软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(11):4953−4974 [doi: 10.13328/j.cnki.jos.007356] [CSTR: 32375.14.jos.007356]

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

区块链跨链协议 IBC 形式化分析*

魏秋阳[1,2,3,4], 赵旭峰[1,2,4], 朱雪阳[1,2,4], 张文辉[1,2,4], 卢奕函[1,2,3,4]

[1](基础软件与系统重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[3](国科大杭州高等研究院, 浙江 杭州 310024)

[4](中国科学院大学, 北京 100049)

通信作者: 朱雪阳, E-mail: zxy@ios.ac.cn

摘　要: 自从比特币诞生以来, 区块链技术在许多领域产生了重大的影响. 然而, 异构、孤立的区块链系统之间缺乏有效的通信机制, 限制了区块链生态的长远发展. 因此, 跨链技术迅速发展并成为了新的研究热点. 由于区块链的去中心化本质和跨链场景的复杂性, 跨链技术面临巨大的安全风险. IBC 协议是目前最广泛使用的跨链通信协议之一. 对 IBC 协议进行形式化分析, 以期帮助开发者更可靠地设计和实现跨链技术. 使用基于时序逻辑的规约语言 TLA+对 IBC 协议进行形式化建模, 并使用模型检测工具 TLC 验证 IBC 协议应满足的重要性质. 通过对验证结果深入分析, 发现一些影响数据包传输和代币转移正确性的重要问题, 并提出建议来消除相关安全风险. 这些问题已经向 IBC 开发者社区汇报, 其中大部分得到确认.

关键词: 区块链; 跨链; IBC 协议; 形式化分析; TLA+

中图法分类号: TP311

中文引用格式: 魏秋阳, 赵旭峰, 朱雪阳, 张文辉, 卢奕函. 区块链跨链协议IBC形式化分析. 软件学报, 2025, 36(11): 4953–4974. http://www.jos.org.cn/1000-9825/7356.htm

英文引用格式: Wei QY, Zhao XF, Zhu XY, Zhang WH, Lu YH. Formal Analysis of Cross-chain Protocol IBC. Ruan Jian Xue Bao/Journal of Software, 2025, 36(11): 4953–4974 (in Chinese). http://www.jos.org.cn/1000-9825/7356.htm

Formal Analysis of Cross-chain Protocol IBC

WEI Qiu-Yang[1,2,3,4], ZHAO Xu-Feng[1,2,4], ZHU Xue-Yang[1,2,4], ZHANG Wen-Hui[1,2,4], LU Yi-Han[1,2,3,4]

[1](Key Laboratory of System Software (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[3](Hangzhou Institute for Advanced Study, UCAS, Hangzhou 310024, China)

[4](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Since the advent of Bitcoin, blockchain technology has profoundly influenced numerous fields. However, the absence of effective communication mechanisms between heterogeneous and isolated blockchain systems has hindered the advancement and sustainable development of the blockchain ecosystem. In response, cross-chain technology has emerged as a rapidly evolving field and a focal point of research. The decentralized nature of blockchain, coupled with the complexity of cross-chain scenarios, introduces significant security challenges. This study proposes a formal analysis of the IBC (inter-blockchain communications) protocol, one of the most widely adopted cross-chain communication protocols, to assist developers in designing and implementing cross-chain technologies with enhanced security. The IBC protocol is formalized using TLA+, a temporal logic specification language, and its critical properties are verified through the model-checking tool TLC. An in-depth analysis of the verification results reveals several issues impacting the correctness of

* 基金项目: 国家自然科学基金 (62072443); 南方电网网络空间安全联合实验室资助 (037800KC23090002)
本文由“形式化方法与应用”专题特约编辑陈明帅研究员、田聪教授、熊英飞副教授推荐.
收稿时间: 2024-08-25; 修改时间: 2024-10-14; 采用时间: 2024-11-27; jos 在线出版时间: 2025-05-14
CNKI 网络首发时间: 2025-05-15

packet transmission and token transfer. Corresponding recommendations are proposed to mitigate these security risks. The findings have been reported to the IBC developer community, with most of them receiving acknowledgment.

Key words: blockchain; cross-chain; IBC (inter-blockchain communications) protocol; formal analysis; TLA+

区块链本质上是一种去中心化的分布式账本, 它由一系列基于密码学的数据块按时间顺序排列组成. 这些数据块包含交易信息和数据验证信息. 区块链通过密码学技术实现了一种安全和可信的网络[1], 所有网络中的节点共同维护区块链系统的一致和可信. 自中本聪[2]提出区块链概念以来, 区块链的涌现掀起了新的信息技术浪潮. 区块链吸引了大量来自工业界和学术界的关注, 并成为了研究和投资的热点. 例如, 比特币[2]和以太坊[3]发展迅速, 已经达到万亿美元级别的市值[4]. 区块链不仅仅是一些技术的简单整合, 它还作为一个全新的互联网概念, 给金融、政治、社会、科学等众多领域带来了巨大的变革和深远的影响[5]. 然而, 区块链独立的特性使得不同区块链系统之间缺乏有效的交互机制[6]. 区块链如同一个个“孤岛”, 用户无法直接从一个区块链系统向另一个区块链系统进行转账或发送数据. 不同区块链通常还采用不同设计理念和技术框架. 例如, 比特币被设计为一个基于 Proof of Work[7]共识机制的加密货币, 而以太坊则支持智能合约, 允许用户构建和部署去中心化应用程序, 并于 2022 年启用了 Proof of Stake[8]的共识机制. 区块链孤立而异构的现状阻碍了自身的进一步发展. 随着越来越多区块链的出现, 这个丰富多彩而支离破碎的生态愈发需要数据和功能的互操作性, 而这种迫切的需求最终促使了跨链技术的诞生.

跨链技术作为不同区块链之间的桥梁, 实现了资产和数据的交换功能. 不同的区块链可以通过跨链技术相互连通, 共同向更多的用户提供更便利和广泛的服务, 提高每个区块链个体的影响力和价值. 大部分跨链技术主要关注与公链有关的资产交换. 例如, Interledger[9]是一种支持在不同区块链之间转账的支付协议; BTC Relay 允许用户在以太坊去中心化应用中使用比特币进行支付[10]; Rainbow bridge 帮助用户将资产从以太坊迁移至 NERA 链中[11]. 这些跨链技术为不同加密货币交易以及同质化或非同质化资产交换提供了便利和原子性保证. 尽管面向公链的跨链技术解决了备受市场关注的资产跨链问题, 但它们并不支持更为通用的数据跨链传递. 而且创建和维护跨多条公链的去中心化应用十分困难. 因此, 面向多链的跨链技术应运而生[12].

作为提升区块链可拓展性的解决方案之一, 由多条区块链组成的多链拥有比单链更好的吞吐量和灵活性. 多链通常会提供可复用的数据、网络、共识、激励等机制, 以方便用户创建面向特定应用的区块链[12]. Cosmos[13]是目前市值最高的多链之一. Cosmos 默认实现了网络和共识层 CometBFT, 并提供 Cosmos SDK 来帮助用户定制应用层开发. 为了支持内部链之间的互操作性, 多链需要一种比资产交换更为通用的跨链技术. 这样的跨链技术被叫做跨链通信协议. Cosmos 通过 IBC (inter-blockchain communications)[14]协议来连接其自治的内部平行链.

跨链技术连接了孤立的区块链, 促进了区块链生态的进一步发展, 但也削弱了区块链系统自身的安全性. 实际上, 跨链项目已经频频成为攻击目标并造成了巨大的损失. 例如, 对跨链项目 Ronin bridge 的入侵是 Defi 历史上损失最大的攻击事件[4]. 根据统计, 仅 2022 年就有超过 10 亿美元的资产由于跨链项目的不恰当设计、实现或部署而被偷或冻结在链上[15]. 由于链上智能合约的漏洞, Poly Network 损失了大约 6 亿美元[16], 而 Wormhole 被偷超过 3 亿美元[17]. 这些攻击不仅给用户造成了巨大的损失, 还动摇了他们对于跨链技术的信任. 然而跨链技术涉及链上链下的复杂处理, 开发者很难通过人工审计的方式全面检查他们的设计和实现. 因此有必要研究如何更有效地验证协议的安全性和可靠性问题.

形式化分析已经在各种协议的研究中展现出了巨大的价值[18]. 一般是通过对协议及其应满足性质进行形式化建模, 再利用相应的验证工具分析协议行为是否满足所需性质, 进而发现协议的潜在问题. 总的来说, 形式验证技术可分为定理证明和模型检测两类. 前者基于演绎推理进行验证; 后者通过探索待验证系统的全状态空间来验证性质是否被满足, 验证过程可自动执行. 本文使用模型检测技术作为形式分析的后端支撑. IBC 协议是目前最广泛使用的跨链通信协议之一. 至今, IBC 协议支持 46 条链, 总市值超过 75 亿美元[19]. IBC 协议包括核心的传输、认证、有序层 (TAO 层) 和描述数据包在具体应用中数据结构和处理语义的应用层 (APP 层).

本文工作选择 IBC 协议为例来展示如何应用形式化方法来提高跨链技术的安全性和可靠性. 本文全面总结作者在 IBC 协议形式分析方面的工作. 针对 TAO 层的形式化分析的前期工作已经发表于文献 [20]. 限于篇幅, 本文对 TAO 层的介绍仅从文章自完整性角度进行概述 (具体细节参见文献 [20]), 而对新增的 APP 层分析及全面的实验评估进行详细阐述. 本文工作研究框架如图 1 所示, 主要贡献包括 4 部分.

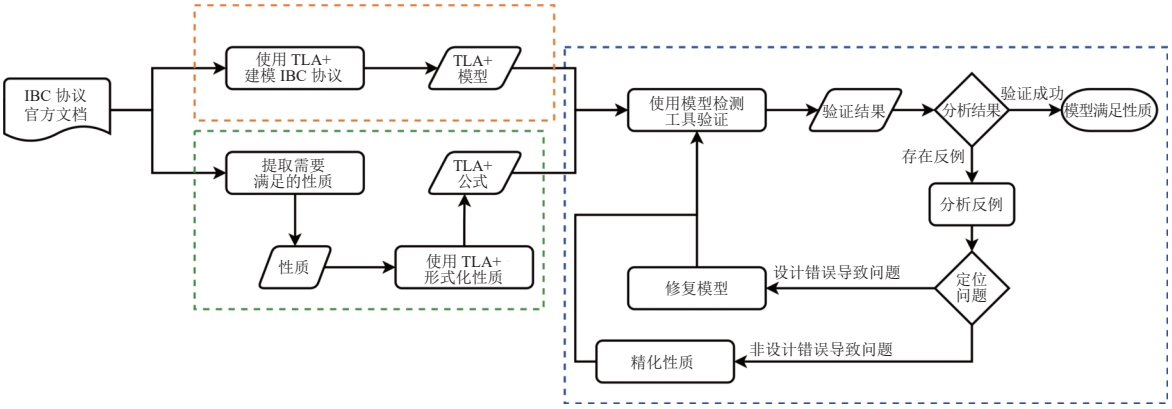


图 1 IBC 协议形式化分析研究框架

1) 总结了 IBC 协议进行跨链通信的整体框架. IBC 协议由一系列标准组成, 每个标准描述了协议一部分所涉及的数据结构、函数和行为等内容, 但是协议文档缺乏对整体框架和各部分之间关联的说明. 本文参考协议官方文档和两种协议实现, 提炼出区块链使用 IBC 协议进行跨链通信的整体框架, 并解释了 TAO 层和 APP 层主要实体的行为和两者之间的关联.

2) 从官方文档和协议实际使用中提取 IBC 协议应当满足的重要性质作为验证目标, 并对这些性质进行形式化说明以帮助开发者和用户更好地理解 IBC 协议. 本文将提取的性质扩展到 APP 层的典型应用代币转移, 并增加了对性质形式化为公式过程的介绍.

3) 在区块链自身是安全以及轻客户端能够正确验证中继提交的区块头和证明的假设下, 使用 TLA+[21]规约语言对 IBC 协议进行建模. 考虑到区块链系统中各参与者是不可信的特点, 本文主要关注链上模块和链下中继等实体的不确定行为对协议安全性和可靠性的影响. 同时, 本文的建模在保留了核心语义前提下进行了适当简化, 以兼顾实验规模的可行性和分析的准确性.

4) 基于所建模型, 使用模型检测工具 TLC 验证当前的 IBC 协议设计是否满足应有性质. 在前期工作的基础上, 本文总结了两类分析方法. 通过对验证结果的分析, 发现了一系列的重要问题, 例如无法完成的握手、被陷住的数据包和异常撤回的代币等. 其中, 异常撤回的代币问题揭示了 TAO 层数据包的错误处理对于 APP 层应用正确性的影响. 这些问题已经汇报给协议开发者社区, 大部分已被确认. 本文还进一步分析了这些问题的产生原因, 并提供修复建议.

本文第 1 节介绍跨链、多链以及协议形式化分析的相关研究现状. 第 2 节介绍 IBC 协议, 包括整体框架、TAO 层和 APP 层主要组成部分. 第 3 节介绍安全假设和 IBC 协议需要满足的性质. 第 4 节介绍 TLA+基础概念和本文的建模方法. 第 5、6 节介绍实验结果和分析建议, 以及总结全文.

1 相关工作

1.1 跨链和多链

尽管已经有一些实际部署的跨链方案在工业界取得了一定成就, 跨链在学术界仍是一个较新的研究话题. 目前关于跨链的研究主要关注对已有跨链技术的整理和分析. Cao 等人[22]和 Ou 等人[6]都对现有跨链技术进行了分类. Cao 等人同时根据已有的跨链模型提出了改进的原子交换跨链协议. Ou 等人根据采用的跨链技术进一步介绍了一些实际部署的跨链方案. Robinsom[10]根据使用场景将跨链通信协议分为价值交换、跨链信息和状态锚定这 3 类, 并简单介绍了主流跨链通信协议的大致工作流程以及存在的不足. Belchior 等人[12]则关注于区块链互操作性这个更为广泛的概念, 比较了在不同区块链系统中实现互操作性的技术差异以及具有的优势和局限性.

一些研究者对跨链交换机制的设计和分析感兴趣. 例如, Nehai 等人[23]形式化地描述了跨链交换问题. Pillai

等人[24]提出了一种基于数字签名和哈希时间锁的跨链资产交换协议, 并对部署有该协议的区块链系统进行了模拟和验证[25]. Herlihy 等人[26]在传统原子交易的基础上提出了跨链交易的新概念来解决对抗环境中分布式系统资产交换的问题. 同时, 一些研究者开始关注跨链技术的安全问题. Lee 等人[15]介绍了跨链桥的主要组成部分和处理流程, 然后分析了已经发生的专门针对跨链桥的攻击事件. Zhang 等人[27]记录了 3 类关于跨链桥的安全漏洞并提出了一系列对应的安全属性和识别模式. 基于这些模式, Zhang 等人设计了自动化工具来发现跨链桥中的安全风险并检测现实世界中的攻击.

相比于公链, 面向多链的跨链研究和实现更为缺乏. Polkadot[28]和 Cosmos 是目前市值最高的多链[12]. Polkadot 设计了 XCMP 协议[29]用于平行链之间直接通信, 但目前仍处于开发中. Cosmos 设计和实现了 IBC 协议来支持内部链之间的通信. 但是只要实现了协议要求的接口和功能, IBC 协议也可以应用于其他的区块链系统. 尽管 Cosmos 社区中有开发者利用形式化方法验证了自己的部分实现代码[30], 但据我们所知目前尚未有专门针对面向多链的跨链通信协议安全性和可靠性的研究.

1.2 协议的形式化分析

形式化方法被广泛应用于网络协议验证, 涵盖从抽象模型到具体实现的各个方面. Basin 等人[18]使用安全协议验证工具 Tamarin 验证了 5G AKA 协议的安全性质. Kobeissi 等人[31]将协议实现代码自动转换成可被验证的模型, 并利用这种方法实现和分析一种常见信号协议的变体. Zhang 等人[32]基于符号模型检测对 QUIC 握手协议进行安全性分析. 除了针对具体协议设计或实现的分析, 也有研究者利用形式化方法分析网络其他方面. 例如, Prabhu 等人[33]关注网络配置的验证, 通过结合等价类划分和模型检测技术高效验证了工业级别网络的配置策略. Zhang 等人[34]提出了一个概率分析框架来量化验证网络的可用性, 以帮助实际的网络管理.

TLA+[21]是一种高层次的规约语言, 通常用来规约分布式和并发系统, 也可以用于验证网络协议. 基于 TLA+的研究更多关注协议的功能性质而不是与加密有关的安全性质[18]. Yin 等人[35]利用 TLA+验证了一种支持崩溃恢复的原子广播协议的活性性质以及安全性质. Akhtar 等人[36]基于 TLA+验证了 Message Queue Telemetry Transport 协议的活性性质和安全性质. 目前将形式化方法应用到区块链领域协议分析的尝试还很有限. Kuaharenko 等人[37]和 Braihwaite 等人[38]都用 TLA+验证了区块链系统中的拜占庭容错共识协议的正确性. Kuaharenko 等人还对比了常见的拜占庭容错共识协议, 而 Braihwaite 等人则比较了 TLC 和 APALACHE 在验证时间上的差距. Grundmann 等人[39]尝试规约了闪电网络所用支付通道协议的抽象版本, 并验证了协议应当满足的功能和安全属性.

本文使用 TLA+对跨链通信协议 IBC 在设计层面的正确性进行形式化分析.

2 IBC 协议

IBC 协议官方文档[40]由一系列标准构成. 为了更好地理解协议, 本文还参考了两个使用不同编程语言的 IBC 协议实现[41,42]. 本节通过展示 IBC 协议的整体框架和主要内容 (TAO 层和 APP 层) 来介绍跨链通信是如何在 IBC 协议中实现的.

2.1 整体框架

IBC 协议分为 TAO 和 APP 两层. TAO 层作为基础设施层定义了通用的数据包传输功能; APP 层建立在 TAO 层之上, 定义了应用如何处理通过 TAO 层传输的数据包. 这样的分离设计使得 IBC 协议能够在提供通用的基础传输层的同时, 支持上层应用的模块化和可组合性. 两条区块链上的应用之间通过 IBC 协议进行跨链通信的整体框架如后文图 2 所示.

2.2 TAO 层

图 3 展示了在两条区块链上的模块之间使用 TAO 层进行数据包传输的整体框架. 轻客户端、连接、通道和模块是区块链上的关键实体, 而中继是链下可以同时访问源链和目标链的进程.

轻客户端是区块链的轻量化表示. 轻客户端用于高效验证一个特定消息是否存在于该区块链上, 而不储存所

有的历史消息或者执行交易. 轻客户端的概念并不是 IBC 特有的, 最早被提出用来实现比特币中的简易支付验证 (simplified payment verification)[10]. 简单来说, 轻客户端会记录对应区块链被验证的区块头序列. 为了证明一个消息存在于区块链上, 用户需要生成该消息的密码学证明. 然后轻客户端可以通过这个消息的哈希和被生成的证明计算根哈希, 并与自己记录的区块头中的根哈希比较. 密码学机制保障只有使用存在的消息和正确的证明才能计算出匹配的根哈希[42]. 这意味着使用轻客户端允许区块链之间在没有可信第三方的情况下交换消息.

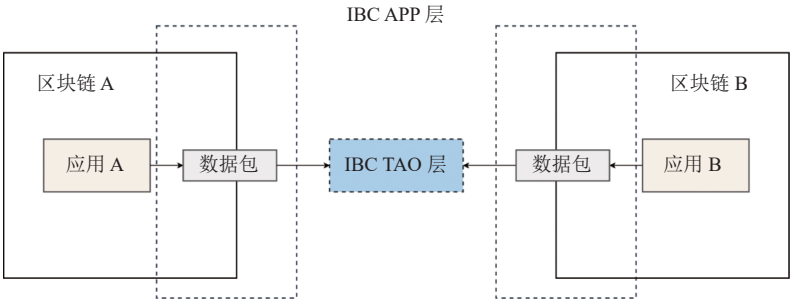


图 2　IBC 协议整体框架图

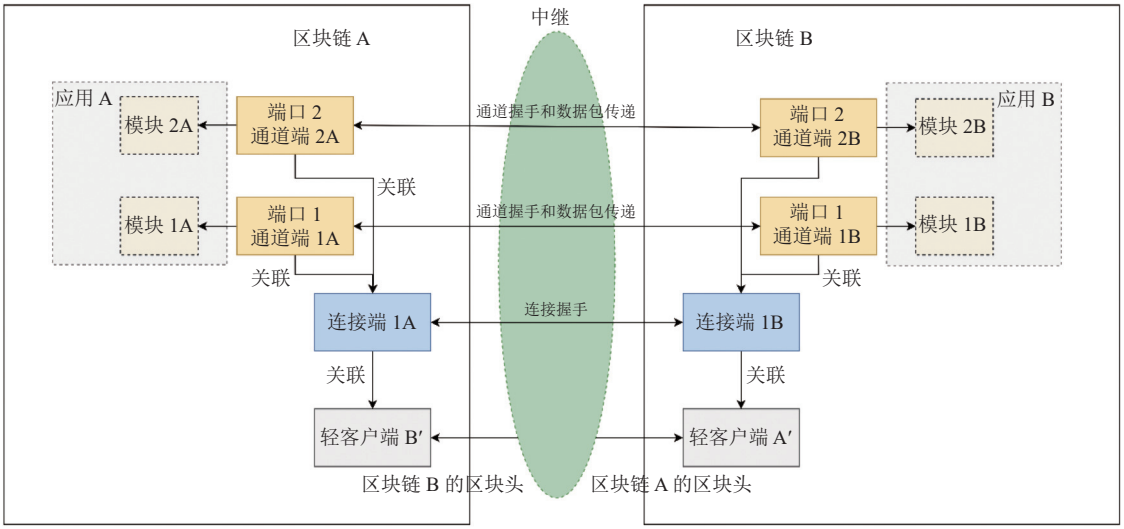


图 3　TAO 层框架图

区块链之间并不直接互相发送消息, 而是由不可信的链下中继在区块链之间中继消息. 中继会周期性检查需要中继的链上相关的数据结构, 例如连接端、通道端等. 当检查到需要中继的状态, 中继会按照协议要求, 在对应的链上构造数据报, 提交交易并执行.

IBC 协议将跨链通信相关的数据结构和处理函数抽象成 3 个层次: 连接、通道和数据包. 连接是由两个位于不同区块链上的连接端构成的抽象概念, 每个连接端会关联一个表示对方链的轻客户端. 连接利用轻客户端的功能来向通道提供跨链的状态验证. 通道则作为在两个位于不同链上的模块之间传输数据包的管道. 每个通道会关联一个特定的连接, 而一个连接可以关联多个通道. 通过这种方式, 不同的通道可以共享一个连接并均摊跨链验证的费用. 连接和通道通过握手建立, 参与握手的两条链需要对连接端或通道端状态、协商的版本号等达成一致.

模块是区块链应用的重要组成部分, 实现了其绝大部分事务逻辑. 模块会将需要传输的数据封装成数据包, 并通过 TAO 层发送给另一条区块链上的模块. 为了发送和接收数据包, 模块需要先绑定一个端口并创建一个通道. 实际上, 中继并不会直接将数据包发送给接收模块. IBC 处理模块和其他模块之间存在一个辅助的路由模块. 模块

需要调用 IBC 处理模块来发送数据报和数据包, 而路由模块负责接收外部数据包和管理模块. 数据报是特殊的外部数据, 用来触发对应的函数执行. 数据包是应用跨链传递信息的载体. 当一个数据包到达时, 路由模块会根据查询表调用对应的模块, 从而简化了中继的工作.

TAO 层协议主要包含与连接、通道和数据包相关的标准, 这些标准定义了跨链数据包传输所需要的数据结构和处理函数. 本文前期工作[20]对这 3 个标准有更详细介绍.

2.3 APP 层

目前 APP 层包括同质化代币转移 (fungible token transfer)、非同质化代币转移 (non-fungible token transfer)、链间账户和跨链验证等 9 个标准. 每个标准定义了一种应用中数据如何被打包成数据包以及在源链和目标链如何被解释. 代币转移是区块链领域常见而重要的需求. 有许多使用场景涉及到代币转移, 例如资产的代币化和金融项目的首次币发行等. 通过 IBC 协议, 用户可以在独立的区块链之间进行代币转移. 因此本文以代币转移作为代表说明对应用层的分析.

在区块链中代币通常被分为同质化代币和非同质化代币. 同质化代币指在同一种代币内可以互相交换和替代的代币, 通常用于支付和表示价值. 而非同质化代币的每个实例都有唯一的标识, 彼此不可互换, 一般用来代表艺术品、收藏品等具有独特性的资产. 在 IBC 协议 APP 层中有两个标准来分别定义同质化代币和非同质化代币转移[40]. 下文将进一步解释同质化代币和非同质化代币转移标准的主要内容.

2.3.1 同质化代币转移标准

同质化代币以数据包为载体进行跨链转移. 其对应数据包的主要结构包括: 代币的面额 (denomination), 相同面额的代币可以互相替代; 被转移的代币数量; 转移的发送者账户和接收者账户. 当接收者收到数据包后需要发送给发送者一个确认 (acknowledgement) 来告知转移结果. 同质化代币转移标准定义了两者确认类型: 成功和失败.

由于区块链是独立的系统, 每个区块链甚至应用都拥有自己的代币和发行机制. IBC 协议作为通用的跨链协议, 并不涉及也不会影响区块链上应用的经济体系. 因此转移代币并不会使代币真正从一个区块链上转移到另一个区块链. 同质化代币转移标准使用了双向锚定技术 (two-way peg)[43]. 当代币从源链发送到目标链时, 发送者需要先将自己需要转移的代币发送给源链上的特定的托管账户来锁定代币, 然后发送带有代币转移信息的数据包. 当目标链收到数据包时, 会在目标链上铸造对应的凭证 (voucher) 来代表收到的代币. 如果目标链希望将凭证转移回源链, 则需要先销毁目标链上的凭证, 然后发送数据包. 当源链收到数据包, 则可以从托管账户取回自己托管的代币.

同质化代币转移标准并没有区分代币和凭证, 两者都通过面额和数量来表示. 目标链收到来自其他链的代币后铸造的凭证, 同样可以作为一种代币被再次转移到其他区块链. 因此该标准通过面额来记录收到的代币的来源和转移路径, 代币转移经过的通道和端口标识符会以前缀的形式加在原面额上. 例如面额 PortB/ChannelB/CoinA 表示初始面额为 CoinA 的代币经过绑定端口 PortB 的通道 ChannelB 转移. 同质化代币转移标准将操作代币的链分为源链和目标链. 当链通过与代币面额最新前缀不同的端口和通道发送代币时, 该链为源链, 即发送的代币不来自即将接收代币的链. 此时链会将发送的代币转移到托管账户. 而接收代币的链为目标链, 会将收到代币的面额加上自身的端口和通道的标识符作为铸造的凭证的面额. 反之, 当链通过与代币面额最新前缀相同的端口和通道发送代币时, 该链为目标链, 即代币将被发送回原来的链. 此时链会销毁自己铸造的凭证. 而接收代币的链为源链, 会从托管账户取回自己先前托管的代币. 代币通过面额进行区分, 只有完全相同面额的代币, 即相同来源和转移路径的代币, 才可以互相替代.

同质化代币转移过程涉及到的主要函数如图 4 所示. 其中蓝色虚线框内为同质化代币转移标准定义函数, 橙色虚线框内为数据包标准定义函数. 用户可以通过调用 sendFungibleTokens 来发起代币转移, 该函数会封装数据包并通过 sendPacket 函数发送. 目标链会调用 recvPacket 函数来处理收到的数据包, 并调用 onRecvPacket 函数来接收代币. 用户可以通过函数 writeAcknowledgement 来发送确认. 对方链收到确认会调用函数 acknow-ledgePacket 确认数据包被接收, 并调用函数 onAcknowledgePacket 确认代币转移结果. 如果代币未能成功被接收, 则会调用函

数 refundToken 来取回被托管的代币. 当数据包超时未被接收时, 用户也可以调用 timeoutPacket 来判定数据包超时, 从而调用函数 refundToken 取回代币.

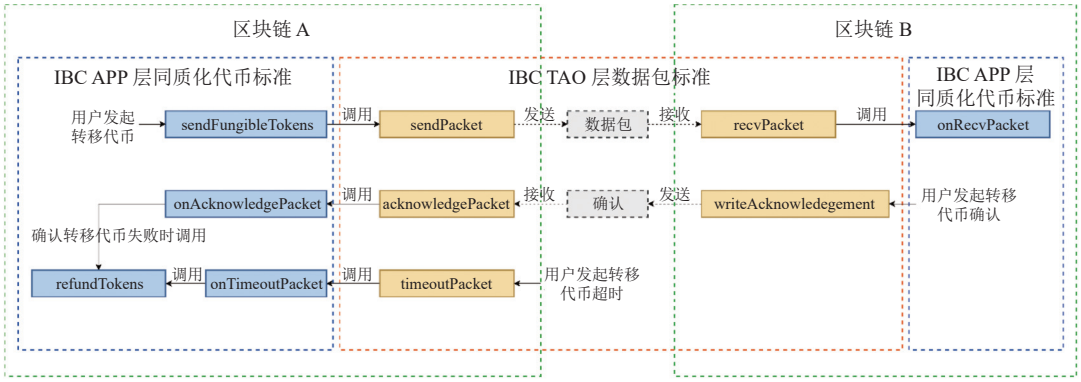


图 4　同质化代币转移涉及主要函数

图 5 进一步展示了链 A 上的用户 A 向链 B 上的用户 B 转移代币, 然后用户 B 再转移回用户 A 的状态迁移过程. 其中数据包传输的通道由链 A 上的通道端 PortA/ChannelA 和链 B 上的通道端 PortB/ChannelB 组成, EscrowA 和 EscrowB 分别为链 A 和链 B 上对应的托管账户. 函数 sendFungibleTokens 和 onRecvPacket 的参数分别为转移代币面额, 转移代币数量, 发送者和接收者. 状态①时用户 A 拥有数量为 1 的代币 CoinA. 链 A 通过执行函数 sendFungibleToken 向用户 B 发起代币转移, 此时数量为 1 的代币 CoinA 将转移给托管账户 (状态②). 当链 B 上执行函数 onRecvPacket 并成功接收代币时会铸造对应的凭证 (状态③). 凭证同样可以通过函数 sendFungibleTokens 进行跨链转移, 但此时会销毁凭证而不是转移给托管账户 (状态④). 当用户 A 收到自己先前转移的代币时, 会从托管账户取回自己托管的代币 (状态⑤). 在状态 3 时, 如果 B 链上的用户 B 向另一条链 C 上的用户 C 发送凭证. 此时凭证也会被当作一种代币, 因此代币面额会增加链 C 上的端口号和通道标识符 (状态⑥).

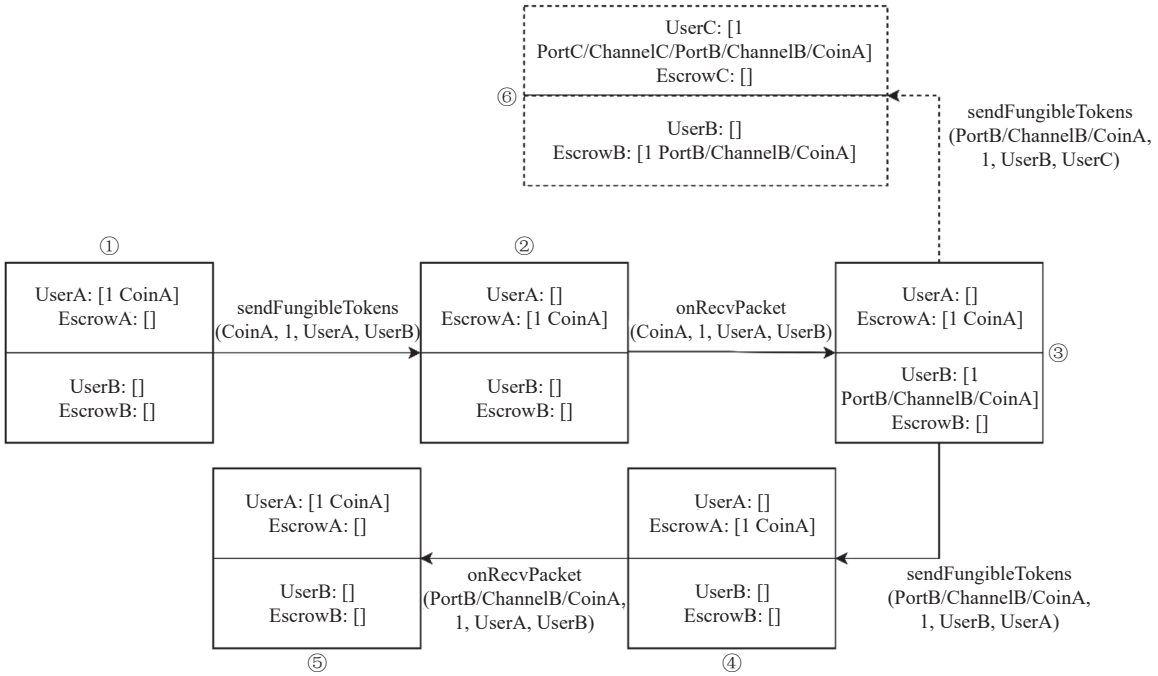


图 5　链 A 和链 B 状态迁移图

2.3.2 非同质化代币转移标准

非同质化代币同样以数据包为载体进行跨链转移. 对应数据包的主要结构包括: classId 用来表示转移的代币所属的类/收藏品/合约等, tokenId 则是类中代币的唯一标识.

尽管同质化代币和非同质化代币作为两个不同的代币类型有着不同的应用场景, 但是两种代币在转移行为上十分相似. 非同质化代币的 classId 用来区分代币的所属类别, 对应同质化代币的面额, 在转移过程中也有与其一样的修改前缀的过程. 但是同一类的非同质化代币不能相互替代, 因此需要通过 tokenId 来唯一标识. 同时, 多个非同质化代币也不能通过数量表示, 一次转移多个同类的非同质化代币只能通过列表来表示, 并逐一对每个代币进行操作, 有一个代币转移失败都会导致此次转移中的所有代币被退回. 非同质化代币转移同样通过双向锚定实现, 涉及的函数与同质化代币转移函数一一对应. 因此, 本文不再赘述其细节.

3 安全假设和性质

IBC 协议作为跨链协议实现了不同区块链之间的通信功能. 本文假设使用 IBC 协议的区块链自身是安全的, 主要关注协议定义的功能是否能够满足预期需求.

3.1 安全假设

IBC 协议涉及区块链、轻客户端、模块、中继等参与者和实体. 本文的分析基于以下安全假设.

1) 区块链的假设

IBC 协议向独立区块链提供了通信机制, 并确保链间通信的可靠性. IBC 可被应用于多种类型的区块链, 包括不同的共识机制和区块链架构. IBC 协议并不关心区块链内部错误, 例如共识机制的失效等. 因此, 本文假设使用 IBC 协议的区块链本身是安全的, 并且正确实现了 IBC 协议所需要的基本功能, 例如代币转移标准中需要的链内转移, 凭证铸造和销毁等功能.

2) 轻客户端的假设

IBC 使用的轻客户端可以结合区块头和密码学证明来验证区块链状态中特定路径是否包含或不包含特定值. 密码学原语保证了不能通过错误的值得到与区块头中根哈希匹配的哈希值. 为了记录对应区块链的正确区块头, 轻客户端还需要验证中继提交的区块头是否已经被目标链确认. 区块头的有效性验证反映了对应区块链和其共识机制的行为. 综上考虑, 本文假设轻客户端能够正确验证中继提交的区块头和证明.

3) 中继和模块的假设

链下中继进程承担了区块链间的消息传递, 并通过构造数据报触发相应的处理函数. 在 IBC 协议中, 中继是不被信任的. 模块是链上的应用程序, 与中继一样是协议相关处理函数的调用者. 由于应用的多样性、开发者可能的错误或疏忽以及潜在的恶意用户, 模块的行为是不确定的. 在对 IBC 协议的分析中, 本文主要关注在握手、数据包传输和代币转移过程中, 由链下中继和链上模块的不确定行为引起的安全问题. 本文希望通过探索尽可能多的模块和中继的行为来全面分析用户可能面临的安全风险.

3.2 性 质

IBC 协议包含一系列标准, 本文从官方文档和协议使用中提取和解释这些标准应满足的性质. 这些性质代表了用户对 IBC 协议标准定义的功能能够正确运作的合理预期. 以下概述 TAO 层应满足的主要性质, 并详细介绍 APP 层模块应满足性质.

3.2.1 连接、通道和数据包标准

连接和通道标准分别定义了连接握手和通道握手机制. 数据包标准定义了数据包传输相关的发送、接收、确认和超时机制. 模块需要依次完成连接握手和通道握手, 才能使用建立的通道传输数据包.

连接标准和通道标准都需满足与握手相关的 3 个性质.

性质 1. 只有正确的握手状态可以按顺序发生;

性质 2. 初始化的连接握手最终会完成;

性质 3. 如果握手完成, 两条链上的连接端或通道端最终会有一致的状态.

数据包标准需满足与传输相关的 4 个性质.

性质 1. 只有正确的数据包传递状态可以按顺序发生;

性质 2. 对于有序通道类型,先发送的数据包先接收;

性质 3. 一个数据包不能同时被确认接收和超时;

性质 4. 每一个发送的数据包,最终会被确认接收或超时.

上述每条性质含义的具体解释可参见本文前期工作[20].

3.2.2 同质化代币转移标准

性质 1. 只有正确的同质化代币转移状态可以按顺序发生, 即: 只有源链托管了转移的代币, 目标链才可以铸造对应的凭证或者源链撤回 (refund) 转移的代币; 只有目标链销毁了凭证, 源链才可以取回托管的代币或者目标链撤回转移的凭证.

性质 2. 每条链上的原生同质化代币数量恒定, 即: 代币转移并不实际增加或减少链上原有的代币, 只会铸造或销毁凭证. 因此在代币转移过程中, 链上原有的代币数量不受影响.

性质 3. 目标链铸造的凭证数量不超过源链对应代币被托管的数量, 并且两者最终会相等, 即: 双向锚定过程中, 源链需要先托管转移的代币, 目标链再铸造对应的凭证. 因此凭证的数量不会超过源链托管代币的数量, 并且无论转移成功或失败, 两者数量最终会相等.

性质 4. 只有收到目标链代币转移失败的确认信息或者载有转移代币信息的数据包超时, 源链才可以撤回转移的代币: 只有数据包超时, 或者收到失败的确认信息, 源链才可以主动确认转移代币失败, 从而撤回代币.

3.2.3 非同质化代币转移标准

非同质化代币转移与同质化代币转移有相似的行为, 所以两者应满足的性质也相似. 非同质化代币转移标准的性质也有 4 条, 其中性质 3 与同质化代币转移标准的相同, 性质 1、2 和 4 与其类似, 只需要将性质描述中的同质化代币替换成非同质化代币即可. 因为每一个非同质化代币都是唯一的, 所以在转移过程中, 每个被托管非同质化代币和其凭证最终会一一对应.

4 形式化建模

本节介绍使用 TLA+对 IBC 协议及其应满足性质进行形式化的主要思想. 完整 TLA+代码已经开源[44].

4.1 TLA+和 TLC

TLA+是一种基于线性时序逻辑的规约语言. TLA+常用于对分布式和并发系统进行分析与验证. 在 TLA+中, 系统以状态机的形式表达, 系统的执行表示状态序列, 这样的序列叫做行为. 状态之间的迁移用带撇号的变量 (代表新状态) 和不带撇号的变量 (代表当前状态) 构成的动作表示. 其中有一种踏步 (stutter) 的特殊动作, 表示所有变量维持原值不变. 在 TLA+中, 状态机和性质没有形式上的区别, 两者都通过时序逻辑公式来描述. 时序操作符主要包括两种类型: $\Box p$ 和 $\Diamond p$, 前者表示一个公式 p 在序列上的所有状态都为真, 而后者表示一个公式 p 至少在序列的一个状态为真. 此外, 有一种动作性质的特殊时序性质用于约束系统的动作. 动作性质可以表达为: $\Box[P(x', x)]\_x$, 表示要么变量 x 的值保持不变, 要么公式 $P(x', x)$ 为真, 该公式描述了变量 x 在新状态和当前状态取值的关系.

系统的规约通常表示为: Init∧□[Next]_vars∧Fairness. 其中 Init 表示系统所有可能的初始状态, 而 Next 定义了系统所有可能的动作的并集. vars 表示系统中所有变量组成的集合. 公式□[Next]_vars 表示系统要么按照 Next 规定的迁移关系执行, 要么所有变量维持不变. 通常本文需要通过 Fairness 来规定一些动作必须发生, 使系统能够按照预期执行.

TLA+与其模型检测工具通常用于消除设计层面的错误, 这些错误在代码层面很难被发现或纠正. TLC 是其中一个常见的模型检测工具, 可以验证安全和活性性质. TLC 会探索系统所有可能的状态序列, 并检查每个序列是否满足给定的性质. 如果所有序列都满足性质则会返回 Success. 如果 TLC 发现某个序列不满足性质, 则会返回该

状态序列作为反例. 本文使用 TLC 的 VS Code 插件版本[45].

以下通过一个简单的例子来展示使用 TLA+ 描述和验证程序的过程. 图 6 是一个简单的 C 程序, 该程序表示变量 i 从 0 一直加 1 直到 i 等于 10. 图 7 展示了使用 TLA+建模该程序的代码. 图 7 中, 第 1 行定义了变量 i, 并在第 2 行初始化为 0. 第 3–9 行定义了该系统可能的动作: 当 i 满足小于 10 时 i 增加 1, 当 i 满足大于或等于 10 时则保持不变. 第 10 行定义了对于系统行为的公平性约束: 如果 Next 表示的所有动作中有修改变量 i 的动作一直处于可行状态 (enabled), 那该动作最终会发生. Pro1 和 Pro2 是两条不同类型的性质, 使用了两种时序操作符. Pro1 表示变量 i 总是大于或等于 0, Pro2 表示变量 i 最终会等于 10. 因为 Pro1 描述的性质对于系统每个状态都成立, 所以 Pro1 为真. 同时因为公平性约束确保系统不会一直处于踏步动作使得变量 i 的值不变, 也就是说当 i 小于 10 时, i 增加 1 的动作一直是可行的并最终会发生, 所以 i 最终会等于 10, Pro2 也为真.

```
1  void main(){
2      int i = 0;
3      while (i < 10){
4          i++;
5      }
6  }
```

图 6 简单 C 程序

```
1   VARIABLE i
2   Init == /\ i = 0
3   Next ==
4       \/
5           /\ i <10
6           /\ i' = i + 1
7       \/
8           /\ i >= 10
9           /\ UNCHANGED i
10  Fairness == /\ WF_i(Next)
11  Spec == Init /\ [][Next]_i /\ Fairness
12  Pro1 == [] (i>=0)
13  Pro2 == <> (i = 10)
```

图 7 C 程序对应的 TLA+代码

使用 TLA+描述系统的代码会保存在以.tla 为后缀的文件中, 可以通过修改.cfg 为后缀的配置文件来指定 TLC 验证的模型和性质. 对于成立的性质, TLC 只会显示 Success 的结果, 反之 TLC 则会显示 Error, 并给出一个违背性质的反例.

4.2 建模选择

模型检测通过探索系统状态来检测性质的违背, 复杂的系统需要消耗大量的时间来验证. 因此需要通过合适的建模选择在保留系统核心语义的同时简化系统建模, 以便于模型检测工具在合理的时间内返回验证结果.

4.2.1 区块链和轻客户端抽象

基于使用 IBC 协议的区块链自身安全的假设, 本文在建模中不会深入区块链的实现细节. 在本文的模型中, 区块链只是按照标准存储 IBC 协议相关数据结构的状态机, 并提供了 IBC 协议要求的握手、数据包传输和代币转移相关的处理函数和基础功能. 与区块链类似, 轻客户端和证明的实现细节并不在 IBC 协议文档讨论范围内. 轻客户端和证明的作用是保证中继无法伪造提交的数据报和数据包. 本文的建模通过要求中继正确描述对方链的状态来确保中继无法伪造不存在的信息.

4.2.2 连接、通道、数据包和应用的建模分层

如第 2 节所述, 本文主要关注 TAO 层的连接、通道和数据包 3 部分标准和 APP 层两个代币转移标准. 这 5 部分标准定义了相对独立的数据结构和处理函数, 但彼此又相互关联. 连接向通道提供了状态验证的支持, 数据包通过通道进行传输, 而代币转移又以数据包为载体. 因此, 连接必须在通道建立、数据包传输和代币转移之前建立, 并且连接在建立后状态就不再改变. 具体来说, 当连接端已经创建但连接握手还未完成时, 模块可以利用该连接端乐观地创建一个通道端来开始通道握手. 但通道握手后续阶段都需要连接端状态已经为 OPEN, 因此通道握手只能在连接握手完成后才能完成. 通道是数据包传输的管道, 数据包传输的结果受通道状态影响. 模块可以在通道端创建但握手未完成时乐观地发送数据包, 但是数据包的接收和确认都要求通道端状态为 OPEN. 而数据包超时只要求通道端已创建, 不要求状态为 OPEN. 而代币转移要求连接、通道都已建立并在转移过程中保持 OPEN 状态. 因此本文从 4 个层次来建模 IBC 协议 TAO 层和 APP 层的标准: 连接、通道、数据包和代币转移. 除了连接自身层次, 其他层次建模中连接均保持在 OPEN 状态. 在通道自身和数据包层次建模中, 通道状态为不确定, 而代

币转移中通道保持在 OPEN 状态. 本文主要关注连接握手是否能成功完成, 在不同通道状态下发送的数据包是否有一个正确的结果以及代币能否正确地跨链转移.

4.2.3 数据报提交、处理函数执行和回调函数执行的封装和调用

数据报是特殊的外部数据用来触发对应的函数执行. 当数据报通过交易提交到路由模块时, 关联的处理函数会执行, 包括对应模块的回调函数 (如果有的话). 尽管数据报提交、处理函数和回调函数执行是异步的过程, 但错误的数据报会导致交易回退, 因此只有正确的数据报才会执行完处理函数和回调函数. 同时模型检测是一种时间开销较高的状态遍历方法, 完整建模数据报提交 (以消息队列的形式), 处理函数执行和回调函数执行的过程, 不仅时间代价高昂也没有必要. 因此, 本文在 TLA+模型中将这 3 部分封装成一个动作: 当输入参数 (代表数据报的内容) 正确时, 执行对应的处理函数和回调函数, 否则不改变任何变量.

在 IBC 协议的实际使用中, 用户、模块以及中继等实体通过交易、远程调用等方式使用协议相关的功能. 这个过程相当于以特定参数调用特定的处理函数. 因此本文不建模各实体调用函数的具体过程, 而是通过一个环境以任意参数调用任意处理函数的方式来模拟协议在实际使用中可能出现的情况. 如果在这样的模拟设置中协议模型满足性质, 则说明参与实体的错误行为不会影响协议正确结果. 反之如果发现了违背性质的行为, 本文会进一步分析造成问题的原因以及该问题是否会影响协议的实际使用.

4.2.4 数据包超时机制的模拟

模块通过比较数据包目标链的当前高度和数据包中指定的超时高度来判断数据包是否超时. 由于对区块链结构的抽象, 本文无法通过这种方式直接建模数据包超时机制. 然而, 本文的验证目的是协议是否能在任何场景正确运作, 所以使用一个数据结构来存储每个数据是否被判断为超时. 这个数据结构的值会在模型初始化时被指定, 并且覆盖所有的情况.

4.3 IBC 协议的 TLA+模型

本节从整体框架、数据结构和函数这 3 个方面介绍如何为 IBC 协议形式建模.

4.3.1 TLA+模型整体框架

在 TLA+中系统以模块的结构进行建模, 每个模块可以有自己的变量、操作符和性质, 可以通过 EXTEND 和 INSTANCE 关键字来引入其他模块的代码. 根据建模选择, IBC 协议被分为连接、通道、数据包和代币转移这 4 个层次, 而每个层次都被建模成两个模块: Chain 和 Environment. 模块 Chain 表示区块链, 主要包括连接、通道、数据包和代币转移相关的数据结构和处理函数. 模块 Environment 模拟区块链可能的执行环境: 通过 INSTANCE 关键字实例化两个 Chain 模块来表示两个使用 IBC 协议的区块链. 按照对数据报提交、处理函数执行和回调函数执行的封装和调用的建模选择, 模块 Environment 会用任意参数以任意顺序调用任意一个 Chain 模块的任意处理函数, 来模拟链上模块和链下中继所有可能的行为. 基于本文对区块链和轻客户端的安全假设, Environment 模块在每次调用其中一个 Chain 模块时, 都会附上如实描述另一个 Chain 模块状态的证明, 被调用的 Chain 模块可以通过这个证明来验证对方 Chain 模块的状态是否符合预期, 如果符合预期则执行处理函数.

4.3.2 TLA+模型数据结构

TLA+只支持有限的数据类型和结构, 因此需要对协议的数据结构做转化. TLA+支持布尔值、整数和字符常量这 3 种数据类型, 以及集合和函数两种数据结构. TLA+中的函数是从一个集合到另一个集合的映射关系. 数组可以理解为从自然数到数组值的函数, 结构体可以理解为从结构体成员到对应值的函数.

TAO 层的建模主要分为连接握手、通道握手和数据包传输这 3 个部分. 基于本文对轻客户端的安全假设, 本文并不建模轻客户端. 连接握手和通道握手相关的数据结构为连接端和通道端. 连接端和通道端是存储握手状态的结构体. 不同的连接端和通道端通过标识符进行索引, 并且在每一次初始化握手时会分配唯一的标识符. 为了能自动生成唯一的标识符并进行索引, 本文用数组建模连接端和通道端的存储, 并将数组索引作为标识符. 图 8 展示了连接端建模的数据结构. 数据包传输相关的数据结构包括记录已发送、接收、确认的数据包和下一个发送、接收、确认的数据包序号. 因为不同通道的数据包分开存储, 本文同样用数组建模了数据包存储并通过通道标识符索引. 而同一个通道的数据包只需要判断是否已包含, 因此本文使用集合建模.

```
connectionEnds:[{
 'state':string, // 连接端状态
 'connectionID':int, // 连接端标识符
 'clientID':string, // 连接端关联轻客户端标识符
 'counterpartyConnectionID':int, // 对方连接端标识符
 'counterpartyClientID':string, // 对方连接端关联轻客户端标识符
 'versions':Set(int) // 协商的版本
}]
```

图 8　连接端建模数据结构

在 TAO 层的数据结构的基础上, 代币转移还需要存储代币的数据结构. 如图 9 所示, 同质化代币转移账户被建模为一个函数, 表示每个用户持有的每种代币的数量, 其中 Seq(Set(string)) 表示由字符串集合中的元素任意排列组成的序列构成的集合. 该函数定义域中的元素为用户和面额构成的二元组, 值域则为对应的代币数量. 同质化代币的面额由通道端标识符, 端口标识符和初始面额拼接组成. 通道端标识符和端口标识符本身为整数类型, 但正如第 4.2.2 节中对建模分层的说明, 在代币转移标准中, 通道已经建立并保持不变, 因此可以将通道标识符映射到常量字符串. 非同质化代币账户则表示为从代币到其所有者的函数. 该函数定义域中的元素为非同质化代币 classId 和 tokenId 构成的二元组, 值域则为该代币的所有者.

```
// 同质化代币账户
FungibleTokenAccounts:[(string, Seq(Set(string))) -> int]
// 非同质化代币账户
NonFungibleTokenAccounts:[(Seq(Set(string)), int) -> string]
```

图 9　代币转移账户建模数据结构

除了 IBC 协议所规定的数据结构, 本文在建模过程还需要一些辅助性质描述的数据结构, 例如记录每个发送数据包或者转移代币的状态. 通过记录的状态本文可以描述模型的行为, 并验证其是否满足性质.

4.3.3　TLA+模型函数

TLA+与编程语言中的函数类似的概念是操作符. 操作符可以是对输入参数进行运算的表达式, 也可以是将变量从当前状态修改为新状态的动作. 根据建模选择, 整个处理函数和回调函数作为一个动作, 会在参数正确时执行所有操作. 在 TLA+中, 通过逻辑符号连接的多个操作符会在同一状态上执行操作, 因此后面的语句不能依赖之前语句的执行结果. 而 LET 表达式是一种特殊的表达式, 类似编程语言中的宏定义. 在 LET 表达式中, 后面的语句可以使用之前语句定义的表达式. 本文需要根据语义将标准中的伪代码函数转换为 TLA+的代码. 具体转换过程如下.

1) 简化与区块链和轻客户端具体实现相关的变量和语句, 例如区块高度, 时间戳等.

2) 将函数中的变量转换为合适的 TLA+的数据类型和结构.

3) 实现并封装伪代码中调用但没定义的外部函数. 标准并不会规定或展示这些函数的实现细节, 基于本文对区块链的安全假设, 本文会根据语义正确实现并封装这些函数.

4) 使用 LET 表达式提前定义函数中所有的临时变量, 使得 TLA+其余语句中可以直接使用.

5) 将条件语句调整至其他语句之前, 使得便于转换为逻辑条件的与或组合.

6) 添加辅助条件和变量以便验证和性质描述.

以下用一个例子来展示建模的过程. 函数 sendPacket(简化伪代码如图 10 所示) 对应的 TLA+代码如图 11 所示. 首先本文去除了区块高度和时间戳相关的语句 (图 10 第 4、11、12、13、14 行), 其中 timeoutHeight 本用于实现超时机制. 如第 4.2.4 节所述, 本文通过其他方式来验证超时情况带来的影响. 同时本文使用 TLA+的数据类型和结构实现了伪代码所需要的变量和数据. 其次, 本文实现并封装了伪代码中直接调用的对 channel、connection 和 sequence 的读函数以及对 sequence 和 commitment 的写函数 (图 10 第 6、9、15、16、17 行). 接下来本文通过 LET 表达式提前定义所需的临时变量 (图 10 第 6、9、15 行), 使得在后续 TLA+代码中可以直接使用 channel、connection 和 seq 等变量. 此外还调整了条件语句的顺序. 该例子所展示函数在条件成立时执行对应的所有操作.

最后, 本文增加了一个 seq<=MaxSeq 的条件来限制验证规模 (图 11 第 9 行) 和一个辅助变量 packetLog 以便于性质描述 (图 11 第 18 行).

```
1 function sendPacket(
2  sourcePort,
3  sourceChannelr,
4  timeoutHeight,
5  data):{
6  channel = getChannel(sourcePort, sourceChannel)
7  require(channel != null)
8  require(channel.state != CLOSED)
9  connection = getConnection(channel.connection)
10 require(connection != null)
11 require(timeoutHeight != 0)
12 client = getClient(connection.clientIdentifier)
13 latestClientHeight = client.latestClientHeight()
14 require(latestClientHeight < timeoutHeight)
15 sequence = getNextSequenceSend(sourcePort, sourceChannel)
16 setNextSequenceSend(sourcePort, sourceChannel, sequence+1)
17 setPacketCommitment(sourcePort, sourceChannel, sequence, hash(hash(data),timeoutHeight))
18 }
```

图 10　协议中函数 sendPacket 伪代码

```
1 HandleSendPacket(sourcePort, sourceChannel) ==
2  LET
3  channel == getChannelEnd(sourcePort, sourceChannel)
4  connection == getConnection(channel.connectionID)
5  seq == getNextSeqSend(sourcePort, sourceChannel)
6  log == LogEntry(sourcePort, sourceChannel, seq, "send")
7  commit == Commit(sourcePort, sourceChannel, seq)
8  IN
9 /\ seq <= MaxSeq
10 /\ channel /= nullChannelEnd
11 /\ channel.state /= 'CLOSED'
12 /\ connection /= nullConnectionEnd
13 /\ chainStore' = [chainStore EXCEPT
14        !.nextSequenceSend = [chainStore.nextSequenceSend
15              EXCEPT ![sourceChannel] = seq+1],
16        !.commitments = [chainStore.commitments
17              EXCEPT ![sourceChannel] = @ \union {commit}]]
18 /\ packetLog' = Append(packetLog, log)
```

图 11　函数 sendPacket 的 TLA+代码

总而言之, 本文的建模在保留协议本身语义和行为的前提下, 简化了数据结构和处理函数, 以便于模型检测工具验证.

4.4　使用 TLA+形式化性质

本文在第 3.2 节使用自然语言描述了 IBC 协议需要满足的性质. 然而, 这些描述可能存在歧义而不能用于验证. 所以需要将这些性质形式化成 TLA+公式, 从而可以用于模型检测工具进行验证. 为了将性质转化为 TLA+公式, 本文首先要将对性质的描述细化为若干个约束. 这些约束都形如某件事情在某个条件下会发生或不能发生, 便于转化为逻辑公式. 而逻辑公式中涉及到的条件, 行为和结果等都会对应到建模代码中变量的值, 从而最终表达为 TLA+公式.

本文所描述的性质对于系统的任意时刻都应成立. 这些性质涉及到的时序逻辑公式可以分为如下 4 类 (4 类公式可以组合).

1) P 总是成立, 对应公式$\Box P$. 例如同质化代币转移标准中的性质 2: 链上的原生代币数量总是等于初始值.

2) P 可成立无限多次, 对应公式$\Box\Diamond P$. 例如连接标准中的性质 2: 初始化的连接最终会建立.

3) 如果 P 成立则 Q 成立, 对应公式$\Box(P\rightarrow Q)$. 例如数据包标准中的性质 3: 如果一个数据包已经被确认接收,

则这个数据包没有超时, 反之亦然.

4) 如果 P 成立则最终 Q 会成立, 对应公式$\Box(P \rightarrow \Diamond Q)$. 例如数据包标准中的性质 4: 如果一个数据包已经发送, 则这个数据包最后会被接收或超时.

整个形式化过程受描述、建模和变量选择等诸多因素影响, 最后的公式结果并不唯一. 但不同的公式表达了相似的约束, 并可以通过选择不同的建模和变量互相转换. 例如连接标准中的性质 2: 初始化的连接握手最终会完成. 如果本文选择用连接端的状态作为标志, 则该约束可以表达为 connection.state = INIT → ◇ (connection.state = OPEN), 对应上述第 4 类公式. 但如果 connection 变量本就在初始化时才被创建, 则该约束可以直接被表达为◇ (connection.state = OPEN), 对应上述第 2 类公式. 这两个不同的公式实际表达了相同的约束.

表 1 给出了所有性质对应的约束和时序逻辑公式. 其中 connection′表示变量 connection 的下一个状态. 具体 TLA+公式可以查看本文的开源代码[44].

表 1 IBC 协议需要满足的性质及其对应的约束和逻辑公式

序号	所属标准	性质	约束	逻辑公式
1	连接	只有正确的握手状态可以按顺序发生	1. 如果连接端当前状态为UNINIT, 则连接端要么保持状态不变, 要么转为INIT或TRYOPEN状态 2. 如果连接端当前状态为INIT或TRYOPEN或OPEN, 则连接端要么保持状态不变, 要么转为OPEN状态	1. □ (connection.state = connection′.state or (connection.state = UNINIT => (connection′.state = INIT or connection′.state = TRYOPEN))) 2. □ (connection.state = connection′.state or ((connection.state = INIT or connection.state = TRYOPEN or connection.state = OPEN) => connection′.state = OPEN))
2	连接	初始化的连接握手最终会完成	初始化的连接握手最终会完成	□ (◇ (connection.state = OPEN))
3	连接	如果握手完成, 两条链上的连接端最终会有一致的状态	如果当前连接端状态为OPEN, 则与它的对方连接端标识符匹配的连接端状态最终为OPEN, 并且有相同的版本号	□ (connection.state = OPEN =>◇ (counterpartyConnection.state = OPEN and counterpartyConnection.versions = connection.versions))
4	通道	只有正确的握手状态可以按顺序发生	1. 如果通道端当前状态为UNINIT, 则通道端要么保持状态不变, 要么转为INIT或TRYOPEN状态 2. 如果通道端当前状态为INIT或TRYOPEN或OPEN, 则通道端要么保持状态不变, 要么转为OPEN或CLOSED状态 3. 如果通道端当前状态为CLOSED, 则通道端保持状态不变	1. □ (channel.state = channel′.state or (channel.state = UNINIT => (channel′.state = INIT or channel′.state = TRYOPEN))) 2. □ (channel.state = channel′.state or ((channel.state = INIT or channel.state = TRYOPEN or channel.state = OPEN) => (channel′.state = OPEN or channel′.state = CLOSED))) 3. □ (channel.state = CLOSED => channel′.state = CLOSED)
5	通道	如果没有初始化关闭握手, 初始化的打开握手最终会完成; 否则关闭握手最终会完成	1. 如果不允许关闭通道, 初始化的通道握手最终会完成 2. 如果允许关闭通道, 初始化的通道握手最终会关闭	1. □ ((not allowCloseChannel) => ◇ (channel.state = OPEN)) 2. □ (allowCloseChannel => ◇ (channel.state = CLOSED))
6	通道	如果握手完成, 两条链上的通道端最终会有一致的状态	如果当前通道端状态为OPEN, 则与它的对方通道端标识符和对方端口标识符匹配的通道端状态最终为OPEN, 并且有相同的版本号	□ (channel.state = OPEN =>◇ (counterpartyChannel.state = OPEN and counterpartyChannel.version = channel.version))
7	数据包	只有正确的数据包传递状态可以按顺序发生	1. 如果当前数据包被接收或超时, 则该数据包已经被发送 2. 如果当前数据包被确认, 则该数据包已经被接收	1. □ ((isReceived(packet) or isTimeout(packet)) => isSent(packet)) 2. □ (isAcknowledged(packet) => isReceived (packet))
8	数据包	对于有序通道类型, 先发送的数据包先接收	如果当前通道为有序通道, 则先被接收的数据包一定是先发送的	□ (((not channel.order = UNORDERED) and receiveLog(packet1).index < receiveLog(packet2).index) => packet1.sequence < packet2.sequence)

表 1 IBC 协议需要满足的性质及其对应的约束和逻辑公式 (续)

序号	所属标准	性质	约束	逻辑公式
9	数据包	一个数据包不能同时被确认接收和超时	1. 如果当前数据包已经被确认接收, 则该数据包不会超时 2. 如果当前数据包已经超时, 则该数据包不会被确认接收	1. □ (isAcknowledged(packet) => □ (not isTimeout(packet))) 2. □ (isTimeout(packet)) => □ (not isAcknowledged(packet)))
10	数据包	每一个发送的数据包, 最终会被确认接收或超时	每一个发送的数据包, 最终会被确认接收或超时	□ (isSent(packet) => ◇ (isAcknowledged(packet) or isTimeout(packet)))
11	同质化代币转移	只有正确的同质化代币转移状态可以按顺序发生	1. 如果转移的代币发生了退回, 则该代币已经被托管或凭证被销毁 2. 如果铸造了凭证, 则对方已经托管代币 3. 如果取回了转移的代币, 则对方已经销毁凭证	1. □ (isRefunded(token) => (isEscrowed(token) or isBurned(voucher))) 2. □ (isMinted(voucher) => isEscrowed(token)) 3. □ (isWithdrawn(token) => isBurned(voucher))
12	同质化代币转移	链上的原生同质化代币数量恒定	链上的原生同质化代币数量恒定	□ (Sum(nativeTokens) = initAmount)
13	同质化代币转移	目标链铸造的凭证数量不超过源链对应代币被托管的数量, 并且两者最终会相等	1. 目标链铸造的凭证数量不超过源链对应代币被托管的数量 2. 目标链铸造的凭证数量最终会等于源链对应代币被托管的数量	1. □ (Sum(vouchers) <= Sum(escrowTokens)) 2. □ (◇ Sum(vouchers) = Sum(escrowTokens))
14	同质化代币转移	只有收到目标链代币转移失败的确认信息或者载有转移代币信息的数据包超时, 源链才可以退回转移的代币	如果转移的代币发生了退回, 则对方接收代币失败或者数据包超时	□ (isRefunded(token) => (isReceivedFailure(token) or isTimeout(packet)))
15	非同质化代币转移	只有正确的非同质化代币转移状态可以按顺序发生	1. 如果转移的代币发生了退回, 则该代币已经被托管或凭证被销毁 2. 如果铸造了凭证, 则对方已经托管代币 3. 如果取回了转移的代币, 则对方已经销毁凭证	1. □ (isRefunded(token) => (isEscrowed(token) or isBurned(voucher))) 2. □ (isMinted(voucher) => isEscrowed(token)) 3. □ (isWithdrawn(token) => isBurned(voucher))
16	非同质化代币转移	链上的原生非同质化代币数量恒定	链上的原生非同质化代币数量恒定	□ (Sum(nativeTokens) = initAmount)
17	非同质化代币转移	目标链铸造的凭证总可以对应源链被托管的代币, 并且两者最终会一一对应	1. 每一个目标链铸造的凭证可以对应一个源链被托管的代币 2. 每一个源链被托管的代币最终可以对应一个目标链铸造的凭证	1. □ (Sum(voucher) <= Sum(escrowToken)) 2. □ (◇ Sum(voucher) = Sum(escrowToken))
18	非同质化代币转移	只有收到目标链代币转移失败的确认信息或者载有转移代币信息的数据包超时, 源链才可以退回转移的代币	如果转移的代币发生了退回, 则对方接收代币失败或者数据包超时	□ (isRefunded(token) => (isReceivedFailure(token) or isTimeout(packet)))

5 验证和分析

5.1 验证结果

我们使用模型检测工具 TLC 验证模型和性质. 目前, 我们验证了两条链之间的通信情况. 这样的设置已经足够表达 IBC 协议应当满足的性质, 并且可以通过类似的方式拓展到更多的链的情况. 连接端、通道端、握手中双方可以选择的版本号、数据包的个数和原生代币初始数量都是可以设置的参数. 它们都可能影响验证时间.

表 2 展示了性质的验证结果. 符号×表示模型检测工具发现该性质被违背并且给出了反例, 而符号√表示模型

检测工具验证该性质满足. 从验证结果来看, 初始化的连接握手和通道握手可能无法完成; 发送的数据包可能出现既无法被确认接收也无法超时的情况; 发起转移的代币可能异常被退回.

表 2　IBC 协议需要满足的性质及其验证结果

性质	验证结果
表1所列性质1、3、4、6–9、11–13、15–17	√
表1所列性质2、5、10、14、18	×

表 3 进一步展示了每条性质在不同参数配置下验证的状态数和所需时间. 连接部分可配置每条链可创建连接端和可选的版本号个数; 通道部分可配置每条链可创建通道端和可选的版本号个数; 数据包部分可配置每条链可创建通道端和可发送数据包个数; 代币转移部分可配置每条链上已创建通道端, 可发送数据包和初始代币个数. 实验环境为: 操作系统是 Ubuntu 22.04, CPU 是 256 核 AMD 霄龙 EPYC 9534 @2.45 GHz, 内存大小是 512 GB. TLC 可以通过 worker 参数设置多线程运行. 本文设置该参数为 auto 使 TLC 能够使用尽可能多的 CPU 资源. TLC 通过探索系统的可能状态来验证性质是否满足. 因此性质的验证时间主要受系统的状态数、每一条性质包含的公式和每个公式需要检查的状态序列长度影响. 从表 3 中可以看出系统状态数随配置参数增加而大幅增长. 例如对于数据包性质的验证中, 可发送的数据包从 1 增长到 2, 而总状态数扩大超过 100 倍. 因此本文目前只实验了有限的参数, 但通过对这些参数的验证已经发现 IBC 协议使用中各实体许多错误的行为.

表 3　不同参数配置下性质验证的状态数和时间

性质序号	所属标准	最多连接端个数	最多通道端个数	版本号个数	最多数据包个数	初始代币个数	验证结果	状态数	唯一状态数	验证时间(s)
1	连接	1	/	1	/	/	√	24	9	2.09
		1		2				856	253	4.10
		2		1				679	130	2.83
		2		2				83 291	10 788	5.26
2	连接	1	/	1	/	/	×	24	9	1.47
		1		2				856	253	3.65
		2		1				679	130	2.46
		2		2				75 149	10 307	4.61
3	连接	1	/	1	/	/	√	24	9	3.22
		1		2				856	253	4.38
		2		1				679	130	3.66
		2		2				83 921	10 788	17.62
4	通道	1	1	1	/	/	√	604	128	2.17
			1	2				5 542	404	3.40
			2	1				292 879	18 269	6.52
			2	2				10 535 506	190 544	13.26
5	通道	1	1	1	/	/	×	604	128	2.29
			1	2				5 542	404	8.06
			2	1				292 879	18 269	26.29
			2	2				10 535 506	190 544	92.35
6	通道	1	1	1	/	/	√	604	128	3.87
			1	2				5 542	404	12.24
			2	1				292 879	18 269	26.29
			2	2				10 535 506	190 544	198.89
7	数据包	1	1	/	1	/	√	16 401	2 741	5.98
					2			2 767 873	413 103	58.57
8	数据包	1	1	/	1	/	√	16 401	2 741	6.40
					2			2 767 873	413 103	40.59
9	数据包	1	1	/	1	/	√	16 401	2 741	5.57
					2			2 767 873	413 103	39.29

表 3　不同参数配置下性质验证的状态数和时间 (续)

性质序号	所属标准	最多连接端个数	最多通道端个数	版本号个数	最多数据包个数	初始代币个数	验证结果	状态数	唯一状态数	验证时间 (s)
10	数据包	1	1	/	1 2	/	×	8 531 2 267 441	1 712 364 252	4.75 65.34
11	同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	√	462 1 552 439 104 032	176 230 247 33 044	4.44 332.23 28.20
12	同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	√	462 1 552 439 104 032	176 230 247 33 044	3.69 81.31 11.65
13	同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	√	462 1 552 439 104 032	176 230 247 33 044	5.47 566.65 131.74
14	同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	×	462 780 403 104 032	176 130 022 33 044	2.98 72.56 16.86
15	非同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	√	462 4 335 329 99 976	176 595 849 31 268	5.01 2 815.98 48.52
16	非同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	√	462 4 335 329 99 976	176 595 849 31 268	4.58 1 041.01 18.98
17	非同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	√	462 4 335 329 99 976	176 595 849 31 268	6.18 8 876.97 243.06
18	非同质化代币转移	1	1 1 2	/	1 2 1	1 2 1	×	462 461 936 99 976	176 81 908 31 268	3.43 84.69 24.73

5.2　分析和建议

我们可以通过分析 TLC 给出的反例来确定导致问题的原因. 如果发现是由于协议设计错误导致的问题, 我们会根据可能的问题修复模型, 并再次验证. 如果验证成功, 则表明修复后的模型满足性质. 如果发现并不是由于协议错误设计导致的反例 (例如资源有限), 我们会尝试精化性质排除已知的反例来发现更多的问题. 通过重复修复模型或精化性质直到模型检测工具验证成功, 逐步确定 IBC 协议的设计在什么情况下才会满足性质. 图 1 蓝色线框部分展示了该过程.

本节通过 3 类问题具体介绍发现问题、分析反例和修复建议的过程. 其余问题可以参照本文前期工作[20]. 其中第 1 类问题无法完成的握手展示了本文逐步精化性质发现更多反例的过程, 第 2 类问题展示了如何通过分析反例来修复设计, 而第 3 类问题则展示了形式化对于代码开发的帮助.

5.2.1　无法完成的握手

连接和通道握手是数据包传递的前置条件, 发起的连接握手和通道打开握手本应当能够完成, 但是性质 2 和性质 5 的违背表明发起的握手可能由于缺乏可分配的标识符或者无法匹配而无法完成.

以连接握手为例, 无论在 OpenInit 阶段还是 OpenTry 阶段都需要创建一个新的连接端并分配唯一的标识符. 很容易想象有限的标识符只能用来创建有限的通道端, 也就是只有有限数量的发起的连接握手才能最终成功完成. 但是使得情况变得更糟的是连接握手在 OpenInit 和 OpenTry 阶段都需要创建新的连接端并占用标识符. 如果标识符已经被 OpenInit 阶段创建的连接端占用完, 那么就无法在 OpenTry 阶段创建新的连接端, 导致没有连接握手可以完成. 例如, 链 A 和链 B 都只能分配两个标识符. 如果链 A 和链 B 都执行两次 OpenInit, 两条链上都会创建两个 INIT 状态的连接端. 此时两条链都无法分配新的标识符, 也就无法执行 OpenTry 阶段来创建 TRYOPEN

状态的连接端继续与对方的连接握手. 所有创建的连接端都会冻结在 INIT 状态, 双方都无法继续握手或发起新的握手.

上述问题是由于每次创建连接端都需要分配新的标识符, 并且无法重新分配, 而可分配的标识符又是有限的. 然而, 考虑到跨链通信协议相比于传统通信协议拥有更大数据表示范围和更高的发起连接成本, 这样的反例在现实发生的可能性并不大. 因此, 需要让模型检测工具探索更多的状态来寻找危害更大的反例. 由于在 OpenInit 和 OpenTry 阶段创建连接端是完成握手必须的过程, 所以本文精化性质 2 来验证如果 OpenTry 已经创建 TRYOPEN 状态的连接端是否握手一定能完成. 结果表明并非如此. ConnOpenTry 函数只验证对方连接端确实处于 INIT 状态, 而不会检查是否已经创建过 TRYOPEN 状态的连接端与其配对. 因此重复执行 ConnOpenTry 函数 (由于并发竞争或者中继重复提交数据报) 会创建多个 TRYOPEN 状态的连接端, 但是只有其中一个会被对方模块执行 OpenAck 阶段时被指定为其对方端, 从而参与握手的后续阶段. 剩余的连接端会停留在 TRYOPEN 状态. 这些看起来处于握手过程中但实际无法完成的连接端会干扰用户的正常使用.

为了确定连接握手完成的充分条件, 本文进一步精化性质 2, 并成功验证如果 TRYOPEN 状态的连接端已经创建并且有配对的 INIT 状态的连接端, 则连接握手一定能完成. 并且如果连接握手完成, 显然存在配对的 TRYOPEN 状态的连接端和 INIT 状态的连接端, 所以这也是必要条件.

对于太多 INIT 状态连接端导致无法创建 TRYOPEN 状态的问题, 解决方法是允许利用已有的连接端来完成连接握手, 而不是总是创建新的连接端. 如果两条链都发起与对方连接握手并创建了 INIT 的状态的连接端, 则应该使其中一方执行 OpenAck 阶段来完成握手. 这样不需要创建新的连接端, 而按照现有的方案则会新建两个 TRYOPEN 状态的连接端来完成两个连接握手. 而重复创建 TRYOPEN 状态连接端从而干扰用户正常使用的问题则可以通过引入关闭机制或者标记这些有问题的连接端以警示用户来解决.

本文向开发者汇报这两个问题, 并给出了修复建议[46]. 本文发现第 2 个问题已经发生在实际的 IBC 协议使用中: 两个 TRYOPEN 状态的通道端只有与 INIT 状态通道端配对完成了握手. 而一个用户使用了未能完成握手的通道端发送数据包, 从而导致交易无法完成或回退[47]. 最终用户通过其他方式回退了该交易, 而开发者只是提醒用户却没有采取措施彻底解决该问题.

5.2.2 异常退回的代币转移

同质化代币和非同质化代币转移标准都定义了两种确认信息: 成功和失败. 当源链收到目标链的转移失败的确认信息或数据包超时可以撤回自己转移的代币. 然而, 表 2 所示的性质 14 和 18 的违背表示代币可能在其他情况被异常撤回. 这是因为协议在处理确认信息的函数 onAcknowledgePacket 里将撤回的条件定义为!ack.success, 也就是说, 当收到的确认信息不是成功时就撤回代币. 这样的设计意味着开发者默认读取到的确认信息只有成功和失败两个可能, 而忽略了存储确认信息的路径上可能存在默认的哨兵值 (sentinel value). 当源链收到确认信息的哨兵值时就会导致异常的代币撤回.

哨兵值的存在是因为确认信息的存储方式. IBC 协议主机状态机 (host state machine) 标准定义了协议相关数据结构的存储方式. 数据包的确认信息会存储在主机状态机的特定路径, 并通过端口号, 通道标识符和数据包序号进行检索. IBC 协议通过轻客户端来验证对方链提交的证明来判断对方链的消息是否如实反应了对方链的状态. 这些证明包括特定路径是否存在特定值和特定路径上是否不存在特定值两者类型. 同时轻客户端标准定义了轻客户端可以通过验证一个哨兵值 ABSENSE 的存在来支持那些不能提供不存在证明的区块链. 也就是说, 在一些使用 IBC 协议的区块链上, 存储数据包确认信息的路径上可能有默认的哨兵值. 这意味在目标链接收数据包写下确认信息之前, 恶意的中继可以提交关于对方链哨兵值的证明. 哨兵值是真实存在的, 因此可以通过轻客户端的验证, 并且不是成功的确认信息, 从而使得函数 onAcknowledgePacket 触发代币撤回. 这不仅会造成资源的浪费, 而且会干扰用户的正常使用.

为了进一步确定这个错误设计带来的现实影响, 本文分析 IBC 协议的 Go 语言实现和 Solidity 语言实现, 来研究在这两种实现中通过哨兵值的存在触发代币退回的可行性. Go 语言版本正确实现了函数 onAcknowledgePacket, 也就是在收到失败确认信息时才触发代币撤回. 这可能是其开发者意识到了协议设计的错误. Solidity 语言同协议

设计一样, 在收到的确认不是成功时触发代币退回. 尽管以太坊区块链可以提供不存在证明, 因此不需要哨兵值, 但是 Solidity 语言版本直接实现使用了以太坊区块链的 API 功能, 并且没有区分待验证的是存在证明还是不存在证明. 因此在目标链写下确认信息之前, 中继可以提供确认信息的不存在证明并通过源链的验证, 从而使得源链认为目标链没有返回成功的确认信息并撤回代币. Solidity 语言版本开发者确认了问题存在, 只是认为很难被刻意触发[48]. 这表示协议层面的错误设计确实影响了协议的实现.

上述问题是由于协议开发者并未准确实现其设计语义. 转移的代币应该在目标链接收失败时被撤回, 所以数 onAcknowledgePacket 里退回的条件应该定义为 ack.failure. 本文向协议开发者汇报了该问题和修复建议. 开发者确认了问题存在并需要修复[49].

5.2.3 其 他

IBC 协议标准是一个由开发者社区持续维护的项目. 标准中的一些代码可能随着版本的更新已经不再生效, 而这些冗余代码可能给读者理解造成困难. 普通的人工检查难以发现这些冗余的代码. 而通过形式化分析不仅可以发现还能验证这些代码已不再生效. 例如中继标准中的函数 pendingDatagrams 在判断条件中检查了 remoteEnd 是否不存在或者状态为 INIT, 给读者感觉 localEnd 对应的 remoteEnd 可能存在. 然而实际上 OpenInit 阶段创建的 INIT 状态的连接端并没有指定对方连接端, 所以 remoteEnd 值只能为 null. 这些冗余代码不仅不再生效还可能误导协议的用户和开发者. 通过模型检测工具, 可以将其表达为性质公式并验证其 remoteEnd 不会为 INIT 状态. 我们向开发者汇报了这个问题并被修复[50].

IBC 协议包含一系列标准. 这些标准存在关联, 开发者更新一个标准可能忽略了其他关联的标准. 例如, 路由模块标准和连接标准对于连接初始化的处理函数 connOpenInit 的参数定义不同. 在形式化分析协议标准时, 我们需要全面地理解所有涉及的标准. 我们发现了一些标准不一致的情况, 并向开发者汇报了这些问题, 目前都已被修复[51,52].

5.3 问题反馈

我们向协议开发者汇报了所发现的问题并得到积极反馈, 具体如表 4 所示. 开发者确认我们汇报的问题在协议层面存在, 但对于这些问题是否需要修复与我们存在一些分歧. 协议开发者认为, 序号 4 和 6 的问题应该通过应用开发者限制使用来避免, 而不需要在协议层面修复. 然而如果协议开发者不给予警示, 应用开发者难以意识到这些问题需要被针对性处理. 开发者还认为, 序号 2 和序号 5 的问题不需要修复, 但在 IBC 协议的实际使用已经出现过用户因为这些问题而交易出错, 但是开发者只是帮助用户撤回了交易而没有彻底修复问题. 由于实际使用中可分配标识符足够多, 因此开发者认为序号 1 的问题不需要修复. 其余问题都已被修复或者将修复.

表 4 向协议开发者汇报的问题及反馈

序号	所属类别	汇报问题	是否存在	是否需要修复
1	无法完成的握手	标识符可能在INIT阶段耗尽, 导致TRYOPEN阶段无法创建连接端和通道端	存在	不需要修复
2	无法完成的握手	重复执行TRYOPEN阶段会创建重复的连接端和通道端, 但只有其中一个可以完成握手	存在	不需要修复, 但该问题已在使用中发生
3	有序通道的不正确设计	特殊有序通道的数据包超时条件错误	存在	已修复
4	有序通道的不正确设计	在有序通道中, 数据包的超时操作未要求有序执行	存在	由应用开发者处理
5	通道异常状态的不正确处理	数据包乐观发送所使用的通道若在未匹配前关闭会导致数据包无法被接收和超时	存在	目前协议实现禁用了乐观发送, 但该问题已在使用中发生
6	通道异常状态的不正确处理	通道关闭后已被接收的数据包无法被确认	存在	由应用开发者处理
7	异常退回的代币	数据包确认信息存在成功和失败以外的状态, 使得未确认的代币转移可以被异常退回	存在	需要修复
8	代码冗余	标准中不再生效的代码会误导读者	存在	已修复
9	标准不一致	不同标准对同一函数或数据结构定义不同	存在	需要修复

6 总 结

本文对目前使用最为广泛的跨链协议 IBC 及其应满足的性质进行形式化建模和验证, 并对验证结果进行分析. 通过分析, 发现了一些可能会导致用户损失的关键问题, 特别是在数据包传输相关方面. 这些问题都已反馈到开发者社区, 其中大部分得到确认.

本文的分析成果也展示了形式化方法应用于跨链协议这样复杂场景的有效性. 所用到的分析和建模中的安全假设和建模选择也为未来研究提供了参考. 在未来工作中, 我们考虑将研究拓展到其他的跨链协议, 希望以本文的工作作为跨链协议设计开发的基础.

References:

[1] Zheng ZB, Xie SA, Dai HN, Chen XP, Wang HM. Blockchain challenges and opportunities: A survey. Int'l Journal of Web and Grid Services, 2018, 14(4): 352–375. [doi: 10.1504/IJWGS.2018.095647]

[2] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. Satoshi Nakamoto Institute, 2008.

[3] Wood G. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014, 151: 1–32.

[4] Xie TC, Zhang JH, Cheng ZR, Zhang F, Zhang YP, Jia YZ, Boneh D, Song D. zkBridge: Trustless cross-chain bridges made practical. In: Proc. of the 2022 ACM SIGSAC Conf. on Computer and Communications Security. Los Angeles: ACM, 2022. 3003–3017. [doi: 10.1145/3548606.3560652]

[5] Swan M. Blockchain: Blueprint for A New Economy. Sebastopol: O'Reilly Media, Inc., 2015.

[6] Ou W, Huang SY, Zheng JJ, Zhang QL, Zeng G, Han WB. An overview on cross-chain: Mechanism, platforms, challenges and advances. Computer Networks, 2022, 218: 109378. [doi: 10.1016/j.comnet.2022.109378]

[7] Gervais A, Karame GO, Wüst K, Glykantzis V, Ritzdorf H, Capkun S. On the security and performance of proof of work blockchains. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 3–16. [doi: 10.1145/2976749.2978341]

[8] Nguyen CT, Hoang DT, Nguyen DN, Niyato D, Nguyen HT, Dutkiewicz E. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. IEEE Access, 2019, 7: 85727–85745. [doi: 10.1109/ACCESS.2019.2925010]

[9] Thomas S, Schwartz E. A protocol for interledger payments. 2015. https://interledger.org/interledger.pdf

[10] Robinson P. Survey of crosschain communications protocols. Computer Networks, 2021, 200: 108488. [doi: 10.1016/j.comnet.2021.108488]

[11] Aurora Labs. Rainbow bridge. https://rainbowbridge.app/transfer

[12] Belchior R, Vasconcelos A, Guerreiro S, Correia M. A survey on blockchain interoperability: Past, present, and future trends. ACM Computing Surveys, 2022, 54(8): 168. [doi: 10.1145/3471140]

[13] Kwon J, Buchman E. Cosmos whitepaper. A Network of Distributed Ledgers, 2019, 27: 1–32.

[14] Goes C. The interblockchain communication protocol: An overview. arXiv:2006.15918, 2020.

[15] Lee SS, Murashkin A, Derka M, Gorzny J. SoK: Not quite water under the bridge: Review of cross-chain bridge hacks. In: Proc. of the 2023 IEEE Int'l Conf. on Blockchain and Cryptocurrency (ICBC). Dubai: IEEE, 2023. 1–14. [doi: 10.1109/ICBC56567.2023.10174993]

[16] Poly Network. Honour, exploit, and code: How we lost 610M dollar and got it back. 2021. https://medium.com/poly-network/honour-exploit-and-code-how-we-lost-610m-dollar-and-got-it-back-c4a7d0606267

[17] Wormhole. Wormhole incident report. 2022. https://wormholecrypto.medium.com/wormhole-incident-report-02-02-22-ad9b8f21eec6

[18] Basin D, Dreier J, Hirschi L, Radomirovic S, Sasse R, Stettler V. A formal analysis of 5G authentication. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 1383–1396. [doi: 10.1145/3243734.3243846]

[19] Cosmos Network. Cosmos market capitalization. 2024. https://cosmos.network/ecosystem/tokens

[20] Wei QY, Zhao XF, Zhu XY, Zhang WH. Formal analysis of IBC protocol. In: Proc. of the 31st Int'l Conf. on Network Protocols (ICNP). Reykjavik: IEEE, 2023. 1–11. [doi: 10.1109/ICNP59255.2023.10355573]

[21] Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Boston: Addison-Wesley Longman Publishing Co. Inc., 2002.

[22] Cao L, Song B. Blockchain cross-chain protocol and platform research and development. In: Proc. of the 2021 Int'l Conf. on Electronics, Circuits and Information Engineering (ECIE). Zhengzhou: IEEE, 2021. 264–269. [doi: 10.1109/ECIE52353.2021.00063]

[23] Nehaï Z, Bobot F, Tucci-Piergiovanni S, Delporte-Gallet C, Fauconnier H. A TLA+ formal proof of a cross-chain swap. In: Proc. of the 23rd Int'l Conf. on Distributed Computing and Networking. Delhi: ACM, 2022. 148–159. [doi: 10.1145/3491003.3491006]

[24] Pillai B, Biswas K, Hóu Z, Muthukkumarasamy V. Burn-to-Claim: An asset transfer protocol for blockchain interoperability. Computer Networks, 2021, 200: 108495. [doi: 10.1016/j.comnet.2021.108495]

[25] Pillai B, Hóu Z, Biswas K, Muthukkumarasamy V. Formal verification of the Burn-to-Claim protocol for blockchain interoperability. Journal of Latex Class Files, 2021, 14(8): 1–15.

[26] Herlihy M, Liskov B, Shrira L. Cross-chain deals and adversarial commerce. Proc. of the VLDB Endowment, 2019, 13(2): 100–113. [doi: 10.14778/3364324.3364326]

[27] Zhang JS, Gao JB, Li Y, Chen ZM, Guan Z, Chen Z. Xscope: Hunting for cross-chain bridge attacks. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 171. [doi: 10.1145/3551349.3559520]

[28] Wood G. Polkadot: Vision for a heterogeneous multi-chain framework. White Paper, 2016, 21(2327): 4662.

[29] Polkadot. Introduction to cross-consensus message format (XCM). 2024. https://wiki.polkadot.network/docs/learn-xcm

[30] GitHub, Inc. IBC relayer in rust. 2024. https://github.com/informalsystems/hermes

[31] Kobeissi N, Bhargavan K, Blanchet B. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In: Proc. of the 2017 IEEE European Symp. on Security and Privacy (EuroS&P). Paris: IEEE, 2017. 435–450. [doi: 10.1109/EuroSP.2017.38]

[32] Zhang JJ, Yang L, Gao XM, Tang GG, Zhang JY, Wang Q. Formal analysis of QUIC handshake protocol using symbolic model checking. IEEE Access, 2021, 9: 14836–14848. [doi: 10.1109/ACCESS.2021.3052578]

[33] Prabhu S, Chou KY, Kheradmand A, Godfrey PB, Caesar M. Plankton: Scalable network configuration verification through model checking. In: Proc. of the 17th USENIX Symp. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2020. 953–968.

[34] Zhang YM, Xu H, Xue CJ, Kuo TW. Probabilistic analysis of network availability. In: Proc. of the 30th Int'l Conf. on Network Protocols (ICNP). Lexington: IEEE, 2022. 1–11. [doi: 10.1109/ICNP55882.2022.9940438]

[35] Yin JQ, Zhu HB, Fei Y. Specification and verification of the Zab protocol with TLA+. Journal of Computer Science and Technology, 2020, 35(6): 1312–1323. [doi: 10.1007/s11390-020-0538-7]

[36] Akhtar S, Zahoor E. Formal specification and verification of MQTT protocol in PlusCal-2. Wireless Personal Communications, 2021, 119(2): 1589–1606. [doi: 10.1007/s11277-021-08296-4]

[37] Kukharenko V, Ziborov K, Sadykov R, Rezin R. Verification of hotstuff BFT consensus protocol with TLA+/TLC in an industrial setting. In: Silhavy R, ed. Informatics and Cybernetics in Intelligent Systems. Cham: Springer, 2021. 77–95. [doi: 10.1007/978-3-030-77448-6_9]

[38] Braithwaite S, Buchman E, Konnov I, Milosevic Z, Stoilkovska I, Widder J, Zamfir A. Formal specification and model checking of the tendermint blockchain synchronization protocol (short paper). In: Proc. of the 2nd Workshop on Formal Methods for Blockchains. Los Angeles, 2020. 10: 1–10: 8. [doi: 10.4230/OASIcs.FMBC.2020.10]

[39] Grundmann M, Hartenstein H. Verifying payment channels with TLA+. In: Proc. of the 2022 IEEE Int'l Conf. on Blockchain and Cryptocurrency (ICBC). Shanghai: IEEE, 2022. 1–3. [doi: 10.1109/ICBC54727.2022.9805487]

[40] GitHub, Inc. Interchain Standards (ICS) for the Cosmos network & interchain ecosystem. 2020. https://github.com/cosmos/ibc

[41] GitHub, Inc. Inter-Blockchain Communication Protocol (IBC) implementation in Golang. 2024. https://github.com/cosmos/ibc-go

[42] GitHub, Inc. IBC in solidity. 2024. https://github.com/hyperledger-labs/yui-ibc-solidity

[43] Back A, Corallo M, Dashjr L, Friedenbach M, Maxwell G, Miller A, Poelstra A, Timón J, Wuille P. Enabling blockchain innovations with pegged sidechains. 2024. https://blockstream.com/sidechains.pdf

[44] GitHub, Inc. Public code of our formal analysis of IBC protocol. https://github.com/michwqy/ibc-tla

[45] GitHub, Inc. TLA+ language support for Visual Studio Code. https://github.com/tlaplus/vscode-tlaplus

[46] GitHub, Inc. ICS 03/04: Some questions about handshake. 2023. https://github.com/cosmos/ibc/issues/1001

[47] GitHub, Inc. ICS4: Unrecoverable optimistic SendPacket. 2022. https://github.com/cosmos/ibc/issues/645

[48] GitHub, Inc. Question about proof of acknowledgement. 2023. https://github.com/hyperledger-labs/yui-ibc-solidity/issues/233

[49] GitHub, Inc. ICS20: Question about onAcknowledgePacket. 2023. https://github.com/cosmos/ibc/issues/1035

[50] GitHub, Inc. Something confusing about ICS03, ICS18. 2023. https://github.com/cosmos/ibc/issues/960

[51] GitHub, Inc. ICS03/ICS26: Some inconsistencies. 2023. https://github.com/cosmos/ibc/issues/961

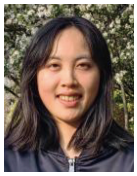
[52] GitHub, Inc. ICS18: Some possible mistakes. 2023. https://github.com/cosmos/ibc/issues/934

魏秋阳(1998－), 男, 硕士, 主要研究领域为区块链, 形式化分析.

赵旭峰(1998－), 男, 硕士, 主要研究领域为形式化方法, 模型检测, 智能合约.

朱雪阳(1971－), 女, 博士, 副研究员, CCF 高级会员, 主要研究领域为形式化方法, 嵌入式系统设计.

张文辉(1963－), 男, 博士, 研究员, CCF 高级会员, 主要研究领域为形式化方法, 演绎推理, 模型检测.



卢奕函(2001－), 女, 硕士生, CCF 学生会员, 主要研究领域为形式化方法.