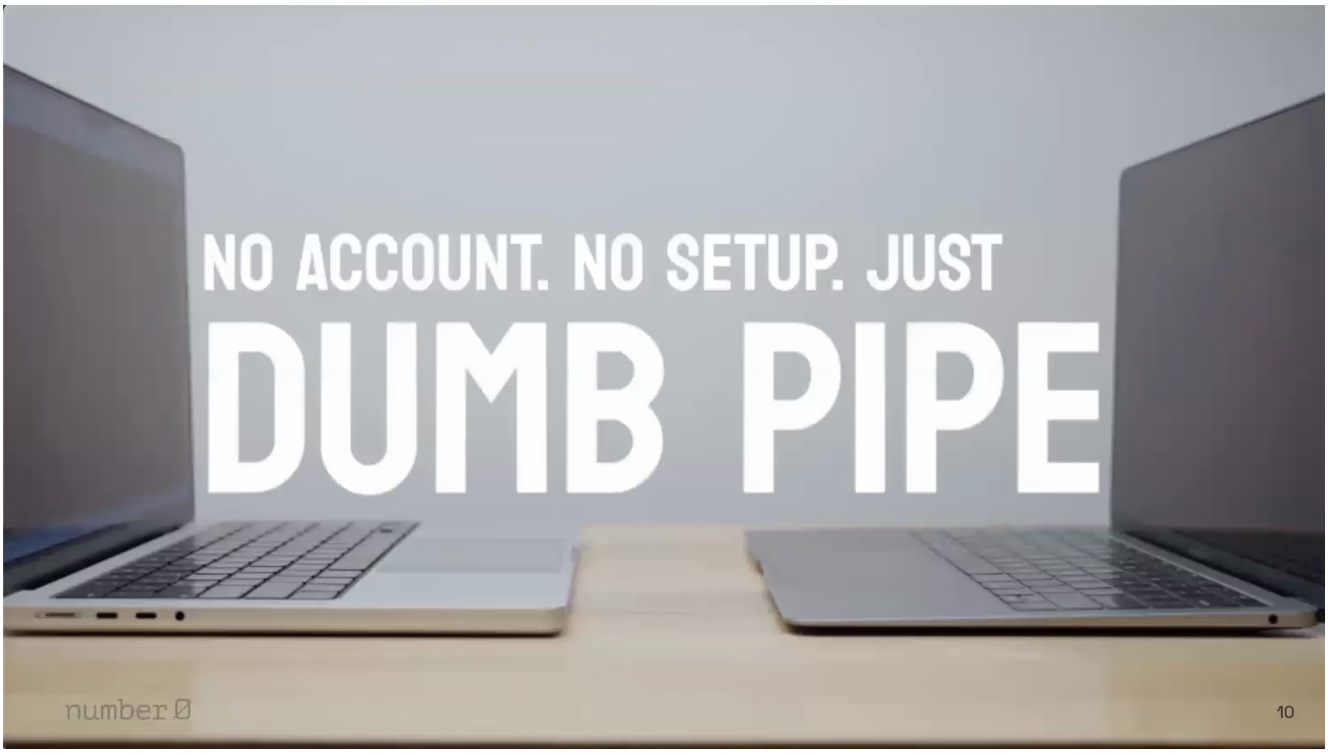


还有一些是我们自己做的，纯粹是为了演示这套东西。



Dumbpipe 我们做过不少：它字面意义上就是两台设备之间的一根 Unix 管道，我们把它接上 FFmpeg，于是直接就能做直播。



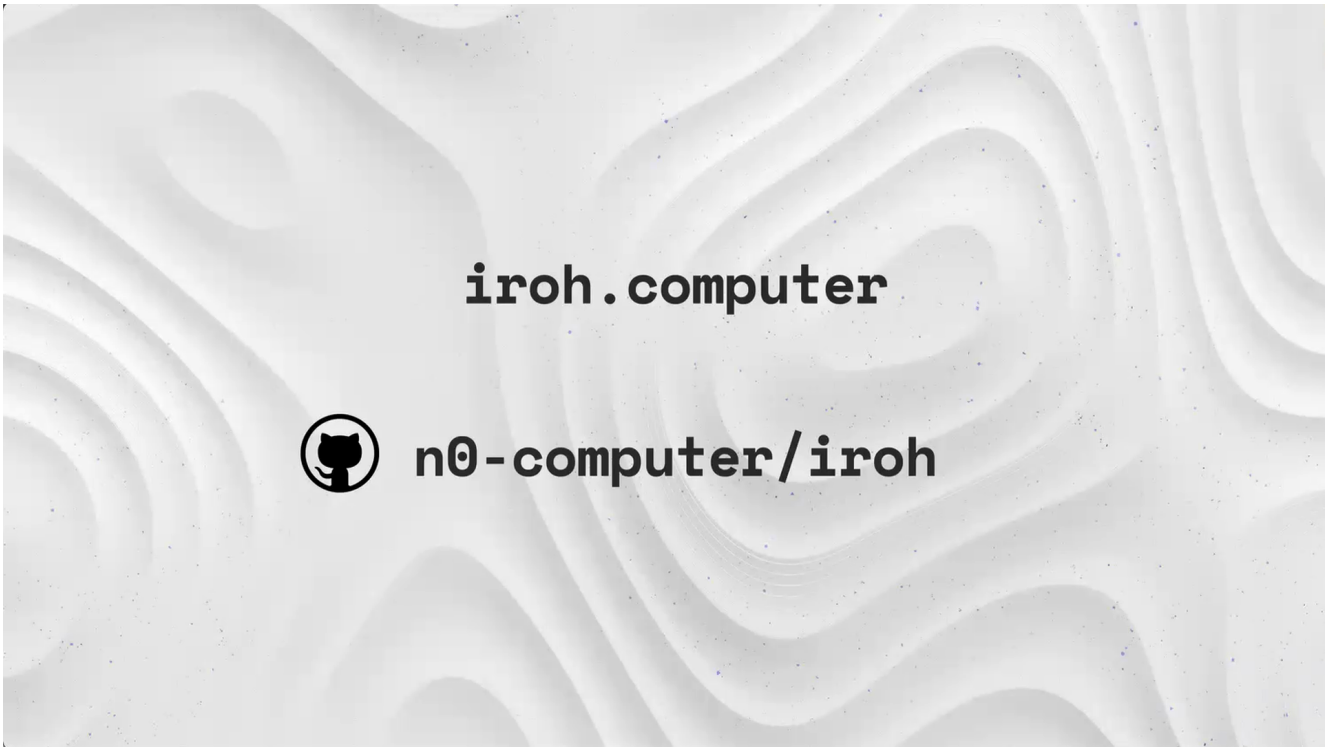


同样，不需要任何账号，接上就连通了。



这就是我们最想对大多数人说的那件事。在 local-first 的世界里，我们一直认为，是网络栈的短板在拖住整个社区：如果你手上没有一个真正可靠、能用、并且是为 local-first 场景而生的点对点网络库，我们就到不了那些更激动人心的地方。而我们认为，那些更激动人心的地方，正是你能做出一个真正兑现 local-first 理想的应用。所以我们认为，iroh 和 local-first 的想法是天作之合——你现在有了一个网络栈，它有望交付你做一个像样的、完整的 local-first 应用所真正需要的那些东西。

所以，希望你们会对自己的网络栈提出更高的要求。我们看到 iroh 在 local-first 领域被大量采用，我们非常兴奋——老实说，这是我们见到的最有意思的一批东西。在这个领域做事的很多人，真的理解推进技术前沿、从另一套原则出发建造的必要性。不要害怕 CRDT——很多人还需要逐渐建立起这样的雄心。而 iroh 这个项目，希望能配得上那份野心：你可以真的说，不行，我们确实需要一个能做到这些的网络栈，一个哪里都能用的、能自然而无缝地通信的、能替我们把用了哪种传输这件事抹平的网络栈，并且把设计协议、表达我们真正需要什么的完整控制权交还给我们。



大概就是这些。网站是 iroh.computer，仓库在 GitHub 上。有什么问题吗？

交流与问答

主持人 · 16:37

非常感谢，讲得太精彩了。大家如果有问题的话，现在就可以提出来。各位听众，现在可以提问。也可以把你的中文问题放在聊天区，或者我可以帮你把它念出来。

观众 · 17:05

我有个问题，我可以直接问吗？

主持人 · 17:08

可以，请。

观众 · 17:11

谢谢 Brendan 的精彩分享。我想问的是 iroh 的历史和定位。我大概两年前第一次看到它，当时以为它是 IPFS 节点的 Rust 实现。后来它慢慢长成了自己的东西，但底子是 libp2p。现在我感觉它的定位几乎像是 libp2p 的替代品，或者说一个更好的版本——我这么理解对吗？这个问题背后其实是：iroh 的愿景和定位是什么？它是不是想成为当年 libp2p 想成为的那个东西——基础设施层，是那个统一协议，架在传输层上无论跑的是什么协议之上？它想做的是这个吗？

Brendan · 18:05

对，我觉得没错。我们这帮人都是从 IPFS 出来的（*ex*-IPFS devs），我把链接贴到聊天区。像你说的，我们一开始——或者说我们这段旅程的起点——确实是一个 IPFS 的实现。如果你想听完整的长版本，我最近在那期 YouTube 视频里讲过，有兴趣可以去看。

但我觉得，是的，这个答案是对的，只是有一点要区分开。iroh 和 libp2p 最大的一个区别是，它想做一个高度以互联网标准驱动的项目。我们特别希望能到达这样一个位置：iroh 里 90%、95% 的东西都只是 IETF（Internet Engineering Task Force）的规范，而不是我们自己发明的新点子。我们非常努力地让自己像软件木匠一样思考，只做出真正能用的东西，因为我们想把 iroh 定位成一项互联网技术。我们希望有一天这东西的归宿就是：很多东西都跑在它上面，那就把它交给标准组织落定，让它成为我们所有人都能依赖的一块基础设施。把这件事建出来，是非常有意思的。

我想 libp2p 当年也有不少这样的野心，但它更把去中心化当作核心的驱动目的。iroh 是高度分布式的，只要可能，它也更偏好去中心化而不是中心化。但我们把目标换成了：做一项务实的、互联网优先的技术。任何会拖慢首字节时间（time to first byte）的改动，我们都不会往 iroh 里加。它必须能和常规 Web 技术竞争。为了做到这一点，我们愿意在意识形态上做妥协。我认为这是这个项目的一个核心差别——它仍然是开源的，仍然有大量共享的 DNA，但对我们来说，所有有可能建立的连接，100% 都得建得起来。如果设备是物理隔离（air-gapped）的，那我们确实无能为力。但你应该可以预期：任何时刻，只要能连到对面那台设备，iroh 就会替你连上，并且稳定地保持住。如果必须走中继，那就走中继——那并不算极度去中心化，那是一个分布式系统，是另一套保证。所以其实两者非常接近，这是在分毫之间较真。在“我们可以改进互联网、我们可以对网络栈提出更高要求”这个大愿景上，它们高度一致。

观众 • 20:31

我想大家看到 IPFS 被 Protocol Labs 放下、libp2p 也一样，心里都有点难过，但我觉得 iroh 是它最好的精神继承者。我一直很好奇：从 libp2p 身上能学到什么教训，它为什么没有成。我猜你刚才已经回答了大部分——是务实的那一面，有时候为了让系统真的能用，必须做妥协；还有你说的互联网规范，与其偏离 IETF，不如贴得更紧。你觉得这是我们能从 libp2p 学到的最重要的教训吗？

Brendan • 21:13

我还想再补一条。有两件事我想说。第一，没有 libp2p 和 IPFS，就没有 iroh，就这么简单。我认为是它们证明了这件事有需求、有人关心。我们就是从那个世界里出来的，我们热爱那个世界，看到 libp2p 的工作慢下来我们也难过——但我不认为它彻底结束了，还有很多项目建在它上面。

我们在 iroh 上真正刻意改掉的一件事是：我们非常坚定地认为，iroh 应该尽可能自我配置（self-configuring）。我们看到，用 libp2p 建东西时一个真正的难点，是你可能做的那张配置矩阵：你可以选加密层，用 Noise 或者 TLS；你可以选传输层、网络层，用 QUIC 或者 TCP。所有这些可选项，一开始看起来非常好，直到你意识到，某些组合如果拿错了，会变成安全上的噩梦。你可能会掉进这种坑：哦，其实如果你把 Noise 和 TCP socket 组合起来、再声明某个特定的东西，某些组合并不能给你你以为的那种保证；尤其是你还挑了一个和它并不相配的加密算法，突然之间，密码学家看了会说：别这么干。而你自己根本不会知道。

所以在 iroh 里，我们努力把这些容易误用的设计陷阱（footgun）拿掉，办法就是让 iroh 自我配置。所谓自我配置，意思是我们要求 iroh 这个软件自己去搞清楚：你的路由器是否支持 UPnP——这是一个很底层的技术，你本来不该需要关心它，但 iroh 知道它。iroh 应该知道你现在在一台 Android 设备上、你的网络状况变了，它应该对此作出反应，改变自身去匹配这些变化和差异。

这是一个根本性的转向。我认为 libp2p 的设计是让你对所有这些东西拥有完全的控制权；而 iroh 想告诉你的是：其实你要的只是非常稳健的端到端加密连接，你希望它在任何时候都活着，而且希望拿到当下最好的那一条。这件事我们替你做到。所以它确实少了一些旋钮——我们也越来越多地被要求给 iroh 加旋钮，让人能伸手进去调；随着项目演进，我想我们正是在那里把很多控制权拿回来。但这份控制是通过我们建的这套引擎给出的，它让 iroh 尽可能自我配置。当 iroh 跑在数据中心里，它知道自己处在数据中心这种上下文里——它并不真的"知道"，而是我们写了 if 语句去处理那个分支。这意味着某些东西会开始自动开开关关，我们让整件事跑得更好一点。

我们认为这件事非常关键：从性能角度、从安全角度，也从把心智带宽还给开发者的角度。如果你能不再操心这些，只要说：好，iroh 给我连接，它能用，我拨的是密钥——接下来我的工作就是想清楚我要什么协议、我要聊什么，我们认为这就把你的创新预算还给了你，让你专心去建你的应用。我们认为这非常重要，因为我们自己在 libp2p 上花过大量时间。所以我们真的很想确保，iroh 是高度自我配置的。不知道这样有没有帮到你。

观众 · 24:30

这个回答很棒，谢谢 Brendan。

Brendan · 24:33

不客气。

主持人 · 24:46

还有其他问题吗？大家可以提出。这里有一个问题——他问的是，有没有可能出现公钥冲突？

Brendan · 25:27

你是想问它们会不会碰撞（collide）？我理解得对吗？

主持人 · 25:31

是的，我想是的。

Brendan · 25:33

是的。密钥空间大到这种程度：碰撞的概率小到可以忽略。理论上有可能，但可能性低到我们日常根本不会为它操心。我们每天确实会产生数以十亿计的密钥，而我们用的爱德华曲线密钥（Edwards curve key，即 Ed25519）的密钥空间，非常非常非常大。它的设计前提就是：机器在紧密的循环里不停地跑，也永远撞不上。所以，是的，我认为你不需要担心碰撞，这是简短的回答。希望这能让你安心。我不担心这个，我从来没担心过。

观众 · 26:21

如果存在恶意攻击，用户是否可以指定自己想用的密钥去建立隧道？而当这种情况发生时，它会不会和原本的密钥冲突？

Brendan · 26:39

你问这个问题是对的。我想这个问题触及的是：iroh 的安全性到底从哪里来？不把密钥分享出去，就是整个安全模型的源头。iroh 启动时，默认——就是我给你们看的那些代码示例——它会凭空生成一把随机密钥，并且只放在内存里，不落到任何地方。这么做正是为了防住这一类设想。

如果你以某种方式拿到了别人的密钥，你就可以冒充他，把流量引向你自己。这是真实存在的。整个系统确实系于此，而这对所有密码学系统都成立：私钥一旦被分享出去，就等同于把账号交出去。所以值得记住：是的，如果你以某种方式拿到了别人的私钥，那是个大问题，因为在 iroh 眼里你基本上就是他。

日常我们有很多办法应对：密钥轮换（key rotation），以及只用临时密钥（ephemeral key）。临时密钥基本上就是不写下来的密钥——应用启动时生成一把，这次连接就用它，用完就扔。这是很常见的模式，在 iroh 里其实就是默认。如果你想要一把持久的密钥，你得主动把它存下来、重复使用。

但没错，这确实可能发生，所以值得确保你的代码在这件事上写对。不过话说回来，iroh 非常努力地不在这个过程里留下容易误用的设计陷阱。要凭空构造出一把你本来没有的密钥，非常困难；用别的手段去把密钥搞到手，反而快得多。而想猜出一把密钥的私钥那一侧，计算代价高到离谱——即使按我们今天对量子计算之类的理解，我们认为距离这成为一个大问题，至少还有几年。

希望这回答了你的问题。这是个好问题，因为去想安全这件事本身就是对的。而它最后归结为一句话：把私钥保管好。别把它挪出设备，就放在那儿。只要你不挪动它、不让它跑出去，就没问题。SSH 就是这么建的，很多东西都是这么建的。这是个非常简单的原则，但把它做对很重要。

Brendan · 29:06

好。还有别人要问吗？今天特别开心。

主持人 · 29:16

还有其他问题吗？如果不方便开麦的话，也可以放在聊天区。

Brendan · 29:49

好的，那今天大概就到这里。如果之后还有问题，最好的去处是 iroh.computer/discord，我们都在那儿，随时聊。另外，我们很想看看你们做出了什么。如果你做了什么东西，就到 Discord 里发出来；我们还有一个 awesome 列表，awesome-iroh。很期待看到你们做的东西。

主持人 · 30:22

非常感谢，非常感谢。

Brendan · 30:26

也非常感谢你们邀请我，真的很感激。谢谢各位的时间，希望今天讲的对你们有用。期待在互联网上再见到大家。

观众 · 30:35

谢谢 Brendan，非常棒。

Brendan・30:38

多谢。我先下线了，拜拜。

主持人・30:41

祝你今天余下的时间愉快，拜拜。