The-Book-of-Swarm.pdf

前言

目标读者

本书的主要目标是汇总 Swarm 项目第一阶段的大量成果，作为各团队和个人在未来阶段中推进 Swarm 项目时的参考资料。

本书面向技术倾向明显的读者，这些读者希望将 Swarm 集成到其开发栈中，并理解该技术背后的动机与设计选择。此外，我们也邀请研究人员、学者及去中心化领域的专家对我们的论点进行审查，并对 Swarm 整体设计的逻辑性进行审核。对于核心开发人员以及通过构建组件、工具或客户端实现为整个生态系统贡献力量的开发者来说，本书提供了具体的规范及背后的设计思路。

本书的结构

本书分为两个主要部分。

第一部分・序曲

深入探讨了 Swarm 项目的动机，通过描述历史背景奠定了公平数据经济的基础，并展示了 Swarm 的愿景。

第二部分・设计与架构

全面且深入地阐述了 Swarm 的设计和架构。该部分涵盖了 Swarm 核心功能相关的所有领域。

为了补充主要内容，本书还提供了有用的资源，包括索引、定义技术术语和缩写的术语表，以及附录，为读者提供一个全面且完整的参考手册。

如何使用本书

“序曲”和“设计与架构”两个部分已无缝衔接，形成了连贯而完整的叙述。对于那些希望直接了解技术细节的读者，可以从“设计与架构”部分开始，跳过“序曲”部分。

另一方面，Swarm 客户端的开发者可以将本书作为背景阅读材料，以全面了解相关规范。当需要更广泛的背景信息，或者想要了解开发过程中设计决策背后的理由时，这种方法尤为有用。

致谢

编辑

许多人在本书的写作过程中给予了我帮助，但有几位在书籍的最终成型过程中发挥了非常积极的作用，值得特别感谢。Daniel Nickless 是所有图表的设计者；我们花费了许多小时，将我即兴创作的可视化图示成型，而他总是乐于多次重绘。Daniel 是 Illustrator 的高手，并且为了排版出漂亮的树形图，还成为了 LaTeX 的专家。

我对 Črt Ahlin 深表感激，他不仅负责管理本书项目，还贡献了一些高质量的文本内容。他还承担了许多繁琐无趣的工作，例如编制索引和术语表。Črt 和以英语为母语的 Daniel，以及 Attila Lendvai 和最近加入的 Noah Maizels，在校对和纠正错误与错别字方面都做得非常出色。第二版的文本质量在 Andrea Robert 的辛勤工作下得到了大幅提升。除了构思和起草本书封面的设计理念，她还与 Anna Csúthy 紧密合作，后者以完美的像素实现了设计效果。Andrea 还与 Gyuri Barabás 一同监督排版，并将所有内容准备好交付给印刷厂。

特别感谢 Edina Lovas，她的支持与热情一直是推动我前进的动力。

作者

Swarm 的共同创始人是我敬重的朋友和同事——杰出的 Dániel A. Nagy。Dániel 设计了 Swarm 的基础架构，应当归功于他的一些重要架构决策、创新思想和本书中所呈现的正式分析。

Aron Fisher 对 Swarm 的贡献是无以复加的。Swarm 现在的大部分核心思想，都是在与 Aron 的讨论中萌发或完善的。他不仅在整体设计中做出了贡献，还共同撰写了最初的几篇“橙皮书”。此外，他始终活跃在第一线，在各类会议和聚会上展示并讲解我们的技术。

我感谢 Dániel 和 Aron 包容我的随意构思和未成熟的想法，并带来清晰的思路，始终理解那些复杂的数学逻辑。尽管他们不一定完全赞同或对本书的内容负责，但我仍将 Dániel 和 Aron 视为本书的共同作者。

我对我的“战友” Gregor Žavcer 怀有深深的感激之情。他在担任 Swarm 基金会现任总监的过程中，一直推动项目前进。Gregor 对项目的热诚与不懈的投入，在我经历的诸多低谷时刻中给予了我极大的鼓舞。从技术创新到道德方向，我们对去中心化未来数据经济的共同愿景构成了我们合作的基础。本书的许多内容都在我们通宵的头脑风暴会议中逐步成型，Gregor 甚至为本书贡献了部分内容。

我要感谢 Rinke Hendriksen 和 Ábel Bodó，他们主要在激励机制方面提供了重要的洞察和创新。Fabio Barone 和 Ralph Pilcher 在激励机制方面作出了重大贡献，Javier Peletier 在信息流模块方面提供了支持，而 Louis Holbrook 则在信息流和 PSS（Swarm 消息协议）方面做出了贡献。最后但同样重要的是，我们的常驻天才 Viktor Toth（又名 Nugaon），他的创造力和抱负总是让我惊叹不已。

反馈

本书在反馈中受益匪浅。那些经受住了阅读早期版本的痛苦并对正在进行的工作提供了评论、提出批评或指出不清晰之处的人值得被提及：Henning Dietrich、Brendan Graetz、Attila Lendvai，以及 IOV Labs 的团队成员：Marcelo Ortelli、Santiago Regusci、Vojt ě ch Šimetka；还有 Swarm 团队成员：Ábel Bod ó 、Elad Nachmias、Janoš Guljaš、Petar Radović、Louis Holbrook 和 Zahoor Mohamed。

概念与影响

《The book of Swarm》本身就是 Swarm 背后伟大理念的表达：去中心化存储和基于区块链的消息传递系统的愿景。Swarm 作为实现去中心化网络“圣三位一体”支柱之一的概念和最初构想，早在 2015 年初 Ethereum 启动之前就已经出现。Vitalik Buterin、Gavin Wood 和 Jeffrey Wilcke 等以太坊创始人将 Swarm 带入了蜜蜂笑话（Bee jokes）和极客幽默（Geek humor）的滑坡中。“bzz” 和 “shh” 这两个协议标签都是由 Vitalik 取名的。

那些见证 Swarm 初创阶段的人包括 Alex Leverington 和 Felix Lange。与他们的早期讨论催化了 Swarm 设计中一些基础性决策的诞生，最终形成了如今的 Swarm。

Swarm 的基础是在 2015 年奠定的。D á niel 与因轻客户端而闻名的 Zsolt Felföldi 合作，他的代码至今仍可在基于 Go Ethereum 的 Swarm 实现中见到。Zsolt 的思想在 Swarm 所设定的方向中留下了鲜明的印记。

我们对 Elad Verbin 深表感激，多年来他一直是 Swarm 的科学和战略顾问。Elad 在项目中投入了大量精力，他在 Swarm 各个方面无与伦比的洞察力和深厚理解令人受益匪浅。尤其是他对基于指针的函数式数据结构与内容寻址分布式数据之间同构关系的理解，对我们处理高级功能的方式产生了重大影响。我们关于 Swarm 解释器的工作启发了 Swarm 脚本的设计。

特别感谢 D á niel Varga、Attila Lendvai 和 Attila Gazs ó ，感谢那些彻夜的头脑风暴，我从你们每个人身上都学到了很多。感谢 Alexey Akhunov 和 Jordi Baylina 提供的重要技术洞察与灵感。我非常感谢我的挚友 Anand Jaisingh，他对我和项目的坚定信任，以及他在 Swarm 光环下催化出的独特灵感与协作效应。

在项目初期，我们与 Alex Van Der Sande 和 Fabian Vogelsteller 花费了大量时间讨论 Swarm 及其潜力。许多得以实现的想法（包括清单和 HTTP API）都要归功于他们。帮助塑造 Swarm 理念的以太坊基金会内部及周边的人员包括 Taylor Gerring、Christian Reitwiessner、Stephan Tual、Alex Beregsz á szi、Piper Merriam 和 Nick Savers。

团队

首先，我要感谢 Jeffrey Wilcke，以太坊三位创始人之一，也是阿姆斯特丹 go-ethereum 团队的负责人。他在困难时期始终支持 Dani、Fefe 和我所在的 Ethersphere 子团队，为项目提供了保护。我永远感激 Swarm 团队的所有现任和过去成员：特别感谢 Nick Johnson，他在团队短暂的任期内创建了 ENS。

我同样向那些长期陪伴团队的特别成员表示感谢：Louis Holbrook、Zahoor Mohamed 和 Fabio Barone。他们的创造力与信念在帮助我们度过艰难时期方面起到了至关重要的作用。感谢 Anton Evangelatov、B á lint G á bor、Janoš Guljaš 和 Elad Nachmias 对代码库的重大贡献。感谢 Ferenc Szab ó 、Vlad Gluhovsky、Rafael Matias 及其他许多无法一一列出的贡献者。Ralph Pichler 值得特别提及，他多年来一直是项目的热心支持者。逐渐地，他成为了团队的荣誉成员，并在实现整个智能合约套件（swap、swear 和 swindle）的初始版本中发挥了关键作用。此外，他还推动了区块链交互、带宽激励机制和密钥管理方面的开发。

我要特别致敬 Janoš Guljaš，他勇敢地接任工程负责人，在贝尔格莱德组建了一支由 Petar Radović、Svetomir Smiljković 和 Ivan Vandot 组成的卓越团队。

我也感谢 Vojt ě ch Šimetka 和 IOV Labs 的 Swarm 团队，他们在项目的关键时刻发挥了拯救作用。Marcelo Ortelli 和 Santiago Regusci 与贝尔格莱德团队一起为当前代码库做出了重大贡献。

同时感谢 Tim Bansemer，他是一个独特而高效的人才，拥有不可思议的执行力和动力。他在团队、代码、文档以及流程方面的贡献将被长期铭记。

最新的 “bee” 代码库主要是由四位“火枪手” Alok Nerurkar、Esad Akar、Anatol Lupanescu 和 Peter Mrekaj 开发的，他们是由传奇而低调的 Elad Nachmias 招募的。这支核心团队得到了 Vlado Pajić 和 Marko Blazvić 的协助。

特别感谢我们无可替代的统计与数学专家 Gyuri Barab á s，他的贡献至关重要。如果没有传奇人物 Callum Toner，Swarm 根本不可能走到今天。他整合了所有研发工作，几乎凭一己之力推动项目交付成为真正可用的产品。

生态系统

Swarm 一直吸引着大量的爱好者社区以及公司和项目组成的生态系统，他们的支持和鼓励在一些艰难的日子里给予了我们生命力。特别感谢 Jarrad Hope、Jacek Sieka 和 Oscar Thoren（来自 Status），Marcin Rabenda（来自 Consensys），Tadej Fius、Zenel Batagelj（来自 Datafund），Javier 和 Alfonso Peletier（来自 Epic Labs），Eric Tang 和 Doug Petkanics（来自 LivePeer），Sourabh Niyogi（来自 Wolk），Lewis Marshall、Vaughn McKenzie、Fred Tibbles 和 Oren Sokolovsky（来自 JAAK），Carl Youngblood、Paul Le Cam、Shane Howley 和 Doug Leonard（来自 Mainframe）。我们同样感谢 Anand Jaisingh、Dimitry Kholkhov、Igor Sharudin（来自 BeeFree）以及 Attila Gazs ó （来自 Felfele 基金会）所做出的贡献。

最近，我们收到了两位未隶属于任何组织的社区成员的大量帮助，他们在 Discord 上的用户名分别是 Ldeffenb 和 Mfw78。

我要感谢所有 Swarm 峰会、黑客周和其他聚会的参与者。本书中的许多想法都源自这些活动中的对话交流。我还要感谢 Michelle Thuy 和 Kevin Mohanan。此外，还要感谢无数的资助申请者和赏金任务提交者，他们为应用程序生态系统的繁荣作出了贡献。

支持者

感谢我的朋友和同事 Péter Földiák 的持续支持，他对项目发自内心的信任一直是推动力。在 Swarm 还未诞生之前，他就已经相信它的愿景。感谢 ICO 参与者以及早期支持者，他们的贡献帮助我们执行了这一雄心勃勃的路线图。

传承

最后，我要向我的导师和老师们致以最深切的谢意：András Kornai、László Kálmán、Péter Halácsy 和 Péter Rebrus，他们塑造了我的思维方式和技能。他们永远是我心中的智识英雄。

第一部分序曲

第 1 章 演变

本章提供了有关 Swarm 发展动机及其当前愿景的背景信息。第 1.1 节对万维网 (World Wide Web) 进行了历史分析，聚焦于它是如何演变成如今的模样。第 1.2 节引入了相关概念，并强调了数据主权、集体信息和公平数据经济的重要性，讨论了自治社会在集体托管、传输和处理数据方面所需的基础设施。最后，第 1.3 节回顾了该愿景背后的核心价值观，阐明了技术需求，并确立了我们实现 Swarm 所遵循的设计原则。

1.1 历史背景

尽管互联网（尤其是万维网）极大地降低了信息传播的成本，这些成本仍然不是零，而且这些成本的分配对谁能够发布何种内容以及谁会消费这些内容有着重大影响。

为了理解我们试图解决的问题，有必要回顾万维网的历史演变。

1.1.1 Web 1.0

在 Web 1.0 时代，如果想要让你的内容向全世界公开，你通常需要启动一个 Web 服务器或使用一些免费或廉价的网页托管空间，将内容上传，用户可以通过一系列 HTML 页面进行浏览。如果你的内容不受欢迎，你仍需承担维护服务器或支付托管费用以保持内容可访问。然而，真正的灾难发生在内容因某种原因变得受欢迎时（例如，你被“Slashdot 效应”击中）。此时，你的流量费用会暴涨，直至服务器因过载崩溃，或托管服务商限制了你的带宽，使得大部分受众无法访问你的内容。如果你想保持热门，你必须投资高可用性集群并使用大容量带宽连接，而随之而来的成本会随着人气的增长而增加，而你却没有显而易见的方式来分摊这些成本。几乎没有任何实际方式允许（更不用说要求）你的受众直接分担这笔财务负担。

当时普遍的看法是互联网服务提供商（ISP）会出手相助，解决这些难题。在 Web 革命的早期，ISP 之间关于对等互联协议的谈判通常围绕提供方和接收方的位置以及哪个 ISP 从其他 ISP 的网络中获利展开。确实，当 TCP 连接请求（即 SYN 数据包）发起者之间存在显著不平衡时，请求发起方 ISP 通常需要向接收方 ISP 提供补偿。这种设置在一定程度上激励了接收方 ISP 支持那些托管热门内容的用户。然而，在实践中，这种激励结构通常导致某些 ISP 在服务器机房中放置免费的“色情”或“盗版”服务器，以重新平衡 SYN 数据包的计数。这意味着迎合小众受众的博客无法与之竞争，通常会被冷落。然而，需要注意的是，当时内容创作者-发布者仍然通常拥有自己的内容所有权。

1.1.2 Web 2.0

向 Web 2.0 的过渡改变了这一切。人们从运行在自己服务器上的个人主页——使用 Tim Berners-Lee 优雅简洁且易于使用的超文本标记语言（HTML）——迁移到使用 CGI 网关、Perl 脚本和无处不在的 PHP 进行服务器端脚本编写，从而偏离了“每个人都可以使用简单工具编写和运行自己网站”这一美好理念。这种偏离使得网络走向了一个越来越复杂且令人望而却步的脚本语言和数据库技术栈。突然之间，万维网不再是初学者友好的地方。与

此同时，新技术的出现使得用户界面更加简单的 Web 应用成为可能，这些应用让非技术型的发布者只需将数据 POST 到服务器，而无需承担将内容传输到终端用户的责任。这标志着 Web 2.0 的诞生。

最初承载网络创作者精神的网站，如 MySpace 和 Geocities，开始占据主导地位。这些平台为用户提供了属于自己的个性化互联网角落，配备了滚动的文本标语、闪烁的粉色闪光 Comic Sans 横幅，以及所有脚本爱好者梦寐以求的 XSS 攻击技巧。这就像网络中的网络——一个开放且欢迎用户发布自己内容的环境，越来越多的人无需学习 HTML 便可发布内容，而且没有规则限制。平台数量激增，突然之间，每个人都找到了自己的归属，甚至为每一个可想象的小众兴趣都出现了专属的 phpBB 论坛。网络充满了活力，而互联网泡沫让硅谷财源滚滚。

然而，这种充满青春气息的天真烂漫、多彩缤纷的乐园并没有持续太久。因其开放的脚本策略而臭名昭著的 MySpace，最终因其不可靠的性质导致用户页面变得不稳定，致使整个平台几乎无法使用。当 Facebook 带着简洁、美观、稳定运行的界面出现时，人们意识到 MySpace 的时代结束了，用户成群结队地转向 Facebook。流行的互联网变得带有一种自命不凡的色彩，我们被引导至 Facebook 那简洁冷淡的白色“公司办公室”风格页面。然而，问题正在酝酿。尽管提供“免费”服务，但 Zuckerberg 及其他人背后有自己的目的。为了托管我们的数据，我们（the dumb f*cks; Carlson 2010）不得不将信任交给他们。显然，我们的确这样做了。在当时，平台表面上没有明确的商业模式，只是吸引更多的风险投资，积累庞大的用户群体，并采取“以后再处理问题”的态度。然而，从一开始，冗长而难以理解的用户条款就将所有内容版权授予了平台。在 Web 1.0 时代，人们可以轻松备份自己的网站并迁移至新的主机，甚至可以在家中自行托管，而在 Web 2.0 时代，那些持有争议性观点的人却面临了新的挑战：“被去平台化”（deplatforming）。

在基础设施层面，这种中心化随着超大规模数据中心的出现而变得显而易见。杰夫·贝索斯（Jeff Bezos）将他的图书销售业务转型为全球首富，方法是为那些在实施日益复杂和昂贵的基础设施时遇到技术和资金困难的人提供解决方案。这种新型的 Web 服务体系能够应对过去可能导致热门内容崩溃的流量激增。不久之后，互联网上的大部分内容都由少数几家大型公司托管。企业收购和源源不断的风险投资资金进一步加剧了权力的集中。一个被遗忘的联盟——开源程序员（他们创建了免版税的 Apache Web 服务器）和 Google（引入了重新定义数据组织与访问方式的范式）——对 Microsoft 企图将网络永远囚禁在 Internet Explorer 6 专有体系中的尝试给予了沉重一击。然而，Google 最终接受了“家长监督”，放弃了“不作恶”的承诺，屈服于自己的权力野心，开始吞并竞争对手。逐渐地，电子邮件变成了 Gmail，网络广告变成了 AdSense，而 Google 逐步渗透到人们在网络生活的方方面面。

表面上看，一切都很美好。技术乌托邦以前所未有的方式将世界超连接起来。网络不再只是学者和超级极客的专属领域——它让人类知识的总和对所有人都触手可及，并且随着智能手机的普及，人们可以随时随地访问这些知识。Wikipedia 让每个人都拥有超乎常人的知识储备，而 Google 则让我们瞬间找到并访问这些信息。Facebook 让我们可以与每一个认识过的人免费保持联系。然而，在这闪闪发光的表面之下，隐藏着一个问题。Google 知道他们在做什么，Amazon、Facebook 和 Microsoft 也同样心知肚明。而自 1984 年起，一些“朋克（punk）”也明白这一点。

当这些庞然大物式的平台积累了海量用户后，终于到了要向投资者兑现回报的时候。他们再也不能拖延，继续“摸索”商业模式了。为了向股东提供回报，这些平台将广告收入视为灵丹妙药。Google 和其他平台或许探讨过其他潜在的收入来源，但最终都未采纳任何重大替代方案。至此，网络开始变得复杂且令人分心。广告无处不在，那些闪烁着粉色亮片的横幅又回来了，这次它们将你的注意力从你真正想要的内容上转移，推送到下一个用户引流

的机会。

然而，这还远远不够，更多的恐怖接踵而至。当数据激增到难以掌控的程度时，网络失去了最后一丝天真，算法被用来“优化”我们获取内容的方式。现在，平台已经掌握了我们的所有数据，能够分析出我们想看到的内容，仿佛比我们自己更了解自己。然而，这里面隐藏着一个陷阱——这些无所不包的数据集和秘密算法可以出售给出价最高的人。资金充裕的政治组织能够以前所未有的精准度和效率锁定摇摆选民。网络空间突然变得过于真实，而关于“共识现实”的概念却成了过去式。新闻不仅仅是虚假的，它已进化为个性化的操控手段，经常引导我们做出违背自身利益的行为，而我们甚至毫无察觉。为了节省托管费用，每个人都成了可以轻易被操控的“提线木偶”。

与此同时，更令人不寒而栗的真相浮现。曾经支撑一个值得信赖的互联网建设的平等主义理念被证明是最为天真的想法。推动互联网早期发展的正是美国国防部（DoD），而如今他们试图重新夺回控制权。Edward Snowden 离开 NSA 时带走了一大堆机密文件，其内容之骇人听闻超出了所有人的想象（除非你一直认为《谍影重重》是部纪录片）。事实证明，这些协议早已被破坏，而“警告金丝雀（warrant canaries）”们早已死去。世界上的主要政府一直在对全球人口进行大规模监控——不断存储、处理、分类、索引每个人的在线活动，并可随时按需获取所有数据。只需按下 XKeyscore 系统（美国国家安全局（NSA）开发和使用的一款大规模监控和数据收集系统。）中的一个按钮，就可以调取这些数据——这是一只全知全能、永不闭眼的“光学神经”，秉持“收集一切”和“掌握一切”的信条，不论对象或背景为何。老大哥（Big Brother）的样貌最终看起来像极了索伦（英国作家 J.R.R. 托尔金（J.R.R. Tolkien）创作的奇幻史诗《魔戒》（The Lord of the Rings）和《精灵宝钻》（The Silmarillion）中的主要反派角色。）。这种对隐私的严重侵蚀，以及全球范围内权力中毒或妄想症缠身的国家和个人行为者对被压迫者、政治对手和记者进行追踪和审查的种种行为，共同促成了 Tor 项目的诞生。Tor 项目是 美国军方、麻省理工学院 和 电子前哨基金会（EFF） 之间的一次非凡合作，它提供了一种方式来混淆请求的来源，并以受保护、匿名的方式传输内容。尽管 Tor 项目在一些小众领域取得了巨大成功并广为人知，但由于其固有的不高效性导致相对较高的延迟，使其难以被主流用户采用。

到 Snowden（是一位美国前中央情报局（CIA）雇员和国家安全局（NSA）承包商，因揭露美国政府的大规模监控计划而闻名于世。） 爆料时，互联网已经变得无处不在，融入了人类生活的几乎每一个方面，其中绝大部分由大型企业运营。尽管可靠性问题已成为过去，但代价却不菲。内容创作者被提供了“上下文敏感、精准投放”的广告模式，这无异于与魔鬼签订了契约。这些提议伴随着一种心照不宣的微笑，透露出这些企业早已知道内容创作者别无选择。“我们会为你提供可扩展的托管服务，能够处理你所有受众带来的流量”，他们唱道，“但作为交换，你必须将内容的控制权交给我们。我们会追踪你每位受众的行为，并尽可能多地收集（并拥有，*whistle*）他们的个人数据。当然，我们会根据我们的权利，决定谁能看到内容，谁不能看到。此外，我们会主动审查这些内容，并在必要时与当局分享你的数据，以保护我们的业务。”因此，数百万小型内容创作者为少数巨型公司创造了巨大价值，而他们得到的回报少之又少——通常只是免费托管和一些广告收入。这真是“天赐良机”!

撇开我们在 Web 2.0 数据和新闻“末日景象”中所目睹的恐惧和不安，网络底层架构本身也存在一些技术问题。企业化的方法导致了“中心化至上”的格局，所有请求都必须通过某个主干网络，传到一个单体数据中心，再经过处理，最后返回给请求方——哪怕只是发送一条信息给隔壁房间的人。这种客户端-服务器架构的安全性极其脆弱，漏洞频发，以至于数据泄露已成为新常态，大量未加密的个人数据甚至明文密码散布在网络的各个角落。最后一根压倒骆驼的稻草，是这种架构催生出的大量不连贯的标准和接口。如今，随着复杂度的增加，网络被拆解成各种微服务模块，而这些模块往往是“意大利面条式”代码拼凑而成。

即便是资金雄厚的公司，也越来越难以承担这些开发费用，初创公司则常常因无止境的技术债务而溺死在“功能工厂”的海洋中。现代 Web 应用程序的技术栈简直就是一个戈德堡机械装置（是一种设计复杂、步骤繁琐的装置，用于完成一个极其简单的任务。），由无数相互依赖的组件拼凑而成，以至于即便是超国家级企业也很难维持和开发这些实现而不出现大量的漏洞和安全缺陷。不过，说实话，Google 和 Amazon 除外。无论如何，我们早该重启这一切了。归根结底，这些数据描述的是我们的生活。他们已经试图这么做了，但他们还没有能力将我们永久锁定在这个泥潭中。

1.1.3 点对点网络

当中心化的 Web 2.0 接管世界时，点对点（P2P）革命也在悄然并行发展，并迅速积聚力量。P2P 流量迅速占据了网络传输数据包的大部分，超越了前文提到的诱发 SYN 数据包的服务器。这证明了，通过共同协作利用那些过去严重闲置的上行带宽，终端用户可以实现与大公司及其连接互联网主干最粗管道的数据中心同等的内容可用性和吞吐量。更重要的是，这一切的成本仅为原有方案的一小部分。而且，用户对自己的数据拥有了更高的控制权和自由度。最终，这种数据分发模式即便在强大且资金雄厚的势力拼命封锁时，也表现出惊人的韧性。

然而，即便是发展到最成熟阶段的 P2P 文件共享模式——无追踪器的 BitTorrent（Pouwelse 等人，2005）——本质上仍然只是文件级的共享。这种模式根本不适用于提供 Web 2.0 应用程序用户所期望的那种交互式和高响应体验。此外，尽管 BitTorrent 极受欢迎，但其设计时并未考虑经济学或博弈论因素。这是那个时代的产物——在全球尚未意识到它将引发的革命之前，也就是在任何人理解区块链、加密货币和激励机制的力量之前。

1.1.4 BitTorrent 的经济学及其局限性

BitTorrent 的天才之处在于其巧妙的资源优化方法（Cohen，2003）：如果有多个客户端希望从某位用户处下载相同的内容，在第一阶段，该用户会将内容的不同部分分发给各个客户端。在第二阶段，这些客户端以“对等交换”的方式互相交换这些内容部分，直至每个客户端都获得完整内容。这样一来，托管内容的用户（在 BitTorrent 术语中称为“种子”）的上行带宽成本基本保持不变，无论同时下载该内容的客户端数量有多少。这解决了支撑万维网的超文本传输协议（HTTP）古老、中心化的“主从”设计中的最棘手问题。

为了防止作弊行为（例如向对等节点提供无效数据），BitTorrent 采用了分段哈希校验机制。每个供下载的包都由一个简短的哈希标识，任何部分都可以通过加密验证为该包的特定组成部分，而无需了解其他部分，并且只需极小的计算开销。

但这种看似简单优雅的方法也存在五个重要缺陷，这些缺陷之间有一定关联（见 Locher 等人，2006；Piatek 等人，2007）：

缺乏经济激励

用户在为他人提供内容种子以供下载时没有内置的激励机制。尤其是，缺乏一种机制将上传带宽与下载内容所需的带宽进行交换。实际上，为用户提供内容种子的上传带宽没有得到任何回报。由于尽可能多的上传带宽可以提升某些在线游戏的体验，因此关闭种子上传服务在理性上可能是一个自私但合理的选择。而一旦懒惰作祟，这种上传功能就可能被永久关闭。

初始延迟

下载通常会以较慢的速度开始，并且会有一些延迟。在下载进度较快的客户端相较新加入的客户端拥有更多的内容片段可提供，而新客户端在刚开始时无法为进度更快的节点提供任何可交换的内容。因此，BitTorrent 下载通常会以涓涓细流开始，随后才会变成真正的“数据洪流”。这一特性严重限制了 BitTorrent 在需要快速响应和高带宽的交互式应用程序中的使用，即便在其他场景下它是一个绝妙的解决方案。

缺乏细粒度的数据寻址

小数据块只能作为更大文件的一部分进行共享，而无法单独访问这些小块以优化内容获取。下载的对等节点只能通过查询分布式哈希表（DHT）来寻找目标文件，而无法在数据块级别查找节点。由于可用内容的广告仅在文件级别进行，这导致了效率低下，因为相同的数据块往往会在多个文件中原样出现。因此，尽管理论上所有拥有该数据块的对等节点都可以提供该数据块，但在缺乏封装文件的名称或哈希信息时，无法找到这些对等节点。

缺乏持续共享的激励

一旦节点从其他节点获取了所有所需的文件，它们便不会因为继续共享其资源（存储和带宽）而获得奖励。

缺乏隐私和模糊性

节点会公开广告其所提供的种子内容，这使得攻击者能够轻易发现托管目标内容的节点的 IP 地址。这样，任何对手都可以轻松发起 DDoS 攻击，而企业和国家则能够向互联网服务提供商（ISP）请求获取该连接的物理位置。这促使用户转向 VPN 服务以规避此类风险。尽管这些服务声称能够保障隐私，但由于系统通常是闭源的，因此难以验证其实际效果。

尽管 BitTorrent 非常流行且用途广泛，但它仅仅是一个基础的（尽管无疑是天才的）第一步。通过简单地共享我们的上传带宽、硬盘空间和少量计算资源，我们已经走得很远了——即便缺乏完善的记账和索引系统。然而，令人惊喜的是，通过引入几项新兴技术，尤其是区块链技术，我们能够实现真正配得上“Web 3.0”称号的系统：一个去中心化、抗审查的平台，用户可以在其中共享并集体创造内容，同时完全保有对内容的控制权。更令人欣喜的是，这一切的成本几乎完全可以通过共享我们现有资源来覆盖，而这些资源正是我们手中那台在过去标准看来几乎是“超级计算机”却仍然被大量闲置的个人设备提供的。

1.1.5 迈向 Web 3.0

“《泰晤士报》2009 年 1 月 3 日 财政大臣即将对银行实施第二轮救助”比特币创世区块（Genesis Block） 中包含的一条信息，出现在比特币区块链的第一个区块中。

2009 年 1 月 3 日早晨 6 点 15 分，一位神秘的密码朋克创造了一个足以环绕整个世界的链条的第一个区块，从根本上改变了我们对金钱的认知。加密货币的魔瓶被打开了。中本聪 实现了一件他人无法做到的事情——通过去中心化、无需信任的价值转移，实现了事实上的（尽管是小规模的）银行中介消解。从那一刻起，我们实际上回归了“金本位”——每个人现在都可以拥有“硬通货”，这种货币不会被任何人任意增发或稀释价值。更令人惊讶的是，我们甚至可以发行自己的货币，制定任意的货币政策，并拥有一个全球部署的电子传输系统。虽然我们尚未完全理解这将如何改变我们的经济体系，但这个系统已经吸引了前所未有的财富，从传统资产类别中抽离出来，使加密货币的总市值突破了惊人的 1 万亿美元。

这一步是一个划时代的转折点。系统内在核心就具备了身份验证和价值转移的功能。然而，尽管这一概念在理论上无比出色，但在实际应用上仍存在一些大大小小的问题。它确实允许数字价值的传输，人们甚至可以为代币“着色”或传递简短信息（例如记录创世区块那天的简短文本），但也仅限于此。而在扩展性方面……每一笔交易都必须存储在每一个节点上。分片 并非内置机制。更糟糕的是，为了保护数字资产的安全，每个节点必须始终与其他节点处理完全相同的数据。这与并行计算集群背道而驰，速度要慢上数百万倍。

当 Vitalik 构思 Ethereum（以太坊）时，他接受了这些局限性，但却让系统的实用性实现了巨大的飞跃。他通过 Ethereum Virtual Machine（EVM） 实现了图灵完备的计算能力，使得各种各样的应用可以在这个无需信任的环境中运行。这个概念既是一次惊人的范式转变，又是对比特币的合理进化。事实上，比特币本身就是基于一个小型虚拟机，每一笔交易其实都是一个微型程序——尽管许多人对此并不知情。然而，以太坊彻底释放了这一潜力，并再次改变了一切。无数令人向往的可能性出现了，Web 3.0 由此诞生。

然而，为了真正超越 Web 2.0 的世界，还有一个问题需要解决——在区块链上存储数据的成本极其高昂，仅适用于极少量的数据。比特币和以太坊都采用了类似 BitTorrent 的架构，并在此基础上加入了交易功能，但将非系统性数据的存储问题推迟到了以后解决。比特币引入了一个不太安全的“二级回路”：候选交易在区块发布之前被悄悄传播，仿佛是二等公民一样，甚至没有正式协议的保护。而以太坊更进一步，将区块头与区块主体分离，创建了一个第三层结构，用于根据需要传输实际的区块数据。由于这些数据类型对于系统的运行都至关重要，这些设计被认为是关键性的缺陷。比特币的创始人或许并未预料到挖矿会成为一项高度专业化精英垄断的行为。在他的设想中，每个交易者都应该能够自己“挖出”自己的交易。而以太坊的开发者们则面临更棘手的数据可用性问题，并可能假设这一问题可以在未来得到解决，因此暂时忽略了它。

另一方面，ZeroNet 成功将 BitTorrent 直接用于 Web 内容分发（ZeroNet 社区，2019）。然而，由于 BitTorrent 存在上述问题，ZeroNet 无法满足现代 Web 用户对高响应速度的需求。

为了实现响应式的分布式 Web 应用（即 dapps），星际文件系统（IPFS）（IPFS 2014）在 BitTorrent 的基础上进行了重大改进。其中一项突出特性是其高度兼容 Web 的、基于 URL 的数据检索方案。此外，可用数据的目录索引（与 BitTorrent 类似，采用了 DHT 结构）也得到了极大改进，使用户能够搜索任何文件的小部分数据，即所谓的“数据块”。

还有许多其他项目致力于填补这一空白，为 Web 2.0 开发者习惯的服务器和服务集群提供一个合格的 Web 3.0 替代方案，帮助开发者摆脱对中心化架构的依赖，这些架构正是“数据收割者”赖以生存的基础。这些角色的替代并非易事，因为即便是最简单的 Web 应用，也依赖于一系列复杂的概念和范式，而这些都需要重新映射到 Web 3.0 的无需信任环境中。从许多方面来看，这个问题甚至比在区块链上实现无需信任的计算还要复杂。Swarm 通过精心设计的数据结构，帮助应用程序开发者在 Web 3.0 中重新实现 Web 2.0 中熟悉的概念。Swarm 重新构想了当前的网络功能，并基于坚实的加密经济学基础重新实现了这些功能。

可以想象一个光谱，在最左端是大文件、低访问频率和单一的 API，在最右端是小数据包、高访问频率和细化的 API。在这个光谱上，像 POSIX 文件系统、S3、Storj 和 BitTorrent 这样的文件存储与检索系统位于左侧，而 LevelDB 这样的键值存储和 MongoDB、Postgres 这样的数据库则位于右侧。构建一个实用的应用程序需要跨越这一光谱的不同模式。此外，还需要能够在必要时合并数据，并确保只有授权方能够访问受保护的数据。在中心化模型中，初期处理这些问题相对容易，但随着系统的增长会变得越来越复杂，而光谱的每个区间都有专用的软件解决方案。然而，在去中心化模型中，这些方案往往无从实现。授权必须通过加

密手段进行，这限制了数据的合并。因此，在当今新兴且不断演进的 Web 3.0 技术栈中，许多解决方案只能零散地应对这一光谱的一部分需求。本书将展示 Swarm 如何跨越整个光谱，并为 Web 3.0 开发者提供高层次的工具。我们的愿景是，从基础设施的角度来看，Web 3.0 的开发体验能够重现 Web 1.0 那个充满自由的黄金时代，同时提供前所未有的自主权、可用性、安全性和隐私保护。

为了在文件共享的底层实现隐私保护——就像以太坊在其设计中实现的那样——Swarm 在根本和绝对的层面上实施匿名性。Web 2.0 的经验教会了我们，信任应该谨慎地给予那些值得信赖且会尊重这种信任的对象。数据是“有毒的”（Schneier 2019），我们必须小心处理数据，以对自己以及那些我们负有责任的人保持负责。稍后，我们将解释 Swarm 如何实现彻底且根本的用户隐私保护。

当然，要完全过渡到去中心化的 Web 3.0 世界，我们必须解决激励机制和信任问题，而这些问题在传统模式中是通过将责任交给（通常并不值得信赖的）中心化“守门人”来“解决”的。正如我们之前提到的，BitTorrent 也曾尝试解决这一问题，并通过种子比例和私人（即中心化的）追踪器做出回应。

在 ZeroNet 和 MaidSafe 等项目中，缺乏可靠托管和存储内容的激励机制问题非常明显。在区块链技术背景下，如何激励分布式文档存储仍然是一个相对较新的研究领域。Tor 网络曾提出激励方案的建议（Jansen 等人 2014；Ghosh 等人 2014），但这些大多是学术上的探索，尚未融入系统的核心机制中。Bitcoin 已被重新用于驱动其他系统，如 Permacoin（Miller 等人 2014），而 Sia（Vorick 和 Champine 2014）或 Filecoin（2014）等项目则为 IPFS 创建了自己的区块链。BitTorrent 目前正在通过自己的代币测试区块链激励机制（Tron Foundation 2019；BitTorrent Foundation 2019）。然而，即便这些方法全部结合起来，要满足 Web 3.0 去中心化应用程序开发者的特定需求，仍然面临着许多障碍。

稍后我们将探讨 Swarm 如何提供一整套激励措施，并实施其他制衡机制，确保节点为整个“swarm”（网络群体）的利益而协作。这其中包括一项功能，即允许用户将大量磁盘空间出租给愿意付费的用户——无论其内容是否受欢迎。同时，还支持在云端存储可交互的动态内容，我们称之为“上传即隐身”功能。

对等内容分发系统的任何激励机制的目标，都是鼓励合作行为并抑制“搭便车”现象——即不受补偿地消耗有限资源的行为。本文概述的激励策略旨在满足以下约束条件：

1 无论其他节点是否遵循该策略，该策略都符合节点自身的利益。

2 消耗其他节点资源的成本应当是高昂的。

3 不应引入不合理的系统开销。

4 能够与“naive”节点良好兼容。

5 奖励遵循该策略、行为良好的节点，包括那些友善协作的节点。

在 Swarm 的语境中，存储和带宽是最重要的稀缺资源，这一点也在我们的激励机制中得到了体现。带宽激励旨在实现快速且可靠的数据传输，而存储激励旨在确保数据的长期保存。这种全面的方法满足了所有 Web 应用开发的需求，并使激励机制与节点的行为保持一致，从而让每个节点的行动不仅有利于自身，也有利于整个网络。

1.2 公平数据经济

在 Web 3.0 时代，互联网已不再只是极客们玩耍的小众空间，而是成为了价值创造的重要渠道，并在整体经济活动中占据了巨大的比重。然而，目前的数据经济状态远谈不上公

平，因为数据的收益分配被那些掌握数据的人——主要是自建数据孤岛的公司——所掌控。要实现公平数据经济的目标，需要解决许多社会、法律和技术问题。接下来，我们将介绍当前存在的一些问题，并阐述 Swarm 如何应对这些挑战。

1.2.1 当前数据经济的现状

数字“镜像世界”已经存在——这些虚拟空间映射着现实世界中的事物，并由难以想象的海量数据组成（Economist 2020a）。随着越来越多的数据被同步到这些平行世界，新型基础设施、市场和商业机会应运而生。尽管目前对整体数据经济规模的衡量方法还相对粗略，但据估计，2019 年美国的数据经济（包括相关软件和知识产权）的总价值在 1.4 万亿至 2 万亿美元之间（Economist 2020a）。欧盟委员会预计，到 2025 年，EU27（欧盟 27 个成员国）的数据经济总值将达 8290 亿欧元（相比 2018 年的 3010 亿欧元大幅增长；European Commission 2020a）。

尽管数据经济创造了如此巨大的价值，但现有数据经济所产生的财富分配极不平衡，这已被视为一项重大的全球性人道主义问题（Economist 2020c）。数据质量和数量的不断提升，确实推动了效率和生产力的大幅提高，但由此产生的利润却分配不均。公司拥有的数据集越庞大，就能从中学习到越多的信息，吸引到更多用户，从而进一步积累更多数据。这种现象在目前由 FAANG（科技巨头公司）主导的市场中尤为明显，但预计在非科技领域乃至国家层面上也将愈发显著。因此，各大公司竞相在各自行业中取得主导地位，而这些平台所在的国家因此获得了更多优势。正如联合国贸易和发展会议所警告的那样，由于非洲和拉丁美洲本土几乎没有类似的平台，它们有可能沦为“原始数据”出口国，并为进口基于这些数据生成的智能成果支付高额费用（Economist 2020c）。

此外，当某家大公司垄断了特定的数据市场时，问题就更加严重了——这家公司可能成为唯一的数据购买者，从而完全控制价格制定。这种控制为操纵“数据报酬”提供了可能，使数据提供者的收益被人为压低。在很多方面，我们已经看到了这种现象的证据。

数据流动正日益受到各国政府的阻碍和过滤，其理由依然是出于保护公民安全、主权和国家经济的需要（Economist 2020b）。多位安全专家的爆料显示，为了切实考虑国家安全，政府认为数据应尽可能存储在本国境内，而非留在其他国家。例如，GDPR（通用数据保护条例）就是树立“数字边界”的一个典型案例——只有在具备适当的保护措施时，数据才可离开欧盟。印度、俄罗斯和中国等其他国家也实施了各自的数据地理限制。欧盟委员会承诺将密切关注这些国家的政策，并在贸易谈判中应对任何限制数据流动的行为。此外，欧盟委员会还在世界贸易组织（WTO）内采取必要措施（European Commission 2020b），以倡导公平和无障碍的数据交换实践。

尽管人们对数据流动的关注度不断上升，但大型科技公司依然牢牢掌控着大部分数据，而细节中隐藏着问题所在。Swarm 采用隐私优先的模型，确保不需要向任何第三方披露数据，并且所有数据默认端到端加密，从而防止服务提供商聚合和利用庞大的数据集。因此，与其将数据集中在服务提供商手中，不如将数据的控制权去中心化，回归到与数据相关的个人手中。与此同时，控制权意味着权力也随之而来。对此，请预期会有不利的舆论报道。

1.2.2 数据主权的现状与问题

由于上述“浮士德式的交易”，当前的万维网模式存在多种缺陷。由于基础设施规模经

济效应和社交媒体网络效应的意外结果，各种平台逐渐演变为庞大的数据孤岛，掌握着大量用户数据，而这些数据的保留、共享或删除完全取决于单一组织的意志。这种中心化数据模型的“副作用”使得大型私营公司能够通过其位于中心瓶颈位置的数据“虹吸管”——即一切交汇的云服务器——来收集、聚合和分析用户数据。这正是 David Chaum 在 1984 年所预言的情景，并由此点燃了“密码朋克”运动，这也是 Swarm 的重要灵感来源。

随着人与人之间的互动逐渐被计算机介导的互动取代，加之社交媒体和智能手机的普及，有关我们个人生活和社交生活的信息变得越来越容易被提供数据流服务的公司获取。这些公司能够进入极具价值的数据市场，在这些市场中，用户的人口统计数据与其行为模式相互关联，从而使这些公司比用户自己更了解他们。这是营销人员梦寐以求的“宝库”。

包括大型科技公司在内的数据公司，以及其他参与用户数据收集、聚合和分析的实体，已经不断演化其商业模式，如今他们更专注于通过数据销售获利，而非最初所提供的服务。他们的主要收入来源是向广告商、市场营销人员及其他希望“引导”公众行为的群体出售用户画像的分析结果。这一循环通过对用户平台上的反应进行细致监控并用来改进未来广告的精准投放，从而形成了一个无止境的反馈回路。在这一信息洪流中，一个全新的行业已经崛起，并涌现出一系列复杂的系统，用于预测、引导和影响用户，以捕获他们的注意力和金钱。这个行业公然且有意识地利用人类的认知偏差，常常采用经过精密计算、充满功利心的心理操控手段。不可否认的事实是，以商业利益为名的大规模操控，已经导致现代社会的现实——即便是最有意识的人，也难以真正行使自由选择的权利，并在内容、商品和服务的消费上保持其固有的自主偏好。

平台将重心从最初的主要目的转向广告业务，这一变化同样体现在用户服务质量的下降上。用户的需求已经被置于“真正的客户”——广告商的需求之下。在社交平台上，这种现象尤为严重，因为网络效应的惯性导致用户难以离开平台。这种激励机制的错位亟需被纠正，以便在不依赖中心化数据模型的情况下，为用户提供相同的服务，而不再伴随不良的激励后果。

此外，缺乏对自身数据的控制权对用户的经济潜力产生了严重影响。一些人将这种情况（尽管略显夸张地）称为“数据奴役”。但从技术角度来看，这种说法是正确的：我们的“数字化身”被这些公司所囚禁，并被用来创造收入。作为用户，我们放弃了大量的自主权，因为我们自愿分享的数据被用来操控我们，使我们变得更无知、更不自由。

将数据存储在彼此隔离的数据集中的现行系统存在多种弊端：

1 机会不平等

中心化实体从实际创造价值的用户手中吸走了不成比例的利润，从而加剧了不平等。

2 缺乏容错性

这些数据集在技术基础设施上存在单点故障，尤其是在安全性方面。

3 腐败风险

决策权的集中使这些数据集更容易成为社会工程攻击、政治压力和体制化腐败的目标。

4 单一攻击目标

将大量数据集中在同一个安全系统中，会因潜在收益巨大而吸引黑客进行攻击。

5 缺乏服务连续性保障

服务的连续性掌握在平台组织手中，而声誉激励机制对其作用有限。这导致服务因破产或监管/法律行动而被意外终止的风险增加。

6 审查

对数据访问的中心化控制允许并最终会导致言论自由的受限。

7 监控

流经中心化基础设施的数据，为流量分析和其他监控手段提供了完美的切入点。

8 操控

对数据展示层的垄断，使得数据收集方可以通过控制数据的呈现方式、顺序和发布时间来操控舆论，从而动摇个体决策权的主权。

1.2.3 迈向数据自主权

我们相信，去中心化是一个改变游戏规则的因素，能够有效解决上述列出的许多问题。

我们认为，区块链技术是实现真正自主互联网的赛博朋克理想的最后一块拼图。正如埃里克·休斯（Eric Hughes）在 1993 年的《赛博朋克宣言》（Hughes 1993）中所概述的那样，“我们必须团结起来，创建允许匿名交易的系统。”本书的目标之一是展示如何将去中心化共识和对等网络技术结合起来，形成一个坚如磐石的基础层基础设施。这一基础不仅具有弹性、容错性和可扩展性，而且在设计良好的激励机制下，还具有平等性和经济可持续性。参与者的低门槛确保了适应性激励，从而使价格自动趋近于边际成本。在此基础上，Swarm 在隐私和安全领域提供了强大的价值主张。

Swarm 是一个去中心化、激励化和安全的 Web 3.0 技术栈。特别是，该平台为参与者提供了数据存储、传输、访问和认证的全面解决方案——这些服务在经济互动中变得越来越重要。Swarm 通过提供具有强大隐私保障的普遍访问，并且不受边界或外部限制的影响，培养了全球志愿主义精神，并成为自主数字社会的基础设施。

1.2.4 人工智能与自我主权数据

人工智能（AI）对社会有着巨大的发展潜力，它为我们带来了新的商业机会，并增强了各行各业所使用的工具集。但另一方面，人工智能也可能带来威胁，尤其是在某些职业和工作岗位被取代的情况下（Lee 2018）。

对于目前主流的人工智能类型——机器学习（ML），尤其是深度学习，需要三个核心“要素”：计算能力、模型和数据。如今，计算能力已经触手可及，并且还在开发专用硬件来进一步提高处理效率。在过去十多年里，围绕 AI 人才的争夺一直激烈进行，而少数公司已经垄断了拥有构建模型和分析数据所需专业技能的工作市场。然而，当今 AI 和深度学习领域的“秘密”在于，算法和所谓的“智能数学”实际上已经被普及化，成为开放资源，任何人都可以自由获取。这并不是 Google 或 Palantir 赚钱的关键。真正的“魔术”在于这些公司能够获取尽可能庞大的数据集，以充分释放其 AI 系统的潜力，获得巨大的利润回报。

数据经济中的主要参与者正系统性地积累大量数据。他们通常通过搜索引擎或社交媒体平台等免费应用提供某些功能服务，同时在未经用户明确同意或知情的情况下收集并囤积用户数据。这种数据垄断使跨国公司获得了前所未有的利润，但仅对贡献数据的用户做出微不足道的收益分享。更糟糕的是，这些囤积的数据没有得到充分利用，剥夺了个人和整个社会从数据中获取变革性价值的机会。

或许并非巧合，目前在数据和人工智能领域的“超级大国”正是美国和中国及其境内的大型企业。一场人工智能的“军备竞赛”正公开上演，而世界大部分国家却落后于此，成为所谓的“数据殖民地”（Harari 2020）。有警告指出，按目前的趋势发展，中国和美国将在人工智能领域积累无法超越的优势（Lee 2018）。

但情况不一定非得如此。事实上，这种现状对数十亿人来说是一场不公平的交易。去中心化技术和加密技术为数据隐私保护提供了一条新路径，同时培育出一个公平的数据经济，

该经济保留了当前中心化系统的优势，但摒弃了其有害的弊端。全球众多消费者组织和技术公司正努力实现这一目标。他们致力于推动对抗数据巨头的行动，因为越来越多的用户开始意识到，他们的数据被“巧取豪夺”。Swarm 的基础设施将在这场解放运动中发挥关键作用。

自我主权存储 可能是用户重新掌控自己数据和隐私的唯一途径。它是打破“信息茧房”、重新连接用户与自身文化的第一步。为了解决当今互联网及数据分发和存储所面临的种种挑战，Swarm 从底层架构出发，就内置了强大的数据加密机制，并确保通信的安全性和完全无泄漏。此外，它允许用户根据自己的意愿选择性地向第三方共享特定数据，并为此提供获得经济补偿的可能性。因此，支付和激励机制也是 Swarm 不可或缺的一部分。

正如 Hughes 所写道：“在一个开放的社会中，隐私需要匿名交易系统……匿名交易系统并不是秘密交易系统。匿名系统赋予个人在希望时揭示身份的能力，而只有在他们希望时才揭示身份；这就是隐私的本质。”

使用 Swarm 可以利用更全面的数据集来创建更优质的服务，同时仍然可以通过可自证的匿名化方式为全球公益做出贡献。这是一种集多种优势于一体的解决方案。

这种新获得的数据可用性，将为年轻的学术研究者和在人工智能与大数据领域有颠覆性创意的初创公司带来巨大的推动力，极大地促进整个领域的发展。这对于科学进步、医疗健康、消除贫困、环境保护、灾害预防等方面充满了希望。然而，尽管有一些令人瞩目的成功吸引了“强权掠夺者”和不良国家势力，许多子领域目前仍面临挑战和停滞不前的局面。Swarm 的数据解放能力有望打破这一僵局，释放该领域在各个领域的潜在贡献。

Swarm 提供的功能将为公司和服务提供商开启一系列强大的新选项。随着数据的广泛去中心化，我们可以共同拥有构建最先进 AI 模型所需的超大且极具价值的数据集。拥抱数据可携性——这一趋势在传统科技领域已初见端倪——将促进竞争，并为个体提供更个性化的服务。公平的竞争环境将为所有人铺平道路，推动与 21 世纪第三个十年的需求相匹配的创新发展。

1.2.5 集体信息

自互联网诞生以来，集体信息便在不断积累，但这一概念直到最近才受到关注，并在“开源”、“公平数据”或“信息公地”等各种话题下被讨论。

根据 Wikipedia 的定义，集体是：

“一群实体，它们因至少一个共同的问题或兴趣而联系在一起，或为实现一个共同目标而共同努力。”

互联网使集体的形成达到了过去无法想象的规模，超越了地理位置、政治信仰、社会地位、财富，甚至一般意义上的自由及其他人口统计学属性。这些集体通过在公共论坛、评论区、投票、代码仓库、文章和民意调查等平台上的互动所产生的数据，以及由此衍生的元数据，都构成了集体信息。由于目前大多数互动由运行中心化服务器的盈利性实体提供服务，这些集体信息最终存储在由商业实体拥有的数据孤岛中，往往集中在少数几家大型科技公司的手中。尽管这些互动成果本身通常是公开的，但其中的元数据往往更加有价值、更加强大，也更加具有潜在危险性，却通常在保密状态下被收集和货币化。

这些“平台经济”已经成为数字化社会中不可或缺的组成部分，而且其重要性与日俱增。然而，这些商业公司获取的用户信息正越来越多地被用来对抗用户的最佳利益。简单来说，这引发了一个疑问：这些公司是否能够承担管理我们集体信息所带来的伦理责任。

尽管许多国家行为体都试图获得对集体个人数据的无障碍访问权，一些国家甚至要求设

置“万能密钥”式的后门访问权限，但也有一些例外。

由于人工智能存在被滥用及道德争议的风险，一些国家已经启动了“道德规范”计划、法规和人工智能使用认证机制，例如德国的数据伦理委员会或丹麦的数据伦理认证标志。

然而，即便能够迫使企业变得更加值得信任，以符合其巨大责任的要求，仅仅是数据孤岛的存在，就足以扼杀创新。客户端-服务器架构的基本形态本身导致了这一问题，因为这种架构默认将数据集中存储在“服务器农场”内的“服务器”上（见 1.1.1 和 1.1.2）。相比之下，像 Swarm（见 1.1.3）这样的高效点对点网络有可能改变这种架构的拓扑结构，实现集体信息的集体所有权。

1.3 愿景

Swarm 是为自我主权社会提供的基础设施。

1.3.1 价值观

自我主权意味着自由。如果我们将其拆解，可以得出以下元价值观：

1 包容性

公众参与和无需许可的参与方式

2 完整性

隐私保护和可验证的来源证明

3 激励机制

节点与网络利益的一致性

4 公正性

内容和价值的中立性

这些元价值观可以被视为系统性的品质，有助于赋能个人和集体实现自我主权。

包容性 意味着我们致力于将弱势群体纳入数据经济，并降低定义复杂数据流和构建去中心化应用程序的门槛。Swarm 是一个开放参与的网络，允许用户自由提供服务，并无需许可即可发布、共享和投资自己的数据。

在自由表达其意图（即“行动”）并保留对匿名或分享其互动和偏好的自主决定权时，用户也必须维护其线上身份的完整性。

经济激励 确保参与者的行为与网络期望的涌现行为保持一致（见第 3 章）。

公正性 保证了内容的中立性，并防止出现“守门人”现象。同时，它重申了其他三个价值观不仅是必要的，也是充分的：它消除了可能将某一特定群体视为特权群体或偏向某一特定来源的内容或数据的价值观。

1.3.2 设计原则

信息社会及其数据经济带来了一个在线交易和大数据不可或缺于日常生活的时代，使支撑这些活动的技术基础设施成为功能性社会的重要组成部分。因此，这一底层基础设施必须具备面向未来的能力，并提供长期持续性的强有力保障。

技术基础设施的持久性通过以下系统属性所表达的一般性要求来实现：

1 稳定性

规范和软件实现具有稳定性，能够抵御参与度变化或政治因素（政治压力、审查等）的影响。

2 可扩展性

解决方案能够在用户数量和数据量级别大幅增长的情况下保持扩展能力，而不会在大规模采用时对性能或可靠性造成不利影响。

3 安全性

网络能够抵御恶意攻击，不受社会压力和政治影响的干扰，并在其技术依赖（如区块链、编程语言）方面表现出容错能力。

4 自我维持性

该解决方案作为一个自主系统运行，不依赖于个人或组织对集体行动的协调，也不依赖任何法律实体的商业运作、专有技术、硬件或网络基础设施。

1.3.3 目标

当我们谈论“数据流动”时，其中的核心要素是信息在不同模式之间的完整性可验证性，详见表 1.1。这与 Ethereum（以太坊）最初提出的“世界计算机”愿景相对应，即构建一个无需信任（即完全可信）的数据生态基础架构，支持数据的存储、传输和处理。

dimension	model	project area
time	memory	storage
space	messaging	communication
symbolic	manipulation	processing

表 1.1：Swarm 在三个维度上的范围和数据完整性方面。

如果将以太坊区块链视为“世界计算机”的 CPU，那么 Swarm 则可视为其“硬盘”。当然，这一比喻简化了 Swarm 的复杂特性，因为 Swarm 的功能远不止简单的存储，正如我们即将讨论的那样。

Swarm 项目的目标是实现这一愿景，构建世界计算机的存储和通信系统。

1.3.4 影响领域

以下是我们所识别的功能领域，这些领域最能体现或促进上述包容性、完整性、激励机制和公正性四个价值观。

在无需许可的参与方面，去中心化的点对点网络是实现包容性的最佳保障。

允许节点提供服务并因此获得报酬，可以为用户提供零现金加入生态系统的途径：新用户在没有货币的情况下，可以通过为其他节点提供服务累积货币，直至他们可以使用服务。同样，一个去中心化且无“守门人”的分布式存储网络也具有包容性和公正性，因为它允许内容创作者在不被压制性权力剥夺言论自由的情况下发布内容，从而规避“被去平台化”的风险。

协议中内置的经济激励系统在对等交互的上下文中最好能跟踪那些会产生成本的行为。例如，在消息中继过程中体现的带宽共享就是其中一种行为，当节点接收到对其有价值的信

息时，可以立即进行记账。然而，诸如承诺长期保存数据的服务等预期性服务，必须在验证后才能给予奖励。为了避免“公地悲剧”问题，这类承诺性服务应通过惩罚性措施确保个人责任，例如通过质押保险机制来实现这一点。

完整性通过确保真实性易于验证而同时保持匿名性来维护。可验证的包含性和唯一性是实现无需信任的数据转换的基础。

1.3.5 未来

在当今数字化社会中，人类面临许多挑战，使未来充满不确定性。然而，为了实现主权并掌控我们的命运，无论是国家还是个人，都必须保有对自身数据和通信的访问权和控制权。

Swarm 的愿景和目标植根于去中心化技术社区的价值观。最初，Swarm 被构想为与 Ethereum（以太坊）和 Whisper 共同构建“世界计算机”的文件存储组件。如今，Swarm 继续秉承这一使命，致力于构建一个具有强大韧性的去中心化数字生态系统。

Swarm 为运行在用户设备上的去中心化应用程序（dapps）提供必要的响应能力，同时利用从智能手机到高可用集群等各种存储基础设施，实现激励化的存储服务。通过精心设计的带宽和存储激励机制，可以确保服务的持续性。

内容创作者将因其提供的内容获得公平的报酬，而内容消费者则需为其消费的内容付费。通过消除目前受益于网络效应的中间服务商，收益将分布到整个网络中。

但这远不止如此。每个个体和每个设备都会留下数据痕迹，而这些数据被收集并存储在“数据孤岛”中，其潜力仅被部分挖掘，且主要为大型公司所用。

Swarm 将成为“数字镜像世界”的首选平台，为个人、社会和国家提供独立于任何大型提供商的云存储解决方案。

个人将完全掌控自己的数据，不再受制于当前“数据奴役”体系，即在不透明且剥削性的服务平台上以个人数据换取服务。此外，用户还能够组建“数据集体”或“数据合作社”，通过数据共享资源（数据公地）来实现共同目标。

各国将建立自我主权的 Swarm 云作为数据空间，以满足新兴的人工智能范式在工业、医疗、交通及其他领域的需求。这些数据云将以点对点方式运行，可能仅限于特定区域，并且第三方无法通过监控、审查或操控数据流进行干预。然而，授权方可以访问这些数据，以促进人工智能及其相关服务的公平竞争环境。

从某种意义上说，Swarm 可以成为存储数据的“中心”平台。拥抱这一技术将为个人和社会带来对数据的强大访问能力、控制权和公平的价值分配，从而实现数据在集体利益中的最大化利用。

在未来的社会中，Swarm 将无处不在，以透明且安全的方式，将数据从个人和设备传递给公平数据经济中的数据消费者。

第二部分
设计与架构

Swarm 项目旨在构建一个无需许可的存储和通信基础设施，服务于未来自我主权的数字社会。从开发者的角度来看，Swarm 可以被视为支持实时交互式 Web 应用程序的公共基础设施，这些应用程序类似于 Web 2.0 时代的应用。它提供了面向低级别操作的 API，这些操作构成了复杂应用程序的构建模块，并为基于 Swarm 的 Web 3.0 开发栈提供了工具和库的基础。为了实现从传统 Web 浏览器访问的便捷性，API 和工具被设计为确保 Swarm 能够快速提供当今万维网（WWW）的私密、去中心化替代方案。

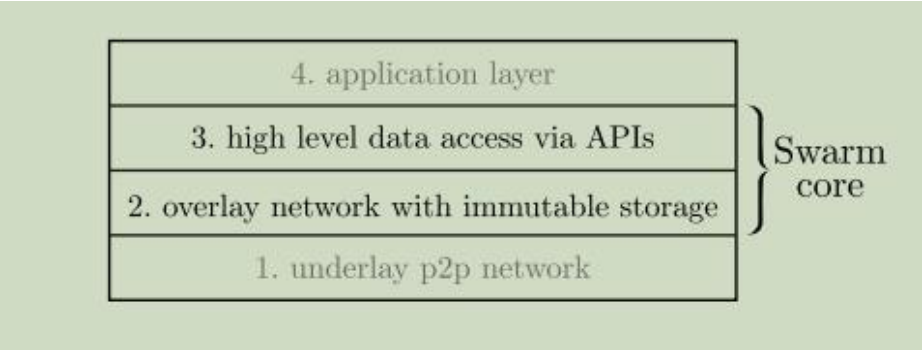

图 1.1 Swarm 的分层设计

本部分详细介绍系统的设计与架构。根据 1.3.2 节中列出的原则，我们为 Swarm 设计了模块化的架构，并将其划分为彼此依赖的独立层（见图 1.1）：

1 用作基础传输的点对点网络协议。

2 支持分布式不可变数据块（固定大小数据块）存储的覆盖网络协议。

3 提供高级数据访问并定义底层功能 API 的组件。

4 定义标准并提供复杂用例最佳实践的应用层。

我们将第 2 层和第 3 层视为 Swarm 的核心。由于网络层依赖这些核心内容，我们还将提出第 1 层的相关要求，但对于第 1 层和第 4 层的详细讨论超出了本书的范围。

Swarm 设计的核心是其覆盖网络架构（图 1.1 中的第 2 层），第 2 章对此进行了讨论。第 3 章补充描述了 Swarm 系统中的经济激励机制，该机制使 Swarm 实现自我维持。在第 4 章中，我们介绍了 Swarm 将数据概念映射到数据块层所需的算法和约定，以支持存储和通信的高级功能，这些功能包括文件系统、数据库、访问控制、索引数据流和直接消息传递，它们共同构成了 Swarm 的第 3 层。在第 5 章中，我们提出了一些方法，以防止被垃圾回收的数据块从网络中消失，这些方法包括纠删码、固定和保险机制，并提供了监控、恢复和重新上传数据块的方法，如缺失数据块通知和保险质询。最后，在第 6 章中，我们将从开发者的角度介绍基于 Swarm 构建应用程序时的相关功能。

第 2 章 网络

本章阐述了 Swarm 覆盖网络如何基于点对点网络协议构建拓扑结构，从而实现节点间消息路由（2.1）。在 2.2 节中，我们描述了这种网络如何为数据块提供可扩展的分布式存储解决方案（2.2.1），并展示了支撑检索/下载和同步/上传协议的逻辑（2.3）。

2.1 拓扑与路由

本节通过明确底层网络的假设为 Swarm 的覆盖网络奠定基础（2.1.1）。2.1.2 介绍了覆盖地址空间，并解释了节点如何被分配地址。在 2.1.3 节中，我们展示了 Kademlia 覆盖拓扑（连接模式），并说明其如何解决节点之间的路由问题。2.1.4 描述了运行 Swarm 客户端的节点如何发现彼此、启动引导并维持覆盖拓扑。

2.1.1 底层网络的要求

Swarm 是由其用户运营的网络。网络中的每个节点都需运行符合协议规范的客户端。在最底层，节点通过点对点网络协议作为传输层进行连接，这被称为底层网络。Swarm 的总体设计对所使用的具体底层传输协议保持中立，只要满足以下要求即可：

1 地址定位

节点通过其底层地址进行标识。

2 拨号

节点可以通过拨打底层地址来发起与对等节点的直接连接。

3 监听

节点能够监听其他节点拨打它们的请求，并接受传入的连接请求。不接受传入连接的节点称为“轻节点”。

4 实时连接

节点连接建立后即创建通信通道，表示远程节点在线并接受消息，直到明确断开连接。

5 通道安全性

通道提供身份验证，并实现加密和身份认证的传输，防止中间人攻击。

6 协议多路复用

底层网络服务能够在同一连接上运行多个协议。对等节点通过协议名称和版本来通信，底层服务识别兼容协议并在匹配协议上启动对等连接。

7 传输保证

协议消息必须保证可传输性，即因网络问题导致传输失败时应直接返回错误响应。消息在每个协议内的传输顺序需得到保证，理想情况下，底层协议应提供优先级机制。如果在同一传输通道上进行协议多路复用，应实现分帧机制，以防长消息阻塞高优先级消息。

8 序列化

协议消息构造需支持任意数据结构序列化约定。

libp2p 库可以提供所有必要的功能，是规范中指定的底层连接驱动。

2.1.2 覆盖地址

虽然客户端使用底层地址与对等节点建立连接，但每个运行 Swarm 的节点还通过一个覆盖地址进行标识。正是这个覆盖地址决定了节点将与哪些对等节点建立连接，并指引消息的转发路径。覆盖地址被视为稳定的，因为它在会话之间定义了节点的身份，并最终影响节点本地存储中优先存储的内容。

节点的覆盖地址是通过将对应的椭圆曲线公钥与 bzz 网络 ID 一起使用 256 位 Keccak 哈希算法进行哈希运算而得出的。引入 bzz 网络 ID 是因为可能存在多个 Swarm 网络（例如测试网、主网或私有 Swarm 网络），该 ID 确保相同地址不会在不同网络间重复使用。在假设任意基本账户样本是独立选择的情况下，生成的覆盖地址预计会在 256 位整数地址空间中呈均匀分布。从公钥导出地址非常重要，因为这允许节点使用可被第三方验证的加密签名，发出与覆盖地址位置相关的承诺。

通过底层网络的长生命周期通信通道，Swarm 节点形成了具有准永久对等连接的网络。

生成的连接图可以实现定义在地址空间上的特定拓扑结构。所选的覆盖网络拓扑结构被称为 Kademlia，它通过仅使用底层对等连接的消息中继策略，实现 Swarm 网络中任意两个节点之间的通信。描述节点如何共享自身及其他对等节点信息的协议称为“hive”。节点如何使用该协议引导覆盖拓扑将在 2.1.4 节中讨论。

覆盖地址空间必须涵盖 256 位整数的完整范围，这是至关重要的。在 Swarm 中的一个核心概念是接近度阶（PO），用于在离散尺度上量化两个地址之间的相关性。给定两个地址 x 和 y，PO(x, y) 统计其二进制表示从最高有效位开始到第一个不同位为止的匹配位数。因此，最高的接近度阶为 256，表示最大相关性，即当 x = y 时的情况。

2.1.3 Kademlia 路由

Kademlia 拓扑可以利用覆盖地址在网络节点之间路由消息。它具有极佳的可扩展性，因为它实现了通用路由，使得 (1) 跳数和 (2) 对等连接数始终与网络规模成对数关系。

下面我们将展示两种常见的路由方式：迭代/缩放式 和 递归/转发式。Swarm 的设计关

键依赖于后者，即转发式路由，这使它不同于点对点文献中更常见的迭代式路由方法，该方法也被大多数其他实现所采用（参见 Maymounkov and Mazieres 2002、Baumgart and Mies 2007、Lua et al. 2005）。为了全面理解这两种方式，我们将逐步讲解它们的独特之处。

迭代式与转发式 Kademlia

令 R 为网络中节点之间的任意二元关系。与节点 x 存在关系 R 的节点称为 x 的对等节点。x 的对等节点可以根据相对于 x 的接近度阶（PO）进行索引。这些对等节点的等价类称为接近度阶桶，简称桶。当这些对等节点被安排在桶中时，这些分组构成了节点 x 的 Kademlia 表（见图 2.1）。

如果节点 x 的 Kademlia 表满足以下条件，则称其为饱和的 Kademlia 表：存在一个 $0 \leqslant dx \leqslant maxPO$ 的值，称为邻域深度，并满足 (1) 节点在从 0 到但不包括 dx 的每个接近度阶桶中至少有一个对等节点，以及 (2) 所有接近度阶至少为 dx（称为最近邻节点）的节点都是 x 的对等节点。如果网络中的每个节点都具有饱和的 Kademlia 表，则称该网络具有 Kademlia 拓扑结构。

令 R 为“已知”关系：如果 x 同时拥有 y 的覆盖地址和底层地址信息，则称 y“为 x 所知”。在迭代式 Kademlia 路由中，请求节点通过迭代方式扩展其已知对等节点的图。请求节点利用其底层地址与已知的最接近目标地址的对等节点进行联系，以进一步查询更接近目标地址的对等节点（通常使用 UDP）。在每次迭代中，对等节点与目标地址的距离至少减少一个阶（见图 2.3）。由于 Kademlia 路由准则，请求节点最终会发现目标节点的底层地址，并能够与其建立直接通信。此迭代策略 3 关键依赖于节点发现当前在线对等节点的能力。为了找到这样的对等节点，节点需要为每个桶收集多个候选节点。对等节点可用性的最佳预测因素是该节点最近一次响应的时间，因此应根据响应的最新时间对桶内的对等节点进行优先排序。

Swarm 采用了 Heep (2010) 首次描述的另一种 Kademlia 路由方式，并由 Trón et al. 进行了扩展和完善。

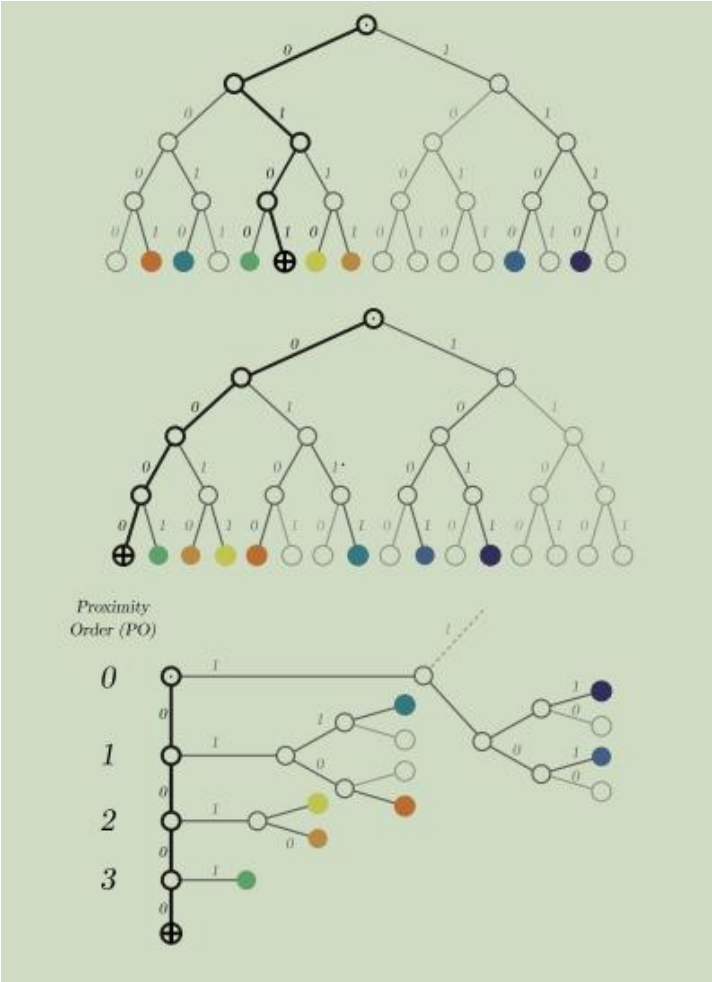

图 2.1：从覆盖地址空间到 Kademlia 表。顶部：覆盖地址空间表示为二叉树，带颜色的叶节点为实际节点。枢轴节点（+）的路径用加粗线条表示。中部：枢轴节点的对等节点按其与枢轴节点 xor 距离的位进行标注。这里，0 代表与枢轴节点匹配的位，1 代表不同的位。叶节点按其与枢轴节点的 xor 距离从左到右排列。底部：枢轴节点的 Kademlia 表：枢轴路径左侧的子树分支被显示为表格中的行，表示接近度阶桶，按接近度阶递增排列。

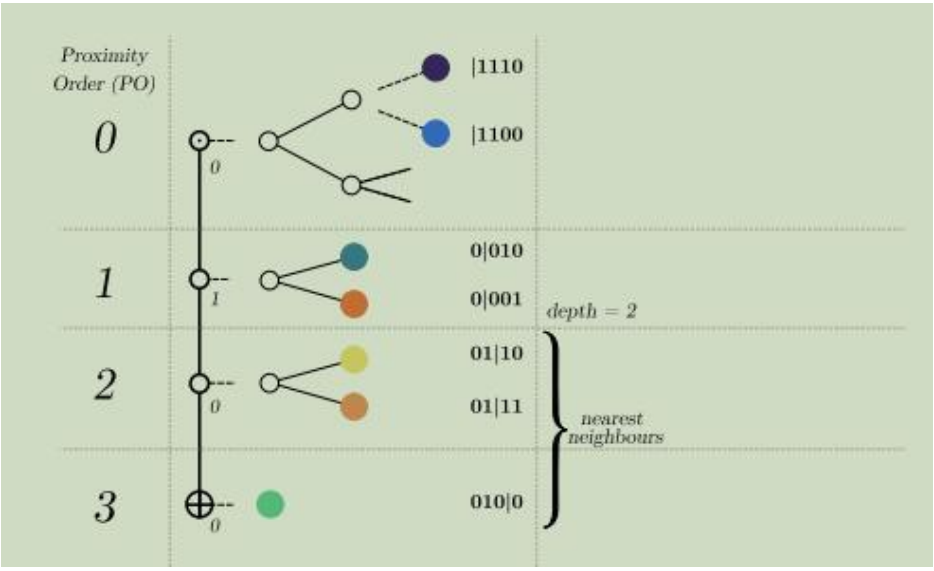

图 2.2：4 bit 网络中 d = 2 的最近邻节点

在这种递归方法中，迭代过程的每一步都被“外包”给下游的对等节点。每个节点将消息递归地传递给至少比目标地址接近度阶（PO）更高一级的直接对等节点。因此，使用这种路由方法意味着通过一系列逐渐更接近目标地址的对等节点链中继消息，如图 2.3 所示。

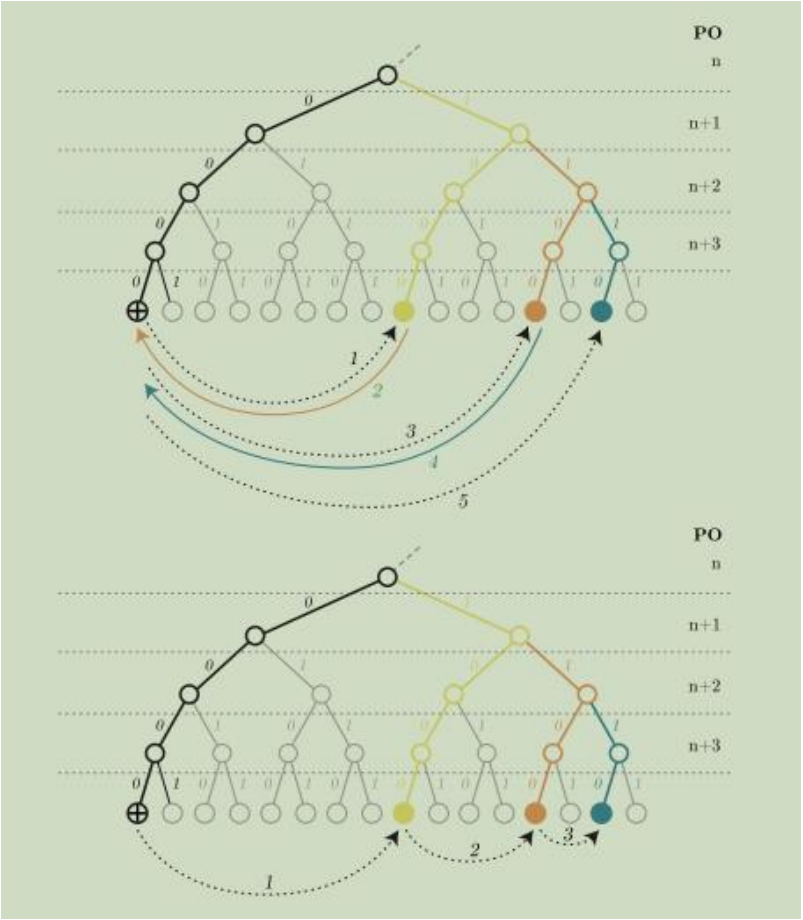

图 2.3：迭代式和转发式 Kademlia 路由在图中，请求节点用圆圈中的“十字”表示，其地址为 ...0000...，它希望将消息路由到目标地址 ...1111...，此目标地址的最近在线对等节点是蓝色圆圈表示的节点（地址 ...1110...）。这些初始省略号表示请求节点和目标地址共享的前缀长度为 n 位。顶部：在迭代式路由中，请求节点联系它已知的最接近目标地址的对等节点（步骤 1，用黑色虚线箭头表示）。在线的对等节点（黄色）响应有关更接近节点的信息（绿色箭头，步骤 2），这样请求节点可以使用这些更接近的对等节点再次进行查询（绿色，步骤 3）。在每次迭代中，对等节点（黄色、绿色和蓝色）至少比上一跳更接近目标地址一个 PO，直到请求节点直接与最接近目标地址的节点建立联系。底部：在转发式路由中，请求节点将消息转发给它已知的最接近目标地址的已连接对等节点（黄色）。接收节点同样如此。应用这种策略，消息通过对等节点链（黄色、绿色、蓝色）递归中继，每一跳都比上一跳更接近目标地址至少一个 PO。

在转发式 Kademlia 网络中，如果从发送节点到目标地址存在一条可以中继消息的路径，则称该消息是可路由的。在具有成熟 Kademlia 拓扑的子网络中，每条消息都是可路由的。如果所有对等连接都保持稳定在线，那么一个“精简的 Kademlia 表”（即每个桶最多包含一个对等节点）就足以保证节点之间的路由。然而，在实际中，网络中节点的状态是变化的，即节点会定期离线。为了在这种节点波动（churn）情况下确保可路由性，网络需要维持 Kademlia 拓扑。这意味着每个节点在任何时候都需要保持其 Kademlia 表的饱和状态。

通过在每个接近度阶桶中保持多个已连接的对等节点，节点可以确保节点掉线不会破坏

其 Kademlia 表的饱和状态。根据节点掉线的模型，我们可以计算出每个桶中需要的最小对等节点数量，以保证节点的饱和概率接近 1。节点在某个接近度阶桶中保留的对等节点越多，该节点与消息目标地址共享的前缀长度越长。因此，将消息转发给这些节点时，接近度阶会更快增加，消息比每个桶中拥有更少节点的情况更接近目标地址（参见图 2.4）。

在保证 Kademlia 饱和状态的前提下，节点将始终能够转发消息并确保可路由性。如果节点遵守转发原则（这一点可通过激励机制的对齐来保证），唯一可能导致中继失败的情况是节点在接收到消息后但尚未成功转发时掉线。

转发式 Kademlia 的一个重要优势是，这种路由方法比迭代算法需要的带宽要少得多。在迭代版本中，已知的对等节点不一定在线，因此找到可用节点会增加不确定性。

发送者匿名性

发送者匿名性是 Swarm 的一项关键功能。在请求消息从对等节点间逐跳中继时，重要的是下游节点永远无法知道请求的发起者是谁。

上述 Kademlia 路由的严格定义表明，如果节点从对等节点接收到一条消息，而该消息和对等节点的接近度阶（PO）为 0，那么接收节点可以推断出发送此消息的对等节点就是消息的发送者。然而，如果允许轻节点 Swarm 客户端（即由于资源限制而不保持完整的 Kademlia 饱和度，仅维护本地邻域连接的客户端），那么即使来自 PO 0 桶中的对等节点的消息，其来源仍然是模糊不清的。

桶密度与多阶跳跃

由于对数距离和节点分布均匀，随着距离增加，对等节点的数量呈指数增长。这意味着，如果桶中的连接数量不随距离的增加而成倍增长，那么越浅的桶总是有更多可供选择的节点用于潜在连接。特别是，节点可以增加每个桶中的连接数量，以使对等节点地址尽可能密集（即在接近度阶桶 b 中，对等节点地址的后续位组成平衡二叉树）。这种排列是最佳的，因为对于深度为 d 的桶，节点能够中继所有消息，使得目标地址的接近度阶在一次跳跃中增加 d（见图 2.4）。

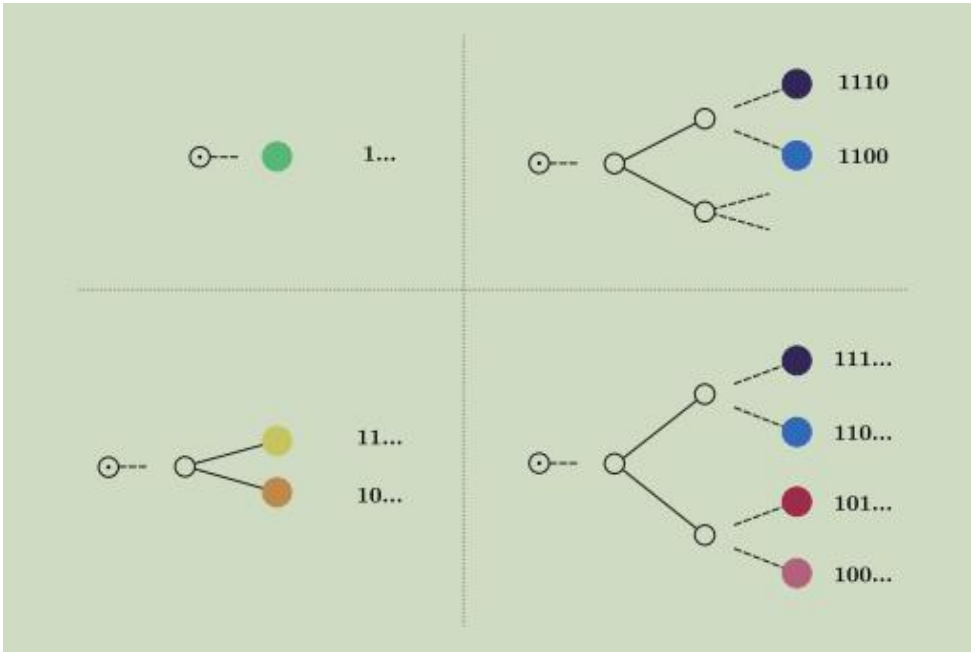

图 2.4：Bin 密度：对于覆盖地址以比特 0 开头的节点，PO bin 0 的饱和类型。左上角：一个“稀疏”的 bin 只有一个对等节点，无法应对节点流失，并且在一跳中仅增加 1 个 PO。右上角：至少需要两个对等节点才能在节点流失的情况下维持 Kademlia 拓扑；当不平衡时，两个对等节点无法保证多跳顺序。左下角：两个平衡的对等节点可以保证在一跳中增加 2

个 PO。右下角：四个平衡的对等节点可以保证在一跳中增加 3 个 PO。

考虑底层网络的接近性

随着 Swarm 客户端的不断发展，节点在选择对等节点时可能会考虑吞吐量。在其他条件相同的情况下，物理距离更近的节点往往具有更高的吞吐量，因此在长远来看会更受偏好。这就是转发式 Kademlia 隐式感知底层拓扑结构的方式（Heep 2010）。

2.1.4 引导和维持 Kademlia 拓扑

本节讨论稳定的 Kademlia 拓扑是如何形成的，特别阐述了每个节点为达到并维持饱和的 Kademlia 连接性所必须遵循的引导协议。加入去中心化网络的节点通常是“新手”，可能仅通过一个已知对等节点进行初始连接，并没有其他预先知识。因此，引导过程需要包括一个初始步骤，以帮助新手节点开始交换彼此的信息。这个发现过程称为 hive 协议。

引导节点

Swarm 没有专门的引导节点类型或模式。这意味着新手节点应能够连接到网络中的任意节点并引导其期望的连接。为了避免任何单个节点承载过多负担，应尽量避免将某个特定节点选为初始连接节点。新手节点的引导角色应尽可能分布在参与网络的节点之间。这可以通过邀请系统实现，或者通过运行公共网关的集中式引导节点服务来实现，该网关响应 API 请求并返回在线对等节点中随机选定节点的 bzz 地址。

一旦新手节点与网络中的某个节点建立连接，hive 协议便会启动，新手节点开始学习其他节点的 bzz 地址，从而开始引导其连接构建过程。

建立连接

最初，每个节点的饱和深度为 0。节点会在其饱和深度发生变化时，将当前深度广告发送给已连接的对等节点。当节点 A 收到来自节点 B 的新连接请求时，A 仅在连接节点 A 的其他对等节点与节点 B 的接近度阶不低于该对等节点的广告饱和深度时，才通知它们节点 B 的连接情况。通知始终发送给与新连接共享相同接近度阶桶的对等节点。形式上，当 y 连接到 x 时，x 会通知其部分已连接对等节点。如果 $PO(x,p) = PO(x, y)$ 或 $dp \leqslant PO(y,p)$，则节点 p 属于被通知的对等节点集合。通知以协议消息的形式发送，并包括完整的覆盖地址和底层地址信息。

成熟连接

当连接了足够多的节点时，某个桶就会达到饱和状态，此时节点的邻域深度开始增加。节点在深度变化时，通过广告通知对等节点最新深度。随着深度的增加，节点接收到的对等节点通知会逐渐减少。当节点找到所有最近邻节点并饱和了所有桶时，不会再接收到新的对等节点。因此，如果节点在一段时间内未接收到新的对等节点，就可以推断其已达到饱和的 Kademlia 状态。

为了避免硬性截止日期和饱和状态的二元判定，可以通过考虑自上次新对等节点到达以来的时间来量化饱和的确定性。在连接稳定的情况下，每个节点最终都会认识其最近邻节点并与其建立连接，同时确保每个桶至 d 的深度保持非空。因此，每个节点都会收敛到饱和状态。

为了在对等节点连接变化的情况下保持稳健的 Kademlia 拓扑，确保每个接近度阶桶中包含多个对等节点至关重要。这可以防止节点在没有新节点加入网络时退化为更低的饱和状态。

2.2 Swarm 存储

在本节中，2.2.1 节首先展示了具有 Kademlia 拓扑的准永久连接网络如何支持负载均衡的、分布式的固定大小数据块存储。在 2.2.1 节中，我们详细说明了数据块的通用要求，并引入了实际的数据块类型。最后，在 2.2.5 节中，我们讨论了通过邻域复制提供冗余，作为应对节点波动的第一道防线。

2.2.1 分布式不可变数据块存储

本节讨论了如何利用 Kademlia 覆盖路由网络实现去服务器化的分布式哈希表（DHT）存储解决方案。然后，我们引入了 Swarm DISC 模型，这是 Swarm 对分布式哈希表在存储方面的狭义解释。该模型对数据块提出了一些要求，并引入了“上传”协议。

正如 Swarm 中的惯例，我们提供了一些该缩写的解释，概括了其最重要的特性：

Decentralised infrastructure for storage and communication（存储与通信的去中心化基础设施），

Distributed immutable store for chunks（分布式不可变数据块存储），

Data integrity by signature or content address（通过签名或内容地址实现数据完整性），

Driven by incentives with smart contracts（通过智能合约驱动的激励机制）。

从 DHT 到 DISC

Swarm DISC 与广为人知的分布式哈希表（DHT）有许多相似之处。最重要的区别在于 Swarm 不追踪文件的存储位置，而是将文件片段直接存储在距离地址最近的节点上。以下内容详细探讨了 DHT 和 DISC 之间的相似性与差异。

分布式哈希表使用覆盖网络在节点之间实现键值容器的分布式存储（见图 2.5）。基本思想是将键空间映射到覆盖地址空间，并将容器中键的值存储在与该键地址相邻的节点中。在最简单的情况下，假设这是存储该值的最接近该键的单个节点。在具有 Kademlia 连接的网络中，任何节点都可以路由到最接近键地址的节点，因此查找（即查找键对应的值）只需进行路由请求即可。

用于分布式存储的 DHT 通常将内容标识符（作为键/地址）与可以提供该内容的种子节点列表（作为值）相关联（IPFS 2014、Crosby and Wallach 2007）。然而，相同的结构也可以直接使用：在 Swarm 中，存储在与地址最近的节点上的不是内容位置的信息，而是内容本身（见图 2.6）。

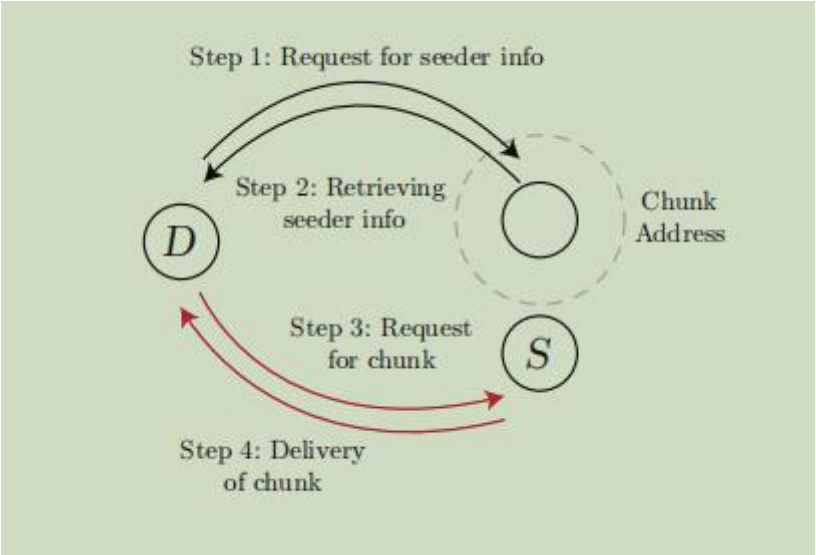

图 2.5: 用于存储的分布式哈希表 (DHT)：节点 D (下载者) 在步骤 1 中使用 Kademlia 路由查询数据块地址邻域中的节点，以在步骤 2 中检索种子节点信息。随后使用该种子节点信息在步骤 3 和步骤 4 中直接联系节点 S（种子节点）请求并接收数据块。

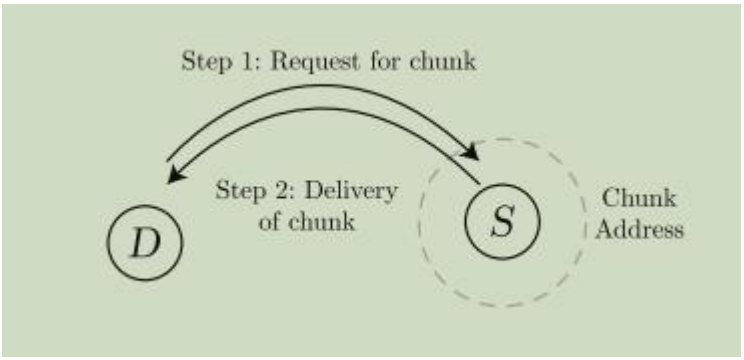

图 2.6: Swarm DISC（分布式不可变数据块存储）：在步骤 1 中，下载节点 D 使用 Kademlia 连接向更接近数据块地址的对等存储节点发送请求。该对等节点重复这一过程，直到找到持有该数据块的节点 S。换句话说，对等节点通过实时对等连接递归中继请求，最终到达数据块地址的邻域（请求转发）。在步骤 2 中，数据块沿相同路径以相反方向返回给请求节点（响应回传）。

约束条件

DISC 存储模型对哪些节点存储哪些内容有明确规定，这意味着以下限制：

固定大小的数据块

节点之间需要进行内容负载均衡，这通过将内容拆分为称为数据块（chunks）的等大小单元来实现（见 2.2.1 节）。

同步

必须有一个过程确保无论由哪个节点上传数据块，它们都能到达应存储的位置（见 2.3.2 节）。

合理的否认性

由于节点无法控制其存储的内容，因此应采取措施为节点运营者提供法律保护基础，使其能够合理地否认知道或有能力知道数据块内容（见 2.2.4 节）。

垃圾回收

由于节点承诺存储接近其地址的数据，因此需要有策略来选择在存储空间受限的情况下

保留或丢弃哪些数据块。

数据块（Chunks）

数据块是 Swarm 网络层使用的基本存储单元，它是地址与内容的关联。在 Swarm（2.3.1 节）中，假设数据块存储在与其地址接近的节点上，为了实现公平且均衡的负载，数据块的地址应在地址空间中均匀分布，其内容大小应受限制并大致相同。

在检索数据块时，下载节点必须能够根据地址验证内容的正确性。这样的完整性验证保证了地址关联内容的唯一性。为了防止无效网络流量，中继节点的第三方应该能够使用仅在节点本地可用的信息来验证数据块的完整性。

地址的确定性和无碰撞特性意味着数据块作为键值关联是唯一的：如果存在某个地址的数据块，则不会有其他有效数据块具有相同的地址。这一假设至关重要，因为它使数据块存储不可变，即不支持替换/更新操作。在数据块中继的上下文中，不可变性非常有利，因为节点可以仅通过检查地址来协商其拥有的数据块信息。这在流协议（见 2.3.3 节）中发挥了重要作用，并证明了 DISC 作为分布式不可变数据块存储的合理性。

总结

数据块地址需要满足以下要求：

确定性

支持本地验证。

无碰撞性

提供完整性保证。

均匀分布

实现负载均衡。

在当前版本的 Swarm 中，我们支持两种类型的数据块：内容地址数据块和单所有者数据块。

2.2.2 内容地址数据块

内容地址数据块并不被视为有特定意义的存储单元，即它们可以只是将较大数据块或文件拆分而成的任意数据片段。有关上传时如何将文件拆分为数据块、下载时如何从数据块重新组装文件的方法将在 4.1 节详细介绍。内容地址 Swarm 数据块的数据大小限制为 4 KB。使用这种小数据块大小的一个理想结果是，即使是相对较小的文件，也可以实现并发检索，从而降低下载延迟。

二叉 Merkle 树哈希

在 Swarm 中，标准的内容地址数据块称为二叉 Merkle 树数据块（BMT 数据块）。BMT 数据块的地址通过二叉 Merkle 树哈希算法（BMT 哈希）计算得出。BMT 使用的基础哈希算法是 Keccak256，其特性如均匀性、不可逆性和抗碰撞性都继承至 BMT 哈希算法。由于具有均匀性，一组随机的分块内容将生成在地址空间中均匀分布的地址，这在节点间实现了存储需求的负载均衡。

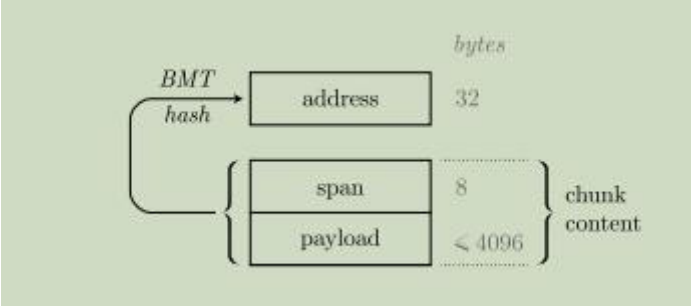

图 2.7：内容寻址块。一个最多 4KB 的有效载荷，前面加上一个 64 位小端编码的跨度（span），构成了传输中使用的块内容。块的内容地址是跨度和有效载荷的 BMT 根连接后的字节切片的哈希值。

BMT 数据块地址是 8 字节跨度和基于底层数据 32 字节段构建的二叉 Merkle 树（BMT）根哈希值的哈希（见图 2.8）。如果数据块内容小于 4 KB，则计算哈希时需将数据块填充为 4096 字节长度，全零填充至不足之处。

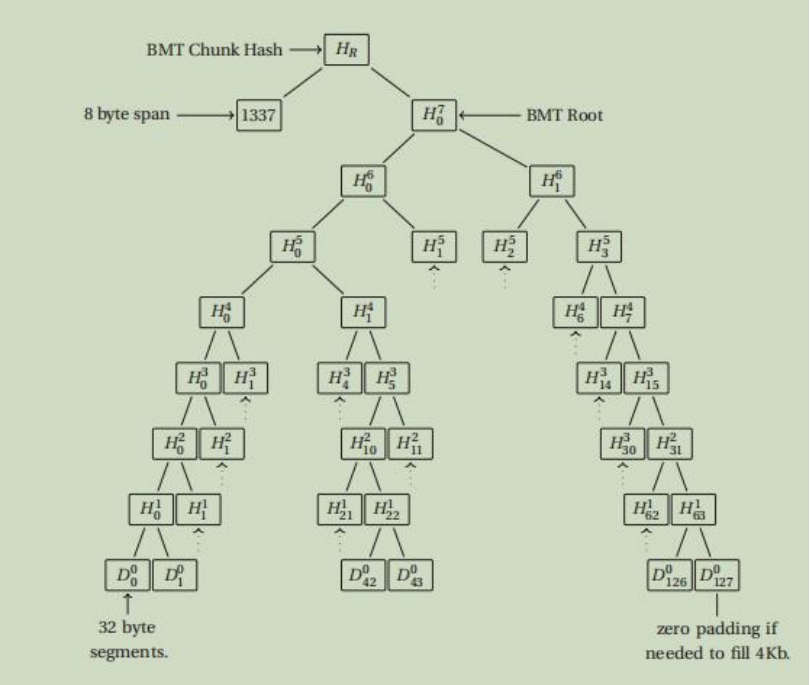

图 2.8：Swarm 中的二进制默克尔树（Merkle Tree）分块哈希：1337 字节的分块数据被分割成 32 字节的段。使用零填充(Zero padding)将剩余部分填充至 4 千字节。使用 Keccak256 将成对的段哈希在一起，以构建二叉树。在第 8 层，二进制默克尔根（Merkle root）前加上 8 字节的跨度（span）并进行哈希，生成 BMT 分块哈希。

这种结构支持具有 32 字节分辨率的紧凑型包含证明。包含证明是用于验证一个字符串是另一个字符串的子字符串的证明，例如验证某个字符串包含在数据块中。包含证明基于特定索引的数据段进行定义，详见图 2.9。这样的 Merkle 证明也用于存储节点提供其拥有某个数据块的证据（即保管证明，见 3.4 节）。结合 Swarm 文件哈希（见 4.1.1 节），这些证明能够为文件提供对数级别的包含证明，即证明某个字符串是文件的一部分。

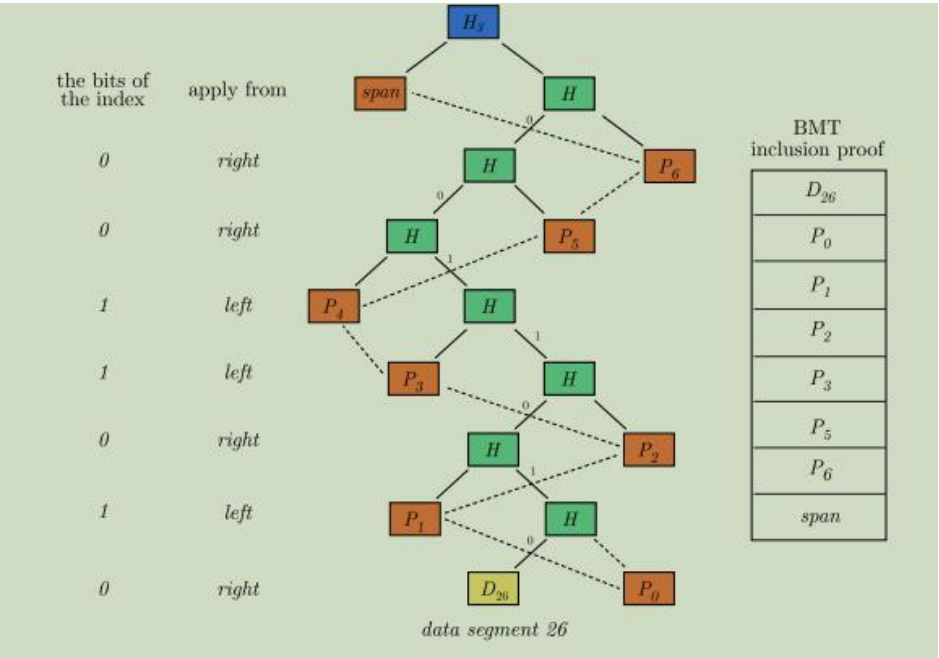

图 2.9：块的紧凑段包含证明。假设我们需要为一个块的段 26（黄色）提供证明。BMT 中的橙色哈希是从数据段到根路径上的姐妹节点，它们构成了证明所需的部分。当这些哈希与根哈希和段索引一起提供时，证明可以被验证。证明项 i 需要应用在哪一侧取决于索引的二进制表示中的第 i 位（从最低有效位开始）。最后，前置跨度，生成的哈希应与块的根哈希匹配。

2.2.3 单所有者数据块

通过单所有者数据块，用户可以将任意数据分配给一个地址，并使用其数字签名证明数据块的完整性。地址通过标识符和所有者的哈希值计算得出。数据块的内容结构由标识符、有效负载和用于证明标识符与有效负载关联性的签名组成（见图 2.10）。

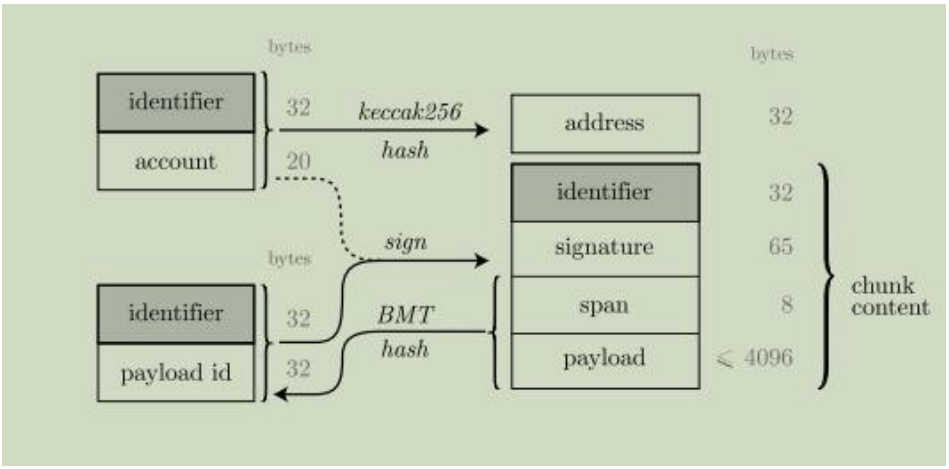

图 2.10：单一所有者数据块。数据块内容由头部和最多 4KB 的有效载荷组成。最后一个头部字段是 8 字节的跨度，与内容寻址数据块中的前置方式相同。前两个头部字段提供了单一所有者的完整性认证：一个标识符和一个签名，该签名用于确认标识符以及跨度和有效载荷的 BMT 哈希。地址是标识符和签名者账户的哈希值。

内容
有效负载
32 字节的任意标识符

65 字节的椭圆曲线签名〈r, s, v〉表示形式（32+32+1 字节）

8 字节的 uint64 类型数据块跨度（二进制小端格式）
最多 4096 字节的常规数据块数据
地址
Keccak256 哈希值，由标识符 + 所有者账户计算得出
单所有者数据块的验证过程如下：
将数据块内容反序列化为标识符、签名和有效负载字段。
构造由标识符和有效负载的 BMT 哈希组成的预期明文。
使用该明文从签名中恢复所有者地址。
将标识符和所有者地址的哈希值（预期地址）与数据块地址进行比较。

单所有者数据块为地址空间的某些部分提供了虚拟分区，这些分区与单一所有者相关联。检查单所有者数据块的有效性实际上是在验证所有者是否拥有使用正确标识符写入该地址的权限。

从数据块的跨度和有效负载长度可以看出，单所有者数据块可以封装常规的内容地址数据块。任何人都可以简单地将常规数据块重新分配到其标识符指定的子空间地址中（参见 4.4.4 节）。

需要注意的是，单所有者数据块的完整性概念比内容地址数据块要弱一些：毕竟，从原则上讲，可以将任意有效负载分配给某个标识符并签名。然而，鉴于数据块只能由单一所有者（即拥有签名所需私钥的用户）创建，因此可以合理地期望其具有唯一性保证，因为节点通常会遵循应用协议以实现期望的结果。然而，如果私钥所有者使用相同标识符对两个不同的有效负载进行签名并将这两个数据块上传到 Swarm，则网络行为将变得不可预测。在第 3 层中可以采取措施来缓解这一问题，4.3.3 节对此有详细讨论。

总结

通过两种类型的数据块，数据完整性与无碰撞哈希摘要相关联，该摘要要么源自单所有者和由签名证明的任意标识符，要么直接源自内容。这也证明了 DISC 这一缩写的含义：通过签名或内容地址实现数据完整性。

2.2.4 数据块加密

数据块应默认加密。除了满足客户端对机密性的需求之外，加密还有两个重要作用：(1) 加密对数据块内容的混淆提供了一定程度的合理否认性，在整个系统中普遍采用加密可以增强这种防御能力。(2) 能够选择任意加密密钥并保持均匀分布的特性，为“挖掘”数据块提供了可预测的方法，即生成相同内容的加密版本，使生成的数据块地址满足某些约束条件，例如更接近或更远离特定地址。这一特性在以下两种场景中至关重要：(1) 价格套利（见 3.1.2 节），(2) 邮票资源的高效利用（见 3.3 节）。

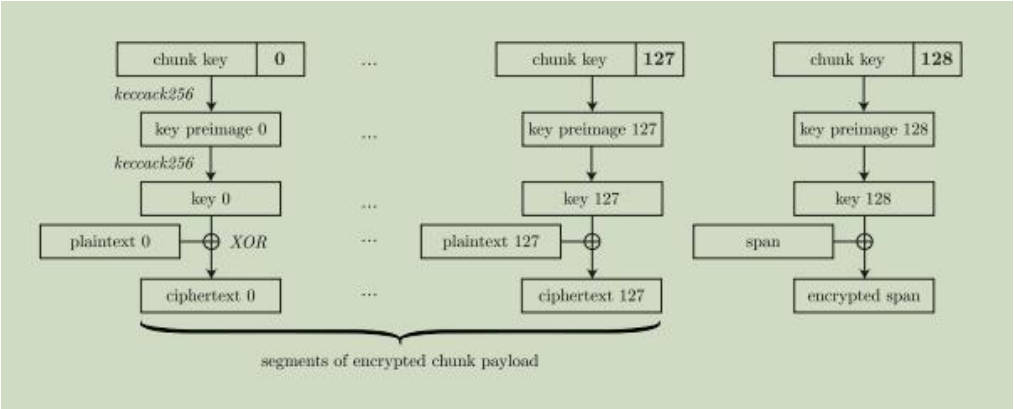

图 2.11：Swarm 中的分块加密。使用修改后的计数器模式分组密码进行对称加密。明文输入是填充了随机字节至 4 千字节的内容。跨度字节也被加密，就好像它们是有效载荷的延续一样。

加密方法

长度小于 4 KB 的数据块会使用从数据块加密种子生成的随机字节进行填充。完整的数据块明文使用流加密算法进行加密和解密，通过对称密钥种子生成的伪随机数发生器 (PRNG) 进行异或操作。为了避免引入额外的加密算法而增加攻击面，流加密算法选择 Keccak256 计数模式，即将密钥与每个连续 32 字节段的计数器组合后进行哈希计算。

为了实现加密文件中各个数据段的选择性披露，同时不泄露文件的其余部分，我们在此方案中增加了一个步骤，用于为数据块中的某个段派生加密密钥。这种方案在 EVM（以太坊虚拟机）中易于实现，且成本低廉，适用于包含加密 Swarm 内容明文的智能合约。

前置的编码元数据（表示数据块跨度）也会被加密，就像它是数据块的延续一样，即使用计数器值 128 进行加密。加密的数据块内容与未加密的数据块一样，使用 BMT 哈希摘要进行哈希计算。在网络层中，尽管可以通过跨度值推测出数据块已被加密，但除此之外，加密数据块在行为上与未加密数据块完全一致。

单所有者数据块的加密

单所有者数据块也可以加密，这意味着它们封装了一个加密的常规数据块。因此，其有效负载和跨度符合上述数据块内容加密方式，而签名标识符的哈希值是加密后跨度和有效负载的 BMT 哈希值，即与被封装的数据块的哈希值相同。

2.2.5 通过局部复制实现冗余

请求数据的方式必须具有高度的弹性和可靠性。为此，Swarm 实现了“纵深防御”策略。当由于转发问题导致请求失败时，可以尝试向另一个节点重新发送请求。或者，为了防止此类情况的发生，节点可以立即发起并发检索请求。然而，如果存储数据块的唯一节点从网络中掉线，这些备用方案将无效。因此，为了确保数据可用性，实现冗余至关重要。如果最近的节点是存储请求数据的唯一节点且掉线，那么将无法检索到内容。对此基本场景的解决方案是确保每个最近邻节点集群都持有最接近它们的每个数据块的副本，从而实现数据块的多重存储和冗余。

最近邻集群的大小

如果 Kademlia 拓扑定义于存储节点，那么在具有 Kademlia 拓扑的网络中，存在深度 d，使得：(1) 每个小于 d 的接近度顺序桶中至少包含 k 个存储节点，(2) 所有接近度顺序为 d 或更高的存储节点实际上都是连接的节点。为了确保数据冗余，还可以增加一个条件：

(3) 由深度 d 定义的最近邻集群至少包含 r 个节点。

定义节点 x 深度为 d 的邻域大小为 NHSx(d)，即由深度 d 定义的邻域的节点数量。当满足以下条件时，节点具有冗余因子为 r 的 Kademlia 连通性：存在深度 d，使得 (1) 每个接近度顺序桶小于 d 的桶中至少有 k 个存储节点（k 为桶密度参数，见 2.1.3 节），(2) 所有接近度顺序为 d 或更高的存储节点实际上都已连接，(3) NHSx(d) ⩾ r。

我们可以取满足条件 (1) 和 (2) 的最大深度 d′。这样的 d 是一定存在的，并且 hive 协议总是能够引导它。当 d′ 减小时，不同的邻域数量成比例增加，因此对于任何不超过网络节点总数的冗余参数 r ⩽ N = NHSx(0)，总会有某个 0 < dr ⩽ d′ 满足 NHSx(dr) ⩾ r。因此，冗余的 Kademlia 连通性始终是可以实现的。

对于特定的冗余级别，完全连接的邻域定义了一个责任区。责任区的接近度顺序边界定义了节点的责任半径。当某个数据块地址落入节点的责任半径范围内时，该节点被认为是该数据块的“责任节点”，即需要存储该数据块。

此时，通过图 2.12 展示邻域的结构是有帮助的。

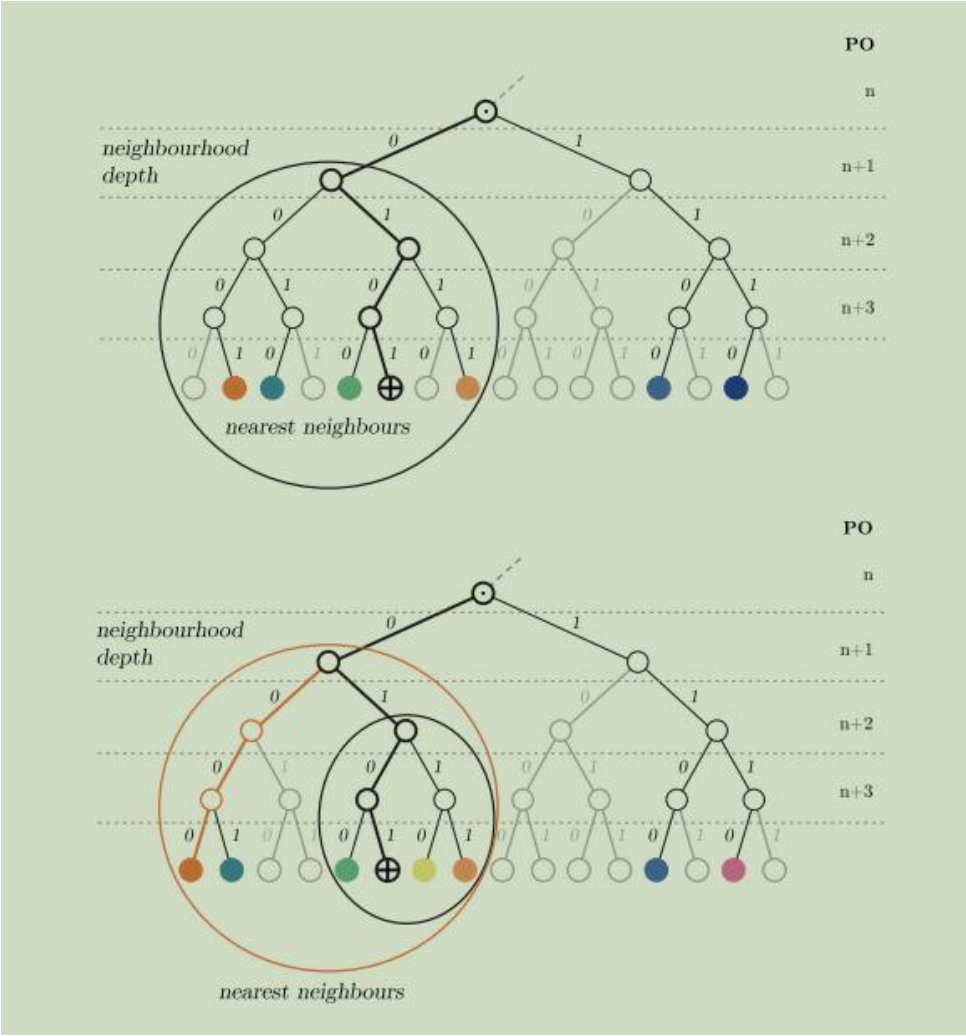

图 2.12：最近邻节点。上图：每个 PO 定义一个邻域，节点（黑色圆圈）的邻域深度定义为最高 PO，使得该邻域至少有 R=4 个对等节点（冗余参数），并且所有较浅的区间均为非空。下图：一个不对称的邻域。橙色节点的最近邻节点包括黑色节点，但反之则不然。

冗余可检索性

如果某个数据块在最多 r 个责任节点离开网络后仍然可以被检索到，则称该数据块具有冗余度为 r 的可检索性。目前所介绍的“单个最近节点保存数据块”的方式可以解释为

冗余度为 0 的可检索性。如果节点在其责任范围内完全复制其内容（见 2.3.3 节），则 Swarm DISC 中的每个数据块都可以实现冗余度为 r 的可检索性。

设节点 x 是距离数据块 c 最近的节点。由于节点 x 具有冗余度为 r 的 Kademlia 连通性，因此在完全连接并复制内容的邻域中，有 r+1 个节点负责存储该数据块。在 r 个责任节点掉线后，仍有一个节点保留该数据块。然而，如果在 r 个节点离开时保持 Kademlia 连通性，则该节点仍可被网络中的任何其他节点访问，因此该数据块仍然可以被检索到。

为了确保所有数据块在冗余度为 r 的情况下保持可检索性，当网络重组形成新的邻域时，这些邻域中的节点必须通过重新同步其内容来满足协议的复制标准。这被称为“最终一致性保障”。

资源限制

我们假设以下两点成立：（1）转发策略确保请求沿着稳定的节点链路进行中继，（2）存储策略确保最近邻集群中的每个节点（r 个存储节点）存储所有地址落入其责任半径范围内的数据块。只要这些假设成立，即使有 r 个存储节点同时掉线，每个数据块仍然是可检索的。然而，对于（2），我们还需假设最近邻集中每个节点都能够存储每个数据块。然而，现实情况是，所有节点都有资源限制。随着时间推移，上传到 Swarm 的数据块总量将无限增长。除非整体存储容量稳定增加，否则我们应预期 Swarm 节点只能存储部分数据块。当某些节点达到其存储容量上限时，将面临以下选择：是停止通过同步接收新的数据块，还是通过删除现有数据块来腾出空间。

从本地存储中清理数据块的过程称为垃圾回收。决定选择哪些数据块进行垃圾回收的过程称为垃圾回收策略。对于一个以利润最大化为目标的节点而言，始终应优先回收预计在未来收益最小的数据块。为了最大化利润，节点需要准确预测哪些数据块的价值较低（见 3.3 节）。因此，为了考虑这些存储容量限制，我们引入数据块价值的概念，并用最小价值约束修改定义：

在 Swarm 的 DISC 模型中，任何时候都存在一个数据块价值 v，使得每个价值大于 v 的数据块都是可检索的，并且在同步后具有冗余度为 r 的可检索性。

理想情况下，该价值应对应于上传者指示的保持数据块的重要性。为了让存储节点遵守该价值，该价值还应与数据块的盈利性保持一致，因此该价值通过上传的定价来体现（见 3.3.4 节）。

2.3 推送与拉取：数据块的检索与同步

在本节中，我们展示了数据块在网络中的实际移动方式：数据块在上传时如何被推送到它们所属邻域的存储节点，以及在下载时如何从存储节点被拉取。

2.3.1 数据块检索

在分布式数据块存储系统中，当请求方与距离数据块最近的节点之间可以路由消息时，我们称该数据块为可访问数据块。向数据块地址发送检索请求消息会到达该节点。由于最终一致性的特性，最接近数据块地址的节点将存储该数据块。因此，在具有良好 Kademlia 拓扑的 DISC 分布式数据块存储系统中，对于网络中的每个节点来说，所有数据块始终都是可访问的。

数据块传送

为了实现检索功能，仅数据块的可访问性是不够的，还需要一个将内容传回请求节点的过程，最好仅使用数据块地址即可实现。实现这一目标的方法至少有三种（见图 2.13）：

直接传送

通过底层网络连接直接将数据块传送回去。

路由传送

使用路由协议将数据块作为消息发送至请求方。

反向传送

数据块响应沿着请求被转发的路径逆向传递，直至返回给请求发起者。

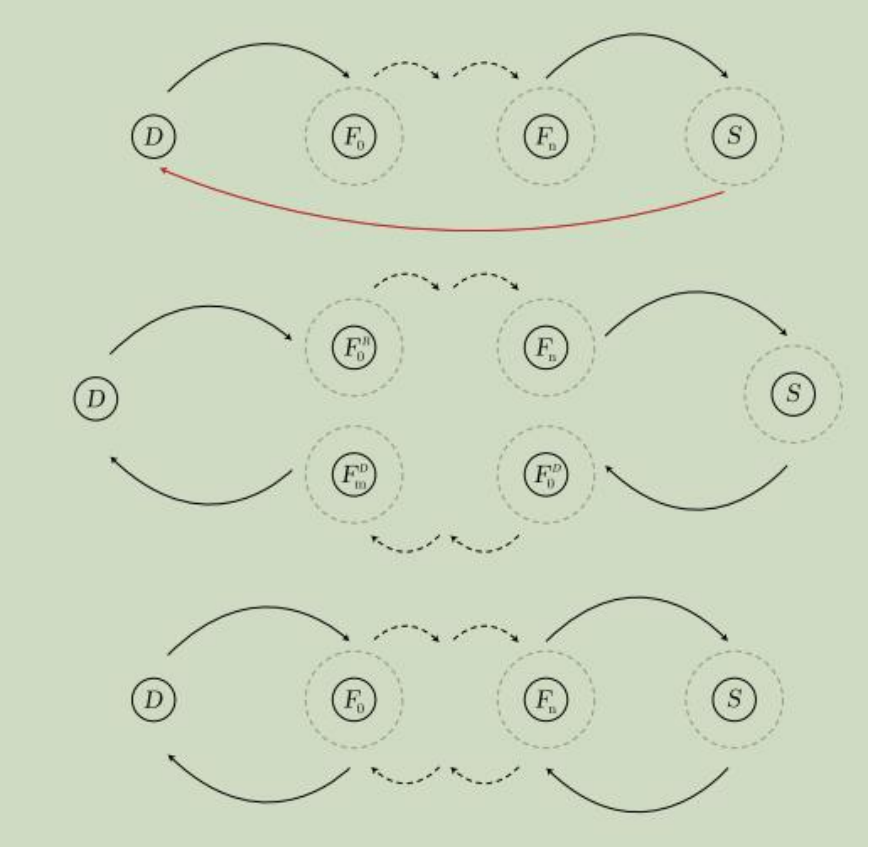

图 2.13：分块传输的替代方式。顶部：直接传输：通过直接的底层连接。中部：路由传输：使用 Kademlia 路由发送分块。底部：反向传输：重新使用请求路径上的确切对等节点来中继传输响应。

首先，通过显而易见的直接传送方式，数据块通过低级网络协议一步传送给请求节点。这需要建立临时连接，以降低延迟为代价提高传输速度，但会牺牲隐私安全性。其次，通过路由传送方式，数据块根据存储节点发送时决定的路由返回至请求方。无论是直接传送还是路由传送，这两种方式都假设请求方地址（至少部分）被存储节点和路由节点所知。因此，这些方法会泄露可以识别请求方的信息。

然而，采用转发-反向传送的 Kademlia 路由模式并不需要这样做：存储节点将数据块传送给向其发出请求的节点，而中继节点会记住其哪个对等节点请求了哪个数据块。当数据块被传送时，中继节点将数据块传回给它们的直接请求节点，以此类推，直到数据块最终到达发起请求的节点。换句话说，数据块响应沿着请求路径反向传送至发起节点（见图 2.14）。由于这种方式是转发的反向过程，因此我们可以将其戏称为反向传送。Swarm 使用此选项，从而确保在检索过程中不泄露请求方的任何身份信息，实现完全匿名的数据块检索。

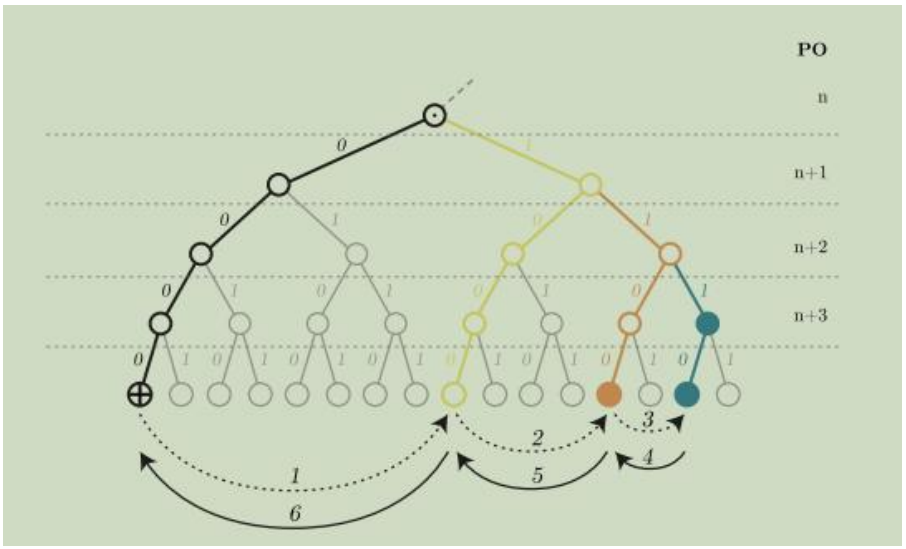

图 2.14：反向传输：在转发 Kademlia 中用于匿名请求-响应往返的模式。这里一个具有覆盖地址……0000……的节点向目标……1111……发送请求，而距离目标最近的在线节点是……1110……。开头的省略号表示请求者和目标共享的前缀，其长度为 n 位。结尾的省略号表示对于路由来说不相关的地址部分，因为在该深度节点已经是唯一的。请求使用通常的 Kademlia 转发，但沿途的转发节点会记住请求来自的对等节点，以便当响应到达时，它们可以反向传输（即沿相同路径传回）该响应。

在检索协议中默认确保请求方匿名性是 Swarm 坚持的重要特性。该功能旨在保护用户隐私，并提供抗审查访问能力。

实现如图 2.15 所示的反向传送检索方案还有助于防范垃圾数据、防止滥用、提高可扩展性并实现激励机制，这将在本节其余部分进行讨论。

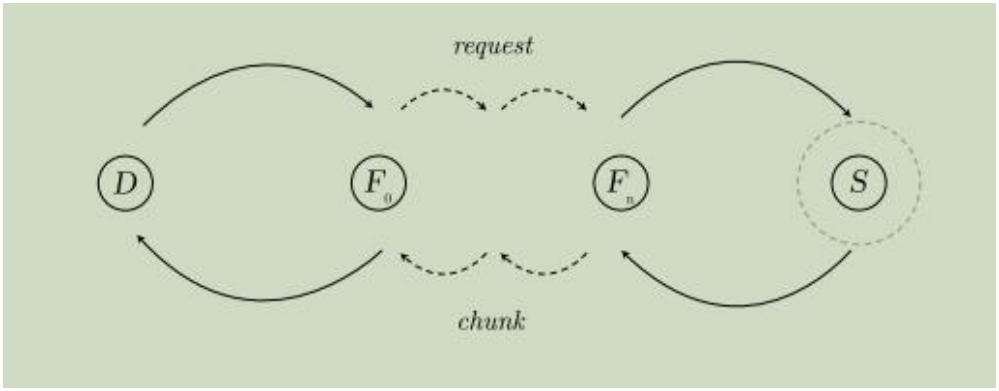

图 2.15：检索。节点 D（下载器）向块的地址发送检索请求。检索使用转发 Kademlia，因此请求通过转发节点 F0，……，Fn 一路被转发到节点 S，即最接近块地址的存储节点。然后块通过沿相同路径返回被传递给下载器。

防范无请求数据块

为了记住请求，转发节点需要分配一定的资源，这些资源会占用内存空间并带来一定成本。对于没有返回相应数据块传送的请求，应进行垃圾回收，因此需要定义请求在其生命周期内的活动时间窗口。下游对等节点还需要知晓请求的超时时间，这是合理的，因为请求发起方需要在请求中附加生存时间（TTL）以表示它愿意等待响应的时长。

发送无请求数据块（即未被请求的数据块）是一种违规行为，因为它可能导致拒绝服务攻击（DoS）。通过记住请求，节点能够识别无请求数据块传送并惩罚发送这些数据块的对

等节点。请求过期后传送的数据块将被视为无请求数据块。由于节点间对于过期时间的判断可能存在差异，因此需要对无请求数据块传送保持一定的容忍度，但如果超过特定（但仍然很小）的请求比例，违规节点将被断开连接并加入黑名单。这种本地制裁是激励遵守协议的最简单、最有效方式（见 3.2.5）。

重新请求

Swarm 网络中大部分节点可能并不总是稳定在线。在这种高频节点上下线的情况下，如果采用将请求转发给任意一个更接近节点的简单策略，问题将非常严重：如果传输路径上的某个节点在数据块传送完成之前下线，请求-响应回路将被中断，从而导致该数据块无法检索。如果与请求节点的连接断开，请求方对数据块的支付承诺将被视为无效，因此重新请求该数据块不会带来任何损失。

超时与未找到

需要注意的是，Swarm 并没有针对未找到的数据块提供显式的负响应。在原则上，距离数据块地址最近的节点可以确定该地址下不存在数据块，并返回“未找到”响应。然而，这样做并不理想，原因如下：尽管最接近数据块的节点能够验证其在网络中期望位置的缺失情况，但距离较远的节点无法可靠地得出相同结论，因为它们缺乏一手验证信息，并且之后获取的任何关于该数据块可检索性的积极证据都可能具有合理的可否认性。

总而言之，只要数据块传送有可能为存储节点带来收益，最佳策略就是保持挂起请求，直到超时，并准备好在数据块出现时进行传送。数据块在请求发出后出现的几种方式包括：（1）通过现有对等节点同步、（2）新节点的加入、（3）请求在上传之前已存在，例如请求方已经“订阅”了单所有者地址（见 6.3 节）以降低检索延迟。这与传统的服务器-客户端架构在资源要么存在要么不存在的预期情况截然不同。

机会性缓存

使用反向传送机制将数据块传送至请求方的响应方式也实现了机会性缓存，即当转发节点接收到数据块时，会将其保存以备后续再次请求。这一机制对于确保 Swarm 自动扩展流行内容的存储和分发至关重要（见 3.1.2 节）。

激励机制

到目前为止，我们已经展示了通过使用检索协议和维护 Kademlia 连接，网络中的节点能够成功检索数据块。然而，由于转发操作会消耗稀缺资源（如带宽），如果无法对这些带宽使用进行计量，网络的可靠性将取决于节点间搭便车行为与利他行为的比例。为了解决这一问题，我们将在第 3 节概述一套经济激励体系，该体系与网络中节点的期望行为保持一致。当节点运营者采用这些以利润最大化为目标的策略时，将产生有利于整个网络用户的涌现行为。

2.3.2 推送同步

在前面的章节中，我们介绍了如何利用保持 Kademlia 覆盖拓扑结构的节点网络作为分布式数据块存储系统，以及如何使用转发 Kademlia 路由来定义数据块检索协议。在讨论数据块检索时，我们假定数据块被存储在与其地址最接近的节点上。本节描述了实现这一假设的协议：确保数据块在上传到任意节点后能够传送到其指定的存储节点。

这一网络协议称为推送同步，它与数据块检索类似：首先，将数据块通过与检索请求相同的路径中继至与数据块地址最接近的节点，然后该节点沿相同路径返回一份托管声明（见图 2.16）。存储节点发送回给上传节点的托管声明表明，该数据块已经到达了可供全网节点检索的邻居节点范围。通过跟踪每个上传数据的组成数据块的这些响应，上传者可以确保

其上传的数据在分享或发布上传地址前已在整个网络中可被任何节点成功检索。这些推送同步数据块和收到的托管声明的计数可作为上传进度条的后台数据，向上传者反馈其数据在网络中成功传播的情况（见 6.1）。

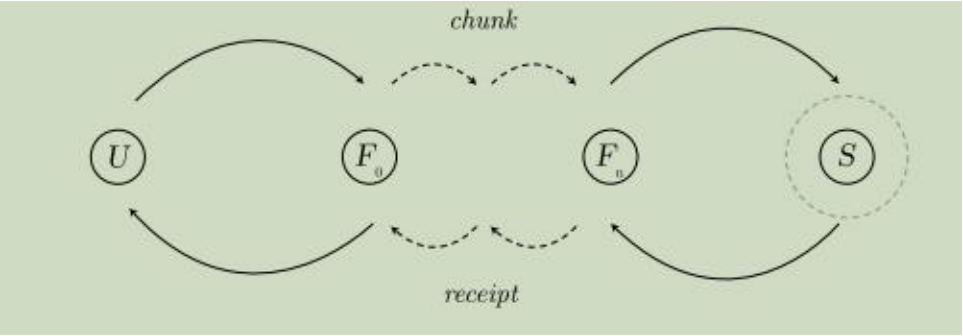

图 2.16：推送同步。节点 U（上传者）将数据块推送同步到该数据块的地址。推送同步使用转发机制，因此数据块通过转发节点 F0, ..., Fn 一路中继到节点 S，即最接近数据块地址的存储节点（箭头表示通过直接的点对点连接传输数据块）。随后，由 S 签署的保管收据声明将沿着相同的路径返回，作为对上传者的确认。

托管声明由声称是最接近数据块地址的节点签署。同样地，上传者的身份也可以保持匿名，因此转发 Kademlia 同样能够实现匿名上传。

另一个相似之处是，为了允许响应的反向传送，节点需要记录是由哪个对等节点发送了特定的数据块。这一记录应在托管声明响应预计返回的短时间内保留。当此时间段结束后，记录会被移除。与记录不匹配的托管声明被视为未经请求的数据块，仅允许其占推送同步流量总量的一小部分。若超出这一容忍阈值，则会对发送方进行惩罚，包括断开连接和加入黑名单（见 3.2.5）。

本节介绍了如何通过使用转发 Kademlia 路由和响应反向传送的网络协议来组织数据块上传的物流过程。然而，这一方案只有在激励机制与安全机制配合时才算完整：节点遵循协议的策略需要通过激励机制加以推动，同时需要对拒绝服务攻击（DoS）行为进行抑制。相关细节将在 3.3 和 3.1.3 节中进行深入探讨。

2.3.3 拉取同步

拉取同步协议负责实现以下两个属性：

最终一致性

在拓扑因节点离线或新节点加入而变化时，同步邻居节点的数据。

最大资源利用

节点可以从其对等节点拉取数据块以填充其剩余存储空间。

拉取同步是以节点为中心，而非以数据块为中心，即它确保节点在需要时填充存储，并在邻居节点间同步数据块。当两个节点建立连接时，会开始双向同步，即在每个对等连接中，数据块在两个方向上同时传输。同步的两个方向由独立的数据流管理。在数据流上下文中，数据流的消费者称为下游对等节点或客户端，而提供者称为上游对等节点或服务器。

当两个节点连接并进行数据块同步时，上游节点会在每个接近度顺序 bin 中提供其本地存储的数据块流。为了接收比上游节点距离更接近下游节点的数据块，下游节点可以订阅 Kademlia 表中上游节点所在的接近度顺序 bin 的数据块流。如果对等连接在邻居深度 ddd 之内，客户端会订阅所有接近度顺序 bin 为 ddd 或更大的数据块流。结果是，对等节点最

终会复制属于其职责范围内的所有数据块。

拉取同步服务器在数据流协议中被称为流提供者。节点通过升序的存储计数（称为 bin ID）索引本地存储数据块的时间记录。对于每个接近度顺序 bin，上游节点会按存储时间戳的降序提供数据块流。在每个对等连接的同步流中，数据块可以从多个上游节点同步到下游节点。为了避免向已经拥有相同数据块的对等节点发送数据块，从而节省带宽，流协议实现了一个往返机制：在发送数据块之前，上游节点会先发送一批数据块的地址列表。下游节点随后回应该批次中它实际需要的数据块（见图 2.17）。需要注意的是，下游节点根据数据块地址判断其是否已拥有该数据块。因此，这种方法依赖于 2.2.1 节中讨论的数据块完整性假设。

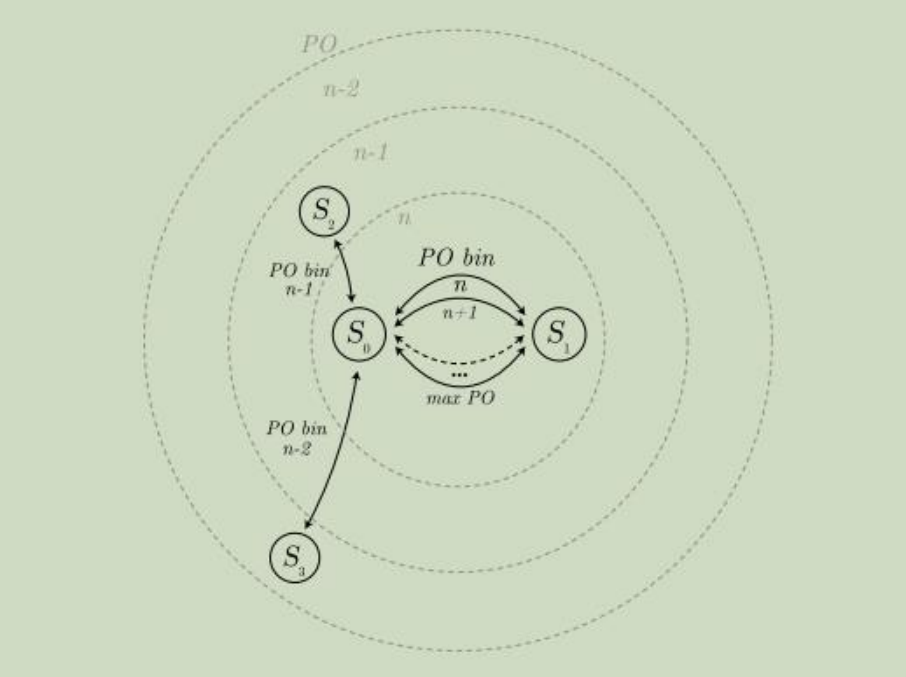

图 2.17：拉取同步。节点持续同步其最近的邻居。如果它们有空闲容量，它们还会从超出邻居深度的对等节点拉取属于较浅分区的同步块。

在对等连接的上下文中，当客户端同步了上游节点的所有数据块时，被称为“已同步”。需要注意的是，由于磁盘容量的限制，节点必须设置一个值的阈值，因此“所有数据块”实际上是“值大于 vvv 的所有数据块”（vvv 是一个常量排名函数，其来源将在 3.3.3 节中讨论）。为了让一个节点承诺其存储了所有值大于 vvv 的数据块，必须确保其所有邻居节点也存储了值大于 vvv 的数据块。换句话说，节点在同步过程中会继承其存储对等节点中最大的数据块值。

如果数据块按照存储顺序进行同步，节点未必总是拥有最有价值（请求频率最高）的数据块。因此，建议从上游节点提供的最热门数据块开始同步，并在达到存储容量时结束同步。这样可以优化节点有限的存储资源。关于同步和垃圾回收的详细内容，请参见 3.3 和 3.3.4 节。

在本节的结尾，我们展示了健康的 Swarm 如何满足最终一致性的标准。任何节点本地存储中的数据块在同步至其存储节点后都将变得可检索。这是因为，只要网络中的节点不断拉取比上游节点更靠近自己的数据块，每个数据块的传输路径也符合推送同步协议中有效转发路径的要求。如果新节点加入或旧节点退出，邻居节点会发生变化，但只要本地冗余足够高，节点流失就不会导致先前可检索的数据块变得不可检索，邻居节点最终会复制其内容并恢复冗余。假设出现一个极不可能的场景，即形成了一个全新的邻域，而最初存储这些内容

的节点被排除在新邻域之外，从而导致这些数据块暂时不可用。但即便如此，只要相关 bin 上有节点运行拉取同步流，冗余可检性最终也会恢复。

2.3.4 轻节点

轻节点的概念是针对带宽不足的环境，例如低速网络的移动设备或仅允许临时或低容量存储的设备。

轻节点是指未完全参与前面所述的常规协议的节点，即未参与检索、推送同步或拉取同步。

由于带宽限制或无法维持底层连接，轻节点不应被期望按照 Kademlia 路由规则转发消息。这需要向其对等节点传达，以便它们不会将消息中继至该节点。

由于 Swarm 中的所有协议都是模块化的，因此节点可以根据其能力和收益需求独立启用或禁用任一协议。例如：如果节点没有可用存储空间，但有剩余带宽，它可以仅作为转发节点参与。当然，尽管技术上可以关闭某些协议，但节点始终需要考虑其对等节点对服务质量的预期。如果节点提供的服务未达到对等节点的预期，对等节点可能会选择不与该节点进行交互。

由于转发操作可以带来收益，这些轻节点仍可能被激励接受数据块的检索请求。然而，如果轻节点在接近度顺序 bin ppp 以上具有 Kademlia 连通性（即连接了深度为 ddd 且邻居数量为 rrr 的最近邻居节点集中的所有存储节点，并且从 ppp 到 ddd 的每个接近度顺序 bin 中至少有一个对等节点），它们可以对此进行广播，从而参与转发操作。

当它们想要检索或推送数据块时，如果数据块地址位于一个没有对等节点的接近度顺序 bin 中，它们可以选择另一个 bin 中的对等节点。尽管这可能导致虚假跳跃（即消息目标地址与最新对等节点的接近度并未因中继操作而增加），但 Kademlia 关于路由可以在对数步数内完成的假设仍然成立。

被标记为存储/缓存节点的节点应存储所有高于特定价值的数据块。为了保持一致性，这些节点需要在其责任范围内同步内容，这就要求它们运行拉取同步协议。对于那些具备可用存储空间并上线以通过拉取同步流填充存储容量的潜在存储节点而言，同样适用。在早期阶段，让一个节点与其他已满载的存储节点进行同步并没有意义。然而，与其他同样为新加入的节点进行同步仍可能有助于其填充数据，尤其是当存储节点的带宽已经接近极限时。

关键在于，为了实现冗余和跳跃路由功能，在计算网络的饱和度时，不应将具有不完整、未饱和 Kademlia 表的轻节点纳入其他节点的计算范围。

第 3 章 激励机制

Swarm 网络由多个独立节点组成，这些节点运行实现 Swarm 协议的软件。需要认识到，尽管所有节点运行相同的协议，但网络的涌现行为并不能仅仅依靠协议本身来保证。由于节点是自主的，它们基本上“自由”地以任何方式响应来自其他节点的消息。然而，可以通过使节点以有利于网络期望涌现行为的方式行动而获得收益，同时让不利于网络的行为付出代价。在 Swarm 中，这主要通过使网络资源的使用者（净使用者）向提供者（净提供者）转移价值来实现。

3.1 带宽共享

3.1.1 服务和转发的激励

转发型 Kademlia 和重复交易

数据块的检索最终是由访问内容的用户发起的，因此与此检索相关的所有成本都应由用户承担。尽管在现今“免费”的网络中，收费检索听起来不受欢迎，但当前网络中的许多问题正是因为消费者无法直接与内容发布者分享托管和分发成本。从原则上讲，数据块的检索可以被视为一个功能单元，在这个单元中，存储者充当服务提供者，请求者充当消费者。提供者向消费者提供服务，而消费者应提供相应的补偿。这种直接交易通常需要交易双方互相知晓身份，因此如果要保持下载的匿名性，就必须以一种新的方式来构建补偿机制。

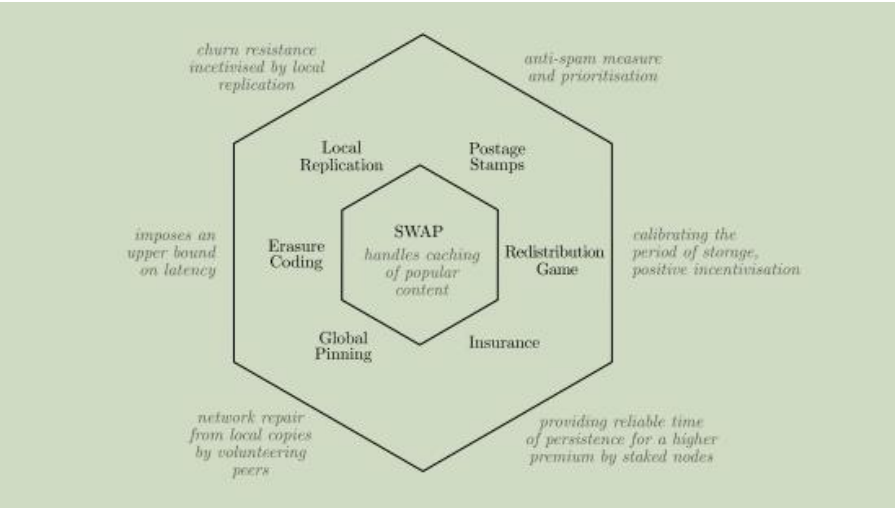

图 3.1：激励设计

由于使用了转发型 Kademlia，数据块检索包含一系列由转发节点执行的转发操作。由于这些节点是独立的个体，因此有必要对每次转发行为单独进行激励。重要的是，如果只有转发行为本身才算数，那么交易行为仅限于连接的节点之间，无论计费和补偿的细节如何（参

见 3.2.1）。考虑到已连接节点集在各次会话中形成了一个准永久集合，因此可以将这些交互框架置于“重复交易”的背景下。在这种情况下，参与各方会更倾向于保持良好的行为记录。优先与历史记录无污点的节点进行交互是合理的。此外，由于已连接节点集的数量与网络规模呈对数关系，因此与节点进行的重复交互所需的账目管理或区块链合约数量是可控的，提供了一种可扩展的解决方案。反过来可以说，与有限数量的节点保持余额记录，以及请求来源的模糊性，正是节点选择精简 Kademlia 桶连接数的原因。

为回传响应收费

如果接受检索请求就已经为转发节点带来收益（即在响应传回之前触发了对下游节点的记账事件），那么就会产生不转发请求的反向激励。最自然的解决方案是将请求收益与成功检索挂钩：仅在所请求的数据块被传回请求者时才触发记账事件（参见图 3.2）。

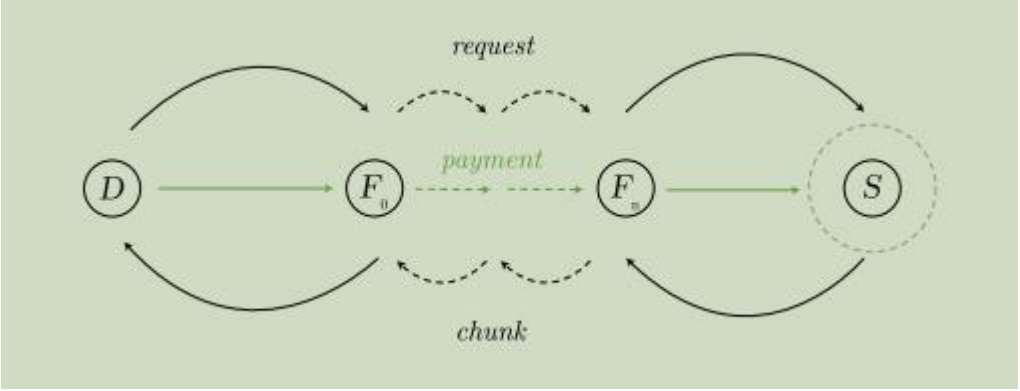

图 3.2：激励检索。节点 D（下载者）向数据块的地址发送检索请求。检索使用转发机制，因此请求通过转发节点 F0, ..., Fn 依次传递，直到到达最接近数据块地址的存储节点 S。

数据块通过相同的路径传递回下载者。接收到数据块响应会触发一个记账事件。

然而，如果发出请求无需任何成本，那么就可能出现大量针对不存在的数据块（随机地址）的非法请求。这可以通过对发送过多无效请求的节点进行制裁来轻松解决（参见 3.2.5）。

一旦节点发起（或转发）请求，就承诺在数据块在定义的生存时间（TTL）内传回时为其支付费用。因此，当数据块被传回时，节点没有理由阻止及时传递。此外，这种承诺还可以阻止节点无谓地向过多的节点请求同一数据块，因为如果多个节点同时传回数据块，则每一个响应都需要支付费用。

3.1.2 数据块检索的定价协议

接下来，我们描述节点在 Swarm 网络中用于传达数据块交付价格的协议。在该协议基础上，节点可以实施自己的策略，以在服务质量和价格方面与其他节点进行市场竞争。

价格发现

该协议的主要优点是允许基于本地决策的价格发现机制，这对于以下原因至关重要：

全球带宽成本差异：允许节点通过其定价反映成本结构，有助于在价格和质量上进行竞争，最终使终端用户受益。

带宽需求波动：由于使用情况或连接状况的变化，带宽资源需求始终在不断变化。

快速响应变化：能够直接响应变化，从而创建自我调节系统。

实际上，如果没有这种机制，当成本上升时，节点运营者可能会选择关闭节点；反之，当成本或需求降低且节点之间缺乏竞争压力时，终端用户可能会在较长时间内支付过高费用。

带宽是一种提供“即时满足”的服务，因此对其成本的即时确认和记录是合理的。由于

很难设想带宽资源供需总体情况中存在外部因素或非线性情况，因此需要一种具备以下特征的定价机制：

高效且即时的信号传递；

具有最小切换和发现成本的竞争选择。

这种机制最有可能促成实现全球最优资源配置的策略。

协议设计

为了实现这一点，我们引入了一种用于向上游节点传达价格的协议消息。可以将该消息视为请求的替代响应。节点会为每个接近度等级记录与各个对等节点相关联的价格。因此，当节点发出检索请求时，只要下游节点在生存时间（TTL）内成功交付有效的数据块，该节点就已经清楚需要支付的价格。

然而，限制价格信号仅用于响应是没有意义的：如果某个节点出于某种原因决定更改价格，双方都有兴趣交换此信息，即使此时并未发出请求响应。为了防止通过发送大量价格变更消息对上游节点发起拒绝服务（DoS）攻击，价格消息的发送频率被限制。

合理的行为

行为良好且定价有竞争力的节点会受到其对等节点的青睐；如果某节点的价格设置过高或价格波动远高于网络中其他节点，对等节点将不愿向其请求数据块。

为了简化讨论，我们假定默认价格为零，即免费服务（利他策略）。

不同接近度的差异化定价

如果在所有接近度范围内，数据块的价格相同，那么节点转发请求的唯一动机就是缓存数据块并通过转售获得收入。然而，这种做法很难在处理新数据块时合理化，尤其是当这些数据块位于节点较浅的接近度范围内（此时数据块被请求的概率较低）。更重要的是，如果数据块在所有接近度范围内的定价都是统一的，那么串通的节点可以生成数据块流量，并将发送与接收的费用保持一致，从而实现近乎“零成本”的拒绝服务（DoS）攻击（见图 3.3）。

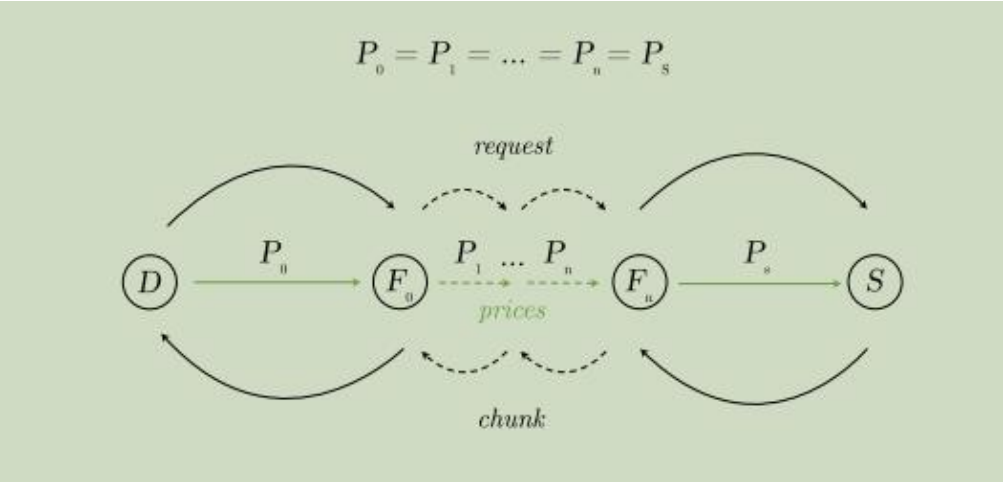

图 3.3：跨距离的统一区块价格可能导致 DoS 攻击。攻击者可以通过向节点 S 发送只有 S 能够服务的检索请求，从而在节点 D 和 S 之间创建流量。如果跨距离的价格相同，这种攻击对攻击者来说将不会产生任何成本。

为了缓解这种攻击，数据请求者为数据块支付的价格需要严格大于存储节点在请求路由过程中获得的补偿金额。因此，我们需要建立一种激励中继节点的定价方案，这就要求根据节点的接近度实施差异化定价。如果节点距离数据块越远，交付价格越低，那么请求总是可以沿该路径发送，因为中继节点能够赚取价差并获得利润。这意味着一个有效的差异化方案将收敛到这样一种定价模式：节点离数据块地址越远，交付成本越高，即数据块交付奖励是接近度的递减函数。

节点路径的竞争性影响

由于沿着交付路径和邻域中的竞争压力，我们预期节点在下游价格上的差价将趋近于每次转发的边际成本。下游价格由节点的分箱密度决定。假设每个分箱是平衡的，包含 2n2^n2n 个节点，则节点可以在一次跳跃中将接近度增加 n。同时，这也意味着节点可以将成本分摊在 n 个接近度分箱中，从而降低总体价格。

同一接近度范围内的价格一致性

假设节点 A 需要转发一个数据块请求，该请求落在 A 的接近度分箱 n 中。需要注意的是，A 的所有其他 n + 1、n + 2 等分箱中的对等节点也拥有接近度为 n 的数据块。如果这些对等节点中有节点 B 对接近度 n 的定价比 A 便宜，A 可以降低其对接近度分箱 n 的定价，将增加的流量全部转发给 B 并赚取价差（见图 3.4）。需要注意的是，这种做法会引入不必要的跳跃，从而在路由中增加了无效转发。

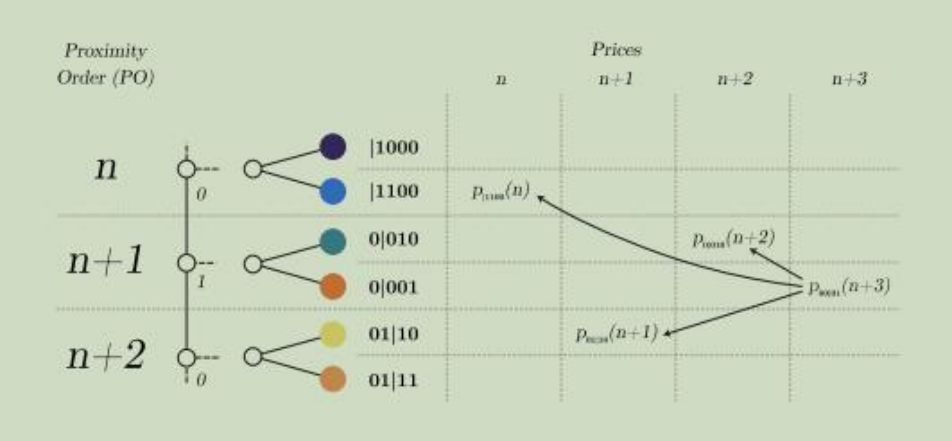

图 3.4：价格套利。节点为每个对等方的每个邻近顺序的价格维护一个价格表。图中显示节点 0101 试图转发一个针对 0000 的检索请求。箭头从最近的节点发出，指向其他对等方虽然离数据块较远，但提供更便宜的转发选项的单元格。选择更便宜的对等方将引导流量远离价格过高的对等方，并对两者施加调整压力。

同样，如果 A 在较浅分箱中的对等节点对其所在分箱的定价更低，例如，位于 n - 1 分箱的节点 B 定价低于 A 的 n 分箱定价，则 A 也可以将请求转发给 B 并赚取价差。

定价策略的调整

假设所有对等节点的价格表随着接近度的降低单调递减。同时，假设对于小于 n 的分箱，价格较高，而对于大于 n 的分箱，所有较深的对等节点在接近度 n 处的价格相同。假设 B、C、D 和 E 是 n 分箱中均衡分布的节点。A 想将数据块转发给一个对等节点，以便将其目标地址的接近度提升 3。如果 B 和 C 试图串通提高将数据块转发到 n + 3 分箱的价格，它们仍会受到 n + 2 分箱中 D 和 E 定价的约束，尤其是当 D 和 E 的定价低于 B 和 C 时。

这种价格差异为节点提供了套利机会：选择将请求转发给价格最低的对等节点的策略会将流量从高价节点引导至低价节点，从而促使价格进行调整。

综上所述，通过这种价格套利策略可以实现以下目标：

同一接近度范围内的价格一致；

价格随接近度的减小线性下降；

节点能够增加连接数量并保持较低的价格。

通过这种方式，激励机制的设计使得对单个节点有利的策略能够整齐地与有利于整个系统健康运作的目标保持一致。

分箱密度

根据下游对等节点与数据块的接近度来定价，会产生一个重要结果：一次非本地交付请求的净收入是数据块与自身节点接近度和请求转发目标节点接近度差值的单调递增函数。换句话说，在一次转发请求中，覆盖的距离越大，赚取的收益就越多。

这种激励机制符合下载者希望减少跳跃次数的利益，从而实现更低延迟的传输和减少带宽开销。该方案激励节点保持尽可能深的无间隙、平衡的 Kademlia 分箱地址集合（见图 2.4），即保持高密度的 Kademlia 分箱优于稀疏分箱。

能够保持高密度分箱的节点，其成本与低密度节点相同，但因为减少了跳跃次数，提升了延迟表现和效率，使其相比同价位的对等节点更具吸引力。这种流量的增加最终可能导致带宽竞争，从而推动价格上涨。

需要注意的是，这种套利行为在较浅的分箱中更高效，因为可选的对等节点数量更多。这与位于责任区深层的分箱形成鲜明对比。如果节点不复制邻域中的数据块，有些数据块需要从距离更远的节点请求回来，这样操作只能导致亏损。在 3.4 节中讨论了激励邻居节点复制其责任区数据块的方案。然而，一旦存储了责任区数据块，节点可以自由设置其价格。

缓存与自动扩展

节点每次提供数据块时都会获得奖励，因此数据块的收益与其受欢迎程度成正比：请求次数越多，相较于每单位时间的存储固定成本，其收益就越高。当节点达到存储容量上限并需要决定删除哪些数据块时，一个理性的节点会选择删除收益最低的数据块。而最近请求时间是收益低的数据块的一个合理预测指标。

为了最大化可供选择的数据块集合，节点会进行机会性缓存，将它们转发和同步的数据块缓存下来。这使得受欢迎的数据块分布更广，响应速度更快，从而将整个 Swarm 网络转化为一个自动扩展、自动平衡的内容分发网络。

非缓存节点

任何确保中继节点盈利的方案都会为仅进行转发、不进行缓存的节点进入网络提供正向激励。然而，这类节点本质上并不对网络有益，因为它们增加了不必要的带宽开销。一方面，这些节点的存在原则上可以减轻存储节点的中继流量负担，因此在浅分箱中使用它们可能无害。另一方面，靠近邻域深度时，同行节点会更倾向于选择缓存/存储节点，而非这些非缓存节点，因为在其假定责任区内，这些非缓存节点不具备存储优势。

非缓存节点还可以增强网络的匿名性（参见 2.3.1）。

3.1.3 激励推送同步

推送同步（参见 2.3.2）协议确保上传到网络的数据块到达其指定地址。接下来，我们将解释如何激励转发行为。

推送同步类似于检索协议，在消息交换的顺序上，两者的路线是相同的。推送同步协议中的数据块交付类似于检索请求，而推送同步中的接收声明则类似于检索中的数据块交付响应。

原则上，推送同步可以不需要明确的转发激励。由于检索协议，节点期望数据块能够在其地址的邻域中找到，因此，Swarm 中的参与者至少在某种程度上被激励帮助将上传的数据块传递到目标地址。然而，我们需要考虑到，数据块可能是通过距离目标更远的节点上传的，而不是直接由请求者上传（如轻节点或重试节点）。因此，如果推送同步不收费，节点可能会消耗大量带宽。

仅对下游节点进行推送同步交付的收费，会使得转发节点处于与存储节点就数据块交付进行讨价还价的位置。持有数据块对潜在的存储节点来说非常有价值，因为系统中也有针对

存储的奖励机制（见 3.4）。基于这一点，理论上，转发节点可以在存储节点支付的价格高于持有该数据块的价值时，选择保留数据块，考虑到存储激励所带来的潜在利润。特别地，由于从上传者开始的转发节点数量较少，存储奖励机制所产生的任何利润都可能被这些转发节点所获取。

然而，在推送同步中，通过将接收声明设为付费消息，角色发生了转变。转发节点不再处于讨价还价的位置。为了理解原因，我们考虑一个场景：假设转发节点尝试保留数据块以便与存储节点谈判价格。此时，上传者将无法在预期的时间内收到接收声明。结果，上传者会认为尝试失败并通过不同的路线重新上传数据块。此时，原先的转发节点被迫与其他转发节点竞争，以获得其带宽成本的补偿。由于所有转发节点都意识到这一动态，网络中的自发行为将促使一系列愿意以相对较小的补偿和带宽成本将数据块转发给存储节点的对等节点。这消除了原始转发节点与存储节点讨价还价的需求：相反，它们可以通过简单地返回接收声明立即赚取小额利润。

这一方案突出了两个协议激励机制基于相同前提的原因：对于一种同质化的商品（接收声明），存在多个卖方（转发节点）和唯一的买方（上传者）。因此，服务的价格（将数据块交付给存储节点）由沿路每个节点的转发边际成本之和决定。同时，存储节点可以从存储补偿机制中捕获所有利润。

通过这种方式，我们可以确保：(1) 存储节点实际响应并提供接收声明，(2) 有办法检测超时或未经请求的接收响应，以防止拒绝服务攻击（DoS），见图 3.5。

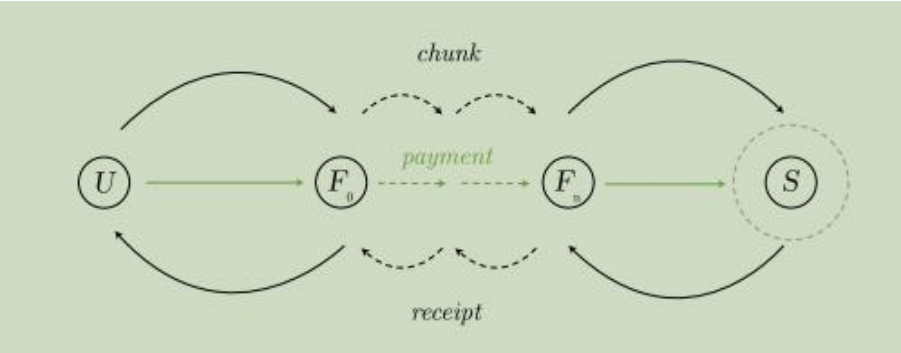

图 3.5：推送同步协议的激励机制。节点 U（上传者）将数据块发送至其地址，最近的节点是节点 S（存储者），通过转发节点 F0,...Fn。存储者节点会返回一份保管收据声明，该声明通过相同的转发节点 Fn,...F0 传回给上传者。接收到保管收据声明会触发一个记账事件。

与检索协议类似，这一方案中的定价预计会根据不同的邻近性而有所变化（参见 3.1.2）。此外，由于网络中节点的成本会波动（取决于容量利用率和节点效率），定价也会随着时间变化。鉴于补偿是在会计过程中为一个数据块和一条简短的消息（检索请求和接收声明）计算的，我们可以得出结论，两个协议的转发定价结构是相同的。因此，可以对两个协议应用统一的转发定价方案，如 3.1.2 中所讨论的。推送同步与检索协议的区别在于，在检索中，数据块会被交付回并且其完整性可以得到验证，而推送同步中的会计事件是接收声明，接收声明是可以伪造的。由于转发激励，节点可能会被激励延迟转发并伪装成存储节点，通过发出接收声明来进行欺骗。这使得通过备用路线查询（检索）数据块变得更加可取。如果这种检索尝试失败，可能需要尝试通过备用路径进行推送同步。

3.2 交换：会计与结算

本节讨论与带宽共享相关的激励机制。在 3.2.1 中，我们介绍了一种机制，用于跟踪对等节点之间的数据流量，并提供对等节点间的消息转发会计。随后，在 3.2.2 中，我们描述了补偿不平衡服务的条件，并展示了如何实现结算。特别是，我们介绍了支票和支票簿合同的概念。在 3.2.3 中，我们讨论了豁免作为一种优化手段，可以在交易成本上实现额外节省。在 3.2.4 中，我们讨论了如何通过激励服务发送现金交易，启用零现金进入 Swarm，最后，在 3.2.5 中，我们深入探讨了一组基本的制裁措施，它们作为激励，促使节点遵循协议并友好地互动。

3.2.1 对等节点会计

Trón 等人 (2016) 介绍了一种对等节点会计协议，称为 Swap。Swap 是一种以牙还牙的会计方案，能够扩展微交易。该方案允许直接连接的对等节点交换支付或支付承诺。该系统的关键特征通过 SWAP 缩写的不同记忆法得以生动体现：

Swarm 会计协议

Swarm 用于记录带宽互换的协议。

服务需求与提供

允许服务交换。

通过自动支付结算

当支付阈值超出时，发送支票。

发送豁免作为支付

债务可以通过未兑现的支票的价值来豁免。

无需一分钱即可开始

单向交换支持零现金进入。

服务交换

Swap 允许连接的对等节点之间进行服务交换。当时间上的消费量保持平衡且波动较小，双向服务的账户可以无需任何支付就进行结算。数据转发就是这种服务的一个例子，这使得 Swap 非常适合在内容交付或网格网络中实施带宽激励机制。

通过支付结算

在服务消费量存在较大波动或不平衡的情况下，最终余额将会显著偏向某一方。在这种情况下，负债方会向债权方支付一定金额，以恢复名义上的平衡。这个过程是自动化的，证明了 Swap 协议的“通过自动支付结算（余额）”的概念（见图 3.6）。这些支付可以是承诺支付，而非即时交易。

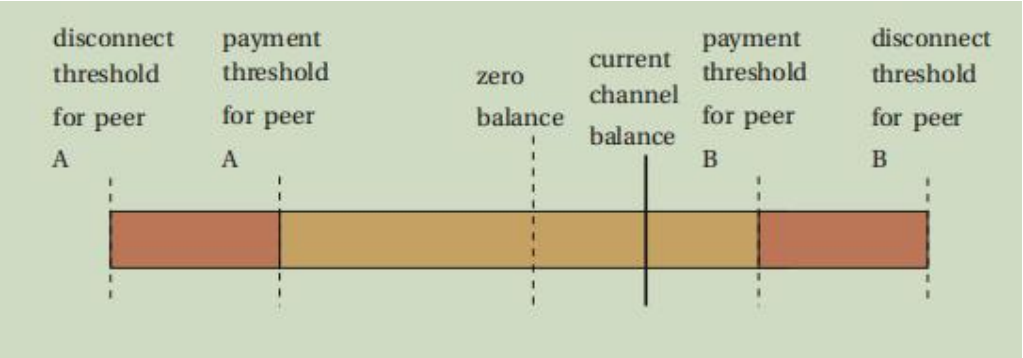

图 3.6：交换余额和交换阈值。中间的零余额表示服务的消费和提供相等。当前通道余额表示未补偿服务提供的差异：如果余额在零的右侧，则倾向于 A，B 处于债务状态；如果在左侧，则倾向于 B，A 处于债务状态。橙色区间表示损失容忍度。当余额超过支付阈值时，负债方会向其对方发送支票。如果达到断开阈值，负债方将被断开连接。

支付阈值

为了量化什么构成“显著偏移”，Swap 协议要求对等节点在握手过程中发布一个支付阈值：当它们相对于对等节点的债务超过这个阈值时，会向对方发送包含支付信息的消息。当债务达到这个水平时，任何节点发送消息是合理的，因为此时还会有断开连接的阈值。断开连接的阈值可以由任何节点自由设置，但建议选择一个值，以考虑两个节点之间常见的会计余额波动。可以通过考虑支付阈值和断开连接阈值之间的差异来完成这一设置。

原子性

发送支票并更新接收方余额无法做到原子操作，因为这会增加大量复杂性。例如，在接收并处理消息之间，客户端可能会崩溃，因此即使发送方没有遇到错误返回，发送方也无法确定支付是否已被接收，这可能导致双方会计上的差异。支付阈值和断开连接阈值之间的差异（DisconnectThreshold − PaymentThreshold）提供了对这种情况的容错保护，也就是说，如果这种崩溃发生的频率较低，并且在两个节点之间发生的概率大致相同，那么由此产生的微小差异将被过滤掉。这个机制可以防止节点面临制裁。

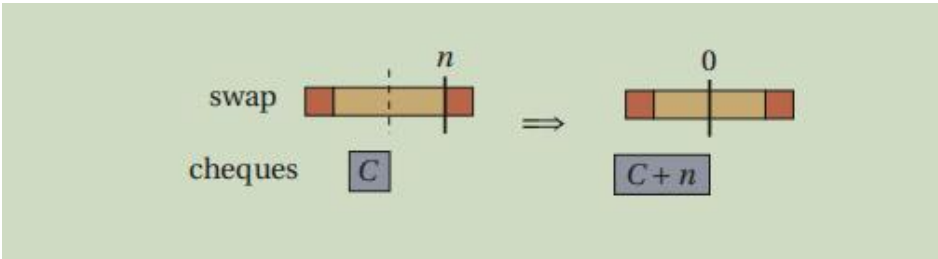

图 3.7：对等节点 B 的交换余额（相对于 A）达到支付阈值（左），B 向对等节点 A 发送支票。B 保留支票并将交换余额恢复为零。

3.2.2 支票作为链外支付承诺

在区块链网络中进行直接链上支付的主要挑战之一是每个参与节点处理每笔交易时所产生的高交易成本。然而，创建一个支付而无需在链上呈现该支付是可能的。这类支付被称为二层支付策略。其一种策略是延迟支付并批量处理。作为降低成本的交换，受益方必须愿意承担更高的结算失败风险。我们认为，这在 Swarm 的带宽激励机制中是完全可接受的，因为在这种机制下，对等节点会进行重复交易。

支票本合约

Tr ó n 等人（2016）提出了一种简单的智能合约——支票本合约，允许受益方决定支付的时间。这种合约充当一个钱包，能够处理其所有者发出的支票。类似于传统的金融交易，支票的发行人会签署支票，指定受益人、日期和金额，并将其作为支付承诺的凭证交给收款人，承诺在以后某个时间支付。智能合约则充当银行的角色。当收款人希望获得支付时，他们可以通过将支票提交给智能合约来“兑现支票”。合约在验证支票上的签名、日期和金额后，将相应金额转入受益人的账户（见图 3.8）。类似于人们带着支票去银行兑现，任何人都可以在交易中发送数字支票到所有者的支票本账户，启动转账。

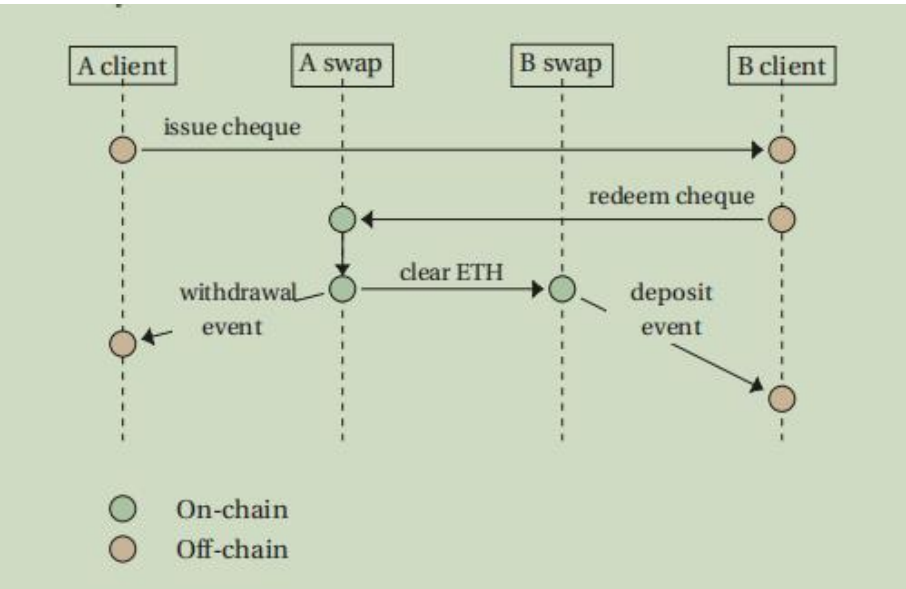

图 3.8：交换支票簿的基本交互序列

Swap 协议规定，当支付阈值被超过时，债权方会通过发送支票的方式进行支付。这些支票可以立即通过发送到发行人支票本合约来兑现。或者，支票也可以保留，这实际上是一种信用贷款形式，使交易方能够节省交易成本。

存入支票本的钱（全局余额）作为支票的担保，并被池化至所有未清支票的受益人之间。在这种最简单的形式中，支票本提供了与现实世界支票相同的保证：没有保证。因为资金可以随时从支票本钱包中自由转出，所以无法保证兑现时的偿付能力：如果支票本的余额少于提交的支票金额，支票将无法兑现。这是交易成本与结算失败风险之间的权衡。

严格来说，支票本并不提供偿付能力的保障，也没有明确的惩罚措施应对破产情况，但支票被退回可能会对发行人的声誉造成负面影响，因为支票本合约会记录这类事件。在支票交换发生在重复交易的前提下，节点会避免超出其余额发行支票。换句话说，节点维护良好声誉的利益足以成为维持其偿付能力的激励。

双重兑现

由于这些数字支票是文件，因此可以被复制，因此实施防止支票被重复兑现的措施至关重要。通过为每张发给特定受益人的支票分配一个序列号，并在支票兑现时将序列号存储在合约中，可以防止“重复兑现”。支票本合约可以依赖这个序列号来确保支票按顺序兑现，因此每个受益人只需要存储一个序列号。

另外，为了应对对同一受益人进行重复支付，支票可以包含该受益人所收到的累计总金额。合约会记录每个受益人已经兑现的总金额，当提交新的支票时，合约会将支票上的金额与存储的总金额进行比较。金额小于或等于存储总额的支票会被忽略，而金额大于存储总额的支票将导致差额转账给受益人。

这一简单的技巧还使得批量兑现支票成为可能，因为只需要处理最新的“最后一张支票”，从而显著减少交易成本。

无需以以太坊兑现支票

并非所有 Swarm 网络中的节点都能够提供支付交易费用所需的以太坊。支票本合约允许第三方兑现支票。交易的发送方将因执行服务而获得奖励。

3.2.3 放弃权

如果 swap 通道的不平衡是由于高度波动而非不平等消费所致，经过一段时间的支票积累后，通道的余额开始向相反方向倾斜。通常，这时由另一方向其对等方发行支票，从而导致双方都积累了未兑现的支票。为了进一步节省交易成本，可能希望能够将这些支票相互抵消。

这种过程是可行的，但它需要在支票本合约中进行一些重要的修改。特别是，支票的兑现不能再是即时的，而必须引入一个安全延迟，这一概念在其他支付通道实现中已经很常见（Poon 和 Dryja 2015，Ferrante 2017，McDonald 2017，Tremback 和 Hess 2015）。

让我们设想一个与支票退回发行人类似的系统。假设 A 节点向 B 节点发出了支票，并且余额已恢复为零。稍后，余额倾向于有利于 A，但 A 向 B 发出的支票尚未兑现。在传统金融世界中，B 可以简单地将最后一张支票退还给 A，或者证明销毁了该支票。在我们的案例中，情况并不那么简单；我们需要一种机制，使得 B 承诺不兑现那张特定的支票。这种承诺可以有多种形式；它可以通过 B 签署一条消息，允许 A 向其发出新的“最后支票”，其累计总额低于之前的金额，或者 B 可以向 A 的支票本发出一种“负”支票，实际上抵消该金额，就好像已经支付了一张相同金额的支票。

这些实现方式有一个共同的特点，那就是不允许即时兑现支票。当接收到兑现支票的请求时，合约必须等待，直到另一方提交关于已取消支票或减少金额的可能缺失信息。为了适应使用单一支票本的（半）双向支付，我们引入以下修改：

用户 A 向用户 B 发出的所有支票必须包含一个序列号。

用户 A 向用户 B 发出的每张新支票的序列号必须高于上一张支票的序列号。

A 的支票本合约记录 B 兑现的最后一张支票的序列号。

在兑现延迟期间，任何有效的序列号更高的支票会替代之前提交的支票，无论其面额如何。

任何减少先前提交支票支付金额的支票，只有在受益人签署的情况下才有效。

有了这些规则，支票取消的过程就变得容易理解了。假设用户 A 向用户 B 发出了支票 $c_0 \cdots c_n$，累计金额为 $t_0 \cdots t_n$。假设 B 兑现的最后一张支票是 c_i。支票本合约已经记录 B 已经收到金额为 t_i 的支付，并且最后兑现的支票的序列号为 i。

假设余额开始向 A 一方倾斜，倾斜的金额为 x。如果 B 已经兑现了支票 c_n，那么 B 将需要使用 B 的支票本发行一张支票，以 A 为受益人。然而，由于支票 c_{i+1} 到 c_n 仍未兑现，B 可以选择向 A 发送一张支票，支票的来源为 A 的支票本，B 为受益人，序列号为 n+1，累计金额为 $t_{n+1} = t_n - x$。根据上述规则，A 会接受这张支票作为 B 支付的等值金额 x。在这种情况下，B 并没有直接向 A 发送支票，而是放弃了之前的部分应得款项。这就验证了 SWAP 的概念——将放弃权作为支付手段。

这一过程可以重复多次，直到累计金额恢复到 t_i 为止。此时，所有未偿还的债务实际上已被取消，任何进一步的支付必须通过 B 的支票本向 A 发出正式的支票（见图 3.9）。

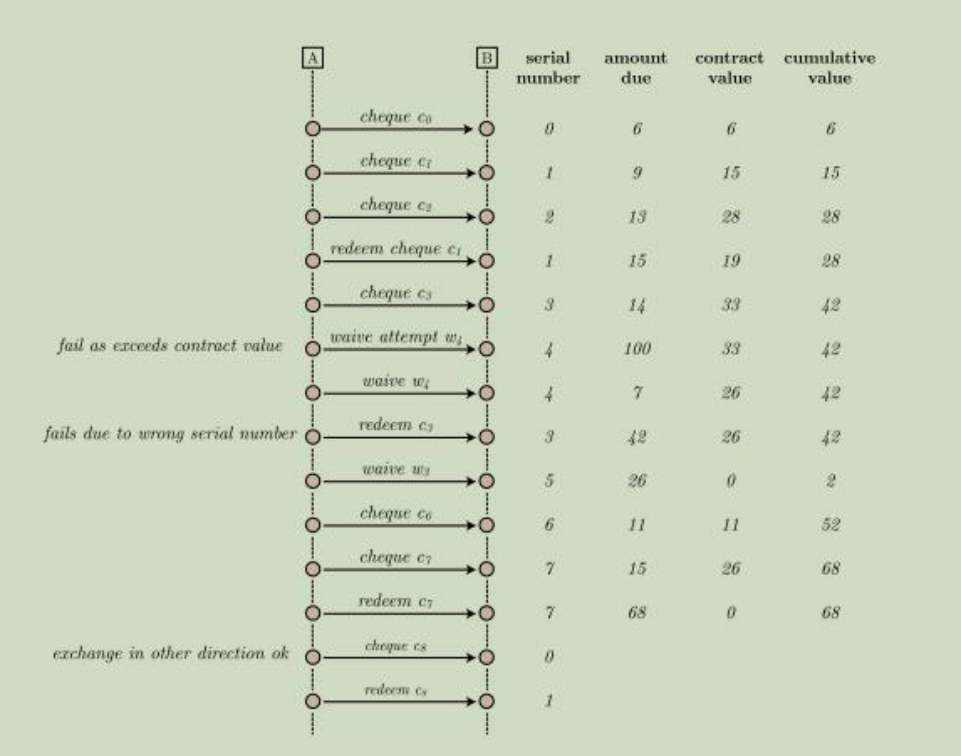

图 3.9：混合支票和豁免交换的示例序列

3.2.4 零现金入场

Swap 会计也可以单向运作。当一方以零流动资金（新用户）进入系统，但连接到拥有资金的节点（老用户）时，新用户可以开始提供服务（而不使用任何服务），从而赚取正的 swap 余额。

如果老用户有支票本，他们可以直接用支票支付新用户。然而，这有一个前提条件：新用户能够凭借提供的服务赚取支票，但无法兑现这些支票。兑现支票需要向区块链发送交易，因此需要支付 gas 费用，除非节点能够说服其对等节点代为执行该交易。为了简化这个过程，节点可以签署他们希望发送的结构，然后扩展 Swap 合约，添加一个预处理步骤，触发支付给新用户，支付的金额包括交易的 gas 费用以及支付交易的发送者的服务费用。新用户的支票可以由任何老用户兑现（见图 3.10）。这个功能验证了 SWAP 的概念——从零开始，无需资金，通过对等节点发送。

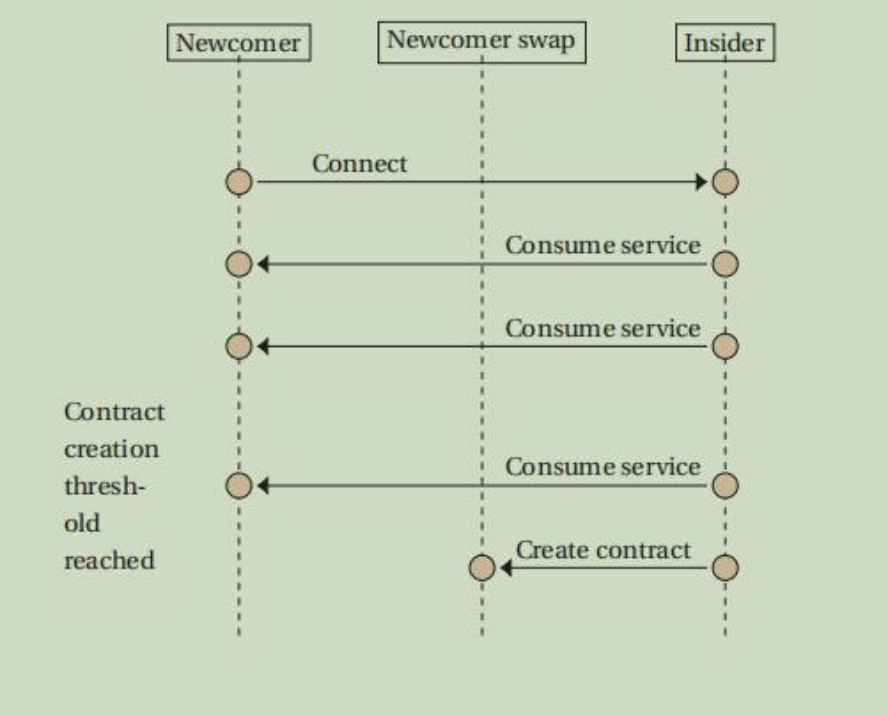

图 3.10：引导过程或如何作为一个能够交换的节点启动，消费和提供服务并赚钱。

无需启动资金就能赚取少量资金的可能性至关重要，因为它为新用户提供了不需要购买代币即可进入 Swarm 的方式。这一优势也扩展到了以太坊生态系统：通过使用 Swarm，任何人都可以赚取少量资金，以便开始支付 gas 费用来支持其 dapp，而无需在入场前经历购买代币的痛苦过程。

3.2.5 制裁与黑名单

本节通过额外的激励措施和防止恶意行为的保护机制，补充了 SWAP 方案。

协议违约

在点对点的无信任环境中，对不良节点行为实施细致的制裁可能会面临挑战。然而，当交互的基本规则被违反时，发现问题的节点可以简单地与该节点断开连接。为了避免因尝试重新连接而导致的死锁，针对违规节点施加的制裁还包括将该节点的地址记录到黑名单中。这个简单的措施足以为那些试图利用协议的节点提供明确的威慑作用。

过度无理请求

无论是检索协议还是推送同步协议，都具有一种激励机制，只有对请求的响应才能产生收益。尽管这种机制创造了强烈的激励让节点积极参与，但也有必要采取措施，确保节点不会以无意义的请求轰炸网络，这些请求没有实际的成本。在推送同步协议中，尤其重要的是不能允许数据块随意替换其他数据块而不付出代价。这将是后面章节中介绍邮票印刷概念的主题（见 3.3）。

对于拉取同步的检索协议，潜在的攻击包括请求不存在的数据块，并导致下游节点因请求持续存储而产生大量网络流量和一些内存消耗，尤其是在请求的 TTL（生存时间）期间。如果请求者无意中请求了不存在的数据块，且该数据块已经被垃圾回收，且请求者出于善意发起请求，那么这也可能是无心之举。

为了解决这一问题，每个节点都会记录来自每个对等节点的检索请求次数，并更新失败

请求的相对频率，即即使该节点已经转发请求，仍超时未完成的请求。如果失败请求占总请求的比例超过某一阈值，将对该节点实施制裁：断开连接并将其加入黑名单。

通过记住自己转发的请求，节点可以区分合法响应和潜在的拒绝服务攻击（DoS）：对于检索协议，如果交付的数据块无法满足一个未完成的请求，则视为无请求响应；对于推送同步协议，如果一项接收托管声明与已转发的数据块记录不符，则视为无请求响应。

超时在此非常关键。在请求的 TTL 期满后，未完成请求的记录可以被移除。此后收到的任何响应都将视为无请求响应，因为它与那些从未请求过的消息无法区分。

为了应对时间测量上的小幅差异，在断开连接并加入黑名单之前，再次容忍来自某个节点的小比例不合法消息。

服务质量

除了无请求消息的数量，节点还可能通过其他方式引起不满，比如设置过高的价格、网络吞吐量低或响应延迟长。类似于过度无理的请求，是否属于恶意攻击或是以良好信念提供的非最优（质量差、价格过高）服务之间不需要做区分。因此，缓解服务质量问题的讨论通常是在转发和连接的对等节点选择策略中进行的。

黑名单

黑名单是一种补充断开连接的策略，用于作为对对等节点的惩戒措施。它旨在延伸我们在断开连接时做出的判断，即该对等节点不适合继续合作。黑名单作为参考，在接受传入连接时以及在连接驱动器的对等节点推荐策略中起到作用。一方面，黑名单可以防止节点在恶意对等节点不断尝试重新连接时陷入死锁。另一方面，必须小心避免将善意行为的节点列入黑名单，因为这可能会负面影响网络的连通性。

3.3 邮票

邮票是与数据块相关的可验证支付证明，通过所有者的签名进行见证。它具有两个目的：通过施加预付成本来防止无意义的上传，并通过赋予数据块一个特定的 BZZ 数量来指示数据块的相对重要性。存储节点可以使用这些信息来优先保留和服务哪些数据块，并在容量有限时垃圾回收其他数据块。

在本节中，我们首先介绍邮票批次的概念，这允许批量购买邮票（3.3.1）。在 3.3.2 中，我们解释如何表示和强制执行限量发放。在 3.3.3 中，我们介绍储备的概念，并概述存储节点如何最大化利用储备的规则。最后，在 3.3.4 中，我们探讨储备容量、有效需求、节点数量及其对数据可用性的影响。

3.3.1 购买上传容量

上传者可以通过从以太坊区块链上的邮票智能合约中购买邮票批次来批量获取邮票。创建邮票批次涉及向合约的批次创建端点发起交易，交易中包括一定数量的 BZZ 代币和指定参数的交易数据。在交易处理时，邮票合约会注册一个新的批次条目，包含以下信息：

批次标识符

随机生成的 ID，用作此批次的引用。

批次深度

发行量的以 2 为底的对数，即可以使用此批次盖章的数据块数量。

所有者地址

根据随创建交易一起发送的交易数据，或如果未指定，则为交易发送者的以太坊地址，表明有权发放邮票的所有者。

每数据块余额

随交易发送的总金额除以发行量。

可变性

一个布尔值标志，指示如果批次的时间戳较旧，批次的存储槽是否可以重新分配给另一个带邮票的数据块。

均匀深度

存储槽被安排成的相等大小的桶的数量的以 2 为底的对数。

邮票合约为用户提供了修改批次每数据块余额的端点。这允许用户向批次中添加资金，以延长该批次发行的邮票的有效期（充值），或增加发行量以降低有效期（稀释）。虽然任何人都可以选择在稍后为批次充值，但只有所有者才有权进行稀释。

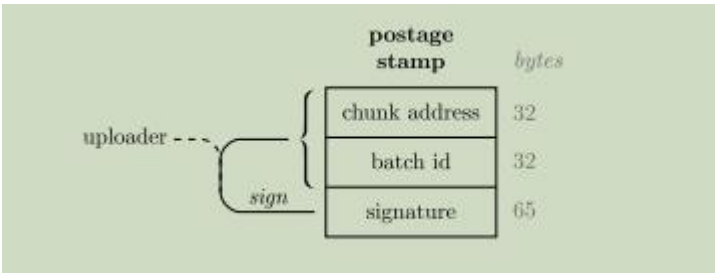

图 3.11：邮戳是一种数据结构，由邮戳合约批次 ID、存储槽索引、时间戳、数据块地址以及证明这四者关联的见证签名组成。上传者和转发者必须为每个上传的数据块附加一个有效的邮戳。

所有者发行邮票，以便将其附加到数据块上。每个批次都有若干存储槽，这些存储槽有效地被安排到若干等大小的桶中。发行邮票意味着将数据块分配给一个存储槽。邮票是一个数据结构，包含以下字段（见图 3.11）：

数据块地址

邮票附加到的数据块的地址。

批次标识符

引用发行批次的 ID（在创建时生成）

存储槽

引用批次中某个等大小桶的索引，并引用数据块被分配到的存储槽内索引

时间戳

数据块被盖章的时间。

见证

批次所有者的签名，证明存储槽与数据块之间的关联。

邮票的有效性可以通过验证以下五个属性是否都为真来检查：

authentic（真实）

批次标识符已在邮票合约的存储中注册。

alive（有效）

引用的批次余额尚未耗尽。

authorised（授权）

邮票由指定为批次所有者的地址签名。

available（可用）

引用的存储槽在基于批次深度的有效范围内，并且对于不可变批次，不存在重复。

aligned（对齐）

引用的存储槽具有指定的桶，并且与盖章的数据块地址对齐。

所有这些可以通过 Swarm 中的节点轻松检查，只需使用在公共区块链上的信息（邮票合约的只读端点）。当数据块被上传时，沿着推同步路线的转发节点会验证附加邮票的有效性（见图 3.12）。

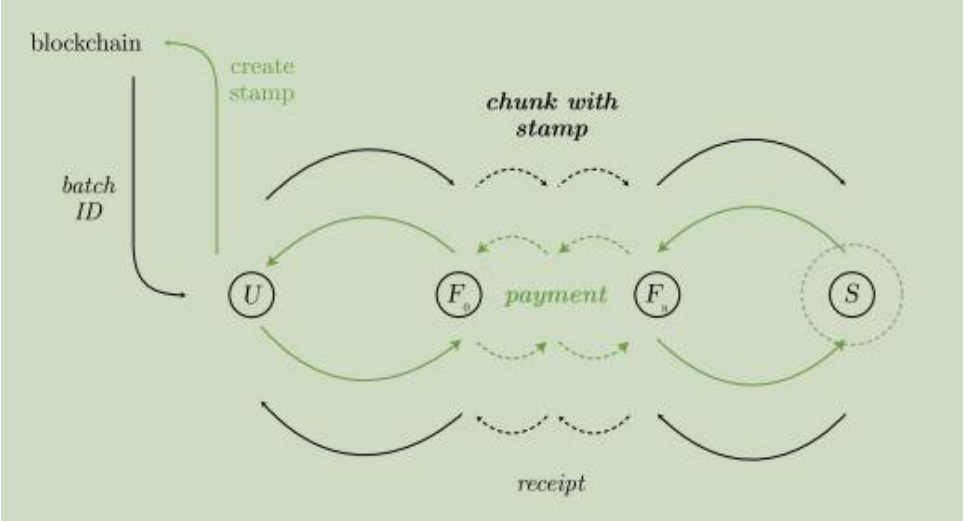

图 3.12：邮资邮票在区块链上批量购买，并在上传时附加到数据块上。它们沿着推送同步路径一起传递，其有效性由每个跃点的转发者进行检查。

规范化的每块余额计算

一个批次的规范化每块余额是通过将批次的入款金额除以批次在块存储槽中的大小来计算的。块余额被解释为预先承诺用于存储的金额。随着时间的推移，余额会减少，就像每个区块都按价格预言合约规定的价格支付存储租金一样。

这种存储预付款制度消除了对未来存储价格或货币汇率波动的猜测需求。通过降低过期日期的确定性，用户获得了对价格波动的韧性。此外，上传者可以通过占用更多的批次余额来享受“不参与”的奢侈；尽管这部分余额作为对抗价格上涨的抵押品，如果价格未上涨，资金仍然可以用于存储。

3.3.2 有限发行

购买一个邮资批次实际上使得拥有者有权按批次 ID 发行一定数量的邮票，这个数量称为发行量或批次大小。它被限制为 2 的幂，并且通过该数量的以 2 为底的对数来指定，称为批次深度。

批次中的存储槽被安排成桶，每个槽在其相应的桶内被分配一个索引。桶的数量也被限制为 2 的幂，并通过其以 2 为底的对数来指定，称为均匀性深度。为了确保批次大小的限制（批次深度 d 和均匀性深度 u），必须满足以下条件：
— 桶的索引范围从 0 到 2^u −1。
— 桶内的索引范围从 0 到 2^(d−u) −1。
— 没有重复的索引。

前两个条件可以通过任何第三方轻松验证，而最后一个则不能。为了使索引冲突能够被单个存储节点检测到，均匀性深度必须足够大，以便落在节点的责任范围内。只要满足这一条件，所有同一个桶中的块都会保证落在同一个邻域内，因此，节点可以在本地检测到重复分配（见图 3.13）。

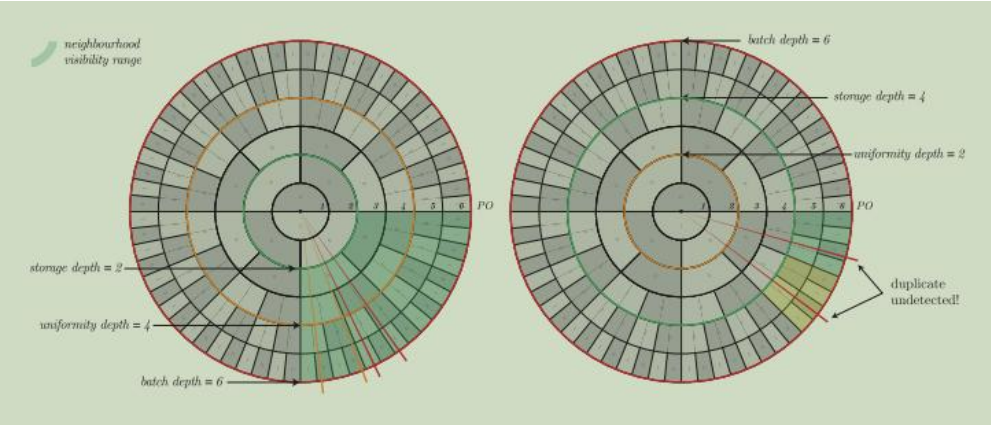

图 3.13：批次包含 2u 个大小相等的桶（u 为均匀深度，由橙色圆圈标记），每个桶包含相同数量的存储槽（2d−u），总计批次容量为 2d 个块（d 为批次深度，由红色圆圈标记）。存储槽被索引，每个索引通过邮票签名与一个块关联。只要桶的深度大于存储节点的存储深度（绿色圆圈），存储节点就可以在本地检测到邮票的过度发行，如左图所示。在这种情况下，它们将接收到所有正确分配到相关桶的块（橙色半径），并通过禁止超出范围（⩾ 2d −u）或多次分配的索引来正确识别冲突（红色半径）。相比之下，右图显示，对于 u = 2 的批次，存储深度为 4 的节点无法识别重复块。

为了保持邮票无冲突，上传者需要为每个批次中的每个桶维护计数器，记录他们为每个桶发行了多少邮票，并确保该数量不超过桶的最大容量。

一般来说，批次最有效的利用方式是将每个桶填满。持续的非均匀性（即目标性发行）会导致批次的低效利用，因此会使得上传和存储每个块的单价更高。这个特性有一个期望的副作用，即它对非均匀上传施加了前期成本：上传中块的分布越集中，邮资批次中的未使用存储槽就越多。通过这种方式，我们确保了针对邻域的目标性拒绝服务攻击（即在特定地址范围内上传不成比例数量的块）是代价高昂的，因为由于批次低效利用的惰性成本是随着偏斜的深度呈指数增长的。

除了拒绝服务保护外，邮票还可以作为一种信托信号，表明用户在 Swarm 中持久存储一个块的价值。

特别地，批次的每块余额可以提供差异化的先验偏差，决定在没有足够证据预测其盈利能力的情况下，哪些块应当被保护，免受垃圾回收的影响。

3.3.3 储备规则

储备是指为存储节点责任区域内的块而预留的一定大小的存储空间。只要储备中的块具有有效的邮票，它们就会受到保护，不受垃圾回收的影响。当批次过期，即它们的余额完全耗尽时，已标记的块将不再受到驱逐保护。它们从储备中被驱逐后，会释放出一些空间，这些空间可以容纳新的或更远的属于有效批次的块。

从激励机制的角度来看，同一相对距离顺序和同一批次中的块被视为等价的。对于由于批次过期而发生的驱逐，这些等价类（称为批次桶）将作为一个整体处理：批次桶中的块会一起从储备中被驱逐，并通过一个原子操作插入缓存。

假设有一个全局预言机来确定租金的单价，并且为节点规定了固定的储备容量，储备的内容将根据一组关于批次桶的约束进行协调，这些约束被称为储备规则：

1 如果某一相对距离顺序（PO）下的批次桶被预留，则所有较近相对距离顺序（较高

PO）的批次桶也应当被预留。

2 如果某一相对距离顺序（PO）下的批次桶被预留，则所有属于同一 PO 且每块余额更高的批次桶也应当被预留。

3 储备不能超过容量。

4 储备应该被最大化利用，即在遵守规则 1-3 的前提下，不能进一步扩展。

第一条规则意味着储备对于 PO 是向上封闭的，这表示全球优先存储离节点地址更近的块。这一机制通过路由激励：通过存储离节点地址更近的块，节点可以最大化它能发行的收据数量和响应的检索请求数量。此外，存储更近的块还能确保在邻域内更广泛的覆盖，即使邻域不再提供所需的冗余。

第二条规则表达了储备对于 PO 的约束，即每块余额按升序存储。这个约束反映了同一相对距离内块的次要偏好，优先选择那些使用余额更高的批次标记的块。这一激励机制来源于块所承诺的差异化绝对利润：由于余额不可撤销的限制，余额较高的块在过期时间上较晚，因此相比于较早过期的块，它们为存储者提供了更大的绝对利润，尽管在有效期内它们支付的租金相同。

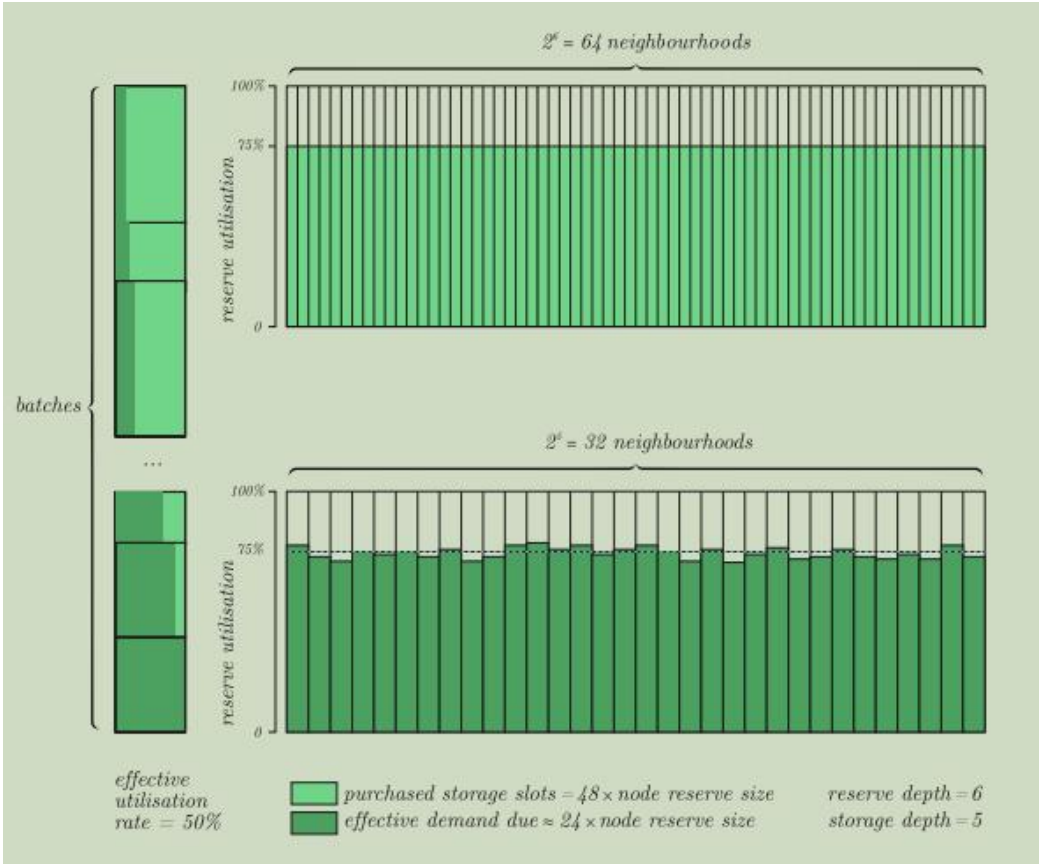

图 3.14：区块链上所有非零余额批次的总大小（左侧）表明了对块存储的潜在需求。存储此容量的邻里深度下限是预留深度（右上）。存储深度表示上传并存储在邻里预留中的块的有效体积（右下）。它们之间的差异是部分批次利用的结果。通过高效利用邮资批次来激励邻里间块体积的均匀性。

当一个新块通过拉同步、推同步或上传到达 Swarm 时，会验证附带的邮票的有效性。如果该块的 PO 低于批次深度，节点会将该块插入垃圾回收索引，表示它可能会被垃圾回收。另一方面，如果 PO 等于或大于批次深度，则该块被认为处于储备中。如果储备的大小超过了指定容量，则会识别出一些批次桶，这些批次桶的总大小足以覆盖超出部分。这些批次桶将从储备中被驱逐，从而减少储备大小，满足容量要求。

3.3.4 储备深度、存储深度、邻域深度

储备深度

存储在 DISC 中的块的潜在需求通过有效批次中存储槽的总数来量化。这一数值是所有未过期批次大小的总和。由于批次及其余额记录在邮票合约中，因此 DISC 的储备大小通过共识达成一致。

储备深度通过对 DISC 储备大小取以 2 为底的对数，并将结果向上取整为最接近的整数来确定。它表示最浅的 PO，在此 PO 下，非重叠的邻域可以共同容纳与总支付块数相对应的数据量。假设邻域内的节点具有固定的规定存储容量，用于存储它们的储备份额。

储备深度也作为拉同步的安全下限，即邻域需要同步的最远批次桶，以确保储备被存储。相反，如果任何由储备深度标记的邻域中没有节点，则 Swarm 没有正常工作，即有效邮票的块没有被保护，避免丢失（见图 3.15）。

存储深度

有效的存储需求对应于实际上传的块的总数。虽然储备中的每个块都与一个有效的邮票批次相关联，并且分配了一个存储槽，但一个邮票批次可能有未分配的存储槽。因此，实际存储在 DISC 中的块数可能少于 DISC 储备大小。

有效的责任区域由储备中最远批次桶的相对距离顺序标记，假设节点遵守储备规则。

节点的存储深度定义为最浅的完整桶，即储备中最远批次桶所对应的最低 PO。如果节点的储备中的最远桶不完整，则存储深度等于储备边缘 PO 加 1。

存储深度是拉同步的最优下限，即它表示节点需要与邻域同步的最远批次桶，以实现最大储备利用率。应当通过存储激励来鼓励最大化储备的利用。

实际存储深度与储备深度之间的差距存在，是因为邮票的批量购买。由于整个批次的邮票会预留存储槽，这些槽只会在块实际上传时才分配给块，因此批次的利用率可能大大低于 1。当所有批次都被完全利用时，存储深度和储备深度将相同。

邻域深度

Swarm 对本地复制有要求，规定每个由存储深度指定的邻域应至少包含四个节点。如果邻域由单个节点组成，那么该节点的故障将导致该节点责任区域中的块无法检索。若邻域中有两个节点，虽然能显著提高对临时故障的抗压能力，但由于连接延迟，双节点邻域仍然可能导致不稳定的用户体验。理想的情况是每个完整连接的邻域包含四个节点，这促成了以下定义：某节点的邻域深度是指最高的 PO d，使得由节点覆盖网络的 d 位前缀指定的地址范围内包含至少三个其他对等节点。

图 3.15 详细描述了三种深度的潜在相对顺序以及它们对 Swarm 健康、效率和冗余性的影响。

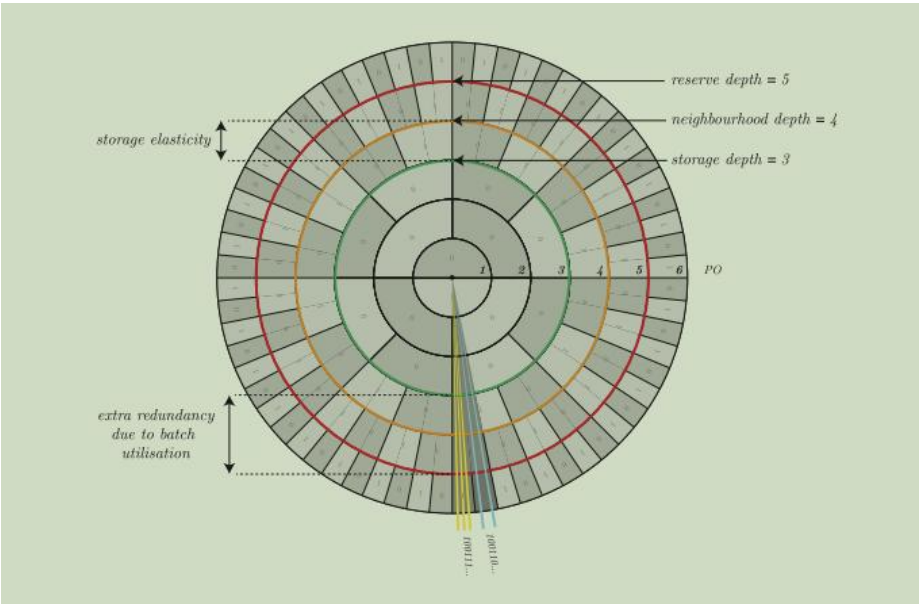

图 3.15：3 种深度（储备深度、存储深度和邻域深度）分别表示保留容量的数量级（潜在需求，红色圆圈）、上传分块的数量级（有效需求，绿色圆圈）和节点数量的数量级（有效供应，橙色圆圈）。它们可能的排序反映了对数据可用性有不同影响的各种场景。存储深度不能大于储备深度。存储深度与储备深度之间的差距量化了平均批次利用率。存储深度与更深的邻域深度之间的差距量化了存储的弹性：这一差异表示在冗余低于所需水平之前，有效容量可以翻倍的次数。虽然这种供应过剩可能是对需求增长的预期，但如果邻域深度长期保持比存储深度更深，则可能表明利润过高。相反的顺序则表示供应不足（冗余低于期望水平）。

3.4 公平再分配

Swarm 中的正向存储激励系统旨在将上传者在邮票合约中存入的 BZZ 代币重新分配给存储提供者。合约上的整体余额涵盖了奖励池，代表了特定时期内所有邮票批次的累计存储租金。存储租金必须以确保存储提供者的收益与其贡献成比例的方式进行再分配，权衡存储空间、服务质量和承诺期限。

再分配过程最好被看作是一个由区块链上的一套智能合约协同运作的博弈。节点通过参与存储和同步块赚取参与权，获胜者可以通过发起与游戏合约的交易来领取奖励。

在 3.4.1 节中，我们将再分配的概念表述为概率性支出，以便轻松证明公平性。随后在 3.4.2 节中，我们将概述再分配博弈的机制。3.4.3 节和 3.4.4 节解释了如何强制最大化专用存储的利用率，以冗余方式持久化相关内容。最后，在 3.4.5 节中，我们讨论了如何将博弈的某些方面解释为价格信号，使网络通过自动价格发现实现自我调节。

3.4.1 邻域、均匀性和概率性支出

在这一节中，我们论证了邮票批次的高效使用激励了均衡的块分布，从而导致邻域之间存储深度的均匀性。然后，我们解释了这一点如何通过概率性支出机制实现公平的再分配系统。

假设存在一个设定存储单价的预言机，那么可以计算出某一特定时期内某个批次的存储租金。一个批次的租金单位数是批次大小与指定时期内区块数的乘积。租金价格是通过将租金单位数乘以单价计算得出的。所有批次在两个支出周期之间的存储租金总和构成了这一回合的奖励池。

奖励池并不是定期在邻域之间分配，而是可以在每回合中将整个奖励池转移到某一个目标邻域中的（代表节点）。这一概率性支出方案保证了邻域之间的公平性，只要选择某个邻域作为目标的概率与其对整个网络存储的相对贡献成正比。考虑到固定的储备容量和邻域内节点对储备内容的复制，每个邻域（由存储深度定义）对网络的贡献是平等的。

在 3.3 节中，我们提到上传者有强烈的激励，要求他们以一种均匀分布的方式使用他们的邮票批次，以使得他们用这些邮票标记的块在地址空间中均匀分布。所有批次都遵循这一点，便创造了块在 DISC 中的均匀分布。特别地，共享相同前缀的块集预计大小大致相等。因此，我们预计节点将按相同的相对顺序填充其规定的储备容量，而不考虑它们在地址空间中的位置。换句话说，存储深度在节点之间是均匀的，因此在邻域之间也是均匀的。邻域深度相等时，邻域的均匀采样可以通过在地址空间中随机投放一个锚点（称为邻域选择锚点）来模拟（见图 3.16）。

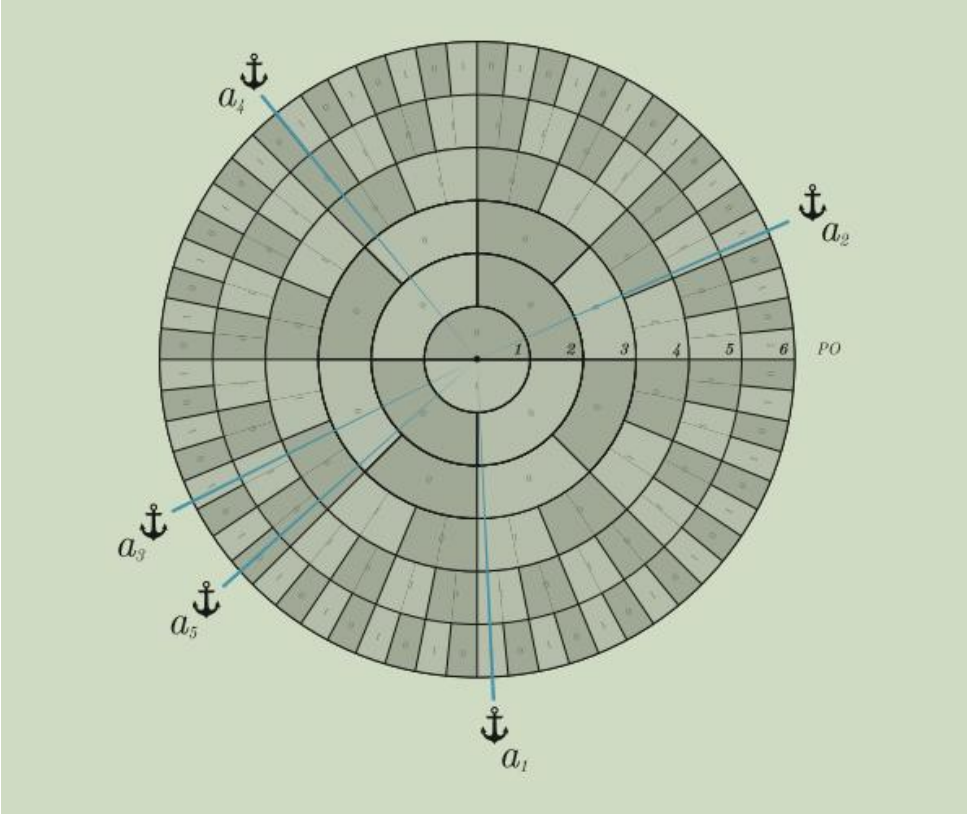

图 3.16：邻域选择和奖励池重新分配。获胜的局部区域由邻域选择锚点确定。在其存储深度内包含该锚点的邻域被邀请通过提交共识储备样本来申请。

3.4.2 再分配博弈的机制

为了网络的利益，节点之间协同努力将数据冗余存储，这一过程由一个 Schelling 博弈支撑，目的是证明节点一致地存储它们需要存储的块，并且确实执行了存储。再分配博弈由游戏合约协调，游戏合约是 4 个智能合约体系中的一个构件，这一体系共同推动 Swarm 存

储激励系统（见图 3.17）：

邮票合约

作为批次存储，向上传者销售邮票批次，跟踪批次余额、批次过期、存储租金和奖励池本身。

游戏合约

协调再分配回合，处理选定邻域中的存储提供者提交的承诺、揭示和索赔交易。

质押合约

管理质押登记，保持节点的承诺质押和质押余额，支持冻结和惩罚质押，及允许质押者提取多余余额。

价格预言机

维护存储租金的单价，根据供需关系接受来自游戏合约的信号动态调整，为其他三个合约提供当前价格预言机服务。

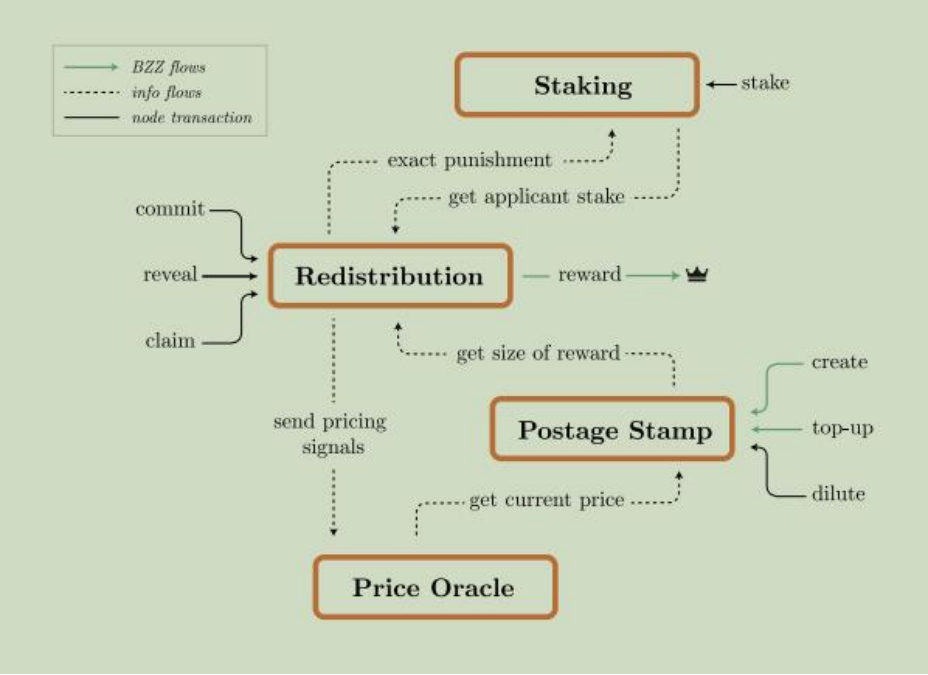

图 3.17：Swarm 存储激励智能合约的交互。图中用虚线展示了组成存储激励智能合约套件的四个合约之间的信息流，以及它们接受的公共交易类型。

博弈被构建为一系列回合。每回合持续固定数量的区块，并周期性地重复。每回合由三个阶段组成：承诺、揭示和索赔。这些阶段的命名反映了智能合约在每个阶段中期望的交易类型，节点需要在选定的邻域中提交这些交易（见图 3.18）。

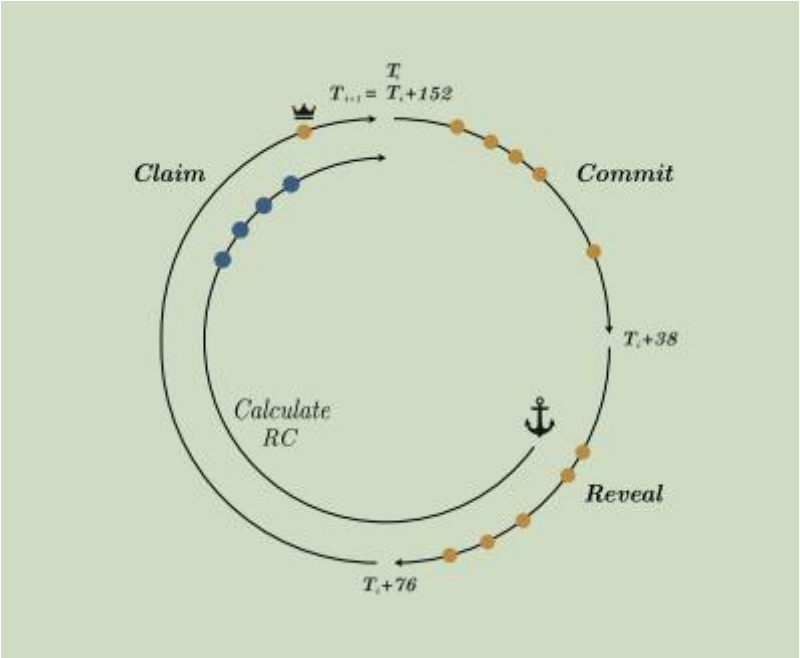

图 3.18：一轮再分配游戏的阶段。该图展示了再分配游戏重复轮次的时间线及其阶段。在智能合约交互的上下文中，逻辑上从提交阶段开始，随后是揭示和声明阶段。从客户端节点参与的角度来看，从揭示阶段结束并揭示邻域选择锚点开始，被选中的邻域中的节点开始计算其储备样本，并在下一个提交阶段结束前提交。如果它们被选为诚实节点且为获胜者，则会在声明交易中提交其权益证明。

一旦揭示阶段结束，邻域选择锚点就会被揭示。具有该锚点的邻域内的节点将有资格参与下一回合（见图 3.16）。

假设邻域内的存储节点在其储备块上达成共识，并向区块链提供证据，称为“权利证明”（将在 3.4.4 节详细讨论）。在这样的博弈中，每个节点的 Nash 最优策略是遵循协议，并与其他节点协调，确保所有邻居节点根据共享信息得出相同的权利证明。由于权利证明需要共识且不可窃取，因此必须使用承诺/揭示方案。

在承诺阶段，邻域内的节点将通过提交被任意密钥模糊化的储备承诺进行申请（该密钥将在后续揭示）。接收承诺交易的智能合约验证该节点是否已质押，即质押合约的注册表中是否包含该节点覆盖网络的条目，并且质押值高于最低要求的质押金额。

在揭示阶段，之前已提交承诺的节点通过提交包含其储备承诺、存储深度、覆盖地址以及用于模糊化承诺的密钥的交易来揭示其承诺。在接收到揭示交易后，合约会验证揭示的数据，在序列化后是否确实与其承诺哈希匹配。同时，还会检查该节点是否属于由邻域选择锚点指定的邻域，即是否在揭示中提供的存储深度内。

在索赔阶段，获胜的节点需要提交索赔交易。首先，为了确定 Schelling 博弈的结果，从揭示阶段提交的揭示中选择一个。被选中的揭示代表了真相；与之一致的应用集合代表了邻域中的诚实节点，而不同意的节点被视为骗子，未提交揭示或提交无效数据的节点则被视为破坏者。诚实行为通过对骗子和破坏者的惩罚来激励。在接下来的内容中，我们将介绍质押，这对于选择过程和惩罚措施都是必要的。

3.4.3 质押

共享存储的邻域

为了提供对偶然节点流失的强有力保护，即确保即使部分节点离线，邻域中的块仍然可被检索，Swarm 要求每个邻域内有若干独立的存储节点物理复制内容。如果按照每个提交权利证明的节点发放奖励，运营者可能会通过创建虚假节点专门为申请奖励而操控系统。虽然可以采取措施强制要求这些虚假节点必须在网络中运行，但运营者仍可能选择运行多个节点并将其存储共享在单一硬件上。因此，激励系统必须确保存储提供者不会采取这种策略。为此，我们引入了质押机制。

质押作为合约用来确定真实储备承诺（真相选择）并识别诚实节点中获胜者（获胜者选择）的权重。获胜的概率由节点之间的相对质押决定，因此质押是一个累加因子。因此，运营者的利润仅取决于他们在邻域内的总质押。考虑到运行节点的成本，运营者将没有动机将其质押分配给多个共享同一存储硬件的节点。

承诺质押和质押余额

在质押合约中注册时，质押者承诺一个以租金单位为单位的质押金额，称为承诺质押。承诺质押金额必须有下限。提交交易时发送的金额将被记录，并作为抵押品，称为质押余额。质押可以随时创建或补充，但更新的时间和金额会被记录。参与者仅限于那些最近没有更改质押的节点，确保在得知选定的邻域后，质押不会被更改。查询节点质押时，合约返回以 BZZ 为单位的绝对承诺质押金额。该值是通过以下两种方式计算的其中之一：(1) 承诺质押的租金单位数乘以租金单价，或者 (2) 整个质押余额，取其较小者。

质押必须能够在覆盖地址之间转移，以便在邻域内质押分布不平衡时进行邻域跳转。

剩余质押余额的可提取性

承诺质押使得运营者能够表明他们的利润率以及实现该利润的时间偏好。然而，由于利润只有在邻域内的相对质押已知时才会透明，节点可能需要一段时间才能发现其最佳质押。

如果 BZZ 代币的价格上涨，而租金单价下降，则节点在质押登记中的条目将显示出盈余余额。该盈余可以随时提取，使质押者能够实现其 BZZ 增值带来的利润。

3.4.4 邻域对储备内容的一致性

申请参与本轮的节点必须就哪些块属于各自的储备达成共识。最低要求是申请者必须就其责任区域达成一致，该区域可以通过其存储深度和覆盖地址推导出来。对储备内容的共识通过储备样本的身份进行验证。样本由储备中前 k 个块组成，采用基于块修改哈希的排序方式。修改后的块哈希是通过块内容和该轮特定的盐值计算得到的。除非节点在盐值被揭示时或者之后存储了所有（或大量）有效块及其数据，否则任何节点都无法构造出该样本集。

时效性与采样

储备样本必须排除非常近期的块，以防恶意上传者通过向邻域中的节点投放非同一组块来扰乱储备共识，进而破坏储备内容的共识。一种防止此类攻击的方式是将每个块及其存储时间一起保存到本地数据库中。通过使用拉取同步协议（pull-sync）进行的邻节点之间的块同步遵循按存储时间排序的顺序。我们要求实时同步，即连接建立后收到的块同步，延迟不超过一个预定义的常量时长，称为最大同步延迟（或简称最大同步滞后）。超过此同步延迟的同行连接，不符合协议中的合法存储节点要求。该限制确保恶意节点不能在最大同步滞后时间之后伪造新块的时间戳，从而失去其存储节点身份。

为了达成共识，必须确保邻域中的任何节点在不晚于 l 的时间内接收到的所有块，在索赔阶段之前已分发到邻域内的每个节点。如果我们将 l 的值设置为允许的同步滞后时间的 2 倍，那么每个最先到达节点的块，都有足够时间到达每个节点，并安全地包含在共识样本中。

存储深度与诚实邻域的大小

为了决定哪个揭示代表当前轮次的真相，从所有揭示中随机选择一个提交，选择的概率与揭示者的质押量成正比。更准确地说：基于每个邻域大小的质押量，即质押密度。因此，储备样本哈希值和报告的存储深度被视为当前轮次的真相。

现在我们可以理解为什么节点会正确报告实际存储深度。如果一个节点选择与其他邻居相比拥有更大的邻域，它将比其他节点更频繁地被选择。然而，由于与同行相比，承诺的存储深度减少，该节点的质押相较于报告较深存储半径的同行将以指数衰减的值进行计算，使得这种攻击变得代价高昂。

过度报告存储深度是可能的，只要节点落入邻域选择锚点的这个较窄范围。因此，系统性的利用该漏洞要求恶意行为者控制每个真实诚实邻域的子邻域中的一个质押节点。此外，获胜者需要提供证据，证明它们存储深度内的块集确实填满了它们的储备。样本中转换后块地址的实际整数值包含有关原始样本集大小的信息。通过要求样本集的大小落在预期范围内（具有足够的确定性），对样本值的上限施加了约束。这个结构被称为密度证明。

请注意，基于样本的密度证明可能会被伪造，如果攻击者通过某种方式挖掘内容过滤块，使得转换后的块地址形成足够密集的样本，然后使用自己的邮资批次为其盖章。为了增强对这种攻击的防护，可以实施额外的措施。一种方法是要求对整个邮资戳集进行承诺，并通过随机样本证明拥有足够的数量。通过施加这一要求，欺诈性索赔者不仅需要生成内容，还必须有足够的存储槽来伪造样本。这将要求攻击者购买整个网络规模的邮资批次，或者跟踪并存储网络中实际存在的邮资戳。前者对攻击者施加了禁止性的成本，而在后者的情况下，恶意索赔者必须承担依赖诚实邻居在事后检索证明其权益的见证块数据的风险。

跳过的轮次与滚动

如果在某一轮中没有提出索赔，奖励池中的资金将直接转移并增加下一轮重新分配的支付金额。这个政策是迄今为止最容易实现的，结果是最低的 gas 开销。

资格验证的八条规则

在这里，我们总结了验证索赔的八条规则，这些规则涉及承诺和揭示储备承诺，以及提交作为资格证明的证据；另见表 3.1：

复制

由于撒谎者会被冻结，即因揭示的储备承诺哈希或存储深度与赢家不同而被排除在游戏之外一段时间，邻域内的节点被激励通过使用拉取同步协议同步它们存储的块来复制它们的储备。

冗余

质押作为权重，用于确定邻域内一个节点被选为赢家的概率。因此，提交多个索赔没有任何优势。在一个邻域中运行多个节点（共享存储）的运营商，与运行单个节点并拥有相同总质押量的运营商相比，并没有获得任何优势。假设这种抑制过度扩展的机制有效，质押可以被视为保证真实冗余的手段。

责任

在揭示时，验证邻域选择锚点是否落在节点的责任半径内，即它是否属于节点所覆盖的地址范围，其中这些地址到节点覆盖地址的距离不小于它们报告的存储深度。

相关性

使用以储备承诺哈希为根的见证证明，我们提供证据，表明储备样本中任意选择的片段对应于见证块的地址。提交有效的邮资戳，表明在储备中存储该块对某人有意义（并且已经支付）。

保持

提供段包含证明，作为证据表明块数据在完整性方面得到了保存。

时效性

用于转换后的储备样本的盐值来源于当前轮次的随机数，证明该储备样本必须是最近编制的。见证和段索引来自下一个游戏的随机种子，确保在编制和承诺时，不会有足够的块数据被压缩或部分保留。

可检索性

通过基于接近度的路由来展示块的可检索性，表明它的地址属于邻域覆盖的地址范围。块的接近度顺序到节点的覆盖地址不小于其报告的存储深度。

资源

资源保持通过估计块和邮资戳的密度来验证构成储备的资源量，从而估算样本集的大小。

proof of	construct used	attacks mitigated
REPLICATION	Shelling-game over reserve sample	non-syncing, laggy syncing
REDUNDANCY	share of reward proportional to stake	shared storage, over-application
RESPONSIBILITY	proximity to anchor	depth/neighbourhood misreporting
RELEVANCE	scarcity of postage stamps	generated data
RETENTION	segment inclusion proof	non-storage, partial storage
RECENCY	round-specific salt for reserve sample	create proof once and forget data
RETRIEVABILITY	proximity of chunk	depth over-reporting
RESOURCES	density-based reserve size estimation	targeted chunk generation (mining)

表 3.1：r8：用作奖励资格证明的证据。

3.4.5 定价与网络动态

在本节中，我们首先将 redistribution（再分配）方案放在自我可持续性的框架中，并为价格发现提供一个简单的解决方案。

为了使 Swarm 成为一个真正自给自足的系统，存储租赁的单价需要根据供需关系进行调整。理想情况下，价格应根据可靠信号自动调整，从而实现动态的自我调节。这里的核心理念是，存储节点在再分配游戏中提供的信息也充当了价格信号。换句话说，再分配游戏充当了去中心化的价格预言机。

邻域的拆分与合并

存储深度表示邻域内存储节点保持所有带有效邮资戳的块并填充其储备的接近半径。

如果最近购买的批次所新增的存储插槽数量 (进入率) 与过期存储插槽的数量 (退出率) 相平衡，存储深度将保持不变。然而，假设当储备的块数量增加时会发生什么。随着客户端的储备容量固定，随着时间的推移，节点能够用与之前最远的块相比最多只有一个接近顺序的块来填充它们的容量。这样会导致它们的存储深度增加。具体而言，当储备的块数量翻倍时，存储深度将增加一个。

为了在相同的冗余约束下存储这些多余的数据，网络需要双倍的节点数量。如果其他条

件相等，网络范围内的储备翻倍，邮资收入翻倍，因此整体的再分配奖励池也会翻倍。当邻域在吸收新增数据量时进行拆分，它们同时会释放出旧深度 PO 磁桶中的块，即现在由它们的姐妹节点存储的块。

利用率

利用率是以有机的方式对节点的储备进行压力测试的手段，旨在避免节点过度填满储备并且确保及时发现容量压力。这为触发的激励措施提供了足够的安全缓冲。例如，如果利用率为 1/8，存储深度最多可以比储备深度浅 3 个 PO。现在，进入率可以非常高，并且将储备深度降低到存储深度。当检测到潜在（预留）和实际（观察到的） DISC 利用率之间的差距缩小时，激励措施的任何变化都将有足够的缓冲来生效，而不会威胁到目标冗余性。

诚实节点数量作为价格信号

由于存储容量已达上限，供需比直接反映在参与 Schelling 游戏的诚实节点数量上。

我们假设，长期留在网络中的节点表示它们具有盈利能力。对于一个稳定的 swarm，邻域只需要至少 4 个（平衡的）节点。假设节点的质押相等（或者更准确地说，假设相对质押使得节点运营商的盈利能力相等），如果一个邻域中有 n 个节点，它们的长期利润是平均分配的，这个数量在节点数量为 4 个（n = 4）时是最优的。然而，邻域中可以有更多的节点，因为投机性的运营商可能会预见到由于容量需求，邻域会发生拆分，并因此启动自己的节点。随着这些节点的加入，同一邻域的长期收益将分配给比最优情况更多的节点。然而，节点容忍这一点意味着奖励过多（价格过高），网络能够容忍价格下降。

另一方面，如果诚实的揭示者数量低于邻域的冗余需求，这则表明容量短缺，因此需要增加存储租赁价格。

价格预言机的参数化

从一轮到下一轮更新价格的规则是，将当前价格乘以一个值 m，该值取决于当前轮次诚实揭示者的数量。从数学角度来看，pt+1 = m * pt，其中 pt 是 t 轮的价格（pt+1 则是下一轮的价格）。我们根据诚实揭示者的数量 r 和一个稳定性参数 σ 来定义乘数 m，σ 决定了价格增加或减少的速度，其他条件不变。

具体而言，我们选择 m = 2^σ(4−r)，因此有 pt+1 = 2^σ(4−r) * pt。这个公式表达了揭示者数量 4 − r 与最优值 4 之间的偏差如何映射到价格的指数变化。稳定性参数 σ 决定了价格变化在多轮之间的平滑程度，表示在持续出现最低供给不足信号的情况下，租赁价格需要多少轮才能翻倍（或者在持续供给过剩的情况下，价格需要多少轮才能减半）。图 3.19 展示了该价格模型的工作原理。

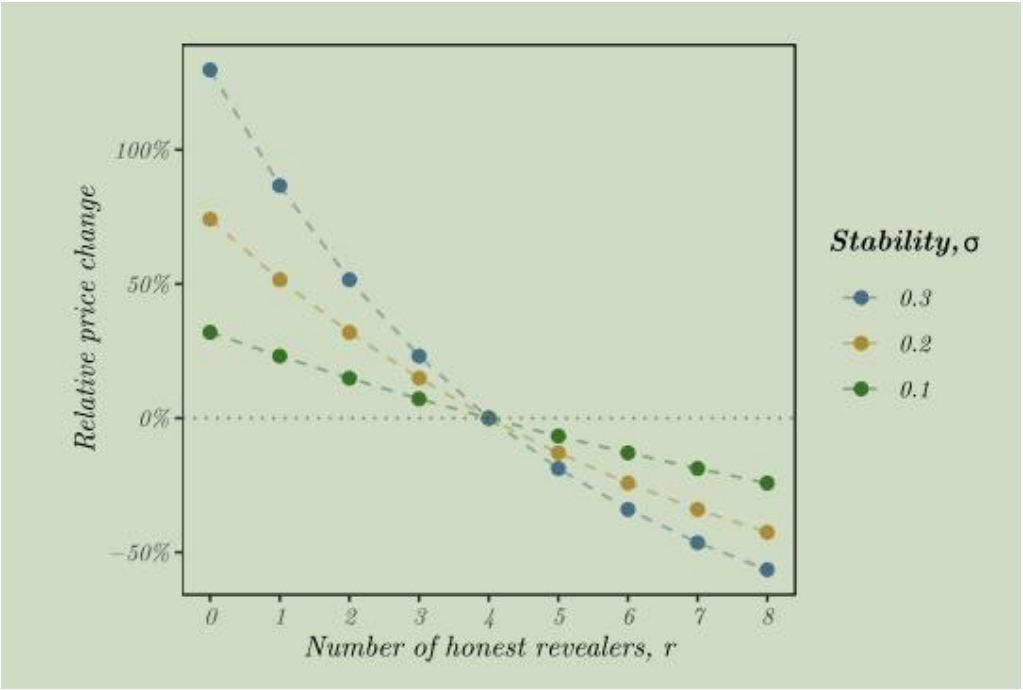

图 3.19：自适应定价。价格相对变化（y 轴），数学上表示为下一轮价格除以当前轮价格减一（pt+1/pt −1），与当前轮中诚实揭示者数量 r（x 轴）的关系被展示出来。针对稳定性参数 σ 的三种不同值（颜色）进行了展示。点表示实际的价格变化值；连接的虚线仅用于视觉辅助。水平虚线标记了价格无变化的点。当诚实揭示者数量为四时，任何 σ 值下的价格变化恰好为零。否则，随着诚实揭示者数量的变化，较大的 σ 值会导致较大的相对价格变化。

该简单模型应用了两个小调整。首先，r 的最大值被限制为 rmax（在我们的例子中选择为 8）。也就是说，r 实际上应该解释为诚实揭示者数量与 rmax 之间的较小值。其次，价格永远不能降到预定的最低值 pmin 以下。也就是说，如果价格在一轮到下一轮之间下降并达到低于预定最低值 pmin 的水平，价格将保持在 pmin 不变。

3.5 保险：负面激励

迄今为止，所提出的存储激励措施是指通过向存储节点提供货币奖励来鼓励内容的保存。这是通过一个链上游戏实现的，游戏通过公平地再分配邮费支付给存储节点，从而在集体层面提供了正向激励。然而，这样的系统容易受到“公地悲剧”问题的影响，即消失的内容（任何特定块丢失）对存储节点没有负面后果（任何一个存储节点都不会被惩罚）。缺乏个体责任感使得存储激励作为防止数据丢失的安全措施受到限制。另一方面，引入竞争性保险增加了一层负面激励，迫使存储节点在承诺时非常精准，以确保对用户的可靠性。激励系统的设计需要特别注意，确保未能存储每一位承诺的内容不仅没有利润，而且对保险人来说是灾难性的。

3.5.1 惩罚措施

与带宽激励中检索立即结算不同，长期存储保证具有承诺性质，只有在有效期结束时才能决定承诺是否得到履行。仅仅失去声誉在这些情况下并不足以作为对不正当行为的威慑：由于新节点必须立即提供服务，作弊者可以通过新身份继续销售（空的）存储承诺。

我们需要惩罚措施的威胁来确保存储承诺的履行。这可以通过押金系统来实现。希望出售存储承诺的节点应该在做出承诺时进行质押验证并锁定资金。这意味着节点需要提前注册，签署合同并支付保证金。一旦注册，节点可以出售覆盖其资金锁定期间的存储承诺。如果在其注册有效期内，发现它们丢失了其承诺覆盖的块，则会失去押金。

我们可以从一些合理的指导原则出发：— 所有者在提交存储时需要表达他们的风险偏好。 — 存储者在承诺存储时需要表达他们的风险偏好。 — 需要有合理的市场机制来匹配需求和供应。 — 需要保证所有者的内容得到安全存储。 — 需要有一个诉讼系统，存储者未能履行承诺时可以被追责。

所有者的风险偏好包括所覆盖的时间段以及对可靠性的要求。这些偏好应该在每个块的基础上指定，并且应该在协议层面上完全灵活。

满足存储者的风险偏好意味着他们有方式表达他们对存储内容的保存可靠性的确定性，并将其纳入定价中。有些节点可能不希望提供长期存储保证，而其他节点可能无法承诺大量押金。这使得节点在竞争服务提供时有所差异。

市场机制意味着存在灵活的价格谈判或发现，或者自动反馈回路，这些机制通常会对供需变化做出响应。

接下来，我们将详细阐述一种激励机制类别，我们称之为“交换、发誓与欺诈”，因为其基本组成部分如下： 交换
节点与其注册的同行保持准永久的长期联系。在这些连接中，同行之间交换块和收据，从而触发交换记账（见 3.2.1）。

发誓

注册在 Swarm 网络上的节点需要为其行为负责，如果被发现违反 Swarm 规则，将面临链上诉讼过程中的押金丧失。

欺诈

节点监控其他节点，检查其是否遵守承诺，并通过提交挑战来进行诉讼过程。

3.5.2 通过收据的合同

诉讼程序要求各方之间存在合同协议，最终将支付费用以确保未来内容可用性的所有者与负责保存并确保内容在未来任何时刻都能立即访问的存储节点联系起来。激励结构需要确保诉讼仅作为最后的手段使用。

管理存储交易的最直接方法是通过所有者与存储节点之间的直接合同。这可以通过让存储节点返回它们接受存储的块的签名收据来实现。所有者然后可以直接或通过托管方式支付这些收据。在后者的情况下，只有当存储节点能够提供存储证明时，才会将锁定的资金奖励给存储节点。这个过程类似于邮票彩票流程。保险可以通过特别标记的邮票形式购买。保管收据声明可以闭合循环，表示上传者与存储者之间的合同。基于保管证明的支付可以像彩票一样实现。

即使试图访问块的消费者并不是存储和提供请求内容协议的一方，未能交付存储内容仍

然会面临处罚。因此，诉讼预计将向希望检索内容的第三方开放。

如果块与收据的配对是公开且可访问的，那么内容的消费者/下载者（不仅仅是创作者/上传者）可以在发现某个块丢失时提起诉讼。

注册

在节点能够出售长期存储承诺之前，它必须首先通过我们称为 SWEAR（安全确保档案或 Swarm 执行与注册的安全方式）合同在区块链上注册。

SWEAR 合同允许节点注册它们的公钥，成为 Swarm 中的可追责参与者。注册过程涉及将押金发送到 SWEAR 合同，这作为保证金，以防注册节点违反它们“发誓”遵守的条款（即节点未能履行存储承诺）。注册仅在设定的时间内有效，届时 Swarm 节点有资格取回其押金。Swarm 的用户应该能够依赖押金丧失作为防止恶意行为的激励，前提是节点保持注册状态。因此，在注册期到期之前，押金不得退还。保险期的到期应该包括一个最终阶段，在此期间节点不允许发放新收据，但仍然可以被挑战。

当一个注册的保险节点收到存储最接近它的块的请求时，它可以通过签署收据来确认请求。正是这些签署的收据被用来执行因内容丢失而产生的处罚。由于这些收据背后有锁定的保证金，它们可以被视为对存储和提供特定块的有担保承诺。

3.5.3 提交挑战

如果一个节点未能遵守其发誓要遵守的 Swarm 规则，那么执行惩罚措施必须先经过诉讼程序。该过程的实施被称为 SWINDLE（带有保险存款的诉讼与托管）。

当用户尝试检索受保险内容并未找到某个块时，他们可以通过提交挑战来报告丢失。这种情形是启动诉讼的典型情境。这类似于法院案件，其中收据的发行者是被告，必须证明其无罪。与法院程序类似，区块链上的公开诉讼应当是最后的手段，当尽管有威慑措施和积极激励，规则仍被滥用时才使用。

挑战以发送到 SWINDLE 合同的交易形式存在，其中挑战者提交丢失块的收据。任何节点只要持有有效的收据（即使该收据不一定是由他们自己获得的），都可以为某个块提交挑战。该交易还发送了覆盖上传该块费用的存款。挑战的有效性及其反驳需要由合同轻松验证。合同通过检查以下条件来验证收据的有效性：

真实的

收据是由注册节点的公钥签署的。

活跃的

收据的到期日期尚未过期。

有资金的

与收据一起发送了足够的资金，以补偿对方在挑战被驳回时上传该块的费用。

上述最后一点旨在防止无谓的诉讼，即防止通过向区块链提交虚假挑战来发起可能造成的 DoS 攻击。

该合同提供了一个访问器，用于检查某个节点是否被挑战（即可能需要承担罚款），因此被挑战的节点可以被通知，要求其及时提交该块。挑战会在固定的时间内保持开放，挑战的结束时间实际上设置了反驳挑战的截止日期。

在验证反驳格式后，合同通过对比块负载的哈希与争议哈希，或者验证保管证明来检查其有效性。

如果挑战在挑战开放期间内被反驳，则被告节点的存款保持不变。上传该块的费用必须从挑战者的存款中退还给上传者。为了防止 DoS 攻击，这笔存款应当远高于上传费用（例

如，相应燃气费用的小整数倍）。反驳成功后，挑战会从区块链状态中清除。

该挑战机制是最简单的方式：(1) 让被告反驳挑战，(2) 使争议数据对需要它的节点可用。

3.5.4 成功挑战与执行

如果在截止日期前未能成功反驳挑战，那么该费用将被视为已被证明，案件进入执行阶段。被证明丧失块的节点将失去其押金。由于押金被锁定在 SWEAR 合约中，因此执行将确保成功。

如果在诉讼中，发现一个（已由收据覆盖的）块丢失，那么押金必须至少部分被销毁。请注意，这是必要的，因为如果将罚款作为赔偿支付给丢失块的收据持有者，这将为已注册节点提供一种通过“丢失”虚假块来提前退出的途径，这些虚假块是由合谋用户存入的。由于 Swarm 的用户关注其信息是否得到可靠存储，他们保持收据的主要激励是确保 Swarm 保持动力来实现这一目标，而不是在未能存储的情况下获取赔偿。如果押金数额较大，我们可以通过支付启动诉讼的赔偿来解决这一问题，但我们必须销毁大部分（例如 95%）押金，以确保这一轻松退出的途径被关闭。

惩罚可以包括暂停，这意味着被判定有罪的节点将不再被视为已注册的 Swarm 节点。此类节点只有在创建新身份并再次缴纳押金后，才能恢复出售存储收据。

激励承诺服务

没有锁定资金的延期付款会使存储者面临无法获得付款的风险。另一方面，预付款（即在签订合同时支付，而非存储期结束后支付）则使买方容易受到欺诈。若不限制节点可出售的收据总价值，恶意节点可以收取超过其押金的金额然后消失。尽管违约并丧失押金，恶意节点仍然可以获利。鉴于网络规模和相对稳定的保险存储需求，押金可以设定得足够高，从而使此类攻击变得不再经济可行。

锁定全部金额消除了存储者因被保险方可能的无力偿付而产生的不信任。当支付保险时，这笔资金应覆盖存储一个块在整个存储期内的总费用。该金额将被锁定，并分期释放，前提是节点提供保管证明。另一方面，由于付款是延迟的，因此在工作完成之前无法收取资金，这彻底消除了“收款跑掉”的攻击。

3.6 总结

在本书架构部分的前两章中，我们介绍了 Swarm 的核心内容：第二章描述的点对点网络层实现了一个分布式的不可变块存储，这一系统通过随后的章节中描述的激励机制得到补充。最终形成的基础层系统提供了以下功能：

— 无许可参与和访问，
— 对节点运营者零现金进入，
— 最大化资源利用，
— 数据的负载均衡分发，
— 可扩展性，
— 存储和检索过程中的抗审查和隐私保护，
— 自动扩展流行内容，
— 基本的合理否认和机密性，

— 在节点掉线的动态网络中具有抗波动性和最终一致性，
— 通过内建的经济激励机制实现可持续性，无需干预，
— 强大的私有点对点记账，
— 激励带宽共享，
— 链下微承诺与链上结算，
— 抗 DoS 攻击和垃圾邮件防护，
— 存储的正向激励（即通过奖励激励），
— 对数据丢失的负向激励（即通过惩罚措施的威胁进行抑制）。

第 4 章 基于 DISC 的构建

本章在分布式块存储的基础上，引入了数据结构和流程，这些结构和流程使得更高层次的功能成为可能，提供了处理数据的丰富体验。特别地，我们展示了如何组织块以表示文件（4.1.1），以及如何组织文件以表示集合（4.1.2）。我们还介绍了键值映射（4.1.4），并简要讨论了任意功能性数据结构的潜力。接着，我们将重点介绍提供机密性和访问控制的解决方案（4.2）。

在 4.3 中，我们介绍了 Swarm 订阅，旨在表示各种顺序数据，如可变资源的版本更新或实时数据交换的消息索引：提供了一种持久化拉取消息系统。为了实现各种类型的推送通知，4.4 引入了新概念——特洛伊块，这些块允许消息伪装成块并定向到其在 Swarm 中的预定接收者。我们解释了如何将特洛伊块和订阅结合起来，形成一个具有强大隐私功能的完备通信系统。

4.1 数据结构

在前两章中，我们假设数据以块的形式组织，即固定大小的数据块。然而，在本节中，我们将介绍支持表示任意长度数据的算法和结构。我们将介绍 Swarm 清单，它们构成了表示集合、索引和路由表的基础，使得 Swarm 能够托管网站并提供基于 URL 的寻址功能。

4.1.1 文件和 Swarm 哈希

在本节中，我们介绍 Swarm 哈希的概念，它提供了一种机制，通过组合块来表示更大规模的结构化数据，例如文件。Swarm 哈希算法的核心思想是将块安排成 Merkle 树。在这种结构中，叶节点对应于来自输入数据连续段的块，而中间节点对应于由其子节点的块引用组成的块。这些组合后的块被打包成另一个块，如图 4.1 所示。

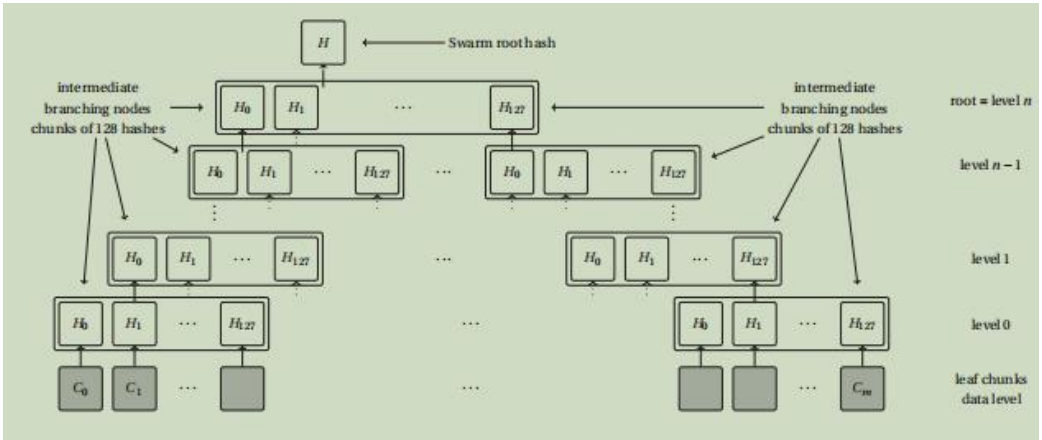

图 4.1：Swarm 哈希：数据输入被分割为 4 千字节的分块（灰色），并进行 BMT 哈希处理。它们的哈希值被打包成从第 0 层开始的中间分块，直到第 n 层只剩下一个分块。

分支因子和引用大小

树的分支因子是通过将块大小除以引用大小计算得出的。对于未加密的内容，块引用就是块的 BMT 哈希，它是 32 字节，因此分支因子为 4096/32 = 128。在中间节点下引用的块组称为一个批次。如果内容被加密，块引用由块哈希和解密密钥的连接组成。两者都是 32 字节长，因此加密块引用为 64 字节，分支因子为 64。

因此，一个单独的块可以代表 Swarm 哈希树中的一个中间节点，这时其内容可以被分割成引用，从而可以检索它们的子节点。这些子节点本身可能是中间块，如图 4.2 所示。通过递归地从根块向下解包，我们最终可以获得一系列数据块。

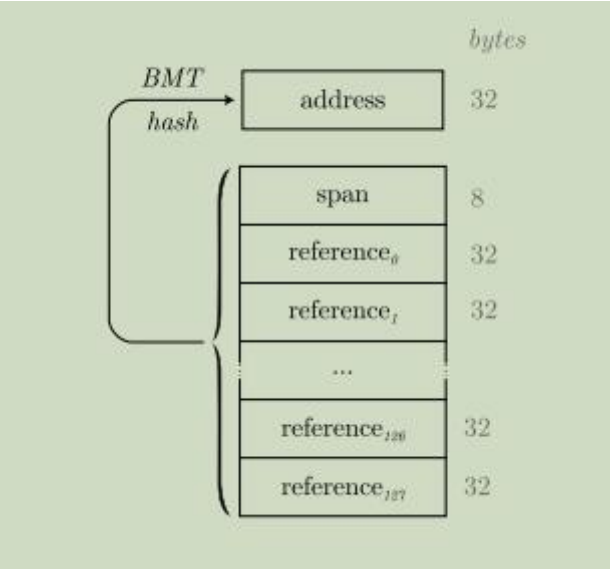

图 4.2：中间分块。它封装了对其子分块的引用。

块跨度和深度完整性

在一个中间块下涵盖的数据的长度称为块跨度。为了能够判断一个块是否为数据块，块跨度会以 64 位小端二进制表示法被附加到块数据前面。在计算块的 BMT 哈希时，这个跨度构成了需要添加到 BMT 根并一同哈希的元数据，最终得到块地址。当从哈希开始组装一个文件时，可以通过查看跨度来判断一个块是数据块还是中间块。如果跨度大于 4K，则该块为中间块，其内容需要解释为一系列子块的哈希值；否则，它是数据块。

理论上，如果文件的长度已经知道，那么中间块的跨度就不再需要，因为我们可以计算出树所需的中间级别数。然而，使用跨度则不允许将中间级别恢复为数据层。通过这种方式，我们强制要求深度的完整性。

追加和恢复中断的上传

Swarm 哈希具有一个有趣的特性，即任何与中间块对应的数据跨度也可以视为文件，因此可以像中间块是其根哈希一样引用它。这一特性非常重要，因为它使得在不重复块的情况下（除了 Merkle 树的右边缘未完成部分），可以向文件中追加数据并保留对早期状态的历史引用。向文件中追加数据特别适用于在上传大文件过程中发生崩溃后恢复上传的场景。

随机访问

请注意，文件中的所有块，除了右边缘，都是完全填充的。由于块具有固定大小，因此可以提前计算出到特定块的路径以及在该块内搜索任何任意数据偏移量的偏移量。因此，文件的随机访问是立即支持的（见图 4.3）。

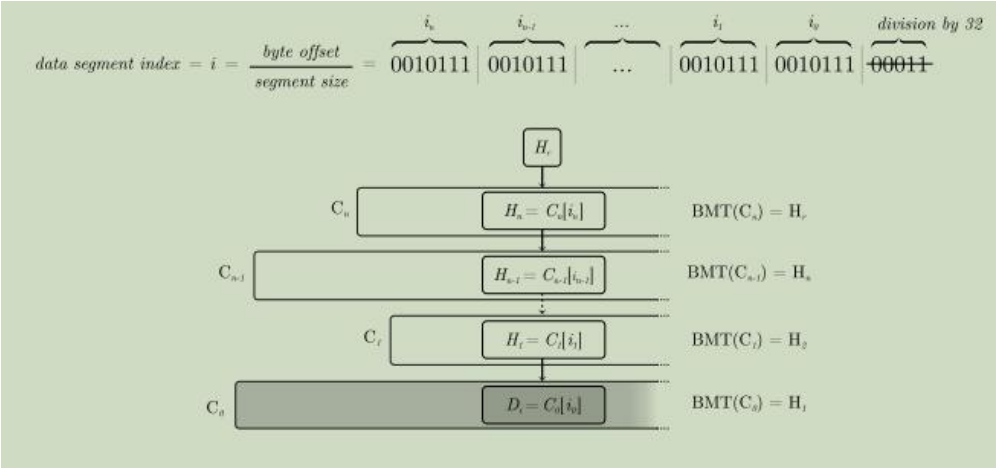

图 4.3: 使用 Swarm 哈希在任意偏移处进行随机访问。任意偏移告诉我们如何遍历 Swarm 哈希树。

文件的紧凑包含证明

假设我们需要证明一个子字符串在文件中特定偏移位置的包含情况。我们已经观察到，当将偏移量应用于数据时，它会沿着 Swarm 哈希遍历一条确定性的路径。由于子字符串包含证明简单地归结为一系列数据段路径的证明，块地址是 BMT 哈希的结果，其中基础段为 32 字节长。这意味着在中间块中，BMT 基础段与它们子节点的地址对齐。因此，证明一个中间块在特定跨度偏移处的子块是等同于给出子块哈希的段包含证明。因此，可以通过一系列 BMT 包含证明来证明文件中子字符串的包含性，其中序列的长度对应于 Swarm 哈希树的深度（见图 4.4）。

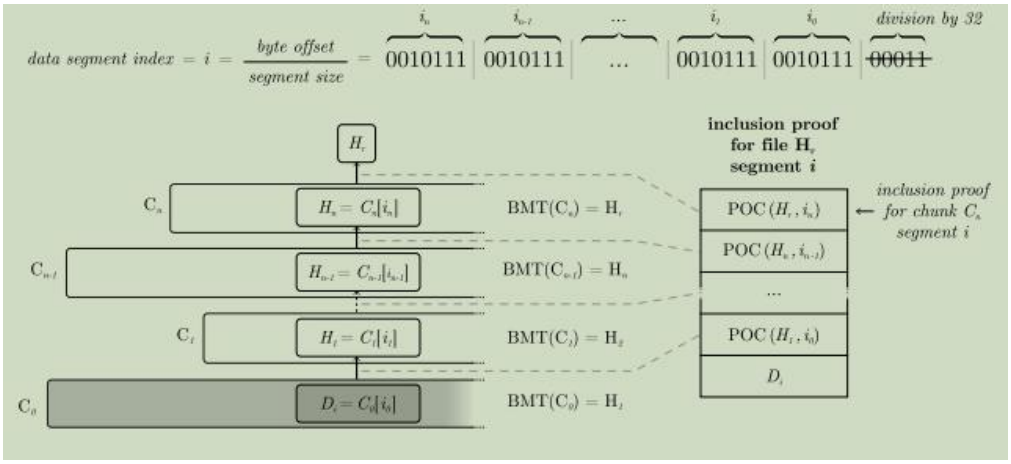

图 4.4：文件的紧凑包含证明。如果我们需要证明段 i 的包含性，在除以 32（段内位置）之后，我们遵循 7 位比特的组来找到中间节点的相应段。

请注意，即使在加密数据的情况下，这样的包含证明也是可能的。因为可以选择性地披露某个段位置的解密密钥，而不暴露任何可能泄露其他部分加密信息的内容。

在本节中，我们介绍了 Swarm 哈希，这是一种基于块的数据结构，用于表示文件，支持以下功能：

随机访问

文件可以从任何任意偏移量读取，而不会产生额外的成本。

追加

支持无重复的追加操作。

保持长度的编辑

支持长度保持的编辑，且不会重复未修改的部分。

紧凑的包含证明

允许以空间对数级别的 32 字节分辨率进行包含证明，且证明文件大小。

4.1.2 集合与清单

Swarm 清单作为框架，定义了任意路径与文件之间的映射，以表示集合。它包含与集合及其对象（文件）相关的元数据。清单条目包含对文件的引用，或更准确地说，包含对文件表示的 Swarm 根块的引用（见 4.1.1），并指定文件的媒体 MIME 类型，以确保浏览器能够适当地处理它。清单可以被视为（1）路由表、（2）目录树或（3）索引，这使得 Swarm 能够实现（1）网站、（2）文件系统目录或（3）键值存储（见 4.1.4）。清单在 Swarm 中用于基于 URL 的寻址（见 4.1.3）。

清单以压缩的 Trie 结构（也称为前缀树或字典树）表示，其中每个 Trie 节点都序列化为块。路径与一个清单条目相关联，该条目至少指定引用。若路径是多个路径的公共前缀，则引用可能指向嵌入的清单，从而在 Trie 中实现分支，如图 4.5 所示。

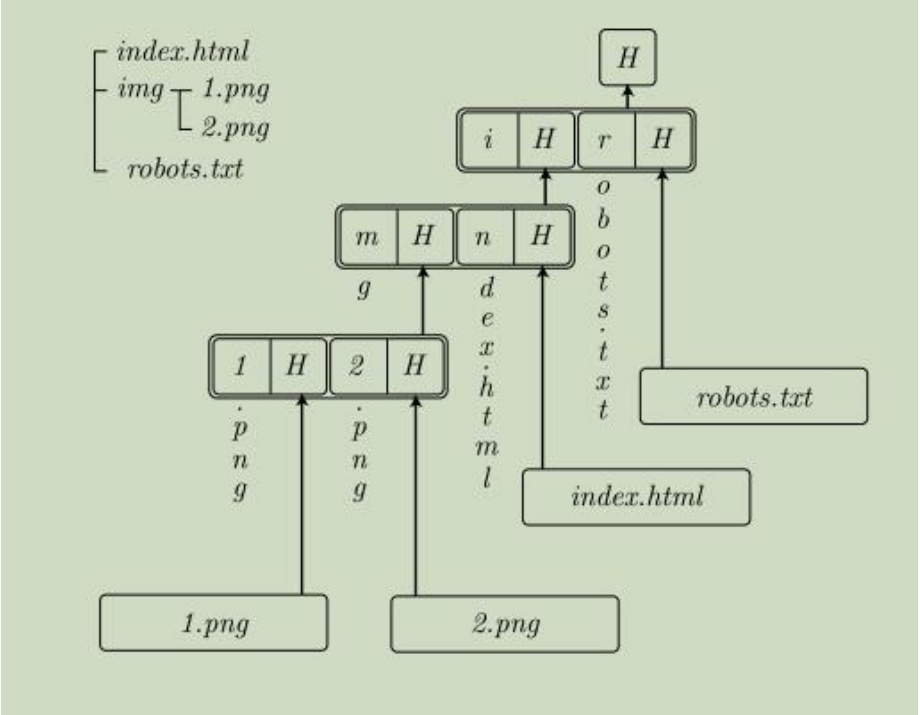

图 4.5：清单结构。节点表示一个通用的 Trie 节点：它包含描述共享前缀的延续的分支。分支通过键的下一个字节进行索引，其值包含对子节点的 Swarm 引用以及最长前缀（压缩）。

清单条目本质上是一个引用，提供关于文件或目录的信息。其元数据涉及以下几个方面：

1 下载组件的参数，负责将块组装为字节流。这包括访问控制信息、纠删码参数以及用于块恢复的发布者。

2 与客户端渲染相关的信息，由浏览器处理。这可能包括内容类型头，或通用的 HTTP 头，由本地 Swarm 客户端的 API 获取，并在检索文件时设置在响应头中。

3 下载时映射到文件系统的文件信息，例如文件权限。

清单的高级 API 提供了上传和下载文件及目录的功能。它还提供了一个接口，用于将文

档添加到集合中的某个路径或从集合中删除文档。请注意，这里的删除仅意味着创建一个新清单，其中省略了相关路径。在 Swarm 中没有其他形式的删除，即被删除清单条目中的引用值仍然存在于 Swarm 中。Swarm 通过 bzz URL 方案公开清单 API。

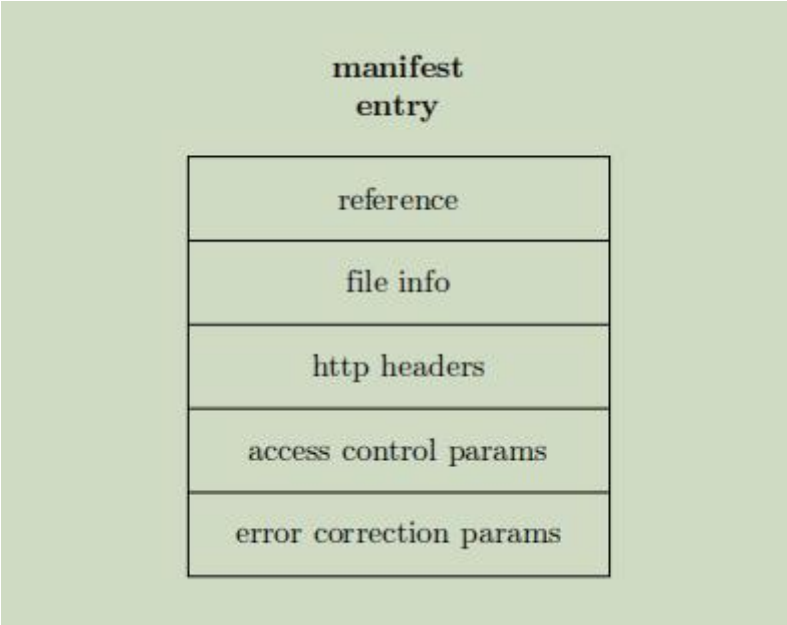

图 4.6: 清单条目是一种数据结构，包含对文件的引用，包括与汇编器、访问控制和 HTTP 头相关的文件或目录的元数据。

4.1.3 基于 URL 的寻址和名称解析

之前，我们介绍了 Swarm 的低级网络组件，作为分布式不可变的块存储 (DISC，见 2.2.1)。在前两节中，我们讨论了如何在 Swarm 中表示文件（4.1.1）和集合（4.1.2），并通过块引用来标识它们。清单提供了一种方法，用于在集合中索引单个文档，使其能够作为在 Swarm 上托管的网站的表示。根清单作为虚拟托管网站的入口点，因此类比于托管服务器。在当前的网络中，域名解析为托管服务器的 IP 地址，URL 路径（对于静态站点）则根据其相对于主机设置的文档根路径映射到目录树中的条目。类似地，在 Swarm 中，域名解析为根清单的引用，URL 路径则根据其路径映射到清单条目。

当 HTTP API 提供 URL 服务时，会执行以下步骤:

域名解析

Swarm 将主机部分解析为根清单的引用。

清单遍历

沿着与 URL 路径匹配的路径递归遍历嵌入的清单，直到到达清单条目。

提供文件

获取清单条目中引用的文件，并在浏览器中渲染，文件的元数据（特别是内容类型）通过清单条目中的元数据获取。

Swarm 支持使用以太坊名称服务 (ENS) 进行域名解析。ENS 系统类似于旧 Web 的 DNS，能够将人类可读的名称转换为系统特定的标识符，即在 Swarm 中的引用。为了使用 ENS，Swarm 节点需要连接到支持以太坊 API 的 EVM 基础区块链（如 ETH 主网、Ropsten、ETC 等）。ENS 的用户可以在区块链上注册域名，并将其解析为一个引用，最常见的是公开（未加密）清单根的内容哈希。如果这个清单表示一个包含网站资产的目录，可以将该哈希的默

认路径设置为期望的根 HTML 页面。当使用 Swarm 启用的浏览器或网关访问 ENS 名称时，Swarm 将直接渲染根 HTML 页面，并提供相对路径中提供的其余资产。Swarm 使得网站托管变得简单，同时也为旧版浏览器提供了接口，并在传统 DNS 系统上提供了去中心化的改进。

4.1.4 映射和键值存储

本节描述了在 Swarm 中实现简单分布式键值存储的两种方法。两种方法都仅依赖于已经介绍过的工具和 API。

第一种方法涉及使用清单：路径表示键，清单条目中具有特定路径的引用指向值。这种方法的优势在于通过 bzz 清单 API 提供了完整的 API，支持插入、更新和删除操作。由于清单是以压缩的 Trie 结构构建的，因此这种键值存储具有可扩展性。索引元数据的存储需求与键值对的数量成对数关系。查找操作需要对数级带宽。该数据结构还支持按键顺序进行迭代。

单一所有者块（single owner chunk）也提供了一种定义键值存储的方法。

第二种方法则简单地假设单一所有者块的索引是通过数据库名称的哈希与键的哈希连接而成的。这种结构只支持插入操作，不支持更新或删除。插入和查找操作都是常数空间和带宽的。然而，查找操作不能防止假阴性，即如果未找到表示键值对的块，并不意味着该键值对从未被创建（例如，它可能已被垃圾回收）。因此，基于单一所有者块的键值存储最好用于以下场景：(1) 有界的可重新计算值的缓存，(2) 表示之间的映射，例如 Swarm 哈希和以太坊区块链状态树节点中使用的 Keccak256 哈希之间的转换，或(3) 常规的关系链接，例如社交媒体帖子上的点赞、投票和评论。

4.2 访问控制

本节首先讨论如何通过加密保障内容的机密性。当用户被授予管理他人访问受限内容的权限时，加密变得尤为重要。这包括管理私人共享内容、授予成员访问特定区域的权限等场景。通过这种方式，我们提供了一个强大而简单的 API 来管理访问控制，这一功能通常由集中式的权限控制来处理，而集中式权限控制常常面临频繁且灾难性的安全漏洞问题。

4.2.1 加密

本节着重讨论如何在分布式公共数据存储中实现机密性。我们探讨了如何满足许多用例中存储私人信息的自然需求，同时确保信息仅能被特定授权方访问，且能够使用 Swarm 进行存取。

显然，目前 web 应用中普遍使用的基于服务器的访问控制所提供的伪机密性是不足够的。在 Swarm 中，节点之间预期会相互分享数据块，实际上，数据块的存储者会受到激励，将其提供给任何请求它们的用户。这种去中心化的架构使得节点无法充当受信任的守门人来控制数据访问。此外，由于网络中的任何节点都可能成为存储节点，因此机密性解决方案必须确保不会泄露任何可能区分私人数据块与随机数据的信息。因此，防止未经授权方访问私人数据块的唯一方法就是通过加密。在 Swarm 中，如果请求方被授权访问某个数据块，他们必须拥有解密密钥，以便解密该数据块。另一方面，未经授权的方则不能访问解密密钥。顺

便提一下，这一机制也为合理否认提供了基础。

数据块级别的加密在 2.2.4 节中已有描述。其具有几乎与数据块存储层独立的优点，使用与存储和检索未加密内容相同的底层基础设施。访问私人数据和公共数据的唯一区别在于数据块引用中是否包含解密/加密密钥，以及相关的少量加密计算开销。

存储 API 的原始 GET 端点同时支持加密和未加密的数据块引用。如果数据块引用的大小为双倍，则触发解密；它由加密数据块的地址和解密密钥组成。使用该地址，可以检索、存储并使用提供的解密密钥解密加密的数据块。API 将返回解密后的明文。

存储 API 的 POST 端点要求用户指定是否希望对上传内容进行加密。在两种情况下，数据块都会存储并推送同步到网络中，但如果需要加密，则必须首先创建加密的数据块。如果没有提供更多上下文，系统会生成一个随机加密密钥，用它作为种子生成随机填充物，以便将数据块填充到完整的 4096 字节，如果需要的话，最后用该密钥对明文进行加密。在加密的情况下，API 的 POST 调用将返回 Swarm 引用，包括 Swarm 哈希作为数据块地址和加密密钥。

为了保证加密密钥的唯一性，并减轻操作系统熵池的负担，建议（但不是强制要求）使用内存中存储的（半）永久随机密钥生成密钥，并将其用作明文的 MAC。这个密钥可以是永久性的，并通过启动时提供的密码使用 scrypt（Percival 2009）生成。除了明文外，还可以使用清单条目的命名空间和路径作为上下文。以这种方式使用密钥派生函数的结果是，只要上下文相同，数据块的加密将是确定性的：如果我们交换文件中的一个字节，并用相同的上下文对其进行加密，则文件中除被修改的部分外，所有数据块将与原始数据块完全相同。因此，加密有助于去重。

4.2.2 管理访问

本节描述了客户端获取加密内容完整引用所需遵循的过程。该协议依赖于基本的元信息，这些元信息仅作为明文元数据进行编码，并明确地包含在文档的根清单条目中。此级别的访问权限称为根访问权限，不需要特殊权限。

与此相对，授权访问是一种选择性访问，需要同时具备根访问权限和访问凭证，即授权的私钥或密码短语。授权访问允许多个共享相同根访问权限的方对内容进行不同级别的访问。通过这种方式，可以在不更改访问凭证的情况下更新内容。授权访问是通过在引用上增加一层加密实现的。

引用的对称加密称为加密引用，该层使用的对称密钥称为访问密钥。

在授权访问的情况下，根访问元信息包含加密引用和通过访问凭证获取访问密钥所需的附加信息。一旦获取了访问密钥，就可以通过使用该访问密钥解密加密引用来获得内容的引用。最终的完整引用包括根数据块的地址和根数据块的解密密钥。然后，可以使用标准方法检索并解密所请求的数据。

访问密钥可以从多种来源获得，我们将定义其中三种。

首先，基于提供的凭证派生出会话密钥。在授予单一方访问权限的情况下，会话密钥直接用作访问密钥，如图 4.7 所示。然而，在多个方的情况下，使用额外的机制将会话密钥转换为访问密钥。

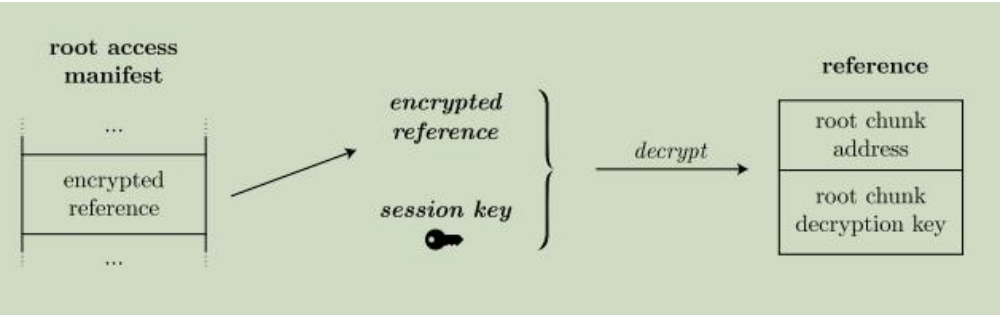

图 4.7：访问密钥作为单方访问的会话密钥。

密码短语

派生会话密钥的最简单凭证是密码短语。使用 scrypt 根据根访问元信息中指定的参数从密码短语中获取会话密钥。scrypt 的输出是一个 32 字节的密钥，可以直接用于 Swarm 的加密和解密算法。

在典型的使用场景中，密码短语通过安全措施充分的离带方式分发，或通过面对面交换。任何知道从密码短语派生出密钥的用户都能够访问该内容。

非对称派生

更复杂的凭证是私钥，类似于在整个以太坊中用于访问账户的私钥，即使用 secp256k1 的椭圆曲线。为了获取会话密钥，必须在内容发布者和被授权人之间执行椭圆曲线 Diffie-Hellman（ECDH）密钥协商。由此产生的共享秘密与盐值一起进行哈希处理。内容发布者的公钥和盐值被包含在根访问清单的元数据中。基于 ECDH 的标准假设，只有发布者和被授权人可以计算出该会话密钥，其他任何人都无法计算。一旦访问授权给单一公钥，通过这种方式派生出的会话密钥可以直接用作解密加密引用的访问密钥。图 4.8 总结了使用凭证派生会话密钥的过程。

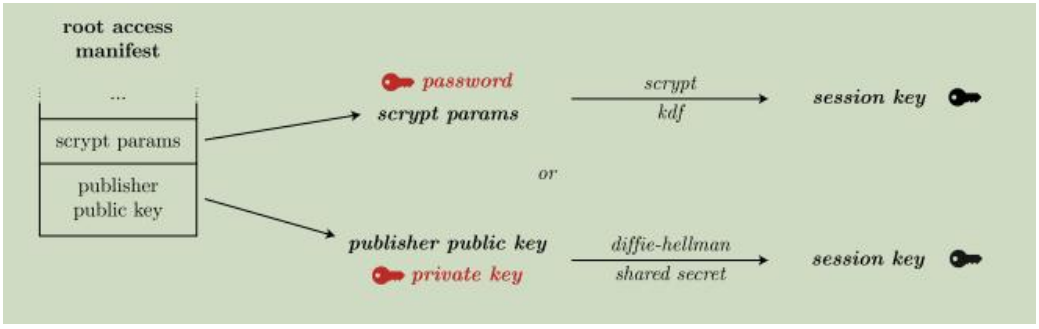

图 4.8：用于派生会话密钥的凭据

4.2.3 多方选择性访问

为了管理多个方对同一内容的访问，增加了一层机制，用于从会话密钥派生访问密钥。在这种变体中，被授权人可以使用任意类型的凭证进行认证，但是，如上所述派生的会话密钥不会直接用作解密引用的访问密钥。相反，两个密钥是从会话密钥派生的：一个查找密钥和一个访问密钥解密密钥。这些密钥通过将会话密钥分别与两个常数（0 和 1）进行哈希处理获得。

在授权访问时，发布者需要生成一个全局访问密钥来加密完整的引用，然后使用每个被授权人的访问密钥解密密钥对其进行加密。随后，创建一个查找表，将每个被授权人的查找密钥映射到他们的加密访问密钥。然后，对于每个查找密钥，访问密钥将使用相应的访问密

钥解密密钥进行加密。

这个查找表以访问控制 Trie（ACT）的形式实现，并采用 Swarm 清单格式。ACT 中的路径对应查找密钥，并且清单条目包含加密访问密钥的密文，作为元数据属性值。ACT 清单是一个独立的资源，通过 URL 引用，并且包含在根访问元数据中，指示是否使用 ACT。

在访问内容时，用户遵循以下步骤：首先，用户检索根访问元数据，识别 ACT 资源，然后使用他们的密码短语和 scrypt 参数，或者使用发布者的公钥、私钥和盐值计算会话密钥。根据会话密钥，用户通过与 0 哈希处理来派生查找密钥，然后从 ACT 中检索清单条目。为此，他们需要知道 ACT 清单的根，并使用查找密钥作为 URL 路径。如果条目存在，用户将获得访问密钥属性的值，以密文形式呈现，并通过将会话密钥与常数 1 哈希得到的密钥解密。然后，得到的访问密钥可以用来解密根访问清单中包含的加密引用，如图 4.9 所示。一旦解锁了清单根，所有引用都将包含解密密钥。

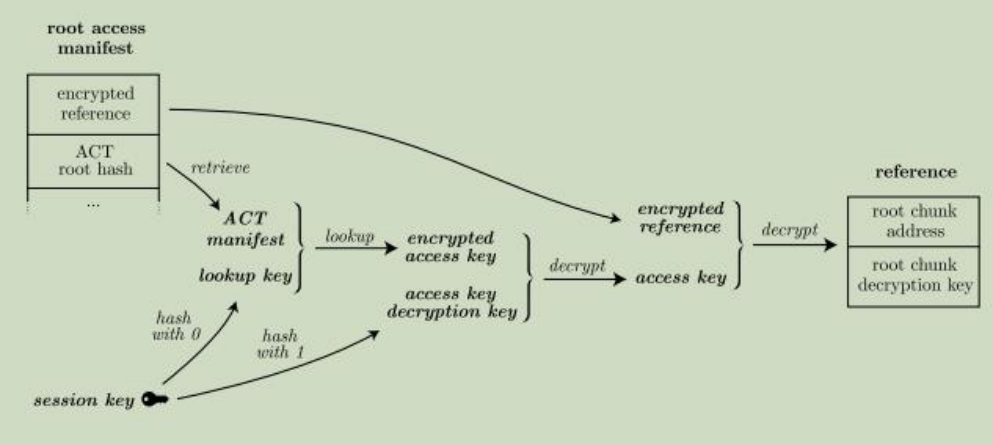

图 4.9：针对多个被授权者的访问控制涉及从会话密钥到访问密钥的额外层级。每个用户必须查找专门为其加密的全局访问密钥。用于查找的密钥和解密访问密钥的密钥均从会话密钥派生而来，而会话密钥又需要他们的凭证。

该访问控制方案提供了几个理想的特性： — 检查和查询自己访问权限的操作在 ACT 大小上是对数级别的。 — ACT 的大小仅提供了授权人数量的上限，而不会向第三方泄露关于授权人集合的任何信息。即使是 ACT 中的成员也只能知道自己是授权人，但无法获知其他授权人的信息，除了授权人数量的上限之外。 — 授予一个额外密钥的访问权限只需通过增加一个条目来扩展 ACT，这个操作在 ACT 大小上是对数级别的。 — 撤销访问权限需要更改访问密钥，因此需要重建 ACT。注意，这也要求发布者在 ACT 初始创建后保留授权人的公钥记录。

4.2.4 访问层次结构

在最简单的情况下，访问密钥是一个对称密钥。然而，这只是更灵活解决方案的特例，其中访问密钥由一个对称密钥和一个密钥派生路径组成，该密钥通过该路径从根密钥派生出来。在这种情况下，除了加密的引用外，还可以包括一个派生路径。任何持有访问密钥且其派生路径是引用派生路径前缀的一方，都可以通过使用他们自己的密钥和引用的其余派生路径来解密引用。

这允许形成一个类似树状的角色层次结构，可能反映组织结构。只要角色变更是“晋升”，即结果是增加权限，那么对于每个角色变更，只需修改一个 ACT 条目即可。

4.3 Feeds：在不可变存储中的可变性

Feeds 是 Swarm 的一个独特功能，构成了单一所有者块的主要用例。它们有广泛的应用，包括可变资源的版本修订、索引按顺序更新的主题、发布流中的部分内容，或在通信频道中发布连续的消息。Feeds 实现了持久化拉取消息机制，也可以解释为一个发布-订阅系统。

在 4.3.1 节中，我们介绍了如何将 feeds 组成单一所有者块，并使用索引方案。我们在 4.3.2 节中讨论了索引方案的选择。接着，我们深入探讨了 feed 完整性的重要性以及在 4.3.3 节中验证和执行其完整性的方法。4.3.4 节描述了基于时期的 feeds，这些 feeds 提供了一个搜索机制，用于接收间歇性更新的 feeds。最后，在 4.3.5 节中，我们展示了如何将 feeds 用作发送和接收通信频道中后续消息的发件箱。

4.3.1 Feed 块

Feed 块是一个单一所有者块，其标识符由 feed 主题的哈希值和 feed 索引组成。主题是一个 32 字节的任意字节数组，通常是一个或多个指定 feed 主题及可选子主题的可读字符串的 Keccak256 哈希值（参见图 4.10 的可视化表示）。

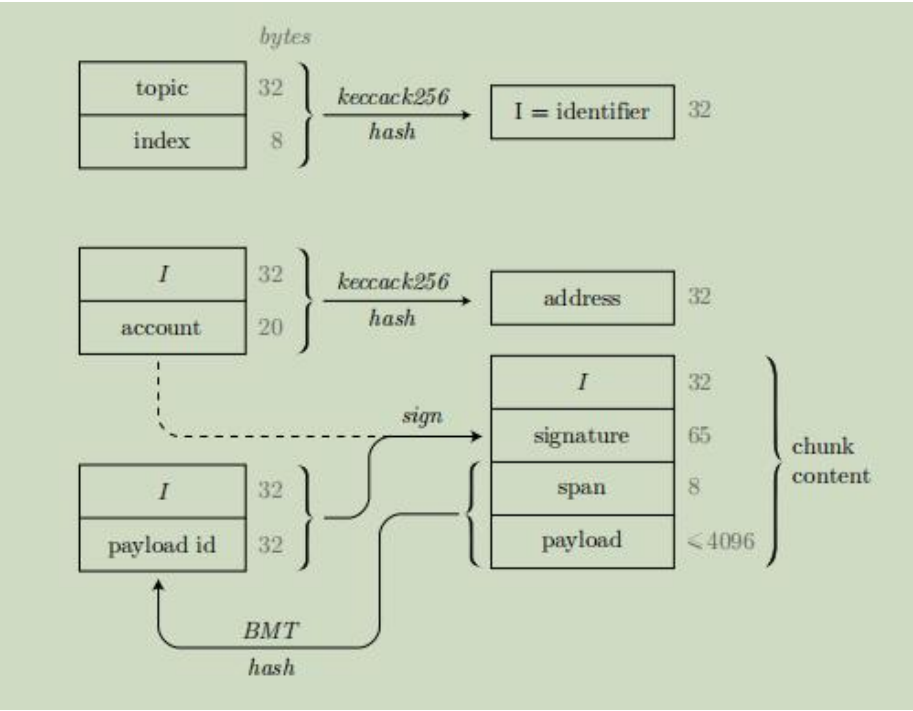

图 4.10：Feed 分块是单一所有者分块，其标识符是主题哈希和索引的组合。索引是根据索引方案计算出的确定性序列。同一主题的后续索引代表 Feed 更新的标识符。

Feed 的索引可以有多种形式，定义了几种可能的 feed 类型。本节讨论的有：

使用增量整数作为索引的简单 feed（4.3.2）；

使用时期 ID 作为索引的基于时期的 feed（4.3.4）；

使用通过双重齿轮链生成的随机数作为索引的私有频道 feed（4.3.5）。

这些 feed 类型的共同特征是，发布者（所有者）和消费者必须了解索引方案，才能有

效地与 feed 进行交互。

发布者对 feed 块拥有独占的所有权，并且只有发布者有权向其 feed 发布更新。发布更新需要：(1) 从主题和正确的索引构造标识符，(2) 签名标识符并将其与更新内容的哈希值连接。由于标识符指定了所有者子地址空间中的地址，因此此签名有效地将负载分配到该地址（参见 2.2.3）。通过这种方式，可以验证在特定 feed 上发布的所有项是由相应私钥的所有者创建的。

在消费者端，用户可以通过指定块地址来检索 feed。检索特定更新要求消费者从所有者的公钥和标识符构造地址。为了计算标识符，用户需要两个信息：主题和适当的索引，对于这两个信息，用户需要知道索引方案。

Feeds 使 Swarm 用户能够表示一系列内容更新。每个更新的内容作为负载，feed 所有者针对标识符进行签名。负载可以是一个 Swarm 引用，允许用户检索相关数据。

4.3.2 索引方案

不同类型的 feeds 需要不同的索引方案和不同的查找策略。在以下各节中，我们介绍了几个主要独立的维度，通过这些维度可以对 feeds 进行分类，并且在做出选择时可能会考虑到这些维度。

实际使用的索引方案，甚至是否存在索引方案（即单一所有者块是否为 feed 块），在 feed 块的结构中未被揭示，因为对于转发节点来说，这些信息在验证块时并不需要。明确将子类型包含在结构中只会导致不必要的信息泄露。

更新语义

Feed 的更新可以分为三种子类型，每种子类型有不同的语义。

表示同一语义实体修订的 feeds 称为可变资源更新。这些资源发生变化是因为底层语义实体发生了变化，例如更新个人简历或扩展资源描述（如关于罗马皇帝的维基百科条目）。用户通常对这些资源的最新更新感兴趣，过去的版本仅具有历史意义。正如“修订”这一术语所暗示的，针对可变资源的更新解释是替代性的。

第二种子类型的 feeds 被称为系列更新，表示通过共同的线索、主题或作者联系起来的一系列内容。在一个系列中，每个更新都被视为一个替代的、独立的实例或剧集，这些实例或剧集按时间顺序展开，例如社交媒体状态更新、个人博客文章或区块链的区块。系列更新呈现了一个连贯的叙事或信息进展，使得用户可以以顺序和相互关联的方式与内容进行互动。

最后，还有表示分区的 feeds，其中更新是累积性的：后续更新按顺序添加到先前的更新中。一个常见的例子是由多个部分组成的视频流。分区 feed 主要与系列更新不同之处在于，单个 feed 更新本身并没有独立的意义或可解释性。相反，更新的时间序列可能表示一种处理顺序，该顺序对应于资源结构的某种序列化，而不是反映时间上的继承关系。在访问分区 feed 时，可能需要累积所有部分才能确保表示资源的完整性。这意味着每个更新都是建立在之前更新的基础上，从而形成整个资源的全面表示。

如果 feed 的后续更新包含对数据结构的引用，该数据结构用于索引先前的更新（例如，使用更新的时间戳作为键值存储，或简单地使用更新内容连接的根哈希），那么所有三种 feed 类型的查找策略就简化为检索最新的更新。

更新频率

随时间更新的 feeds 可以分为几种类型。有些是间歇性更新的 feeds，具有不规则的异步性，即更新可能存在不可预测的间隔。另一种类型是定期更新的 feeds，其中更新以定期的间隔发布。

此外，我们还将讨论实时更新的 feeds，其中更新频率可能没有规律的模式，而是在实时人类交互的时间跨度内变化，即更新间隔在秒到分钟的范围内断断续续地发生。

订阅

Feeds 可以被解释为提供持久化的发布/订阅（pub/sub）系统，支持异步拉取。在接下来的讨论中，我们将分析索引方案的选择如何影响作为 pub/sub 实现的 feeds 订阅。

为了满足订阅者的需求，需要跟踪更新。当我们知道最新的更新时，可以使用定期轮询来获取后续更新。如果 feed 是定期更新的，可以在已知的时间周期之后开始轮询。或者，如果 feed 更新的频率足够高（最多比所需的轮询频率低一个数量级的数字），那么轮询也是可行的。然而，如果 feed 是间歇性更新且更新发生不可预测，那么轮询可能不切实际。在这种情况下，为了确保及时更新，推送通知（参见 4.4.1 和 4.4.4）等替代方法变得更加可取。

如果我们由于离线错过了一段时间的轮询，或是刚刚创建了订阅，也可以依赖推送通知或使用查找策略来检索所需的更新。

查找分区不会遇到困难，因为每个更新需要被提取并累积。在这种情况下，仅仅遍历连续的索引策略无法得到改进。对于定期更新的 feeds，我们可以简单地根据给定时间计算索引，因此异步访问是高效且简单的。然而，查找间歇性更新的 feed 的最新版本则需要一些搜索，因此会受益于基于时间周期的索引。

聚合索引

一组间歇性的信息流可以通过信息流聚合转换为周期性的信息流。例如，想象一个类似 Reddit 的多用户论坛，每个注册用户都会在帖子上发布评论，使用间歇性的信息流。在这种情况下，如果每个用户都要监控其他所有用户的评论信息流，并在他们的间歇性信息流中查找更新以获取线程中的所有评论，那将是非常不切实际的。更高效的方法是只为所有用户执行一次聚合。索引器正是这样做的：它们将每个人的评论聚合到一个索引中，这个数据结构的根可以作为一个周期性的信息流发布，见图 4.11。更新的频率可以选择为提供实时信息流体验；即使更新的变化率不足以支持这一点，也就是说，某些更新可能是多余的，但这种成本会分摊到所有使用聚合信息流的用户上，从而使其在经济上可持续。

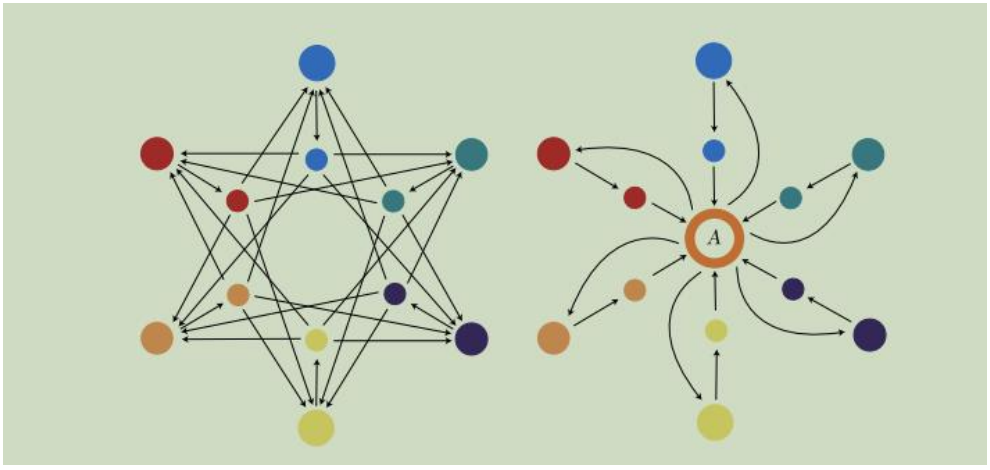

图 4.11：Feed 聚合用于合并来自多个源 Feed 的信息，以避免消费者重复工作。左图：参与群组通信的 6 个节点（讨论帖子、实时聊天或异步电子邮件线程）。每个节点将其贡献发布为发件箱 Feed 更新（彩色小圆圈）。每个参与者轮询其他节点的基于 epoch 的 Feed，在查找过程中重复工作。右图：这 6 个节点现在注册为聚合器的源，聚合器轮询节点的 Feed 并创建索引，将源聚合到一个数据结构中，每个参与者随后可以拉取该数据结构。

这种服务可以在任意的安全级别下提供，且是无需信任的，不依赖于声誉。通过使用共

识数据结构进行聚合，可以通过负担得起且简洁的包含证明来证明不正确的索引（见 4.1.1），因此与正确性相关的任何挑战都可以在链上进行评估。如果挑战未被反驳，提供者将面临失去押金的风险，这为他们维持高标准的服务质量提供了强有力的激励。

4.3.3 完整性

我们认为一个信息流是完整的，当它的每个更新都是不含歧义的。从形式上来说，这意味着对于每个索引，相应的信息流标识符只会被分配给一个有效负载。顺便提一下，这也意味着相应的信息流区块具有完整性。如 2.2.3 节所讨论的，这是保持一致性检索的前提条件。如果连续更新的有效负载被想象成区块链中的区块，那么完整性的标准要求信息流拥有者避免在其链中创建分叉。

事实上，信息流的完整性只能由信息流的拥有者保证。然而，重要的是要考虑是否可以有效地检查或强制执行其完整性。信息流的拥有者可以通过在区块链上质押押金来承诺信息流的完整性，如果他们在更新上双重签名，就会失去这笔押金。虽然这在长期内可能为避免分叉信息流提供了强有力的反激励，但仅凭这一点不足以向信息流的消费者提供关于完整性的充分保障。因此，我们必须设计能够主动强制执行所需完整性标准的索引方案。

权威版本历史

可变资源更新信息流的版本跟踪与以太坊名称服务的方式非常相似。当一个版本被巩固时，比如一个网站更新，所有者希望注册当前版本的内容地址。为了保证历史没有争议，负载需要包含前一个负载的哈希值。这一要求意味着负载必须是一个复合结构。然而，如果目标是让负载仅由清单或清单条目组成，以便它可以被绑定到一个 URL 路径或直接显示，那么这是不可行的。此外，如果在某些情况下，信息流内容不是负载哈希，ENS 即便没有对应的 Swarm 区块，仍然注册了负载哈希，这就违反了 ENS 的语义。

一个将前一个负载哈希纳入后续索引的索引方案类似于区块链的运作方式。它表达了所有者对特定历史的明确承诺，并要求任何读取并使用该信息流的消费者承认并接受该历史。查找这样的信息流只能通过检索自上次已知更新以来的每个更新来实现。地址指向更新区块，因此注册更新地址既保证了历史完整性，也保留了 ENS 语义，使注册的地址仅仅是指向区块的 Swarm 引用。这种信息流建立了一个权威的版本历史，也就是说，它们提供了一个安全的审计轨迹，记录了对可变资源所做的修订。

实时完整性检查

一个确定性索引的信息流提供了执行实时完整性检查的能力。在代表区块链（账本/侧链）的信息流的上下文中，完整性指的是拥有一个不分叉且唯一的链承诺。能够在实时强制执行这一点，使得交易终结的定义既快速又安全。

我们通过一个链下点对点支付网络的例子来说明这一点，其中每个节点的锁仓资金被分配给一组固定的债权人（更多细节见 Trón 等，2019a）。一个节点的债权人需要验证重新分配的准确性，即总增加额是否由经过签名的减少额所覆盖。如果一个债务人不断发布一个包含所有债权人的存款分配表，通过向目标债权人发出两个替代方案，债务人将能够操纵双重支付。相反，如果这个分配表的唯一性得到确认，债权人就可以自信地得出交易终结的结论。

我们声明，使用 Swarm 信息流，唯一性约束可以在实时中得到验证。

关键的洞察是，无法有意义地控制对单个拥有者区块请求的响应：即使攻击者控制了整个区块地址的邻域，也没有系统的方法对特定请求者给出特定版本的响应。这是由于转发的 Kademlia 协议中请求发起者的固有模糊性。假设攻击者使用一些复杂的流量分析，有 1/n 的

机会识别出发起者并给出不同的响应。然而，通过从随机地址发送多个请求，可以测试完整性，并将一致性响应视为确认终结性的要求。攻击者能够对测试的债权人通过 k 个独立请求提供一致的不同响应的概率是 1/n^k。随着 k 增加，带宽成本线性增长，但我们可以获得关于更新唯一性的指数级确定性。如果债权人观察到响应的一致性，他就可以得出结论，确认没有替代的分配表。

通过要求分配表作为信息流更新进行传播，我们可以利用去权限化、可用性和匿名性的优势来强制执行信息流的完整性。如果信息流是类似区块链的账本，那么实时完整性检查就转化为分叉终结。

4.3.4 基于时代的索引

为了使用单一拥有者区块实现具有灵活更新频率的间歇性信息流，我们引入了基于时代的索引方案。在该方案中，单一拥有者区块的标识符包含与发布时间相关的锚点。为了能够找到最新的更新，我们引入了一种自适应查找算法。

时代网格

时代是一个从特定时间点（称为时代基准时间）开始并持续一定时长的定义时间段。这些时间段的长度以 2 的幂表示，单位为秒，最短的周期是 2^0 = 1 秒，最长的周期为 2^31 秒。

时代网格是时代的排列方式，其中每一行表示将时间分割成多个长度相同、不重叠的时代。行，也称为层级，通过时代长度的对数进行索引，按约定，0 级的 1 秒周期位于最底部，见图 4.12。

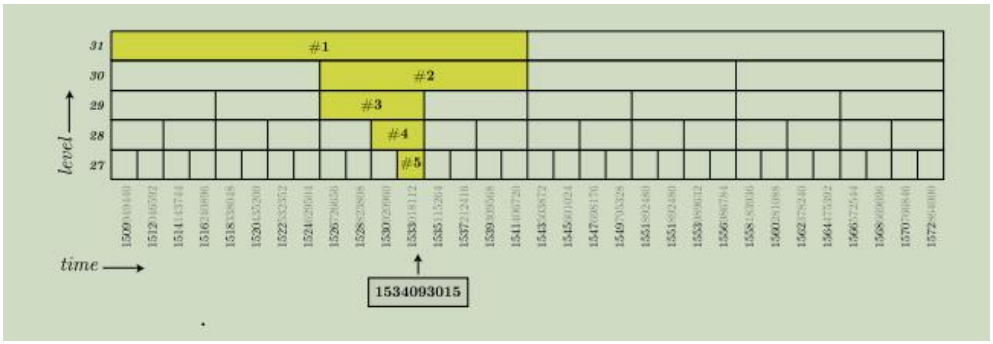

图 4.12：显示基于纪元的馈送的前几次更新的纪元网格。被占用的纪元以黄色标记，并编号以反映它们所代表的更新顺序。

当在时代网格中表示基于时代的信息流时，每个更新会根据其时间戳被分配到网格中的特定时代。具体来说，更新会映射到包含该时间戳的最长空闲时代。这种结构使得更新序列具有连续性，从而便于搜索。连续性要求意味着，通过知道前一个更新的时代，可以毫不含糊地将随后的更新映射到一个特定的时代。

要在时代网格中标识特定的时代，我们需要知道时代基准时间和层级。这对被称为时代引用。为了计算在特定层级 l 下某一时刻 t 的时代基准时间，我们需要去掉 t 的最低 l 位。层级需要一个字节，时代基准时间（使用 Linux 秒数）需要 4 字节，因此时代引用可以在 5 字节内进行序列化。值得注意的是，任何基于时代的信息流的初始更新的时代引用始终是相同的。

将时代映射到信息流更新区块

序列化的时代引用作为信息流索引用于将信息流更新映射到信息流区块。信息流主题与索引结合后，得到的信息流标识符用于构建表达信息流区块的单一拥有者区块。

为了确定存储随后的更新的适当时代，发布者需要知道前一个更新的位置。如果发布者没有跟踪此信息，他们可以使用查找算法来查找其最近的更新。

查找算法

当消费者检索信息流时，他们的目标通常是查看某个特定时间点的信息流状态（历史查找），或者获取最新更新。

如果需要基于目标时间的历史查找，更新可以包含一个数据结构，将时间戳映射到相应的状态。在这种情况下，可以通过查找任何比目标时间更晚的更新，来确定性地查找早期的状态。

如果没有可用的索引，历史查找需要找到时间戳早于目标时间的最短已填充时代。

为了选择一个最佳的起始时代来开始遍历网格，我们必须假设最坏的情况，即自从上次看到该资源以来，它从未被更新过。如果我们不知道资源上次更新时间，我们假设它为 0。

我们可以通过猜测一个起始层级来确定开始位置，这个起始层级是从左边开始计数，lastUpdate⊻NOW 的第一个非零位的位置。上次更新时间与当前时间之间的时间差越大，层级就越高。

4.3.5 实时数据交换

信息流可以用来表示一个通信通道，即一个个体的外发消息。这种类型的信息流，称为外箱信息流，可以创建来提供类似电子邮件的通信或即时消息传递，甚至是两者的结合。对于类似电子邮件的异步通信，可以使用基于时代的索引，而对于即时消息传递，更适合使用确定性序列索引。在群聊或群邮件中，机密性通过对数据结构进行访问控制 trie 管理，索引每个参与方对线程的贡献。通信客户端可以检索与特定线程相关的每个群组成员的信息流，并合并他们的时间线进行渲染。

即使是论坛也可以通过上述描述的外箱机制实现。然而，随着注册参与者的增加，在客户端侧聚合所有外箱可能变得不切实际。在这种情况下，可能需要使用索引聚合器或其他方案来众包数据的组合。

双向私密通道

双方私密通信也可以使用外箱信息流来实现，见图 4.13。这些信息流的参数在初始密钥交换或注册协议中确定（见 4.4.2），确保各方就索引方案和使用的加密方式达成共识。

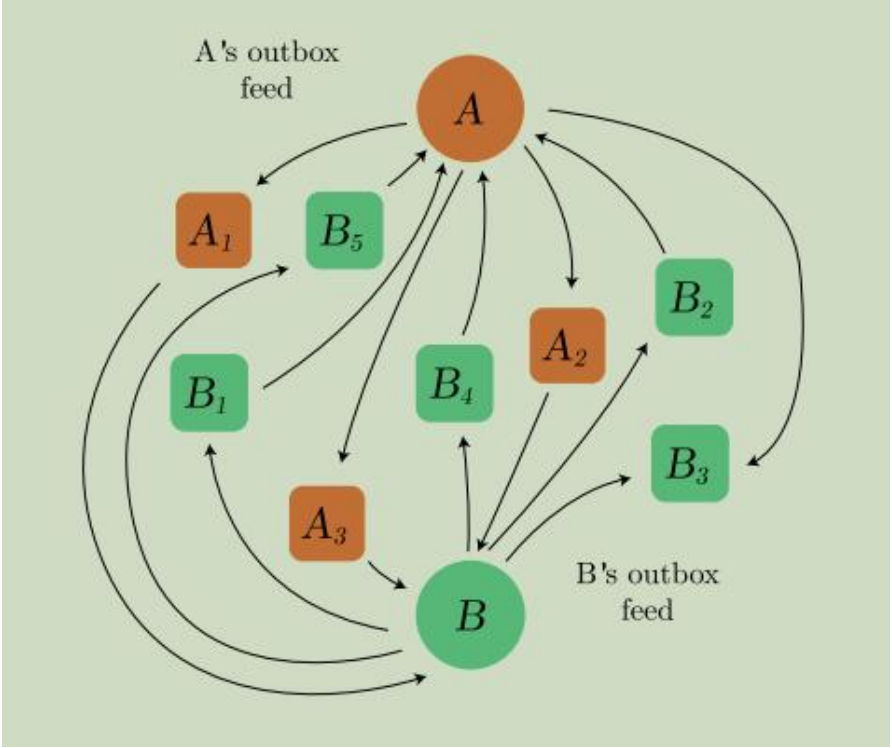

图 4.13：Swarm 消息流作为私密通信的发件箱。发件箱消息流代表对话中一方连续发出的消息。索引方案可以遵循一个具有强隐私性的密钥管理系统，该系统混淆了通信渠道本身，使得拦截攻击的成本高到难以承受。

对于使用系列信息流进行实时即时消息传递，重要的是拥有一个支持至少几个更新后的确定性继续的索引方案。这使得可以提前发送即将到来的更新的检索请求，即在处理前一条消息时或甚至在处理之前。当这些检索请求到达请求的更新地址附近的节点时，预计该区块将不可用，因为对方还没有发送它们。然而，即使是这些存储节点也会有动力保持检索请求活跃直到它们过期（如 2.3.1 所讨论）。这意味着，直到它们的生存时间（30 秒）结束，这些请求将充当订阅：更新区块的到达将触发对未完成请求的交付响应，就像它是发给订阅者的通知一样。这将把期望的消息延迟降低到不到单向转发路径平均时间的两倍，见图 4.14。

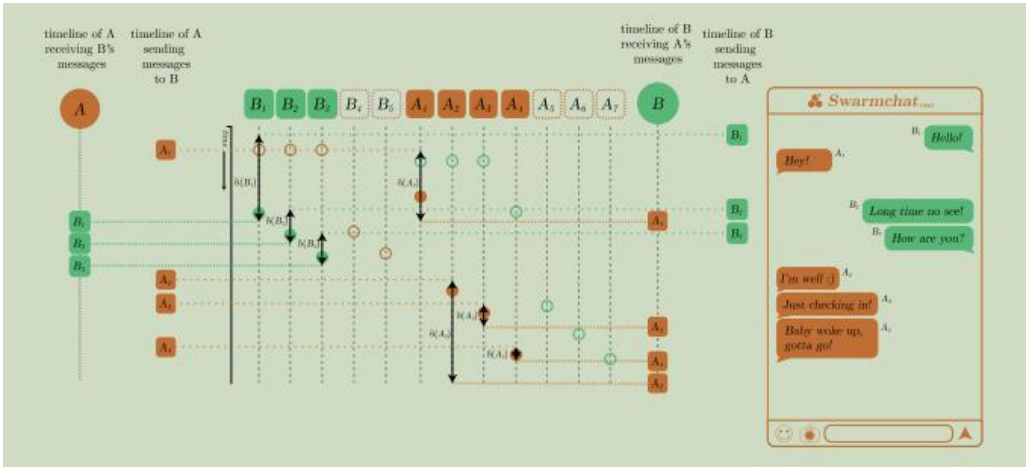

图 4.14：对未来更新的提前请求。该图表显示了两方 A 和 B 使用发件箱消息流进行即时消息传递期间事件的时间线。各列代表消息流更新地址的邻里位置。圆圈显示协议消息到达的时间：颜色表示数据的来源，空心圆圈是检索请求，实心圆圈是到达相应邻里的推送同步交付。请注意，发件箱地址是确定性的，比消息提前 3 条，允许在相应更新到达之前发送

检索请求。重要的是，一方发送消息 m 而另一方接收它的延迟显示为 δ (m)。消息 A3 和 A4 比 A2 先到达，这可以被报告并修复。如果地址可预测性仅限于提前 1 条消息，那么 B2 和 B3 都将有更长的延迟。还请注意，B2 和 B3 的延迟得益于提前请求：在收到 B1 和 B2 时发送 B4 和 B5 的检索请求，并分别与消息 B2 和 B3 到达它们的邻里的时间相同。如果地址可预测性仅限于提前 1 条消息，这将对 B2 和 B3 的延迟产生负面影响。

泄露后安全性

一种被称为“双梭锁”（double ratchet）的密钥管理解决方案是即时消息加密中的行业标准。通常，使用扩展的三重 Diffie – Hellmann 密钥交换（X3DH）来为双梭锁密钥链建立初始参数（见 4.4.2）。

双梭锁方法结合了基于连续密钥协商协议的梭锁和基于密钥派生函数的梭锁 (Perrin 和 Marlinspike 2016）。该方案可以推广（Alwen 等 2019），并且可以理解为多个已知原语的组合。它已被证明可以提供：（1）前向保密性，（2）后向保密性，（3）即时解密和消息丢失恢复能力。

除了端到端加密所提供的保密性外，Swarm 还提供了对攻击的进一步抵抗。由于使用了转发的 Kademlia 协议，发送者可以保持模糊和不可否认的状态。此外，正常的推送同步和拉取同步流量有助于混淆消息。为了使攻击者更难以破解，我们可以通过将更多的密钥链添加到双梭锁机制中，使索引序列也提供未来的保密性。除了根密钥、发送和接收加密密钥链外，还引入了两个附加密钥：外发和内收外箱索引密钥链，见图 4.15。通过这一措施，底层的通信通道被混淆化，即截获外发更新区块并知道其索引，无法揭示关于前后续外发更新索引的任何信息。这使得后续消息的监控或拦截变得极为困难且成本高昂。

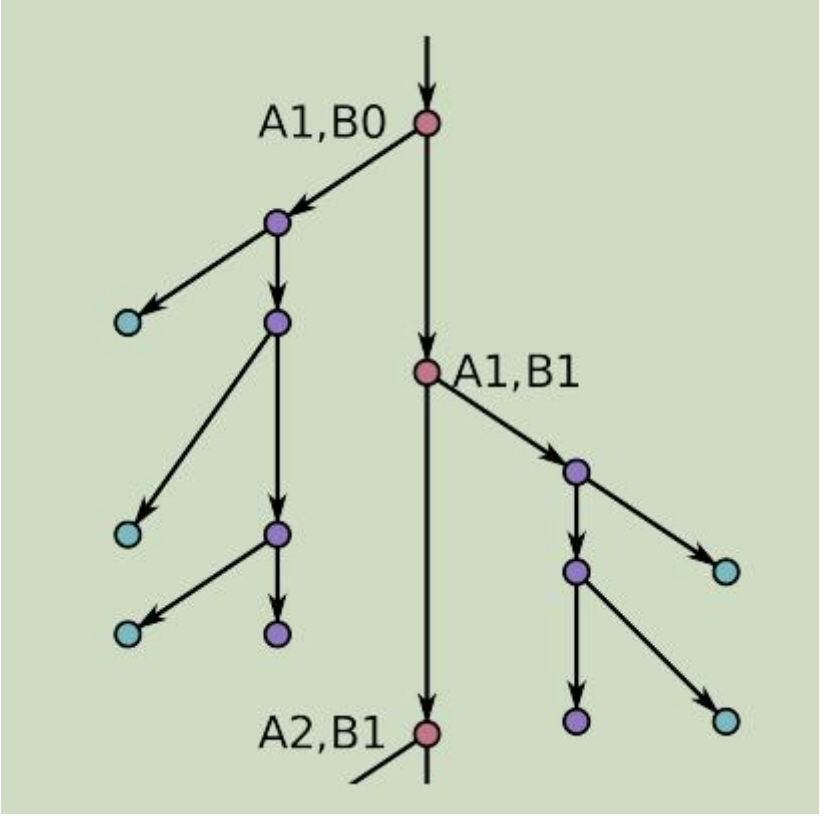

图 4.15：更新地址的未来保密性

在 4.3.3 中，我们通过将负载哈希纳入索引方案来实现链的不可合并性 (明确的历史)。受此启发，我们提出将负载哈希也纳入随后的信息流更新索引中。这引入了一个额外的属性，称为恢复安全性（recover security），直观地说，它确保一旦攻击者成功伪造了从 A 到 B 的

消息，那么 B 将不会接受任何来自 A 的未来消息。如果 A 发送给 B 的消息的真实性影响了随后的信息流索引，那么这一点得以保证。如果存在不匹配（表明伪造的消息），则消息将在错误的地址进行查找，导致通信通道的废弃并启动一个新的通道。通过实施这种方法，通信通道实现了完全的保密性，并成为实时消息传递的零泄漏解决方案。

4.4 Pss：带邮箱功能的直接推送消息传递

本节介绍了 pss，这是 Swarm 的直接节点对节点推送消息传递解决方案。其功能和存在动机通过对该术语的不同解释形式得到了生动的呈现：

Swarm 上的邮政服务

在接收方在线时传递消息，如果不在线则存储消息以便下载。

pss 是 bzz 低语

除了与“中文耳语”相关的联想，pss 无疑承载了以太坊 Whisper 的精神和愿景。pss 依赖 Swarm 的分布式存储来存储块，因此继承了其转发和持久性的完整激励机制。同时，它还借用了 Whisper 的加密、信封结构和 API。

pss! 指令以保持安静/低语

这唤起了不向第三方透露信息的努力，这正是 pss 的标语所传达的：真正的零泄漏消息传递，在这里，除了匿名性和保密性外，消息传递的行为本身也是不可检测的。

发布/订阅系统

该 API 允许发布和订阅某个主题。

首先，在 4.4.1 中，我们介绍了 Trojan Chunks（特洛伊块），这是一种发送给存储者的消息，它伪装成块，其内容地址恰好位于其预定接收者的附近。4.4.2 节讨论了使用 pss 发送联系人消息，以建立实时通信通道。在 4.4.3 中，我们探讨了通过挖掘信息流标识符来定位特定邻域，使用单个所有者块的地址，并呈现了一个有地址的信封结构。最后，在特洛伊块和有地址信封的基础上，4.4.4 节介绍了更新通知请求。

4.4.1 特洛伊块

承诺提供私密消息传递的前沿系统往往难以实现真正的零泄漏通信 (Kwon 等，2016)。尽管加密证明了链接发送者和接收者是不可能的，但抵抗流量分析则更加难以实现。保持足够大的匿名集需要随时提供大量流量。在缺乏广泛采用的情况下，保证专用消息网络中的高消息率需要不断产生虚假流量。然而，借助 Swarm，可以将消息伪装成块流量，有效地混淆消息传递的行为本身。

我们定义“特洛伊块” (Trojan chunk) 为一种具有固定内部内容结构的内容寻址块（见图 4.17）：

1 span

8 字节小端格式的 uint64 整数，表示消息的长度

2 nonce

32 字节的任意 nonce

3 Trojan message

4064 字节不对称加密的消息密文，其底层明文由以下部分组成：

2 字节小端格式的消息长度编码，长度为 $0 \leqslant l \leqslant 4030$

32 字节模糊化的主题 ID
消息的 m 字节
4030−m 字节的随机字节

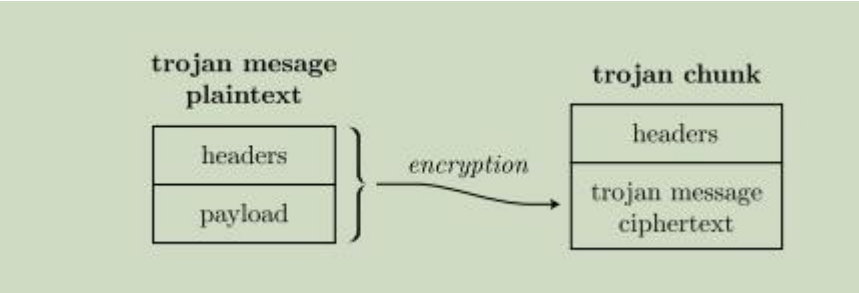

图 4.16：pss 消息是一个特洛伊分块，它用特洛伊消息包装了一个模糊化的主题标识符，而特洛伊消息又包装了实际的消息负载，该负载将由处理它的应用程序解释。

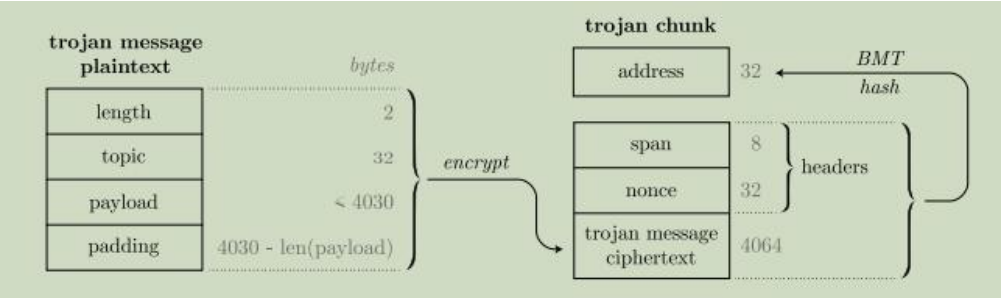

图 4.17：特洛伊分块包裹了一条非对称加密的特洛伊消息。

知道接收者的公钥后，发送者按照特定的过程发送特洛伊消息。首先，消息被封装为特洛伊消息格式，通过在消息前添加长度信息并将其填充至 4030 字节。然后，发送者使用接收者的公钥对消息进行加密，获得通过非对称加密生成的特洛伊块的密文载荷。接着，发送者生成一个随机 nonce，这样当它被添加到载荷前时，块的哈希值就会以目标地址的前缀开始。目标地址是一个比特序列，表示地址空间中的特定邻域。如果目标是接收者覆盖地址的前缀生成的部分地址，则匹配目标意味着该块落入接收者的邻域。如果仅知道公钥，则假设它是接收者的 bzz 账户，即可以从公钥计算出其覆盖地址（见 2.1.2）。随后，发送者将生成的块上传到 Swarm，并选择适当的邮资印章，这样块就会同步到接收者地址的邻域。如果接收者节点在线，并且匹配目标的比特长度大于接收者的邻域深度，则他们将接收到该块。在实践中，目标的比特长度应为 n + c，其中 n 是 Swarm 中估算的平均深度，c 是一个小的整数。

接收特洛伊消息

接收者只有在使用与其公钥对应的私钥成功解开特洛伊消息后，才能知道一个块是一个 pss 消息（见 4.4.2），并且进行完整性检查和主题匹配。希望接收此类特洛伊消息的节点将继续尝试解密它们最接近的所有消息。转发节点（或除发送者和接收者之外的任何其他节点）无法区分随机加密的块和特洛伊消息，这意味着通信完全以通用块流量的方式进行混淆。

接收者使用非对称解密打开信封后，进行完整性检查和主题匹配的结合步骤。通过消息的前两个字节获知载荷的长度后，接收者取出载荷片段并计算其 Keccak256 哈希值。对于客户端订阅的每个主题，接收者将载荷哈希与主题进行哈希运算。如果结果段与主题进行异或操作后，匹配消息中的模糊主题 ID，则说明消息确实是针对该主题的，注册的处理程序将以载荷作为参数被调用。

异步投递的邮件箱功能

如果接收者不在线，特洛伊块将像任何其他块一样存在，具体取决于它所附带的邮资印章。每当接收者节点上线时，它会从最接近的邻域拉取同步块，其中包括所有特洛伊块和它自己未接收的消息。换句话说，通过特洛伊消息，pss 自动提供了异步邮件箱功能，允许未投递的消息得以保存，并在接收者上线时无须发送者的额外操作即可被接收。邮件箱功能的持续时间由邮资印章控制，实际上，它与常规块存储没有区别。

寻找邻近性

找到接收者地址附近的哈希过程类似于区块链上的区块挖矿。特洛伊块中的 nonce 段也具有与区块链区块 nonce 完全相同的功能：它提供足够的熵以保证解答的出现。挖矿的难度与目标地址的长度相关：确保接收者能够收到消息所需的最小邻近度阶数需要高于接收者上线时的邻域深度，因此它是网络中节点数量的对数。每个特洛伊消息在找到适当的内容地址之前需要尝试的 nonce 数量在难度上是指数级的，因此等于网络中节点的数量。实际上，即使是单个节点，挖掘特洛伊块也永远不会变得过于昂贵或缓慢，因为找到 nonce 所需的计算周期的期望次数等于网络的大小。在拥有十亿个节点的网络中，可能会出现第二个范围的轻微延迟，但考虑到特洛伊消息仅用于启动渠道等一次性实例，这种延迟是可以接受的。随后的实时交换将使用先前描述的双向外箱模型，通过单一所有者块进行。

匿名邮件箱

只要邮资印章未过期，就保证可以异步访问 pss 消息。接收者只需要创建一个与目标地址相对应的覆盖地址节点，作为接收者的常驻地址。这使得可以创建一个匿名邮件箱，代表客户端接收 pss 消息，并将其发布到一个独立的私密信息流中。预期的接收者随后可以在他们重新上线时阅读这些消息。

注册聚合索引

如 4.3.2 节所述，聚合索引服务帮助节点监控间歇性的内容流。例如，论坛索引器可以聚合注册会员的贡献内容流，从而高效跟踪和访问论坛更新。在公开论坛的情况下，链外注册是一种可行的选项，允许用户在不直接与区块链交互的情况下进行注册。这可以通过简单地向聚合器发送一条 pss 消息来实现。

4.4.2 密钥交换的初始联系

加密通信需要一个握手协议来建立初始参数，这些参数将作为对称密钥生成方案的输入。扩展三重 Diffie－Hellman 密钥交换协议（X3DH）就是其中一种协议（Marlinspike 和 Perrin 2016）。X3DH 用于建立握手后的通信协议的初始参数，如本节关于信息流的讨论中提到的双重齿轮（double-ratchet）方案（见 4.3.5）。

为了在无服务器环境下实现 X3DH 协议，Swarm 使用与以太坊常用的相同基本工具，即 secp256k 椭圆曲线、Keccak256 哈希函数和 64 字节的 EC 公钥编码。

Swarm 的 X3DH 协议便于在两方之间建立共享密钥，作为用于确定双向通信过程中加密密钥的输入。发起方是启动双向通信的主体。响应方应当发布必要的信息，以便以前未知的各方可以发起联系。通过首先执行 X3DH 协议，可以实现零泄漏通信。该协议用于建立双重齿轮协议所使用的种子密钥，用于加密数据，以及使用的信息流索引方法。这将使响应方能够检索外发信息流的更新。

X3DH 使用以下密钥：

K_r^{ENS}

响应方的长期公钥身份

K_r^{Res}

响应方的常驻密钥（即签名的预密钥）

K_i^{ID}

发起方的长期身份密钥

K_i^{EPH}

发起方用于对话的临时密钥

一个预密钥包包含了发起方需要了解的关于响应方的所有信息。然而，这些信息并不是存储在外部服务器上，而是存储在 Swarm 中。为了便于身份管理，可以选择使用 ENS 提供基于用户名的身份。ENS 解析器的所有者代表了该人物的认证长期公钥身份。通过利用长期身份地址，可以创建一个基于纪元（epoch）的信息流，并附带一个主题 ID，表示该信息流提供了潜在通信方的预密钥包。当与新身份发起通信时，发起方从信息流中获取最新的更新，其中包含当前的常驻密钥（即签名的预密钥）和当前的住址，即该人物预期接收 pss 消息的（可能是多个）覆盖目标地址。信息流更新块中的签名会同时签署常驻密钥（参见签名的预密钥）和目标地址。通过该签名恢复的公钥即为长期身份公钥，见图 4.18。

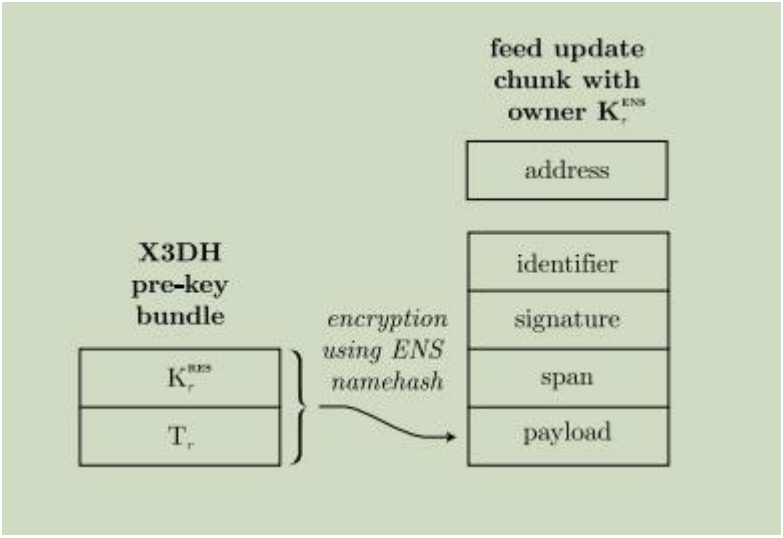

图 4.18：X3DH 预密钥包更新包含驻留密钥和驻留地址，并可选择与 ENS 名称哈希一起加密，以证明唯一性并提供身份验证。

为了邀请响应方加入基于外发信息流的私密通信频道，发起方首先查找响应方的公有预密钥包信息流，并向响应方发送一条初始消息（见图 4.19），表示其沟通意图。接着，发起方分享启动加密对话所需的参数，包括其长期身份的公钥和为该对话专门生成的临时密钥对的公钥。这些细节通过发送一条目标为响应方当前住址的特洛伊 pss 消息来传递给潜在的响应方，该住址也在响应方的预密钥包信息流中发布。

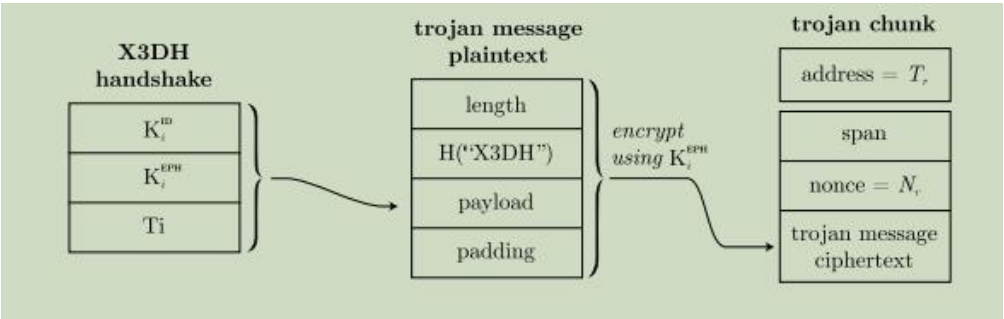

图 4.19：X3DH 初始消息。发起者检索 ENS 所有者以及包含驻留密钥和驻留地址的响应者预密钥包更新的最新信息。发起者使用驻留密钥加密，将自己的身份密钥和一个临时密钥发送到响应者的驻留地址。

在响应方收到这些信息后，双方都具备了生成三重 Diffie-Hellmann 共享密钥所需的所有要素（见图 4.20）。这个共享密钥构成了双重齿轮连续密钥协议的种子密钥，如信号协议中所使用的那样。双重齿轮方案确保了前向保密性和在泄露后安全性，从而保障端到端加密的安全性。通过为外发信息流的索引方案应用独立的密钥链，可以实现附加的恢复安全性，即抵御消息插入攻击。然而，更为重要的是，通过为外发地址增加前向和后向保密性，混淆了通信渠道，这使得对顺序消息的拦截依赖于与加密相同的安全假设。这消除了双重齿轮加密已知的唯一攻击面。基于外发信息流的通信渠道的混淆性和可否认性，加上初始 X3DH 消息被伪装，无法与常规块区分开，从而使这种通信方式被归类为零泄漏通信。

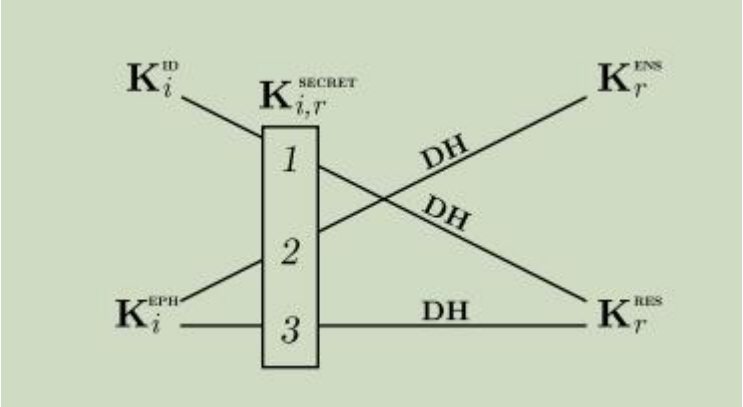

图 4.20：X3DH 密钥。双方都可以计算三重 Diffie-Hellmann 密钥并通过异或运算得到 X3DH 共享密钥，该密钥用作握手后协议的种子。

4.4.3 定向信封

挖掘单一所有者块地址

一个立刻引发的问题是，挖掘单一所有者块是否可行。由于此时地址是 32 字节标识符和 20 字节账户地址的哈希，因此 ID 提供了足够的熵来挖掘地址，即使所有者账户是固定的。因此，对于特定账户，如果我们发现一个 ID，使得生成的单一所有者块地址接近目标覆盖地址，那么该块就可以像特洛伊块一样作为消息使用。然而，重要的是，由于地址可以在块内容与其关联之前就进行挖掘，这种构造可以作为一个定向信封。

让我们明确这个构造中相关的角色：

发行者 (I)

通过挖掘地址来创建信封。

发布者 (P)

将内容放入信封中，并作为有效的单一所有者块发布到 Swarm。

所有者 (O)

拥有地址部分的私钥，因此可以签署负载与标识符的关联。这实际上决定了信封的内容。

目标 (T)

挖掘的约束条件：必须形成挖掘地址前缀的比特序列。它表示在覆盖地址空间中信封将被发送到的邻域。目标序列的长度对应于挖掘的难度。目标越长，信封能够到达的邻域就越小。

接收者 (R)

其覆盖地址具有目标序列作为前缀，因此是消息的接收方。

信封可以被视为是打开的：由于发布者也是响应的单一所有者块的所有者，因此他们能

够控制块中放入什么内容。通过构造这样的信封，发行者实际上允许发布者向目标发送任意消息，而无需消耗计算资源来挖掘该块。见图 4.21。

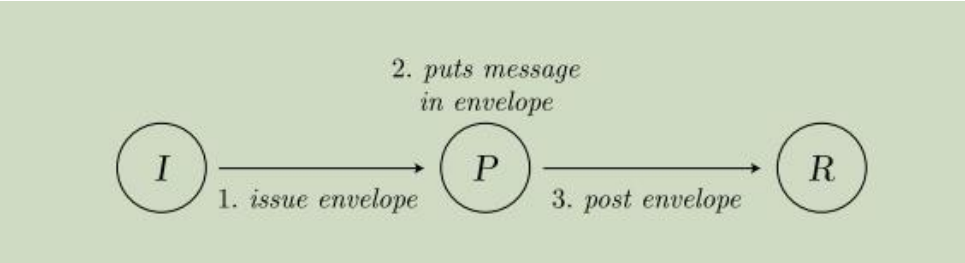

图 4.21：加盖地址的信封事件时间线。发行者 I 为 P 创建了一个加密信封，并使用一个标识符，使得 P 作为分块的唯一所有者，生成一个落在接收者 R 的邻域中的地址。因此，（仅）P 可以用任意内容填充信封，然后使用简单的推送同步将其发布给 R。

当发布者想要向接收者发送消息时，他们只需要创建一条特洛伊消息，并使用与发行者在挖掘地址过程中特定账户相同的私钥对标识符进行签名。这样生成的块就会是有效的。见图 4.22。

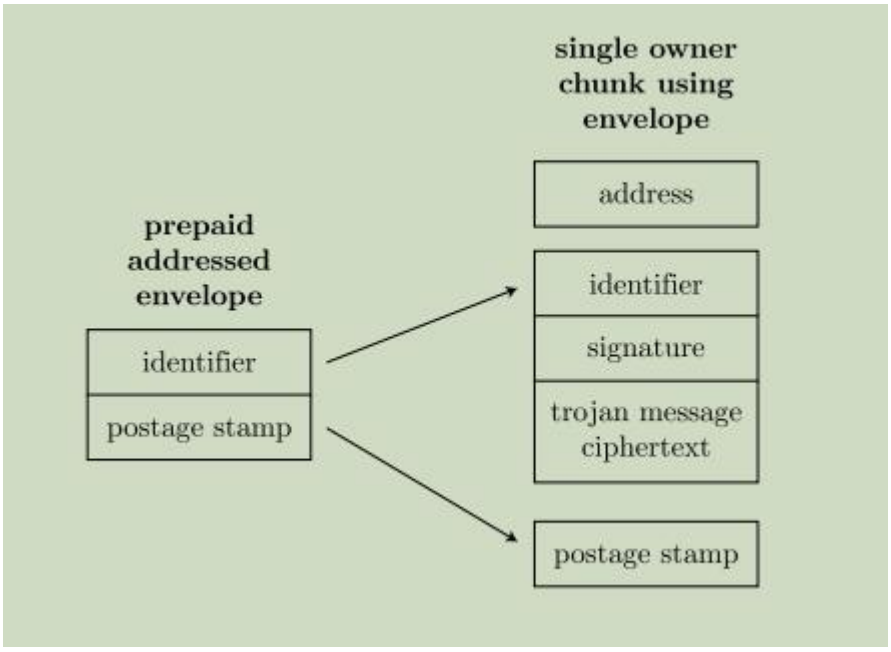

图 4.22：为 P 签发并寄给 R 的带邮票地址的信封包含一个通过挖矿生成的标识符，当该标识符用于创建由 P 拥有的单所有者分块时，它会生成一个落在 R 邻域内的地址。这使得 P 能够构建一条消息，使用该标识符对其进行签名，并利用邮票通过网络的正常推送同步功能免费将其发送给 R。

预付邮资

这些块表现出与普通特洛伊消息相同的行为，保持其隐私属性，在某些情况下，甚至改善了这些属性。发行者/接收者可以为加密消息关联一个随机公钥，或者使用对称加密。如果一个地址的邮资已经预付并交给某人稍后发布，他们可以使用推送同步将块发送到目标，而发布者无需承担任何额外费用，因为邮资已经通过加盖邮票的地址信封支付。这种构造有效地实现了带有预付邮资的地址信封，并作为各种高层通信需求的基础层解决方案，如：(1) 向订阅者发送推送通知，而发送者无需承担计算或财务上的邮资负担，(2) 免费的联系方式凭证，以及 (3) 零延迟的直接消息响应。

发布加盖邮票的地址信封

发布加盖邮票的地址信封涉及以下过程：

假设

发行者 I、预期发布者 P 和预期接收者 R 拥有公钥 KI、KP、KR 和覆盖地址 AI、AP、AR。

挖掘

I 找到一个随机数 NR，使得当其作为标识符用于创建单一所有者块时，块的地址哈希为 HR，且该地址位于 AR 的最近邻域中。

支付邮资

I 使用私钥签署 HR 以生成一个证据，表明已为适当的邮资支付生成邮票 PSR。

封装

将 NR 和 PSR（它们表示预付的、预先地址化的信封）打包，并使用 KP 对其加密，然后将其包装成一个特洛伊块。

挖掘

找到一个随机数 NP，使得该特洛伊块的哈希值为 HP，且该地址位于 AP 的最近邻域中。

接收加盖邮票的地址信封

假设预期发布者 P 接收到一个由预付邮资信封 E 组成的特洛伊消息。为了打开它，P 执行以下步骤：

解密

使用 KP 对应的私钥解密消息。

反序列化

解包并识别 PSR 和 NR，从 PSR 中提取 HR。

验证

验证邮票 PSR，检查 NR 与 KP 的账户哈希是否等于 HR，以确保关联的地址实际上是 P 拥有的。

存储

存储 NR 和 PSR。

发布加盖邮票的地址信封

当发布者希望使用该信封向接收者 R（接收者 R 可能对发送者是未知的）发送任意消息 M 时，他们必须按照以下步骤操作：

加密

使用 KR 对消息内容 M 进行加密，生成负载并将其包装成特洛伊消息 T。

哈希

对加密后的特洛伊消息进行哈希，得到 HT。

签名

使用 KP 对应的私钥对 HT 进行签名，生成签名 W。

封装

将随机数 NR 作为标识符，签名 W 和特洛伊消息 T 作为有效单一所有者块的负载，块的地址为 HR。

发布

发布该块，并附上有效的邮票 PSR。

接收发布的地址信封

当 R 接收到地址为 HR 的块时：

验证

验证邮票 PSR 并将块验证为一个有效的单一所有者块，负载为 T。

解密

使用属于 KR 的私钥解密 T。

反序列化

将明文反序列化为一个特洛伊消息，识别消息负载 M 并检查其完整性。

消费

消费 M。

4.4.4 通知请求

本节详细阐述了地址信封的概念，并提出了三种不同类型的通知实现方式。

来自发布者的直接通知

如果发行者希望通知接收者某个信息流的下一活动，她需要构造一个带邮票的地址信封，将其嵌入到一个常规的特洛伊消息中，并发送给发布者，如图 4.23 所示。如果发行者也是接收者，则可以在请求和响应信封中使用相同的账户。

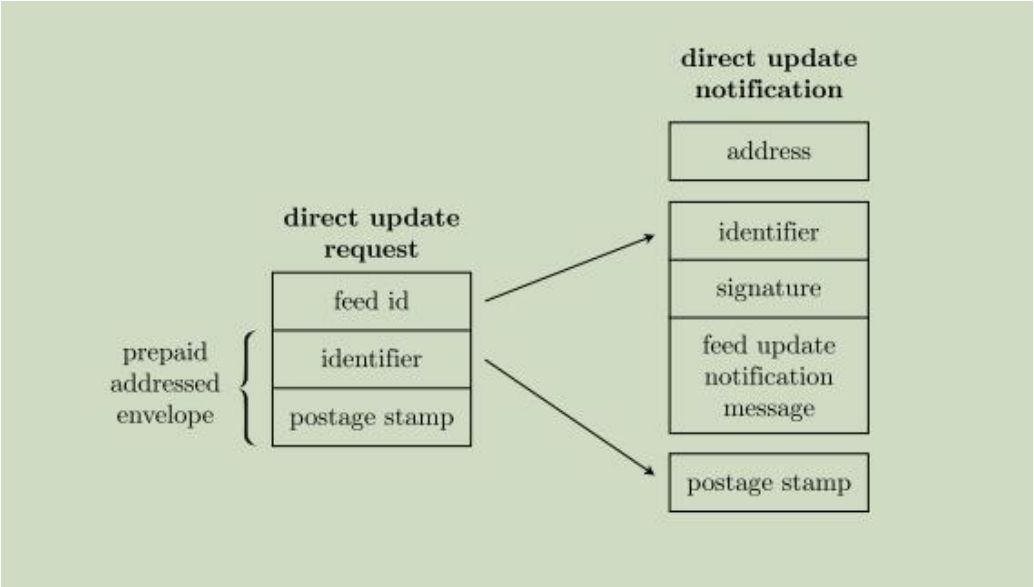

图 4.23：直接通知请求包含对 feed 的引用，并封装了一个预付费信封，该信封是为 P（发布者或 feed 的已知分发者）挖掘的，并发送给接收者 R。响应遵循与通用带邮资地址信封相同的过程，唯一的区别在于消息预期为 feed 更新或其内容的引用。

当信息流的拥有者发布更新时，他们将在信封中包含对更新块的引用，并将其发送给接收者。更正式地说，发布者使用预先地址化信封的标识符创建一个单一所有者块，并签署与信息流更新内容相关的标识符，将其作为单一所有者块的负载。随后，发布者（现在作为发布者）将该块推送同步到 Swarm。这一过程称为来自发布者的直接通知。

特洛伊消息使用公钥加密，并在其主题中指定它是一个通知。由于地址被挖掘以匹配接收者的覆盖地址并与足够长的前缀匹配，消息最终会推送同步到接收者的邻域。当接收者上线并接收到该块时，她会检测到它是作为消息发送的。这可以通过使用他们之前宣传的地址的密钥成功解密块内容，或者在接收者自己发布了预先地址化信封的情况下，通过查找他们在发布时保存的地址记录来完成。请参见图 4.24 了解事件的时间线。

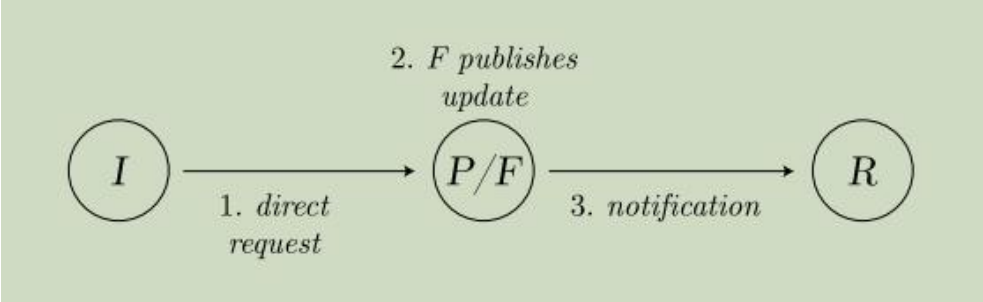

图 4.24：发布者事件时间线的直接通知。发行者 I 为发布者或已知的订阅源分发者 P/F 构建一个预付信封，并发送给接收者 R。它与订阅源主题 I 一起，被封装在一个 pss 木马消息中发送给 P/F。P/F 接收并存储该请求。当他们发布更新时，将其封装在信封中，即针对订阅源更新通知消息签署从 I 接收到的标识符，并将其作为一个分块发布，R 将接收到该分块，从而被通知订阅源的更新。

来自发布者的直接通知允许发布者在信封中包含任意内容。发布者，作为信封的拥有者，有权在发布信封时签署任何内容以与标识符对接。为了提前创建通知块地址，发行者需要知道预期发布者的账户。然而，信息流更新地址并不需要是固定的，这使得该方案适用于（间歇性）基于纪元的信息流。

来自邻域的通知

考虑一个情形，其中信息流的拥有者选择不公开其覆盖地址（目标地址），或拒绝实现通知功能。在这种情况下，信息流使用简单的顺序索引进行不定期更新，这使得依赖轮询检查最新更新变得不切实际。此外，消费者也可能随时下线。那么，是否有方法确保消费者仍然能够收到通知?

我们引入了另一种构造，称为“邻域通知”，它在通知发布者不知道潜在发布者身份的情况下工作。然而，它要求事先知道内容或其哈希值，以便由发布者本人签名。

为了确保接收者能够收到下一次更新的通知，发布者可以通过将消息封装在特洛伊块中来创建邻域通知请求。该消息包含一个带地址的信封（标识符、签名和邮票），发布者可以使用该信封构建作为通知的单一所有者块。请注意，通知消息无需包含任何新信息；仅仅接收到它对接收者来说就足够了。通知的负载包含信息流更新地址，以指示这是什么通知。当接收者收到通知后，他们可以简单地发送一个常规的检索请求，以获取实际的信息流更新块，其中包含（或指向）消息的内容。参见图 4.25 了解邻域通知和通知请求的结构。

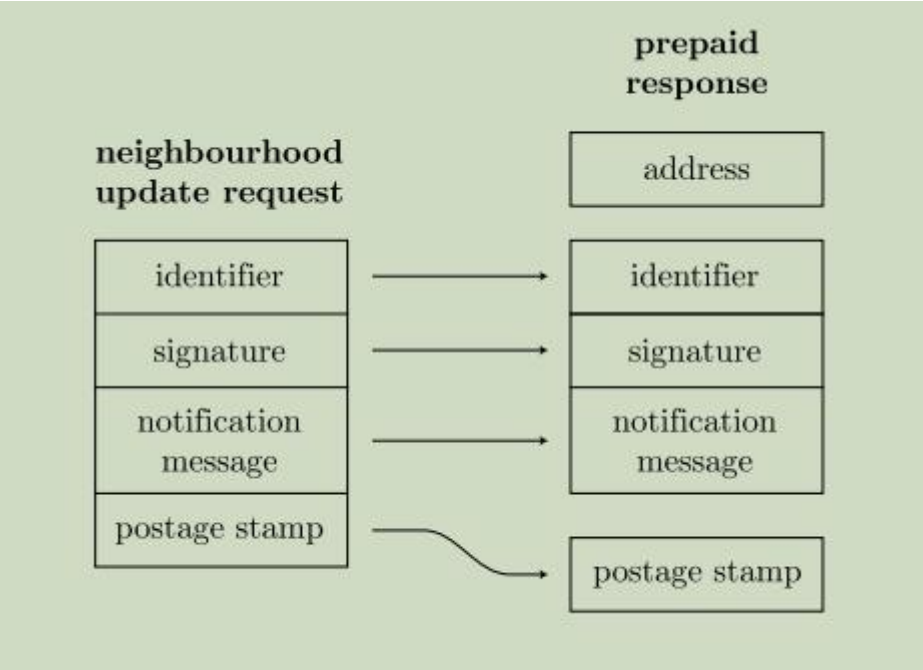

图 4.25：邻域通知请求不仅包括标识符和邮戳，还包括通知消息及其签名，以针对地址进行验证。因此，P 需要构建实际的通知，当发布者 F 将其更新发布到 P 的邻居时，P 可以将通知发布给接收者 R。

如果通知只需要包含信息流更新地址作为其负载，那么与标识符的关联可以由发布者本人签名。此签名与标识符一起应视为信封的必要组成部分，并必须包含在通知请求中。与直接用于发布者的开放信封不同，邻域通知可以视为封闭信封，其中内容由发布者预先批准。

在这种情况下，发布者，而非发布者，成为块的所有者，并且当发布者创建并发布通知时，签名已经可以使用。因此，发布者不需要提供任何公钥或账户地址信息。事实上，发布者的身份不需要是固定的，因为任何节点都可以作为潜在的发布者候选人。发布者可以将通知请求发送到信息流更新块的邻域。最近的邻居将继续持有请求块，直到它们收到适当的信息流更新块，从而确认已收到通知。参见图 4.26 了解使用邻域通知时的事件时间线。

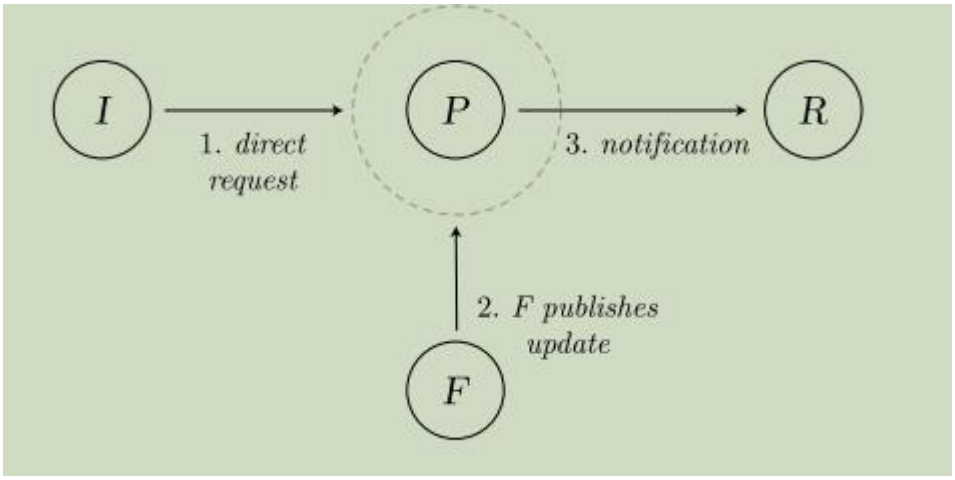

图 4.26：邻域事件通知时间线。发布者 I 为一个单一所有者分块挖掘一个标识符，并将自己作为所有者，使得该分块地址落入接收者 R 的邻域范围内。发布者作为所有者，还必须针对预制的提醒对标识符进行签名，并记住应通知哪个节点。当 P 同步订阅源更新时，通知会通过常规的推送同步机制发送给 R。

但是，如何确保通知不会过早或过晚发送呢？虽然通知块的完整性由发布者保证为单一所有者，但还需要额外的措施来防止管理通知的节点提前发送通知。如果通知不能在更新到达之前发布，将是理想的，否则恶意节点可能会通过处理请求生成误报。

一个简单的措施是使用一个对称加密密钥加密请求中的消息，只有在收到信息流更新时，潜在发布者才能解密该密钥。例如，信息流更新标识符的哈希值可以作为该密钥。为了揭示通知请求需要与信息流更新到达时匹配，主题必须保持未加密，因此这里我们不对 pss 信封进行非对称加密，只对消息进行对称加密。

请注意，如果信息流是公开的，那么信息流更新地址和标识符是可以知道的，因此邻域通知应当用于后续标识符未公开的信息流。

目标块传输

通常，在 Swarm 的 DISC 模型中，块是通过检索请求获取的，这些请求会根据请求的块地址转发到指定的邻域（见 2.3.1）。路由中的第一个拥有该块的节点会响应并将块作为向后传输的响应，沿着相同的路由返回。然而，在某些情况下，可能需要一种机制来请求一个已知存储在特定邻域中的块，并将其发送到另一个已知需要该块的邻域。为此用途而设计的一种机制称为目标块传输：请求是一个特洛伊 pss 消息，而响应（即传输）必须是一个单一所有者块，该块包装了请求的块。该传输块的地址通过挖矿生成，使其落入接收者的邻域。

这些“块内块”响应在结构上类似于邻域通知，因为有效载荷的哈希值已经是已知的（即请求块的内容地址）。请求者可以对其与标识符的关联进行签名，并将签名与标识符一起包含在请求消息中（见图 4.28）。当一个节点接收到此类请求并且存储了请求的块时，它可以构建一个有效的单一所有者块，并将其定向到接收者的邻域作为响应。这样，请求就变得通用，即不与潜在发布者的身份绑定，因此可以发送到任何需要该内容的邻域。甚至可以同时发送多个请求，因为有效响应的唯一性确保了块的完整性（见 2.2.3 和 4.3.3）。目标传输被用于缺失块恢复，见 5.2.3。

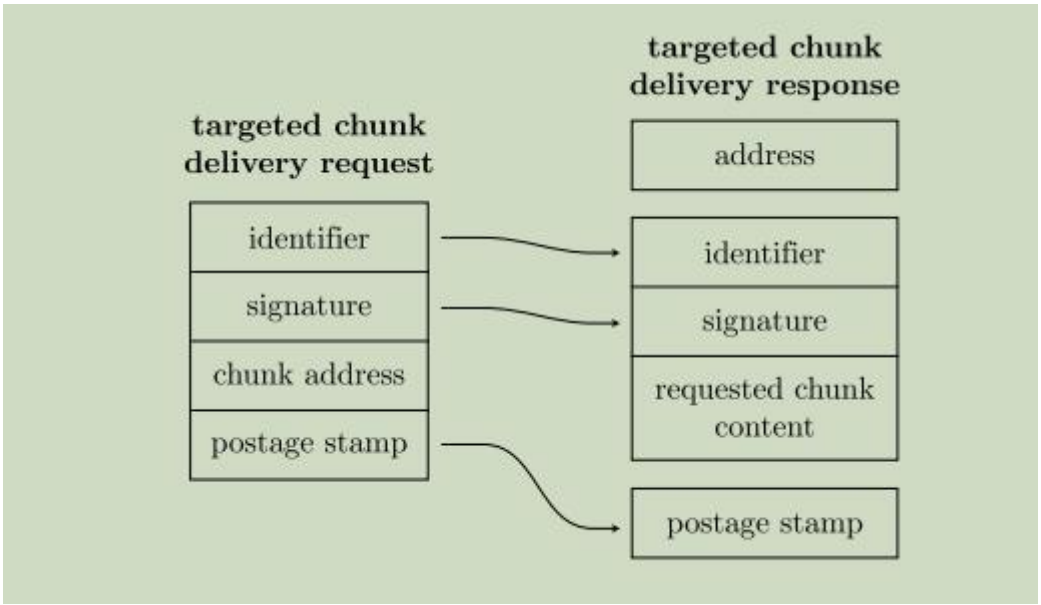

图 4.27：定向分块传递与邻域通知类似，因为它们都是预先寻址、预先支付和预先签名的。同样，标识符是通过挖矿生成的，因此给定发行者 I 作为所有者，它会生成一个落在接收者 R 的邻域中的分块地址。在这里，发行者针对他们希望发布到 R 的内容寻址分块的哈希值对标识符进行签名。如果定向分块传递请求到达任何拥有所请求分块的节点，它们可以使用信封和分块内容生成有效响应，这是一个包裹内容寻址分块的单一所有者分块。

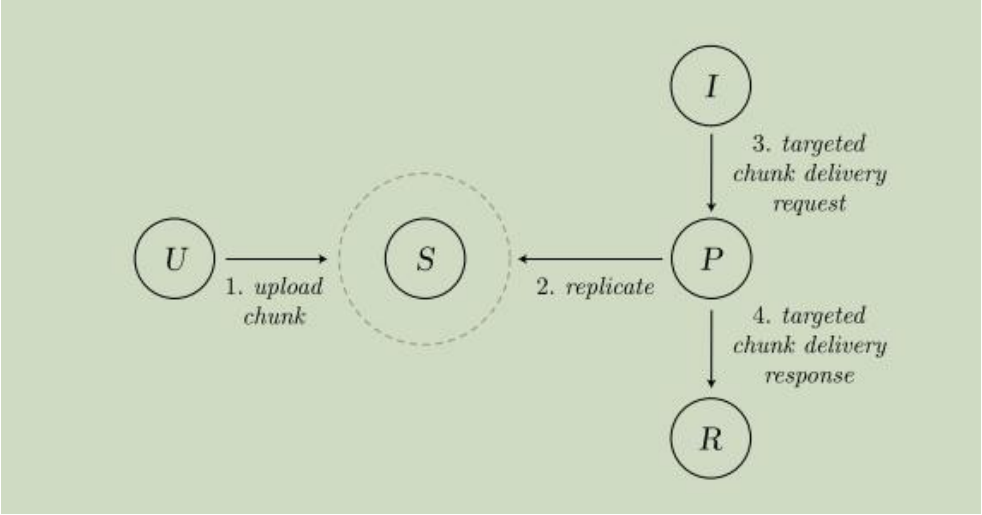

图 4.28：目标分块交付的事件时间线。上传者 U 将分块上传到 Swarm，并由存储节点 S 接收。P 范围内的节点复制并固定该分块。现在，如果发布者 I 希望将该分块交付给 R，他们会将预付费的信封（地址为 R）以未加密的方式发送到已知主机的邻域。这样，任何拥有该分块的人都可以构建通知并发送或存储以供后续使用。

不同之处在于，邻域通知不会提供任何新信息，而目标块传输提供了块数据。此外，包装在单一所有者块中的块传输不涉及消息包装，并且没有主题。在请求和响应中都不使用加密。不使用加密的结果是，如果存在多个目标，初始的特洛伊请求只需匹配其中任何一个目标即可。

表 4.1 总结了三种类似通知的结构的各种属性。

type	owner	poster	request encryption	notification
direct	poster	publisher	asymmetric on pss	feed update content
neighbourhood	issuer	any	symmetric on envelope	feed update arrival
targeted delivery	issuer	any	none	chunk content

表 4.1：关于各类饲料更新通知和目标数据块传递的请求与响应。

第 5 章 持久性

在本章中，我们将重点讨论数据持久性，即确保内容在 Swarm 上始终可用的方式。我们介绍了可以提供跨邻域冗余的错误编码方案，以确保在面对节点波动（churn）时的可用性，尽管这会带来更高的存储开销。特别地，擦除编码（5.1）和纠缠编码提供了针对不同访问模式的文档优化的冗余。此外，我们还介绍了“本地固定”（local pinning）的概念（见 5.2），允许用户在其 Swarm 本地存储中标记特定内容为“粘性”，即该内容将被持久化。我们将讨论如何通过“管理权”这一概念，使这种本地固定有助于实现网络中的全球持久性。我们定义了一种缺失块通知协议，允许内容维护者确保他们发布的内容在某些块被垃圾收集后能够恢复。此恢复过程通过代理检索选定的内容固定器来实现。

在 5.3 节中，我们讨论了不可变块存储如何支持用户控制的内容删除。

5.1 跨邻域冗余：擦除编码和分散副本

首先，在 5.1.1 节中，我们介绍了擦除编码。接着，在 5.1.2 节中，我们详细讲解了它们如何应用于 Swarm 中的文件。在 5.1.3 节中，我们介绍了一种使单个块能够完成擦除编码并实现跨邻域冗余的构造。最后，在 5.1.4 节中，我们探讨了系统化编码，这种编码有助于实现多种擦除编码文件的检索策略，同时保持随机访问能力。

5.1.1 错误纠正编码

错误纠正编码在数据存储和传输的背景下被广泛使用，以确保在系统故障的情况下数据的完整性。错误纠正方案通过在上传或传输之前向数据的表示中添加冗余来重新排列原始数据（编码），使其能够在检索或接收时纠正损坏的数据或恢复丢失的内容（解码）。不同的编码方案通过量化其强度（即在数据损坏和丢失的容忍度方面）来评估，同时考虑其成本（即存储和计算开销）。

在计算机硬件架构的背景下，同步磁盘阵列对于提供数据中心的弹性存储至关重要。特别是在擦除编码中，问题可以被表述为：如何将存储的数据编码成分片并分布在磁盘上，以确保在任何一个磁盘出现故障的情况下，数据仍然可以完全恢复？同样，在 Swarm 的分布式不可变块存储的背景下，问题可以重新表述为：如何将存储的数据编码成块并分布在网络中的邻域，以确保在任何一个块无法检索的情况下，数据仍然可以完全恢复？

Reed-Solomon 编码（RS）（Bloemer 等，1995；Plank 和 Xu，2006；Li 和 Li，2013）是所有错误纠正编码的鼻祖，也是最广泛使用和实现的编码之一。当应用于 m 个固定大小的数据块（即长度为 m 的消息）时，它生成 m + k 个码字（相同大小的块）的编码，其中任何 m 个块足以重建原始数据。反之，k 设置了允许丢失的擦除数量的上限（即不可用块的数量），即它表示了（最大）容忍丢失的能力。k 还表示校验块的数量，量化了在编码过程中增加的存储开销，即它表示存储开销。因此，RS 对存储来说是最优的（因为丢失容忍度不能超过存储开销），但对于本地修复过程有较高的带宽需求。解码器需要检索 m 个块才能恢复一个特定的不可用块。因此，理想情况下，RS 应用于那些需要完整下载的文件，而不适用于仅需要本地修复的用例。

使用 RS 时，通常采用系统化编码，即原始数据作为编码的一部分存在，换句话说，校验块实际上是添加到原始数据中的。

5.1.2 Swarm 哈希树中的擦除编码

Swarm 使用 Swarm 哈希树来表示文件。该结构是一个默克尔树（Merkle 1980），其叶子节点是输入数据流的连续段。这些段被转化为块，并分布在 Swarm 节点中进行存储。连续的块引用（可以是地址或地址和加密密钥的形式）被写入到更高层次的块中。这些所谓的“打包地址块”（PACs）构成了树的中间块。分支因子 b 被选择为使其子节点的引用填满一个完整的块。对于未加密的内容，引用大小为 32 或 64（哈希大小为 32），块大小为 4096 字节时，b 为 128；对于加密内容，b 为 64（见图 5.1）。

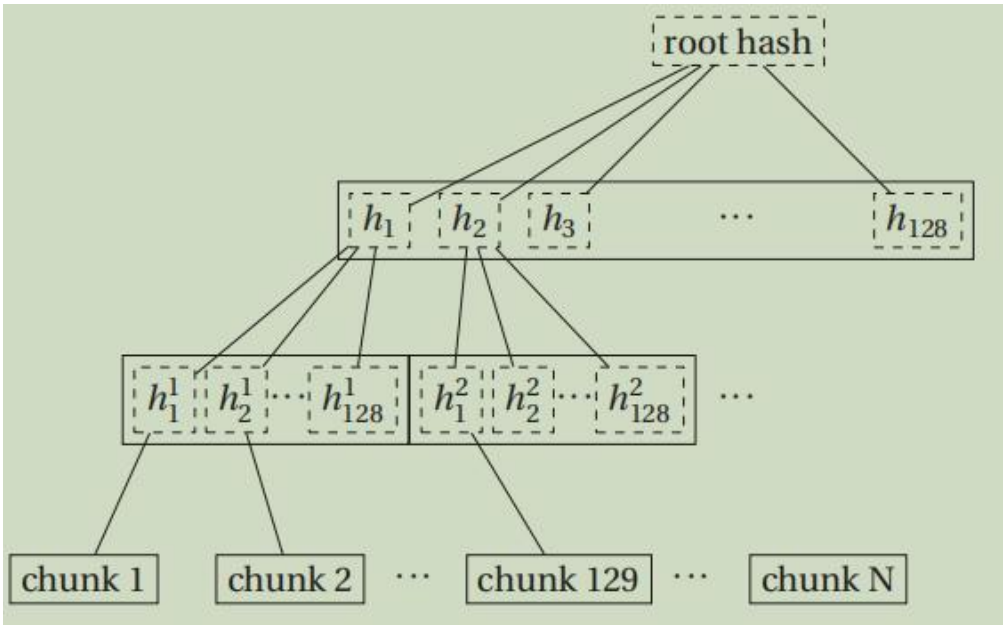

图 5.1：Swarm 树是编码文档如何被分割成块的数据结构。

需要注意的是，在哈希树的右边缘，每一层的最后一个块可能会小于 4K：事实上，除非文件的大小恰好是 4・b^n 千字节，否则总会有至少一个不完整的块。重要的是，包装单个块引用到一个 PAC 中没有意义，因此它被附加到第一个存在空余块的层次。这样的“悬挂”块只会出现在文件的 b 进制表示中有零的位时。

在文件检索过程中，Swarm 客户端从根哈希引用开始，检索相应的块。通过将元数据解释为表示块下包含的数据范围，如果该范围超过最大块大小，客户端就认为该块是一个 PAC。在标准的文件下载中，PAC 中打包的所有引用都会被跟随，即检索被引用的块数据。

PAC 提供了一种自然且优雅的方式，在 Swarm 哈希树中实现一致的冗余。擦除编码实例的输入数据是子块的数据，这些子块的数据对应于被打包在其中的连续引用的大小相等的“桶”。其思想是，不是每一个 b 个打包的引用都代表子节点，而是其中的 m 个引用代表子节点，其余的 k = b − m 个引用编码 RS 校验数据（见图 5.2）。

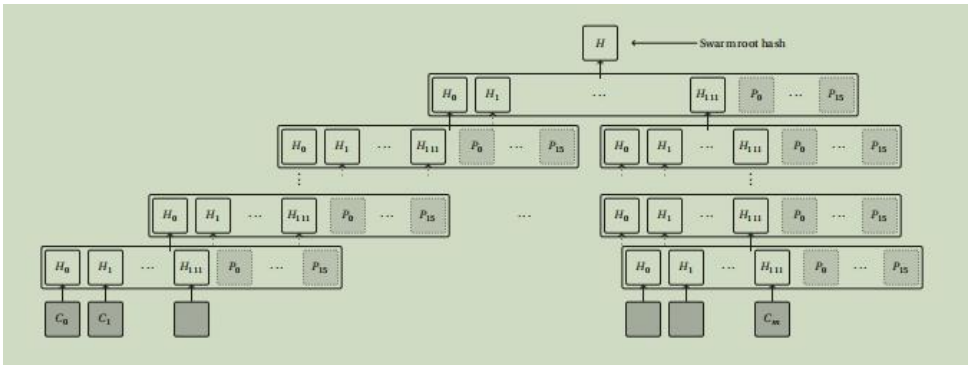

图 5.2：使用 m = 112（n = 128）RS 编码的 Swarm 树，带有额外的奇偶校验分块。分块 P0 到 P15 是每一层中间分块 H0 到 H111 的奇偶校验数据。

包含 PAC 范围的 RS 编码的块算法将如下工作：

将输入设置为实际数据层，并从数据流的连续 4K 段生成块序列。选择 m 和 k，使得 m + k = b 为分支因子（未加密内容为 128，加密内容为 64）。

一次读取一个块。通过递增计数器 i 来计数块。

重复步骤 2，直到 i = m 或没有更多数据可用。

对最后的 i ⩽ m 个块使用 RS 编码方案，生成 k 个校验块，从而得到总共 n = i + k ⩽

b 个块。

将所有这些块的引用连接起来，生成一个打包地址块（大小为 h · n，位于上一层）。如果这是该层的第一个块，则将输入设置为该层，并从步骤 2 开始执行相同的过程。

当输入被消耗完时，向下一层发出输入结束的信号并退出常规操作。如果没有下一层，则将单个块记录为根块，并使用引用来表示整个文件。

这个模式会一直在树中重复。因此，哈希 H_{m+1} 到 H_{127} 指向由 H_0 到 H_m 指向的块的校验数据。由于校验块 P_i 没有子块，树结构没有统一的深度。

5.1.3 不完整的块和分散的副本

如果文件的块数不是 m 的倍数，就无法像处理其他批次一样处理最后一批块。我们提议使用擦除编码对剩余的块进行编码，以确保至少提供与其他块相同级别的安全性。为了过度补偿，即使数据块少于 m，我们仍然需要相同数量的校验块。

这就留下了一个特殊情况：我们无法对单个块使用 m-out-of-n 编码方案（m = 1），因为这将导致 k + 1 个相同块的副本。问题在于，相同块的副本具有相同的哈希值，因此会被自动去重。每当剩下单个块（m = 1）（即根块本身）时，我们需要以某种方式复制该块，确保（1）副本在地址空间中均匀分布，并且（2）其地址可以由检索者得知，检索者理想情况下只知道原始块地址的引用。

我们的解决方案使用了 Swarm 的特殊结构——单一所有者块（SOC；见 2.2.3）。根块的副本通过将块数据作为多个 SOC 的有效载荷来创建。这些 SOC 的地址必须能够根据确定性的约定，从原始根哈希推导出来，该约定为上传者和下载者所共享。

SOC 的地址是其 ID 的哈希值与其所有者的以太坊地址的组合。为了创建有效的 SOC，上传者需要使用所有者的身份对 SOC 进行签名，因此 SOC 的所有者必须是一个共识身份，并且其私钥公开可见。

地址的另一个组成部分，SOC ID，必须满足两个标准：（1）它需要与有效载荷的哈希值匹配至 31 字节；（2）它必须提供足够的熵，以便将整个块挖掘到足够数量的不同邻域中。（1）作为副本 SOC 的特殊情况的验证标准，而（2）确保我们能够在地址空间内均匀地找到副本。这个结构被称为分散副本：

假设 c 是我们需要复制的内容地址块；n 是可用于查找生成 2^k 个完美平衡副本的随机数的熵位数；初始化一个长度为 2^k 的块数组 ρ，开始时设置 n 位整数 i = 0 和副本计数器 C = 0。

1 通过获取 addr(c) 并将最后一个字节（位于索引位置 31）改为 i 来创建 SOC ID。

2 通过连接 ID id 和所有者 o^10 计算 SOC 地址，并使用 Keccak256 基础哈希对结果进行哈希计算 $a_i := H(id \oplus o)$，然后记录 $c_i = SOC(id, o, c)$。

3 通过将哈希值的前 k 位作为大端二进制数 j 计算该哈希属于哪个 bin，$0 \leq j < 2^k$。

4 如果 ρ[j] 未分配，则让 $\rho[j] := c_i$ 并增加 C。

5 如果 C = 2^k，则退出。

6 将 i 增加 1，如果 i = 2^n，则退出。

7 从步骤 1 重复。

通过这个解决方案，我们能够为任何长度的数据提供任意级别的冗余存储。

然后，根据策略，下载者可以选择从哪个地址检索该块。显而易见的选择是离请求节点的覆盖地址最近的副本。

5.1.4 检索的预取策略

在下载过程中，按层次的系统性擦除编码允许采用不同的预取策略：

NONE = 直接下载，无恢复；节俭型

不进行预取，如果存在 RS 校验块，则忽略它们。检索只涉及原始块，不进行恢复。

DATA = 预取数据，但无恢复；廉价型

预取数据块，忽略存在的 RS 校验块，不进行恢复。

PROX = 基于距离的选择；廉价型

对于所有中间块，首先检索预计最快下载的 m 个块（例如，离节点最近的 m 个块）。

RACE = 延迟优化；昂贵型

启动范围内（最大为 m + k）的所有块的请求，只需要等待前 m 个块的交付即可继续。这相当于说可以忽略最慢的 k 个块的检索，因此这种策略在延迟优化的同时，代价较高。

总体来说，使用恢复的策略能够有效克服块的偶尔不可用问题，无论是由于网络竞争、Kademlia 表中的连接间隙、节点更替、邻域价格过高，还是恶意攻击针对特定邻域。

类似地，考虑到典型的块检索网络延迟模型，RACE 模式中的擦除编码可以保证检索延迟的上限。

5.2 数据托管：固定、重新上传与恢复

本节介绍了“固定（pinning）”这一概念，即一种保护本地固定内容免于被存储节点的垃圾回收例程移除的方法（5.2.1）。在 5.2.2 节中，我们讨论了内容固定器如何协同工作，在整个网络范围内固定内容。5.1 节定义了一个恢复协议，下载者可以使用该协议通知存储者缺失的块，这些块属于他们负责在全球范围内固定的内容。通过这种按需修复和所实现的不中断下载体验，恢复实现了一种低成本的持久性度量，能够确保特定块在网络范围内的可用性，而不需要支付保险的财务支出。

5.2.1 本地固定

本地固定是使内容变得“粘性”的机制，它防止内容被垃圾回收。它仅在节点的本地存储中锚定内容，以实现数据的本地持久性和快速检索。固定过程在客户端本地数据库中按单个块的层次操作，并为用户提供固定和取消固定文件及集合的 API（见 6.1.4）。

为了固定构成文件的所有块，客户端需要为每个块维护一个引用计数。每当块被固定时，该计数就会增加；每当块被取消固定时，该计数就会减少。只要引用计数不为零，块就被认为是至少一个固定文档的一部分，因此不会被垃圾回收。只有当引用计数归零——即该块被取消固定且每次之前都曾被固定——时，它才会再次变得有资格被垃圾回收。

本地固定可以被视为一种功能，使得 Swarm 用户能够标记特定的文件和集合为重要的，因此不可删除。将文件固定在本地存储中还会使它即使在没有互联网连接的情况下，也能始终对本地节点可访问。因此，使用固定内容来存储本地应用数据可以启用离线优先的应用范式，Swarm 在重新连接时无缝地管理网络活动。然而，如果一个块不在节点的责任范围内，仅靠本地固定不足以确保该块对其他节点的可用性。这一限制产生的原因是固定者不在该块应存储的 Kademlia 邻域内，因此在使用拉取同步协议请求时，该块无法被找到。为了提供

这一功能，我们必须实现固定协议的第二部分。

5.2.2 全球固定

如果某个块由于存储节点所在邻域的垃圾回收而被移除，网络中其他地方的本地固定节点无法独立地重新检索它。为了让固定机制有助于全球网络持久性，必须解决以下两个挑战：

— 当属于固定内容的块丢失时，全球固定节点需要得到通知，以便它们可以重新上传该块，

— 被垃圾回收但全球固定的块必须保持可检索，以确保用户的下载不中断。

实现这一目标的一种简单方法是定期检查网络中固定的块，并在未找到时重新上传内容。然而，这种方法涉及大量冗余的检索尝试，具有巨大的带宽开销，并且最终无法减少延迟。

另一种替代的、反应式的方法是组织对固定器的通知，当用户尝试访问固定内容但遇到不可用块时触发通知。理想情况下，下载者会向固定器发送一条消息，触发两个动作：（1）重新上传缺失的块，（2）响应下载者的请求，提供该块。

回退到网关

假设一组固定节点已在本地固定了内容，我们的任务是允许回退到这些节点。在最简单的场景下，我们可以将节点设置为网关（可能会负载均衡到多个固定节点）：如果文件或集合的清单条目中包含此网关，用户就能知道这个网关。如果用户由于缺失块无法从 Swarm 下载文件或集合，他们可以简单地转向网关，并在本地找到所有的块。这个解决方案因其简便性而受益，因此很可能成为实现全球持久性的第一个里程碑。

将块挖矿到固定节点的邻域

一种更复杂的解决方案，尽管其复杂度较高，涉及到发布者以一种方式组织文件的块，使它们都位于固定者（或该集合中的任何固定节点）的邻域内。为了做到这一点，发布者需要为文件的每个块找到一个加密密钥，使得加密块的地址至少在前 d 位与固定者的地址匹配，其中 d 的值选择得比该固定者的邻域深度要大得多。

这些解决方案可以实现持久性，但它们也重新引入了一定程度的集中化，并将维护和控制服务器基础设施的责任交给了发布者。此外，它们还存在传统的客户端-服务器架构的缺点，即容错性较差，并且由于请求集中，性能可能会受到影响。这些解决方案也未能解决内容分发到固定节点的问题，并忽略了隐私考虑。因此，尽管 Swarm 为用户提供了低级别的固定 API，但它仍然是一个不太理想的替代方案，远不如激励型文件保险系统，后者被认为是金标准。因此，使用案例应仔细评估，以确保它们不会从激励型系统提供的增强隐私和弹性中受益。

5.2.3 恢复

在接下来的部分中，我们概述了一个简单的协议，用于通知固定者块的丢失。固定者可以通过以下两种方式反应：（1）重新上传丢失的块到网络，同时（2）响应通知者，向他们交付缺失的块。

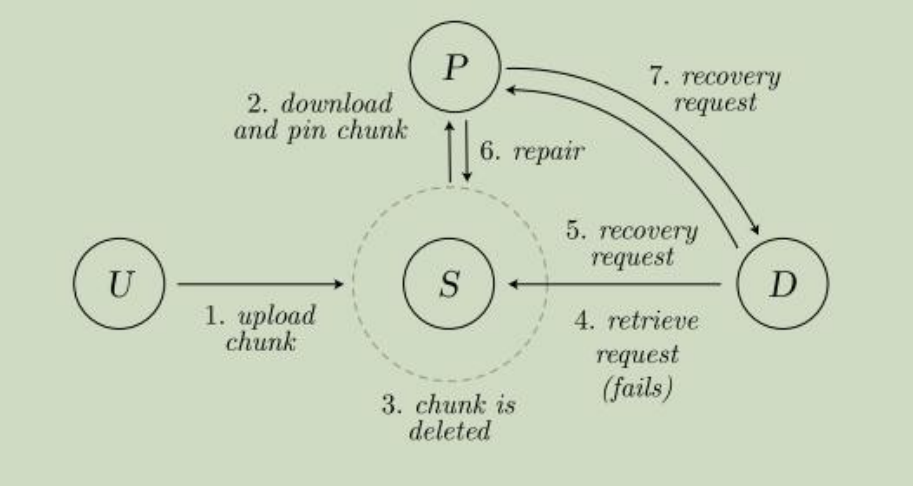

图 5.3：通知缺失分块的过程与定向分块交付的过程类似。在这里，下载者挖掘标识符以匹配自己的地址，即自我通知。如果下载者 D 遇到缺失的分块（请求超时），他们会向可能找到缺失分块的几个邻域之一发送恢复请求。如果恢复请求成功，D 会收到一个包含缺失分块的单一所有者分块形式的响应。该分块也会被重新上传到网络中的适当位置。

当块的副本在愿意的主机（固定者）之间分布时，无法找到特定块的下载者可以通过使用名为 prod 的缺失块通知协议，从这些主机中的一个回退到恢复过程。与往常一样，这个缩写的解析捕捉了协议的特性：

1 协议：删除后恢复协议

请求者和固定者之间的协议，用于在垃圾回收后协调块的恢复。当下载者无法检索块时，该过程由下载者触发。

2 下载者的请求过程

恢复主机持续监听来自下载者的潜在恢复请求。

3 修复 DISC 的服务

Prod 提供修复 Swarm DISC 的服务，以应对数据丢失。

4 数据的固定和重新上传

主机在接收到恢复请求后，会固定数据并将其重新上传到指定区域。

5 按需快速响应

如果请求者要求快速的直接响应，主机会通过恢复响应信封发布缺失的块，并附上所需的邮资印章。

恢复主机

恢复主机是自愿提供固定块以便恢复的固定者。这些固定者明确表示愿意承担此任务，并已下载所有相关块并在其本地实例中将其固定。

为了将其发布的内容宣传为全球固定或可修复的，发布者会收集这些自愿恢复主机的覆盖地址。实际上，仅收集覆盖地址的前缀（长度大于 Swarm 的深度）即可。这些部分地址被称为恢复目标。

发布者将通过一个名为“恢复订阅”的数据流，将其内容的恢复目标广告给数据消费者。消费者可以通过一种约定来追踪这个数据流，构建一个简单的顺序索引方案。

恢复请求

一旦恢复主机的覆盖目标被下载节点揭示，下载节点应该在遇到缺失块时通过发送恢复请求“触发”其中一个恢复主机。此实例的定向块交付（参见 4.4.4）是一个公开的未加密 Trojan 消息，至少包含缺失块的地址（见图 5.4）。

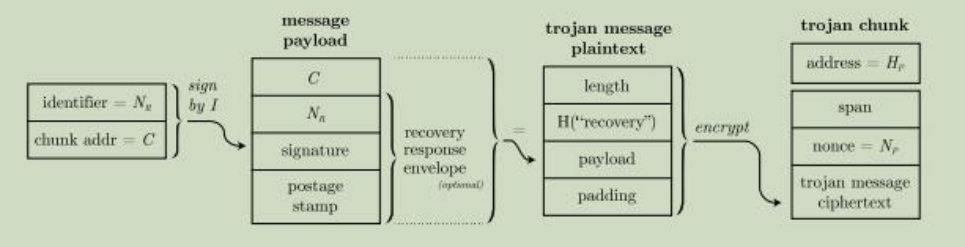

图 5.4：恢复请求是一种用于缺失分块通知的特洛伊分块。它是未加密的，其有效载荷被结构化为一个 pss 消息，包含需要恢复的分块的地址。可选地，恢复请求还包括一个特殊的恢复响应信封，这是一个带有签名的标识符，用于验证标识符与缺失分块哈希之间的关联。

为了创建一个恢复请求，下载节点需要：(1) 创建消息的有效载荷；(2) 找到一个 nonce，当该 nonce 被添加到有效载荷前面时，生成一个与发布者指示的恢复目标匹配的内容地址。匹配的目标表明，推送同步该块将把它发送到与该目标前缀相关联的恢复主机邻域。

如果目标恢复主机在线，它们将接收到恢复请求，提取缺失块的地址，从本地固定存储中检索相应的块，并用新的邮资标签将其重新上传到网络。

恢复响应封装

恢复响应封装用于使恢复主机能够迅速直接地回应恢复请求的发起者，而无需承担额外的成本或计算负担。它是定向块交付响应（参见 4.4.4）的一种形式，属于一个地址封装结构，对发布者保持中立，但对内容是固定的。请求消息包含了潜在发布者创建有效定向块交付响应所需的必要组件：一个带有签名的标识符，签名验证标识符与缺失块哈希之间的关联。标识符的选择使得单一所有者块的地址（即标识符与所有者账户的哈希）落入请求者的邻域，并通过网络的推送同步协议实现自动交付。

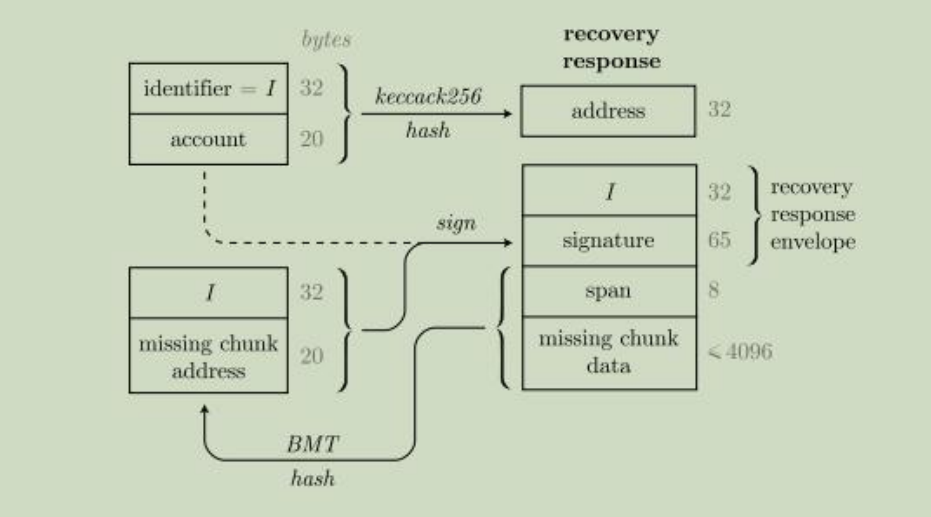

图 5.5：恢复响应是一个包裹了缺失分块的单一所有者分块。任何拥有头部信息和分块数据的人都能够构建一个有效的单一所有者分块，并将其免费发送给下载者。

如果目标恢复主机在线，它们会接收到恢复请求并从恢复响应中提取缺失块地址、标识符和签名。在从本地存储中检索到相应的固定块后，恢复主机可以继续创建恢复响应。为了创建恢复响应，恢复主机将标识符的哈希值与有效载荷哈希值结合，形成签名的明文。恢复请求中接收到的签名使恢复主机能够恢复请求者的公钥。使用该公钥，恢复主机能够计算出请求者的地址，最后，通过将请求者的地址与标识符哈希，得到单一所有者块的地址。将邮资标签附加到请求上并随恢复响应一起发送是有益的。请求者可以支付与返回块相关的 Swap 成本，补偿全球固定节点。这一方案使得下载者能够通过微支付智能地覆盖节点的运

营成本，从而确保整个网络中数据的可用性。

5.3 Dream：删除和不可变内容

数据的数字化程度不断增加以及云服务的广泛使用，前所未有地放大了关于个人信息隐私和控制的担忧。作为回应，监管机构通常会对可删除性提出要求，但并未意识到一旦信息被看到，就几乎不可能将其“让人忘记”。然而，现实是，运营支撑数字出版基础设施的实体控制着提供内容的技术，并且可以通过拒绝访问来实现“删除”。这种集中的把关方式从监管角度来看可能很有吸引力，因为它可以被视为执法工具。尽管这种干预很少能解决不切实际的隐私权问题，但把关者仍提供了一个明确的责任方，并且采取有效的行动：不提供审查过的内容。这虽然给不知情的人带来一种虚假的安全感，却只不过是对审查问题的进一步体现：出版平台拥有筛选内容的技术手段，而集中的控制使得它们能够更轻松地对内容施加影响。最初作为内容策划的合理措施，最终变成了现存的审查制度。尤其是在由于网络效应而获得准垄断地位的社交平台上，这个问题更加严重。由于切换平台的高成本，许多无辜的内容创作者发现自己被“去平台化（deplatformed）”。同样，通过法律手段识别主机并拒绝访问内容的能力，使得掌握这种权力的人能够侵犯言论自由。

在 Web 3 的去中心化范式中，不再有单一的运营实体控制出版平台或托管基础设施。这使得审查的成本变得不可行。然而，个人数据可能永久暴露的潜在风险仍然是大多数人关注的重大问题。因此，需要新的解决方案，来恢复集中的把关者所代表的安全感。

5.3.1 删除和撤销访问权限

首先，请注意，任何关于信息删除的说法，即将其从所有物理存储设备中抹去，既是无法执行的，也是不可行的。即使是最严格的数据保护审计，也不要求从备份带中擦除有问题的数据。一般来说，默契的假设是，被命令删除的信息应该通过典型的先前方法变得不可访问。

接下来，我们制定了一个适用于去中心化存储系统的最强有力的删除定义，并提出了一种实现这一定义的构建方法。重要的是，这种方法纯粹是技术性的，侧重于主要行为者的能力和成本，而不是依赖于那些要求中介方尊重主要行为者权利的程序性措施。

在此背景下，主要行为者是上传者，他们希望通过授予多个方（称为下载者）读取访问权限来共享内容。授予访问权限被定义为提供一个规范的内容引用，允许系统最终检索出计划公开的完整信息。任何有特权访问此信息的方，都能够存储、重新编码，并可能传播这些信息。这提供了多种方式，在任何以后时刻使内容变得可访问，从而绕过任何被视为删除或移除访问权限的过程。根据定义，对于这种不利情况是无法提供保护的。因此，任何合法的删除概念（事后撤销访问）应该被解释为更狭义的：即以比完全存储内容的成本更低的方式重放相同的访问方法的可行性。

现在我们将删除定义为一种上传内容并撤销访问的方案，满足以下要求：

专业化

上传者能够在发布时选择一个专门的构造，允许回溯性地撤销访问权限。

主权

上传者是数据的主权所有者，拥有唯一的手段来撤销之前授予任何方的访问权限。

安全性

在撤销访问后，受权方无法通过相同的引用或任何比已删除内容更短的提示来访问内容。

请注意，定期上传的内容可能会被遗忘：如果没有人支付存储费用且该内容没有频繁访问，构成该内容的区块将会被垃圾回收。然而，带有过期邮资标签的区块不能被视为可靠地删除，以满足主权和安全性要求。

5.3.2 构造

本节的目标是构建一个基于 DISC 模型的可撤销访问模型的正式构造。我们将方案限制为区块，这是 Swarm 的 DISC 模型中的基本固定大小存储单元。提出的“Dream”构造实现了一个可删除内容存储和访问模型，满足专业化、主权和安全性的要求。使用“梦想 (Dream)”一词暗示了这样一个构造在不可变的 DISC 模型中竟然是可能的，这一发现有些出乎意料。除此之外，正如 Swarm 中的惯例，它也作为一个助记首字母缩写，代表着表达访问控制要求的 5 个梦想属性。

D – 可否认

Dream 密钥充当一次性解密密钥。由于多个内容区块（事实上，任何任意内容）都可以使用相同的 Dream 密钥，因此密钥与任何内容的关联是可以否认的。

R – 可撤销

通过 Dream 密钥授予的访问权限是可撤销的。撤销所有方的访问权限，包括上传者自身的访问权限，视为删除。

E – 可过期

该方案允许一次性使用，即密钥只能被检索一次。

A – 可寻址

可以授予一个特定邻域的访问权限，只有在特定覆盖地址范围内运行节点的客户端才能访问该内容。

M – 可变

该构造对网络的波动和规模变化具有韧性；它可以在独立的受权方之间重复使用，并且可升级。

该方案建立在 DISC 的 API 之上，可以完全作为第二层解决方案来实现。尽管具有丰富的功能集，该方案并未使用复杂的加密技术，而是利用了各个组件子系统之间的相互作用。

假设我们有一个伪随机确定性函数，它从一个密钥大小（32 字节）的生成器 g 生成一个更长的（例如，区块大小=4K）序列 c。

核心构造称为 dream，是一个区块大小的一次性密钥，充当可删除内容的解密密钥。dream 区块是通过一组节点按照网络协议共同创建的。参与节点序列中的每个节点接收一段数据，并将其与来自其备用数据的输入合并，以生成输出，然后将其发送到链中下一个节点的邻域。只要链中的每个节点执行相同的计算并转发相同的结果，就可以生成相同的一次性密钥，从而启用可删除内容的检索和读取。

这里的洞察是，检索一个可删除区块的引用涉及使用由上传者控制的一组不可变区块进行计算。上传者改变这组基础区块的能力是提供删除所需变动性的关键。

设 b 为由 U 拥有的邮资批次的批次 ID（一个 32 字节的哈希值），并让 Chunks(γ,b,p) 表示在指定的枢轴 p 的桶中，邮资批次 b 在区块 γ 上戳记的所有区块。定义$\overline{x}$(γ,b,p)为在枢轴 p 指定的桶中，邮资批次 b 戳记的早于 γ 的区块，且其地址与枢轴 p 的 XOR 距离最小的区块。

$$\overline{\mathcal{X}} : \Gamma \times \text{Batches} \times \text{Segment} \longrightarrow \text{Chunks}$$

$$\overline{\chi}(\gamma, b, p) \stackrel{\text{def}}{=} \arg \min_{c \in \text{Chunks}(\gamma, b, p)} \chi(p, \text{ADDRESS}(c))$$

如果批次 b 未充分利用，则由 p 指定的桶中可能不包含属于该批次的任何区块。因此，将没有最接近 p 的区块，这使得$\overline{\chi}$函数在这种情况下未定义。现在，我们定义批次 ID 和输入地址 p 的 OTP 更新函数Δ(γ ,b)如下：

$$\Delta : \Gamma \times \text{Batches} \to \text{Chunks} \to \text{Chunks}$$
$$\Delta(\gamma, b) : \text{Chunks} \to \text{Chunks}$$

$$\Delta(\gamma, b)(c) \stackrel{\text{def}}{=} \mathcal{G}[4K](\text{ADDRESS}(\overline{\chi}(\gamma, b, c)))$$

由于批次的均匀度深度预计大于节点的存储深度，因此预期在由 p 指定的批次桶中，所有区块都应出现在由 p 指定的邻域内的每个节点的备用存储中。

然而，如果 Swarm 扩展并且某些节点的存储深度高于批次的均匀度深度，那么由 p 指定的邻域可能不包含桶中的所有区块，从而影响 OTP 更新函数的可计算性。

尽管如此，如果满足两个条件：1) 由 p 指定的 b 批次桶（bucket）不为空，且 2) 邻域中包含一个或多个桶，那么更新函数可以通过邻域内的任何存储节点进行计算。

由于更新函数可以应用于其自身的输出，我们可以如下定义 dream 路径：

$$\Pi(b, g) \stackrel{\text{def}}{=} c_0, \ldots, c_n \in \text{Chunks}\{n\}$$
$$c_i \stackrel{\text{def}}{=} \begin{cases} \mathcal{G}[4K](g) \text{ if } i = 0 \\ \Delta(b, c_{i-1}) \text{otherwise} \end{cases}$$

Dream 路径函数是一个由有限递归定义的伪随机生成器。dream 是生成器和一次性密钥的特定配对，作为 dream 路径的输入和输出。给定一个长度为 n 的 dream 路径和受权覆盖地址 a，以及具有均匀度深度 d 的批次 b，上传者需要找到生成器 g，使得梦想路径的第 n 个区块落在 a 的邻域内（即，PO(a, ADDRESS(c_{n-1})) ⩾ d）。

$$\text{dream}(b, n, a) \subset \text{Keys} \times \text{Chunks}$$

$$\langle g, k \rangle \in \text{dream}(b, n, a)$$
$$\Leftrightarrow$$
$$k = \Pi(b, g)[n] \wedge$$
$$\text{PO}(a, H_{\text{bmt}}(k)) \geq d$$

上传者能够构造梦想密钥，这也允许他们在给定 g 的情况下计算 k。由于 H(k)是均匀的，PO(a, H(k)) ⩾ d 的概率为 1/2^d。

现在我们可以转向 dream 的定义，它是一种基于网络协议的访问方法，具有可否认、可撤销、可过期、可寻址和可变性特征。为了让下载者计算 dream 密钥，他们必须依赖于网络，其中每个递归步骤由邻域内的节点根据相应输入区块的地址计算。

为了确保正确的终止，必须满足以下条件：

— 由 $p_0,\ldots,p_{n-1}$ 指定的批次 b 的所有桶必须非空；

— 批次的均匀度深度必须高于由 $p_0,...,p_{n-1}$ 指定的邻域中节点的存储深度；

— 每一步的输出，由指定为 pi 的邻域中的节点生成，必须发送到指定为 pi+1 的邻域；

— 沿着每个梦想路径跳跃的邻域中的节点（由 $p_0,...,p_{n-1}$ 指定）必须有激励计算 OTP 更新，并将生成的输出区块转发到下一个邻域。

我们现在定义一个名为梦想（dream）的网络协议。协议中的参与者监听包装输入的单一所有者区块。我们假设他们提取批次标识符 b 和有效负载 CAC c，作为 OTP 更新函数的参数。它们计算输出密钥，将其包装为梦想区块，并将其发送到网络，朝着输出区块的地址发送。这使得目标邻域能够计算下一步。如果协议遵循了 n 个步骤，那么目标节点将在地址 a 处接收到 k = c(n)。

接下来我们转向梦想区块的构造。上传者 U 希望授予下载者 D（在覆盖地址 a 处）对内容区块 C 的可撤销访问。U 选择一个自己拥有的邮资批次 b，确保该批次没有完全填满。此外，U 选择一个步骤计数 n 和深度 d。U 创建一个梦想生成器⟨g ,k⟩ ∈ dream(b,d,n,a)，然后计算“密文”C′ = C ⊻k 并将其上传到 Swarm，获得 r = H(C′)作为校验引用。现在，U 创建梦想引用为 ref(C) = ⟨r,b,g⟩，该引用可以交给下载者 D。

5.3.3 正确性、安全性和隐私

检索 D 在拥有 U 计算的梦想区块引用⟨r,b,g⟩（作为 ref(C)）的情况下，构建 p = G[4K](g)，将其包装为梦想区块，并发送到 Swarm。当 D 接收到 b 的梦想区块时，它提取有效负载 k。

D 通过引用 r 检索校验区块 C′ ，然后解码 C = C′ ⊻k。检索过程是显然正确的，只要满足以下条件：

1 校验区块 C′ 可以通过正常的检索方法检索；

2 协作节点遵循梦想协议；

3 梦想路径上的邻域的批次桶底部保持不变。

删除

我们现在转向访问撤销，它等同于通过撤销所有访问来删除内容。

上传者 U 曾通过梦想引用⟨r,b,g⟩授予 D（在地址 a 处）对 C 的访问。U 通过向批次 b 上传额外的区块，从而改变$\overline{x}$，撤销 D’ 的访问。

只要协议沿途经过至少一个诚实的邻域，并且从批次 b 中最接近枢纽 p 的区块发生了变化，下载者 D 将不再能够检索内容 C。

为了证明这一点，首先假设在梦想路径上存在由某个 pi 指定的桶，其中该桶的$\overline{x}$已经改变，这是自上传者构建梦想以来的变化。当由 p_i 指定的邻域中的参与节点计算 OTP 更新函数时，结果将完全不同，梦想链条将无法终止。

隐私 该引用不会泄露任何关于步骤计数或涉及的邻域的信息。协议中的参与者也不知道自己在链中的位置，或关于其他邻域的任何细节，除了他们正在推送同步其区块的紧接着的邻域。

梦想引用的构建是可否认的。除了我们敏感的内容 C，我们还可以考虑 A，任何无争议

的内容区块。在生成 C′ 时，所有者也生成 A′ = A ⊻k 并上传。当被问及 k 时，生成 A 使得否认其他内容（包括 C）变得更为可信。

攻击的解决 一个强大的对手可能会渗透到 Swarm 的每个邻域，并归档所有曾经上传过的信息（即使无法解密）。他们也可能保持上传顺序的日志，这将允许他们按需提供已删除的内容。然而，这种对所有 Swarm 内容的无差别归档付出了巨大的代价，这是可靠击败梦想构建的唯一途径。

现在我们来讨论如何将步骤计数与安全模型进行校准。假设网络中所有邻域的渗透率为 1/2，意味着网络中一半的邻域被假设为恶意并且串通在一起。

当访问被撤销时，所有者在梦想路径上的每个邻域上传新区块。然而，给定一个特定的梦想链，如果梦想路径上的任何一个邻域是诚实的，他们将尊重新到达的区块，并偏离梦想路径，防止它与受赠人终止。

因此，要发生访问泄露，所有邻域必须是恶意的。在我们的安全模型中，邻域是恶意的，具有均匀和独立的概率。对于一个总体渗透率为 1/k 的情况，给定一个随机梦想路径，所有邻域都恶意的概率是 k^−n。对于成功率为 σ 个“九”的安全要求，其中错误率小于 10^− σ，我们可以将该要求表述为：

$$\frac{1}{k^n} < \frac{1}{10^{-\sigma}}$$

现在，将 k 表示为 10^κ，取两边的对数并乘以-1，我们得到：

$$n > \frac{\sigma}{\kappa}$$

当每 10 个邻域中有一个是恶意的时，dream 路径的跳跃数必须与成功率中表达的“九”的数量相同。

第 6 章 开发者接口

本章从用户的角度介绍了前几章中提到的 Swarm 特性。在 6.1 节中，我们讨论了配置上传的开发者接口，包括邮资、固定、擦除编码等方面，以及使用上传标签跟踪块传播进度。接着，我们在 6.2 节中介绍了存储 API，特别是上传集合。最后，在 6.3 节中，我们概述了可用的通信选项。

6.1 配置和跟踪上传

上传接口是 Swarm 功能的基石，使用户能够轻松地将数据传输到网络中。6.1.1 节介绍了用于配置上传 API 的请求头（或查询参数）。在 6.1.2 节中，我们介绍了上传标签，它允许通过进度条跟踪上传过程，并估算完成过程所需的时间。此外，标签还记录了部分根哈希，使中断的上传能够恢复。在 6.1.3 节中，我们描述了与上传付款和在网络中分散数据相关的场景，包括用户如何购买并为块附加邮资。最后，6.1.4 节介绍了可选参数，如加密、固定和擦除编码。

6.1.1 上传选项

本地 HTTP 代理提供了 bzz URL 方案作为存储 API。请求可以指定 Swarm 特定的选项，例如：

tag

使用此上传标签，如果未指定，则在响应头中生成并返回该标签。

stamp

使用此邮资订阅进行上传，如果未给出，则使用最近使用的订阅。

encryption

如果设置，则对内容进行加密；如果设置为 64 字节的十六进制值，表示一个 256 位整数，则使用该值，而不是随机生成的密钥。

pin

固定块；如果设置，则固定上传的所有块。

parities

应用 RS 擦除编码；对所有中间块的每个子批次使用此数量的校验块。

这些选项可以作为 URL 查询参数使用，也可以作为头部进行指定。头部的名称是通过将参数名称前缀为 SWARM- 并大写书写来获得的，例如，SWARM-PARITIES。

6.1.2 上传标签和进度条

在上传文件或集合时，用户知道上传何时完成是很有用的，这意味着所有新创建的块已经同步到网络，并到达可以检索的邻域。此时，上传者可以“消失”，即可以退出客户端。发布者可以传播根哈希，并确信该文件或集合可以从 Swarm 上的每个节点检索到。

为了跟踪上传的进度，我们利用推送同步协议中的每个块的保管收据声明。通过收集并计数这些收据，我们可以追踪已发送块与返回收据的比例，这些数据用于进度条的显示。上传标签是一个表示上传并通过计数已达到特定状态的块来跟踪上传进度的对象。状态包括：

split

块的拆分数量；计数块实例。

stored

本地存储的块数量；计数块实例。

seen

之前存储的块数量（重复块）。

sent

使用推送同步发送的不同块数量。

synced

收到保管收据声明的不同块数量。

借助这些计数，我们可以监控 1) 块拆分 2) 存储 3) 推送同步 和 4) 收据的进度。如果上传标签没有在头部指定，系统会生成一个随机标签，并在文件完全拆分后作为响应头返回。为了在上传过程中监控块拆分和存储的进度，标签需要提前创建，并在请求头中提供。这样，标签可以在上传内容处理的同时并行查询。

已知与未知的文件大小

如果已知文件大小，则可以计算出将生成的块的总数。这使得从一开始就能够根据总块数进行有意义的进度跟踪，包括块拆分和存储的进度。

然而，如果在上传之前不知道文件大小和拆分后的总块数，则块拆分的进度是未定义的。待拆分器完成文件拆分后，总计数可以设置为拆分后的块数。从那时起，就可以提供剩余计数的百分比进度和预计到达时间（ETA）。

请注意，如果上传还包括清单，则总计数仅在设置为拆分后的块数之前作为估算值。随着上传过程中文件大小的增加，这个估算值将趋近于正确值。

重复块

重复块是指在一次上传或跨不同上传中出现多次的块。为了具有本地可验证的定义，我们定义一个块为重复块（或已见块）当且仅当它已经存在于本地存储中。由于块通过上传进入本地存储，并通过推送同步，因此无需再次对已见块进行推送同步。

换句话说，只有新存储的块在评估上传同步预计时间时需要计数。如果我们想要根据已发送和已同步的计数来跟踪进度，它们必须反映已存储的不同块在整体进度中的完成度。

标签 API

HTTP 服务器的 bzz URL 方案提供了标签 API 的标签端点。该端点支持创建、列出和查看单个标签。最重要的是，它提供了一个选项，可以通过 HTTP 流实时跟踪标签的变化。

6.1.3 邮资

为了对上传施加成本并高效地分配网络中的存储资源，所有上传都必须支付费用。在 Web 环境中，这一概念可能不太常见，因此需要开发一种新的用户体验。最接近且最易理解的类比是订阅模式。

邮资订阅

用户可以为特定的时长和存储容量创建一个订阅（例如，1 个月 100MB），并根据客户端软件提供的价格进行相应支付（类似于区块链中交易费用的确定方式）。价格估算可以通过邮资彩票合约获得。订阅是有名称的，但这些名称仅在本地有意义。API 会为选择的存储周期提供一些默认选项，采用对数刻度，例如：

minimal（几小时）

适用于即时发送 pss 消息或属于临时聊天或其他临时文件的单一所有者块。

temporary（一周）

不打算长期存储的文件或邮箱消息，更多是供第三方异步获取。

long term（一年）

长期存储的默认选项。

forever（10 年）

重要内容，不应丢失/遗忘；旨在跨越网络增长及后续批次深度的增加，即使上传者完全离线，内容依然存续。

在上传文件时，用户可以通过指定邮资上传参数的值来指示希望使用的订阅。如果未指定订阅，则使用最近的一个作为默认订阅。如果已知上传文件的大小，当文件超过可用邮资容量时，用户会收到警告。

邮资重用策略

如 3.3 节所述，可以使用加密技术将块挖掘到邮资批次的碰撞槽中。这种将块挖掘到批次中的策略适用于只希望在特定时段上传某个特定文件/集合，或希望将文件/集合的存储周期独立于其他上传进行调整的用户。

另一种选择是预支付一大批邮资批次，并在每个时段保持一定数量的批次处于开放状态。这确保了每个块在某个批次中总有一个可用的碰撞槽。邮资批次按购买时间排序，在为块附加邮资时，首先使用第一个有空闲槽的批次。这一盈利策略最有效地被包括保险公司在内的重度用户使用，他们有能力将流动资金锁定更长的时间。在创建订阅时，必须指定所偏好的策略。

邮资订阅 API

为了管理订阅，bzz URL 协议提供了邮资 API 的 stamp 端点。此 API 允许用户执行各种操作，例如创建邮资订阅、列出订阅、查看详细信息、充值或取消及过期订阅。

在检查其订阅时，用户会被告知该订阅已经上传的数据量以及在当前价格下能存储多长时间（例如，88/100 兆字节可以存储 23 天）。如果估算的存储期限较短，意味着使用该订阅的文件可能面临被垃圾回收的风险，系统会鼓励用户充值订阅以确保内容的持续保护。

6.1.4 其他上传功能

上传时，可以通过使用相关的请求头来指定额外的功能，如交换费用、加密、固定和擦除编码。

交换费用

上传内容会产生交换费用，这与 push-sync 协议相关，并在 SWAP 记账系统中反映 (3.2.1)。当对等连接的余额超过有效支付阈值时，会发出支票并发送给对等方。如果支票合同不存在或没有足够的资金来覆盖未结清的债务，则对等连接会被阻止。这可能导致 Kademlia 未饱和，在此期间，节点被视为轻节点。尽管下载过程仍可继续，但由于某些块可能需要发送到距离请求地址不比节点本身更近的对等方，获取块的平均成本可能会更高。

为了向用户提供透明度，可以通过 SWAP API 访问一些信息，如平均对等数、平均交换余额，以及由于资金不足导致无法连接的对等数。

加密

通过将加密选项设置为非零值，可以启用加密上传。如果提供的值是一个 32 字节种子的十六进制表示，则该种子用于加密。由于加密和解密由 Swarm 本身处理，因此仅应在 HTTP 客户端和 Swarm 代理服务器之间的传输可以确保私密的情况下使用。因此，不应在公共网关上使用，而应通过本地 Swarm 节点或配置了良好 TLS 和自签名证书的私有环境来确保必要的隐私和安全。

固定

上传内容时，用户可以通过将固定上传选项设置为非零值来指定是否希望将内容在本地固定。除此之外，本地 HTTP 服务器的 bzz URL 协议提供了固定 API 的 pin 端点，允许用户管理固定的内容。通过对 pinning 端点发起 GET 请求，用户可以检索固定的文件或连接的列

表。该 API 还接受对哈希（或解析为哈希的域名）发起的 PUT 和 DELETE 请求，分别用于固定和取消固定内容。

当文件或集合被固定时，它应该被检索并存储。固定会触发哈希树的遍历，并增加每个块的引用计数。如果发现某个块在本地丢失，将会被检索。固定后，根哈希将被保存在本地以便将来访问。取消固定会触发哈希树的遍历并减少每个块的引用计数。如果某个块在本地丢失，它将被忽略，并在响应中附加警告。取消固定后，该哈希将从固定哈希列表中移除。

纠删编码

纠删编码（见 5.1）可以在上传过程中启用，通过设置“parities”上传选项来指定每个中间块的子块中应包含的纠删块数量。值得注意的是，使用纠删编码的文件无法通过默认的 Swarm 下载器进行检索。因此，在使用纠删编码时，重要的是通过在封装的清单条目中设置 rs 属性为纠删块的数量来指明纠删编码设置。

6.2 存储

本节将介绍通过 Swarm 的本地 HTTP 代理提供的 bzz URL 方案系列的存储 API。

6.2.1 上传文件

bzz 方案允许通过文件 API 直接上传文件。文件应编码在请求体中或以 multipart 形式提供。所有在 6.1.1 中介绍的查询参数 (或相应的头部) 都可以与此方案一起使用。POST 请求将文件进行分块并上传。创建一个清单条目，该条目包括对上传内容的引用，并反映上传的配置，例如通过 rs 属性指定的每个批次的纠删块数量。清单条目作为响应提供。上传标签允许我们监控上传的状态，它将包含上传文件的引用以及清单条目。

向现有文件追加数据

PUT 请求用于向 Swarm 中的现有数据追加数据，且需要指定 URL 指向特定文件。如果直接引用该文件，设置（如纠删编码的纠删块数量）将从上传头部获取。否则，如果文件未被直接引用，则将使用从封装的清单条目中获得的设置。响应格式与 POST 请求相同。

恢复未完成的上传

作为一种特殊情况，当发生崩溃或用户主动中止上传后，使用追加操作来恢复上传。为便于此操作，有必要通过定期记录上传标签上的根哈希来跟踪部分上传。当请求头中指定的上传标签不完整时，假设该请求意图恢复相同的上传。标签中最后记录的根哈希作为追加目标：现有文件的右边缘将被检索，以初始化分块器的状态。与请求一起发送的文件将从部分上传结束的地方开始读取，偏移量设置为标签中记录的根块的跨度。

6.2.2 集合与清单

如第 4.1.2 节所述，清单可以表示一个通用的索引，将字符串路径映射到条目。这意味着，清单可以作为虚拟网站的路由表、文件集合的目录树，甚至作为键值存储。

清单条目与单例清单

清单条目对于集合非常重要，同样也对于单个文件至关重要，因为它们包含关于文件的关键信息，包括内容类型和纠删编码的详细信息，这对于正确检索文件是必需的。一个为位于空

路径下的文件创建的清单条目称为单例清单。在这种情况下，清单除了条目本身外，不包含其他任何信息。

为便于存储集合，HTTP 服务器提供了 bzz URL 方案，它实现了集合存储 API。当通过使用 bzz 方案的 POST 请求上传单个文件时，会创建并存储一个清单条目，响应中包含对该条目的引用。

上传和更新集合

bzz URL 方案提供了一个支持上传和更新集合的 API。当使用 POST 和 PUT 请求，并且请求的 multipart 载荷中包含 tar 流时，tar 流中编码的目录树结构将被转换为清单。同时，集合中的文件将被分块并上传，并在相应的清单条目中包含它们各自的 Swarm 引用。POST 请求用于上传生成的清单，响应中包括与其相关的 Swarm 引用。PUT 请求要求请求的 URL 引用一个现有清单，该清单将使用 tar 流中的内容进行更新。更新过程包括将新路径与现有清单中的路径合并。如果路径相同，上传的条目将替换现有清单中相应的条目。

该 API 支持向清单中插入路径、更新清单中的路径（PUT），以及从清单中删除路径（DELETE）。对于 PUT 请求，期望在请求体中包含一个文件，上传该文件并将其清单条目插入到 URL 中指定的路径。如果该路径已存在于清单中，则现有条目将被上传生成的条目替换，从而完成更新。如果该路径是新的，上传操作将把一个新条目插入到清单中。

集合 API 支持与文件上传端点相同的头部，包括配置邮资订阅、标签、加密、纠删编码和固定等功能。

直接更新清单

直接操作清单也是受支持的：bzz URL 方案的清单端点支持 PUT 和 DELETE 方法，其行为类似于集合端点。然而，与集合端点不同的是，清单端点不处理清单条目中引用的文件。当使用清单端点时，URL 路径应引用一个清单以及路径 p。对于 PUT 请求，请求体应包含一个清单条目，该条目将放置在路径 p 上。需要的清单块将被创建并存储，并且新的清单的根哈希将在响应中返回。DELETE 方法期望空的请求体，并将路径上的条目从清单中删除：即，它创建一个新的清单，URL 中引用的路径不再存在。

在清单 API 端点上直接发出的 POST 请求会安装一个清单条目。实际上，在文件 POST 的输出上调用清单 POST，相当于在通用存储端点上发出 POST 请求。

给定清单引用 a 和 b，向清单 /merge/<a>/<b> 发出 POST 请求会将 b 合并到 a 上（在发生冲突时优先选择 b），并创建并存储构成合并清单的所有块，合并后的清单的根引用将作为响应返回。

如果 URL 路径引用一个清单，可以在请求体中包含另一个清单，该清单将被合并到引用的清单中。如果发生冲突，上传的清单将优先胜出。

6.2.3 访问控制

如 4.2 所述，访问控制是通过 bzz URL 方案来实现的，该方案提供了访问 API 端点。这个 API 为用户提供了一种便捷的方式，可以对文件、集合或站点应用访问控制，并管理受赠人。

如果 URL 路径引用了集合清单，那么通过 POST 请求发送包含访问控制（AC）设置的 JSON 编码请求体，将加密该清单引用，并将其与提交的 AC 设置一起包装成所谓的根访问清单。该清单随后会被上传，未加密的引用将作为响应体返回。

如果 URL 路径引用了根访问清单，并且访问控制设置指定了 ACT，那么可以使用 POST、PUT 和 DELETE 请求来创建或更新 ACT。所有请求都期望请求体中包含一个受赠人公钥或

URL 的 JSON 数组。如果受赠人是通过 URL 引用的，则通过 ENS 使用解析器提取所有者的公钥。

POST 请求将创建一个包含受赠人列表的 ACT。PUT 请求会通过合并请求体中指定的受赠人来更新现有的 ACT。DELETE 请求将移除请求体中列出的所有受赠人。新的 ACT 根哈希会被更新到根访问清单中，并上传其 Swarm 地址作为响应返回。

6.2.4 下载

下载功能由 bzz URL 方案支持。该 URL 方案假定 URL 的域名部分引用的是清单作为入口点。

值得注意的是，下载过程中涉及的步骤是相同的，即使文件仅被部分检索。如 4.1.1 所示，最低级别支持对文件的任意偏移进行随机访问。因此，指向文件的 URL 的 GET 请求可以在头部包含范围查询。这些范围查询将触发只检索覆盖所需范围的文件块。

检索成本

在 Swarm 中下载文件会产生 SWAP 账户的成本（如 3.2.1 节所述）。这些成本与从网络检索文件块相关。当一个节点的余额超出同行连接的有效支付阈值时，会发出支票并发送给该同行。然而，如果支票合同不存在或没有足够的资金来支付未结清的债务，则该同行连接会被阻塞。这可能会导致 Kademlia 未饱和，在此期间，节点将被视为轻节点。下载仍然可以继续，可能会使用较少的同行，但由于一些文件块需要发送到比节点本身更远的同行，平均检索成本将会更高。

为了向用户提供透明度，展示平均检索请求数量、平均 SWAP 余额以及由于资金不足而不可用的同行连接数量是很有价值的。这些信息可以通过 SWAP API 访问，该 API 可在 bzz URL 方案的 swap 端点中找到。

域名解析

URL 的域名部分可以是一个人类可读的域名或子域名，带有顶级域名（TLD）扩展名。根据 TLD 的不同，可以调用不同的名称解析器。TLD 为 eth 的域名与以太坊主链上的以太坊名称服务（ENS）合约相关联。如果您将 Swarm 哈希注册到 ENS 域名，Swarm 将通过调用 ENS 合约的方式来解析该域名，就像名称服务器一样。

访问控制的身份验证

如果解析后的域名引用了根访问清单，那么检索到的 URL 将受到访问控制的限制（见 4.2）。根据使用的凭据，用户可能会被要求输入密码或一对密钥（密钥 1 用于查找，密钥 2 用于解密访问密钥）。Diffie-Hellman 共享密钥会与 0x00 一起哈希以导出查找密钥，并与 0x01 一起哈希以导出访问密钥解密密钥。根访问清单中 ACT 清单根块的 Swarm 地址从根访问清单中获取。然后，查找密钥被附加到 ACT 地址，生成用于检索清单条目的 URL。此条目中的引用随后使用访问密钥解密密钥进行解密，得到的访问密钥用于解密存储在根访问清单中的原始加密引用。

接下来，检索与 URL 路径对应的清单条目。检索过程考虑以下属性：

1 rs

包含用于检索的 RS 擦除编码所需属性的结构，例如所需的校验块数量。

2 sw3

包含用于诉讼的属性的结构，包括挑战保险公司关于缺失块的情况。

如果提供了 rs 属性，用户可以设置冗余解码的预取策略（见 5.1.4）。

默认策略选择是竞态策略，如果用户希望优先节省下载，则可以通过设置头部

SWARM-RS-STRATEGY=fallback 来强制使用回退策略。或者，可以通过设置 SWARM-RS-STRATEGY=disabled 来完全禁用 RS 策略。擦除编码批次的校验块数量取自封闭清单中的 rs 属性。但是，可以通过设置头部 SWARM-RS-PARITIES 来覆盖此值。

缺失的块

如果请求的块被发现缺失，可以回退到缺失块通知协议（见 5.2）。表示恢复目标集合的数据结构根的引用可以在恢复馈送的最新更新中找到。

当块请求超时时，客户端可以使用一组固定主机（pinner hosts）启动恢复消息的创建。一旦恢复消息创建完成，它会发送到主机。如果此请求也超时，则尝试使用不同的固定主机节点进行下一个恢复块的请求。这个过程会重复，直到恢复块成功检索到，或者所有固定主机和保险人都已耗尽。

6.3 通信

令人惊讶的是，Swarm 的网络层可以作为一个高效的通信平台，具有强大的隐私功能。本节旨在展示如何利用一小组基本原语作为构建全面通信基础设施的基本构件。这些功能涵盖了所有通信方式，包括实时匿名聊天、从先前未连接且可能是匿名的发送者发送和接收消息、异步投递的邮箱功能、长期通知以及发布/订阅接口。

Swarm 核心提供了与通信相关功能的最低级别入口点： — pss 模块提供了发送和接收特洛伊木马块（Trojan chunks）的 API — bzz 模块提供了一种上传单一所有者块的方式。检索单一所有者块仅需要存储 API 提供的功能，而无需任何额外要求

由于特洛伊木马消息处理与存储操作有显著的不同，Swarm 引入了自己独特的 URL 方案，称为 pss。pss 方案提供了专门用于发送消息的 API。当发送 POST 请求时，URL 被解释为引用 X3DH 预密钥捆绑更新块。该块包含必要的公钥用于加密，以及目标地址。目标地址之一应与特洛伊木马消息的地址相匹配（见 4.4.1）。

目标地址作为目标前缀的 buzz 序列化表示，存储为文件。URL 路径指向此文件，或者是顶级域名。主题作为查询参数指定，消息作为请求体发送。

接收消息仅通过内部注册主题处理程序来支持。在 API 的上下文中，这通常涉及通过 WebSocket 进行推送通知。