Scuttlebutt 的开发历程

Secure Scuttlebutt：一个本地优先分布式社交网络

第一次本地优先会议 · 2026 年 9 月 23 日
分享者：Jiang / Social Layer

演讲



Secure Scuttlebutt：一个本地优先分布式社交网络

那我开始今天晚上的分享。大家好，我是 Jiang，Social Layer 的创始人。我也是一个比较资深的程序员，有大概 15 年专业的软件工程开发经验，基本上从初中就开始学编程。在整个职业生涯中，我一直在探索一个问题：怎样构建一个去中心化、分布式的系统和网络。

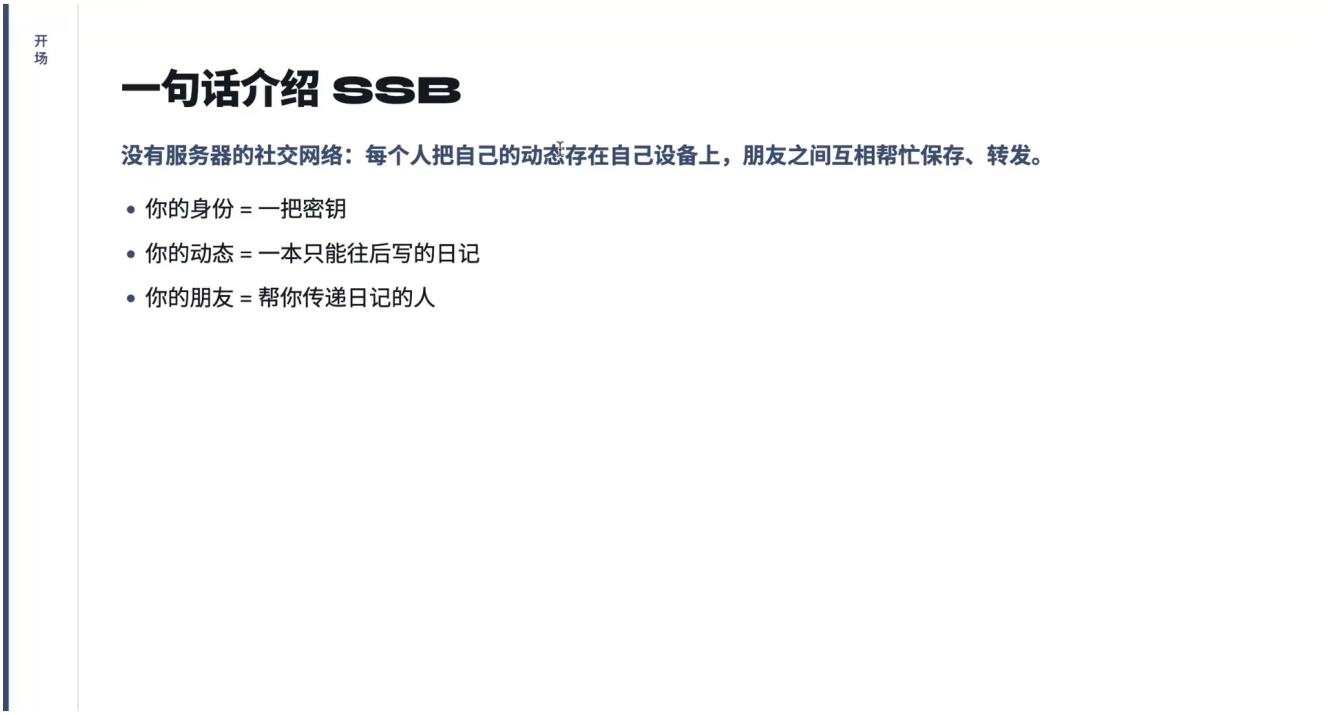
今天我给大家介绍一个叫 Secure Scuttlebutt，有时也称作 Scuttlebutt，简称 SSB 的系统。它是一个本地优先、去中心化的分布式社交网络。今天会讲它的原理、扩展，以及一些非常独特的设计哲学，和它给去中心化社交留下的宝贵经验。



今天聊什么

我们会介绍 SSB 到底是什么、从哪里来、怎样运作，怎样在没有服务器的情况下传播消息，以及它怎么解决去中心化环境中消息传播的性能和通信带宽问题。还有它跟其他协议的比较。

我这边可能会有一些背景音，因为我在咖啡馆里。如果不方便，我可以看看能不能挪到安静的位置；没有问题的话，我就继续。



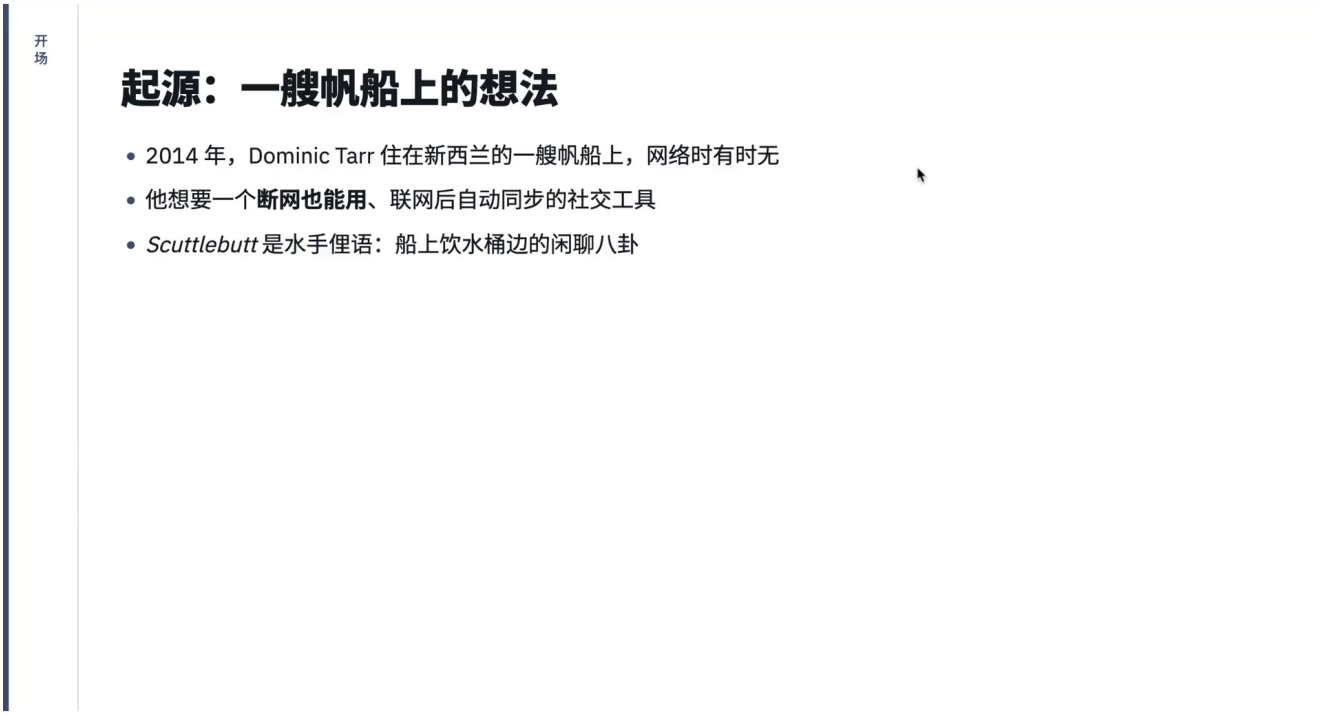
一句话介绍 SSB

SSB 诞生于 2014 年，也是在比特币之后。比特币是 2008 年提出来的，2009 年、2010 年差不多就开发出来了，那时吸引了一些密码朋克和做去中心化网络应用的人。当时还有一个 JavaScript 社区，里面有很多非常优秀的开发者，我也加入了。

我在其中认识了一位来自新西兰的技术极客。他是一个非常厉害的人，我跟他交流，学到了很多东西。后来，2014 年左右，他做了这个叫 Scuttlebutt 的社交网络。

这个社交网络不依赖我们在互联网上常用的一些基础设施，包括域名、IP 地址。每个用户有一对公私钥，同时有自己的动态，或者说日志。每次发消息，都用自己的私钥身份签名，发到这个日志里。

你的朋友会订阅这个日志，帮你保存和传播。这里没有一个中心化的服务器，每个人有自己的软件节点，日志保存在自己的节点上，朋友之间互相保存和转发。



起源：一艘帆船上的想法

Dominic 是一个非常向往自由的人。他是新西兰人，住在奥克兰的一艘帆船上，网络有时有，有时没有。他希望做一个断网时也能用、联网后自动同步的社交工具。

Scuttlebutt 是水手的一个俗语，本意是饮水桶。在水手之间，它还有闲聊、八卦的意思，因为水手们会在饮水桶那里，就像在茶水间一样交流，聊些有的没的。



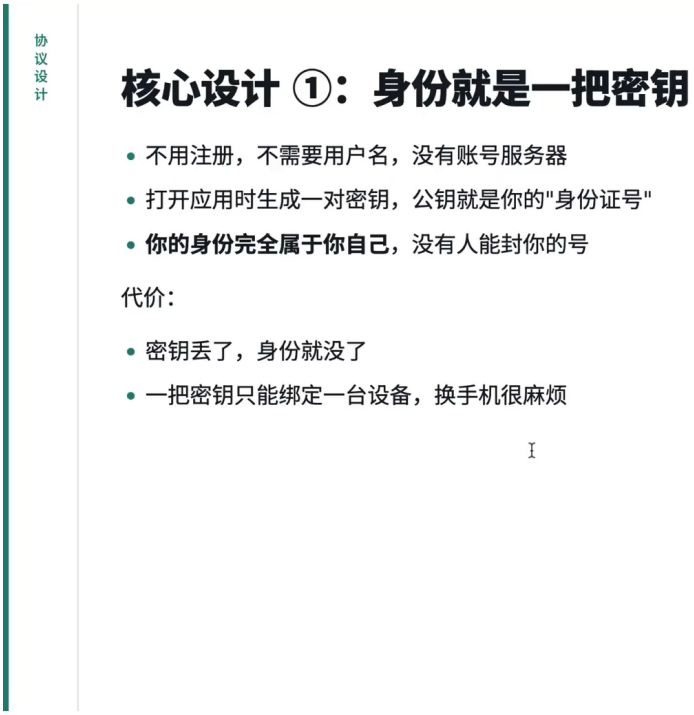
十年历程

后来有人做了 Scuttlebutt 的客户端 Patchwork，非常好用。社区逐渐壮大到数万人，那时有人做过网络节点的账号统计。后来也有一些应用，比如 Manyverse 和 Planetary。

做密码朋克、分布式网络、去中心化应用的这些人，也得到了区块链社区的关注。当时有一个项目叫 DFINITY，ICP 的创始人给了他们 20 万美元的资助。Dominic 没有自己拿着这笔钱，而是把它分成每份 5,000 美元、总共 40 份的 grant。

他在社区里说，谁有兴趣做东西，就写一个提案，我看到了会确认。然后你去实现这个功能，把整个开发过程发在社交网络的公开日志里，让大家监督，最后我就给你这笔钱。于是他们就有了这样的 microgrant。

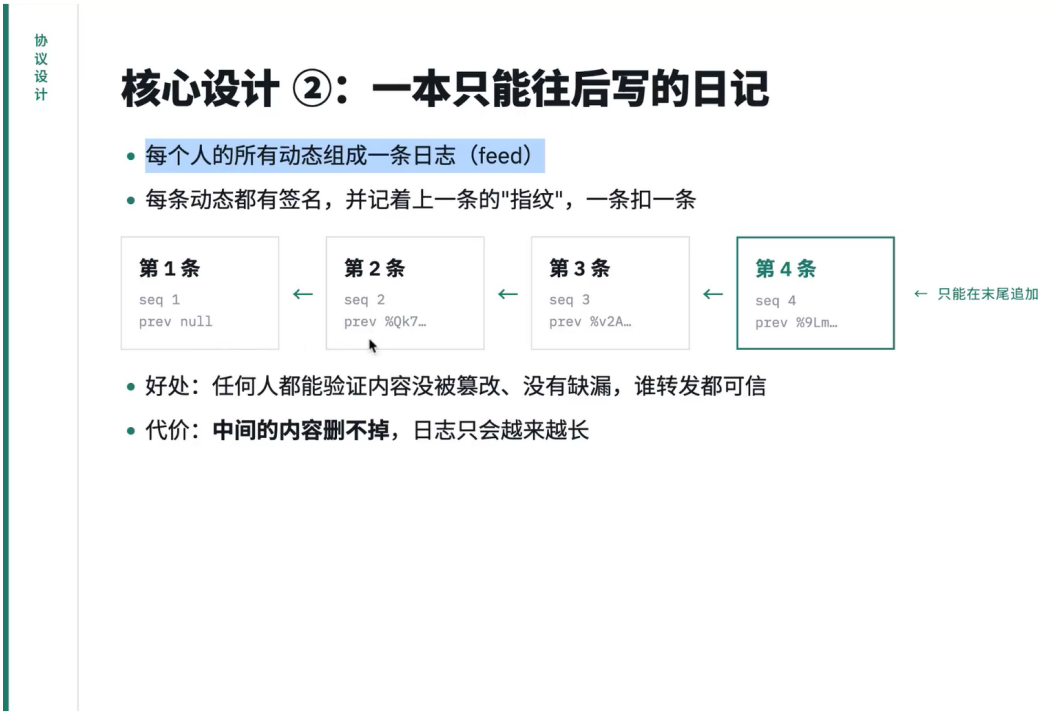
后来，受到 Scuttlebutt 的影响，也产生了 Nostr。你看 Nostr 协议的 GitHub，它在协议比较里也写了 Scuttlebutt。Nostr 有一段时间得到了比特币社区的支持，其中一个 Scuttlebutt 应用也集成了 Nostr。另外有一个手机应用叫 Manyverse，也使用 SSB。目前还是有一些小众爱好者在用。



核心设计①：身份就是一把密钥

每个用户的核心身份就是一把密钥，不需要注册，也不需要用户名。只要有私钥，就可以给你发的消息签名。你的身份完全属于自己，没有人能够屏蔽你的账号。

但是，密钥丢了，身份就没了。而且一个密钥只能绑定一个设备。



核心设计②：一本只能往后写的日记

每个人的所有动态会组成一个信息流，也就是一个 feed。每条记录都有签名、序列号，还有上一条信息的哈希。你可以认为它跟区块链很像，没有人能够篡改内容，历史也没法篡改。所有信息会在历史积累中一起传播。

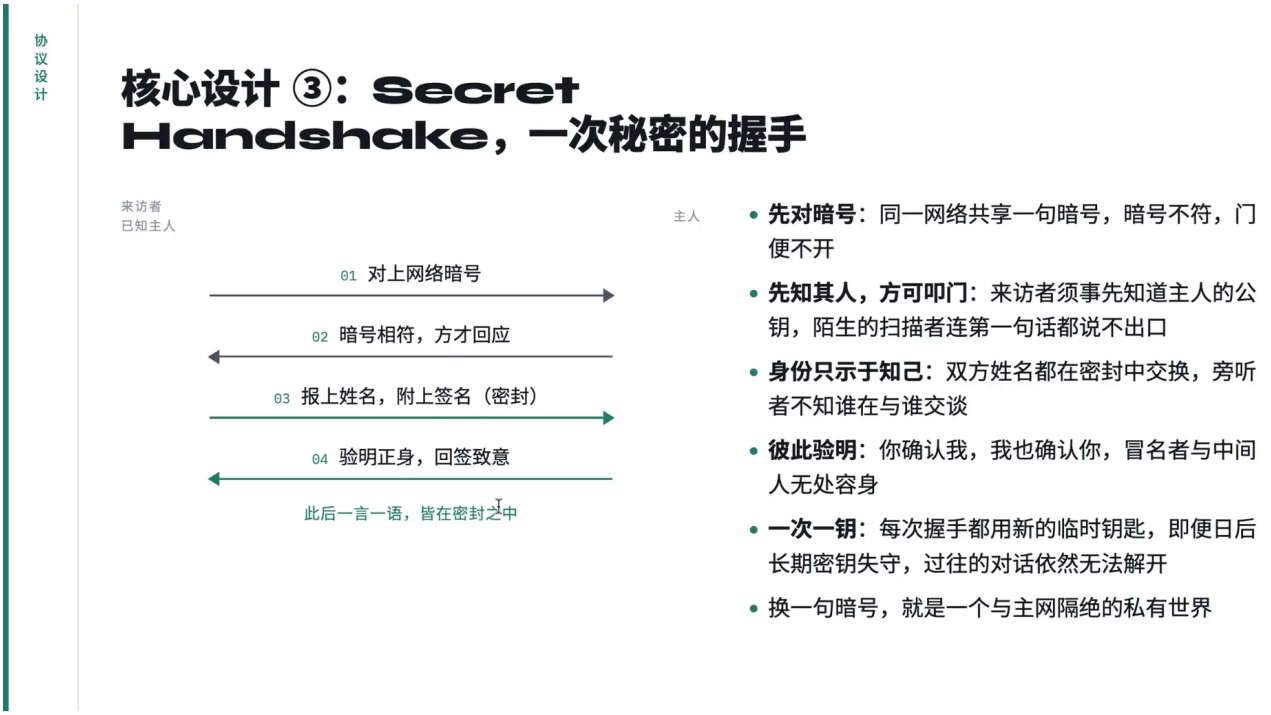
有了签名，你就可以从第三人或者中间方获取其他人的信息，而不需要担心；只要验证签名正确就可以。



SSB 与区块链

SSB 跟区块链挺像，只是它像无数条非常轻量的区块链。区块链的单位是区块，区块里有很多交易，整个共识系统只有一条链，所有人都在这条链上写入。网络的价值来自全网对最新区块的共识，全世界只有一条链，比如比特币或者以太坊。

SSB 每个“区块”只有一条消息，每条链只属于一个作者。它的信任不来自全网共识，只来自签名。每个人有一条链，链与链之间可以互相引用，通过关注、回复交织成一个网络。它很轻盈，无需共识，无需挖矿，只需要为自己的链进行证明。



核心设计③：Secret Handshake，一次秘密的握手

同时，它设计了自己的加密传输协议，叫 Secret Handshake。首先要对上一个暗号，暗号不同，门就不开。发送者要先亮明自己的身份，对方才决定要不要回复。握手过程中会交换密钥，用来加密、解密后续整个通信的信息流。

来访者亮明身份后，被连接的节点决定是否回复。如果决定回复，就用 DH 算法把自己的身份加密后返回，双方互相验明身份，中间人无法冒充。每次也都会用新的临时密钥来加密数据。



Pull, not push

这里有一个非常独特的设计哲学，贯穿 Dominic 和 SSB 的很多部分：信息是被拉取的，而不是被推送的，pull, not push。被叩门、被连接的那一方，可以选择是否回应、是否亮明身份。

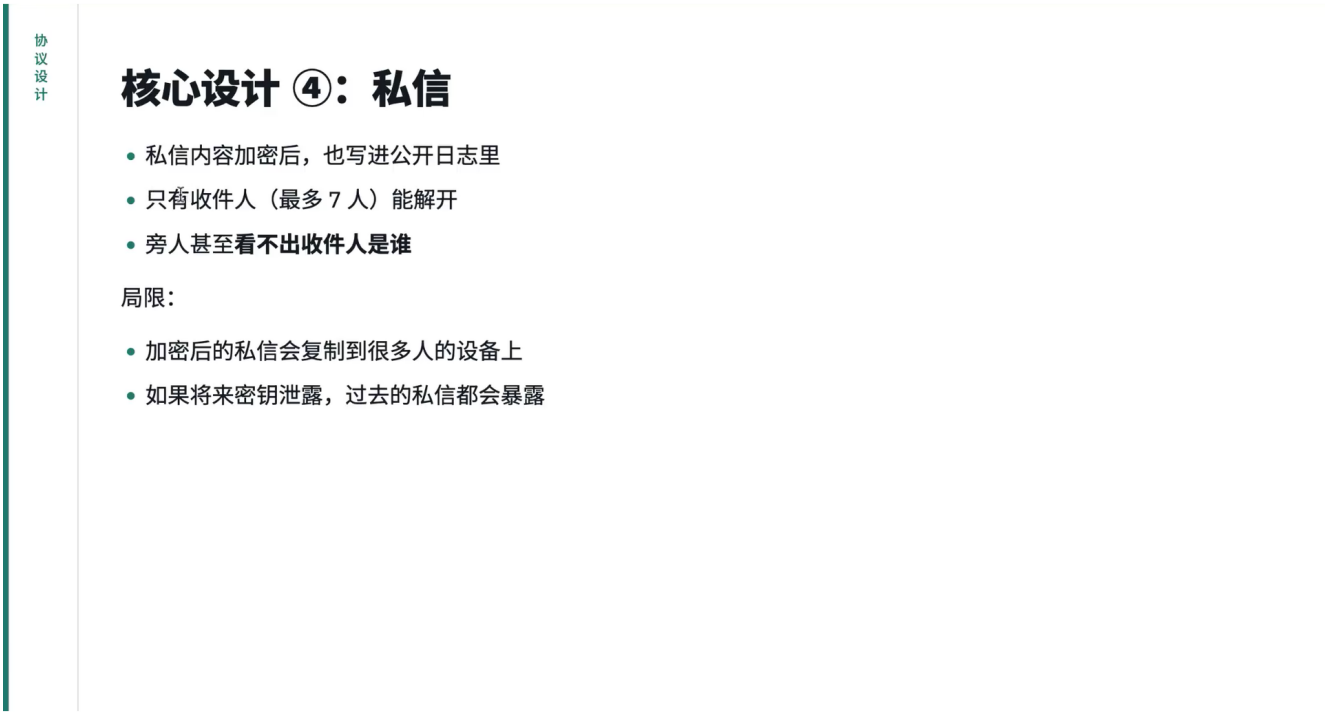
这里有一个很深层的隐喻，如果不跟 Dominic 聊，其实聊不出来。为什么不用 TLS？第一，TLS 太复杂，而且依赖中心化的证书提供方，也就是 PKI。通常你要先知道对方的域名、证书，再去连接。

像我们使用的 HTTP，TLS 是为网站设计的。网站是服务方，敞开大门，希望所有人来连接，然后通过身份认证系统决定你有没有权访问资源。它假设所有人都知道你连的是什么网站。面对陌生人，在 TLS 这一层会回应，再到权限层面决定要不要限流或屏蔽。

SSB 通过 *Secret Handshake*，只为熟人开门。连接者要自报身份，它的假设是只有知道我的人才能找到我。如果是陌生请求，它就不应答，或者回复别的内容，让你以为它是另外一种协议。

像 BT 网络，其实可以通过 DHT 做全网扫描，找到正在下载信息的节点。别人就可能攻击你，或者做流量检测，看你有没有下载版权电影。P2P 网络实际上是一片黑暗森林：如果不是熟悉的声音，就不要回应，这样就不会被扫描。这是很重要的一个特征。

在 SSB 里，数据复制和同步也是如此。订阅意味着一个拉取请求。因为我先订阅你、先作出决定，才给你未来向我推送的权限；而不是把端口敞开，你高兴、想推送什么就推送什么。



核心设计④：私信

刚才说的消息日志是公开的，但网络里也可以有私信。私信也放在 *message* 的消息格式里，只是被加密了。它用每个收件人的公钥来加密解密密钥，所以只有相应收件人才能解开、看到内容。

其他人看到这个信息流，可以帮助传播，但看不出收件人是谁。



核心设计⑤：图片和文件

SSB 的使用效果很像推特。除了公开消息和私有消息，还有图片、文件。这些不直接放进日志，而是通过单独的哈希来引用，也可以在网络里传播。

需要时才下载，不需要的时候，可以清理这些图片和文件的 blob。



设备之间怎么找到彼此

无论手机还是电脑，每个人都运行一个 SSB 节点。如果你跟朋友在同一个 Wi-Fi 下，它用 mDNS 这样的局部广播协议，让节点互相发现。同一台电脑或同一个内网里的节点，可以发现对方并同步。

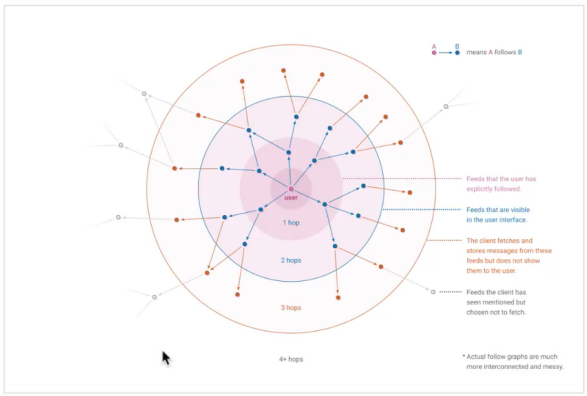
如果不在同一个局域网，可以通过 Pub 机制。Pub 是一直在线的公共节点。Dominic 不希望所有东西都能被所有人连接，所以加入公共节点需要邀请码。加入之后，就能同步其他加入这个节点的账号的信息。

Pub 其实也是一个普通节点，只是一直在线，有公共地址。后来还设计了 Room，也就是房间机制。你可以把 Pub 理解成酒吧，大家进来闲聊，更高效地传播信息。

Room 也保持公开可访问，但只帮助双方连通，自己不存信息，数据是端到端加密的。即使是 Room 服务器，也看不到具体聊天内容。就算没有这些网络，通过 U 盘也能复制、拷贝和同步 SSB 的数据。

同步与传播

同步什么？由你的朋友圈决定



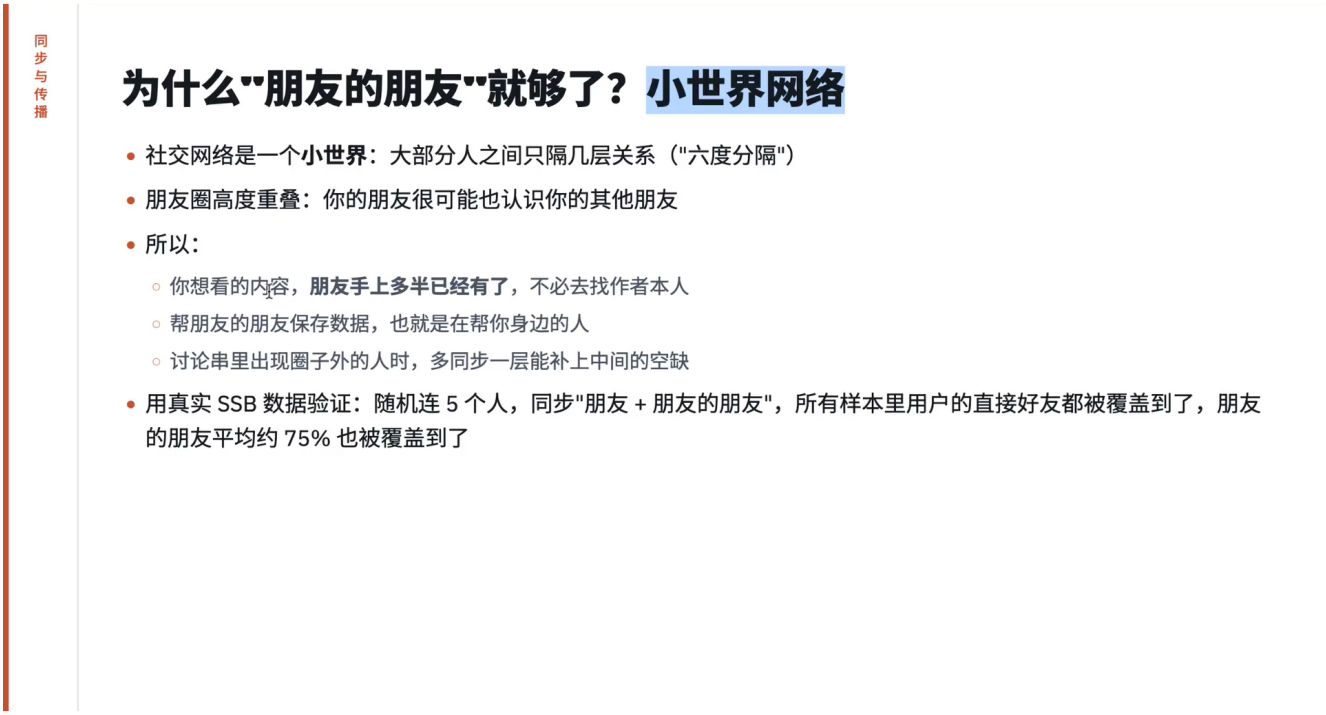
图片来源：Scuttlebutt Protocol Guide

- 关注别人时，会在自己日志里公开写一条"我关注了某某"
- 所有人的关注关系拼起来，就是一张社交关系图
- 以你为中心，按距离一圈圈往外：
  - **1 跳**：你直接关注的人
  - **2 跳**：朋友的朋友，默认在界面上显示
  - **3 跳**：同步下来帮忙保存，但不显示
  - 更远的：知道存在，但不下载

同步什么？由你的朋友图决定

回到 SSB 怎样传播信息。在社交网络中，每个人关注一些用户，也被一些人关注。以你为中心，有一圈直接的紧密朋友，再有朋友的朋友，外圈还有朋友的朋友的朋友。

连接意味着 A follow B。你关注一些人，也有人关注你，可能单向关注，也可能双向关注。这里有你直接关注的人，也有朋友的朋友。以前有一个六度人脉理论，就是通过六层好友关系，可以认识世界上所有的人。



为什么“朋友的朋友”就够了？小世界网络

SSB 用了一种 gossip 算法。两个节点互相连接、交换信息，也交换它们知道的其他人的信息。

假设一个账号关注了 100 个账号，网络中不需要所有账号都直接连接。只需要连接若干账号，通过随机传播，就像社会中的谣言传播，一传十、十传百，让信息被关注它的节点看到。

SSB 很有趣的一点是，它为社交网络而设计，利用了小世界网络，small world。大部分你真正关心的人，只隔一两层关系。你的朋友很可能认识你的其他朋友，朋友圈高度重叠。

有 100 个朋友时，不需要给 100 个人都打电话，只要跟十个、七八个朋友打电话，可能就知道这 100 个人的八卦了。你的朋友有他们的朋友，彼此传播很多信息。

你的节点除了自己的信息流，还会保存朋友、两度朋友圈的数据。即使出现圈子外的人，通常你的朋友的朋友也可能实际关注到。

SSB 做过数据验证：一个节点随机连接五个节点，同步朋友加朋友的朋友，基本上样本里的直接好友都被覆盖到，朋友的朋友平均也有约 75% 被覆盖到。



Network of friends

这解决了传统 BT 网络一个非常重要的问题：谁来保存数据。SSB 是朋友的网络，network of friends。

对比 BT 网络，它的问题是谁来做种。文件是公共资源，假设你下载一部电影，下完的那一刻就没有理由继续分享，看完也可以删除。

SSB 里，自己的信息会长期保存，同时你可能关注 100、200 个朋友，节点也保存他们的信息。因为在同步别人的日志时，你得把日志复制过来。你保存的是自己关注的人的信息，跟他们有关注关系或者朋友关系，所以愿意留存他们的数据，也愿意分享给其他人。

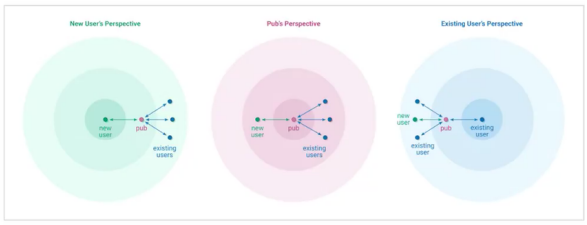
一个人的数据，就很自然地分布在所有关心他的人身边。大家有没有感受到这一点？SSB 首先是人的网络，其次才是数据的网络。我当时使用 SSB、参与社区和设计时，这是让我觉得非常重要、也很惊讶的一点。

跟区块链、共识网络和 BT 网络比，BT 的资源是共享的，但所有人都希望搭便车，尽快下载完电影，没有动力做种，数据很难留存。区块链通过共识机制，让参与网络的节点保留全量数据，给每个人带来巨大负担。现在要跑一个以太坊节点，2 TB 硬盘可能都不太够。

SSB 可能保存 1 GB 数据，里面就是你那一百个朋友，以及更外一圈朋友的数据。它利用社交关系来分布网络数据。

同步与传播

Pub：让新人"两步之内"认识所有人



图片来源：Scuttlebutt Protocol Guide

- 新人加入 Pub = 关注 Pub，Pub 也回关
- 于是新人和 Pub 的所有成员都在 **2 跳之内**，彼此都能看到对方
- 等新人直接关注了一些成员，Pub 就只负责在线中转
- 同样利用了小世界效应：一个"枢纽"就能拉近一整群人
- Pub 就是现实世界的酒吧，是一个固定位置的公共节点，水手们上岸后喝酒聊天，传播谣言

Pub：让新人“两步之内”认识所有人

Dominic 住在船上，出海就没有网。有一段时间我也住在他家，也就是他的船上。在海上没法写代码，就聊天、干点别的。每天傍晚找个离陆地、离网络比较近的地方靠岸，开始在船上工作。

连上网后，就会疯狂同步数据、复制 Git 等等，重新连接这个网络。但如果网络里所有人高度分散，也很难高效分发数据，所以他设计了 Pub 机制。

Pub 的主人在社区里或私下发给你邀请码，你加入 Pub，其实就相当于 Pub 这个账号关注你，也会帮你同步信息。加入同一个 Pub 的其他人都在两跳以内，能够彼此看见。

它创造了一个小世界，让枢纽拉近一群人的关系。Pub 就是现实世界的酒吧，在固定的位置，也像 706 这样的公共空间：来之前不知道谁在屋里，来了以后，连接认识的人，跟恰好在场的朋友沟通大家认识的其他朋友的信息，互相传播八卦。

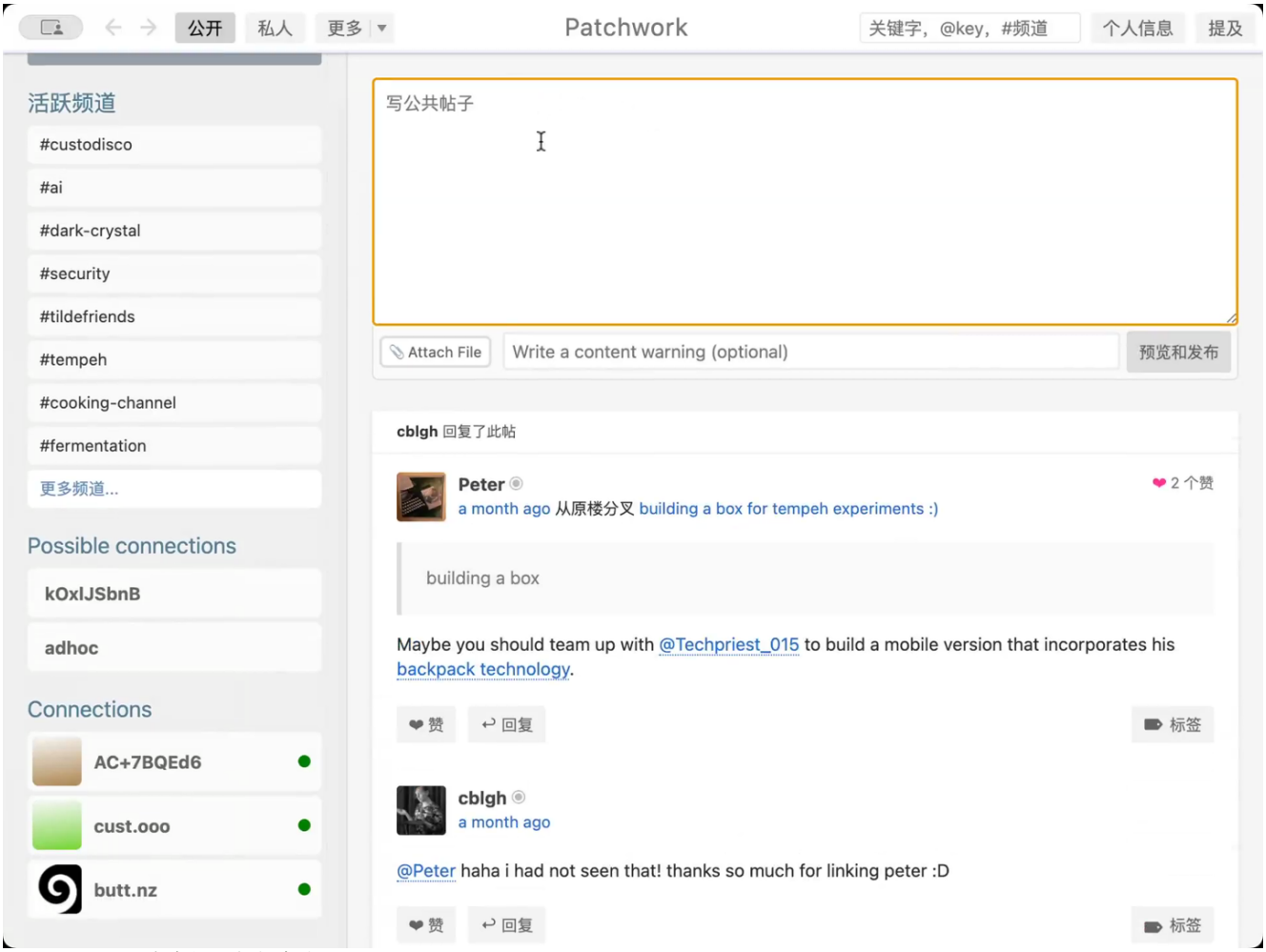
同步与传播

没有服务器，消息怎么传得开？

问题：一条新动态怎样送到成千上万人手里，又不把大家的流量和电量耗光？

SSB 的答案是一步步改进出来的 →

没有服务器，消息怎么传得开？



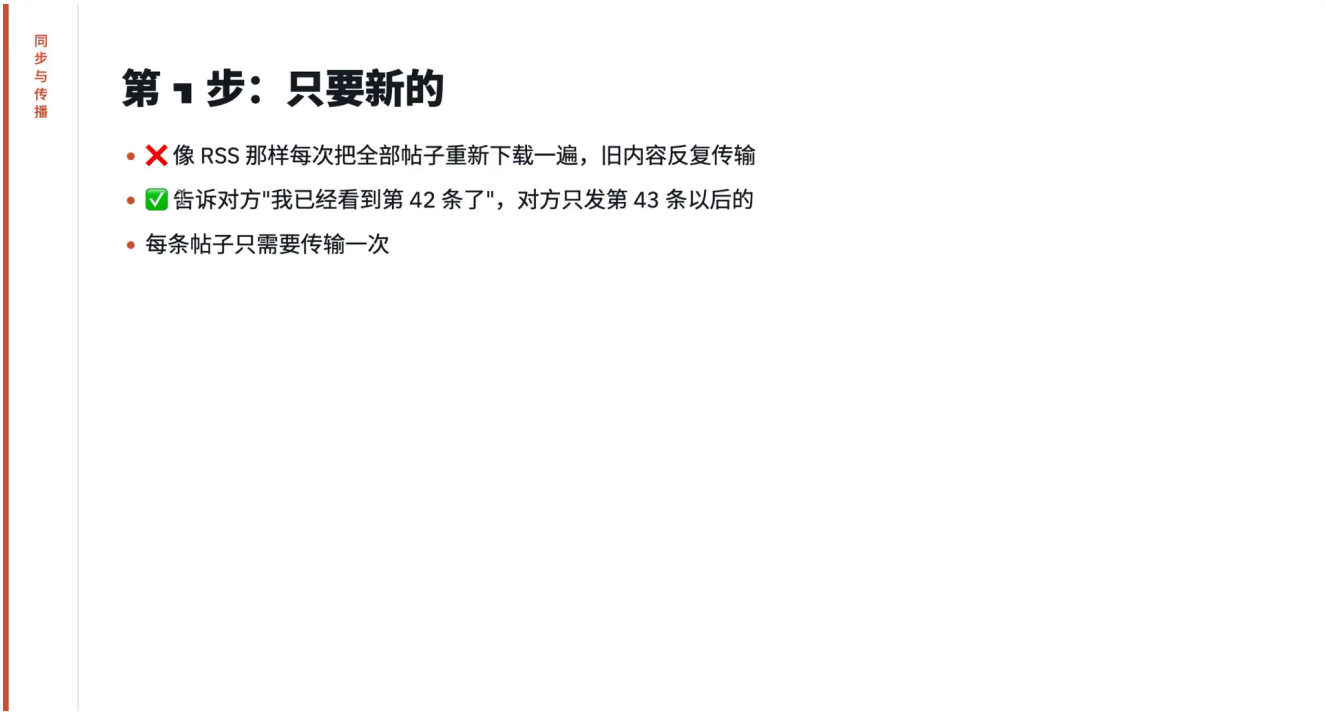
Patchwork：发布帖子与信息流

我先给大家看一下这个节点长什么样。这就是 Patchwork，一个 Scuttlebutt 节点。在这里可以写帖子，支持 Markdown 格式。因为发出的帖子不太容易删除，所以有一个预览。

这里是网络里的其他人。这是我的信息流，也就是我的日志，有些是我回复的帖子。所有消息构成一个像我自己的区块链一样的小型日志，每条消息都有相应信息，可以打开看看。

这是我的公钥地址，是一个 Ed25519 公钥。我可以编辑自己的信息，这其实也是一种特殊类型的帖子。这里可以看到好友，也就是互相关注的人，还有关注我的人、我关注的人；有 mention 机制，可以看谁提到了我，还有私人帖子和公开帖子。

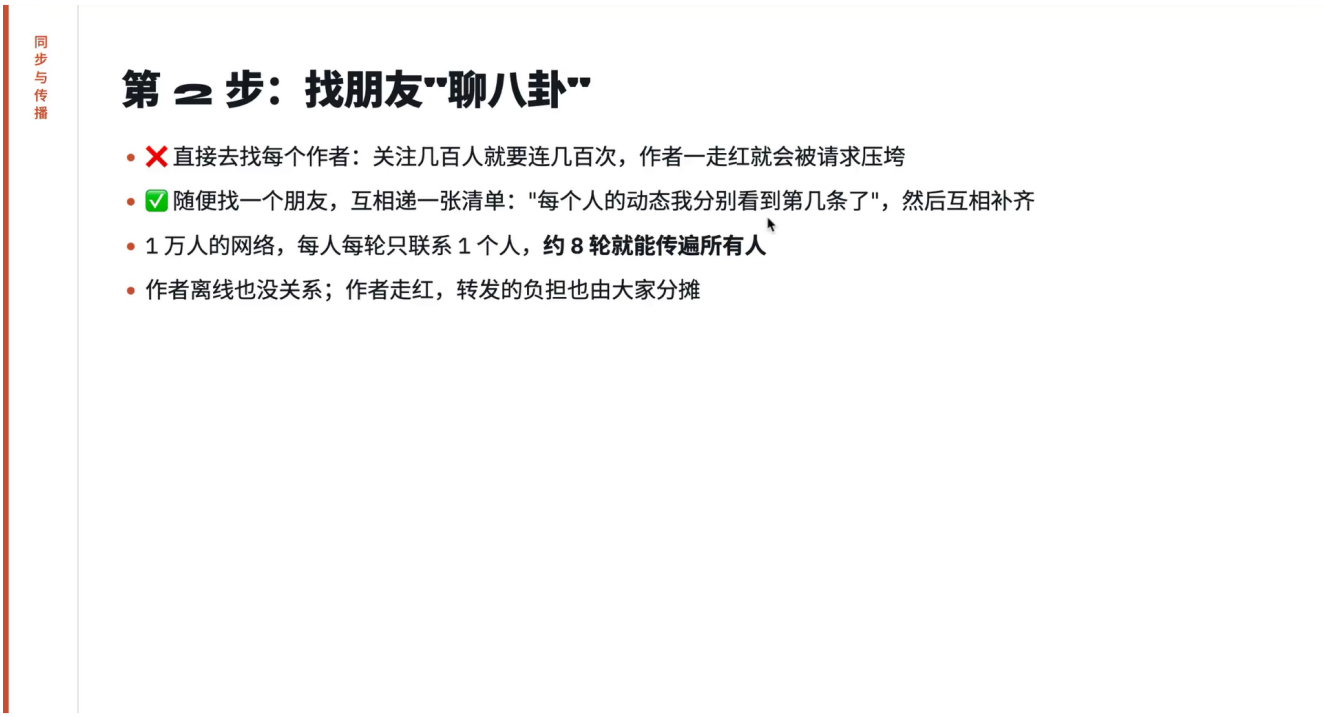
这里是当前能连上的 Pub 节点，以及它的公钥。还有 channel，也就是活跃频道。我可以粘贴邀请码，加入某个服务器。同时，即使没有服务器，在本地也可以互相连接。搭一个 mesh network，或者在线下同一个空间、同一个办公室，就可以互相通信。这里还有搜索功能。



第 1 步：只要新的

没有服务器，怎样传播消息，特别是高效传播？这是 P2P 网络最困难的问题之一。

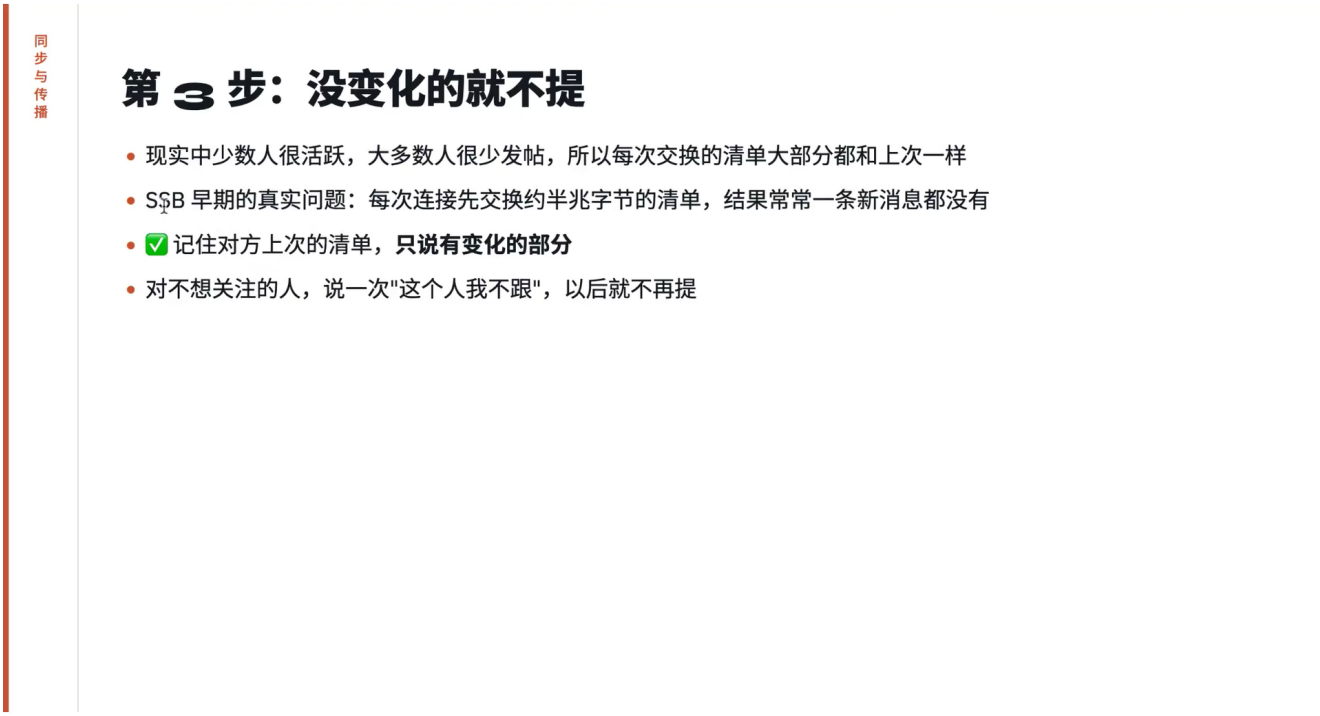
第一步，因为日志链条里每条消息都有序列号，同步时可以说：“我已经看到第 42 条了。”对方就只需要发第 43 条以后的，保证每个帖子在跟一个节点连接时只传输一次。



第 2 步：找朋友“聊八卦”

第二步优化，就是找朋友聊八卦。这是很重要的优化：连接一个朋友，先交换清单，看看我关注的人里，对方也认识哪些共同好友。

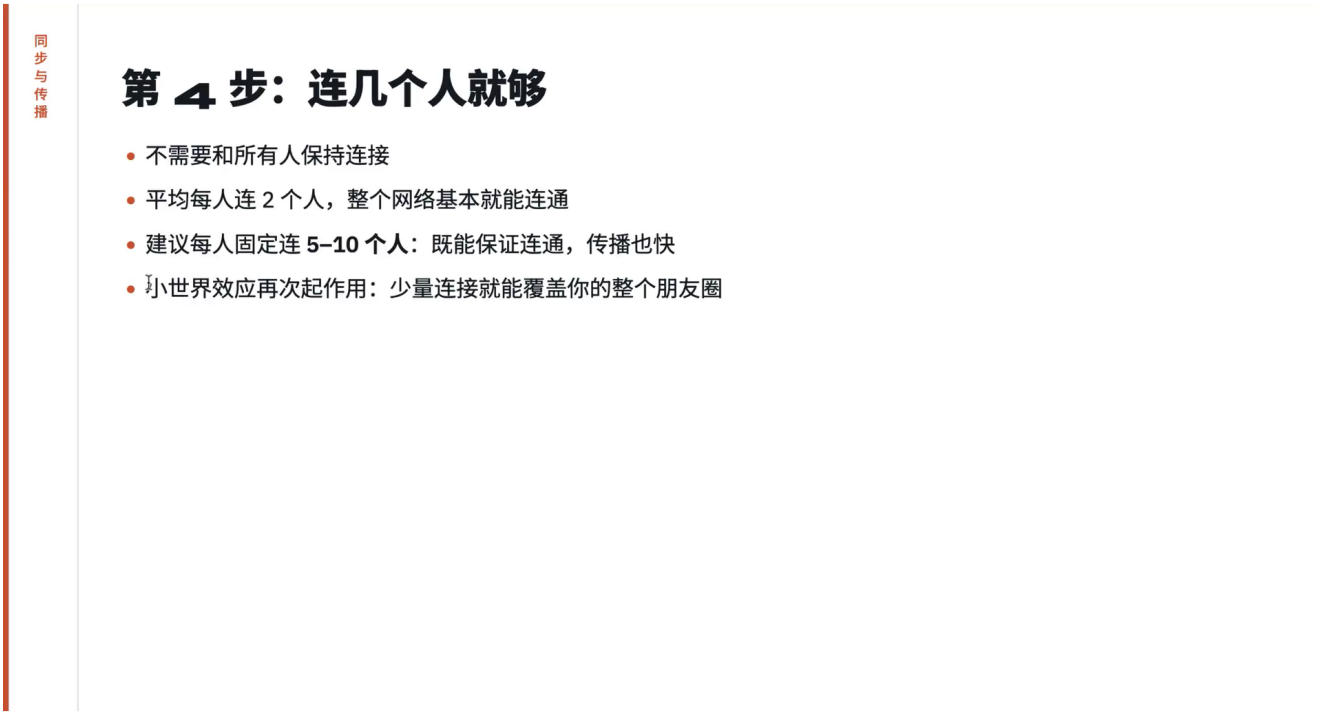
通过连接一个节点，就可以同步所有共同好友的信息。即使有些好友在你连接的时候不在线，也没关系，因为他们的信息可能已经被其他节点转发。



第 3 步：没变化的就不提

第三步，现实中只有极少数人很活跃，大多数人 90%、99% 的时间没有发那么多帖子，交换的信息跟上次一样。

可能每五分钟或每半小时连接一次，如果整个节点没有新消息，双方就说没有新消息，不需要再传新数据。



第 4 步：连几个人就够

另外，刚才也说过，利用小世界效应，连接少量节点就能覆盖整个朋友圈。



第 5 步：连上就别挂断

还有一点，每次重新连接可能比较慢，所以它们实际上有快速推送：当你保持和节点连接，对方知道你的关注清单，有相应信息时就可以直接推过来。消息实时抵达，不需要反复重新连接。

同步与传播

第 6 步：别重复收同一条消息

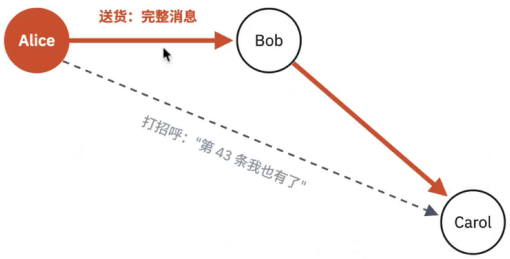
同时连着好几个朋友，同一条消息会从好几个方向送来。 每人连 5 个人时，传输量大约是实际需要的 **9 倍**。

解决方案：**Epidemic Broadcast Trees（EBT）**，思路像快递网络 →

第 6 步：别重复收同一条消息

同步与传播

EBT：一条路送货，其他路打招呼



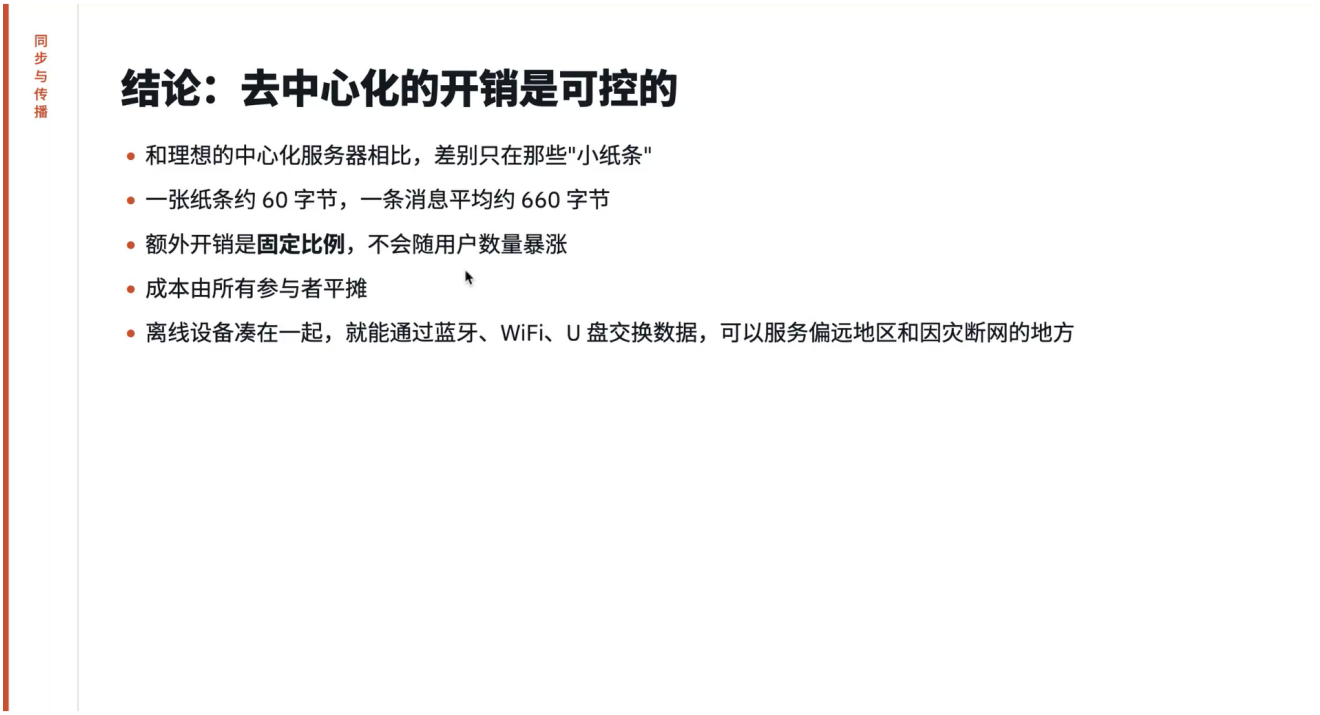
Bob 掉线 → Carol 从招呼里发现漏了 → 请 Alice 补发，改走这条路

- **最快的那条路负责送货**：谁先把消息送到，以后就由这条路送完整内容
- **其他路只打招呼**：发一张很小的纸条，几条可以攒在一起发
- **路断了立刻换**：一个来回就能修好

EBT：一条路送货，其他路打招呼

最后，Dominic 用了一个叫 Epidemic Broadcast Trees，也就是 EBT 的算法。你只需要维护少量活跃连接，消息也可以高效送达。

这是我以前参与开发时觉得非常有意思的部分。信息传播有 pull 和 push，也就是拉模式和推模式。如果保持拉取模式，每次可以批量拉信息，但间隔会比较大。



结论：去中心化的开销是可控的

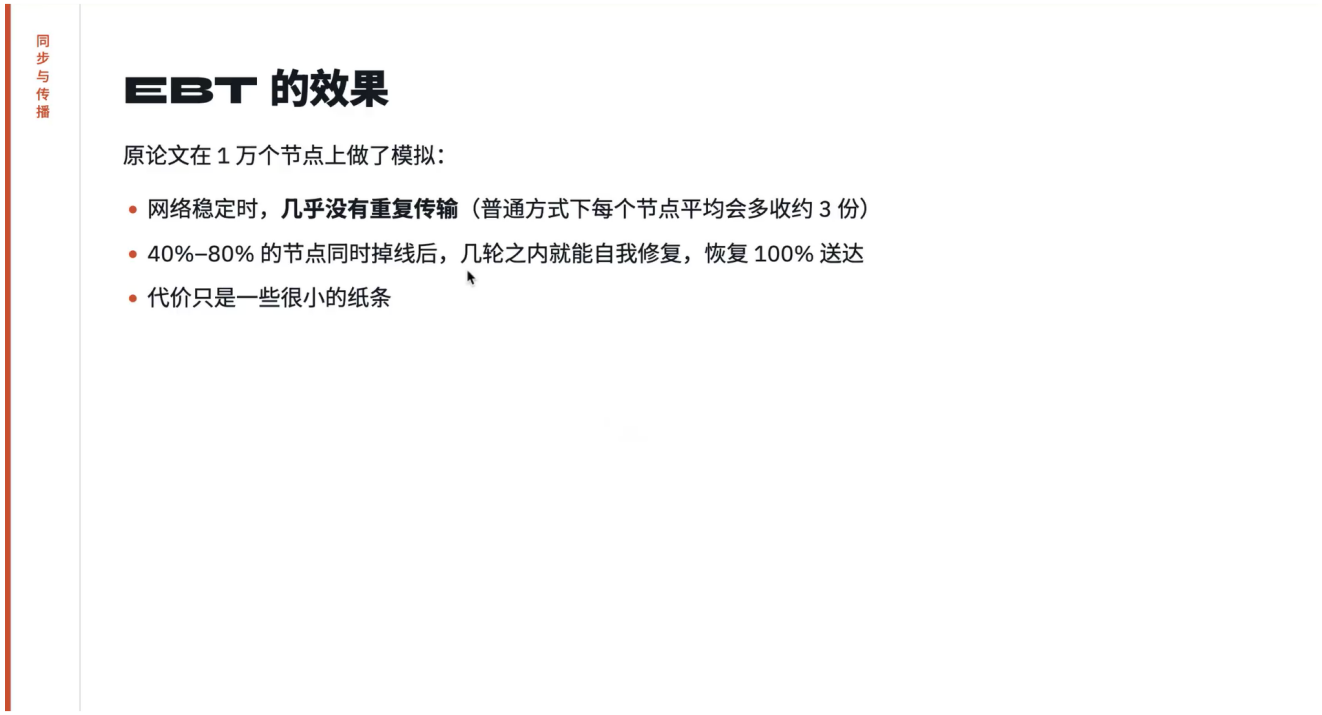
一条信息可能离你隔了几跳，如果每个人过十分钟才拉一次，最长可能要三十分钟才能拿到。你发一个朋友圈，要很久才被朋友看到。

用了 EBT 后，一方面维护活跃连接，同时也会查询其他节点。如果发现别的节点能更快传递信息，就把通路从原来的路径切换过去，提高消息传播的速度。

还有一个很重要的特点。大家设想，在点对点网络里，是不是希望连接的节点至少跟自己在同一个国家？就像用 BT 下载电影，有些节点在美国，有些在日本，有些在中国，从中国的节点、镜像站下载可能更快。

EBT 对节点返回信息的延迟敏感，会带来一个效果：选择离自己更近的节点，找出更快、距离更近的节点来传播同一个信息源。这个节点可能直接就是作者，也可能是网络状态更好，或者离你特别近。

这些都有利于高效传播。P2P 网络里，大家自愿分享，网络环境是动态的，每个人维护节点都消耗资源。通过这些优化和 EBT 算法，可以比较及时地获得信息，也能选择离你更近、响应更快、当前容量更大的节点。



EBT 的效果

根据抓取的网络数据，他们做了模拟：网络稳定时几乎没有重复传输；那些快的节点掉线后，系统依然能迅速恢复，不会丢失网络韧性，代价只是百分之几的额外信息。

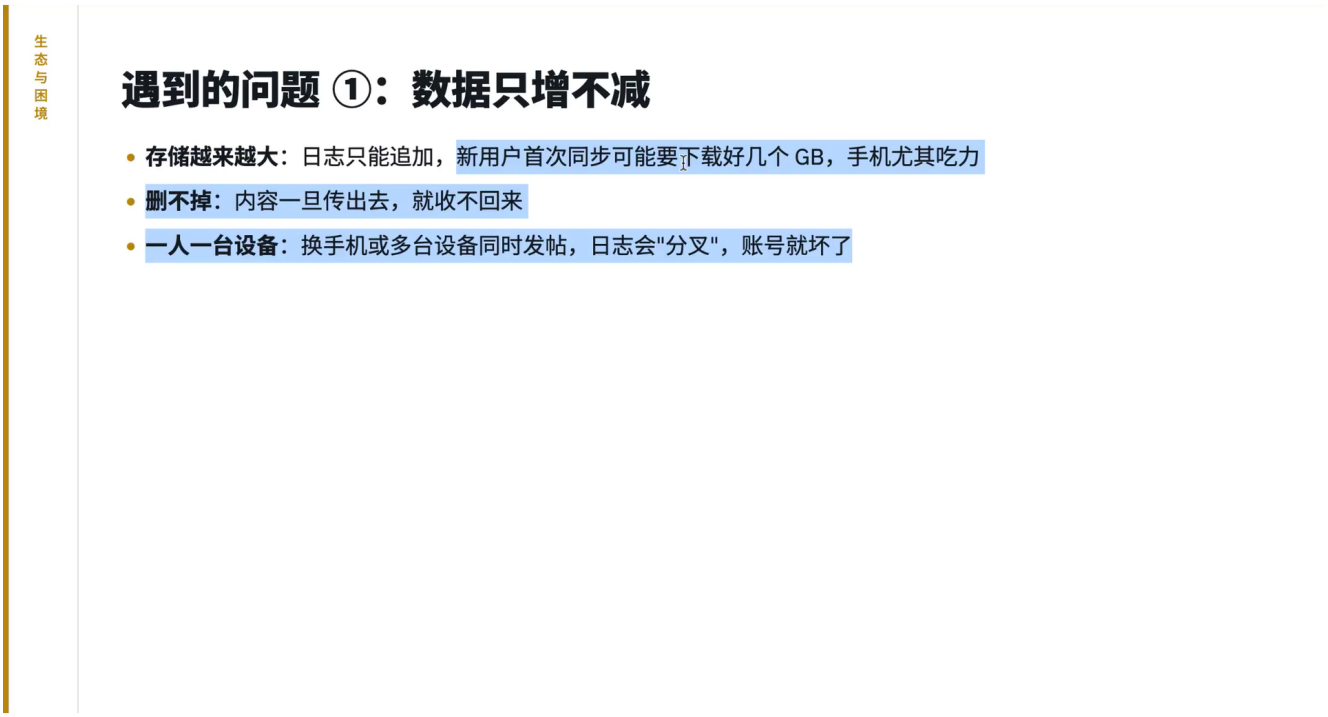
所以，去中心化网络的信息传播开销是可控的，成本由所有参与者平摊，不依赖某几个大节点提供服务。

SSB 特别去中心化，和协议无关。即使设备离线，或者只有蓝牙、Wi-Fi，最低限度只有 U 盘，依然可以同步数据。它完全可以服务于偏远地区，或者因为灾害而中断网络的地方。



有哪些应用

目前有这样一些应用，不过最近几年，这个社区没有什么人在维护了。

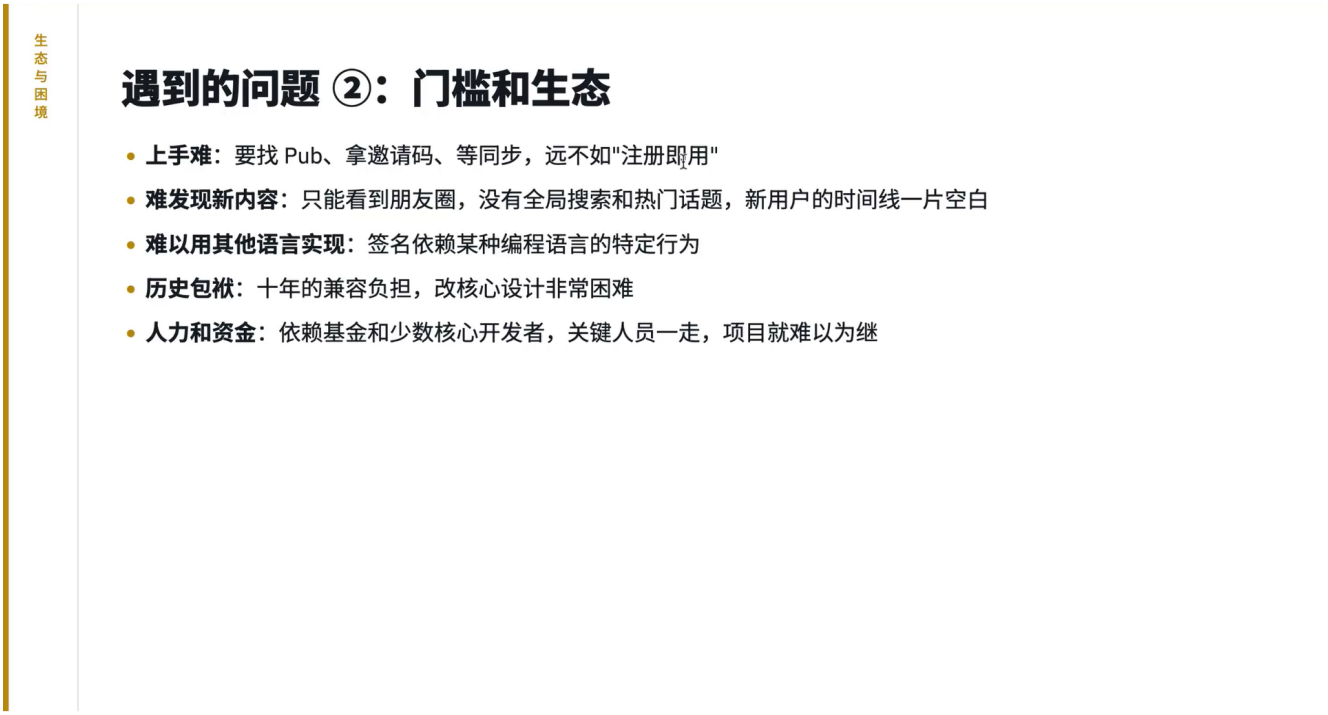


遇到的问题①：数据只增不减

其中一个问题是日志只能追加。不过你天天发帖也发不了几个，一个人的数据很难超过几兆，主要还是图片占空间。

我当时也问过他怎么办。他提的办法是根据好友关系决定是否保留：自己的文件百分之百保留；一度好友的区块可能有 50% 的概率保留；三度好友可能有 10% 的概率保留。尽量让数据均匀分布在网络里，而不是依赖某个节点全量存储。

同时，因为没有一个公共广场索引所有节点，这其实像一种弱隐私功能。



遇到的问题②：门槛和生态

生态与困境

后继者 PZP：换个思路重来

2024 年 Manyverse 团队推出，保留了 SSB 的理念（本地优先、点对点、反监控），改掉了关键问题：

- 存储有上限（默认 100MB），可以删除内容
- 不必同步完整日志
- 原生支持多设备，不会再"分叉"
- 使用标准化的数据格式

但发布的同时，开发者也宣布退出。PZP 目前更像是一份设计遗产。

后继者 PZP：换个思路重来

不认识你的人连接不到你的节点，你也可以自己小圈子搭一个 Pub，别人检查不到。但一开始没有几个网红、大 V 吸引你，普通用户进来没有朋友，不知道做什么，会比较难持续使用。

最后资源实在太少，项目就没有得到太多维护。

对比与启示

横向对比：四种去中心化社交协议

	SSB	Mastodon (ActivityPub)	Bluesky (AT Protocol)	Nostr
架构	纯点对点	许多服务器互联	分层：个人数据仓库 + 全网汇聚	客户端 + 多个中继
身份	一把密钥	服务器上的账号	可迁移的 ID + 域名	一把密钥
换平台	天然可以	有限	可以	天然可以
离线使用	**强**	不行	不行	弱
全局搜索	没有	有限	有	看中继
删除内容	不行	尽力而为	尽力而为	请求中继删除
规模（2026 约数）	数千人	约 200 万月活	4000 万+ 注册	小众

横向对比：四种去中心化社交协议

这里可以对比几个网络协议。SSB 是纯点对点，身份就是一把密钥，可以随时切换应用和平台，也能离线使用。就算 Nostr 或 AT Protocol，也不能这样离线使用。

它没有全局搜索，不利于像谷歌那样索引所有数据，但更像无数个小村落、小 tribe 的聚集。数据不太容易删除，后来也有一些协议告诉大家尽量删除。现在仍有少数人在运行节点。

ActivityPub 其实还是互联网协议的扩展，在一定程度上也可以删除数据。还有 AT Protocol，它有 DID，但数字身份还是基于域名。域名需要注册、续费、维护，所以不能离线使用。它可以索引所有内容，更像推特式的超级广场。

Nostr 的架构有很多 relay。可以把 Nostr 的 relay 理解成 SSB 的 Pub，也可以把它看成纯粹依靠 Pub 运作的 SSB。它同样用密钥作为身份，但没法离线使用。

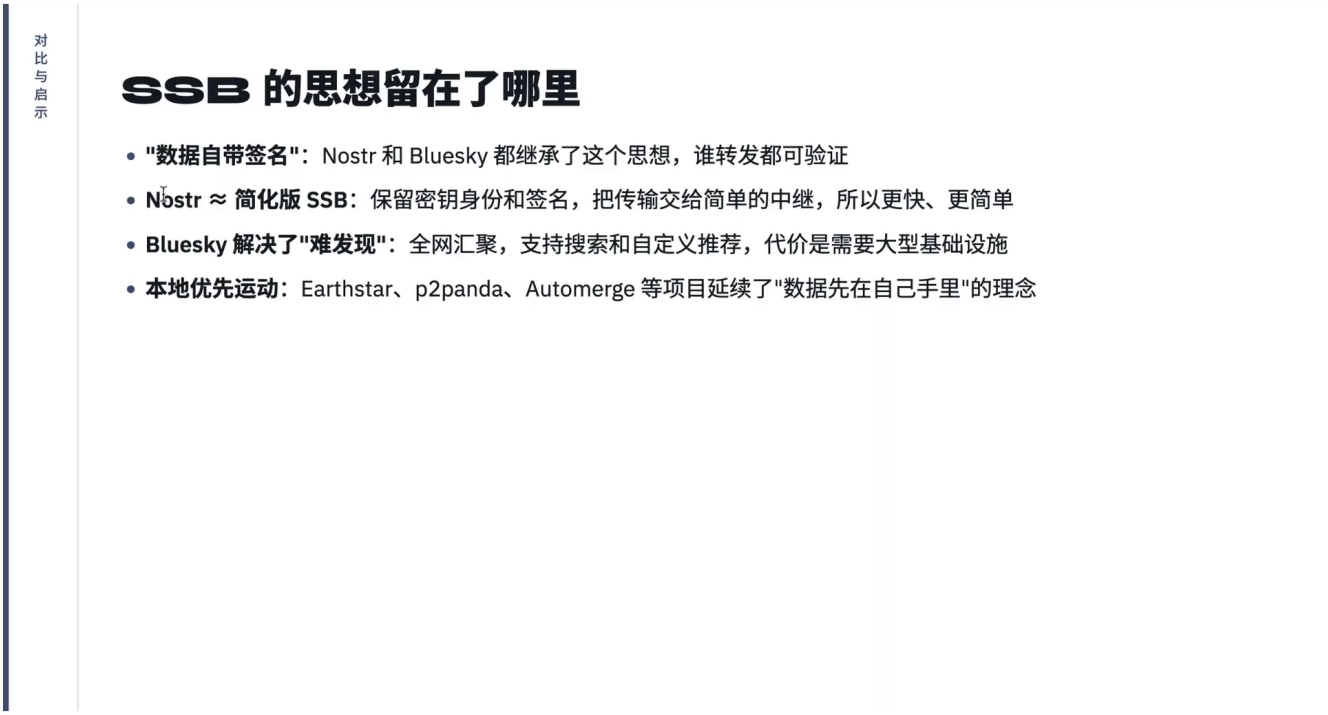


去中心化是一道光谱

可以看到去中心化协议的光谱。最中心化的是推特，一个中心平台存储全世界所有人的帖子。Bluesky 也是比较大的广场，但可以托管自己的服务器节点。Mastodon 则是社区托管小节点，彼此互联。

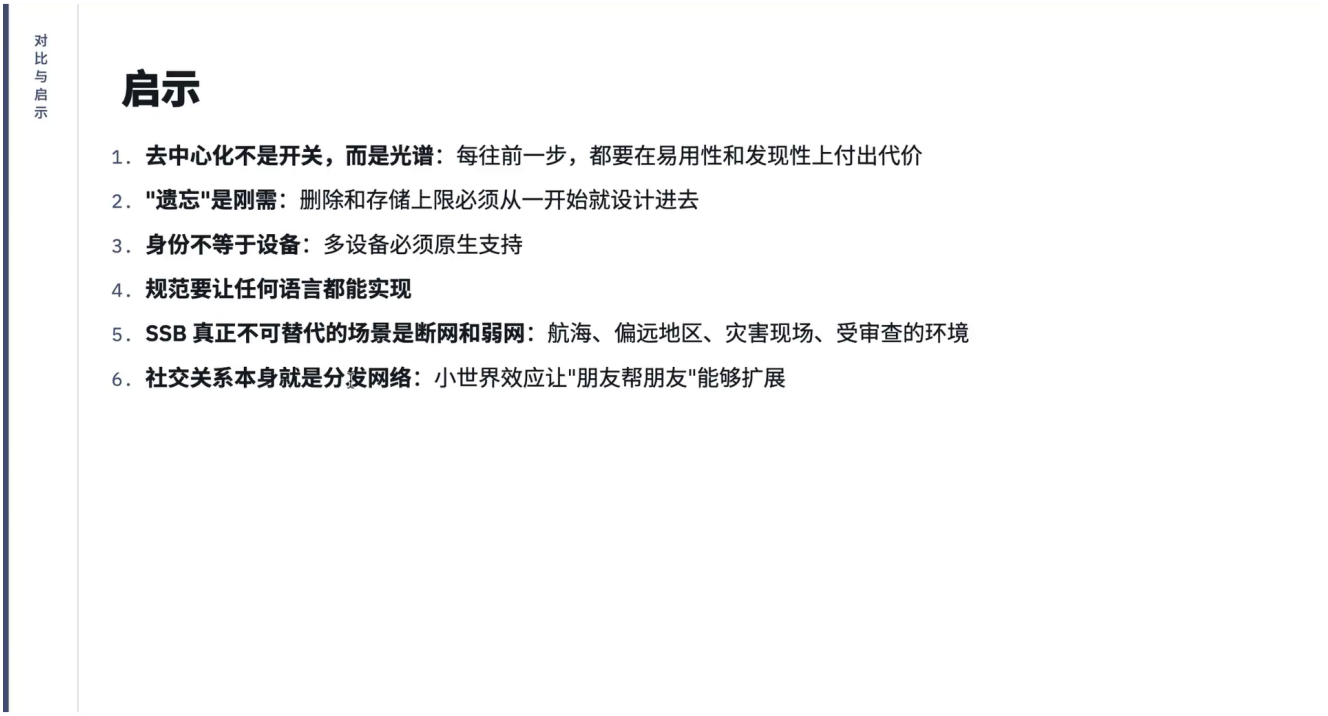
Nostr 跟这些很像，但不依赖域名作为身份。SSB 是纯点对点的，主要基于 TCP/IP。

后来有一位 SSB 社区参与者 Christian，是瑞士巴塞尔大学的计算机教授，也是一位非常厉害的无线电专家。他在做一个新的 tinySSB，基于 LoRa 无线电、蓝牙等传播信息，是巴塞尔大学的 Christian Tschudin 设计的。



SSB 的思想留在了哪里

SSB 的思想影响了很多后来的项目，包括数据签名、微型区块链、个人区块链。它影响了 Nostr 的中继设计。Bluesky 提供了全网索引、搜索功能，但代价是依赖比较大型的服务方。

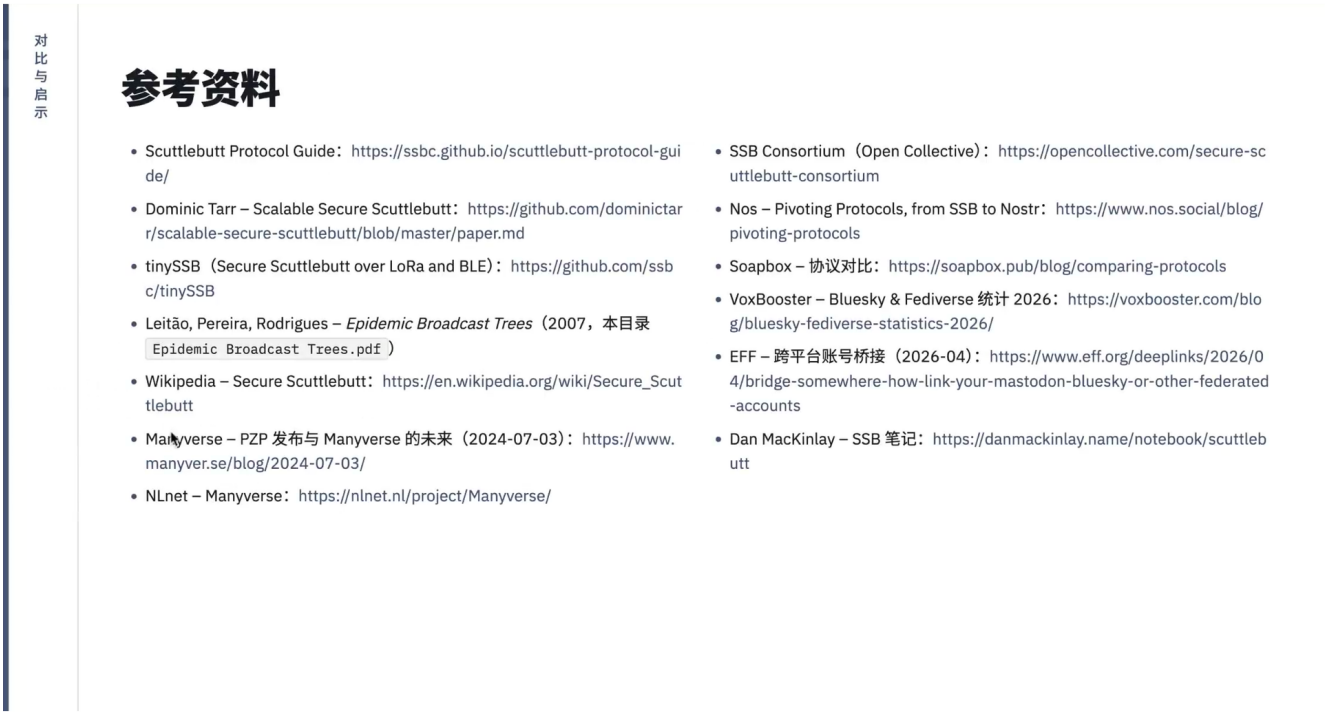


启示

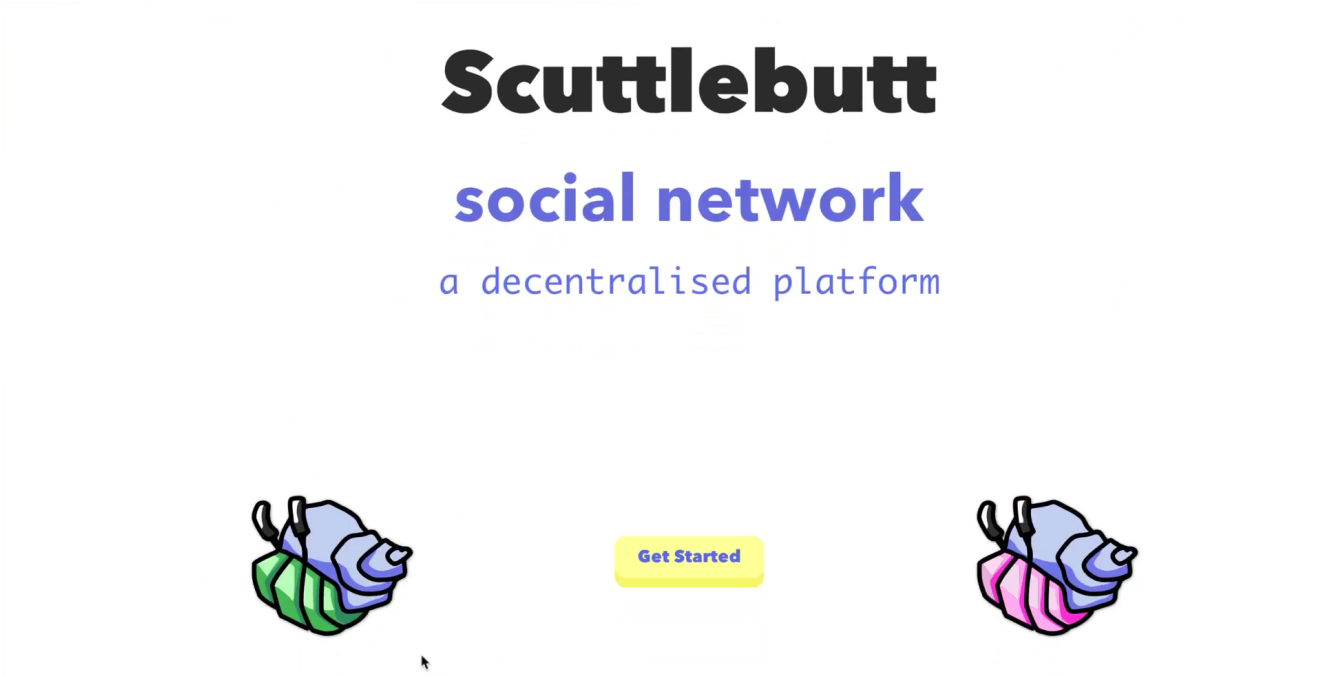
后来很多新协议都基于 SSB 进行设计。去中心化有不同层级，而数据删除、数据留存机制也需要设计。

SSB 真正不可替代的地方，是可以在局域网、甚至断网的环境中运行。最精妙的一点是，它的社交网络本身就是分发网络。

这也跟 Dominic 的思想有关。他是一个非常友善、有很多朋友的人，到哪里都会有人给他一个房间住。他走到哪里，自己就创造一个社交网络，信息也是通过这个社交网络分发的。



参考资料

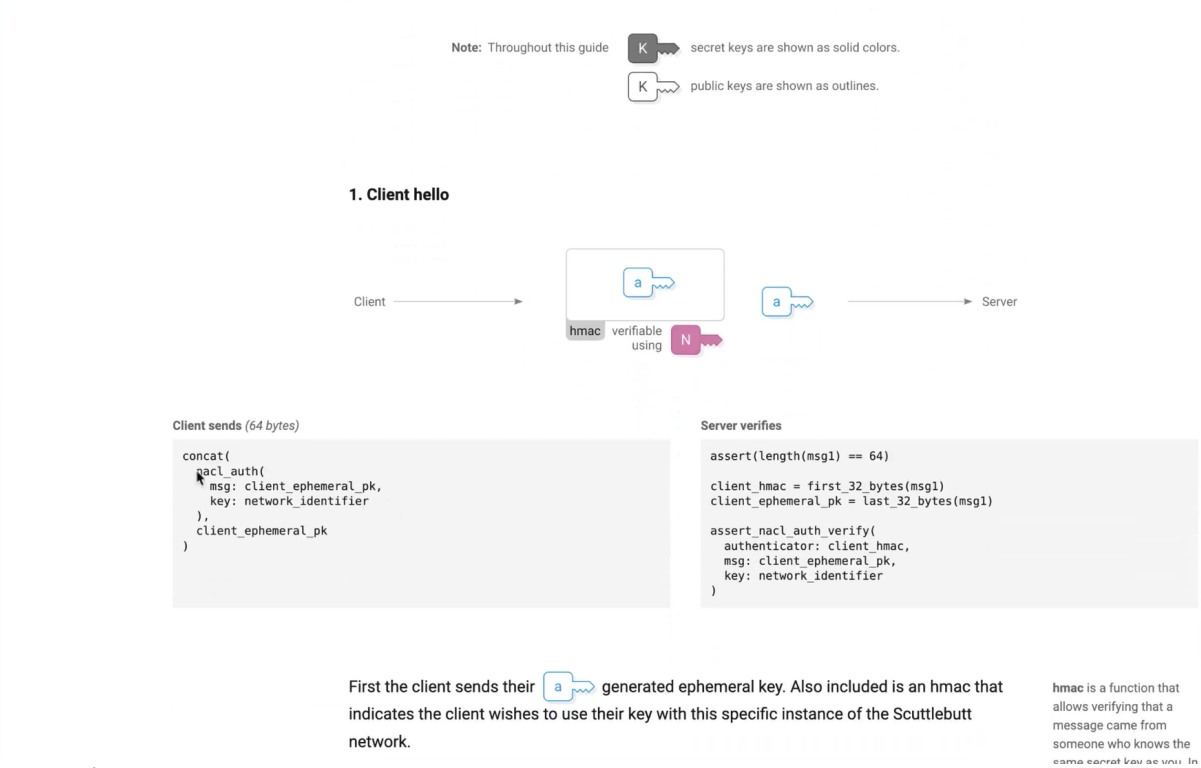


Scuttlebutt 社区网站

SSB 还是一个非常友爱的社区。我参与的那段时间，他们设计了这个，其实是寄居蟹。这是 Dominic 的一个访谈。

所有这些事情都是社区自愿维护的，没有太多资源，维护了大概十年。

在 Christian，也就是那位瑞士教授的推动下，他们还发了几篇论文，很系统地描述 Scuttlebutt 的协议设计，可以在这里看整个系统怎样运转。很多设计以很有趣的方式耦合，做出了跟区块链不一样的设计。Dominic 在去中心化 Web 的活动上也有一些分享。



协议指南：Secret Handshake

我找一个文件。他们做了一件非常好的事情：把整个协议做成可视化的文档，告诉你怎么签名、格式是什么、握手时怎样交换密钥、整个数据格式怎样设计。不是用很枯燥的代码呈现，而是做成非常可读的指南。

每个加密信息流怎么打包，这些格式都能看到。



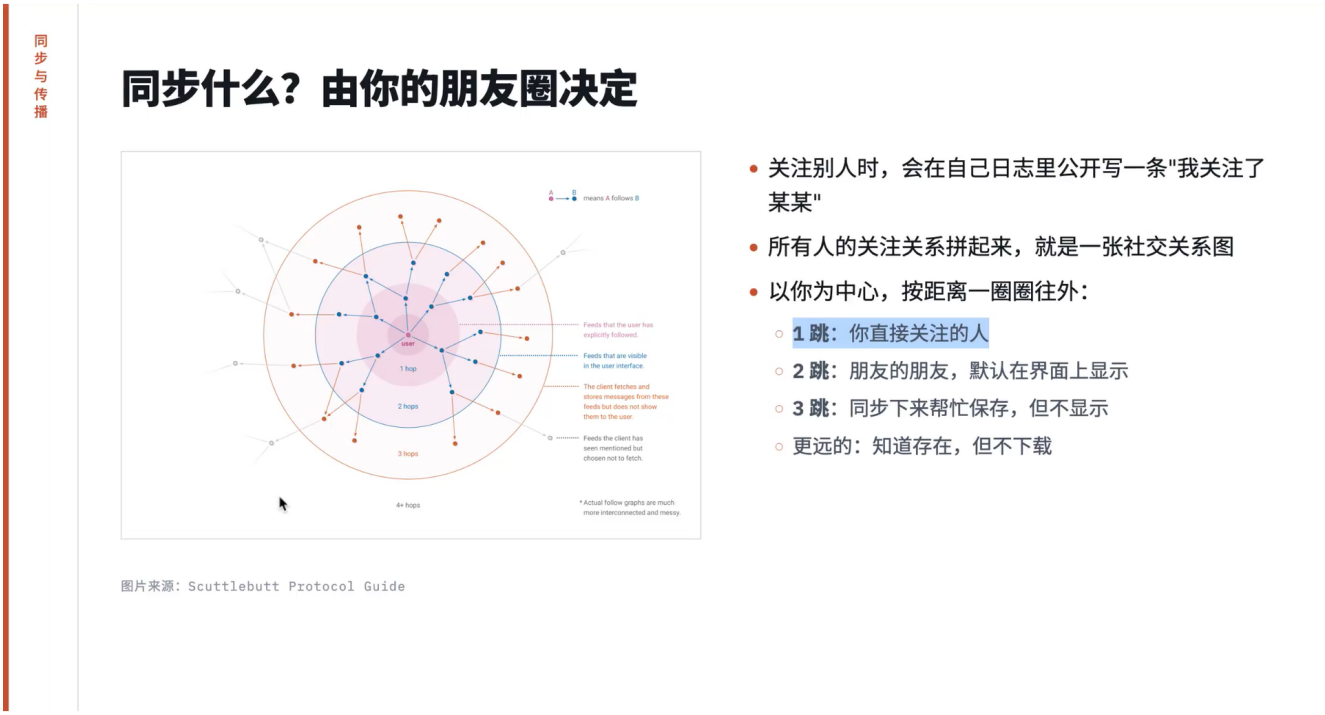
协议指南：消息格式与签名

我们看一条消息。首先是内容，text，例子是“this is the first post”。消息类型是 post，也就是一个帖子。整个这一块是 content，内容部分。

key 的哈希方式使用 SHA-256，这个协议是可选的。发布时间是时间戳，序号是 1，作者是这一位。前一条消息的哈希是空。作者用私钥给这段 string 签名，这是签名，整个信息的 key 是这个。

到第二条消息，就需要引用上一条消息的哈希。还有传播、接收时的时间。

这里描述了链路、微型区块链的消息格式，以及内容怎样设计，帮助你理解整个协议。还有数据复制、同步需要的一些比较复杂的机制，以及快速传播和信息验证。



同步什么？由你的朋友图决定

大家想想，当时把这么复杂的协议做成可视化指南，而不是写成长长的文档，需要多大精力？写文档本身已经很难，做成这样的指南更难。

这是刚才那张图：一个用户的多级好友，以及数据同步的差异。这是 Pub，公共节点，通过它高效传播消息。也要保护公共节点本身，不能让所有人随意加入，所以有邀请码，邀请码也有设计好的格式。还有 Room，信息传播的小空间，以及私信的加密方式，都描述得很详细。

像这样的事情，现在可能一个 prompt 就解决了，但那时候要付出巨大的劳动、创意和热爱才能做好。

Dominic 很有意思。他有很多独特的软件思考，才会把这些东西拼成这样一个无与伦比的软件。到现在，利用社交关系存储信息这一点，我觉得也很少被超越。

现在经常用 proof of work，通过消耗电力，或者通过钱来解决问题。当年是 Dominic 告诉我什么是以太坊的。我研究以太坊几年以后，有一次去他在新西兰的家，我问他：有人用 micropayment，微支付，解决信息传播中的认证、过载和存储问题，你为什么不考虑？

他说了一句话让我很感动：这些要付钱的网络是为富人服务的，穷人用不了。穷人没有钱，但有朋友。他要做一个穷人也能用的、只要有朋友就能用的社交网络，而不是有钱才能用的。

想想 Web3 里那些 Lens、付费社交网络，帮你一夜暴富的社交网络，Farcaster 这种，整天说知识变现、关系变现、流量变现，唯独不说这些人是不是你的朋友。Dominic 那时就说，他只会做这样的设计。



避免集中式和单例模式

Dominic 还有一篇很有洞见的文章，是对去中心化意义的思考，叫 Avoid Centralization and Singletons，避免集中式和单例模式。我可以在这里给大家翻译一下。

什么是单例，或者全局结构？为了避免中心化，每个节点都要有相同能力。服务容量可以有区别，但不能在角色上不对等。

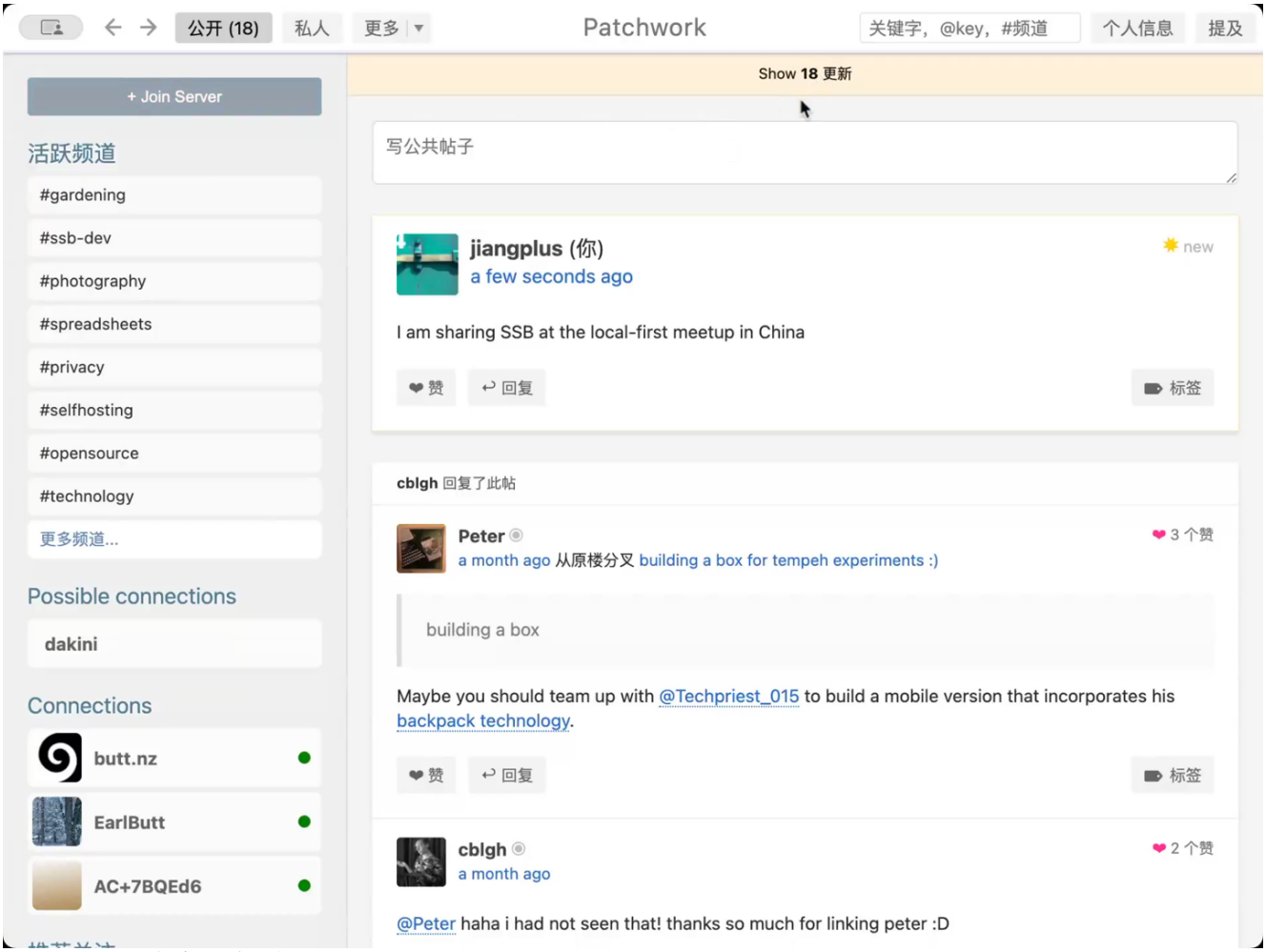
电子邮件表面上是去中心化的，每个人可以运行自己的邮件服务器，但服务器宕机，就收不到邮件。像推特、Facebook，网站挂了或公司破产，数据就没有了。GitHub 也是，如果公司钱花完了、交不起网费，虽然 Git 数据可以各自同步，网络本身却很难再连接。而且它们控制在一个机构、几家公司手上。

区块链虽然由所有人维护，本身仍是一个单例，因为它依赖共识算法挖矿。共识算法受到攻击，服务就会受影响。

Scuttlebutt 的设计是没有任何单例，没有一个全局存在。不必担心天塌了，只要还有几个朋友，朋友之间还是能用 Scuttlebutt。

所以这是一个非常独特的设计。我长篇大论的分享先到这里。

我刚开始搞区块链的时候，就跟区块链圈的人说，你们应该思考一下。2018 年讨论扩容时，我做过一些 SSB 分享：如果每个人有一条链，会怎么样？整个网络的容量就可以是无限的。你的链在自己的硬盘上，你的链很厉害，就会有更多人复制；但如果占用大量空间、信息又没什么价值，就没人复制你的数据。



Patchwork：收到 18 条更新

SSB 经历了很多起伏，主要还是资源匮乏，只有少数志愿者在维护，极少数节点还在运行。我一直有计划重新修复一下。现在运行的这个客户端，已经三四年没人修 bug 了，有点惨。

但你看，刚才启动了一会儿，又有 18 条信息更新过来。这里分享的人都很有才华，以前大家就在这里讨论这个软件该怎么做，非常“吃自己的狗粮”。

SSB 社区的人很有趣。我刚好认识创始人，他一开始拉进来的人也都很有趣。我最早就是在这里认识太阳朋克、Microsolidarity，也就是微观团结，还在这里学习了很多生态主义理论。

交流与问答

Jiang

大家还有什么问题吗？

观众

我想问一下，SSB 是把所有人的消息像区块链一样记录，还是每个人单独记录一条链，后面再同步？

Jiang

每个人有一条链，然后互相同步。你会同步你关注的人。

观众

有人不在线时，同步会出问题吗？比如有 A、B、C 三个人，A 是主要发送方，B 和 C 不一定在线。A 向 B 同步之后，B 下线，A 再向 C 同步。这样循环几次，B、C 之间同步会出现数据冲突吗？

Jiang

不会冲突。这就是刚才说的机制，你从朋友 A 就可以了解你们共有的信息。

观众

万一之前的信息不属于共有的，中间某段出现断层，还没有人同步这段信息，会出现这种情况吗？

Jiang

会有时间延迟，但通常同步还是挺高效的。刚才说 EBT 为什么保留 push 过程，就是为了减少时间延迟。它会选择 push 更快的节点，相当于选择离自己更近的节点。

观众

如果 push 更快的节点，数据并不实时呢？比如它前段时间下线，最初发推文的人的节点还没上线，还没有把数据同步到这个较近的节点。现在这个靠近接收方的节点和接收方都上线了，中间会出现推文同步问题。

Jiang

EBT 优化的是都在线的节点的选择问题。这里有几个层面：底层是 gossip protocol，也就是 gossip 算法。

要了解整个网络的信息，不需要所有节点都保持互相连通，那样传输代价太高。每个节点随机连几个，几轮传播以后，信息基本上就能有效扩散到全网。可以搜一下 gossip，也就是谣言传播算法。

EBT 优化的是当前都在线的节点。比如 A、B、C 都在线，你是 A，可以连接 B，让 B 告诉你 C 的情况；也可以连接 C，让 C 告诉你 B 的信息。如果三个节点的信息是共有的，不需要 B、C 都连，只连其中一个，让它传播另一个节点的信息就行。

怎么选择？看 B 和 C 哪个离你更近，就连哪个，这样速度快、距离近。

观众

还有两个问题。第一个，协议设计时考虑过前向安全吗？

Jiang

SSB 使用的加密传输协议是有前向安全的。

观众

好的。另一个问题，之前说可能有大量无效数据占用存储空间。因为它本身也是一种区块链，能不能把发帖者自己认为不太重要的数据，不一定说是没有意义的数据，算成一个哈希值，用这个哈希值代替前面所有的数据？这样是不是能节省较多空间？

Jiang

它用一个更天才、更简单粗暴的算法：空间不够，先删跟你不熟的人的数据。你听懂没有？

观众

就是说，它们没有那么重要。

Jiang

对。刚才说它很天才的一个设计，就是节点会同步一度、二度、三度好友。空间多的时候，你帮大家都存。按时间屏蔽也是对的，但它有一个基本策略：空间不够，先删跟你不熟的人。

观众

假设用这个协议做一个类似朋友圈或空间的东西，我恰好对所有跟 AI 相关的内容感兴趣，这些人我都不能删，怎么办？

Jiang

总有个取舍吧。删除和留存数据有很多策略，SSB 里有一种其他网络没有的策略，就是先删不熟的。总的来说，空间很紧张，就不要存那么多数据。

观众

我没有别的问题了，谢谢。

Jiang

我只是说，这一点很有趣，是其他网络不存在的设计点。大家还有问题吗？

最后，SSB 确实是本地优先的。它在十二年前，也就是 2014 年就开发了，很适合本地优先的理念。

主持人

谢谢 Jiang。我再补充一个好玩的，刚才在聊天框也说了：其实本地优先就是一个链，因为要让每个操作、数据保持顺序，再累增、合并。

Scuttlebutt 最初有很多思想，可以重新考虑怎么工程化，用 CRDT 再做一遍。我不知道特性边界会怎么样，但可能会很不一样。这几年，尤其这两年，这个领域发展这么快，有这么多新的想法和思想，我感觉会有很多好玩的东西。

大概就是这样，大家应该没有问题了。

Jiang

好，那今晚先到这里，谢谢大家。

主持人

谢谢 Jiang。