区块链间通信协议

Abstract

区块链间通信协议（IBC）是一个端到端的、面向连接的、有状态的协议，用于在独立的分布式账本上的模块之间进行可靠、有序和认证的通信。IBC是为异质账本之间的互操作而设计的，这些账本排列在一个未知的动态拓扑结构中，以不同的共识算法和状态机运行。该协议通过指定足够的数据结构、抽象和通信协议的语义来实现这一点，一旦被参与的账本实施，它们就可以安全地进行通信。IBC与有效载荷无关，并提供了一个跨账本的异步通信基元，可作为各种应用的组成模块。

Index Terms—ibc; interblockchain; dlt

I. Introduction

由于其作为复制状态机的性质，必须在其上保持确定性的执行，因此必须保持对确切的确定性规则集的持续认同，单个分布式账本的吞吐量和灵活性都是有限的，必须用特定的应用优化来换取通用能力，并且只能为建立在其上的应用提供单一的安全模型。为了支持交易吞吐量、应用多样性、成本效率和容错性，以促进分布式账本应用的广泛部署，执行和存储必须被分割到许多独立的账本上，这些账本可以同时运行，独立升级，并以不同的方式进行专业化，其方式是保持不同应用之间的通信能力，这对于无权限创新和复杂的多部分合同至关重要。

一个多账本的设计方向是将一个单一的逻辑账本分散到不同的共识实例中，称为 "分片"，这些分片同时执行并存储状态的不相干部分。为了对安全性和有效性进行全局推理，为了在分片之间正确路由数据和代码，这些设计必须采取 "自上而下的方法"--构建一个特定的网络拓扑结构，通常是一个单一的根分类账和一个星形或树形的分片，以及工程协议规则和激励措施来执行该拓扑结构。然后，消息传递可以通过Polkadot的XCMP[1]和Ethereum 2.0的跨分片通信[2]等系统在这种分片拓扑之上实现。这种方法在简单性和可预测性方面具有优势，但在保证状态转换的有效性方面面临着困难的技术问题[3]，需要所有分片都遵守单一验证器集（或随机选出的子集）和单一虚拟机，并且由于必须就网络拓扑结构或分类账规则集的改变达成全球共识，因此在随着时间推移升级方面面临挑战。此外，这种分片系统是很脆弱的：如果超过了容错阈值，系统需要协调全球停止和重新启动，并可能启动复杂的状态转换回滚程序--不可能安全地隔离网络图的拜占庭部分并继续运行。

区块链间通信协议（IBC）提供了一种机制，通过这种机制，独立的、主权复制的分类账可以安全、自愿地进行互动，同时只共享一个最低限度的必要的公共接口。该协议的设计接近一个不同版本的扩展和互操作性问题：使异质分布式账本的网络能够安全、可靠地互操作，以未知的拓扑结构排列，在可能的情况下保持数据保密，其中账本可以多样化、发展和重新排列，独立于彼此或特定的强加拓扑结构或账本设计。在一个广泛的、动态的互操作账本网络中，预计会出现零星的拜占庭故障，因此协议还必须根据所涉及的应用和账本的要求，检测、缓解和控制拜占庭故障的潜在损害，而不需要使用额外的信任方或全球协调。

为了促进这种异质性的互操作，区块链间通信协议采用了自下而上的方法，规定了在两个分类账之间实现互操作所需的一组要求、功能和属性，然后规定了多个互操作分类账的不同组成方式，以保持更高级别的协议要求。因此，IBC对整个网络拓扑结构没有任何假设和要求，对实施分类账只要求有一组已知的、具有特定属性的最小功能。IBC中的分类账被定义为它们的轻型客户端共识验证功能，从而扩大了 "分类账 "的范围，包括单机和复杂的共识算法。IBC的实现预计将与主机分类账上的更高级别的模块和协议共同驻留。托管IBC的分类账必须为共识成绩单(consensus transcript)验证和加密承诺证明的生成提供一定的功能，IBC数据包中继器（非分类账进程）预计可以访问网络协议和物理数据链，以读取一个分类账的状态并向另一个分类账提交数据。

IBC数据包中的数据有效载荷对协议本身是不透明的--每个分类账上的模块决定了它们之间发送的数据包的语义。对于跨账本的代币转移，数据包可能包含可替换的代币信息，其中资产在一个账本上被锁定，以在另一个账本上铸造相应的凭证。对于跨账本治理，数据包可以包含投票信息，其中一个账本上的账户可以在另一个账本的治理系统中投票。对于跨账本的账户授权，数据包可以包含交易授权信息，允许一个账本上的账户被另一个账本上的账户所控制。对于跨账本的去中心化交易所，数据包可以包含订单意图信息或交易结算信息，这样，不同账本上的资产可以通过过渡性托管和数据包序列进行交换，而不需要离开其主机账本。

这种自下而上的方法与TCP/IP规范[4]非常相似，并直接受其启发，用于包交换计算机网络中主机之间的互操作性。正如TCP/IP定义了两个主机之间的通信协议，以及更高级别的协议将许多双向的主机与主机之间的链接编织成复杂的拓扑结构，IBC定义了两个分类帐之间的通信协议，以及更高级别的协议将许多双向的分类帐之间的链接编织成格式化的多分类帐应用。正如TCP/IP数据包包含不透明的有效载荷数据，其语义由每个主机上的进程解释，IBC数据包包含不透明的有效载荷数据，其语义由每个分类账的模块解释。正如TCP/IP在进程之间提供可靠、有序的数据传输，允许一个主机上的进程推理另一个主机上的进程的状态一样，IBC在模块之间提供可靠、有序的数据传输，允许一个分类账上的模块推理另一个分类账上的模块的状态。

本文旨在概述IBC协议所定义的抽象概念以及构成这些抽象概念的机制。我们首先概述了该协议的结构，包括范围、接口和操作要求。随后，我们详细介绍了协议所定义的抽象概念，包括模块、端口、客户端、连接、通道、数据包和中继器，并描述了用于打开和关闭握手、数据包中继、边缘情况处理和中继器操作的子协议。在解释了协议的内部结构后，我们定义了应用程序可以利用IBC的接口，并简述了一个用于可替换令牌传输的应用级协议的例子。最后，我们叙述了到目前为止该协议的测试和部署工作。附录包括连接握手、信道握手和数据包中继算法的伪代码。

II. Protocol scope & properties

A. Scope

IBC处理独立账本上的模块之间转发的不透明数据包的认证、传输和排序--账本可以在单机上运行，由运行共识算法的许多节点复制，或者由状态可以被验证的任何进程构建。该协议是在两个账本上的模块之间定义的，但被设计为在任意拓扑结构中连接的任意数量的账本上的任意数量的模块之间安全地同时使用。

B. Interface

IBC位于模块--智能合约、其他账本组件或其他独立执行的账本上的应用逻辑片断--和底层共识协议、区块链和网络基础设施（如TCP/IP）之间，位于另一侧。

IBC为模块提供了一系列功能，这些功能很像可能提供给一个模块与同一账本上的另一个模块互动的功能：在已建立的连接和通道上发送数据包和接收数据包，此外还有管理协议状态的调用：打开和关闭连接和通道，选择连接、通道和数据包交付选项，以及检查连接和通道状态。

IBC需要底层账本的某些功能和属性，主要是最终性（或阈值最终性小工具）、廉价的可验证的共识记录（这样，一个轻量级的客户端算法可以用比完整节点少得多的计算和存储来验证共识过程的结果），以及简单的键/值存储功能。在网络方面，IBC只需要最终的数据交付--不需要认证、同步或排序属性。

C. Operation

IBC的主要目的是在独立主机账本上运行的模块之间提供可靠的、经过验证的、有序的通信。这需要在数据中继、数据保密性和可读性、可靠性、流量控制、认证、状态性和复用等方面的协议逻辑。

1. Data Relay

在IBC架构中，各模块并不直接通过网络基础设施相互发送消息，而是创建要发送的消息，然后通过监测 "中继进程 "从一个分类账物理地中继到另一个分类账。IBC假定存在一组中继进程，可以访问底层网络协议栈（可能是TCP/IP、UDP/IP或QUIC/IP）和物理互连基础设施。这些中继进程监控着一组执行IBC协议的分类账，持续扫描每个分类账的状态，并在发出的数据包被提交后请求在另一个分类账上执行交易。对于两个分类账之间连接的正确操作和进展，IBC只要求至少存在一个正确的、活的中继进程，可以在分类账之间中继。

2. Data confidentiality and legibility

IBC协议只要求将IBC协议正确运行所需的最低限度的数据提供给中继者进程并使其可读（以标准化的格式进行序列化），分类账可以选择只将这些数据提供给特定的中继者。这些数据包括共识状态、客户、连接、信道和数据包信息，以及构建状态中特定键/值对的包含或排除证明所需的任何辅助状态结构。所有必须证明给另一个分类账的数据也必须是可读的；也就是说，它必须以两个分类账商定的标准化格式进行序列化。

3. Reliability

网络层和中继器进程可以以任意的方式行事，丢弃、重新排序或重复数据包，故意尝试发送无效的交易，或以其他拜占庭方式行事，而不影响IBC的安全性或有效性。这是通过为通过IBC通道发送的每个数据包分配一个序列号来实现的，该序列号由接收账本上的IBC处理程序（实现IBC协议的账本部分）检查，并为发送账本提供一种方法，在发送更多数据包或采取进一步行动之前检查接收账本是否确实收到并处理了数据包。加密承诺被用来防止数据报伪造：发送分类账对发出的数据包作出承诺，而接收分类账则检查这些承诺，因此在传输过程中被中继者改变的数据报将被拒绝。IBC还支持无序信道，它不强制执行相对于发送的数据包接收顺序，但仍强制执行完全一次的交付。

4. Flow control

IBC并没有为计算层面或经济层面的流量控制提供具体的协议级规定。底层分类账预计会有计算吞吐量限制设备和它们自己的流量控制机制，如气体市场。应用层面的经济流量控制--根据内容限制特定数据包的速率--对于确保安全属性和遏制拜占庭故障的破坏可能是有用的。例如，在IBC通道上传输价值的应用程序可能希望限制每个区块的价值传输率，以限制潜在的拜占庭行为的损害。IBC为模块提供了拒绝数据包的设施，并将具体细节留给更高级别的应用协议。

5. Authentication

所有通过IBC发送的数据都是经过验证的：由发送账本的共识算法最终确定的区块必须通过加密承诺对发出的数据包进行承诺，而接收账本的IBC处理程序必须在对数据报采取行动之前验证共识记录和加密承诺证明。

6. Statefulness

如上所述，可靠性、流量控制和认证要求IBC为每个数据流初始化和维护某些状态信息。这些信息被分成三个抽象：客户、连接和通道。每个客户对象都包含关于对手方账本的共识状态的信息。每个连接对象包含一对特定的命名标识符，由两个分类账在握手协议中商定，它唯一地标识了两个分类账之间的连接。每个通道，具体到一对模块，包含有关协商的编码和复用选项以及状态和序列号的信息。当两个模块希望进行通信时，它们必须在两个分类账之间找到一个现有的连接和通道，如果还没有，则初始化一个新的连接和通道。初始化连接和通道需要一个多步骤的握手，一旦完成，在连接的情况下，确保只有两个预定的分类账被连接，在通道的情况下，确保两个模块被连接，未来转发的数据报将被认证、编码，并按要求排序。

7. Multiplexing

为了让单个主机分类账内的许多模块同时使用IBC连接，IBC允许将任何数量的通道与单个连接相关联。每个通道唯一标识一个数据流，数据包可以按顺序（在有序通道的情况下）发送，并且总是精确地发送一次，到接收分类账上的目标模块。通道通常被期望与每个分类账上的一个模块相关联，但一对多和多对一的通道也是可能的。每个连接的通道数量是无限制的，促进了并发的吞吐量，只受限于底层分类账的吞吐量，只需要一个连接和一对客户端来跟踪共识信息（和共识成绩单的验证成本因此在使用连接的所有通道中摊销）。

III. Host ledger requirements

1) Module system

主账本必须支持一个模块系统，据此，独立的、可能相互不信任的代码包可以安全地在同一个账本上执行，控制它们如何以及何时允许其他模块与之通信，并由控制器模块或执行环境识别和操纵。

2) Key/value Store

主账本必须提供一个键/值存储接口，允许读取、写入和删除值。

这些功能必须被许可给IBC处理模块，以便只有IBC处理模块可以写入或删除某个子集的路径。这可能会被实现为整个分类账所使用的更大的键/值存储的一个子存储（前缀为key-space）

主账本必须提供该接口的实例，该实例是可证明的，这样，主账本的轻客户端算法可以验证是否存在被写入它的特定键值对。

该接口不需要任何特定的存储后端或后端数据布局。分类账可以选择使用根据其需求配置的存储后端，只要上面的存储满足指定接口并提供承诺证明。

3) Consensus state introspection

主机分类账必须提供反省其当前高度、当前共识状态（由主机分类账的轻客户端算法利用）和一定数量的最近共识状态（例如过去的头文件）的能力。这些都是用来防止在与其他分类账建立连接的握手过程中的中间人攻击--每个分类账都会检查其他分类账是否真的在使用其共识状态来验证 使用其共识状态的数据。

4) Timestamp access

为了支持基于时间戳的超时，主机分类帐必须提供一个当前的Unix风格的时间戳。后续头文件中的超时必须是不递减的。

5) Port system

主机分类账必须实现一个端口系统，IBC处理程序可以允许主机分类账中的不同模块与唯一命名的端口绑定。端口由一个标识符来识别，并且必须经过许可，以便:

- 一旦一个模块绑定了一个端口，其他模块就不能使用该端口，直到该模块释放它。
- 一个模块可以与多个端口绑定。
- 端口的分配是先到先得的。
- 为已知模块 "保留 "的端口可以在账本首次启动时被绑定。

这种许可可以用每个端口的唯一引用（对象能力[5]）来实现，也可以用基于来源的认证（如以太坊合约中的msg.sender），或者用其他的访问控制方法，在任何情况下都由主机分类账来执行。

端口一般不打算成为人类可读的标识符--正如DNS名称解析和特定应用的标准化端口号是为了从TCP/IP用户那里抽象出IP地址和端口的细节一样，分类账名称解析和特定应用的标准化端口可以被创建，以抽象出分类账识别和端口选择的细节。这种寻址系统可以很容易地建立在IBC本身之上，这样，通过IBC与寻址系统的初始连接就可以为随后与其他分类账和应用程序的连接进行名称解析。

6) Exception/rollback system

主机分类账必须支持异常或回滚系统，据此，交易可以中止执行，并恢复之前做出的任何状态变化（包括在同一交易中发生的其他模块的状态变化），酌情排除消耗的天然气和费用支付。

7) Data availability

对于交付或超时安全，主机分类账必须具有最终的数据可用性，这样，状态中的任何键/值对最终都可以被中继者检索到。对于确切的一次安全来说，数据的可用性是不需要的。

对于数据包中继的有效性，主机账本必须具有有界的交易有效性，这样，传入的交易会在区块高度或时间戳约束内得到确认（特别是小于分配给数据包的超时）。

IBC数据和其他不直接存储在Merklized状态中但被中继器依赖的数据，必须可以被中继器进程有效计算。

IV. Protocol structure

A. Clients

客户端抽象封装了实施区块链间通信协议的分类账的共识算法所需满足的属性。这些属性对于在更高层次的协议抽象中进行有效和安全的状态验证是必要的。IBC中用来验证另一个分类账的共识笔录和状态子组件的算法被称为 "有效性谓词"，并将其与验证者认为正确的状态配对，形成一个 "轻型客户端"（俗称 "客户端"）。

1) Motivation

在IBC协议中，行为体可以是终端用户、非账本进程或账本，需要能够验证另一个账本的共识算法所同意的另一个账本的状态更新，并拒绝任何另一个账本的共识算法没有同意的可能的更新。轻客户端是一个行为体可以做到这一点的算法。客户端抽象化了这个模型的接口和要求，因此IBC协议可以很容易地与运行新共识算法的新账本集成，只要提供满足所列要求的相关轻客户端算法。

除了本规范中描述的属性外，IBC并没有对分类账的内部操作及其共识算法提出任何要求。一个分类账可以由一个用私钥签署操作的单一进程、一个一致签署的法定进程、许多操作拜占庭容错共识算法的进程（一个复制的或分布式的分类账），或其他尚未发明的配置组成--从IBC的角度来看，一个分类账完全由其轻度客户验证和等价检测逻辑定义。客户端一般不会包括一般的状态转换逻辑的验证（因为这相当于简单地执行其他状态机），但在特定情况下可以选择验证部分状态转换，如果这样做是渐进式的高效，可以验证整个状态转换，也许通过使用SNARK[6]进行压缩。

然而，在外部，IBC客户端使用的轻型客户端验证功能必须具有最终性，这样，经过验证的区块（受制于通常的共识安全假设），一旦被验证，就不能再被恢复。IBC协议的更高抽象层的安全性和提供给使用该协议的应用程序的保证取决于这种最终性的属性。

为了将最终性嫁接到中本聪的共识算法上，比如比特币[7]中使用的算法，客户端可以作为内部非最终性客户端的阈值视图。在利用IBC协议的模块与概率最终性共识算法互动的情况下，不同的应用可能需要不同的最终性阈值，可以创建一个只写的客户端来跟踪标题，许多具有不同最终性阈值（确认深度后，状态根部被认为是最终的）的只读客户端可以使用同一个状态。当然，这将引入与运行共识算法的完整节点所要求的不同的安全假设，以及用户必须在其特定应用安全需求的基础上进行权衡。

客户端协议的设计是为了支持第三方的引入。考虑一下一般的例子。爱丽丝，一个账本上的模块，想把鲍勃，一个爱丽丝认识（也认识爱丽丝）的第二个账本上的模块，介绍给卡罗尔，一个爱丽丝认识但鲍勃不认识的第三个账本上的模块。爱丽丝必须利用现有的渠道向鲍勃传递卡诺尔的可序列化谓词。有效性谓词，然后Bob可以与之建立连接和通道，以便Bob和Carol可以直接对话。如果有必要，Alice也可以在Bob尝试连接之前，向Carol传达Bob的有效性谓词，这样Carol就知道要接受传入的请求。

客户端接口的构造是，只要底层分类账能够提供适当的气体计量机制来收取计算和存储费用，就可以在运行时安全地提供自定义验证逻辑来定义一个自定义客户端。例如，在一个支持WASM执行的主机分类账上，有效性谓词和等价谓词可以在客户端实例创建时作为可执行的WASM函数提供。

2) Definitions

有效性谓词是一个不透明的函数，由客户类型定义，根据当前的共识状态来验证头文件。使用有效性谓词应该比为给定的父标头和网络消息列表重放完整的共识算法和状态机的计算效率高得多。

共识状态是一个不透明的类型，代表有效性谓词的状态。轻客户有效性谓词算法与特定的共识状态相结合，必须能够验证相关共识算法所同意的状态更新。共识状态也必须可以以规范的方式进行清算，以便第三方，如对手方账本，可以检查某一账本是否存储了某一状态。它还必须可以被它所代表的分类账反查，这样分类账就可以在过去的高度上查找自己的共识状态，并将其与另一个分类账的客户端中存储的共识状态进行比较。

承诺根是一种廉价的方式，用于下游逻辑验证键/值对在特定高度的状态下是存在还是没有。通常这将被实例化为梅克尔树的根。头是一个由客户类型定义的不透明的数据结构，它提供信息来更新一个共识状态。头信息可以提交给相关的客户端，以更新存储的共识状态。它们可能包含一个高度，一个证明，一个新的承诺根，以及可能对有效性谓词的更新。

错误行为谓词是一个由客户端类型定义的不透明函数，用于检查数据是否构成对共识协议的违反。这可能是两个状态根数不同但高度相同的签名头，一个包含无效状态转换的签名头，或共识算法所定义的其他渎职证据。

3) Desired properties

轻客户端必须提供一个安全的算法来验证其他分类账的规范头，使用现有的共识状态。然后，更高级别的抽象将能够用存储在共识状态中的承诺根来验证状态的子组件，这些子组件保证已经被其他分类账的共识算法所承诺。

有效性谓词被期望反映运行相应共识算法的完整节点的行为。给定一个共识状态和一个消息列表，如果一个完整的节点接受了一个新的头，那么轻型客户端也必须接受它，如果一个完整的节点拒绝它，那么轻型客户端也必须拒绝它。

轻客户机没有重放整个消息记录，所以在共识不当行为的情况下，轻客户机的行为有可能与完整节点的行为不同。在这种情况下，可以生成一个错误行为证明，证明有效性谓词和完整节点之间的分歧，并提交给分类帐，这样分类帐就可以安全地停用轻型客户机，使过去的状态根基无效，并等待更高级别的干预。

有效性谓词的有效性取决于共识算法的安全模型。例如，共识算法可以是具有受信任的操作者集的BFT授权证明，或具有tokenholder集的BFT股权证明，每一种都有一个定义的阈值，超过这个阈值的拜占庭行为可能导致分歧。客户端可以有时间敏感的有效性谓词，例如，如果在一段时间内没有提供标头（例如，在股权证明系统中的三周解约期），将不再有可能更新客户端。

4) State verification

客户端类型必须定义函数来验证客户端所追踪的分类账的内部状态。内部实现的细节可能有所不同（例如，回环客户端可以简单地直接从状态中读取，不需要证明）。面向外部的客户端可能会验证签名或矢量承诺证明。

5) Example client instantiations

a) Loopback

本地账本的回环客户只是从本地状态中读取，它必须有访问权限。这类似于TCP/IP中的localhost或127.0.0.1。

b) Simple signatures

运行非复制账本的单机客户端用已知的公钥检查该本地机器发送的信息的签名。多重签名或阈值签名方案也可以以这种方式使用。

c) Proxy clients

代理客户端验证另一个（代理）账本对目标账本的验证，方法是在证明中首先包括代理账本上客户状态的证明，然后是目标账本相对于代理账本上客户状态的子状态的二次证明。这使得代理客户端可以避免存储和跟踪目标分类账本身的共识状态，代价是增加代理分类账正确性的安全假设。

d) BFT consensus and verifiable state

对于主权、容错分布式账本之间的互操作性的直接应用，最常见和最有用的客户端类型将是BFT共识算法实例的轻型客户端，如Tendermint[8]、GRANDPA[9]或HotStuff[10]，账本使用Merklized状态树，如IAWL+树[11]或Merkle Patricia树[12]。此类实例的客户端算法将利用BFT共识算法的轻度客户端有效性谓词，并将最低限度的共识等价交换（双重签名）视为不当行为，以及其他可能的不当行为类型，具体到所涉及的权威证明或股权证明系统。

6) Client lifecycle

a) Creation

任何人都可以在任何时候通过指定一个标识符、客户类型和初始共识状态来创建无权限的客户。

b) Update

更新一个客户端是通过提交一个新的头来完成的。当新的标头与存储的客户端状态的有效性谓词和共识状态进行验证时，客户端将相应地更新其内部状态，可能最终确定承诺根，并更新存储的共识状态中的签名授权逻辑。

如果一个客户端不能再被更新（例如，如果解除绑定期已过），将不再可能通过与该客户端相关的连接和通道发送任何数据包，或在飞行中对任何数据包进行超时处理（因为目的地分类账上的高度和时间戳不再能被验证）。必须进行人工干预，以重置客户端的状态或将连接和通道迁移到另一个客户端。这不能安全地自动完成，但实施IBC的分类账可以选择允许治理机制来执行这些行动（甚至可能是每个客户/连接/通道的控制性多重签名或合同）。

c) Misbehaviour

如果客户端检测到错误行为的证据，客户端可以采取适当的行动，可能会使以前有效的承诺根部无效，并阻止未来的更新。什么是错误行为的确切构成取决于有效性谓词验证输出的共识算法。

B. Connections

连接抽象在两个独立的分类账上封装了两个有状态的对象（连接端），每个对象都与另一个分类账的轻客户端相关，它们共同促进了跨分类账的子状态验证和数据包中继（通过通道）。在一个未知的、动态的拓扑结构中，使用握手子协议安全地建立连接。

1) Motivation

IBC协议为数据包提供了授权和排序语义：分别保证数据包已经在发送账本上提交（并根据执行的状态转换，如托管令牌），以及它们已经以特定的顺序提交了一次，并可以以相同的顺序交付一次。连接抽象与客户端抽象一起定义了IBC的授权语义。秩序语义由通道提供。

2) Definitions

连接结束是对一个分类账上的连接结束的状态跟踪，定义如下。

```
enum ConnectionState {
    INIT,
    TRYOPEN,
    OPEN,
}

interface ConnectionEnd {
    state: ConnectionState
    counterpartyConnectionIdentifier: Identifier
    counterpartyPrefix: CommitmentPrefix
    clientIdentifier: Identifier
   counterpartyClientIdentifier: Identifier
   version: string
}
```

- version 字段是一个不透明的字符串，可用于确定利用此连接的通道或数据包的编码或协议。
- state 字段描述了连接端的当前状态。
- counterpartyConnectionIdentifier字段标识了与此连接相关的对手方分类账上的连接端。
- counterpartyPrefix字段包含用于与此连接相关的对手方分类账上的状态验证的前缀。
- clientIdentifier字段标识了与此连接相关的客户。
- counterpartyClientIdentifier字段确定了与此连接相关的对手方分类账上的客户。
- version 字段是一个不透明的字符串，可用于确定利用此连接的通道或数据包的编码或协议。

3) Opening handshake

开放式握手子协议允许每个分类账验证用于引用其他分类账上的连接的标识符，使每个分类账上的模块能够推理出其他分类账上的引用。

开放握手由四个数据报组成。`ConnOpenInit`，`ConnOpenTry`，`ConnOpenAck`，和`ConnOpenConfirm`.

一个正确的协议执行，在两个分类账A和B之间，连接状态格式为（A，B），流程如下。

```
Datagram          Prior state      Posterior state
ConnOpenInit       (-, -)           (INIT, -)
ConnOpenTry        (INIT, none)     (INIT, TRYOPEN)
ConnOpenAck        (INIT, TRYOPEN)  (OPEN, TRYOPEN)
ConnOpenConfirm    (OPEN, TRYOPEN)  (OPEN, OPEN)
```

在实施子协议的两个分类账之间的开放握手结束时，以下属性成立。

- 每个分类账都拥有发起者最初指定的对方的正确共识状态。
- 每个分类账都知道并同意其在另一个分类账上的标识符。
- 每个分类账都知道另一个分类账已经同意了相同的数据。

连接握手可以安全地进行无权限操作，调制反垃圾邮件措施（支付气体）。

ConnOpenInit，在分类账A上执行，初始化分类账A上的连接尝试，为两个分类账上的连接指定一对标识符，为现有的轻型客户指定一对标识符（每个分类账一个）。 分类账A在其状态中存储一个连接结束对象。

在分类账B上执行的ConnOpenTry向分类账B转发了分类账A的连接尝试通知，提供了一对连接标识符、一对客户端标识符和所需版本。ledger B验证这些标识符是否有效，检查版本是否兼容，验证ledger A存储了这些标识符的证明，并验证ledger A用来验证ledger B的轻型客户端拥有ledger B的正确共识状态的证明。

在分类账A上执行的ConnOpenAck，将接受从分类账B到分类账A的连接打开尝试，提供标识符，现在可以用来查找连接端对象。分类账A验证所请求的版本是兼容的，验证分类账B已经存储了与分类账A相同的标识符的证明，并验证分类账B用来验证分类账A的轻客户端具有分类账A的正确共识状态。

在分类账B上执行的ConnOpenConfirm确认了分类账A对分类账B的连接的开放。分类账B只是检查分类账A是否已经执行了ConnOpenAck并将连接标记为开放。Ledger B随后将其连接的末端标记为开放。在执行ConnOpenConfirm后，连接在两端都是开放的，可以立即使用。

4) Versioning

在握手过程中，连接的两端就与该连接相关的版本字节串达成协议。目前，这个版本字节符的内容对IBC核心协议是不透明的。在未来，它可能被用来指示什么样的通道可以利用有关的连接，或者与通道相关的数据报将使用什么样的编码格式。主机分类帐可以利用版本数据来协商编码、优先级或与IBC之上的自定义逻辑有关的连接特定元数据。主机分类帐也可以安全地忽略版本数据或指定一个空字符串。

C. Channels

通道抽象为区块链间通信协议提供了三类消息交付语义：排序、完全一次交付和模块许可。通道作为一个账本上的模块和另一个账本上的模块之间传递数据包的管道，确保数据包只执行一次，按照发送的顺序交付（如果需要），并且只交付给目的地账本上拥有通道另一端的相应模块。每个通道都与一个特定的连接相关联，一个连接可以有任何数量的相关通道，允许使用共同的标识符，并在利用一个连接和轻型客户端的所有通道中摊薄头的验证成本。

通道是与有效载荷无关的。发送和接收IBC数据包的模块决定如何构建数据包以及如何对传入的数据包采取行动，并且必须利用自己的应用逻辑来决定根据数据包包含的数据应用哪些状态事务。

1) Motivation

区块链间通信协议使用跨账本的消息传递模式。IBC数据包由外部中继程序从一个分类账转发到另一个分类账。两个账本，A和B，独立地确认新的区块，从一个账本到另一个账本的数据包可能会被延迟，审查，或任意地重新排序。数据包对中继者是可见的，任何中继者进程都可以从一个分类账中读取数据包，并提交给任何其他分类账。

IBC协议必须提供Ordering（对于ordered的通道）和完全一次的交付保证，以允许应用程序推理两个分类账上连接模块的综合状态。例如，一个应用程序可能希望允许一个单一的代币化资产在多个分类账之间转移并持有，同时保持可替换性和供应保护。当一个特定的IBC包被投入到分类账B时，应用程序可以在分类账B上铸造资产凭证，并要求分类账A上该包的发送者在分类账A上托管同等数量的资产，直到这些凭证后来被反向的IBC包赎回到分类账A。这种订购保证和正确的应用逻辑可以确保两个分类账的总供应量得到保留，并且在分类账B上铸造的任何凭证后来都可以被赎回到分类账A。

2) Definitions

通道是一个管道，用于在不同账本上的特定模块之间精确地传递数据包，它至少有一端能够发送数据包，另一端能够接收数据包。

有序信道是一个信道，其中数据包完全按照它们被发送的顺序交付。

无序信道是一个信道，数据包可以按任何顺序交付，这可能与它们的发送顺序不同。

所有信道都提供精确的一次数据包交付，这意味着在信道一端发送的数据包最终将不多也不少地传递到另一端。通道端是一个数据结构，存储与一个参与的分类账上的通道一端相关的元数据，定义如下。

```
interface ChannelEnd {
    state: ChannelState
    ordering: ChannelOrder
    counterpartyPortIdentifier: Identifier
    counterpartyChannelIdentifier: Identifier
    nextSequenceSend: uint64
    nextSequenceRecv: uint64
    nextSequenceAck: uint64
    connectionHops: [Identifier]
    version: string
}
```

- state 是通道末端的当前状态。
- ordering 字段表示通道是有序的还是无序的。这是一个枚举，而不是一个布尔值，以便将来能够方便地支持其他类型的排序。
- 对手方端口标识符（counterpartyPortIdentifier）确定了拥有通道另一端的对手方分类账上的端口。
- counterpartyChannelIdentifier标识了对手方账本上的通道端。
- nextSequenceSend，单独存储，跟踪下一个要发送的数据包的序列号。
- 单独存储的nextSequenceRecv跟踪下一个待接收的数据包的序列号。
- nextSequenceAck，单独存储，跟踪下一个要被确认的数据包的序列号。
- connectionHops存储了连接标识符的列表，按照顺序，在这个通道上发送的数据包将沿着这个列表移动。目前这个列表的长度为1，将来可能会支持多跳通道。
- version 字符串存储一个不透明的通道版本，这是在握手过程中商定的。这可以确定模块级的配置，如通道使用的是哪种数据包编码。这个版本不被核心IBC协议所使用。

通道末端有一个状态。

```
enum ChannelState {
    INIT,
    TRYOPEN,
    OPEN,
    CLOSED,
}
```

- 处于INIT状态的信道端刚刚开始了开场握手。
- 处于TRYOPEN状态的信道端已经确认了对手方账本上的握手步骤。
- 处于开放状态的信道端已经完成了握手，并准备发送和接收数据包。
- 处于CLOSED状态的信道端已经关闭，不能再用于发送或接收数据包。

包，封装了不透明的数据，通过一个通道从一个模块传输到另一个模块，是一个特殊的接口，定义如下。

```
interface Packet {
    sequence: uint64
    timeoutHeight: uint64
    timeoutTimestamp: uint64
    sourcePort: Identifier
    sourceChannel: Identifier
    destPort: Identifier
    destChannel: Identifier
    data: bytes
}
```

- sequence与发送和接收的顺序相对应，序列号较早的数据包必须在序列号较晚的数据包之前发送和接收。
- 超时高度（timeoutHeight）表示目的地账本上的共识高度，超过这个高度，数据包将不再被处理，而是被算作超时。
- timeoutTimestamp表示目的地账本上的一个时间戳，过了这个时间戳，数据包将不再被处理，而将被算作超时了。
- 源端口（sourcePort）标识了发送分类账上的端口。
- sourceChannel标识了发送分类账上的通道端。
- destPort标识了接收账本上的端口。
- destChannel标识了接收账本上的通道端。
- data是一个不透明的值，可以由相关模块的应用逻辑来定义。

请注意，数据包从未被直接序列化。相反，它是在某些函数调用中使用的一种中介结构，可能需要由调用IBC处理程序的模块来创建或处理。

3) Properties

a) Efficiency

由于信道本身没有施加流量控制，数据包传输和确认的速度只受限于底层分类账的速度。

b) Exactly-once delivery

在信道一端发送的IBC数据包不超过一次就能传递到另一端。确切的一次安全不需要网络同步假设。如果一个或两个分类帐停止，数据包可能被传递不超过一次，一旦分类帐恢复，数据包将能够再次流动。

c) Ordering

在有序通道上，数据包的发送和接收顺序是相同的：如果数据包x在数据包y之前由分类账A的通道端发送，那么数据包x将在数据包y之前由分类账B的相应通道端接收。无序数据包，像有序数据包一样，有单独的超时，以目的地分类账的高度或时间戳来指定。

d) Permissioning

信道被许可给每一端的一个模块，在握手过程中确定，并且在握手后不可改变（高层逻辑可以通过标记端口的所有权来标记信道的所有权）。只有拥有与通道端相关的端口的模块才能在该通道上发送或接收。

4) Channel lifecycle management

a) Opening handshake

两个账本A和B之间的通道开放握手，其状态格式为（A，B），流程如下。

```
| Datagram | Prior state | Posterior state |
| --- | --- | --- |
| ChanOpenInit | (-, -) | (INIT, -) |
| ChanOpenTry | (INIT, -) | (INIT, TRYOPEN) |
| ChanOpenAck | (INIT, TRYOPEN) | (OPEN, TRYOPEN) |
| ChanOpenConfirm | (OPEN, TRYOPEN) | (OPEN, OPEN) |
```

ChanOpenInit，在分类账A上执行，启动了从分类账A上的模块到分类账B上的模块的通道开放握手，提供了本地通道标识符、本地端口、远程端口和远程通道标识符的标识。分类账A在其状态中存储了一个通道端对象。

在账本B上执行的ChanOpenTry，向账本B上的模块转发了通道握手尝试的通知，提供了一对通道标识符、一对端口标识符和所需的版本。账本B验证了账本A已按要求存储这些标识符的证明，查找拥有目标端口的模块，调用该模块以检查所要求的版本是否兼容，并在其状态中存储一个通道端对象。

在分类账A上执行的ChanOpenAck，将对信道握手尝试的接受转达给分类账A上的模块，提供标识符，现在可以用来查询信道端。分类账A验证了分类账B已按要求存储信道元数据的证明，并将其信道端标记为OPEN。

在分类账B上执行的ChanOpenConfirm确认了从分类账A到分类账B的通道的开放。分类账B只是检查分类账A是否执行了ChanOpenAck并将通道标记为开放。Ledger B随后将其通道的一端标记为开放。在执行ChanOpenConfirm后，通道在两端都是开放的，可以立即使用。

当开放握手完成后，发起握手的模块将拥有主机分类帐上创建的通道的一端，而它指定的对手方模块将拥有对手方分类帐上创建的通道的另一端。一旦一个通道被创建，所有权只能通过改变相关端口的所有权来改变。

b) Versioning

在握手过程中，通道的两端就与该通道相关的版本字节串达成协议。这个版本字节串的内容对IBC核心协议是不透明的。主机分类帐可以利用版本数据来表明支持的应用层协议，同意数据包编码格式，或协商其他与IBC之上的自定义逻辑相关的通道元数据。主机分类帐也可以安全地忽略版本数据或指定一个空字符串

c) Closing handshake

两个账本A和B之间的通道关闭握手，其状态格式为（A，B），流程如下。

```
| Datagram | Prior state | Posterior state |
| --- | --- | --- |
| ChanCloseInit | (OPEN, OPEN) | (CLOSED, OPEN) |
| ChanCloseConfirm  | (CLOSED, OPEN) | (CLOSED, CLOSED) |
```

ChanCloseInit，在分类账A上执行，关闭分类账A上的通道末端。

ChanCloseInit，在分类账B上执行，只是验证通道在分类账A上被标记为关闭，并在分类账B上关闭结束。

任何飞行中的数据包都可以在通道关闭后立即超时退出。

一旦关闭，通道就不能再打开，标识符也不能再使用。识别符的重复使用被阻止，因为我们要防止对以前发送的数据包进行潜在的重放。重放问题类似于使用签名信息的序列号，除了轻客户端算法 "签名"信息（IBC数据包），重放预防序列是端口标识符、信道标识符和数据包序列的组合--因此我们不能允许相同的端口标识符和信道标识符在序列重置为零的情况下再次重用，因为这可能允许数据包被重放。如果规定并跟踪特定的最大高度/时间的超时，就有可能安全地重复使用标识符，未来的协议版本可能会纳入这一功能。

5) Sending packets

一个模块调用sendPacket函数，以便在调用模块所拥有的信道端向对手方分类账上的相应模块发送一个IBC数据包。

调用模块必须在调用sendPacket的同时原子化地执行应用逻辑。 IBC处理程序依次执行以下步骤:

- 检查通道和连接是否开放以发送数据包。
- 检查调用模块是否拥有发送端口
- 检查数据包的元数据是否与通道和连接的信息相匹配
- 检查指定的超时高度是否已经在目的地账本上过了。

- 增加与通道相关的发送序列计数器（在有序通道的情况下）。
- 对数据包数据和数据包超时进行恒定大小的承诺

请注意，完整的数据包并不存储在账本的状态中--只是对数据和超时值的简短哈希承诺。数据包的数据可以从交易执行中计算出来，并可能作为日志输出返回，中继者可以索引。

6) Receiving packets

recvPacket函数由一个模块调用，以便接收和处理在对手方账本上相应信道端发送的IBC数据包。调用模块必须在调用recvPacket的同时原子化地执行应用逻辑，很可能事先就计算出确认值。

IBC处理程序依次执行以下步骤。

- 检查通道和连接是否开放以接收数据包。
- 检查调用模块是否拥有接收端口
- 检查数据包的元数据是否与通道和连接信息相匹配
- 检查数据包的序列是通道端期望收到的下一个序列（对于有序的通道）。
- 检查超时高度是否已过
- 检查在传出账本的状态中是否包含数据包承诺的证明
- 在数据包独有的存储路径上设置不透明的确认值（如果确认是非空的或通道是无序的）。
- 增加与通道端相关的数据包接收序列（对于有序的通道）。

a) Acknowledgements

acknowledgePacket函数被一个模块调用，以处理先前由调用模块在通道上向对手方账本上的对手方模块发送的数据包的确认。 acknowledgePacket还清理了数据包的承诺，因为数据包已经被接收并采取行动，所以不再需要。

调用模块可以在调用 acknowledgePacket 的同时，原子化地执行适当的应用程序确认处理逻辑。

IBC处理程序依次执行以下步骤:

- 检查通道和连接是否开放以确认数据包。
- 检查调用模块是否拥有发送端口
- 检查数据包的元数据是否与通道和连接信息相匹配
- 检查数据包是否真的在这个通道上被发送
- 检查数据包序列是通道端期望确认的下一个序列（对于有序的通道）。
- 检查数据包确认数据在接收账本状态下的包含证明
- 删除数据包的承诺（清理状态，防止重放）。
- 增加下一个确认序列（对于有序通道）。

7) Timeouts

应用程序的语义可能需要一些超时：分类账在认为是错误之前等待交易处理的时间的上限。由于两个账本有不同的本地时钟，这是一个明显的双重消费的攻击载体--攻击者可能会推迟收据的转发，或者等到超时后再发送数据包--所以应用程序不能安全地自己实现天真的超时逻辑。为了避免任何可能的 "双重消费 "攻击，超时算法要求目标账本正在运行并可到达。超时必须在接收方账本上得到证明，而不仅仅是在发送方账本上没有回应。

a) Sending end

超时包（timeoutPacket）函数由最初试图向对方模块发送数据包的模块调用，此时对方账本上的超时高度或超时时间戳已过，但数据包没有被提交，以证明该数据包不能再被执行，并允许调用模块安全地执行适当的状态转换。

调用模块可以在调用timeoutPacket的同时原子化地执行适当的应用超时处理逻辑。

IBC处理程序依次执行以下步骤:

- 检查通道和连接是否对超时数据包开放
- 检查调用模块是否拥有发送端口
- 检查数据包的元数据是否与通道和连接的信息相符
- 检查该数据包是否真的在该通道上被发送
- 检查证明该数据包没有在目的地账本上得到确认
- 检查目的地账本是否超过了超时高度或时间戳的证明
- 删除数据包的承诺（清理状态并防止重放）。

在有序信道的情况下，如果有数据包超时，timeoutPacket会自动关闭该信道。无序的通道在面对超时的数据包时应该继续。

如果后续数据包的超时高度之间的关系被强制执行，可以对超时数据包之前的所有数据包进行安全的批量超时。

b) Timing-out on close

如果一个信道被关闭了，飞行中的数据包就永远不能被接收，因此可以安全地超时。timeoutOnClose函数被一个模块调用，以证明一个未接收的数据包所在的通道已经关闭，所以数据包将永远不会被接收（即使还没有达到timeoutHeight或timeoutTimestamp）。然后可以安全地执行适当的应用特定的逻辑。

c) Cleaning up state

如果不写回执（因为在这种情况下处理回执会清理状态），cleanupPacket可以被模块调用，以便从存储中删除收到的数据包承诺。接收端必须已经处理了该数据包（无论是正常的还是过了超时的）。

在有序信道的情况下，cleanupPacket通过证明接收序列已通过另一端的数据包序列来清理有序信道上的数据包。

在无序通道的情况下，cleanupPacket通过证明相关的确认已被写入来清理无序通道上的一个数据包。

D. Relayers

中继器算法是IBC的 "物理 "连接层--非账本进程负责在运行IBC协议的两个账本之间中继数据，方法是扫描每个账本的状态，构建适当的数据报，并在协议允许的情况下在对面的账本上执行这些数据报。

1) Motivation

在IBC协议中，一个分类账只能记录向另一个分类账发送特定数据的意图，它不能直接访问网络传输层。物理数据报中继必须由能够访问传输层（如TCP/IP）的非分类账基础设施来执行。本标准定义了中继器算法的概念，可由具有查询分类账状态能力的非分类账进程执行，以执行这种中继。中继器是一个非账本进程，有能力利用IBC协议读取账本的状态并向一些账本集提交交易。

2) Properties

- IBC没有确切的一次或交付或超时安全的适当联系，这取决于中继者的行为（假定是拜占庭中继者）。
- IBC的小包中继有效性属性仅取决于至少一个正确的、活的中继者的存在。
- 中继可以安全地实现无许可，所有必要的验证都由账本本身执行
- IBC用户和中继者之间的必要通信被最小化
- 核心协议中不包括对中继者激励的规定，但在应用层可以实现。

3) Basic relayer algorithm

中继器算法是在实现IBC协议的一组分类账上定义的。每个中继者不一定能从多账本网络中的所有账本中读取状态和写入数据报（特别是在许可或私有账本的情况下）--不同的中继者可以在不同的子集之间进行中继。每隔一段时间，尽管在任何一个分类账上不超过每个区块一次，中继者会根据两个分类账的状态，计算所有有效数据报的集合，从一个分类账中继到另一个分类账。中继者必须事先了解他们要中继的分类账中IBC协议的哪个子集是由分类账实现的（例如通过阅读源代码）。数据报可以作为单个交易单独提交，如果分类账支持，也可以作为单个交易原子提交。不同的中继者可以在不同的分类账之间进行中继--只要每对分类账至少有一个正确的、活的中继者，并且分类账保持活的，网络中分类账之间流动的所有数据包最终都会被中继。

4) Packets, acknowledgements, timeouts

a) Relaying packets in an ordered channel

有序通道中的数据包可以以基于事件的方式或基于查询的方式进行中继。对于前者，中继器应该观察源分类帐中每当发送数据包时发出的事件，然后使用事件日志中的数据组成数据包。对于后者，中继器应定期查询源分类账上的发送序列，并保留最后转发的序列号，因此，介于两者之间的任何序列都是需要查询的数据包，然后进行转发。在这两种情况下，随后，中继器进程应该通过查询接收序列来检查目的地分类账是否还没有收到数据包，然后进行中继。

b) Relaying packets in an unordered channel

无序通道中的数据包最容易以基于事件的方式进行转发。中继器应该观察源分类账在发送数据包时发出的事件，然后使用事件日志中的数据组成数据包。随后，中继器应该通过查询数据包的序列号是否存在确认来检查目的地分类账是否已经收到数据包，如果还没有确认，中继器就应该中继该数据包。

c) Relaying acknowledgements

确认可以最容易地以一种基于事件的方式进行中继。中继器应该观察目的地分类账在收到数据包和写入确认时发出的事件，然后利用事件日志中的数据组成确认，检查数据包的承诺是否仍然存在于源分类账中（一旦确认被中继，它将被删除），如果是，则将确认中继至源分类账。

d) Relaying timeouts

超时中继稍微复杂一些，因为当数据包超时时，并没有发出具体的事件--只是数据包不能再被中继了，因为超时高度或时间戳在目的账本上已经过了。中继过程必须选择跟踪一组数据包（可以通过扫描事件日志来构建），一旦目的地分类账的高度或时间戳超过被跟踪的数据包，就检查该数据包的承诺是否仍然存在于源分类账上（一旦超时被中继，它将被删除），如果是，就向源分类账中继一个超时。

e) Ordering constraints

在中继器过程中，有一些隐含的顺序限制，决定了哪些数据报必须以什么顺序提交。例如，在数据包可以被中继之前，必须提交一个头，以最终确定存储的共识状态和轻客户端中特定高度的承诺根。中继者进程负责经常查询他们所中继的分类账的状态，以确定什么时候必须中继。

f) Bundling

如果主机分类帐支持它，中继器进程可以将许多数据报捆绑成一个交易，这将导致它们被依次执行，并摊销任何开销成本（例如支付费用的签名检查）。

g) Race conditions

在同一对模块和分类账之间中继的多个中继者可能试图同时中继同一个数据包（或提交同一个头）。如果两个中继者这样做，第一个交易将成功，第二个将失败。中继者之间或发送原始数据包的行为者与中继者之间的带外协调是必要的，以减轻这种情况。

h) Incentivisation

中继过程必须能够访问两个分类账上的账户，并有足够的余额来支付交易费用。中继者可以采用应用层面的方法来收回这些费用，例如在数据包数据中包括对自己的小额付款。

任何数量的中继器进程都可以安全地并行运行（事实上，预计独立的中继器将服务于多账本网络的独立子集）。然而，如果他们多次提交相同的证明，可能会消耗不必要的费用，所以一些最小的协调可能是理想的（例如将特定的中继器分配给特定的数据包，或扫描待处理交易的内存池）。

V. Usage patterns

A. Call receiver

对IBC处理程序的功能至关重要的是一个与运行在同一分类账上的其他模块的接口，这样它就可以接受发送数据包的请求，并可以将传入的数据包路由到模块。这个接口应该尽可能的小，以减少实施的复杂性和对主机分类账的要求。

由于这个原因，IBC的核心逻辑使用了一种只接收的调用模式，与直观的数据流略有不同。正如人们所期望的，模块调用IBC处理程序来创建连接、通道和发送数据包。然而，IBC处理程序在收到另一个分类账的数据包时，不是选择并调用适当的模块，而是模块本身必须在IBC处理程序上调用recvPacket（同样地，接受通道创建握手）。当recvPacket被调用时，IBC处理程序将检查调用的模块是否被授权接收和处理数据包（基于包含的证明和连接/通道的已知状态），执行适当的状态更新（增加序列号以防止重播），并将控制权返回给模块或抛出错误。

IBC处理程序从不直接调用模块 虽然一开始推理起来有点反常，但这种模式有几个显著的优点。

- 它最大限度地减少了对主机分类帐的要求，因为IBC处理程序不需要了解如何调用其他模块或存储对它们的任何引用。
- 它避免了在处理程序状态下管理一个模块查找表的必要性。
- 它避免了处理模块返回数据或失败的必要性。如果一个模块不想接收一个数据包（也许在上面实现了额外的授权），它就不会调用recvPacket。如果路由逻辑是在IBC处理程序中实现的，处理程序将需要处理模块的失败，这对解释是很棘手的。

它也有一个明显的缺点：如果没有一个额外的抽象，中继器的逻辑就会变得更加复杂，因为账外中继器进程将需要跟踪多个模块的状态，以确定何时可以提交数据包。

出于这个原因，分类账可以实现一个额外的IBC "路由模块"，它暴露了一个调用调度接口。

B. Call dispatch

对于常见的中继模式，可以实施一个 "IBC路由模块"，它维护一个模块调度表并简化中继者的工作。

在调用调度模式中，数据报（包含在主机分类帐定义的事务类型中）被直接转发到路由模块，然后路由模块查找适当的模块（拥有数据报被寻址的通道和端口）并调用适当的函数（必须事先在路由模块注册）。这允许模块避免直接处理数据报，并使其更难意外地搞砸必须与发送或接收数据包一起发生的原子状态转换执行（因为模块从未直接处理数据包，而是暴露了路由模块在收到有效数据包后调用的函数）。

此外，路由模块可以实现默认的握手数据报处理逻辑（代表模块接受传入的握手），这对那些不需要实现自己的自定义逻辑的模块来说是很方便的。

VI. Example application-level module

该部分规定了数据包结构和状态机处理逻辑，用于在独立分类账的两个模块之间通过IBC通道传输可替换的代币。所提出的状态机逻辑允许安全的多账簿面额处理和无权限通道开放。这个逻辑构成了一个 "可替换代币转移的桥梁模块"，在IBC路由模块和主机分类账上的现有资产追踪模块之间进行对接。

1) Motivation

通过IBC协议连接的一组分类账的用户可能希望在另一个分类账上使用一个分类账上发行的资产，也许是为了利用额外的功能，如交换或隐私保护，同时保留与发行分类账上的原始资产的可替代性。这个应用层协议允许在与IBC相连的分类账之间转移可替换的代币，其方式是保留资产的可替换性，保留资产所有权，限制拜占庭故障的影响，并且不需要额外的许可。

2) Properties

- 保存可替代性（双向挂钩）。
- 保存总供应量（在单一来源的分类账和模块上保持不变或通货膨胀）。
- 无权限的代币转移，不需要将连接、模块或面额列入白名单
- 对称性（所有分类账实现相同的逻辑）
- 故障遏制：防止由于分类账B的拜占庭行为而导致源自分类账A的代币出现拜占庭式的通货膨胀（尽管任何向分类账B发送代币的用户可能面临风险）。

3) Packet definition

只需要一个数据包类型，即FungibleTokenPacketData，它指定面额、金额、发送账户、接收账户，以及发送分类账是否为资产来源。

```
interface FungibleTokenPacketData {
    denomination: string
    amount: uint256
    sender: string
    receiver: string
}
```

确认数据类型描述了传输是成功还是失败，以及失败的原因（如果有的话）。

```
interface FungibleTokenPacketAcknowledgement {
    success: boolean
    error: Maybe<string>
}
```

4) Packet handling semantics

协议逻辑是对称的，因此源自任何一个分类账的面额都可以在另一个分类账上转换为凭证，然后稍后再赎回。

- 当作为源分类账时，桥模块在发送分类账上托管现有的本地资产面额，并在接收分类账上铸造凭证。
- 当作为汇分类账时，桥模块在发送分类账上烧毁本地凭证，并在接收分类账上解除对本地资产面额的抵押。
- 当数据包超时时，本地资产将被解押回发送方，或将凭证适当地铸回发送方。
- 确认数据用于处理失败，如无效的面额或无效的目标账户。返回失败确认比中止交易更可取，因为它更容易使发送分类账根据失败的性质采取适当的行动。

5) Fault containment

这种实现方式保留了可替代性和供应。如果代币已经被发送到对手方分类账，它们可以在源分类账上以相同的面额和金额被赎回。赎回，并以相同的面额和金额在源分类账上兑换。两个分类账上特定的解锁代币的综合供应量是恒定的，因为每个发送-接收数据包对锁定和铸造的 相同的数量（尽管特定资产的源分类账可以改变本协议范围之外的供应）。

账本可能会以两种方式之一未能遵循这里概述的可替换传输令牌协议：运行共识算法的全部节点可能与轻型客户机发生分歧，或者账本的状态机可能不正确地实现托管和凭证逻辑（无论是无意的还是有意的）。共识分歧最终应该导致错误行为的证据，可以用来冻结客户端，但可能不会立即这样做（而且不能保证这种证据会在更多数据包之前提交），所以从协议隔离故障的目标来看，这些情况必须以同样的方式处理。无法保证资产的恢复--选择将代币转移到分类账的用户要承担该分类账失败的风险--但遏制逻辑可以很容易地在接口边界上实现，即跟踪每种资产的进出供应，并确保不允许任何分类账为超过其最初托管的代币赎回凭证。实质上，特定的通道可以被视为账户，通道另一端的模块不能花费超过它所收到的。由于多账本可互换代币转移系统的孤立拜占庭子图将无法转移出比他们最初收到的更多的代币，这防止了任何源资产的供应膨胀，并确保用户只承担他们有意连接的账本的共识风险。

6) Multi-ledger transfer paths

该协议不直接处理 "钻石问题"，即用户将源自分类账A的代币发送到分类账B，然后再发送到分类账D，并希望通过D->C->A的路径返回--由于供应被跟踪为分类账B所有（并且凭证面值将是"{portD}/{channelD}/{portB}/{channelB}/denom"），分类账C不能作为中间人。由于上文所述的故障遏制要求，这一点是必要的。长赎回路径产生的复杂性可能导致网络拓扑中出现中央账本或自动市场来交换不同赎回路径的资产。

为了跟踪在分类账网络中以各种路径移动的所有面额，对于一个特定的分类账来说，实施一个注册表可能会有帮助，该注册表将跟踪每个面额的 "全球 "源分类账。终端用户服务提供商（如钱包作者）可能希望整合这样一个注册表，或保留他们自己的规范源分类账和人类可读名称的映射，以改善用户体验。

VII. Testing & deployment

区块链协议的完整版本已经在Cosmos SDK[13]中用Go实现，在Rust[14]中的实现正在进行中，未来还计划在其他语言中实现。Go[15]中也实现了一个非账本中继器守护程序。区块链游戏[16]，一个对初始软件发布的实时测试，目前正在进行中。超过一百个模拟区（独立的共识实例和分类账）已经成功地连接在一起[17]。

计划在今年夏天晚些时候向宇宙网络发布和部署产品。由于IBC是一个无权限的、选择加入的协议，采用IBC将取决于分类账自愿选择全部或部分地支持该规范。IBC的采用不需要连接到Cosmos Hub，不需要使用任何特定的令牌，甚至不需要使用任何其他的Cosmos软件--IBC可以在其他状态机框架（如Substrate[18]）之上实现，或者由独立的分类账使用自定义逻辑实现--遵守正确的协议对于成功的互操作既必要又充分