A New Ticket Price Algorithm

（https://blog.decred.org/2017/04/03/A-New-Ticket-Price-Algorithm/）

Decred 的 1.0.0 发布版将会包含 Decred Change Proposal，DCP-0001，已有分股难度算法（即票价）的替代版，如 mainnet 上第一次硬分叉投票问题的算法。这是一个主要的变动并且已经准备好被提出来处理分股难度（简写为“sdiff”）自从 2016 年二月发布以来遇到的各种问题。Decred 发布了一个完成其基本设计目标的 sdiff 算法，也就是保持票池在它的目标大小左右，但是我们已经发现了几个已有的直到进入生产使用才变得透明的算法的缺点和问题。主要问题就是一个单独的所有票在此期间都会被购买的间隔导致了一个持久的共鸣模式，非常受限的价格探索（price exploration）产生，价格摇摆浮动巨大，股份持有人在票费用上竞争。用一个新的算法代替已有的 sdiff 算法将会带来票价更小幅度的变动并且避免各种与已有算法相关的问题。一个理想 sdiff 算法的属性与以下相关：

- 池大小稳定性 pool size stability
- 价格探索 price exploration
- 简单性 simplicity
- 缓和时间 relaxation time
- 稳定状态表现 steady state behavior

下面会对这些做出详细解释。目前，项目开发者和参与者已经提出了一些 sdiff 算法（raedah，animedow 以及其他），我们会使用由 Dave Collins 开发的一个模拟工具，dcrstakesim，来评估这些算法。

Sdiff 算法的目的是什么？

为了理解一个理想的 sdiff 算法，我们必须首先理解目前的 sdiff 算法。为了保持 PoS 津贴可以随时间稳定返还，Decred 必须维持一个稳定的票池大小，所以我们设置目标大小为 40960 张票。如果票池大小增长的特别大或特别小，它就会随之改变股份持有人和网络之间的社会契约，扰乱平均时间投票的可预见性，过期票的百分比，以及返回率。为了维持目标票池大小，票价必须池大小的上升而上升，下降而下降，来刺激保持池的大小在其目标值附近。由于用户必须遵守其票价，选择购买并可能等待他们的票被挖，所以票价必须在一个间隔基础上进行调整。在下面的文章中，sdiff 间隔大小将会避免被再次讨论，只在这个地方提及一次，因为它是票购买过程产生的结果的一个限制。

当前的 sdiff 算法

当前的 sdiff 算法取每个间隔结束时池大小的历史值和每个间隔中前 20 个 sdiff 间隔购买票的数量以及先前的间隔 sdiff 为输入。间隔 N 的 sdiff 公式为：

$$\mathrm{sdiff}_N = \frac{A(\text{pool size},20) \times B(\text{ticket purchases},20)}{\mathrm{sdiff}_{N-1}}$$

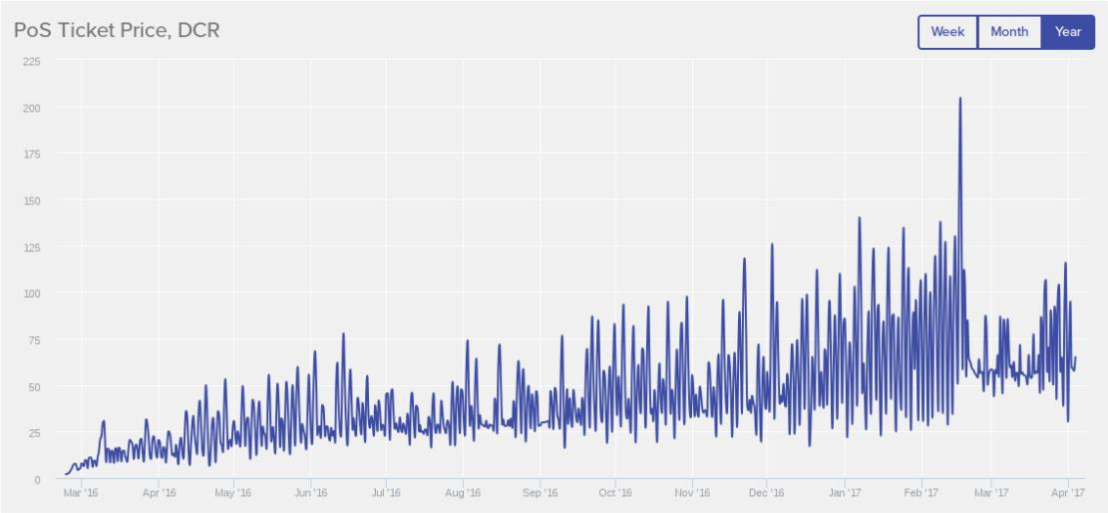
current sdiff algorithm

其中 A 是前 20 个间隔的池大小的函数，B 是前 20 个间隔购买票的函数，sdiff_N-1 是前一个间隔的 sdiff。函数 A 与池的大小大概成线性比例，只要池的大小增长，A 也增长。函数 B 基于票购买历史上下震荡，其中一个全票购买间隔给 B 发送高至 4，一个空的间隔发送低至 0.25。函数 A 和 B 都随着前 20 个间隔使用一个指数移动平均值（exponential moving average, “EMA”），意味着它们的值与过去十天的票购买和池大小历史是呈“平滑”关系的。

目前的 sdiff 满足一个 sdiff 算法过时的最低要求：它维持一个相对稳定的票池大小，票价随池大小上涨而上涨，随池大小缩减而下降，并且价格是间断的。票池大小一度上涨至

45034，但最近又降低到 41-42K 的范围内。当出现一个票的全间隔或接近全间隔时，票价跳过 2 或 3 个间隔，阻止在那些间隔购买，经常直到这些间隔完全为空的时候。不幸的是，在 2016 年二月当前版本 sdiff 算法发布之后一些问题开始随之产生。

Decred 发布之后，很快我们注意到票价已经开始在 3 到 4 个间隔时震荡。一个票的全间隔应该使得价格在 2 到 3 个间隔内保持较高水平，没有票会在那些间隔被买走，然后回到一个接近那个全间隔票的价格。到目前，这个过程已经持续超过一年了，而且是一个由于 sdiff 算法对于价格探索方面不给力而产生的紧急的共鸣状态。在一个全间隔产生之后，票价急速上涨以至于超过人们愿意为购票所支付的价格范围，然后当票价下降到可以接受的水平，它就有接近了之前全间隔时的价格。sdiff 算法的这个共鸣模式已经引起了低票价间隔的费用战争，因为在一个合理价格的票的需求量大于一个全间隔能提供的 2880 最大值。随着时间消逝，一张票的均价在缓慢上升，可以从下图看到票价趋势。



ticket price history

如果我们看 sdiff 算法的函数形式，票价发生了怎样的变化一目了然。回想一下

$$\mathrm{sdiff}_N = \frac{A(\text{pool size},20)\times B(\text{ticket purchases},20)}{\mathrm{sdiff}_{N-1}}$$

其中 A 是随池大小缓慢变化上升的函数，B 是在每个间隔震荡并在 0.25 和 4 之间变化的函数。Mainnet 的历史数据表明票池大小在下降之前上涨至 45K，这转变为函数 A 就是一个比较大的值，提高了函数 B 震荡的幅度。随着池大小上涨，票价间的摆动上升，导致价格探索的更糟糕的问题。

一个理想的 sdiff 算法

根据我们对于目前 sdiff 算法的经验，我们在寻找替代算法时已经发现一个需要考虑的理想 sdiff 算法的几个属性：

- 池大小稳定性 pool size stability
- 价格探索 price exploration
- 简单性 simplicity
- 缓和时间 relaxation time
- 稳定状态表现 steady state behavior

随后将对这些属性进行详细的解释。

池大小稳定性

目前的算法做了一个不错的工作来保持池大小的稳定，但是我们已经见过池大小在某些情况下被驱动超过 45K，这并不理想。为了在池大小很大的时候不是仅仅提升价格震荡的幅度，一个理想的 sdiff 算法要稳定地随池的大小增长而增长，保持池的大小接近其目标的大小。

价格探索

由于一个全间隔引发目前的 sdiff 跳过最多 4 个因子，所以它对于探索用户愿意支付的票价范围作用很小。如果票价的最大变化由用户愿意买他们而支付的价格范围所限制，那限制每个间隔大概 20%的最大上涨幅度的变化就会很合理，举个例子，一个全间隔带来下一个间隔的 20%的票价上涨。一个理想的 sdiff 会保持在用户实际愿意支付的票价范围之内，因为空间隔只传达出没有需求量的消息，稀疏购买的间隔给出需求的定量反馈。相似地，为了探索一个全间隔之后的票价，sdiff 应该缓慢地向下调整，给用户提供在上一个全间隔票价之上的几个价格的购买机会。这将会允许票需求量的逐渐上涨，以至于逐渐地上涨票价。

简单性

目前的 sdiff 算法需要做大量涉及到池大小和前 20 个间隔票购买历史的计算，然后被上一个间隔的 sdiff 除。该算法是复杂的，所以应该简化，使得每个人都可以更容易地理解它。目前，该 sdiff 算法是

$$\mathrm{sdiff}_N = \frac{A(\text{pool size},20)\times B(\text{ticket purchases},20)}{\mathrm{sdiff}_{N-1}}$$

而我提议一个理想的 sdiff 算法应该有如下的形式（带有函数 C）。

$$\mathrm{sdiff}_N = C\left(\text{pool size}, 2\right) \times \mathrm{sdiff}_{N-1}$$

ideal sdiff algorithm

这意味着只需要前两个间隔的池大小来计算，并且它与之前间隔的 sdiff 成线性比例。

缓和时间

当目前的 sdiff 由一个全间隔的票驱动时，它在“缓和 relax”回到其上一个状态之前需要 2 到 3 个空的间隔，也就是说它需要 2 到 3 个间隔从波动状态回到均衡状态。为了防止驱动全间隔价格的可能性导致票池的大小下降，我们有缓和时间应该小于 3 个间隔的要求。尤其是这里我们正在尝试避免的场景是紧随一个全间隔（2880 张票被购买，720 张被调用投票）之和的 3 个或更多的空间隔（没有票被购买，2160 或更多被调用投票）。这施加了一个在票价必须在一个全间隔之后回落的期间数量的限制。

稳定状态表现

稳定状态表购买，也就是每个间隔 720 张票，当前的 sdiff 没有任何人希望看到的表现：维持一个大于目标的池大小应该有一个上升的价格以及维持一个小于目标的池大小应该有一个下跌的价格。一个理想的 sdiff 会有当稳定状态票购买发生时的属性，价格在池大小高于目标时上涨，在池大小低于目标时下跌。就上面提到的 sdiff 算法的简化形式而言，这说明函数 C 应该有两个组成，一个是前两个池大小之间的 delta 值的函数，另一个是上一个池大小和目标池大小之间的 delta 值的函数。

提出的 sdiff 代替算法

一些 sdiff 算法替代方案最近由项目开发者和社区成员提出并在一个 dcrd 的 github 发行

中更新（https://github.com/decred/dcrd/issues/584）。目前该算法提出了：

- 跟踪前两个间隔之间池大小的变化的百分比以及根据那个百分比测量的票价，举个例子，票池大小在一个全间隔最多上升 2160，在一个空间隔最多下降 720，这样通过用票池大小变化量除以票池大小来衡量票价，这限制了间隔到间隔的变化最多上升 5.27%、最多下降 1.75%（raedah）。
- 选择一个在当前池大小和目标池大小之间的 delta 值的特定渐进函数，它在 delta 值小的时候大约呈线性关系，当 delta 值大的时候近似呈指数关系，并且用当前票价均值作为基准来计算一个新的票价（animedow）。
- 计算当前股份锁定的币的总数量，然后使用锁定币和目标池大小的比例以及锁定币和当前池大小的比例的线性组合作为新的票价（coblee）。

我们将会评估这些算法和其他我们这两周从社区收到的算法，发布模拟结果并选出我们认为是最好的选择。这些模拟将会是透明的，可复写的，并且届时，所有假设将会详细列出来。

通过模拟来评估

为了选出一个新的 sdiff 算法，Dave Collins 已经创建了一个模拟工具，称作 dcrstakesim（https://github.com/davecgh/dcrstakesim/），我们将使用它来评估每个算法。Dcrstakesim 在一个全新的模拟 mainnet 链上模拟每个提出的 sdiff 算法，并且它包含模仿用户购买票的逻辑作为在给定票价上的收益率函数。另外，它限制了提供买票锁定的币的百分比的上限，来确保它反射我们至今所见。至今提出这么多的算法是一个好的开始，我们会根据上述提到的理想 sdiff 的标准来评估这些算法，这就可能意味着要对已经提出算法进行额外的修改。如果你热衷于提出算法，请详见已经提出的算法或是提供建设性的意见，在 GitHub，Decred Forum 或 our Slack chat 上加入我们。