SDAG白皮书

第一章 区块链的现状

区块链是可信生产关系革命，其安全、透明、可追溯、不可篡改的特性成为一切可信商业的根基。在未来的商业体系中，区块链的作用举足轻。

但目前，区块链应用范围很窄。常见的应用基本是放置型应用，而不是即时型应用。在日常生活中，即时是很重要。刷公交卡要有即时反馈，支付要有即时提示。

目前大部分区块链，支付后并不能即时返回支付结果。是成功还是失败，需要经过一段时间的确认。而这个确认时间，对于目前主流商业而言，速度是远远不够的。

共享单车如果支付后需要等待一分钟确认，会失去大量用户。区块链速度上的短板，严重限制了区块链的应用领域。

在这种情况下，SDAG以出色的10000+TPS、秒级确认，实现了即时支付。可在支付的一刹那确认并得到反馈。因此适用领域大大加强。SDAG凭借遥遥领先的性能，成为构建下一代区块链的通用基础设施。

第二章 SDAG特点

SDAG是基于有向无环图技术全新设计的高性能分布式共识系统，支持高并发，秒级确认，无需挖矿，增加单元无需等待。SDAG立足于下一代区块链技术，彻底解决了传统区块链的扩容问题和回退问题。

SDAG的每个单元通过引用之前的一个或多个单元的哈希值相互关联，多个单元之间的引用形成一个有向无环图。在验证单元的同时，建立起他们的全局顺序关系。系统中没有一个中心化的实体负责管理或协调单元的进入，每个用户只要按照规则构建单元并且对其签名，就可以向SDAG网络中广播新单元。 随着新单元的不断加入，早期单元会收到越来越多的后续单元直接或间接的确认。

SDAG本身是一套防篡改去中心化共识系统，可以存储任意事件。由于建立了任意事件的全局序共识，因此可以基于此开发任何可能的业务，比如发行资产，交易，存证和溯源等，包括体现价值转移的数据，例如货币、产权、负债、股权等。其模块化设计方便用户像搭积木一样搭建自己的区块链业务系统。SDAG插件丰富，拥有完备的工具集，可以对各行各业的应用场景进行广泛地支持。

全局序通过在有向无环图中计算出一条主链来建立，主链是由被称为公证人的用户所创建的公证单元推导出来的。一个单元被主链越早包含，则认为他的全局序越早。用户增加一个单元后，随着后续公证单元的不断确认，这个单元最终会被最新推进的主链确认，一旦被主链确认，这个单元就会达到稳定，并且获取全局序，这是一个确定性的过程，没有概率性的回退问题。

传统的区块链是链式结构的，生成一个区块有严格的时间限制，由于链式结构，导致很多生成的区块无法并发的增加到系统中，而且基于POW的共识会造成极大的能源浪费，当发生网络分区时，还会造成概率性的数据回退。基于有向无环图的共识算法解决了这些问题。任何用户创建的单元可以并发进入系统，没有时间限制，满足实时上链，没有昂贵的挖矿过程，并且共识是确定性的。

SDAG采用RUST语言开发，采用最新的协程技术和无锁数据结构优雅的解决了高并发的难题，可以使系统资源达到最大化的利用。同时RUST语言的高性能和安全性，也保证了SDAG整体的稳定性和安全性。

第三章 SDAG基础概念

对父单元的引用是建立单元排序的基础，这些引用关系同时也生成了整个区块链结构。由于我们尽量的引用多个父单元，因此可以快速的收敛并发的情况。每个单元都可能会有多个的父单元和多个子单元。我们沿着父子关系在历史上的演进可以观察到，当一个单元被多个新产生的单元引用时就会产生分叉，而当一个单元引用多个早期单元时又会看到收敛。这种结构在图论中称为有向无环图或DAG。

在SDAG图里，有些单元是由公证人创建的**公证单元**，有些则是由用户创建的**普通单元**。每个普通单元都包括了若干条事件记录。不同于传统区块链技术的链式结构，SDAG采用图的结构来保存各种事件。这些事件按照特定的共识算法形成了一种全局顺序，也就是说，每个事件都可以计严格计算出唯一的全局逻辑时序，任何两个不同的事件我们都可以通过全局序来判定他们发生的先后顺序。

在SDAG网络中，参与的主体由三种角色构成，HUB，公证人和客户端。他们可以向SDAG网络并发的独立的提交签名单元。形成的单元之间相互验证，相互确认，随着数据的不断推进，所有发生的历史越来越难以篡改。不同于比特币，SDAG中不存在算力攻击，攻击者不可能改变其他人已经确认的历史事实。

单元

在有向无环图中的每一个节点，我们称之为一个**单元**。有向无环图中的每个边，我们都称之为一次**引用**，它只能有一个方向，指向一个比它更早生成的**父单元**。每一次引用都会形成对之前历史的一次确认。那些验证失败的单元不会被引用和广播，因此最终也不会被包含到有向无环图中。SDAG中第一个被创建出来的单元称为**创世单元**，它是由所有的公证人共同签名产生的，通常用字母G来表示。从图中任意一个单元出发，沿着父单元向过去回溯，最终都会到达创世单元。一个单元选择的父单元一定是经过验证的单元，验证失败的单元无法成为父单元。如果一个单元选择了一个验证失败的单元作为父单元，那这个单元本身也不能通过其他人的验证。一个单元允许选择多个父单元，目前我们允许的最大值为16个父单元，选择的这些父单元必须是不同的作者，且这些父单元之间不能相互包含。

如果一个单元A沿着父亲方向回溯能够到达另外一个单元B，我们称单元A包含单元B。显然，所有的单元均包含创世单元。单元A包含单元B，单元A引用单元B，单元B是单元A的祖先，单元A是单元B的子孙，这些表述都是相同的意思。

那些没有被任何人引用的单元，我们称之为**顶点单元**。所有的顶点单元共同决定了一个有向无环图的形状。一个单元引用多个父单元，本质上是对这些父单元所代表的历史的一次总结，从这个意义上来说，所有的顶点单元包含了这个有向无环图的全部历史信息。

一个单元包含了以下的**基本信息**:

- 单元的作者和签名
- 选择的父单元的列表
- 创建该单元时所看到的当前稳定主链单元
- 签名的事件列表
- 版本信息及其他

除了这些基本信息以外，单元还会推导出以下一些**属性信息**（具体参考下面的名词解释）：

- Level
- 见证级别 WL
- 最小见证级别 MIN_WL
- 最优父单元 BP
- 所属的主链序 MCI
- 最近的包含主链序 LIMCI
- 在所属主链序中的子序 SUB_MCI
- 最终稳定后的哈希 BALL

当用户向SDAG增加数据时，他会创建一个相应的单元并广播扩散给其他节点。我们只需要简单的广播单元的基本信息即可，不需要挖矿，不需要许可，随时发送。当新增的单元很少时，SDAG图看起来很像一个链，只有很少的分叉并很快收敛。当大量并发单元同时上链时，SDAG图看起来就会变的复杂起来，不停的发散，又不停的收敛。

就像在传统区块链一样，每个新的区块都可以确认之前的所有区块（包括区块中的交易），SDAG中的每个单元都会确认他的所有祖先单元。如果一个人尝试修改一个单元，必然的要修改单元的哈希值，这势必会破坏引用该单元的所有子孙单元的的签名和哈希值，因为他们都依赖于该单元的哈希。因此，如果不能跟所有子孙单元协作或者直接偷取他们的密钥，修改一个单元是不可能的。一旦一个单元广播到网络上，并且其他用户开始在这个单元上构建自己的单元，把他作为祖先单元，随着图形的推进和单元的不断确认，修改这个单元所要付出的代价就会无限增长。在SDAG中对单元的引用是一种互助行为，用户构建一个单元的同时，也确认了它先前的单元。我为人人，人人为我。

HUB

HUB负责将客户端的签名交易打包，收敛当前的顶点单元，形成一个新的单元，并且将这个新单元向全网广播。HUB也会验证其他人通过网络发送过来的单元。所有的hub形成了SDAG的网络的核心。公证人发送的公证单元会通过连接的Hub进行广播，普通客户端会将自己的签名数据发向hub进行打包广播，同时还可以通过HUB来查询图形的所有相关信息，比如历史交易，账户余额等。HUB在本地保存了SDAG的全部历史记录。每个HUB都会计算自己看到的主链推进情况，从而在本地确定事件的发生顺序，正确的更新当前的业务状态。

公证人

SDAG网络的参与者中，有一些是长期建立了良好声誉，并且对维持网络健康利益攸关的组织或个人，我们称他们为**公证人**。公证人负责对接收到的未稳定普通单元进行收敛确认，并发送由自己签名的公证单元。公证单元并不包含任何业务信息，它仅仅是对先前历史的确认。公证人是一种特殊的客户端，他们保存了全部的图形信息。外界除了知道公证人的地址以外，无法确定跟踪公证人的实际网络位置，因此也进一步增强了网络的整体安全性。每个公证人会对未稳定的普通单元进行独立的验证，并且发送公证单元确认它，直到他观察到SDAG图中已经没有未稳定的普通单元。

虽然默认他们的诚实的，但只信任其中一个或部分公证人也是不合理的。我们期望他们诚实的发布公证单元，那么多数公证人都认可的历史就应该认为是真实的。所谓诚实地发布公证单元，就只有一个简单的规则，即公证人创建的公证单元必须严格顺序的包含自己上一次创建的单元。如果同一个作者发送的单元不包含自己之前发送的单元，这种行为是不被认可的，我称之为**非连续**。我们允许不超过三分之一的公证人发送非连续单元，只要系统有超过三分之二的公证人诚实公证，我们所计算的主链就可以达成共识。

客户端

客户端并不保存全部的图形信息，只是向hub发送由自己签名的事件数据。并且从hub获取可以验证的数据。客户端不必要保存SDAG所有的数据到本地数据库中，因此可以应用到网页APP中和嵌入式设备当中。

主链

SDAG共识的核心算法是推进主链。**主链**是SDAG有向无环图中按照特定规则形成的一条特殊路径。我们总能在图中找到一条包含了绝大多数公证人所创建的公证单元的主链。所有的公证人共同推动了主链的持续推进。每个单元最终都会被主链所包含，并最终**稳定**。稳定之后的单元可以确定它的唯一的全局序。除了单元的基本信息以外，稳定单元还会计算出一系列的属性值，包括MCI，LMCI，事件发生时的状态，以及一个对所有这些历史信息的哈希值（相当于当前单元视角下默克尔树的根）。而主链的计算是和公证人发送的公证单元相关的。公证人不断的发送公证单元，从而可以使得主链持续向前推进，最终使得所有的普通单元都达到稳定。有关主链推进的详细描述参见第七章。

名词解释

单元的属性意义和计算方法如下, 用C来代表当前单元，P来代表父单元

Level

Level代表单元到创世单元的最远距离。Level是图形上事件发生先后的一种逻辑时序。创世单元的Level为0。显然，一个单元的Level取决于它的最大Level的父单元。

```
C.level = Max(P.level) + 1
```

BP

最优父单元表示所有父单元中包含最新大多数公证人的单元。比较两个父单元优先级的算法如下： 首先选择WL较大的单元；如果WL相同，则优先选择公证单元；如果不能确定，则选择Level较小的单元；如果还是无法区分，则选择哈希值小的单元。

WL

见证级别：从当前单元开始沿着最优父单元方向回溯，直到遇到绝大多数的公证人，定义此时的公证单元为相对稳定单元**RP**，它的Level就是当前单元的见证级别。

```
C.wl = RP.level
```

MCI

MCI是主链序Main Chain Index的缩写。主链是图中一条包含了大多数公证单元的路径，可以一直回溯到创世单元。创世单元的MCI为0，新的主链单元MCI依次增加1。被当前主链单元包含且不被上一个主链单元包含的所有单元均属于同一个逻辑时间区间，他们拥有相同的MCI。

```
如果C也在主链上，则:
C.mci = BP.mci + 1
```

LIMCI

LIMCI表示当前单元所包含的最近主链单元的MCI，也就是所包含的主链单元中MCI的最大值。

```
C.limic = Max(P.limci)
```

MIN_WL

最小见证级别是一个推主链时使用的指标，它的计算方式如下

```
C.min_wl = RP.wl
```

SUB_MCI

在相同的MCI的所有单元中，按照他们图形上发生的先后顺序排列所形成的序号。其中Level越小的单元SUB_MCI越小，Level相同的，单元哈希越小的SUB_MCI越小，可以看到，主链单元是SUB_MCI最大的，因为它包含了所有MCI相同的其他单元。

BALL

当一个单元稳定之后，我们可以确定它所包含的事件发生后的业务状态，所有父单元此时均已稳定。由此我们可以计算一个包括其属性，状态以及父单元的最终哈希值。这个哈希值相当于对过去历史的一个摘要，类似于默克尔树的根。它是可以验证的，如果同一个单元两个节点计算出来的BALL值不相同，那么就说明它们之间没有达成共识。

LAST_BALL

为了保证单元构建的正确性和可验证性，我们需要每个单元包含其构建当时视角下最新的主链稳定单元以及对应的BALL值，这个当时的最新主链稳定单元，也叫稳定点，他就是当前单元的**LAST_BALL**。LAST_BALL不能比任何父单元的LAST_BALL更早（MCI更小），也就是说必须保证LAST_BALL要么沿着主链向前推进（MCI增加），要么不动（MCI不变），但不能向过去后退（MCI减小）。

```
C.LastBall.mci >= Max(P.LastBall.mci)
```

LAST_BALL是用户签名保护的一部分。在一个节点验证单元时，它会验证这个稳定主链单元的BALL值是否和自己的计算结果一致，如果不一致，说明共识失败了，这个单元也会被丢弃。

基本推论

- 通过以上定义，我们不难得出以下推论:

```
1. P.level < C.level
2. P.wl <= C.wl
3. P.mci <= C.mci
4. P.limci <= C.limci
```

- 如果单元A和单元B满足:

```
A.level >= B.level 或者
A.wl > B.wl 或者
A.mci > B.mci 或者
A.limci > B.limci 或者
```

则单元A不包含单元B

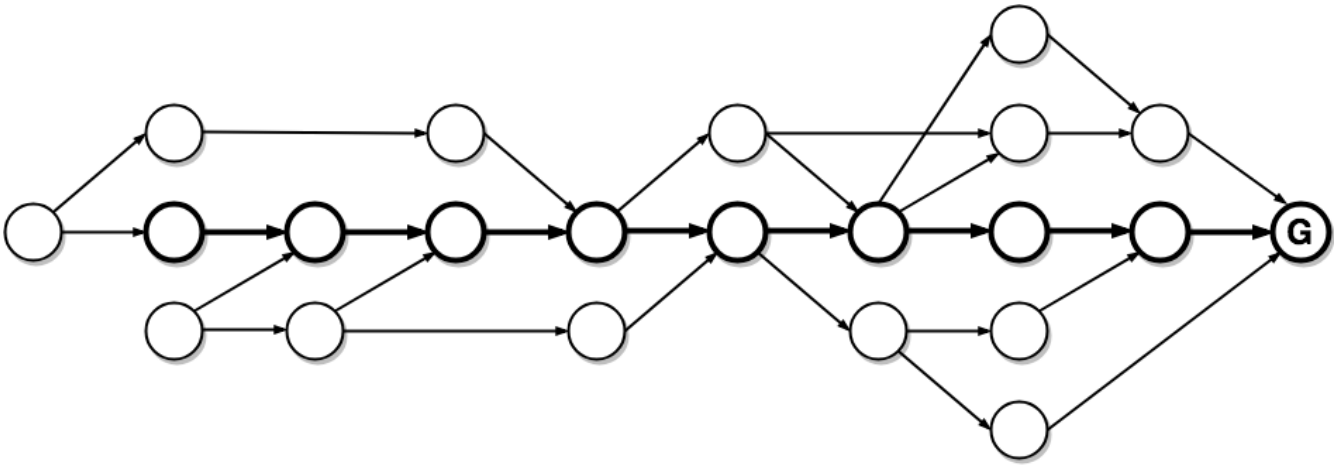
第四章 SDAG共识

概述

SDAG网络中每个节点都维护着自己本地的图，图中包含了该节点所收到的所有的单元信息以及这些单元之间的引用关系，根据图中的信息，节点可以得出这个图中所有单元即所有交易的全局顺序，进而可以验证这些交易的有效性。例如存在两个双花的交易，那就只需要比较两个交易的全局顺序，顺序靠前那笔交易被确认为有效，之后的那笔交易无效。因而SDAG共识的目标就是在SDAG网络中所有节点中，关于所有单元的全局顺序达成一个确定的共识。

主链

有向无环图含有分支，因此对于不同的分支上的单元无法直接比较顺序，在此假设图中存在一条主链，从它的顶点单元可以沿着主链一直回溯到创世单元，如下图所示。



图中加粗的单元和引用表示主链

图中所有的单元可以分为

1. 在主链上；
2. 不在主链上但被主链单元包含；
3. 其它；

- 对于第1种情况的单元，从创世单元开始一直到顶点单元是可以根据引用的拓扑关系直接得出顺序。这个顺序可以用**MCI（主链序）来表示，创世单元的MCI**为0，主链上之后的单元一直到顶点单元**MCI**依次递增；
- 对于第2种情况的单元**U**，如果

  1. 某个主链单元**MCU**包含此单元；并且
  2. **MCU**之前的主链单元不包含此单元；

  该单元**U**的MCI设置为该**MCU**的**MCI**;

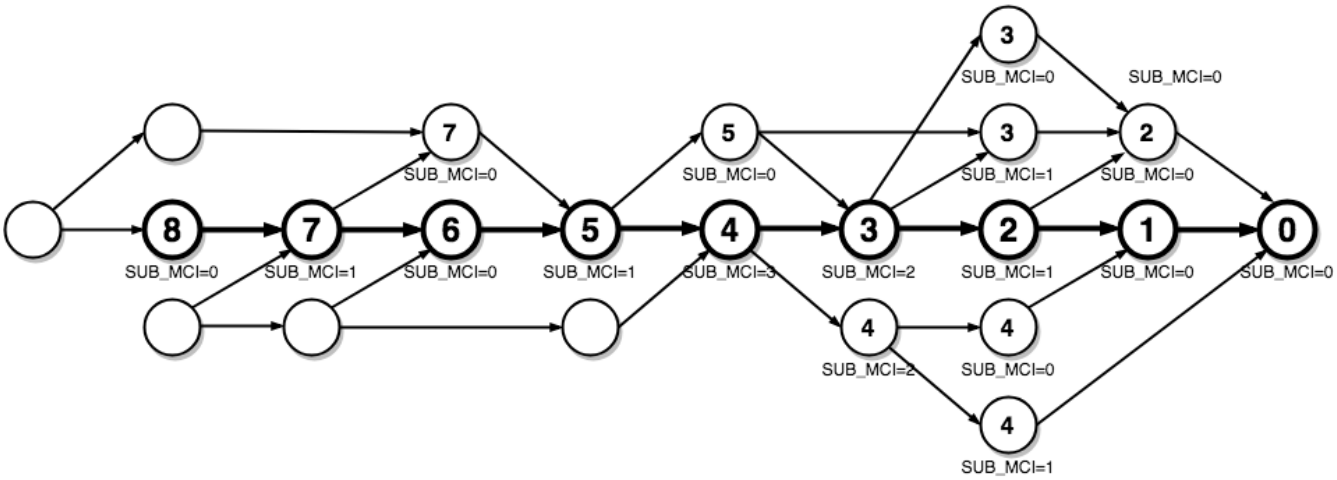
由此图中所有单元可以根据**MCI**的顺序分成若干组，对于**MCI**不同的单元就可以根据**MCI**的大小排列出顺序，而对于**MCI**相同的单元:

1. Level较小，即在图上先发生的单元胜出；
2. 如Level相等，则单元哈希值较小的单元胜出；

这个顺序由**SUB_MCI**表示，相同**MCI**中的单元中，主链单元具有最大的**SUB_MCI**

- 对于第3种情况的单元，暂时无法对其在图中的全局顺序有确定的结果。但是随着新单元进入图中，主链也会增长，这些单元也会逐渐变成第1种或者第2种单元。

上图中的关于这三类单元以及具体的**MCI**和**SUB_MCI**最终结果如下图所示。



主链推进

主链的引入使得在有向无环图中可以得到全局时序，它的推进算法则直接决定了每个节点的本地图中的全局时序。因此主链在每个节点都应该保持一致，从而保证SDAG全网中所有节点最终对于所有单元的时序达成共识。

稳定主链

根据之前对最优父单元的定义，从图中所有顶点沿着各自的最优父节点路径回溯，在经过某个单元**S**以后，这些路径会完全重合，也就是说这些路径都收敛于这个单元，那么从**S**到创世单元这一条路径，可以作为**主链**，因为它是稳定的，也称为**稳定主链**。单元**S**也叫**全局稳定点**。

- 显然，稳定主链的起点是创世单元，创世单元天然是稳定的，也就是说传世单元是最初的全局稳定点。
- S是全局稳定点，它的**MCI**也就叫**最终稳定MCI**，随着图形的增长，S会不断向前推进。

稳定单元

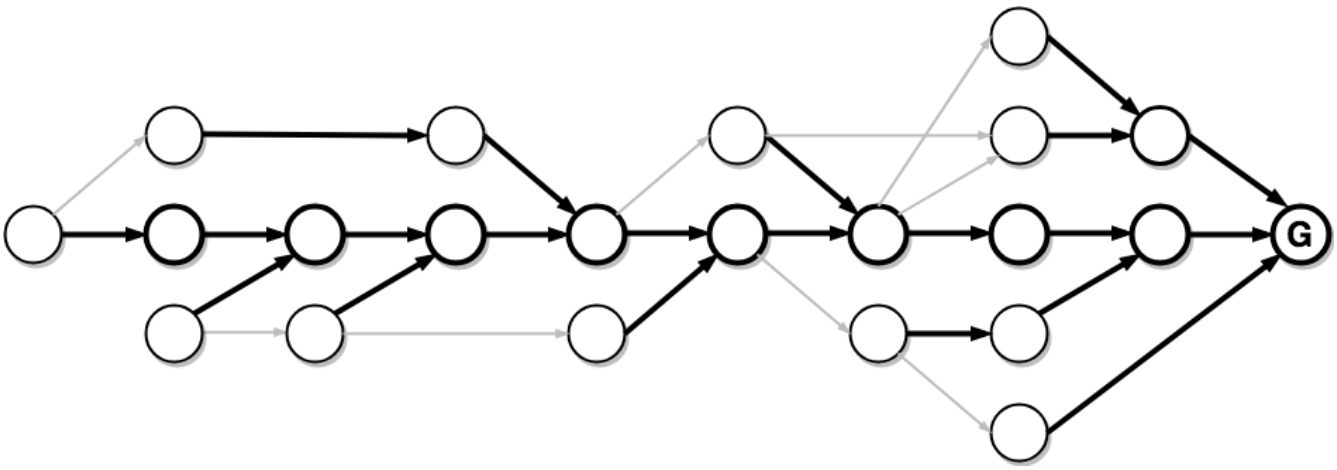
在由主链的稳定点以及它包含的所有单元所组成了一个子图，子图中的所有单元都是可以基于这个主链排出全局时序的，此时这个子图中所有的节点都是稳定的。

根据单元**MCI**以及**最终稳定MCI**的定义，整个图中所有的稳定单元满足：**MCI<=最终稳定MCI**

稳定的条件

最优子单元

最优子单元是指那些选择自己作为最优父单元的孩子单元，可能不存在，也可能有多个。从一个单元出发，沿着孩子方向，最优子单元会形成一颗树，如下图所示。



相对稳定

取主链稳定点S的最优子单元树中被一个后续单元U包含的子集，考虑以下两种情况：

1. 最优子单元树的根只有一个分支B1, U自然是这个分支的叶子节点
    - 在B1上，由U出发向根的方向回溯，收集到多数公证人集合W的公证单元后，记录该单元U，
    - 鉴于公证人发的公证单元必须包含自己之前发的公证单元，集合W中的公证人再发的所有公证单元的WL的最小值就是单元U的WL，称为这个分支的MIN_WL
    - 即便未来有新的分支B2，即使得到剩下全部公证人的支持，B2上所有单元中最大的WL都小于S.Level
    - 此时如果当前仅有分支的MIN_WL > S.Level，假设未来有单元包含了B1和B2，最优父单元还是会指向B1，也就意味新单元的最优父单元路径肯定会经过B1的下一个子单元
2. 还有其它备选分支

- 依旧由U出发回溯，并在此分支B1上计算出MIN_WL
- 考虑其它备选分支的所有单元集合为C，在这个集合里任意单元即使得到剩下全部公证人的支持，可能产生的最大的WL就是C中所有WL增加单元的Level的最大值，称为**MAX_ALT_LEVEL**
- 跟第一种情况类似，此时如果B1的MIN_WL > MAX_ALT_LEVEL，新单元的最优父单元路径肯定会经过B1，主链稳定点就可以先前推进到B1的下一个子单元

也就是说，当一条路径得到足够的公证人的支持时，主链稳定点只能沿着这条路径向前推进，其他的可能路径都别确定性的排除掉了。

由此得到判断单元稳定步骤：

1. 计算U所在分支的MIN_WL，即U.MIN_WL
2. 计算S的MAX_ALT_LEVEL

   a. 若没有备选分支

   ```
   MAX_ALT_LEVEL = S.level
   ```

   b. 若有备选分支

   ```
   MAX_ALT_LEVEL = max { C.level | C.WL_increased }
   ```

   这个过程中，计算C这个集合，可以用对向搜索的算法。
3. 如果`U.MIN_WL > S.MAX_ALT_LEVEL`，则S相对于U是稳定的。

主链稳定点推进

主链稳定点的推进就是重复的判定当前主链稳定点的下一个最优子单元是否相对于最新单元稳定，该子单元必须处于沿着最优父单元回溯的路径上，如果成立则表示主链的稳定点可以推进到这个最优子单元。我们注意到只有那些MIN_WL增加了的单元才能有效推进主链，而普通单元对推进主链没有帮助。也就是说只有公证人单元才会促进主链的不断延伸。

在全网的各个节点当中，每个节点本地的图是不完全一样的，因此主链的稳定点也在不同的位置，但是从长远来看这是一个最终一致的情况。

BALL

在有向无环图中，所有历史信息的不可篡改是由引用所有父单元的哈希来保证的，但是这部分只包含了图中基本拓扑关系的共识，而没有包含我们在图中附加的用于确认全局时序的主链及其相关信息，也就是单元的属性信息。

因此，在单元稳定也就是单元在图中的最终时序确认以后，将这些信息整合于**BALL**中。**BALL**是一个哈希值，类似于默克尔树的根，包括所有父单元引用，MCI，单元最终状态等与主链和最终全局时序相关的信息，如果这些属性不同，生成**BALL**的值也会不同。

于是生成单元时，在每个单元里加入LAST_BALL_UNIT和LAST_BALL，分别表示此时主链的稳定点和该稳定点的**BALL**，也就是在这个单元的视角下，它能看到的所有历史中可确定的全局时序的浓缩。

相应的，在验证一个单元U时，必须验证该单元中的LAST_BALL跟本节点对LAST_BALL_UNIT的**BALL**值一致，这样才能保证所有节点之间对于已经确定时序的历史的共识。而计算LAST_BALL_UNIT的**BALL**值，就需要LAST_BALL_UNIT相对于该单元的最优父单元相对稳定，也就是说从该单元的最优父单元能看到的历史子图中能推导出LAST_BALL_UNIT是主链单元，进而才能对比验证。这个约束条件进一步限制了攻击者随意发送单元的可能性。

结论

主链的推进也就是全局稳定点的推进。整个共识算法只涉及到图的算法，没有传统PoW的无用计算，共识算法可以非常快。

第五章 SDAG业务模型

业务最终状态是每个事件状态变化的和，如果我们可以确定每个事件的发生的先后顺序，则当前状态可以表达为事件造成影响的累积

```
S = Si + ∑△s
```

其中每一个事件造成的转态变化用△s表示，Si是初始化的状态。

临时状态空间和稳定状态空间

对于已经达到稳定的单元来说，我们可以确定事件的全局顺序，因此计算业务的最终状态比较容易。我们把此时修改的状态空间称为稳定状态空间，SSS（Stable State Space）。

对于未稳定的单元来说，由于其全局序没有确定，所以不能直接修改状态空间。但我们依然需要某种算法来阻止非法的状态改变，比如余额不足的账号不能成功发送交易。为了拦截这类非法的事件，我们引入了临时状态空间，TSS（Temp State Space）。

在单元未稳定时，我们修改的是TSS，其验证是并行的，无序的。这样虽然极大地提高效率，但是可能会有少数非法单元，甚至双花单元通过校验。虽然之后这些单元会被主链算法设为稳定状态，但它在之后的入SSS验证阶段，验证是严格按照顺序进行检查的，此时非法单元一定会被检测到。例如，两个双花单元A、B，在TSS验证阶段，A被认为是合法的，B被认为是非法的，由于在这一阶段是并行处理的，因此到其最终稳定顺序不确定，这就影响了A、B的最终状态，他们在稳定状态空间的状态可能和在临时状态空间的状态完全不同。

值得注意的是，SDAG是不可回退的，确定性的，所以保留单元的全部历史状态空间是不必要的，我们仅仅需要保持当前的历史状态。基于此，SDAG的状态空间仅仅保存当前的状态，对状态空间的历史改变不会保留，这就极大地节省了状态空间的存储大小。

状态空间一致性

临时状态空间和稳定状态空间所做的工作基本是相同的。在稳定状态空间里，只包含已稳定单元的状态，且状态不可逆。临时状态空间既包含那些已经上链但是还没稳定的单元状态，也包括已经稳定的单元的状态。对于一个特定的地址，他的临时状态空间内的状态已经更改，但是单元校验失败，临时状态空间会被恢复到之前的状态。例如，当一个地址要花钱时，SDAG首先会更改临时状态空间的状态，然后才会更改稳定状态空间的的状态。如果某地址所有相关单元都已稳定，则该地址的临时状态空间和稳定状态空间是最终一致的。若该地址有未稳定单元，则稳定状态空间相对于临时状态空间，缺少未稳定部分。同时稳定状态空间的最终结果也会影响临时状态空间的，临时状态空间错误的处理可能会被回退，以保持跟稳定状态空间的一致。

结论

临时状态空间的引入，可以使得非法的事件得以提前处理和拦截。极大的提升了SDAG的安全性和效率。临时状态空间和稳定状态空间以接口形式接受事件，与主链共识算法完全解耦。用户完全可以定制自己的业务逻辑，结合SDAG共识算法实现自己的业务需求。UTXO作为一个基本的交易功能，正是业务在这种模型下的一种实现。