软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(3):1327−1354 [doi: 10.13328/j.cnki.jos.007174] [CSTR: 32375.14.jos.007174]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

ChannelLink: 基于链下状态通道的跨片状态迁移协议*

贾林鹏[1], 孙 毅[1,2]

[1](中国科学院 计算技术研究所, 北京 100190)

[2](中国科学院大学 计算机科学与技术学院, 北京 100049)

通信作者: 孙毅, E-mail: sunyi@ict.ac.cn

摘 要: 跨片状态迁移协议是保证跨片交易处理原子性的基础, 其效率高低将直接影响分片系统性能. 现有协议处理过程可以分为源分片状态迁出、片间状态传输和目的分片状态迁入这 3 个阶段, 各阶段依次执行、紧密绑定. 利用链下状态通道灵活度高、即时确认的特点, 提出了 ChannelLink 跨片状态迁移协议, 将现有协议中紧密耦合的三阶段处理过程解耦, 有效降低了跨片交易平均开销, 提升了状态迁移效率. 基于此, 设计了一种低开销链下通道路由算法. 该算法基于状态迁移交易与链下通道拓扑等特征, 通过改进遗传算法, 求解最优状态路由方案, 兼顾迁移效率的同时, 降低了用户跨片状态迁移开销. 最后, 实现了 ChannelLink 协议原型系统, 并基于比特币交易以及闪电网络状态信息构造数据集进行实验验证. 实验结果表明, 该协议在 16 个分片、跨片交易比例为 5.21% 的场景下, 分片系统吞吐量提升 7.04%, 交易确认延迟降低 52.51%, 跨片状态迁移开销下降 45.44% 以上, 并且随着分片数量与跨片交易比例的上升, 该协议的性能优势逐步扩大.

关键词: 区块链; 分片; 状态迁移; 状态通道; 路由算法

中图法分类号: TP393

中文引用格式: 贾林鹏, 孙毅. ChannelLink: 基于链下状态通道的跨片状态迁移协议. 软件学报, 2025, 36(3): 1327–1354. http://www.jos.org.cn/1000-9825/7174.htm

英文引用格式: Jia LP, Sun Y. ChannelLink: Cross-shard State Transition Protocol Based on Off-chain State Channel. Ruan Jian Xue Bao/Journal of Software, 2025, 36(3): 1327–1354 (in Chinese). http://www.jos.org.cn/1000-9825/7174.htm

ChannelLink: Cross-shard State Transition Protocol Based on Off-chain State Channel

JIA Lin-Peng[1], SUN Yi[1,2]

[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

[2](School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: The cross-shard state transition protocol is the basis for ensuring the atomicity of cross-shard transactions, and its efficiency directly affects the performance of the sharding system. The cross-transaction process of the existing protocols can be divided into three phases: source-shard state move-out, cross-shard state transition, and destination-shard state move-in. These phases are executed sequentially, and all phases are tightly coupled. This paper proposes the ChannelLink cross-shard state transition protocol based on the off-chain state channel. Since the off-chain channels are highly flexible and can be confirmed instantly, the ChannelLink protocol can effectively decouple the tightly coupled three-phase process, reducing the average cost of cross-shard transactions, and improving state transition efficiency. On this basis, this paper designs a low-overhead off-chain channel routing algorithm. This algorithm solves the optimal state routing scheme by improving the genetic algorithm based on the characteristics of state transition transactions and off-chain channel topology. It reduces the user's cross-shard state transition overhead and guarantees transition efficiency. Finally, this paper implements the ChannelLink protocol prototype system and uses Bitcoin transactions and the Lightning Network state to construct the dataset for experimental verification. Results show that in a scenario with 16 shards and a cross-shard transaction ratio of 5.21%, the

* 基金项目: 国家重点研发计划 (2021YFB2700301); 国家自然科学基金 (U22B2032)
收稿时间: 2023-09-25; 修改时间: 2023-12-21; 采用时间: 2024-02-12; jos 在线出版时间: 2024-06-14
CNKI 网络首发时间: 2024-06-17

sharding system integrated with the ChannelLink protocol can improve the throughput by 7.04%, reduce the transaction confirmation latency by 52.51%, and reduce the cost of cross-shard state transition by more than 45.44%. Meanwhile, the performance advantages of the ChannelLink protocol gradually increase as the number of shards and the cross-shard transaction ratio increase.

Key words: blockchain; sharding; state transition; state channel; routing algorithm

分片技术[1−8]是目前区块链性能优化的重要手段, 其将全局状态拆分至不同分片保存, 各分片并行完成交易处理与局部状态更新, 使得区块链系统性能得到了大幅提升. 然而, 随着分片系统的运行, 其链上应用的复杂度与关联度不断增加, 分片系统用户不可避免地就会存在跨片状态迁移的需求, 跨片交易也因此产生. 与片内交易不同, 跨片交易处理涉及多个分片的交易处理过程, 交易处理开销大、时间长. 因此, 如何降低跨片交易处理的平均开销、提升处理效率, 已经成为提升分片系统性能的关键要素.

跨片状态迁移协议是保障跨片交易正确处理的基础, 协议的基本执行过程可以分为源分片状态迁出、片间状态传输和目的分片状态迁入这 3 个阶段, 每笔交易的 3 个阶段串行执行、紧密耦合. 根据交易功能的不同, 跨片状态迁移相关交易可分为源分片状态迁出交易与目的分片状态迁入交易两类. 在现有协议中, 1 笔跨片交易至少拆分为 1 笔迁出交易与 1 笔迁入交易, 即平均每笔跨片交易至少关联 2 笔片内交易, 消耗 2 份链上交易处理资源 ($cost_{\text{cross}} \geqslant 2 \times cost_0$, $cost_{\text{cross}}$ 表示跨片交易的链上资源开销; $cost_0$ 表示分片系统中 1 笔片内交易的平均链上资源消耗) 并且至少涉及 2 轮共识过程.

链下状态通道技术[9−11]是当前应用最为广泛的区块链链下性能优化方案. 该方案通过将链上交易卸载至链下执行, 能够有效降低交易处理的链上资源开销. 与此同时, 利用链下通道灵活度高、即时确认的特点, 用户在链下通道间状态迁移的处理效率能够大幅优于链上处理过程. 因此, 我们受链下状态通道技术的启发, 计划将链下通道技术与分片架构相结合, 利用链下状态通道完成片间状态迁移, 解耦跨片状态迁移的三阶段过程, 将单笔状态迁出交易的迁出证明低开销地发往多个目的分片, 并且将目的分片的多笔状态迁入交易压缩为 1 笔链上交易, 提升状态迁移灵活度与效率, 有效降低跨片交易的平均开销与平均延迟, 从而提升分片系统性能.

然而, 现有链下状态通道方案无法满足分片间状态迁移通用性与安全性的需求, 无法完成分片链下通道网络间的通用状态迁移, 并且当异常发生时, 分片系统安全性将遭到严重破坏. 在链下通道网络中, 用户通过链下通道路由算法完成跨片状态迁移, 迁移效率能够满足分片系统性能优化需求. 但是, 现有通道路由算法存在着用户交易费用高的问题, 降低了用户使用链下网络完成跨片状态迁移操作的积极性, 无法充分发挥链下通道的性能优势.

为了解决以上问题, 实现分片系统性能优化的目标, 我们提出了一种基于链下状态通道的跨片状态迁移协议 (ChannelLink, CL). CL 协议利用链下状态通道灵活度高、即时确认的特点, 在保证跨片状态迁移通用性与安全性的前提下, 对分片系统原有跨片状态迁移协议进行扩展, 用户状态可以从链上迁出至其对应分片的链下状态通道网络. 各分片链下通道网络通过相同状态标识的节点构成了分片链下状态通道网络 (简称 CL 网络). CL 网络中用户状态从源分片迁出后可立即通过状态路由算法, 选择 1 条或多条低开销通道路径将该状态迁移至目的分片链下网络, 目的分片用户也可以在 CL 网络中对该状态进行即时确认.

用户状态进入 CL 网络后, 可按需进行任意多次 (多个分片间) 的状态迁移, 状态迁移过程无须进行链上共识, 仅需支付较少的状态路由费用. 当用户存在链上状态结算需求时, 才需消耗少量链上交易处理资源完成分片状态迁入 (CL 协议在目的分片状态迁入时会将多笔状态迁入交易压缩为 1 笔链上交易). 在当前跨片状态迁移协议中原本需要多轮的状态迁移过程, CL 网络只需 1 轮共识过程即可完成 (详见第 3 节), 大大提升了迁移效率, 降低了迁移开销, 更大程度地发挥了分片系统性能优势.

具体而言, 本文的主要贡献如下.

(1) 提出了一种基于链下状态通道的跨片状态迁移协议 ChannelLink, 将现有协议中紧密耦合的三阶段处理过程解耦, 保障了跨片状态迁移的通用性与安全性, 有效降低了跨片状态迁移的链上资源消耗, 提升了迁移效率, 并且给出了 ChannelLink 协议对于分片系统性能优化效果的理论分析与实验验证.

(2) 设计了一种低开销链下通道路由算法 River. ChannelLink 协议中, 用户跨片状态迁移交易费用最小化问题是一种线性规划问题. River 算法基于状态迁移交易与链下通道网络特征, 通过改进遗传算法求解最优状态路由方

案, 在保证状态迁移效率的同时, 大幅降低了跨片状态迁移交易费用.

(3) 实现了 ChannelLink 协议原型系统, 并基于比特币交易[12]和闪电网络状态信息[13,14]构造数据集进行实验验证. 实验结果表明, ChannelLink 系统在 16 个分片、跨片交易比例为 5.21% 的场景下, 分片系统吞吐量提升了 7.04%, 交易确认延迟降低了 52.51%, 跨片状态迁移开销下降了 45.44% 以上. 随着分片数量与跨片交易比例的上升, 本系统的性能优势逐步扩大. 此外, 当网络路由收费标准、最低迁移状态量等条件变化时, 本系统也有着很好的表现.

本文第 1 节介绍目前跨片状态迁移协议与链下状态通道的相关研究工作和基础知识. 第 2 节总体描述 ChannelLink 协议的设计思路. 第 4 节与第 5 节分别给出 ChannelLink 协议核心模块设计原理和 River 算法. 第 6 节阐述 ChannelLink 协议的性能、复杂度等相关分析. 第 7 节描述实验设置并评价 ChannelLink 协议的各指标性能. 最后总结全文.

1 相关工作与基础知识

本节首先分类介绍跨片状态迁移协议的主要类型及代表性工作. 之后, 简要介绍链下通道的基本原理以及链下通道路由算法的研究现状与主要问题. 需要特别说明的是, 在状态迁移及其原子性保障方面, 跨片与跨链交易处理 (状态迁移) 原子性保障机制是通用的. 但是, 在分片间或区块链间信息传输与有效性验证方面, 与跨链场景相比, 分片系统的不同分片间是同构的, 传输与验证方法更加简单, 无须再进行数据格式转换与验证方法加载等操作. 本文专注于跨片状态迁移过程, 在介绍研究现状的过程中, 不再区分所提方案是面向跨片或跨链场景设计.

1.1 跨片状态迁移协议

跨片状态迁移协议是跨片交易处理的基础. 依据跨片状态验证方式的不同, 当前跨片状态迁移协议可以分为 4 类. 图 1 中展示了各类协议处理分片 1 中用户 A 向分片 2 中用户 B 转移 10 份状态的跨片交易的基本流程.

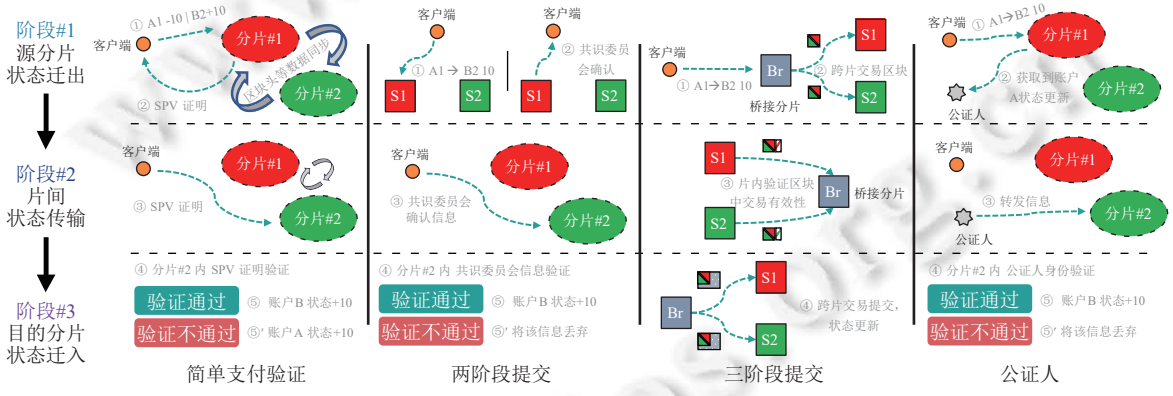


图 1 现有跨片状态迁移协议示意图

(1) 简单支付验证 (simplified payment verification, SPV)

基于 SPV 的方案[5,15]中, 各分片需定期同步存在跨片需求的所有分片的区块头, 并确定唯一主链. 跨片状态迁移时, 源分片节点确认交易已经被正确打包进本分片主链后, 构造 SPV 证明. 该证明包含源分片与目的分片状态更新操作 (阶段 1). 之后, 通过网络传输的方式, 将 SPV 证明发送至目的分片 (阶段 2). 目的分片在收到该交易的 SPV 证明后, 首先验证其所在区块是否在该分片的主链中, 若在主链中且 SPV 证明验证通过, 则根据 SPV 证明内容完成本分片状态更新, 进而完成跨片状态迁移 (阶段 3).

SPV 类方案利用区块链数据“自验证”的特点, 证明构造后无法随意篡改, 比其他方案安全性更高, 但定期同步区块头等信息链上开销更大. 为降低区块头同步与迁移证明的链上资源消耗, 近几年国内外提出了 NIPoPoWs[16]

和 FlyClient[17]方案, 将 SPV 线性的资源开销降低为非线性的资源开销 (随着协议运行时间的增加, 协议持续同步区块头链等信息, 开销线性增加).

(2) 两阶段提交 (two-phase commit)

基于两阶段提交的方案[2,3,8]中, 各分片间无须同步区块头与状态迁移证明, 而是基于哈希时间锁定合约[18] (hashed timelock contract, HTLC)、分片共识委员会状态更新信息等完成跨片交易所涉及分片状态的一致更新. 跨片交易参与方首先对交易内容与两阶段提交协议相关信息达成一致, 并将相关片上状态修改为锁定状态 (阶段 1). 在全部分片状态锁定完成后 (若仅部分分片完成状态锁定, 则协议超时后将已锁定分片状态解锁), 交易参与方交换哈希时间锁定信息或获取源分片 (分片 1) 的共识委员会确认信息, 并依据该信息依次更新交易所涉及的各分片状态, 从而完成跨片状态迁移 (阶段 2).

如图 1 所示, 若阶段 2 中客户端提交信息在目的分片无法通过验证, 则目的分片不进行任何操作. 这是因为, 在基于分片共识委员会的两阶段提交协议中, 若发往目的分片的验证信息不正确, 则一种情况是源分片已经被恶意节点控制, 另一种情况是客户端所发送信息并未通过源分片共识委员会验证. 综合考虑以上两种情况, 在此类方案中, 目的分片为保证本分片状态更新正确且降低共识过程交易处理开销, 当接收到无效信息后, 目的分片会直接丢弃该信息, 不进行额外操作. 与此同时, 为提升状态锁定与解锁效率, AC^{3}WN[19]等并行锁定与更新方案被陆续提出.

(3) 三阶段提交 (three-phase commit)

基于三阶段提交的方案中, 跨片交易处理过程不再由各分片依次处理, 而是通过片间协同的方式进行. 以 Pyramid[7]为例, 在预准备 (pre-prepare) 阶段 (阶段 1), 桥接分片对其所链接的普通分片间的跨片交易进行共识, 并将共识后的跨片区块发送 (依赖片内领导者) 给相关普通分片. 在准备 (prepare) 阶段 (阶段 2), 各普通分片验证该跨片区块其本分片相关内容的正确性, 并将结果返回给桥接分片. 在提交 (commit) 阶段 (阶段 3), 当所有普通分片均验证通过后, 桥接分片将提交信息发送给所有普通分片, 完成跨片交易的一致更新. 与基于两阶段提交的跨片状态迁移协议相比 (例如: OmniLedger[2]、RapidChain[3]等), 三阶段提交协议能够将跨片交易处理压缩在一个共识周期内完成, 使得跨片交易处理周期与片内交易相同. 但是, 三阶段提交协议需要占用 1 个桥接分片与多个普通分片的共识过程, 交易处理开销有所增加.

(4) 公证人

基于公证人的方案[20−22]中, 跨片状态迁移依赖某个公证人或公证人集合完成, 由公证人负责协调源分片与目的分片的状态一致更新. 例如: 图 1 中, 分片 1 中用户 A 发起的交易被分片 1 共识完成后, 经公证人转发至分片 2. 分片 2 节点在验证公证人身份之后, 就直接依据跨片状态迁移交易更新用户 B 状态. 与其他方案相比, 基于公证人的跨片状态迁移协议链上资源开销以及节点资源开销均最低, 但协议安全性完全依赖于公证人, 存在单点依赖的问题. 尽管 Tesseract[20]等方案引入了可信执行环境, 降低了公证人作恶风险, 但此类方案中存在较强假设, 适用场景依旧较为局限.

1.2 链下通道

链下状态通道[9,10] (简称: 链下通道) 是目前使用最广泛的区块链系统链下性能优化方案, 其基于哈希时间锁定合约[18]、多路支付原子性保障协议[23]等技术, 有效卸载了链上交易处理压力, 提升了系统整体交易处理吞吐量水平, 而仅把区块链作为清结算工具与安全性保障. 互相连通的链下通道及其节点构成了链下通道网络. 在链下通道网络[9,10]内, 未直接建立通道的节点间也可以通过链下通道路由的方式完成链下交易处理, 降低了通道建立的链上资源开销, 拓宽了链下状态通道技术的应用范围, 并有效提升了区块链系统的整体性能.

我们受链下状态通道技术的启发, 通过将分片架构与链下通道网络相结合, 使得源分片迁出状态可以不直接迁移至目的分片, 而是迁移至链下通道网络, 依靠分片间链下通道网络路由完成跨片状态迁移. 这样, 源分片的状态迁出交易不仅可以同时发往多个分片, 目的分片的多笔状态迁入交易也可以压缩为一笔链上交易, 有效降低了

跨片状态迁移的平均开销, 并提升了迁移效率 (链下通道间状态路由的处理时间低于链上区块间的出块间隔, 详见第 1.3 节). 然而, 现有链下通道及其路由算法无法满足片间状态迁移通用性与安全性需求, 并且缺乏针对用户交易费用的优化方案. 为此, 本文提出了一种新型跨片状态迁移协议 ChannelLink, 并基于该协议提出了一种低开销的通道路由算法.

1.3 链下通道路由算法

链下通道路由算法是决定链下网络性能的关键. 链下通道路由的目标是根据链下通道网络全局或局部信息, 在节点可用容量、交易费用等条件受限的情况下, 按照某种原则 (交易费低、交易成功率高等), 搜索交易双方间的可用路径集合, 基于该集合完成链下交易处理. 根据链下路径搜索与交易处理过程中是否存在分拆交易, 链下通道路由算法可分为单路路由[9,24,25]和多路路由[26,27]两类.

单路路由适用于交易平均转移状态少或状态无法拆分的情况. 但在大额交易处理过程中, 仅有少量路由路径可以完成交易处理, 单路路由可能使得路由路径选择过于集中, 造成通道长时间阻塞、交易费上涨, 从而使得链下通道网络效率大幅下降. 闪电网络节点数量较少, 网络中节点本地保存网络全局拓扑信息, 依据 Dijkstra 等经典算法, 选择最短路径完成交易处理. 随着网络规模扩大, 蚂蚁路由[24]、Flare[25]等算法被提出, 在此类算法中, 节点无须维护全局拓扑信息, 网络可扩展性较好, 但路径搜索与维护开销较大.

多路路由通过交易拆分, 能够有效降低每条路由路径处理压力, 提高交易处理成功率与效率. 在交易平均转移状态量大的情况下, 多路路由往往能够达到更高的交易成功率. SilentWhisper[26]算法引入了 Landmark[28]思想, 基于通道网络拓扑建立多棵生成树, 每个节点仅需维护到不同生成树根的路径, 提升了路由搜索效率, 但网络结构的变化会消耗较多资源用于生成树的更新, 并且依旧存在通道阻塞的问题. 为了实现对路由状态更细粒度的管理, Spider[27]算法将待迁移状态划分为多个状态单元, 基于包转发的思想提出一种启发式算法, 平衡路由通道中不同方向交易负载, 降低通道阻塞概率, 进一步提升了状态路由交易的成功率.

当前链下通道路由算法的优化指标主要为交易成功率、路径搜索效率、交易处理延迟等, 而缺少对路由交易费的优化. 由于路径搜索与状态路由时间比链上共识过程低了大概 1–2 个数量级, 并且链下通道网络直径较小, 绝大多数链下网络中交易均能在一个链上共识周期内完成 (本次实验中所使用的闪电网络数据集中, 最大连通分量的包含 13 565 个节点 (占全网 99.36%)、57 345 条通道 (占全网 99.92%), 网络直径为 10, 平均最短路径为 3.46). 因此, 与交易处理延迟相比, 在 CL 网络中, 路由交易费用与成功率反而成为更重要的优化指标. 如何在保证交易处理效率满足要求的情况下, 尽可能降低路由交易费用、提升交易成功率就成为 CL 网络通道路由算法所需解决的关键问题 (交易路径搜索失败后并不会对用户状态造成损失, 用户可以通过修改转移数量、增加交易费再次尝试. 路径搜索交易多次失败时, 用户需要通过链上交易的方式完成跨片状态迁移, 迁移开销与延迟将大幅升高).

2 ChannelLink 协议概述

为了解决跨片状态迁移交易的状态迁移开销与迁移效率的协同优化问题, 我们提出了基于链下状态通道的跨片状态迁移协议. 接下来, 本节将给出 CL 协议的系统模型, 并介绍 CL 协议的工作流程.

2.1 系统模型

CL 协议的整体框架如图 2 所示. 根据功能的不同, 本文将 CL 协议同样划分为源分片状态迁出、片间状态传输和目的分片状态迁入这 3 个模块, 分别对应图 2 中标号为①–③的操作步骤. 与当前跨片状态迁移方案不同的是, CL 协议用户能够按需调用状态迁出、网络中状态迁移与状态迁入模块, 模块间不再需要串行调用. 通过“一对多”和“多对一”的方式 (“一对多”是指一笔状态迁出交易, 可以将状态迁移至多个目的分片; “多对一”是指一笔状态迁入交易, 其中包含了来自多笔状态迁出交易的状态 (或状态证明)), CL 协议在保证迁移效率的同时, 能够大幅降低跨片状态迁移的链上交易处理资源消耗, 提升系统性能.

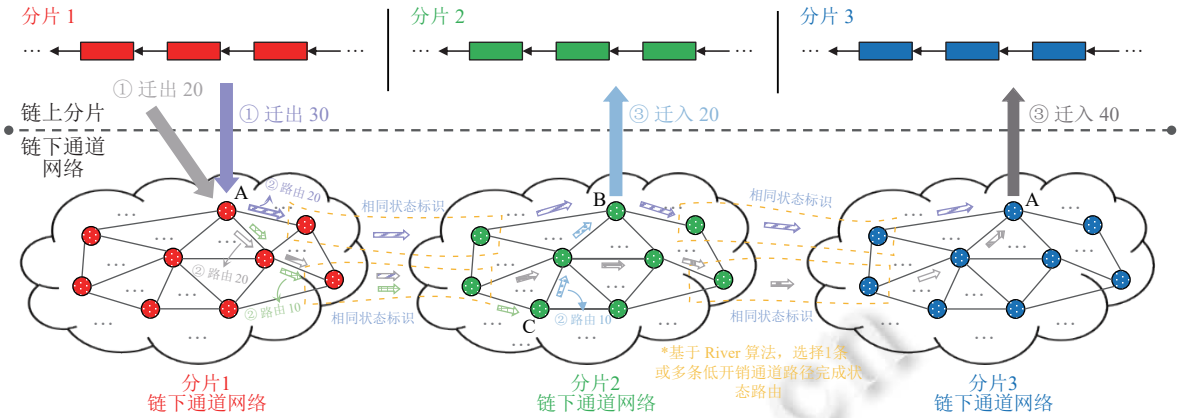


图 2 ChannelLink 协议示意图

在 CL 协议中, 系统分为链上分片系统和分片链下状态通道网络 (简称: CL 网络) 两个部分, 各部分具体定义如下.

● 链上分片系统. 链上分片系统为 CL 协议的分片链下状态通道网络提供了安全性保障, 链上分片系统安全是 CL 网络安全的前提. CL 协议面向通用链上分片协议设计, 是对系统原有跨片状态迁移协议的一种扩展, 协议模块依据不同分片系统接口进行适当修改即可加载使用. 因此, 本文不再详细描述链上分片系统设计细节.

● 分片链下状态通道网络 (CL 网络). CL 网络包含 3 个主要功能: (1) 源分片状态迁出证明接收, 并为相应用户生成对应链下状态; (2) 节点间状态迁移; (3) 目的分片状态迁入证明生成. 在一个包含 S 个分片的链上分片系统中, 其对应 CL 网络由 S 个分片的链下状态通道网络组成, 网络间通过相同标识 (所属权) 的节点相互链接, 构成一个整体. 与闪电网络[9]等链下网络相同, CL 网络由用户依据其状态迁移/路由服务等需求自发建立.

其他设置. CL 协议采用与相关工作[2–4,7]相同的设置: (1) 链上威胁模型为拜占庭容错模型, 并且对手是缓慢自适应的; (2) 节点间使用部分同步网络链接; (3) 共识节点轮换采用 Cuckoo 规则[3]; (4) 包含瞭望塔 (watch tower) 节点[10], 为 CL 网络用户提供链下状态托管、错误发现等服务, 避免因用户掉线、提交错误数据等操作造成诚实用户资产或状态损失.

2.2 工作流程

跨片状态迁移交易的处理开销大、时间长, 大大浪费了链上资源, 降低了用户热情并限制了系统性能. 如第 1.1 节所述, 现有跨片状态迁移协议尽管状态传输方式各有不同, 但其跨片状态迁移交易处理过程均至少涉及 2 轮链上共识. 这是因为在现有协议中, 3 个阶段串行执行、紧密耦合, 这种“一对一”的方式使得现有分片系统性能受到了极大的限制 (“一对一”是指现有跨片迁移协议处理涉及 s 个分片的跨片状态迁移交易时, 至少需要 s 笔对应的链上交易处理资源消耗). 因此, 如何实现跨片状态迁移交易的状态迁移开销与迁移效率的协同优化就成为跨片状态迁移协议的关键问题.

为了解决该问题, 本文首先提出了基于链下状态通道的跨片状态迁移协议 (第 3 节), 将源分片状态迁出、片间传输和目的分片状态迁入这 3 个阶段解耦, 提升链上交易处理资源的利用率, 降低跨片状态迁移开销. 由于分片链下状态通道网络的路由节点间收费标准千差万别, 当前状态路由算法无法满足用户交易费用优化需求. 因此, 本文提出了 River 算法 (第 4 节), 通过改进遗传算法优化状态路由方案, 降低路由费用. 与此同时, 将 CL 协议与区块链分片协议相联合, 能够更好地发挥各协议的优势, 提升系统交易吞吐量.

下面, 以用户 A、用户 B 与用户 C 的状态迁移为例 (如图 2 所示), 介绍 CL 协议各模块的具体工作流程.

● 模块 1: 源分片状态迁出. 源分片状态迁出模块包含链下状态通道建立和通道状态更新两个核心操作. 状态迁移用户首先与迁移服务提供节点建立链下通道. 通道建立后, 用户将其状态存入链上的通道状态管理合约, 更新链下通道状态, 从而完成源分片状态迁出. 如图 2 中步骤①, 用户 A 在通道建立后, 能够多次将其状态转移至链下,

用于跨片状态迁移等操作.

● 模块 2: 片间状态传输. 片间状态传输模块主要包括路径搜索、状态传输与异常处理这 3 个部分. CL 网络用户调用该模块功能, 即可将状态发往目的分片. 目的分片数量与发送次数可以按需设置. 如图 2 所示, 用户 A 从分片 1 迁出 30, 分别发往分片 2 的用户 C 的账户 10 以及用户 A 分片 3 的账户 20.

与此同时, CL 网络由链上分片系统提供安全保障, 状态到达目的分片链下网络后即传输完成, 用户可以立即使用 (CL 网络功能有限, 仅能进行通用状态迁移. 当需要进行复杂状态更新操作时, 需调用目的分片状态迁入模块, 将状态迁移至链上完成), 发给网络中其他用户, 大大降低了链上资源消耗. 如图 2 所示, 用户 C 在接收到用户 A 发来的状态之后, 可立即发给用户 B, 过程中无须进行链上操作.

● 模块 3: 目的分片状态迁入. 由于状态迁移的接收者在目的分片链下通道网络内不一定建立了链下状态通道, 目的分片状态迁入模块会根据状态来源与当前归属生成不同的状态迁入证明, 并在链上完成验证与状态更新. 如图 2 所示, 用户 A 在分片 3 将两笔来自分片 1 的跨片状态迁移交易压缩为 1 笔链上交易. 在 CL 网络中, 不同用户的迁入状态也可以通过迁移服务提供节点压缩为 1 笔链上交易, 大大降低链上交易处理开销与用户状态迁移的交易费.

3 ChannelLink 协议详细设计

在本节中, 我们将介绍 CL 协议的 3 个核心模块的设计原理与优化技巧.

3.1 源分片状态迁出

区块链分片系统用户进行跨片状态迁移时, 首先需要调用 CL 协议源分片状态迁出模块, 生成状态迁出证明, 并将源分片相应状态锁定或消除 (对于通用状态, 用户状态迁出后链上为锁定状态, 所有权归通道节点双方所共有. 对于非通用状态, 用户状态迁出后就在源分片消除, 不再转回, 从而避免“双花”攻击. 若非通用状态长期未迁往目的分片, 则用户可将完整证明发送给目的分片共识节点, 通过链上验证的方式完成状态迁移). 由于用户可能存在定期、多次的跨片状态迁移操作, 若每次生成完整状态迁出证明, 则链上开销大、交易费用高. 因此, CL 协议用户在第 1 次进行跨片状态迁移前, 首先与迁移服务提供节点建立链下状态通道 (如图 3 所示). 通道建立后, 用户每次仅需将少量状态迁移关键信息 (如后文图 4 所示) 上链处理, 并更新链下通道状态, 即可继续完成后续状态迁移操作.

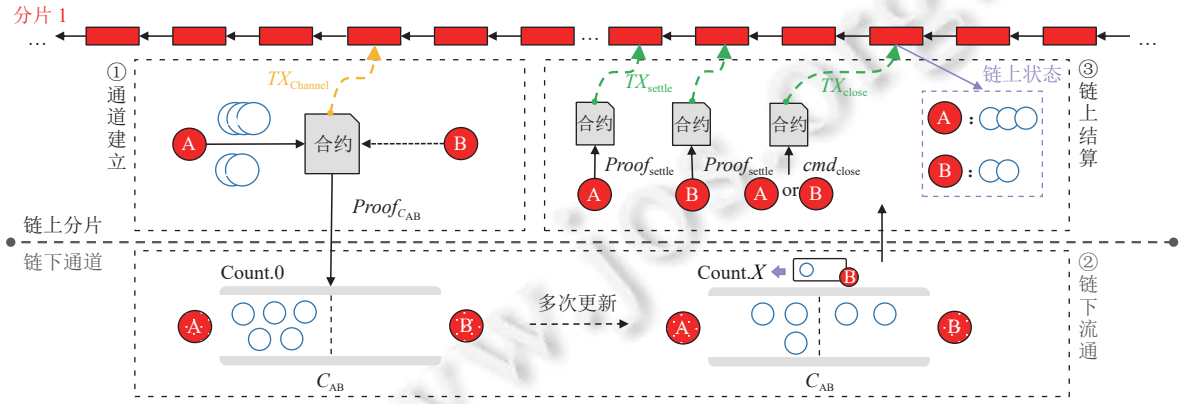


图 3 链下状态通道生命周期示意图

3.1.1 链下状态通道

CL 协议采用与闪电网络[9]、雷电网络[10]相同的链下状态通道建立、更新与通道内异常处理方案. 下面以分片 1 通道 C_{AB} 为例具体说明.

(1) 通道建立. 用户 A 计划将 5 份状态迁出分片 1, 其首先与用户 B 构造通道建立交易 TX_{Channel}, 其中包含用户 A 对于 5 份状态拥有权的证明以及用户签名等信息. 待分片 1 共识节点对本交易完成处理后, 用户 A 在分片 1 的 5 份状态就被成功锁定, 用户 A 与 B 也获取到通道成功建立的证明 $Proof_{C_{AB}}$. 此时, 用户 A 在通道内的余额为 5, 用户 B 的余额为 0.

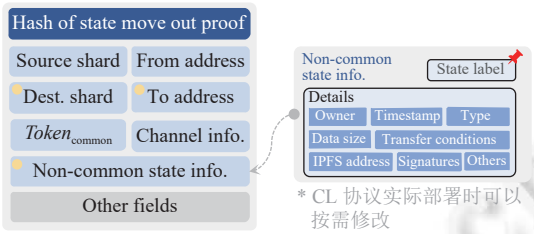


图 4 状态迁出证明数据结构

(2) 链下流通. 通道建立后, 用户 A 与用户 B 可以按需多次更新通道内余额分配方案. 用户双方也可以构造新的状态 (迁出/迁入) 证明, 增加/减少链下通道内状态, 并将必要信息上链处理.

(3) 链上结算. 当通道用户因通道利用率低、链上应用需要等原因需要关闭通道时, 用户 A 与用户 B 需要首先生成通道结算证明 ($Proof_{\text{settle}}$), 并将其上链. 通道结算挑战期[10]过后 (与传统链下状态通道的结算过程不同, CL 协议中的链下通道需要包含链上分片编号等额外标识信息, 验证过程中的存储开销会少量增加), 用户 A 或 B 即可发送通道关闭交易 TX_{close}, 完成链上结算. 与传统链下通道不同的是, CL 网络中通道可能包含非通用状态 (代币) 外其他状态信息, 每种信息都需要设置单独验证规则以保证安全结算.

3.1.2 状态迁出证明

与闪电网络[9]等传统链下通道网络不同, CL 网络中不仅可以传递通用状态 (代币) $Token_{\text{common}}$, 还可以传递非通用状态, 从而无须为每种状态的跨片迁移进行适应性改造, 增强了 CL 协议的通用性. 图 4 展示了状态迁出证明的数据结构, 其指定了状态迁移方向、详细迁移信息以及证明信息等.

图 4 中, 目的分片 (Dest. shard)、目的用户 (To address) 和非通用状态信息 (Non-common state info.) 标记了黄点. 这是因为, 当 CL 协议用户仅进行通用状态转移时, 无须在链上提前指定状态接收方, 仅需指定此次状态迁出的通用状态数量 $Token_{\text{common}}$. 一笔通用状态迁出交易可以将状态迁往多个目的分片, 大幅增加了 CL 网络状态迁移灵活性. 其他字段 (Other fields) 包含状态迁出交易成功执行的 SPV 信息、状态所有权信息等. 当通道关闭、异常处理时, 可以使用 SPV 等信息完成安全结算, 避免诚实节点状态损失.

当用户使用 CL 协议进行非通用状态迁移 (例如: 非同质化代币 NFT、合约自定义状态等) 时, 则需要完整的迁出证明信息, 并补充非通用状态的详细信息 (非通用状态迁移时, CL 协议迁出证明的数据量和链上计算开销与现有跨片协议相同), 否则存在“双花”的风险. 此外, 为了支付状态迁移所需开销, 非通用状态迁出时同样需要存入少量通用状态 (通用状态 (代币) 为区块链分片系统中合约状态流通的基本标定. 在实际部署中, CL 协议可以使用去中心化交易所将非通用状态转换为通用状态, 再支付状态迁出费用), 用于支付跨片状态迁移的开销 (链上计算与状态传输). 由于目的用户在目的分片不一定拥有链下通道, CL 协议同时指定目的分片的状态接收通道信息 (Channel info.). 当目的用户没有链下通道时, 该状态将发往目的分片路由服务节点, 使得目的用户能够正确完成状态接收.

3.1.3 模块优化

CL 协议需要建立链下状态通道, 对于长期使用区块链分片系统的用户而言, 能够显著降低跨片状态迁移开销. 但对于仅使用 1–2 次链上系统的用户而言, 通道建立开销可能已经超过其通过链上进行跨片迁移的成本. 用户可以直接选择链上跨片迁移协议完成状态迁移. 此外, 对于 CL 网络中长时间未成功迁移的状态, 用户可将原始状态迁出证明上链, 并附带通道所有节点签名, 即可将源分片锁定状态解锁或发往目的分片完成状态迁移, 从而避免状态长时间锁定.

3.2 片间状态传输

片间状态传输模块是 CL 跨片状态迁移协议的核心模块, 其性能高低将直接影响跨片状态迁移协议性能. 用户通过使用该模块, 即可将源分片迁出的状态或完整状态证明迁移至目的分片链下状态通道, 供状态接收者继续使用或链上结算. 下面以用户 A 从分片 1 的链下状态通道向其分片 2 主状态迁移 30 份状态为例来说明状态 (状态证明) 的片间传输流程, 其示意图如图 5 所示.

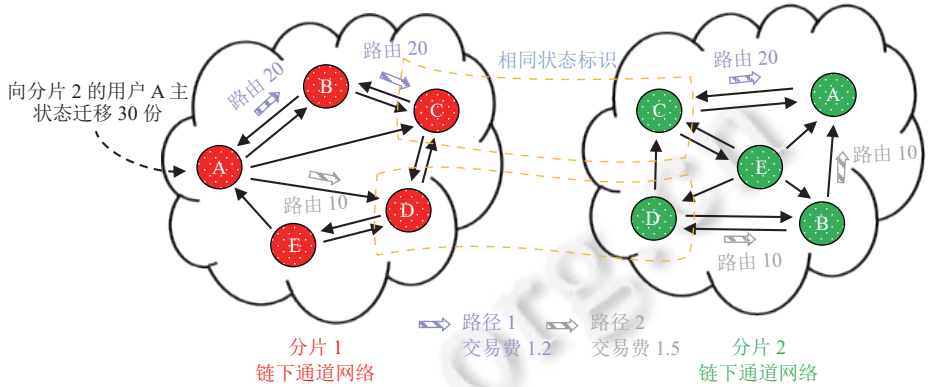


图 5 片间状态传输示意图

3.2.1 步骤 1: 路径搜索

用户 A 使用 River 算法 (详见第 4 节) 搜索状态迁移源节点 (分片 1 用户 A) 与目的节点 (分片 2 用户 A) 之间进行 30 份状态迁移交易费用最低的路由路径与转移状态量分配方案 *Path* . 如图 5 所示, 本次路径搜索共用到两条路径, 各条路径信息及转移状态量如下.

- 路径 1: $A_1 \to B_1 \to C_1 \to C_2 \to A_2$, 路由 20 份, 交易费 1.2.
- 路径 2: $A_1 \to D_1 \to D_2 \to B_2 \to A_2$, 路由 10 份, 交易费 1.5.

其中, A_1 代表用户 A 在分片 1 的链下状态通道网络中的节点, 其他符号同理.

对于通用状态, 状态本身可量化、可拆分, 当单一路径无法完成状态传输时, 用户可增加路由路径数量, 扩大搜索空间, 但同时也会增加固定路由费用. 关于路径及其转移金额的优化, 详见第 4 节. 由于源节点与目的节点间路由容量有限、源节点可以用于支付路由费用的通用状态量有限, 路径搜索过程不一定能够获取到可以完成状态迁移的可用路径 (组合) (在实际部署当中, 本文所提 River 路由算法 (详见第 4 节) 可按需进行分布式扩展. 例如: 参考 Flare[25]的路由表构建方式, 通过节点间通信获取最新可用路径, 并基于遗传算法完成路由方案优化). 路径搜索失败的交易并不会对用户状态造成损失, 用户可以通过修改单次转移数量、增加交易费用等方式再次尝试, 若长期失败则可以将链下状态上链, 而后通过链上完成状态迁移.

对于非通用状态, 其状态为状态迁出用户所特有, 无法在 CL 网络中进行拆分路由, 每次状态迁移仅能使用 1 条路由路径完成. 与通用状态不同, 非通用状态的无须占用大量通道容量, 仅需使用少量通用状态进行路由交易费支付, 故单路路由亦可保证较高成功率. 与此同时, 由于不再使用通用状态完成状态传输, 非通用状态的比例交易费收费标准也从通用状态路由量转变为单次传输的数据量大小, 每次传输依据数据量大小按比例收取比例路由费用.

3.2.2 步骤 2: 状态传输

路径搜索完成后, 用户 A 首先与其他链下通道节点协同, 确定哈希密文 *Secret*、锁定时间等信息, 并基于 HTLC 合约与多路支付原子性保障协议完成向其分片 2 主状态的 30 份状态的原子性迁移. 对于通用状态, 由于有链上分片与通道管理合约保证安全性, 用户在链下接收到通用状态后即可立即使用, 例如: 发往 CL 网络中其他用户、链上结算等. 该设计有效减少了用户通用状态跨片迁移的开销并提升了迁移效率 (用户使用 CL 网络中的链下状态无须上链, 且无须经过链上分片的多轮共识过程, 故状态迁移交易开销 (链上开销、交易费开销) 更低, 处

理延迟更短). 对于非通用状态, 由于其无法拆分且为避免“双花攻击”, 其状态证明中明确指定了目的地址, 状态传输完成后就无法再次传输.

3.2.3 异常处理

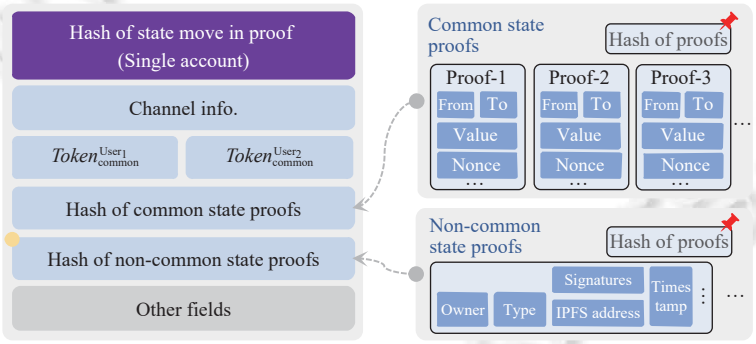
与目前单链链下通道网络相同, CL 网络状态传输过程中也可能存在解锁时间不够或通道对端恶意不解锁链下状态的情况, 从而可能造成诚实节点状态损失. 当出现此类情况时, 与现有链下状态通道方案相同[10], 用户 A 或路径上已经获取哈希时间锁解锁 *Secret* 的节点则立刻将 *Secret* 以链上交易的形式发送至跨片状态迁移所涉及通道的对应分片. 待分片将该交易 (1 笔或多笔) 打包后, 路由路径上其他未解锁通道即可依据 *Secret* 并行解锁, 保障用户状态安全.

3.3 目的分片状态迁入

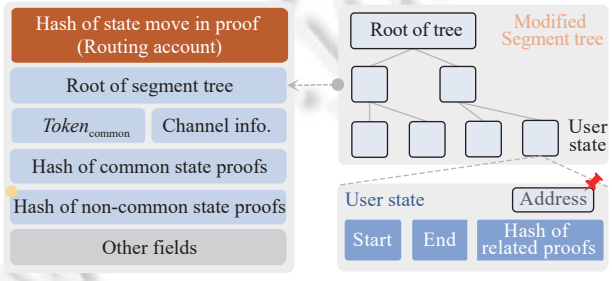
由于跨片状态迁移的目的地址对应用户可能尚未加入目的分片对应链下状态通道网络, 抑或在目的分片加入其链下状态通道, 但通道本身与跨片状态迁移的源地址所关联通道并未连通. 因此, CL 协议依据迁入证明 (或状态) 是否在目的地址所在通道, 设计了两种目的片状态迁入方法: (1) 单账户提交状态迁入证明; (2) 路由账户提交状态迁入证明. 各方法详细设计如下.

3.3.1 单账户提交状态迁入证明

在本方法中, 状态接收方能够在目的分片链下通道网络接收源分片发来的状态. 用户接收状态实质上是更新用户所在的 (某条或几条) 链下通道的通用状态分配方案, 并附加非通用状态迁移证明. 用户目的片状态迁入证明, 即修改后的通道状态分配方案更新证明, 其数据结构如图 6(a) 所示. 基于该结构, 用户可将与一条通道关联的多次通用与非通用状态更新的结果压缩为 1 笔链上交易, 链上也只需验证与最新状态更新相关的证明, 其他证明信息仅存储在链上, 供状态结算冲突时使用, 从而有效降低了链上交易处理资源开销. 此外, 其他字段 (Other fields) 包含了通道通用状态更新证明、非通用状态迁出证明等信息, 便于链上共识节点验证状态迁移与更新的正确性.



(a) 单账户



(b) 路由账户

图 6 状态迁入证明数据结构

3.3.2 路由账户提交状态迁入证明

当用户从源分片状态迁出节点至目的分片状态迁入节点不存在直接的路由通道时, 用户可以借助其他路由账户完成状态迁入目的分片的操作. 与普通用户两两间建立链下双人通道不同, 路由账户通过建立链下多人通道[29]提供状态路由、状态迁移等服务, 其状态迁入证明数据结构如图 6(b) 所示. 借助路由账户的链下多人通道, 状态接收用户无须发起链上通道建立交易, 只需支付少量链下状态路由费用, 可以有效降低跨片迁移开销. 与此同时, 路由账户基于线段树 (segment tree) 构造的状态迁入证明包含了本次通道状态更新所涉及用户的所有通用状态量、非通用状态及相关证明, 从而将一段时间内多用户、多次状态迁移压缩为 1 笔交易, 进一步降低了状态分配正确性验证所需的链上交易处理资源消耗.

4 低开销链下通道路由算法 River

为了降低用户跨片状态迁移的费用, 本文基于遗传算法[30]提出了一种低开销链下通道路由算法 River. 本节将介绍 River 算法具体设计原理.

4.1 链下通道路由算法模型

本文定义 C_{ij} 表示 CL 网络中节点 v_i 与节点 v_j 间通道 (通道 C_{ij} 与通道 C_{ji} 为同一条通道), 其包含两个节点 v_i 和 v_j 以及 $l_{v_i\to v_j}$ 和 $l_{v_j\to v_i}$ 两条链接. 在通道内, $balance_{v_i\to v_j}$ 表示节点 v_i 在通道 C_{ij} 内的余额 ($\geqslant 0$), 即链接 $l_{v_i\to v_j}$ 的可用容量; $balance_{v_j\to v_i}$ 表示节点 v_j 在通道 C_{ij} 内的余额 ($\geqslant 0$), 即链接 $l_{v_j\to v_i}$ 的可用容量.

互相连通的链下通道构成了链下通道网络 $G(V,E)$, 其中, V 代表用户节点集合, $V=\{v_i|i\in[N]\}$, N 表示网络中用户节点数量; E 代表节点间边的集合, 即网络中通道所有链接的集合, $E=\{l_{v_i\to v_j}|i,j\in[N] \text{ and } i\neq j\}$, $|E|$ 表示链接的数量, 其数值是通道数量的 2 倍. $balance_i$ 表示节点 v_i 在其所在的所有通道的余额, 如公式 (1) 所示.

$$balance_{v_i}=\sum_{v_j\in Ne(v_i)} balance_{v_i\to v_j} \tag{1}$$

其中, $Ne(v_i)$ 表示与节点 v_i 所有直接建立通道的邻居节点集合 (当节点间存在多条通道时, 本文将多条通道合并, 并且将收费策略合并, 详见第 6.1 节).

在链下通道网络 G 中, 未直接相连的节点间可以通过单条或多条路径的通道路由完成交易处理, 并且支付给提供路由服务的节点交易费. 下面以节点 v_i 向节点 v_j 通过 K ($K\geqslant 1$) 条路径转移 a 份状态的交易 (如公式 (2)) 为例, 说明链下通道路由中相关链接可用容量更新过程.

$$TX(v_i\to v_j,a,[(path_k,a_k)]) \tag{2}$$

其中, $a=\sum_{k=1}^{K} a_k$; 路径 $path_k=[v_i,v_r,\ldots,v_R,v_j]$ 转移的状态量为 a_k; v_r ($r\in[1,R]$) 表示路径 $path_k$ 中提供路由服务的节点.

4.1.1 交易费函数

由于路径 $path_k$ 中链接 $l_{v_r\to v_{r+1}}$ ($r\in[1,R]$) 提供了路由服务 (为了方便公式表示, 将节点 v_i 的下标记录为 0, 节点 v_j 的下标记录为 $R+1$), 节点 v_i 需要为路径 $path_k$ 支付的总交易费用的计算公式如下:

$$fee(path_k,a_k)=\sum_{r=1}^{R} fee_{v_r\to v_{r+1}}(a_k) \tag{3}$$

其中, $fee_{v_r\to v_{r+1}}(a_k)$ 表示链接 $l_{v_r\to v_{r+1}}$ 的交易费函数. 本文采用与闪电网络相同[31]的交易费计算方法, 节点通过链接 $l_{v_r\to v_{r+1}}$ 进行状态路由时的交易费包含固定交易费 $fee^{\text{base}}_{v_r\to v_{r+1}}$ 和比例交易费两个部分, 其计算公式如下:

$$fee_{v_r\to v_{r+1}}(a_k)=fee^{\text{base}}_{v_r\to v_{r+1}}+fee^{\text{ratio}}_{v_r\to v_{r+1}}\times a_k \tag{4}$$

其中, 每条链接的固定交易费 $fee^{\text{base}}_{v_r\to v_{r+1}}$ 与比例交易费率 $fee^{\text{ratio}}_{v_r\to v_{r+1}}$ 由节点 v_r 独立设置.

4.1.2 最大路由容量

由于每条链接上可用容量有限, 提供路由服务的链接转移状态量 $transfer^{\,k}_{v_r\to v_{r+1}}$ 不能超过其可用容量. 路径

$path_k$ 中链接 $l_{v_r\to v_{r+1}}$ 转移状态量的计算公式如下:

$$transfer^k_{v_r\to v_{r+1}} = a_k + \sum_{r'=r+1}^{R} fee_{v_{r'}\to v_{r'+1}}(a_k) \leqslant balance^k_{v_r\to v_{r+1}} \tag{5}$$

其中, $r\in[0,R]$; 当 $r=0$ 时, v_r 表示节点 v_i; 当 $r=R$ 时, 节点 v_r 与节点 v_j 直接相连, 无须缴纳额外交易费; 当 $r'=r$ 时, 节点 v_r 也不需要缴纳交易费, 故交易费求和时 r' 从 $r+1$ 开始计算; $balance^k_{v_r\to v_{r+1}}$ 表示链接 $l_{v_r\to v_{r+1}}$ 在路径 $path_k$ 中的可用容量, 当一条链接被多条路径复用时, 将其所有可用容量均分给各条路径 (初始时).

基于公式 (5), 当 $r\in[1,R-1]$ 时, 链接 $l_{v_r\to v_{r+1}}$ 的目的节点不为节点 v_j, 链接 $l_{v_r\to v_{r+1}}$ 的最大路由容量 $rc_{v_r\to v_{r+1}}$ 的计算公式如下:

$$rc_{v_r\to v_{r+1}} = \max\left\{0, \frac{balance^k_{v_r\to v_{r+1}} - fee^{\text{base}}_{v_{r+1}\to v_{r+2}}}{1+fee^{\text{ratio}}_{v_{r+1}\to v_{r+2}}}\right\}, r\in[1,R-1] \tag{6}$$

其中, $fee^{\text{base}}_{v_{r+1}\to v_{r+2}}$ 与 $fee^{\text{ratio}}_{v_{r+1}\to v_{r+2}}$ 分别表示路径 $path_k$ 中链接 $l_{v_r\to v_{r+1}}$ 的下一跳 $l_{v_{r+1}\to v_{r+2}}$ 的固定交易费与比例交易费率. 由于固定交易费可能大于链接 $l_{v_r\to v_{r+1}}$ 的可用容量, 当结果小于 0 时统一置 0. 此外, 当 $r=R$ 时, 节点 v_R 由于无须支付路由费用, 则其最大路由容量 $rc_{v_r\to v_j} = balance^k_{v_r\to v_j}$.

4.2 目标函数定义

由于链下状态迁移时间远低于链上出块间隔, 并且状态迁移效率已经能够满足跨片状态迁移需求, 因此, 当给定一个链下通道网络 $G(V,E)$ 时, 本文的目标是找到一个合适的路由路径组合与转移状态量分配方案, 即 $Path=\{(path_k,a_k)|k\in[1,K]\}$, 使得交易处理的路由费用最低. 交易费用最小化问题的目标函数如下:

$$\begin{cases} \min\ fee(Path) = \sum_{k=1}^{K} fee(path_k,a_k) \\ \text{s.t.} \begin{cases} \sum_{k=1}^{K} a_k = a \\ \sum_{k=1}^{K} transfer^k_{v_r\to v_{r+1}} - \sum_{k=1}^{K} transfer^k_{v_{r+1}\to v_r} \leqslant balance_{v_r\to v_{r+1}} \\ \sum_{k=1}^{K} transfer^k_{v_{r+1}\to v_r} - \sum_{k=1}^{K} transfer^k_{v_r\to v_{r+1}} \leqslant balance_{v_{r+1}\to v_r} \\ Variables: \forall\{l_{v_r\to v_{r+1}} \mid l_{v_r\to v_{r+1}}\in path_k, k\in[1,K]\} \end{cases} \end{cases} \tag{7}$$

CL 网络的路由费用最小化问题本质上属于一种线性规划问题. 当网络状态固定且规模较小时, 对于某笔状态迁移交易, 可以通过 SCIP、CMIP、lp_solve、GLPK 等线性规划工具求解最优状态路由方案. 但是, 在区块链分片系统中, 存在低开销跨片状态迁移的节点都将加入链下状态通道网络, 节点数量多, 网络规模大, 并且通道状态会依据交易处理结果实时更新, 传统线性规划工具无法使用. 因此, 我们计划通过改进遗传算法 (genetic algorithm) 来求解状态路由的最优方案.

● 转移状态量约束条件

由于同一条路由链接可能被多条路径同时使用, 并且不同路径可能使用同一通道的不同链接, 为保证交易成功执行, 本次交易处理过程中所有链接的状态转移总量不应超过其可用容量, 即:

$$\sum_{k=1}^{K} transfer^k_{v_r\to v_{r+1}} - \sum_{k=1}^{K} transfer^k_{v_{r+1}\to v_r} \leqslant balance_{v_r\to v_{r+1}},\quad \sum_{k=1}^{K} transfer^k_{v_{r+1}\to v_r} - \sum_{k=1}^{K} transfer^k_{v_r\to v_{r+1}} \leqslant balance_{v_{r+1}\to v_r} \tag{8}$$

其中, 当链接 $l_{v_r\to v_{r+1}}$ 不在路径 $path_k$ 中时, $transfer^k_{v_r\to v_{r+1}} = 0$.

当交易 TX 所有路径的超时时间设置相同, 或所有路径均完成才算执行完成时, 链接转移状态量使用上述约束条件. 当每条路径独立处理时, 即允许本交易仅从节点 v_i 成功转移部分状态至节点 v_j, 则每条链接转移状态量的约束条件可以简化为:

$$\sum_{k=1}^{K} transfer^{k}_{v_r \to v_{r+1}} \leqslant balance_{v_r \to v_{r+1}} \tag{9}$$

与前述约束条件相比, 使用本约束条件能够“部分”完成状态迁移 (部分完成状态迁移是指在多条路由路径的状态迁移过程中, 允许部分路径失败且各路径独立完成状态迁移过程), 且状态锁定时间较短, 但状态总迁移量更少. 与公式 (9) 相比, 使用公式 (8) 的约束条件, 所有路径能够保证同时完成或同时不完成[23]. 在保证同道双向转移状态量的差值不超过可用容量的前提下, 能够大大增加总转移状态量. 由于 CL 网络中对于路由时间并不敏感, 反而是需要转移更多状态, 因此, 使用公式 (8) 的约束条件.

4.3 River 算法设计

遗传算法[30]是一种经典的自适应启发式算法, 算法框架复杂度低、计算开销小. 传统遗传算法初始化时, 基于输入数据 (网络 $G(V,E)$ 状态信息, 状态迁移交易 TX 信息, River 算法参数等) 生成初始种群 P^0, 并作为第 1 轮迭代 ($t=1$) 的父个体 I_c^0. 在每轮迭代 ($t \in [1,T]$) 过程中, 父个体 I_c^{t-1} 依据网络状态信息进行交叉、变异, 并将其生成的子个体 I_c^t 加入种群 P' (此时, 种群 P' 包含所有父个体及其产生的子个体. 由于并非所有个体都保留至下一轮迭代, 此处使用 P' 进行区分). 每轮迭代 ($t \in [1,T]$) 计算结束后, 选择目标函数值结果最优的前 R 个体作为下一轮迭代的父个体 I_c^t, 并更新种群信息 P^t. 算法迭代结束后, 种群中目标函数值最优的个体就是最终搜索结果. 此外, 通过设置合适的最大迭代次数 T、交叉变异概率、个体选择比例等参数, 能够在优化运行结果的同时, 降低算法执行时间.

本文基于传统遗传算法, 针对链下通道网络中用户间状态迁移费用最小化问题, 提出了一种低开销链下通道路由算法 River.

4.3.1 种群初始化

在 CL 网络中, 用户可以选择任意数量 ($K \geqslant 1$) 的路由路径完成状态迁移. 随着路由路径数量的增加, 用户所使用的可用容量也随之增加, 用户单笔交易能够转移更多状态, 交易成功率更高. 但是, 路由路径越多, 所涉及的路由链接数量也越多, 用户为状态路由所支付的固定交易费会不断增加. 与此同时, 由于比例交易费的存在, 如何将总状态转移量划分至 K 条路由路径, 也会在很大程度上影响最终交易费的计算.

基于以上分析, CL 用户间状态迁移费用最小化问题的搜索空间可以划分为 3 个维度: (1) 路径条数 K; (2) 路径信息 $path_k$; (3) 路由状态量 a_k.

为了提升算法优化效果, River 算法初始种群中个体需要分布在较大的搜索空间中. 本文首先基于网络状态 $G(V,E)$ 与状态迁移交易信息 TX ($v_i \to v_j, a$), 使用 Dijkstra 算法获取节点 v_i 与节点 v_j 之间的 $K+M$ 条路由路径, 构成备选路径集合, 如公式 (10) 所示.

$$Path_{\text{init}} = \{path_p | p \in [1, K+M]\} \tag{10}$$

其中, K 表示用户 v_i 能够使用的最大路由路径数, K 的取值与网络状态及交易紧密相关, 我们将在第 6 节中分析讨论; M ($M \geqslant 0$) 用于增大初始种群构建是个体的路由路径选择空间, 随着 M 的增大, 初始种群中个体数量将以指数级增长.

基于备选路径集合 $Path_{\text{init}}$ 构造初始种群中 P^0 中个体, 个体 I_c^0 路由路径数量 $K' \in [1,K]$, 每个个体所选择的路径为备选路径中的一种组合, 即初始种群中路径数量为 K' 的个体数量为 $C_{K+M}^{K'}$, 那么, 初始种群 P^0 的个体总量为 $\sum_{K'=1}^{K} C_{K+M}^{K'}$. 随着 K 的增长, 初始种群中个体数量以指数级增长, 为了优化搜索时间, 本算法规定, 当初始种群中个体数量大于 $MAXI$ 时, 从 $\sum_{K'=1}^{K} C_{K+M}^{K'}$ 个体中随机选择 $MAXI$ 个, 并需保证对于任意路由路径 $K' \in [1,K]$ 均至少包含 1 个个体.

个体路径信息确认后, 本文首先基于网络中状态信息计算每条路径的最大路由容量 $rc_{path_{k'}}$. 每条路径的最大路由容量等于该路径上每条链接最大路由容量的最小值, 即 $rc_{path_{k'}} = \min\{rc_{v_r \to v_{r+1}}\}$. 当一条链接被多条路径共用时, 将其可用容量均匀分配给各条路径, 再计算各路径上该链接的最大路由容量. 个体每条路径路由状态量 $a_{k'}$ 的初始

值为 $\min\{a/K', rc_{path_{k'}}\}$. 若某 (几) 条路径最大路由容量小于平均路由状态量, 则将剩余待路由状态分配给尚有可用路由容量的路径. 若所有路径的最大路由容量之和小于本次状态迁移的状态总量 a, 则将每条路径的路由状态量均设置为 $a_{k'}$, 通过多轮迭代寻找可能完成本交易的个体, 并优化交易费用.

$Path(I_c^0)$ 表示个体 I_c^0 的路由路径与路由状态量的分配方案. 基于网络状态信息即可求解本方案的交易费用 $fee(Path(I_c^0))$. 若个体 I_c^0 的分配方案无法完成本状态迁移交易, 则将该方案的交易费标记为正无穷.

4.3.2 遗传变异操作

为了扩大 River 算法的分配方案搜索空间, 本文定义了路径交叉 (新路径、新路由状态量)、节点变异 (新路径) 和数值变异 (新路由状态量) 这 3 种操作. 种群中父个体在每轮迭代过程中都有概率进行以上 3 种变异操作, 产生新的子个体加入种群. 每轮迭代结束后, 为了提升搜索效率, River 仅保留路由交易费用最低的前 R 个个体作为下一轮迭代的父个体, 并更新种群信息.

● 路径交叉

路径交叉是指父个体中两条路径如果存在交叉点, 则随机交换交叉点前/后的路由路径, 而原路径的路由状态量不变, 所产生的子个体也将加入种群中. 在本文实验中, 父个体如果确定进行路径交叉操作, 则 River 算法将从父个体的路径集合中随机选择 2 条, 尝试进行路径交叉, 其示意图如图 7 所示 (父个体进行路径交叉操作的概率为 $prob_{\text{path\_cross}} \in [0,1]$, 选择交叉点前子路径进行交叉的概率为 $prob_{\text{cross\_front}} \in [0,1]$).

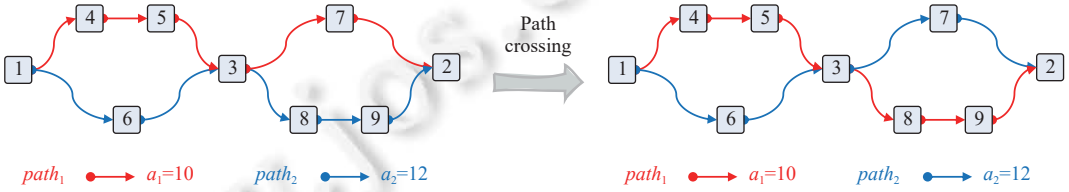


图 7 路径交叉操作示意图

节点 v_3 是路径 1 和路径 2 两条路径的交叉点, 其将路径 1 与路径 2 分割为 4 条子路径. 通过将交叉点前/后的路径交叉, 可以产生两种结果. 一次路径交叉会随机选择一种结果, 产生 1 个子个体. 具体而言: 当 4 条子路径均不相同时, 子个体随机选择交叉点前后路径进行交叉操作; 当交叉点前子路径相同时, 子个体仅选择交叉点后子路径进行交叉, 反之亦然. 当存在多个交叉点时, River 算法将随机 (均匀) 选择一个交叉点进行路径交叉操作. 通过路径交叉能够产生更多的路由路径信息, 并且能够尝试在不同路径上进行路由状态量分配, 增加了搜索空间.

● 节点变异

节点变异是指随机选择父个体中 1 条路径中的 1 个路由节点 v_r, 将其替换为依旧能够保证该路径连通的其他候选路径. 候选路径中包含变异节点 v_r 前后两个节点间的至多 R 条路径 (在算法实现中, 可以独立设置节点变异的候选路径数量, 本文使用与每轮迭代保留父个体数量相同的 R. 由于变异节点前后的节点的路径数量可能少于 R, 故文中说至多 R 条), 产生至多 R 个子个体, 其示意图如图 8 所示 (父个体进行节点变异操作的概率为 $prob_{\text{node\_mutation}} \in [0,1]$). 路径 1 在节点 v_3 处进行变异的过程. 首先从节点 v_3 的前后节点 v_5 与节点 v_7 之间的路径中选出 $R=4$ 条路径构成候选路径集合. 随后依次使用候选路径替换父个体中路径 1, 生成 4 个新的子个体加入种群.

通过节点变异能够产生大量的候选路径信息, 构造更多 (相较于其他两种操作) 子个体. 但是, 由于本文候选路径集合搜索过程采用与备选路径搜索相同的算法, 部分节点间搜索时间可能较长. 为了提升迭代速度, 本文将候选路径的单次搜索时间限制为 0.1 s.

● 数值变异

数值变异是指随机选择父个体中两条路径, 调整路由状态量分配方案, 将其中一条路径路由状态量减少, 另一条路由状态量增加, 其示意图如图 9 所示 (父个体进行数值变异操作的概率为 $prob_{\text{value\_mutation}} \in [0,1]$, 每次路由状态量变化比例为 $percent_{\text{value\_mutation}} \in [0,1]$). 值得注意是, 在路径变异与节点变异之后, 同一个体的分配方案中可能包含多条相同的路径. 因此, 父个体在进行数值变异操作前需要先将相同路径的路由状态量合并.

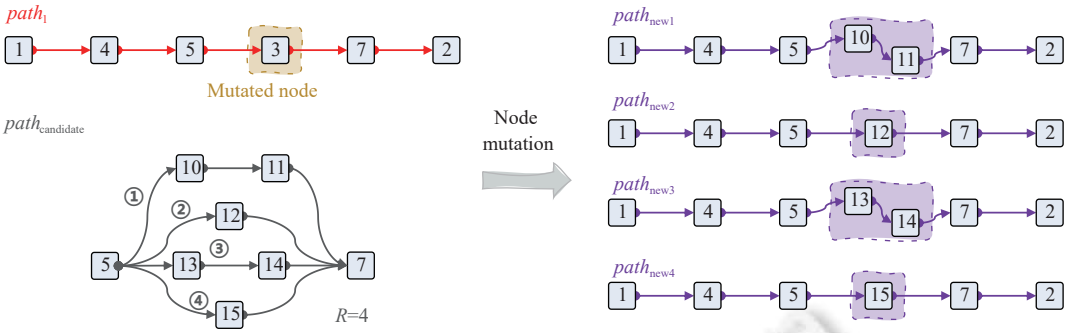


图 8 节点变异操作示意图

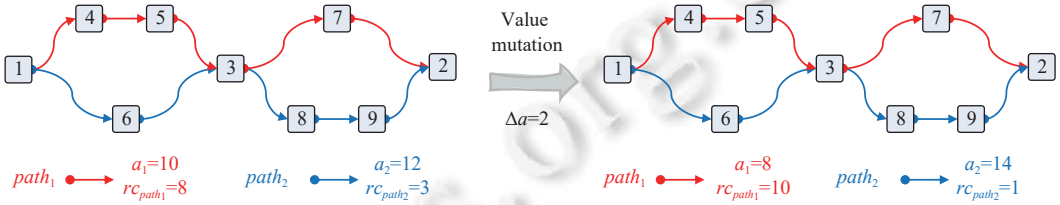


图 9 数值变异操作示意图

图 9 展示了路径 1 和路径 2 进行数值变异的过程. 其中, 路径 1 的路由状态量减少, 路径 2 增加. 路径 2 能够增加的路由状态量等于其最大路由状态容量与当前路由状态量的差值. 如果路径 1 减少状态量大于该差值, 则可能使得该个体无法完成本状态迁移交易. 因此, 路径 1 和路径 2 的路由状态变化量等于两者中的最小值. 通过数值变异能够在个体路由路径集合不变的情况下, 将待路由状态更合理地分配至各路径, 从而以更低的交易费完成更多的状态路由.

● 个体选择

每轮迭代结束后, 依次计算种群中个体路由路径与路由状态量分配方案的路由费用, 并按照从低到高的顺序选择路由交易费用最低的前 R 的个体作为下一轮迭代的父个体, 并更新种群信息 P^t. 若某轮迭代过程中, 种群中个体数量小于 R 个, 则将当前种群中个体随机复制, 进行补全, 使得下一轮迭代的父个体数量至少为 R.

当迭代次数达到最大迭代次数 T 或连续两轮迭代的种群信息不再变化时, 就结束本次搜索过程, 将路由费用最低的个体的路由路径与路由状态量分配方案作为最终结果.

4.4 River 算法流程

River 算法的伪代码如算法 1 所示. 算法输入包括 CL 链下通道网络信息、状态迁移交易信息、最大路由路径数以及最大迭代次数等参数 (River 算法参数含义详见表 1). 本算法首先基于算法输入, 搜索 $K+M$ 条节点间的路径, 构成备选路径集合 (第 2 行). 为避免备选路径搜索时间过长, 本文实验中单次路径搜索时间 (单位: s) 的上限为 1 s. 如果备选路径集合中的路径数量小于 $K+M$, 为保证算法迭代的正常进行, 需要更新两者的取值 (第 3–5 行). 随后, River 算法基于备选路径集合等信息, 构造初始种群, 并求解种群中个体的路由路径与路由状态量分配方案 (第 7–8 行).

遗传算法迭代开始后, River 算法依次对上一轮迭代所保留的父个体进行路径交叉 (第 12–13 行)、节点变异 (第 14–15 行) 与数值变异 (第 16–17 行) 操作, 并将上一轮迭代的所有父个体与本轮新产生的子个体均保存在临时种群中. 每轮迭代结束后, 将交易费用最低的前 R 个个体保留, 并更新当前最优分配方案 $Path$ (第 19–20 行). 如果连续两轮迭代种群信息没有更新, 则结束迭代 (第 22–23 行). 最后, River 算法将路由交易费用最低的分配方案作为结果返回给节点 v_i 等相关节点, 完成本交易的执行.

算法 1. 低开销链下通道路由算法 River.

输入: G, TX, K, T, M, R, $MAXI$, $prob_{\text{path\_cross}}$, $prob_{\text{cross\_front}}$, $prob_{\text{node\_mutation}}$, $prob_{\text{value\_mutation}}$, $percent_{\text{value\_mutation}}$;
输出: $Path$.

1. 增加距离 // 0. 搜索 $K+M$ 条节点 v_i 与节点 v_j 之间的路径, 创建初始种群构造的备选路径集合
2. $Path_{\text{init}} = \text{SearchInitPathSet}(G, TX, K, M)$.
3. **if** $len(Path_{\text{init}}) < K+M$ **then**
4. $K = \min\{K, len(Path_{\text{init}})\}$.
5. $M = len(Path_{\text{init}}) - K$.
6. // 1. 构造初始种群
7. $P^0 = \text{ConstructInitPopu}(Path_{\text{init}}, K, M, MAXI)$.
8. $P^0 = \text{SetPathValueAndCalRoutingFee}(G, P^0)$.
9. 增加距离 // 2. 算法迭代
10. **for** $t = 1, 2, 3, \ldots, T$ **do**
11. // 2.1. 个体变异
12. **if** $random.random() < prob_{\text{path\_cross}}$ **then**
13. $P^{t'} = \text{PathCrossing}(P^{t-1}, prob_{\text{cross\_front}})$.
14. **if** $random.random() < prob_{\text{node\_mutation}}$ **then**
15. $P^{t'} = \text{NodeMutation}(P^{t-1}, P^{t'}, G, R)$.
16. **if** $random.random() < prob_{\text{value\_mutation}}$ **then**
17. $P^{t'} = \text{ValueMutation}(P^{t-1}, P^{t'}, percent_{\text{value\_mutation}})$.
18. // 2.2. 个体选择
19. $P^t = \text{SelectTopRIndi}(P^{t'}, R)$.
20. $Path = \text{MinRoutingFeePath}(P^t)$.
21. // 2.3. 迭代终止条件判断
22. **if** $\text{SHA256}(P^t) == \text{SHA256}(P^{t-1})$ **then**
23. **break**.
24. // 3. 运行结束
25. **Return** $Path$.

表 1 River 算法参数定义

变量	定义	变量	定义
G	ChannelLink链下状态通道网络	$prob_{\text{node\_mutation}}$	父个体进行节点变异操作的概率
TX	状态迁移交易	$prob_{\text{value\_mutation}}$	父个体进行数值变异操作的概率
K	一笔交易的最大路由路径数量	$percent_{\text{value\_mutation}}$	数值变异每次路由状态量变化比例
T	算法最大迭代次数	$Path$	路由路径与转移状态量分配结果
M	路由路径选择空间调整参数*	$Path_{\text{init}}$	初始种群构造的备选路径集合
R	每轮迭代保留的个体数量	P^0	初始种群
$MAXI$	初始种群个体数量最大值	P^t	第 t 轮迭代后的种群信息
$prob_{\text{path\_cross}}$	父个体进行路径交叉操作的概率	$P^{t'}$	第 t 轮迭代过程中的种群信息
$prob_{\text{cross\_front}}$	父个体选择交叉点前子路径进行交叉的概率		

注: * 种群初始化时, River算法将获取节点 v_i 与节点 v_j 之间的 $K+M$ 条路由路径, 构成备选路径集合(详见第4.3.1节)

5 理论分析

在本节中, 我们将给出 CL 协议的性能分析、跨片状态迁移交易原子性的证明、River 算法的收敛性与复杂度以及链上分片系统集成的相关分析.

5.1 ChannelLink 协议性能

CL 协议采用与传统分片方案以及链下状态通道网络相同的安全设置. 当系统安全设置相同且满足安全性假设时, 使用 CL 协议的系统即可保证其跨片状态迁移的原子性 (详见第 5.2 节). 因此, 本节将在系统安全的前提下, 分析 CL 协议的性能. 分析指标方面, 用户在使用 CL 协议时将按需完成跨片状态迁出与迁入, 并且链下网络 TPS 能够满足片间状态迁移需求. 因此, CL 协议性能分析的核心指标就是链上资源开销与跨片交易延迟. 在此基础上, 本文还给出了 CL 协议对于分片系统吞吐量优化效果的理论分析.

5.1.1 链上资源开销

为了便于进行跨片状态迁移协议的开销, 本文假设 1 笔片内交易的链上交易资源平均开销为 $cost_0$, 平均交易费为 fee_0, 每笔跨片交易涉及 2 个分片 (1 个源分片、1 个目的分片).

● 现有协议

现有跨片状态迁移协议处理跨片状态迁移时, 需要至少拆分为源分片状态迁出与目的分片状态迁入两笔交易. 相较于片内交易, 这两类交易需要消耗更多的交易处理资源与更高昂的交易费用. 现有跨片状态迁移协议的链上资源开销 (公式 (11)) 与交易费用 (公式 (12)) 如下:

$$cost_{\text{cross}}^{\text{now}} = (\beta_{\text{out}} + \beta_{\text{in}}) \times cost_0 \tag{11}$$

$$fee_{\text{cross}}^{\text{now}} = (\beta_{fee_{\text{out}}} + \beta_{fee_{\text{in}}}) \times fee_0 \tag{12}$$

其中, β_{out} 与 β_{in} 分别表示状态迁出与状态迁入交易相较片内交易平均链上资源开销的倍数, 取值范围为 $(1, +\infty)$, 即 $cost_{\text{cross}}^{\text{now}} > 2 \times cost_0$; $\beta_{fee_{\text{out}}}$ 与 $\beta_{fee_{\text{in}}}$ 分别表示跨片状态迁出与迁入交易相较片内交易的交易费倍数, 取值范围为 $(1, +\infty)$.

● CL 协议

CL 协议用户在进行跨片状态迁移前, 需要在所涉及分片的链下状态通道网络建立状态通道. 假设链上分片系统包含 S 个分片, 则用户需要至多建立 S 个分片的链下状态通道 (一次性开销). 在 CL 协议中, 用户可以通过“一对多”和“多对一”的方式进行跨片状态迁移, 有效降低跨片状态迁移交易的平均开销. 假设 CL 协议用户一次状态迁出平均迁往 γ_{out} ($\geqslant 1$) 个目的分片; 一笔状态迁入交易平均包含 γ_{in} ($\geqslant 1$) 个状态迁入操作. 那么在状态通道建立后, CL 协议平均每笔跨片状态迁移交易的链上资源开销 (公式 (13)) 与交易费用 (公式 (14)) 如下:

$$cost_{\text{cross}}^{\text{CL}} = (\rho_{\text{out}}/\gamma_{\text{out}} + \rho_{\text{in}}/\gamma_{\text{in}}) \times cost_0 \tag{13}$$

$$fee_{\text{cross}}^{\text{CL}} = (\rho_{fee_{\text{out}}}/\gamma_{\text{out}} + \rho_{fee_{\text{in}}}/\gamma_{\text{in}}) \times fee_0 + \gamma_{fee} \times \gamma_{\text{out}} \times fee_0 \tag{14}$$

其中, ρ_{out} 与 ρ_{in} 分别表示 CL 协议状态迁出与状态迁入交易相较片内交易平均链上资源开销的倍数, 取值范围为 $(1, +\infty)$, 由于 γ_{out} 与 γ_{in} 的取值 $\geqslant 1$, 故 CL 协议的平均链上资源消耗有望接近甚至低于片内交易 (CL 网络中用户在链下接收到通用状态后, 可以直接使用, 而无须消耗链上资源). $\rho_{fee_{\text{out}}}$ 与 $\rho_{fee_{\text{in}}}$ 分别表示 CL 协议跨片状态迁出与迁入交易相较于片内交易的交易费倍数, 取值范围为 $(1, +\infty)$. γ_{fee} 表示 CL 网络每次状态迁移的交易费用相较于片内交易的交易费的倍数.

● 协议开销对比

对于某个用户而言, 假设其使用本系统共进行 Q 次跨片状态迁移操作, 并且在每个分片均建立链下通道, 单个分片链下通道建立平均资源开销相较于单笔片内交易的倍数为 ψ, 交易费倍数为 ψ_{fee}. 通过联立公式 (11)–(14), 用户可以依据其跨片状态迁移交易的频次, 判断是否加入 CL 网络 (在 OmniLedger[2] 分片系统中, 当分片数量为 16 时, 跨片交易平均拆分为 3.78 笔片内交易. 本文采用向上取整的方式, 即每笔跨片交易平均涉及 4 个分片内状态单元的更新. 因此, 现有协议跨片交易平均开销设置为片内交易的 4 倍, CL 协议的迁出与迁入交易关联地迁移

操作数量也设置为 4). 例如: 在某个使用 CL 协议的分片系统中, $\beta_{out}=2$、$\beta_{in}=2$、$\rho_{out}=1.8$、$\rho_{in}=1.8$、$\gamma_{out}=4$、$\gamma_{in}=4$、$S=16$、$\psi=2$. 图 10 展示了 CL 协议与现有跨片状态迁移协议的链上资源开销与用户跨片状态操作次数之间的关系 (其中, “Internal”表示同等数量片内交易开销), 其中, CL 协议用户链下通道建立开销等于 $\psi\times S=32$ 倍的单笔片内交易开销 ($cost_0$).

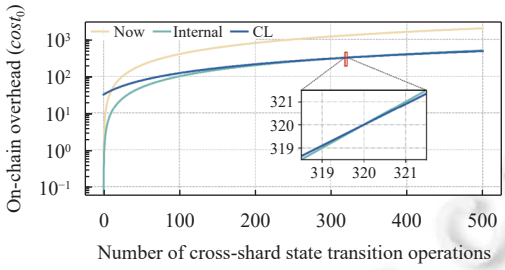


图 10 链上资源开销与跨片状态迁移操作次数间关系图

当 $Q>10.32$ 时, 用户使用 CL 协议可以消耗更低的链上资源. 由于此时用户在所有分片链下通道网络均建立通道, 开销较大. 在实际使用中, 用户可以按需在其频繁交互的分片建立链下通道, 而在低频交互的分片间使用现有跨片状态迁移协议.

当 $Q>320$ 时, 用户跨片状态迁移的平均开销将低于片内交易, 可见对于资金充足且频繁使用系统的用户, 其提供跨片状态路由服务不仅可以获取路由收益, 还能有效地将系统链上交易处理压力迁移至链下, 进一步释放系统性能. 同理, 基于系统参数, 用户也可以完成交易费用开销的分析.

5.1.2 跨片交易延迟

跨片状态迁移交易延迟 (简称: 跨片交易延迟) 是指跨片状态迁移用户从源分片状态迁出至目的分片状态迁入完成所需要的平均时间 (单位: s). 如前文所述, 现有跨片状态迁移协议至少占用两轮分片共识周期, 其跨片交易延迟至少为 2 倍片内交易确认延迟与片间状态 (状态证明) 传输延迟之和. 与链上资源开销分析相同, 用户通用状态的跨片状态迁移即可转换为 CL 网络中不同节点间状态迁移过程, 无须涉及链上共识过程, 其处理延迟等于 CL 网络中路径搜索与状态传输延迟之和, 单笔状态迁移时间远低于链上共识出块间隔 (详见第 6 节), 且多笔交易间可并行执行, 从而有效降低跨片迁移交易的平均延迟. 对于非通用状态, 状态或状态证明无法在 CL 网络中多次迁移, 但与现有协议相比, CL 协议能够一次完成 γ_{in} 个状态迁入, 可以通过压缩状态迁移证明的方式增大 γ_{in}, 从而即时将其状态迁移证明上链, 完成结算, 达到降低跨片交易延迟的目的.

5.1.3 分片系统吞吐量优化效果

与片内交易相比, 分片系统跨片交易处理需要消耗数倍于片内交易的链上交易处理资源, 从而使得分片系统吞吐量优化效果无法逼近其理论上限 S (分片数量为 S 时, 分片系统吞吐量优化效果的理论上限为 S). CL 协议通过将跨片状态迁移的三阶段过程解耦, 大幅降低跨片状态迁移的平均链上交易处理资源消耗, 释放系统性能. 基于公式 (11) 和 (13), 可以得到通道建立后 (由第 5.1.1 节中分析可知, 对于高频用户而言, 其通道建立开销远低于跨片状态迁移次数), 部分用户使用 CL 协议 (公式 (15)) 的分片系统跨片交易平均开销, 其中, $ratio_{use\_CL}$ 表示使用 CL 协议完成的跨片交易比例.

$$cost_{cross}^{CL'}=ratio_{use\_CL}\times cost_{cross}^{CL}+(1-ratio_{use\_CL})\times cost_{cross}^{now} \tag{15}$$

根据第 5.1.1 节中分析, 假设仅有“热点用户”使用 CL 协议, 而普通用户依旧使用现有跨片状态迁移协议. 通过分析 Bitcoin[32]交易数据, 我们发现占全网 2% 的“热点用户”发送了全网 32.44% 的交易, 即 $ratio_{use\_CL}=32.44\%$. 与此同时, 根据 OmniLedger[2]协议设计, 其在 16 个分片的场景下, 跨片交易比例为 98.8%, 跨片交易平均拆分为 3.78 笔片内交易 ($cost_{cross}^{now}=3.78\times cost_0$). 因此, 使第 5.1.1 节中参数进行初始化, 可以得到部分使用 CL 协议分片系统, 跨片交易平均链上资源开销为 $2.85\times cost_0$, 减少了 24.72%.

跨片交易平均链上资源开销的降低, 使得使用 CL 协议的分片系统能够使用更多的资源完成片内交易处理.

在相同测试环境下, 分片系统吞吐量优化效果 IF 的计算公式如下:

$$Q_0 = \frac{V \times S}{ratio_{\text{cross}} \times cost_{\text{cross}} + (1 - ratio_{\text{cross}}) \times cost_0} \tag{16}$$

$$IF = \frac{Q_0}{V \times S} = \frac{1}{ratio_{\text{cross}} \times cost_{\text{cross}} / cost_0 + (1 - ratio_{\text{cross}})} \tag{17}$$

其中, Q_0 表示分片系统饱和状态下每轮共识过程所能处理的交易数量的期望; V 表示分片内每个区块能够打包的片内交易数量, 例如: $V = 4096 \times cost_0$, 即每个区块能包含 4 096 笔交易; $ratio_{\text{cross}}$ 表示跨片交易比例.

从公式 (17) 可以看出, 分片系统吞吐量优化效果等于每轮期望处理交易数量除以分片系统所能处理的片内交易总量, 分母表示没有跨片交易的理想情况下, 分片系统每轮共识过程所能处理的交易数量上限. 然而, 由于片间负载失衡等问题[6], 分片系统运行过程中无法长期处于饱和状态, 系统实际吞吐量优化效果可能与公式结果略有差异. 将以上参数的数值带入公式后得到, 现有协议与部分使用 CL 协议的分片系统的吞吐量优化效果分别为 0.266 9 与 0.354 2, 部分使用 CL 协议的系统能够实现 32.7% 的吞吐量优化效果提升.

5.2 跨片状态迁移交易原子性

ChannelLink 协议中链上分片系统与链下状态通道网络的安全设置与现有方案相同. 因此, 当系统安全设置 (例如: 恶意节点比例、通道路由超时时间、DDoS 攻击防御等) 相同时, ChannelLink 协议不会引入新的攻击漏洞或模式, 并且能够保证与现有系统达到相同的安全等级. 在本节中, 本文将在链上分片系统与链下状态通道网络 (CL 网络) 均安全 (链下状态通道网络安全是指网络中大多数节点为诚实节点、遵守协议, 并且恶意节点的攻击行为无法成功) 的前提下, 从是否允许跨片状态迁移部分完成 (详见第 4.2 节) 两个方面, 证明了 ChannelLink 协议能够保证跨片状态迁移交易的原子性 (不允许) 或跨片状态迁移交易处理的一致性 (允许).

定理 1. 如果不允许跨片状态迁移部分完成, 则 ChannelLink 协议能够保证跨片状态迁移交易的原子性.

证明: ChannelLink 协议分为源分片状态迁出、片间状态传输与目的分片状态迁入这 3 个模块. 在源分片状态迁出模块中, 若用户 v_i 发出的链下状态通道建立交易或通道状态更新交易不正确, 其无法建立链下状态通道或获得正确的状态迁出证明, 系统相关分片或通道状态均没有更新. 当用户 v_i 发出的交易正确时, 其才能完成源分片状态以及对应链下状态通道的状态更新.

由于 CL 网络与跨片状态路由通道相关分片均是安全的, 当用户 v_i 是诚实节点并且 CL 网络中存在满足状态迁移条件的路由路径时, 用户 v_i 的迁移状态相关交易一定会通过片间状态传输与目的分片状态迁入模块, 被目的分片打包至区块中, 完成状态迁入操作, 保证了跨片状态迁移交易的原子性. 当 CL 网络长时间无法完成状态迁移时, 用户 v_i 可将原始状态迁出证明上链, 并附带通道所有节点签名, 从而将源分片锁定状态解锁或发往目的分片完成状态迁移, 也保证了跨片状态迁移交易的原子性. 当用户 v_i 是恶意节点时, 其构造的状态迁移交易无法通过 CL 网络与相关分片验证, 不会影响相关状态的正确更新.

综上所述, 当没有跨片状态迁移交易部分完成时, ChannelLink 协议能够保证跨片状态迁移交易的原子性.

定理 2. 如果允许跨片状态迁移部分完成, 则 ChannelLink 协议能够保证跨片状态迁移交易处理的一致性.

证明: ChannelLink 的片间状态传输模块允许用户部分完成跨片状态迁移操作, 即允许部分路由路径失败且各路由路径独立完成状态迁移过程, 从而避免异常情况下多条路径同时执行造成的状态锁定时间过长的问题. 由于允许跨片状态迁移部分完成, 部分路由路径就存在路由锁定状态解锁等回滚操作. 由于 CL 网络以及相关分片都是安全的, 那么基于相同的状态更新与操作回滚规则, 链上分片系统与 CL 网络即可对每笔状态迁移交易的处理达成共识, 从而完成跨片交易的一致处理, 保证了其一致性.

5.3 River 算法收敛性与复杂度

收敛性方面, 葛继科等人[30]总结了遗传算法收敛性的常见分析方法, 明确指出遗传算法在设置得当时能够收敛至全局最优解. River 算法属于一种遗传算法, 故能够保证收敛. 然而, 在实际运行过程中, 为了保证 River 算法能够在有限时间内结束, 本文与传统遗传算法类似, 定义了最大初始种群个体数量、最大迭代次数与迭代终止条

件, 并且通过扩大初始种群中个体在搜索空间中的分布范围来优化搜索效果.

复杂度方面, 在 ChannelLink 链下状态通道与闪电网络[9]等链下通道网络中, 节点数量远小于边的数量 (即 $N << |E|$), 详见第 6.1 节. 因此, River 算法的路径搜索基础算法选择时间复杂度为 $O(N^2)$ 的 Dijkstra 算法, 而没有选择时间复杂度为 $O(N \times |E|)$ 的 Bellman-Ford 算法. 路径搜索是 River 算法运行中时间复杂度最高的模块. 其中, 备选路径集合生成需要调用 $K+M$ 次 Dijkstra 算法; 若每轮迭代过程中每个个体都进行节点变异操作, 则首轮迭代需要调用 $count'$ 次 ($count' = \min\left\{MAXI, \sum_{K'=1}^{K} C_{K+M}^{K'}\right\} \times R$), 之后每轮迭代需要调用 R^2 次. 假设 River 算法每次迭代次数都达到最大迭代次数, 则 River 算法的时间复杂度上限为 $O(Counts \times N^2)$, 其中, $Counts$ 的取值如公式 (18) 所示. River 算法实际运行过程中复杂度将低于该上限, 关于其运行时间, 将在第 6 节中分析讨论.

$$Counts = K + M + R \times \left[\min\left\{MAXI, \sum_{K'=1}^{K} C_{K+M}^{K'}\right\} + (T-1) \times R\right] \tag{18}$$

5.4 链上分片系统集成

ChannelLink 协议面向通用链上分片系统设计, 是对分片系统原有跨片状态迁移协议的一种扩展. 协议模块可以依据不同分片系统接口进行适当修改后加载使用. 在协议部署时, ChannelLink 协议采用“按需部署”的原则, 在各分片独立部署. 各分片共识节点与用户客户端“按需加载” ChannelLink 协议相关操作与验证模块. 为了进一步降低 ChannelLink 协议相关验证在分片共识过程的开销, 链上分片系统的 ChannelLink 模块仅用于本分片相关状态验证, 而不与其他分片通过 ChannelLink 协议进行直接交互, 从而降低不同分片及其链下状态通道状态更新过程的耦合程度. 分片系统运行期间, 系统用户可以根据其交易需求, 选择是否使用 ChannelLink 协议, 以及在哪些分片建立链下状态通道 (详见第 5.1 节), 在提升跨片状态迁移效率的同时, 降低迁移开销.

6 实验分析

6.1 实验设置

为了验证本文提出的 CL 协议的性能优化效果, 我们基于 Python 开发了一个区块链链下状态通道网络框架. 该框架内包含 CL 网络与修改的 BrokerChain 链上分片系统, 并在相同的环境下对分片系统吞吐量、交易确认延迟等指标进行测试. 与此同时, 本文在该框架中实现了 River 以及对比算法, 并从多个维度进行了对比评估.

基础设置. 本文将原型系统部署在一台 H3C R4900 G3 服务器上. 该服务器使用 Intel (R) Xeon (R) Gold 6230 @2.10 GHz 的 CPU, 内存为 256 GB, 系统版本为 CentOS 7.9 (2009). 单次实验的内存使用上限为 8 GB, 每种实验配置取重复运行 10 次的平均值, 从而减少交易随机性所带来的误差. 为了观察算法运行时的稳定情况, 本文将每次实验过程划分为 10 个时期 (epoch), 每个时期结束时统计路由算法各指标情况 (本节中表格数据可能因四舍五入而与正文数据有所差异, 但二者均使用精度更高的原始数据计算得出). 与大多数链下通道路由算法相同[26,33], 由于无法获取链下通道网络的实际交易数据, 本框架也将独立处理每笔状态迁移交易, 且交易处理完成后不更新网络状态信息, 从而避免测试数据集中状态迁移交易执行顺序对于算法结果的影响.

数据集. 在对比集成 CL 协议前后, 分片系统性能优化效果的实验中, 本文使用比特币区块高度 70 万–71 万的交易作为测试数据集. 在评估 River 算法效果时, 本文基于闪电网络状态信息构造状态迁移交易测试的数据集. 为此, 本文首先基于 lnresearch/topology 项目[14]提供的数据集 (https://storage.googleapis.com/lnresearch/gossip-20220823.gsp.bz2), 获取数据集中所有闪电网络通道的短通道标识 (ShortChannelId) 列表. 随后, 基于该列表调用 1ML 网站[13]所提供的 API 接口, 获取截至 2022 年 10 月 24 日依旧活跃 (active) 的通道相信信息, 共计获得活跃通道 61 959 条, 节点 14 461 个. 当节点间存在多条通道时, 本文将各通道的容量相加, 固定交易费与比例交易费率取最大值, 并且移除收费政策错误设置的通道信息. 经过数据处理后, 本文共获得活跃通道 57 345 条, 节点 13 653 个.

由于无法获取闪电网络各链接中节点间的实际可用容量, 本文将每条通道内容量平均分给两个节点. 在交易的转移金额 (状态量) 方面, 每笔交易的转移金额设置范围为 $[10^4, \min\{0.99 \times balance_{v_i}, 2.4 \times 10^7\}]$ sats (在闪电网络

中, 绝大多数路由交易费约占转移金额的 1% 以下[31]. 为了保证节点拥有足够的余额支付路由交易费, 每笔转移金额的上限不应超过其余额的 99%. 上限设置为 2.4×10^7 sats, 是因为本次实验中节点平均余额为 24 194 212.34 sats, 其 99% 约为 2.4×10^7 sats). 为了降低交易随机性对实验结果的影响, 保证实验结果的稳定性, 本文基于处理后的链下通道网络信息, 随机选取相互连通的节点生成 1 000 万笔交易. 每次实验时, 从 1 000 万笔交易中随机选取 3 万笔交易.

其他参数设置. 在前述参数的基础上, 为了评估不同场景下 River 算法性能表现, 本文增加了交易费比例系数 α 和最低交易金额 min_value 两个参数. 其中, 交易费比例系数 α 是指将路由交易费收取标准提升为当前的 α 倍; 最低交易金额是指每笔交易最低的转移金额 (状态量). River 算法参数设置与链下通道网络拓扑、用户间交易特点息息相关. 本次测试中, River 算法输入参数的默认值分别为:

$$K=4,\ T=20,\ M=2,\ R=6,\ MAXI=100,\ prob_{\text{path\_cross}}=1.0,\ prob_{\text{cross\_front}}=0.5,\ prob_{\text{node\_mutation}}=1.0,$$
$$prob_{\text{value\_mutation}}=1.0,\ percent_{\text{value\_mutation}}=0.1,\ \alpha=1.0,\ min\_value=10^4\ \text{sats}.$$

6.2 对比算法

River 算法包含两个版本: River 和 River-Lite. 其中, River-Lite 算法在种群个体初始化时, 每个个体均包含 K 条路由路径. 为了对比算法性能, 本文将 River 算法与目前链下通道网络[9, 10]普遍采用的 3 种路由算法进行了对比, 3 种算法的基本思路如下.

(1) 朴素单路径路由算法 (naive single path, NSP). 本算法使用 Dijkstra 算法获取一条交易源节点与目的节点间的路由路径, 并使用该路径完成此次状态路由.

(2) 朴素多路径路由算法 (naive multipath, NM). 本算法使用 Dijkstra 算法获取交易源节点与目的节点间的所有路径, 并从中选取最大路由容量排序前 K 的路径作为路由路径 (当 K=1 时, 朴素多路径路由算法 (NM) 可以采用与改进单路径路由算法 (ISP) 相同的路由策略, 从而优化交易费占比与交易成功率方面的表现). 每条路径的路由状态量的赋值方法与 River 算法初始种群中个体路由状态量的复制方法相同.

(3) 改进单路径路由算法 (improved single path, ISP). 本算法使用 Dijkstra 算法获取交易源节点与目的节点间的所有路径, 并依次计算每条路径完成此次交易的交易费用, 最终选择费用最低的路径完成此次状态路由.

6.3 评估结果

6.3.1 分片系统性能优化效果

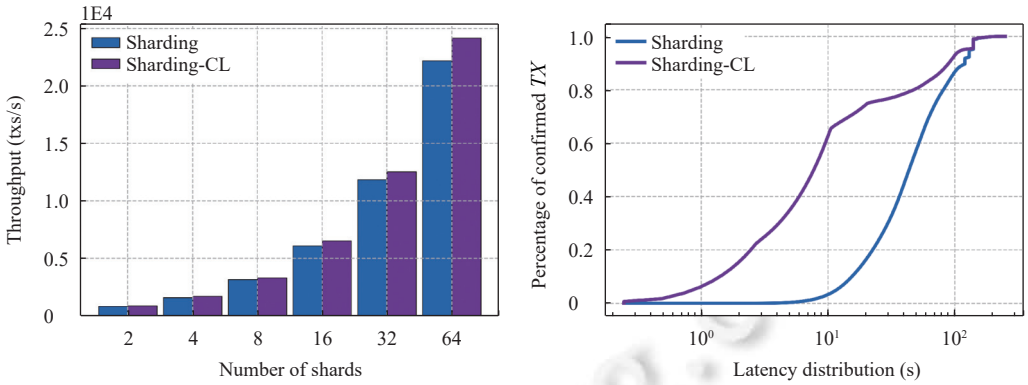
在本文所开发的框架中, 链上分片系统基于修改的 BrokerChain 协议实现, 其中, “热点用户”在每个分片均拥有做市商账户, 普通用户则将所有状态保存在同一分片内. 通过这种修改, 随着分片数量从 2 个逐渐上升至 64 个, 跨片交易比例分别为 3.81%、4.72%、5.11%、5.21%、5.23%、5.14%. 后文图 11 展示了集成 ChannelLink 协议前后, 分片系统的交易吞吐量与交易确认延迟. 其中, Sharding-CL 表示集成 ChannelLink 协议后, 区块链分片系统的实验结果. 此外, 根据第 6.1 节中的分析结果, Sharding-CL 实验中仅有“热点用户”使用 CL 协议完成跨片状态迁移, “热点用户”比例为 2%.

如图 11(a) 所示, 在 16 个分片、跨片交易比例 5.21% 的场景下, Sharding-CL 相较于 Sharding 吞吐量提升了 7.04%, 交易吞吐量达到了单一分片的 15.92 倍, 说明合理使用 CL 协议能够有效降低跨片交易的链上资源开销, 提升系统吞吐量. 与此同时, 随着分片数量的增加, 分片系统跨片交易比例、CL 网络状态迁移灵活度与效率逐渐升高, 集成 CL 协议也因此能够带来更大幅度的吞吐量提升. 交易确认延迟方面, 从图 11(b) 中可以看出, Sharding-CL 能够在链下通道网络中更高效地完成跨片状态迁移. 在相同实验条件下, Sharding-CL 相较于 Sharding 交易确认平均延迟降低 52.51%.

6.3.2 交易费占比

交易费占比是指成功执行交易的交易费之和占成功转移总金额 (状态量) 的比例, 其实验结果如图 12 所示. 表 2 展示了测试结束时各算法测试指标对比情况. 我们发现, River 算法的交易费占比始终保持最优. 与目前链下通道网络普遍采用的路由算法相比, River 算法的交易费占比分别降低了 68.43% (NSP)、74.45% (NM) 和 45.44% (ISP). 与此同时, River 和 River-Lite 算法具有更好的算法稳定性. 在相同实验条件下, 尽管测试数据中交易金额、

交易节点等信息不断变化, River 和 River-Lite 算法的交易费占比始终能够保持相对稳定, 进而也说明了本文提出的遗传算法在不同情况下都能够起到较好的优化效果.



(a) 吞吐量 (b) 交易确认延迟分布

图 11 集成 ChannelLink 协议前后分片系统性能对比

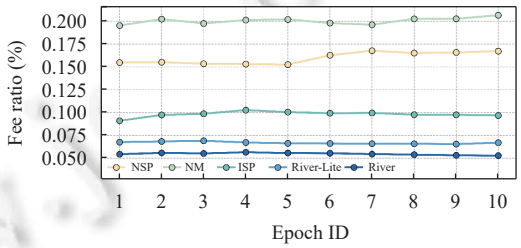


图 12 交易费占比

表 2 路由算法测试指标对比

指标	NSP	NM	ISP	River-Lite	River
交易费占比 (%)	0.167	0.207	0.097	0.067	0.053
交易成功率 (%)	35.988	40.745	40.024	44.293	47.862
平均路径搜索时间 (s)	0.000 1	0.088 4	0.088 7	1.863 6	2.642 7
平均路由路径长度 (通道数量)	3.596	3.605	3.618	6.445	4.559
平均路由路径数量	1	2.968	1	2.703	1.323

6.3.3 交易成功率

交易成功是指一次跨片状态迁移所有关联路径的状态迁移均成功完成. 交易成功率则等于成功完成的跨片状态迁移交易数量占跨片状态迁移交易总数量的比率. 如图 13 所示, River 算法的交易成功率始终保持最优, 相较于其他算法分别提升了 11.87% (NSP)、7.12% (NM)、7.83% (ISP) 和 3.57% (River-Lite). 通过对比 River 和 River-Lite, 我们发现在本次实验所使用的闪电网络数据集中, 节点间路由的最优方案往往位于路由路径数量较少的搜索空间 (详见第 6.3.6 节). 综合观察图 12 和图 13, 我们发现: (1) 朴素多路径路由算法 (NM) 与朴素单路径路由算法 (NSP) 相比, NM 算法更多的路由路径能够同时转移更多状态量, 从而显著提升交易成功率, 但是更多的路由路径也往往会花费更多的路由交易费用; (2) 朴素多路径路由算法 (NM) 与改进单路径路由算法 (ISP) 的候选路由路径集合相同. 但是, NM 算法交易成功率仅有略微提升, 交易费占比却大幅增加. 说明在闪电网络当前收费策略 (以及网络拓扑信息) 下[31], 节点间往往仅有 1 条可用容量充足且路由费用低的路由路径. 对于大多数网络拓扑中的边缘用户而言, 其资金往往仅存在于少量通道 ($\leqslant 2$) 中, 也只与 1–2 个路由节点相连. 盲目增加路由路径数量只会大量复用相同的路由通道, 增加固定交易费支出.

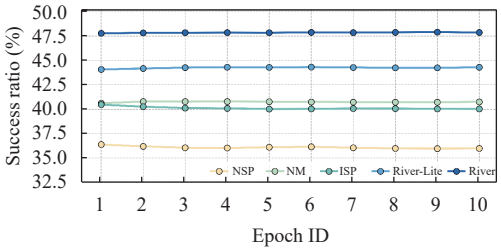


图 13 交易成功率

6.3.4 路径搜索时间

路径搜索时间是指发出状态迁移交易请求至搜索结果返回所需的时间 (单位: s). 如图 14 所示, 朴素单路径路由算法 (NSP) 仅需随机获取 1 条交易源节点与目的节点间的路由路径, 故搜索时间低于其他算法. 朴素多路径路由算法 (NM) 与改进单路径路由算法 (ISP) 的候选路由路径集合构造方法相同, 故分布情况亦基本相同. 由于改进单路径路由算法 (ISP) 的路由交易费求解与判断时间相较朴素多路径路由算法 (NM) 求解各路径路由状态量的计算时间稍长, 故平均路径搜索时间略高于朴素多路径路由算法.

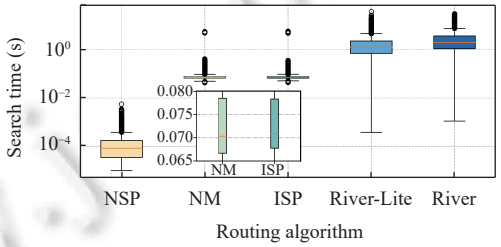


图 14 路径搜索时间

River 和 River-Lite 算法的路径搜索过程需要经过多轮迭代, 故相较于其他算法的路径搜索时间有所增加. 如表 2 所示, 在当前实验设置下, River 和 River-Lite 算法的平均路径搜索时间分别为 2.642 7 s 和 1.863 6 s. River 算法搜索空间更大, 故平均耗时略有增加. 尽管 River 算法平均每笔交易的路径搜索时间有所增长, 但是不同节点间交易的搜索过程可并行完成, 并且远低于当前链上分片系统的出块间隔 (约 10 s). 因此, River 算法的路径搜索性能能够满足本文提出的 CL 协议的需求.

6.3.5 路由路径长度

在链下通道网络中, 通道内节点通过网络通信的方式完成状态更新, 其更新速度与传统网络应用数据更新速度相当 (几十至几百 ms), 并且每条通道状态更新所需时间基本相同. 因此, 在相同网络环境下, 平均路由路径长度 (通道数量) 即可反映该路由算法状态路由所需的平均时间, 其结果如图 15 和表 2 所示. 当一笔交易通过多条路径完成时, 将其最长路径作为结果.

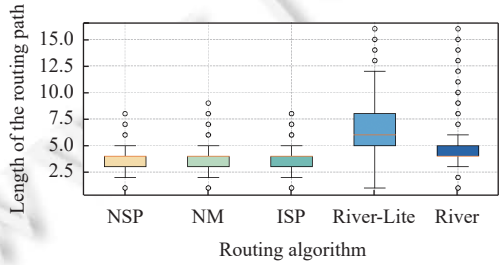


图 15 路由路径长度

朴素单路径路由算法 (NSP)、朴素多路径路由算法 (NM) 和改进单路径路由算法 (ISP) 均使用最短路径完成交易, 故路由路径长度的分布基本相同, 仅会因实验所选交易的不同而略有差别. River 和 River-Lite 算法通过节点变异与路径交叉能够产生更长且费用更低的路由路径, 故平均路由路径长度略有增长. 与朴素单路径路由算法 (NSP) 相比, River 和 River-Lite 算法平均路由路径长度增长约 0.963 与 2.849. 结合第 6.3.3 节中路径搜索时间的结果, 说明 River 算法在交易处理时间方面能够满足 CL 协议的需求.

6.3.6 路由路径数量

图 16 展示了 River、River-Lite 和 NM 这 3 种多路径路由算法路由路径数量的累积分布情况. 需要特别说明的是, 当一笔交易使用 1 条或 2 条路由路径均可成功执行的时候, 用户会优先选择交易费更低的路由方案. 通过观察 River 算法的数据, 我们发现约 75.972% 的交易仅通过 1 条路由路径完成, 从而进一步证明了第 6.3.2 节中的结论: 在当前闪电网络通道与交易金额的设置情况下, 绝大多数交易的最优路由方案位于路由路径数量等于 1 的搜索空间中. 未来我们将设计自适应的个体节点变异机制 (例如: 路由路径数量较少时, 设置多个变异节点等), 增加种群中个体单次迭代的搜索空间, 从而进一步优化 River 算法在交易费占比、交易成功率等方面的表现.

本组实验的最大路由路径数 $K=4$, 当交易源节点与目的节点间路由路径小于 K 时, 朴素多路径路由算法 (NM) 会返回节点间所有最短路径并全部使用. 通过观察图 16 和表 3 中 NM 算法的数据, 我们发现当前测试数据集中约 41.628% 的交易路由路径数量小于等于 3, 从而进一步验证了第 6.3.2 节中关于 NM 与 ISP 交易成功率与交易费占比的分析结果.

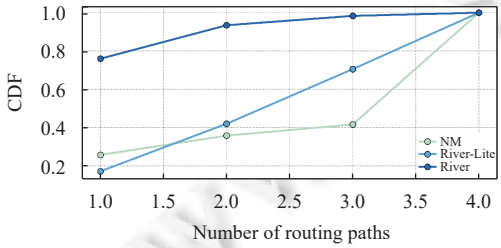


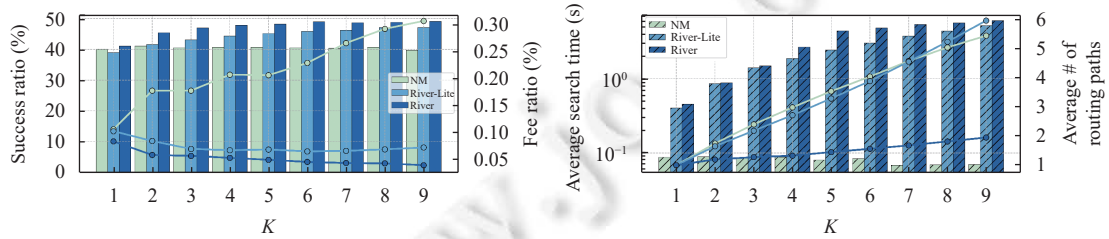
图 16 路由路径数量

表 3 交易路由路径数量占比

指标	交易路由路径数量	NM	River-Lite	River
使用不同路由路径数量成功执行的交易占成功执行交易总数的比例 (%)	1	25.749	17.189	75.972
	2	10.066	24.822	17.463
	3	5.813	28.530	4.898
	4	58.372	29.460	1.667
平均交易路由路径数量		2.968	2.703	1.323

6.3.7 最大路由路径数量

对于 River、River-Lite 与 NM 这 3 种多路径路由算法而言, 增加最大路由路径数量 K, 大多数情况下能够提升单次交易转移的状态量, 增大交易成功率, 但同时也可能增加交易费占比、搜索时间与平均路由路径数量. 为了更好地评估最大路由路径数量 K 对路由算法性能的影响, 我们将 K 从 1 增加至 9, 结果如图 17 所示.



(a) 交易成功率与交易费占比 (b) 路径搜索时间与路由路径数量

图 17 最大路由路径数量对 River 算法性能的影响

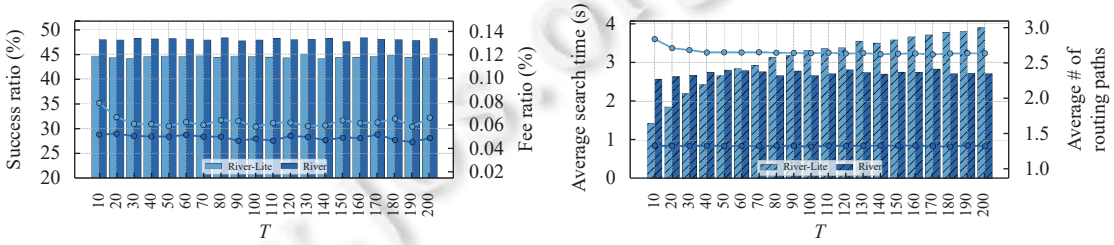
随着 K 的增加, River 算法初始种群的备选集合不断增加. 对于大额交易处理而言, River 算法能够选择更多的路由路径完成交易处理 (交易成功率与平均路由路径数量增加); 对于小额交易处理而言, 一条路径即可处理完成, 备选集合的扩大使得 River 算法能够找到更“经济”的路径完成交易处理 (交易费占比呈现下降趋势). 当 $K \geqslant 4$ 时,

River 算法的交易成功率、交易费占比基本保持相对稳定. 与此同时, 综合考虑 River-Lite 与 NM 算法表现以及平均路径搜索时间变化趋势, 本次实验默认情况下的最大路由路径数量 $K=4$.

River-Lite 算法与 River 算法相比, 初始种群中个体均包含 K 条路径, 搜索空间集中在 K 附近, 搜索空间较小. 因此, 随着 K 的增加, River-Lite 算法平均路由路径数量与搜索时间逐渐增加, 但搜索时间平均值低于 River 算法. 通过观察 NM 算法实验结果, 我们发现, NM 算法在 $K\geqslant 4$ 时, 平均路径搜索时间与交易成功率基本不变, 这也说明了在当前闪电网络拓扑下, 任意节点间平均最短路由路径数量有限. 随着 K 的增加, NM 算法交易路由的固定交易费不断增加, 从而使得交易费占比不断上升.

6.3.8 最大迭代次数

River 和 River-Lite 算法通过多轮遗传、变异、选择过程, 最终选择出路由费用最低的个体, 并依据其路由路径与路由状态量分配方案 Path 完成交易处理. 随着最大迭代次数 T 的增加, River 和 River-Lite 算法的搜索次数会不断增加, 有助于优化算法结果. 但是, 当最大迭代次数 T 已经足够算法收敛时, 持续增加最大迭代次数反而 (可能) 会持续增加路径搜索时间以及交易费占比的波动. 为确定适宜的最大迭代次数设置, 我们将 T 从 10 增加至 200 (间隔 10), 结果如图 18 所示.



(a) 交易成功率与交易费占比 (b) 路径搜索时间与路由路径数量

图 18 最大迭代次数对于 River 算法性能的影响

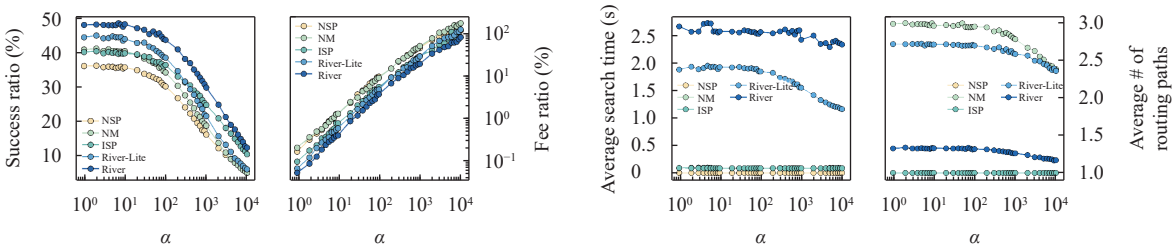
当 $T\geqslant 20$ 时, River 和 River-Lite 算法的交易成功率和平均路由路径数量均相对稳定 (数据波动范围如表 4 所示), 故本次实验默认情况下设置最大迭代次数 $T=20$. 随着 T 的增加, River 算法的平均路径搜索时间与交易费占比都能保持相对稳定, 说明绝大部分情况下 River 算法都能很快收敛, 找到其搜索空间中的局部/全局最优解. 通过观察 River-Lite 的结果, 我们发现, 随着迭代次数的增加 ($T\geqslant 20$), River-Lite 算法相较于 River 的性能指标波动更大, 这是因为 River-Lite 的初始搜索空间较小, 容易陷入局部最优解, 持续增加迭代次数只会“盲目”增加路径搜索时间. 未来在计算与通信资源充足的情况下, CL 协议用户可以通过扩大初始空间与每轮种群数量的方法, 减少此类情况的发生并进一步优化算法结果.

表 4 River 和 River-Lite 算法实验结果数据范围 ($T\geqslant 20$)

算法	交易成功率 (%)	交易费占比 (%)	平均路径搜索时间 (s)	平均路由路径数量
River	47.93 ± 0.413	0.049 3 ± 0.004	2.738 ± 0.095	1.322 ± 0.008
River-Lite	44.575 ± 0.475	0.062 9 ± 0.004	2.877 ± 1.014	2.657 ± 0.045

6.3.9 交易费比例系数

交易费低是闪电网络交易处理的主要特征与核心优势. 在本次使用的数据集中, 固定交易费大于 1 sat 的链接仅占约 9.452%, 比例交易费率大于 0.001 的链接也仅占约 6.028%, 大多数链接的收费收取标准与链上交易费占比存在 2–3 个数量级的差异. 在区块链分片系统中, 越来越多跨片状态迁移将通过链下状态通道完成. 随着网络规模的扩大以及交易数量的增加, 交易费收取标准也可能发生一定的变化. 为评估不同收费标准下的算法性能, 我们将交易费比例系数 α 从 1 逐渐上升至 10^4, 观察不同算法交易成功率、交易费占比等指标的表现, 结果如图 19 所示.



(a) 交易成功率与交易费占比　　(b) 路径搜索时间与路由路径数量

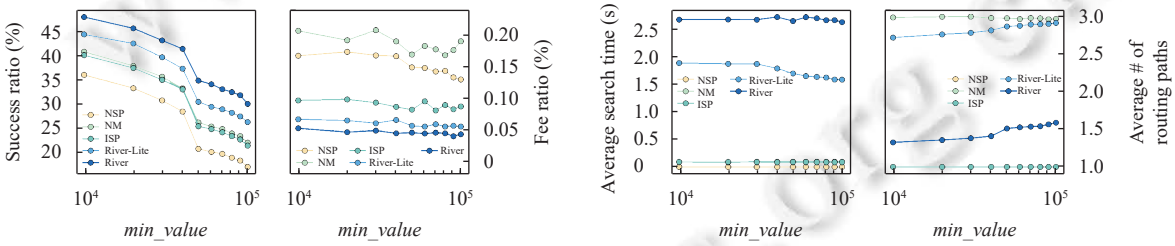
图 19　交易费比例系数对于 River 算法性能的影响

当 $\alpha \in [1,10]$ 时, 交易费收取比例仍远低于链上交易, 用户可用容量足以支撑状态迁移与路由费用支付. 但是, 由于收费标准的变化, River 和 River-Lite 算法的搜索空间和收敛速度可能存在一定的波动 (如图 19(b) 左图中 $\alpha \in [3,7]$). 当 $\alpha \geqslant 100$ 时, 随着交易费比例系数的升高, 我们发现 River-Lite 算法交易成功率逐渐低于改进单路径路由算法 (ISP), River 算法与 ISP 算法交易成功率差值也逐渐降低. 这是因为随着 α 的升高, 用户间往往仅有 1 条路径能够完成状态迁移 (如图 19(b) 右图), River 和 River-Lite 算法的搜索空间大幅缩小 (如图 19(b) 左图), 算法优化效果减弱.

当链下通道网络交易费收取比例较高或与链上跨片状态迁移费用相当的时候, 用户可以通过链上的方式完成跨片状态迁移. 与此同时, 在区块链分片系统中, 基于 CL 协议可以同时部署多个链下通道网络, 各网络之间独立运行. 当某个网络交易费过高时, 用户就会退出该网络并自发前往其他费用更低的网络.

6.3.10　最低交易金额

链下通道网络中的用户为了降低跨片状态迁移的总交易费, 将定期进行跨片状态聚合、分拆操作, 其交易频率与闪电网络用户相比将大幅降低, 但是单次转移状态量会有所上升. 本文通过调整单笔交易的最低成交金额 $min\_value \in [10^4, 10^5]$ sats, 统计不同算法的性能表现. 本次测试结果如图 20 所示.



(a) 交易成功率与交易费占比　　(b) 路径搜索时间与路由路径数量

图 20　最低交易金额对于 River 算法性能的影响

随着最低交易金额的上升, 由于当前网络设置没有充足容量进行状态路由, 各算法交易成功率均大幅下降. 未来在系统实际运行过程中, 为了提升交易路由的成功率, 链下通道网络中的路由节点将提供更多可用状态用于路由过程. 在交易费占比方面, 随着 *min_value* 的上升, 朴素单路径路由算法 (NSP) 与朴素多路径路由算法 (NM) 由于备选路径收费标准存在较大差异, 交易费占比波动相对较大. 其他算法交易费占比虽然有一定的波动, 但总体保持相对稳定.

在搜索空间大小与优化效果方面, 由于最低交易金额的增加, River 和 River-Lite 算法的搜索空间将会缩小, 其中 River-Lite 搜索空间缩小幅度更大 (图 20(b) 平均搜索时间下降更多). River 算法虽然搜索空间有所减小, 但是依旧能够通过更有效地使用更多备选路径 $Path_{\text{init}}$ 的方式提升算法性能.

7 总 结

本文首先通过对比现有跨片状态迁移协议, 指出链上资源开销、用户交易费用和迁移效率是当前跨片状态迁移协议性能优化的关键指标, 明确跨片状态迁移三阶段过程的紧密耦合是使得分片系统性能优化效果受限的主要原因, 状态迁移开销与迁移效率的协同优化问题是跨片协议所需解决的关键问题. 为此, 本文提出了一种基于链下状态通道的跨片状态迁移协议 ChannelLink, 并在此基础上提出了一种低开销链下通道路由算法 River. 实验结果表明, 本协议在 16 个分片、跨片交易比例 5.21% 的情况下, 分片系统吞吐量提升了 7.04%, 交易确认延迟降低了 52.51%, 跨片状态迁移开销下降了 45.44% 以上. 未来, 我们将进一步优化 ChannelLink 协议与区块链分片协议的协同方案, 提升 River 算法搜索效率, 降低分片系统跨片状态迁移开销、提升迁移效率.

References:

[1] Luu L, Narayanan V, Zheng CD, Baweja K, Gilbert S, Saxena P. A secure sharding protocol for open blockchains. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 17–30. [doi: 10.1145/2976749.2978389]

[2] Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Syta E, Ford B. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In: Proc. of the 2018 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2018. 583–598. [doi: 10.1109/SP.2018.000-5]

[3] Zamani M, Movahedi M, Raykova M. RapidChain: Scaling blockchain via full sharding. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 931–948. [doi: 10.1145/3243734.3243853]

[4] Huang HW, Peng XW, Zhan JZ, Zhang SY, Lin Y, Zheng ZB, Guo S. BrokerChain: A cross-shard blockchain protocol for account/balance-based state sharding. In: Proc. of the 2022 IEEE Conf. on Computer Communications. London: IEEE, 2022. 1968–1977. [doi: 10.1109/INFOCOM48880.2022.9796859]

[5] Wang JP, Wang H. Monoxide: Scale out blockchains with asynchronous consensus zones. In: Proc. of the 16th USENIX Symp. on Networked Systems Design and Implementation. Boston: USENIX Association, 2019. 95–112.

[6] Li CL, Huang HW, Zhao YT, Peng XW, Yang RJ, Zheng ZB, Guo S. Achieving scalability and load balance across blockchain shards for state sharding. In: Proc. of the 41st Int'l Symp. on Reliable Distributed Systems. Vienna: IEEE, 2022. 284–294. [doi: 10.1109/SRDS55811.2022.00034]

[7] Hong ZC, Guo S, Li P, Chen WH. Pyramid: A layered sharding blockchain system. In: Proc. of the 2022 IEEE Conf. on Computer Communications. Vancouver: IEEE, 2021. 1–10. [doi: 10.1109/INFOCOM42981.2021.9488747]

[8] Al-Bassam M, Sonnino A, Bano S, Hrycyszyn D, Danezis G. Chainspace: A sharded smart contracts platform. In: Proc. of the 25th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2018.

[9] Poon J, Dryja T. The Bitcoin lightning network: Scalable off-chain instant payments. 2016. https://lightning.network/lightning-network-paper.pdf

[10] BTAG. Raiden network. 2017. https://docs.raiden.network/

[11] Dong M, Liang QK, Li XZ, Liu JD. Celer network: Bring internet scale to every blockchain. arXiv:1810.00037, 2018.

[12] Bitcoin Community. Bitcoin core. 2023. https://github.com/bitcoin/bitcoin

[13] 1ML. Lightning network statistics. 2023. https://1ml.com/

[14] Lightning Network Research Community. Lightning network gossip. 2023. https://github.com/lnresearch/topology

[15] Bitcoin Community. Simplified payment verification. 2022. https://wiki.bitcoinsv.io/index.php/Simplified_Payment_Verification

[16] Kiayias A, Miller A, Zindros D. Non-interactive proofs of proof-of-work. In: Proc. of the 24th Int'l Conf. Kota Kinabalu: Springer, 2020. 505–522. [doi: 10.1007/978-3-030-51280-4_27]

[17] Bünz B, Kiffer L, Luu L, Zamani M. FlyClient: Super-light clients for cryptocurrencies. In: Proc. of the 2020 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2020. 928–946. [doi: 10.1109/SP40000.2020.00049]

[18] Bitcoin Community. Hash time locked contracts. 2021. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts

[19] Zakhary V, Agrawal D, El Abbadi A. Atomic commitment across blockchains. Proc. of the VLDB Endowment, 2020, 13(9): 1319–1331. [doi: 10.14778/3397230.3397231]

[20] Bentov I, Ji Y, Zhang F, Breidenbach L, Daian P, Juels A. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM, 2019. 1521–1538. [doi: 10.1145/3319535.3363221]

[21] Thomas S, Schwartz E. A protocol for interledger payments. 2023. https://www.thebitcoindesk.com/interledger.pdf

[22] Hearn M, Brown RG. Corda: A distributed ledger. 2019. https://www.r3.com/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf

[23] Osuntokun O. AMP: Atomic multi-path payments over lightning. 2018. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html

[24] Grunspan C, Pérez-Marco R. Ant routing algorithm for the lightning network. arXiv:1807.00151, 2018.

[25] Prihodko P, Sakhno K, Ostrovskiy A, Zhigulin S, Osuntokun O. Flare: An approach to routing in lightning network. 2016. https://pdfs.semanticscholar.org/4392/166a1194010c844ec915694fd5c56da94301.pdf

[26] Malavolta G, Moreno-Sanchez P, Kate A, Maffei M. SilentWhispers: Enforcing security and privacy in decentralized credit networks. In: Proc. of the 24th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2017.

[27] Sivaraman V, Venkatakrishnan SB, Ruan K, Negi P, Yang L, Mittal R, Fanti G, Alizadeh M. High throughput cryptocurrency routing in payment channel networks. In: Proc. of the 17th USENIX Symp. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2020. 777–796.

[28] Tsuchiya PF. The Landmark Hierarchy: A new hierarchy for routing in very large networks. In: Proc. of the ACM Symp. on Communications Architectures and Protocols. Stanford: ACM, 1988. 35–42. [doi: 10.1145/52324.52329]

[29] Ye YJ, Ren ZF, Luo XP, Zhang JJ, Wu WG. Garou: An efficient and secure off-blockchain multi-party payment hub. IEEE Trans. on Network and Service Management, 2021, 18(4): 4450–4461. [doi: 10.1109/TNSM.2021.3101808]

[30] Ge JK, Qiu YH, Wu CM, Pu GL. Summary of genetic algorithms research. Application Research of Computers, 2008, 25(10): 2911–2916 (in Chinese with English abstract). [doi: 10.3969/j.issn.1001-3695.2008.10.008]

[31] Chen YJ, Zhu XT, Yu YR, Cheng ZY. Empirical analysis of lightning network: Topology, evolution, and fees. Ran Jian Xue Bao/Journal of Software, 2022, 33(10): 3858–3873 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6380.htm [doi: 10.13328/j.cnki.jos.006380]

[32] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf

[33] Roos S, Moreno-Sanchez P, Kate A, Goldberg I. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In: Proc. of the 25th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2018.

附中文参考文献:

[30] 葛继科, 邱玉辉, 吴春明, 蒲国林. 遗传算法研究综述. 计算机应用研究, 2008, 25(10): 2911–2916. [doi: 10.3969/j.issn.1001-3695.2008.10.008]

[31] 陈艳姣, 朱笑天, 于永瑞, 程子英. 区块链闪电网络实证分析: 拓扑、发展和收费策略. 软件学报, 2022, 33(10): 3858–3873. http://www.jos.org.cn/1000-9825/6380.htm [doi: 10.13328/j.cnki.jos.006380]

贾林鹏(1995－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为区块链高通量技术.

孙毅(1979－), 男, 博士, 研究员, CCF 杰出会员, 主要研究领域为区块链, 社交视频.