On Bitcoin Cash's Target Recalculation Functions

Juan Garay
Department of Computer Science and Engineering
Texas A&M University
garay@tamu.edu

Yu Shen*
School of Informatics
University of Edinburgh
yu.shen@ed.ac.uk

ABSTRACT

Bitcoin Cash, created in 2017, is a "hard fork" from Bitcoin responding to the need for allowing a higher transaction volume. This is achieved by a larger block size, as well as a new difficulty adjustment (target recalculation) function that acts more frequently (as opposed to Bitcoin's difficulty adjustment happening about every two weeks), resulting in a potentially different target *for each block*. While seemingly achieving its goal in practice, to our knowledge there is no formal analysis to back this proposal up.

In this paper we provide the first formal cryptographic analysis of Bitcoin Cash's target recalculation functions—both ASERT and SMA (current and former recalculation functions, respectively)—against all possible adversaries. The main distinction with respect to Bitcoin's is that they are no longer epoch-based, and as such previous analyses fail to hold. We overcome this technical obstacle by introducing a new set of analytical tools focusing on the "calibration" of blocks' timestamps in sliding windows, which yield a measure of closeness to the initial block generation rate. With that measure, we then follow the analytical approach developed in the *Bitcoin backbone protocol* [Eurocrypt 2015 and follow-ups] to first establish the basic properties of the blockchain data structure, from which the properties of a robust transaction ledger (namely, *Consistency* and *Liveness*) can be derived.

We compare our analytical results with data from the Bitcoin Cash network, and conclude that in order to satisfy security (namely, properties satisfied except with negligible probability in the security parameter) considerably larger parameter values should be used with respect to the ones used in practice.

ACM Reference format:
Juan Garay and Yu Shen. 2021. On Bitcoin Cash's Target Recalculation Functions. In *Proceedings of 3rd ACM Conference on Advances in Financial Technologies, Arlington, VA, USA, September 26–28, 2021 (AFT '21),* 13 pages. https://doi.org/10.1145/3479722.3480998

1 INTRODUCTION

While opening up a new era in the area of cryptocurrencies, Nakamoto's Bitcoin protocol [11, 12] has been critized for its heavy use of the computational resources needed by its underlying proof of work (PoW) mechanism as well as its relatively long settlement time of transactions. As a consequence, a number of alternative cryptocurrencies have been proposed with the purpose of ameliorating the above issues. One such proposal is Bitcoin Cash (BCH)[1], created in August 2017 as a "hard fork" of Bitcoin, with the original motivation of increasing the size of blocks, and thus allowing more transactions to be processed.

Due to lesser prominence and popularity, the computational "investment" on these alternate cryptocurrencies is relatively low (for example, the hashing power invested on Bitcoin Cash is approximately 5% of that on Bitcoin). Moreover, miners are able to evaluate their expected reward and rapidly switch among different blockchains in order to achieve a higher profit, giving rise to an environment where the number of participating miners on these minor blockchains may fluctuate wildly, which in turn has a direct effect on suitable difficulty (of PoWs') recalculation mechanisms[2].

The above two aspects—desired higher transaction throughput and higher participation variation—are the motivation for this work. We focus on Bitcoin Cash as a representative of a newly proposed target recalculation function, and perform a formal analysis of the protocol's security under such dynamic environment. The importance of an adequate target recalculation mechanism has already been pointed out in [5], where it is observed that if it is removed, the blockchain protocol becomes insecure in the dynamic setting *even* if all parties behave honestly, resulting in a blockchain that will diverge substantially (i.e., spawning "forks") as the number of miners keeps increasing, thus becoming vulnerable to many known cryptographic attacks. Furthermore, an inadequate target recalculation function may break the balance between miners' invested hashing power and reward, thus reducing their confidence in the system and leading them to quit the mining pool, arguably a situation that should be avoided.

Bitcoin Cash's target recalculation algorithm has gone through three stages. When created, the recalculation mechanism was a combination of Bitcoin's target recalculation function and an *emergency difficulty adjustment* (EDA) mechanism, which would suddenly enlarge targets by 25% (i.e., decrease mining difficulty by 20%) if the block generating interval of 6 blocks exceeds 12 hours.

In November 2017, this initial function was replaced by a new function called *SMA* (for *Simple Moving Average*, or "cw-144"). At a high level, SMA is analogous to Bitcoin's recalculation function in the sense that it determines the next target based on an "epoch" of blocks, except that in the new algorithm the target value is recalculated more frequently—in fact, the target for *each block* varies. Moreover, the epoch of blocks changes with every block in a "sliding window" fashion. Generally speaking, the SMA function calculates target for the next block based on the length of the epoch and the *average* target of the blocks in the epoch.

Finally, the recent (November 2020) update introduces a control-theory-inspired recalculation function called *ASERT* (for *Absolutely Scheduled Exponentially Rising Targets*) [15]), which is completely different from previous recalculation functions. ASERT is not epoch-based, and adjusts the difficulty level of each block simply based on its timestamp and height difference with the "anchor" block.

*Work done while the author was at Texas A&M University.
[1] https://www.bitcoincash.org/.
[2] As a reference, Bitcoin adjusts the PoWs' difficulty level every 2016 blocks – approximately every two weeks.

Specifically, once an anchor block is chosen, a timestamp can be computed for all the subsequent blocks according to its height and ideal block generating interval. The difficult adjustment is an exponential function of the block's timestamp deviation from its scheduled timestamp. the target changes is controlled by a "smoothing factor" m, which we will show is a crucial parameter in our analysis.

Overview of our results. Our main contribution is establishing under what conditions regarding party fluctuation Bitcoin Cash's target recalculation functions (both ASERT and SMA) achieve a steady and close to ideal block generation rate, given that they are not epoch-based, and the recalculation mechanism is invoked for every block—and further, in the case of ASERT, that it is "memoryless," meaning that the target for one block is only decided by the current timestamp and the block's height. As such, previous analyses based on the duration of an epoch no longer hold, and new analytical tools are needed.

As in prior work on dynamic environments, bounds on the ways that miners come in and drop out of the computation are necessary for things to work. We suggest a new methodology to capture how the number of parties can fluctuate. In a nutshell, our definition is comprised of two parts concerning *both* short-term and long-term participation. In such context, we first (Section 3) perform a preliminary analysis of the ASERT function to establish whether in a suitable respecting environment, the blocks in chains created according to the function will have blocks with timestamps to make the probability of producing the next blocks close to the ideal block generation rate (we will call such timestamps "good"). Through a closeness measure based on "calibrated timestamps" and probabilitic analysis we conclude that they do.

Our conclusions then serve as a crucial part of the complete security analysis (Section 4), where, following [4, 5], we present an abstraction of the protocol we term the *Bitcoin Cash backbone*, and follow the "template" of establishing two main properties of the blockchain data structure—"common prefix" and "chain quality"—which serve as an intermediate step to build a robust transaction ledger. As a result, (our abstraction of) the Bitcoin Cash protocol with chains of variable difficult using the ASERT function running in a bounded-delay network and suitably parameterized, satisfies, with overwhelming probability, the properties of consistency and liveness.

In addition (Section 5), we also provide a description and high-level analysis of Bitcoin Cash's previous target recalculation function SMA. Even though the function has now been deprecated, it provides insights and elements of comparison against ASERT with regard to party fluctuation.

Finally (Section 6), we compare our results with data from the Bitcoin Cash network, from which we extract the actual party fluctuation rate and network delay. Our main conclusion is that in order to satisfy security (namely, properties satisfied except with negligible probability) increased parameter values should be used with respect to the ones used in practice—specifically, a larger value of m (the smoothing factor in ASERT and epoch length in SMA)–concretely, $m = 432$, compared to the value $m = 288$ that is being used in ASERT (which in turn corresponds to 2 days)—should be adopted. In addition, regarding the SMA function, a larger dampening filter $\tau = 8$ in the SMA function should be used, instead of $\tau = 2$, which is the value that was last used. Lastly, our comparison with the existing Bitcoin Cash network shows that the ASERT function performs better than the SMA function under a pronounced party fluctuation.

Due to space limitations, some of the proofs, detailed protocol descriptions, and other complementary material are presented in the full version of this paper [7].

2 PRELIMINARIES

In this section we present the network model where we analyze Bitcoin Cash's target recalculation functions as well as the Bitcoin Cash protocol abstraction, as well as some basic blockchain notation. These notions and terminology follow closely [5, 6], and therefore the presentation here is succinct, except for the extension of the notion of respecting environments with respect to [6]. More formal details about the model are presented in the full version [7].

Model. We describe our protocol in the bounded-delay (aka "partially synchronous") network model considered in [6, 13], where there is an upper bound Δ in the delay (measured in number of rounds) that the adversary may inflict to the delivery of any message. The precise value of Δ will be unknown to the protocol. "Rounds" still exist in the model, but now these are not synchronization rounds where messages are supposed to be delivered to honest parties ("miners"). At any given time (round), a fraction of the parties may be "corrupted" and controlled by an adversary $\mathcal{A}$, directing them to behave in an arbitrary and potentially malicious manner. The underlying communication graph is not fully connected and messages are delivered through a "diffusion" mechanism.

As in [6], we assume a dual hash function/network functionality that is available to all parties running the protocol and the adversary, and which abstracts the access of the parties to the hash function and the network. The hash function aspect relates to the parties attempting to solve "proofs of work" (PoW) [3] during the execution of the protocol, and is modeled as parties having access to the oracle $H(\cdot)$. In our analysis, each honest party P_i is allowed to ask one query to the oracle in each round, but unlimited queries for verification. The adversary $\mathcal{A}$, on the other hand, is given at each round r a number of queries that cannot exceed t_r. The "diffusion" aspect of the communication allows the order of messages to be controlled by $\mathcal{A}$. Furthermore, the adversary is allowed to spoof the source information on every message (i.e., communication is not authenticated).

Garay *et al.* [6] refer to the setting defined by the two series $\mathbf{n} = \{n_r\}_{r\in\mathbb{N}}$ and $\mathbf{t} = \{t_r\}_{r\in\mathbb{N}}$, representing the number of "ready" honest parties and the bound on corrupted parties that may be activated at each round r, with the above bounded access to the random oracle functionality, as the *dynamic setting*. As pointed out in [5], protocol's properties cannot be satisfied under arbitrary sequences of parties, and restrictions are in order. In our model, one important difference with respect to [5, 6] is the bound on the *short-term* variation of the number of parties, in addition to the long-term bound, which results in the following definition of accepted variation terms on the number of parties[3]:

[3]Definition 2.1 is consistent with the notion introduced in [5], except that there the party fluctuation is expressed with respect to the number of honest parties. Note that

Definition 2.1. For $\gamma, \Gamma \in \mathbb{R}^+$, we call a sequence $(n_r)_{r\in\mathbb{N}}$ ($\langle\gamma,\sigma\rangle$, $\langle\Gamma,\Sigma\rangle$)-*respecting* if it holds that in a sequence of rounds S with $|S| \le \Sigma$ rounds, $\max_{r\in S} n_r \le \Gamma \cdot \min_{r\in S} n_r$ and for any consecutive sub-sequence rounds $S' \preccurlyeq S$ with $|S'| \le \sigma$ rounds, $\max_{r\in S'} n_r \le \gamma \cdot \min_{r\in S'} n_r$.

Blockchain notation. A *block* is a quadruple of the form $B = \langle r, st, x, ctr\rangle$ with $st \in \{0,1\}^\kappa, x \in \{0,1\}^*$ and $r, ctr \in \mathbb{N}$. They satisfy the predicate $\text{validblock}_q^T(B)$ defined as $(H(ctr, G(r, st, x)) < T) \wedge (ctr \le q)$ where $G(\cdot), H(\cdot)$ are cryptographic hash functions with output in $\{0,1\}^\kappa$. A *blockchain* is a sequence of *blocks*. The rightmost block is the *head* of the block denoted head(C). Note that the empty string ε is also a chain and $\text{head}(\varepsilon) = \varepsilon$. A chain $C$with head(C) can be extended by appending a valid block $B = \langle r', st', x', ctr'\rangle$ that satisfies $(st' = H(ctr, G(r, st, x))) \wedge (r' > r)$. By convention, any valid block may extend the chain $C = \varepsilon$. The *length* of a chain len(C) is its number of blocks. Consider a chain C of length ℓ and any nonnegative integer k, we denote by $C^{\lceil k}$ the chain resulting from "pruning" the k rightmost blocks. Note that for $k \ge \text{len}(C)$, $C^{\lceil k} = \varepsilon$. If C_1 is a prefix of C_2 we write $C_1 \preccurlyeq C_2$. For chains C_1 and C_2, define $C_1 \cap C_2$ to be the chain formed by their common prefix.

3 BITCOIN CASH

A salient distinction of Bitcoin Cash (in addition to the size of blocks) with respect to Bitcoin is its target recalculation function—ASERT. Intriguingly, and in contrast to SMA, this function is "memoryless," so one thing we would like to find out right away is whether such a function is able to maintain, under the conditions allowed by our model, a steady block generation rate. If that's the case, we will then proceed with the protocol abstraction and analysis to establish under what values of parameters the blockchain and ledger properties (cf. Section 4) can be satisfied. (The latter is also required for the SMA analysis.)

3.1 The ASERT Target Recalculation Function

The November 2020 Bitcoin Cash update introduced a new difficulty adjustment algorithm, called *ASERT* (for "Absolutely Scheduled Exponentially Rising Targets"), aimed at achieving a stable block generation interval and transaction confirmation time as well as reducing the advantage of non-steady mining strategies. The new algorithm is derived from the control theory literature, and, specifically, from the notion of *Exponentially Moving Average* (EMA), a type of moving average that places greater weight on the most recent data points, in contrast, to *Simple Moving Average* (SMA, the previous target recalculation function), which applies an equal weight to all the observations in the period. For additional details, refer to [15], where a mathematical derivation of this function based on exponential smoothing, a common technique for removing noise from time series data, is provided.

ASERT adjusts the target based on the *anchor block,* i.e., a block whose target is denoted by T_0, and is used as a reference for all subsequent blocks. We use $f \in (0,1)$ to denote the ideal block generation rate (and thus $1/f$ represents the ideal block generation interval). At a high level, for a given block, ASERT compares its timestamp with the scheduled timestamp, which is a product of the ideal block generating interval and the block's *height* difference (e.g., for the i-th block, it is $(i-1)/f$). If the block's timestamp is ahead of the scheduled time, which means that the number of miners is larger than that corresponding to the anchor block, ASERT decreases the target (i.e., raises the difficulty of generating a block); if it falls behind the scheduled time, then the target is increased (i.e., the difficulty is reduced).

The amount by which the target is changed is based on a value m, the *decay* or *smoothing* factor. Specifically, the target is adjusted exponentially based on the ratio of time difference and the smoothing time (i.e., m/f). For example, if the smoothing time is 2 days, then a block with a timestamp 2 days ahead of the scheduled timestamp would have a target whose value is half of the anchor target's.

Formally, ASERT is defined as follows. (Note that we assume the anchor block to be the first block and timestamps start at 0.)

Definition 3.1 (ASERT). For fixed constants m, T_0, the target calculation function $D_{\text{ASERT}} : \mathbb{Z}^* \to \mathbb{R}$ is defined as

$$D_{\text{ASERT}}(\varepsilon) = T_0 \text{ and } D_{\text{ASERT}}(r_1, \ldots, r_v) = T_0 \cdot 2^{(r_v - (v-1)/f)/(m/f)}, \quad (1)$$

where $(r_1, \ldots, r_v)$ corresponds to a chain of v blocks with r_i the timestamp of the i-th block.

Note that, as opposed to Bitcoin's target recalculation algorithm [5] and also to Bitcoin Cash's recalculation function (SMA, Section 5), ASERT is not epoch based. Moreover, and as mentioned earlier, ASERT is *memoryless*—i.e., the target for one block is only decided by the current timestamp and the block's height. No matter what timestamps the previous blocks have, they would not influence the current block's target value.

Removing the function's "dampening" filter raises the question of whether it would suffer from Bahack's raising difficulty attack [1]. It turns out that it does not, since Equation (1) intrinsically prevents the difficulty from a sudden sharp increase. Concretely, assuming monotonically increasing timestamps, even if the adversary produces m blocks with the same timestamp, he can only double the difficulty value.

3.2 ASERT: Preliminary Analysis

We now provide a preliminary analysis of the ASERT function to establish whether in a suitable environment, the blocks in chains created according to the function will have blocks with timestamps to make the probability of producing the next blocks close to the ideal block generating rate. (With foresight, we will call such timestamps "good.") Our preliminary conclusions will then serve as a crucial part of the complete security analysis, after we introduce the Bitcoin Cash protocol abstraction.

Our security parameter is hash function length κ, and we let the smoothing factor $m = \text{polylog}(\kappa)$. Our probability space is over all executions of length at most some polynomial in κ. We will denote by **Pr** the probability measure of this space.

We use n_r to denote the total number of parties at round r ([5, 6] uses n_r to denote the number of *honest* parties), and t_r to denote the number of corrupted parties. Thus, the number of honest parties at round r is $n_r - t_r$. For simplicity, we use $h_r = n_r - t_r$. This follows

the two fluctuations are related by a constant (concretely, $2 - \delta$; see Table 1). As it turns out, expressing statements in terms of honest-party fluctuation will considerably simplify them.

the tradition in the secure multiparty computation literature and the notation in [4].

Recall that in our model, each party's query bound to the random oracle (RO) per round is $q = 1$. Now suppose that at round r exactly n parties query the RO with a target T. Then the probability that at least one of them will succeed is

$$f(T, h) = 1 - (1 - pT)^h \leq phT, \text{ where } p = 1/2^\kappa.$$

We let $f_0 = f(T_0, h_0)$, where T_0 and h_0 are the initial target and estimate of number of honest parties, respectively. The objective of the target recalculation mechanism is to maintain a target T for each party such that $f(T, h_r) \approx f_0$ for all rounds r. For notational simplicity, we will drop the subscript from f_0, and will always specify the two arguments of $f(\cdot, \cdot)$ to avoid confusion.

We say round r is a target-recalculation point of a valid chain C, if there is a block with timestamp r. Recall that our goal is to show that all the target recalculation points on a chain using the ASERT function are "good." How close should the corresponding generation rate (namely, ph_rT) be?

Intuitively, the block generation rate should satisfy $f/\Gamma \leq ph_rT \leq \Gamma f$, when considered in a respecting environment with long-term party fluctuation ratio Γ (recall Definition 2.1). Moreover, adversarial parties can choose to keep silent, hence decelerate the block production process. One might consider adding an honest-party advantage, say, δ to the lower bound—i.e., $f/(2-\delta)\Gamma \leq ph_rT \leq \Gamma$. As we shall see, it turns out that this modification is not sufficient for a satisfactory security analysis, and is inadequate to absorb all the errors in our model. I.e., the adversary would be able to wait for an appropriate moment to act and then disturb the regular block production. Consequently, we need a looser bound to compensate for adversarial behavior as well as for errors introduced by the (peer-to-peer and dynamic) network. Taking this into account, we proceed to define a "good" target-recalculation point:

Definition 3.2 (Goodness). A target-recalculation point r is *good* if the target T for the next block satisfies $f/2(2-\delta)\Gamma^3 \leq ph_rT \leq 2\Gamma^3 f$.

As mentioned above, ASERT not being epoch-based means that previous analyses regarding the "goodness" of recalculation points for Bitcoin [6] (as well as for SMA—Section 5) do not hold.

We start our analysis by presenting some basic observations regarding the ASERT function. Note that the target for the next block is merely related to the block's timestamp and height. For the i-th block with timestamp r and corresponding number of honest parties h_r, it is not hard to see that if $r = (i-1)/f + (m/f)\log(h_0/h_r)$, the i-th block would have block generation rate exactly f. We call this timestamp r the *calibrated timestamp* for block B_i. In addition, r is a good target recalculation point if it satisfies

$$\frac{i-1}{f} + \frac{m}{f}\log(2(2-\delta)\Gamma^3 \cdot \frac{h_0}{h_r}) \leq r \leq \frac{i-1}{f} + \frac{m}{f}\log(2\Gamma^3 \cdot \frac{h_0}{h_r}). \quad (2)$$

We now define a new random variable to describe the deviation of timestamps: We use X_i to express by how much the i-th block deviates from its calibrated timestamp. For the i-th block with timestamp r_i and number of honest parties h_i,

$$X_1 = 0 \text{ and } X_{i+1} = X_i + (r_{i+1} - r_i) - \frac{1}{f} - \frac{m}{f}\log\frac{h_{i+1}}{h_i} \text{ for } i \geq 0.$$

The three parts in the definition of X_{i+1} are as follows: (1) $(r_{i+1} - r_i)$ represents the difference between their timestamps, (2) $1/f$ is the ideal block interval, and (3) $(m/f)\log(h_{i+1}/h_i)$ is the difference between the respective number of honest parties. For "good" blocks, variable X_i should satisfy $-(m/f)\log 2(2-\delta)\Gamma^3 \leq X_i \leq (m/f)\log 2\Gamma^3$.

As defined, X_i is sensitive to the fluctuation of number of parties. Since we can only bound the fluctuation rate during a fixed number of rounds, X_i is not suitable for the analysis. To overcome this, we consider a new random variable W_i within a $(\langle\gamma, \ell + 2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, and then show that if this new random variable satisfies certain conditions, then X_i presented above satisfies the ideal bound (2).

In more detail, we consider the calibrated timestamp $r = (i-1)/f + (m/f)\log(h_0/h_r)$, and a sliding window of $4(m/f)\log\Gamma$ rounds starting with block B_u and number of honest parties h_u. For each subsequent block in this window, we replace h_r with h_u and call $r' = (i-1)/f + (m/f)\log(h_0/h_u)$ the *relatively calibrated timestamp with respect to* B_u for i-th block $B_i, i \geq u$. We can now define a new random variable W_i expressing by how much the i-th block deviates from its relatively calibrated timestamp (wrt B_u). That is, for i-th block with timestamp r_i,

$$W_u = X_u \text{ and } W_{i+1} = W_i + (r_{i+1} - r_i) - \frac{1}{f} \text{ for } i \geq u.$$

Now the definition of the random variable only consists of two parts: The difference between their timestamps, and the ideal block interval. For good target recalculation points, W_i should satisfy

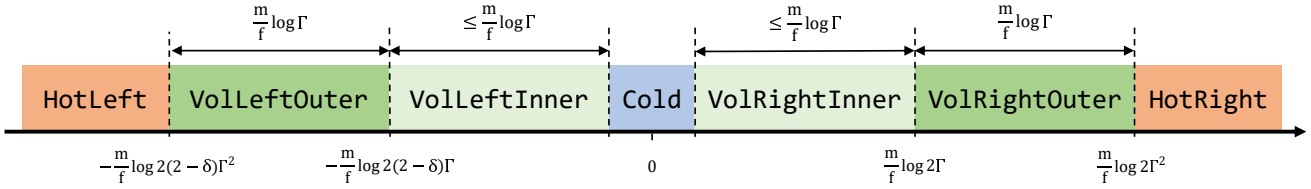
$$-\frac{m}{f}\log 2(2-\delta)\Gamma^2 \leq W_i \leq \frac{m}{f}\log 2\Gamma^2.$$

Next, we define seven states based on values of the random variable W_i. Studying the possible transitions between them allows us to establish that target recalculation points are good. Refer to Figure 1. Let $h_{\max}, h_{\min}$ denote the maximum and minimum number of parties during the sliding window, respectively.

$$\begin{aligned}
\mathsf{HotLeft}_i &\triangleq W_i < -\frac{m}{f}\log 2(2-\delta)\Gamma^2 \\
\mathsf{VolatileLeftOuter}_i &\triangleq -\frac{m}{f}\log 2(2-\delta)\Gamma^2 \leq W_i < -\frac{m}{f}\log 2(2-\delta)\Gamma \\
\mathsf{VolatileLeftInner}_i &\triangleq -\frac{m}{f}\log 2(2-\delta)\Gamma \leq W_i < -\frac{m}{f}\log\frac{2(2-\delta)h_{\max}}{h_u} \\
\mathsf{Cold}_i &\triangleq -\frac{m}{f}\log\frac{2(2-\delta)h_{\max}}{h_u} \leq W_i \leq \frac{m}{f}\log\frac{2h_u}{h_{\min}} \\
\mathsf{VolatileRightInner}_i &\triangleq \frac{m}{f}\log\frac{2h_u}{h_{\min}} < W_i \leq \frac{m}{f}\log 2\Gamma \\
\mathsf{VolatileRightOuter} &\triangleq \frac{m}{f}\log 2\Gamma < W_i \leq \frac{m}{f}\log 2\Gamma^2 \\
\mathsf{HotRight}_i &\triangleq W_i > \frac{m}{f}\log 2\Gamma^2
\end{aligned}$$

States VolatileLeftOuter and VolatileRightOuter are of fixed length $(m/f)\log\Gamma$, while states VolatileLeftInner and VolatileRightInner are of length at most $(m/f)\log\Gamma$. These lengths will play a significant role in the following analysis of the ASERT function.

Figure 1: *The states based on the values of random variable W_i.*



We aim to show that for blocks $B_u, \ldots, B_v$ generated in an interval of length $4(m/f)\log\Gamma$ rounds, the following holds:

- For a block B_i, $i > u$ with W_i (w.r.t. B_u) in state Cold, we can construct a new $4(m/f)\log\Gamma$-round sliding window with W_i (w.r.t. B_i) in state VolatileLeftInner, VolatileRightInner or Cold.
- If W_u is in state VolatileLeftInner, VolatileRightInner or Cold, the probability of W_i, $i > u$ reaching HotLeft or HotRight is negligible.
- If W_u is in state VolatileLeftInner, VolatileRightInner or Cold, W_i, $i > u$ will return to Cold with overwhelming probability.

Lemma 3.3 below follows from the definition of each state and party fluctuation; Lemma 3.4 establishes a basic property the volatile states satisfy.

Lemma 3.3. *For a block B_v, if W_v w.r.t. B_u is in state* Cold*, then W_v w.r.t. B_v is in state* VolatileLeftInner, VolatileRightInner *or* Cold.

Proof. Consider W_v w.r.t. B_u, $X_v = W_v + (m/f)\log(h_v/h_u)$. Combining it with Cold$_v$ as well as $h_{\max} \leq \Gamma \cdot h_v$, $h_v \leq \Gamma \cdot h_{\min}$, we get

$$-\frac{m}{f}\log 2(2-\delta)\Gamma \leq -\frac{m}{f}\log\frac{2(2-\delta)h_{\max}}{h_u} + \frac{m}{f}\log\frac{h_v}{h_u}$$
$$\leq X_v = W_v + \frac{m}{f}\log\frac{h_v}{h_u} \leq \frac{m}{f}\log\frac{2h_u}{h_{\min}} + \frac{m}{f}\log\frac{h_v}{h_u} \leq \frac{m}{f}\log 2\Gamma.$$

By the definition of W_v w.r.t. B_v, $W_v = X_v$, therefore in state Volatile LeftInner, VolatileRightInner or Cold. □

Lemma 3.4. *Until the next block is produced, if W_i is in state* VolatileLeftOuter *or* VolatileLeftInner*, the block generation rate is always below $f/2$; if W_i is in state* VolatileRightInner *or* Volatile RightOuter*, the block generation rate is always above $2f$.*

Proof. For the first part, our goal is to show that even if the adversary and the honest parties join force, they cannot achieve a block generation rate over $f/2$. Suppose the i-th block has timestamp r and number of honest parties h_r. If $W_i < -(m/f)\log[2(2-\delta)h_{\max}/h_u]$, the target of B_i satisfies

$$T_r < T_0 \cdot 2^{(\frac{i-1}{f} + \frac{m}{f}\log\frac{h_0}{h_u} - \frac{m}{f}\log\frac{2(2-\delta)h_{\max}}{h_u} - \frac{i-1}{f})/(m/f)}$$
$$= \frac{T_0}{2}\cdot\frac{h_0}{(2-\delta)h_{\max}} \leq \frac{T_0}{2}\cdot\frac{h_0}{(2-\delta)h_r}.$$

Therefore, the block generating rate at round r is $pT_rh_r < f/[2(2-\delta)]$. Note that $pT_rh_{\max} < f/[2(2-\delta)]$ as well, which implies that while the number of honest parties may raise during the rounds till the next block will be produced, the block generating rate will never exceed $f/[2(2-\delta)]$. Moreover, the adversary may join force to accelerate the block production. Recall that $\forall r, t_r \leq (1-\delta)h_r$, after the adversary joins, the block production rate is still below $f/2$.

For the second part, we prove that the block generation rate will not fall below $2f$ when the honest parties keep working by themselves. Similarly, assuming the i-th block has timestamp r and number of honest parties h_r as well as $W_i \geq (m/f)\log(2h_u/h_{\min})$. Thus, the corresponding target

$$T_r > T_0 \cdot 2^{(\frac{i-1}{f} + \frac{m}{f}\log\frac{h_0}{h_u} + \frac{m}{f}\log\frac{2h_u}{h_{\min}} - \frac{i-1}{f})/(m/f)} = 2T_0\cdot\frac{h_0}{h_{\min}} \geq 2T_0\cdot\frac{h_0}{h_r}.$$

This implies $pT_rh_{\min} > 2f$. Thus, we get that the block generation rate is always above $2f$. □

We are now ready to establish the probability of "escaping" from a volatile state to a hot state.

Lemma 3.5. *Consider blocks $B_u, \ldots, B_v$ with timestamps $r_v - r_u \leq 4(m/f)\log\Gamma$ in a $(\langle\gamma, \ell + 2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment. If W_u is in state* VolatileLeftInner, VolatileRightInner *or* Cold*, the probability of W_i, $i > u$, reaching* HotLeft *or* HotRight *is negligible.*

Proof. Regarding the probability of escaping from VolatileLeft Outer, by Lemma 3.4, at every round it will succeed producing a block with probability at most $f/2$. We view the number of blocks produced as a binomial distribution with success probability $f/2$. And, for worst case, W_i begins at the leftmost point in VolatileRight Inner, it has to go leftwards for $(m/f)\log\Gamma$ in order to reach Hot Left.

Since $W_{i+1} = W_i + (r_{i+1} - r_i) - 1/f$, we get $W_v = W_u + r_v - r_u - (u-v)/f$. Assume now $W_u = -(m/f)\log 2(2-\delta)\Gamma$, if W_v is in Hot Left, $r_v - r_u - (u-v)/f < -(m/f)\log\Gamma$. Obviously, this will never happen in the first $(m/f)\log\Gamma$ rounds. For the rounds with index in $\{(m/f)\log\Gamma + 1, \ldots, 4(m/f)\log\Gamma\}$, split them into segments with length $1/f$. For r in i-th segment with index $\{(m/f)\log\Gamma + (i-1)/f + 1, \ldots, (m/f)\log\Gamma + i/f\}$, suppose we produce a block B_v, in expectation, parties will succeed for $\lfloor(m\log\Gamma + i - 1)/2\rfloor$ times in these rounds. If they succeed for more than $(m\log\Gamma + i)$ times, then $r_v - r_u - (u-v)/f < -(m/f)\log\Gamma$, thus reach HotLeft.

Note that $m\log\Gamma + i \geq 2\cdot\lfloor(m\log\Gamma + i - 1)/2\rfloor$ always holds. By Theorem A.1, let $Z_i, \ldots, Z_T$ $(T = r)$ be independent variables with $\mathrm{E}[Z_i] = f/2$ and $Z_i \in \{0, 1\}$. Let $Z = \sum_{i=1}^{T} Z_i$, $\mu = \sum_{i=1}^{T} f/2 =$

$\mathrm{E}[Z] = \lfloor (m\log\Gamma + i - 1)/2 \rfloor \geq m\log\Gamma$. Then, for $\Lambda = 1$, we get

$$\mathbf{Pr}[Z \geq (1+\Lambda)\mu] \leq \exp\Big[-\frac{\Lambda^2}{2+\Lambda}\cdot m\log\Gamma\Big] \leq 2^{-\Omega(m)}.$$

Eventually, this may happen for $3(m/f)\log\Gamma$ times, thus we get the negligible escape probability

$$1-(1-2^{-\Omega(m)})^{3\frac{m}{f}\log\Gamma} \geq 3\frac{m}{f}\log\Gamma\cdot 2^{-\Omega(m)} = 2^{-\Omega(\mathrm{polylog}(\kappa))}.$$

Consider the probability of escaping from VolatileRightOuter, by Lemma 3.4, at every round it will succeed producing a block with probability at least $2f$. We view the number of blocks produced as a binomial distribution with success probability $2f$. And, for worst case, W_i begins at the rightmost point in VolatileRightInner, it has to go rightwards for $(m/f)\log\Gamma$ in order to reach HotRight.

Since $W_{i+1} = W_i + (r_{i+1} - r_i) - 1/f$, we get $W_v = W_u + r_v - r_u - (u-v)/f$. Assume now $W_u = (m/f)\log 2\Gamma$, if W_v is in Hot Right, $r_v - r_u - (u-v)/f > (m/f)\log\Gamma$. Again, this will never happen in the first $(m/f)\log\Gamma$ rounds. For the rounds with index in $\{(m/f)\log\Gamma + 1, \ldots, 4(m/f)\log\Gamma\}$, split them into segments with length $1/f$. For r in i-th segment with index $\{(m/f)\log\Gamma + (i-1)/f + 1, \ldots, (m/f)\log\Gamma + i/f\}$ suppose we produce a block B_v, in expectation, parties will succeed for $2(m\log\Gamma + i - 1)$ times in these rounds. If they succeed for less than i times, then $r_v - r_u - (u-v)/f > (m/f)\log\Gamma$, thus reach HotRight.

Note that $i \leq (1/2)\cdot 2(m\log\Gamma + i - 1)$ always holds. By Theorem A.1, let $Z_i, \ldots, Z_T$ $(T = r)$ be independent variables with $\mathrm{E}[Z_i] = 2f$ and $Z_i \in \{0,1\}$. Let $Z = \sum_{i=1}^{T} Z_i$, $\mu = \sum_{i=1}^{T} 2f = \mathrm{E}[Z] = 2(m\log\Gamma + i - 1) \geq 2\log\Gamma$. Then, for $\Lambda = 1/2$, we get

$$\mathbf{Pr}[Z \leq (1-\Lambda)\mu] \leq \exp\Big[-\frac{\Lambda^2}{2+\Lambda}\cdot 2m\log\Gamma\Big] \leq 2^{-\Omega(m)}.$$

Eventually, this may happen for $3(m/f)\log\Gamma$ times, and the final escape probability

$$1-(1-2^{-\Omega(m)})^{3\frac{m}{f}\log\Gamma} \geq 3\frac{m}{f}\log\Gamma\cdot 2^{-\Omega(m)} = 2^{-\Omega(\mathrm{polylog}(\kappa))}$$

is still negligible. □

Next, we focus on the "return" probability. Since W_i will travel far away from the *relatively calibrated timestamp* (i.e., HotLeft or HotRight) only with negligible probability, and once it reaches Cold we are done, we consider the following two bad events:

- During $4(m/f)\log\Gamma$ rounds, beginning at $-(m/f)\log 2(2-\delta)\Gamma$ (the leftmost point of VolatileLeftInner), W_i stays in state Volatile LeftOuter and VolatileLeftInner.
- During $4(m/f)\log\Gamma$ rounds, beginning at $(m/f)\log 2\Gamma$ (the rightmost point of VolatileRightInner), W_i stays in the state VolatileRightInner and VolatileRightOuter.

We show that by the concentration of the binomial distribution, these two bad events happen only with negligible probability. Note that our results are achieved considering the largest distance W_i needs to travel and with worst success probability. For those events that start closer to the relatively calibrated timestamp, the events' probability will be much lower.

Lemma 3.6. *Consider blocks $B_u, \ldots, B_v$ with $r_v - r_u \leq 4(m/f)\log\Gamma$ in a $(\langle\gamma, \ell + 2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment. If W_u is in state* VolatileLeftInner, VolatileRightInner *or* Cold, *W_i, $i > u$ will return to* Cold *with overwhelming probability.*

Proof. We consider the two bad events that makes W_i fail to return Cold.

For the first event of staying in the left-side states, Lemma 3.4 shows that at every round it will succeed producing a block with probability at most $f/2$. We view the number of blocks produced as a binomial distribution with success probability $f/2$. And, since we assume it begins at the leftmost point in VolatileLeftInner, it has to go rightwards for at most $(m/f)\log\Gamma$ in order to reach Cold.

Consider the first $(4m/f)\log\Gamma$ rounds with blocks $\{B_u, \ldots, B_v\}$. Since $W_{i+1} = W_i + (r_{i+1} - r_i) - 1/f$, we get $W_v = W_u + r_v - r_u - (u-v)/f$. Assume now $W_u = -(m/f)\log 2(2-\delta)\Gamma$, if W_v is in Cold, $r_v - r_u - (u-v)/f > (m/f)\log\Gamma$. In expectation, parties will succeed for $(2m\log\Gamma)$ times in $(4m/f)\log\Gamma$ rounds. If they succeed for more than $(3m\log\Gamma)$ times, it cannot satisfy $r_v - r_u - (u-v)/f > (m/f)\log\Gamma$, thus fails to reach Cold, i.e., W_i still falls in Volatile LeftInner.

By Theorem A.1, let $Z_i, \ldots, Z_T (T = (4m/f)\log\Gamma)$ be independent variables with $\mathrm{E}[Z_i] = f/2$ and $Z_i \in \{0,1\}$. Let $Z = \sum_{i=1}^{T} Z_i$ and $\mu = \sum_{i=1}^{T} f/2 = \mathrm{E}[Z] = 2m\log\Gamma$. Then, for $\Delta = 1/2$, we get

$$\mathbf{Pr}[Z \geq (1+\Lambda)\mu] \leq \exp\Big(-\frac{\Lambda^2}{2+\Lambda}\cdot 2m\log\Gamma\Big) = 2^{-\Omega(m)}.$$

For the second event of staying in the right-side states, Lemma 3.4 shows that at every round it will succeed producing a block with probability at least $2f$. We view the number of blocks produced as a binomial distribution with success probability $2f$. And, since we assume it begins at the rightmost of VolatileRightInner, it has to go leftwards for at most $(m/f)\log\Gamma$ in order to reach Cold.

Consider the first $(2m/f)\log\Gamma$ rounds with blocks $\{B_u, \ldots, B_v\}$. Since $W_{i+1} = W_i + (r_{i+1} - r_i) - 1/f$, we get $W_v = W_u + r_v - r_u - (u-v)/f$. Assume now $W_u = (m/f)\log 2\Gamma$, if W_v is in Cold, $r_v - r_u - (u-v)/f < -(m/f)\log\Gamma$. In expectation, parties will succeed for $(4m\log\Gamma)$ times in $(2m/f)\log\Gamma$ rounds. If they succeed for less than $(3m\log\Gamma)$ times, it cannot satisfy $r_v - r_u - (u-v)/f > -(m/f)\log\Gamma$, thus fail to reach Cold, i.e., W_i still falls in Volatile RightInner.

By Theorem A.1, let $Z_i, \ldots, Z_T (T = (2m/f)\log\Gamma)$ be independent variables with $\mathrm{E}[Z_i] = 2f$ and $Z_i \in \{0,1\}$. Let $Z = \sum_{i=1}^{T} Z_i$ and $\mu = \sum_{i=1}^{T} 2f = \mathrm{E}[Z] = 4m\log\Gamma$. Then, for $\Delta = 1/4$, we get

$$\mathbf{Pr}[Z \leq (1-\Lambda)\mu] \leq \exp\Big(-\frac{\Lambda^2}{2+\Lambda}\cdot 4m\log\Gamma\Big) = 2^{-\Omega(m)}.$$

Therefore, the return probability is $1 - 2^{-\Omega(\mathrm{polylog}(\kappa))}$. □

Lemma 3.5 guarantees "goodness" in a sliding window with fixed length and ideal start state, while Lemma 3.6 states the desired probability of finding the next ideal start block in such a sliding window. Therefore, we are able to extend our analysis from the first $4(m/f)\log\Gamma$ rounds to the whole execution. Formally:

Theorem 3.7. *All the target recalculation points on a chain in a $(\langle\gamma, \ell + 2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment are good.*

Proof. We show that "goodness" is maintained for all target recalculation points in a sliding window of length $4(m/f)\log\Gamma$, and that it can be extended from the first round to the whole execution. Note that in our assumption, the first block satisfies $pT_1 n_{r_1} = f$ (i.e., the state is Cold), thus, we can establish the first sliding window

starting from B_1. The iteration works as follows: We choose the last block with state Cold in the sliding window, and then establish a new sliding window starting from it. According to Lemma 3.6, such block always exists. For blocks in each sliding window, Lemma 3.5 ensures that all the target recalculation points are good (i.e., never go into the state HotLeft or HotRight). □

We note that if we consider a respecting environment that allows more time for the same party fluctuation, then the goodness parameter—i.e., the upper bound and lower bound of the target recalculation point—can be closer to f. For example, the good target recalculation parameter can be changed to $(1+\Gamma^3)f$ and $f/[(2-\delta)(1+\Gamma^3)]$, with sliding window length $2\Gamma^3(m/f)\log\Gamma$. In the analysis, this is a trade-off. For simplicity, the presentation in this section chooses the more intuitive values ($2f$ and $f/2$).

4 FULL SECURITY ANALYSIS

So a memoryless target recalculation function, under certain conditions, is able to maintain a steady block generation rate. Can this be used by a protocol to satisfy the desired blockchain and ledger properties, as formulated in [4, 5]? That's what this section demonstrates, by providing the relevant protocol abstraction and following their analytical approach, albeit with different parameters and proofs.

4.1 The Bitcoin Cash Backbone Protocol

The main changes introduced by Bitcoin Cash's hard fork from Bitcoin were an increase of the block size, replacement of the difficulty adjustment algorithm, and modification of the transaction rules; the protocol structure remained unchanged. For the analysis, we adopt the protocol abstraction presented in [4] (and follow-ups), consisting of the main algorithm (Algorithm 1), which at the beginning of a round receives input (new transactions as well as chains sent by other miners); validates them and compares them (according to their accumulated difficulty) against the miner's current chain, adopting the one with highest difficulty (it could be the party's own); and attempts to extend the adopted chain by generating a PoW with the current round's difficulty value. Before turning to the protocol description, we first review the desired properties the protocol should satisfy.

Blockchain properties. We review the two main properties to be satisfied by a PoW-based blockchain protocol.

- **Common Prefix** Q_{cp} (Parameterized by $k \in \mathbb{N}$): For any two players P_1, P_2 holding chains C_1, C_2 at rounds r_1, r_2, with $r_1 \leq r_2$, it holds that $C_1^{\lceil k} \preceq C_2$.
- **Chain Quality** Q_{cq} (Parameterized by $\mu \in \mathbb{R}$ and $\ell \in \mathbb{N}$): For any party P with chain C in $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}$, and any segment of that chain of difficulty d such that the timestamp of the first block of the segment is at least ℓ smaller than the timestamp of the last block, the blocks the adversary has contributed in the segment have a total difficulty that is at most $\mu \cdot d$.

Transaction ledger. Similarly to Bitcoin, the main application of the Bitcoin Cash protocol is a *robust transaction ledger*, aimed at maintaining a serialized transaction sequence organized in the form of a blockchain, satisfying the following two properties. Let $\mathcal{L}$ denote such ledger.

- **Consistency**: For any two honest parties P_1, P_2, reporting $\mathcal{L}_1, \mathcal{L}_2$ at rounds $r_1 \leq r_2$, resp., it holds that $\mathcal{L}_1$ is a prefix of $\widetilde{\mathcal{L}}_2$.
- **Liveness** (Parameterized by $u \in \mathbb{N}$, the "wait time" parameter): If a transaction tx is provided to all honest parties for u consecutive rounds, then it holds that for any player P, tx will be in $\mathcal{L}$.

Protocol description. As in [4], in our description of the protocol we intentionally avoid specifying the type of values/content that miners try to insert in the chain, the type of chain validation they perform (beyond checking for its structural properties with respect to the hash functions $G(\cdot), H(\cdot)$, and the way they interpret the chain. These checks and operations are handled by the external functions $V(\cdot), I(\cdot)$ and $R(\cdot)$ (the content validation function, the input contribution function and the chain reading function, resp.) which are specified by the application that runs "on top" of the backbone protocol. The Bitcoin Cash protocol in the dynamic setting comprises three algorithms *chain validation*, *chain comparison* and *proof of work*. Refer to [7] for their full specification.

Chain validation. The validate algorithm performs a validation of the structural properties of a given chain C. It is given as input the value q, as well as hash functions $H(\cdot), G(\cdot)$. It is parameterized by the content validation predicate predicate $V(\cdot)$ as well as by $D(\cdot)$, the *target calculation function* (see Section 3.1). For each block of the chain, the algorithm checks that the proof of work is properly solved (with a target that is suitable as determined by the target calculation function), and that the counter ctr does not exceed q. Furthermore it collects the inputs from all blocks, x_C, and tests them via the predicate $V(\mathbf{x}_C)$. Chains that fail these validation procedure are rejected.

Chain Comparison. The objective of the second algorithm, called maxvalid, is to find the "best possible" chain when given a set of chains. The algorithm is straightforward and is parameterized by a $\max(\cdot)$ function that applies some ordering in the space of chains. The most important aspect is the chains' difficulty in which case $\max(C_1, C_2)$ will return the most difficult of the two. In case $\text{diff}(C_1) = \text{diff}(C_2)$, some other characteristic can be used to break the tie. In our case, $\max(\cdot,\cdot)$ will always return the first operand to reflect the fact that parties adopt the first chain they obtain from the network.

Proof of work. The third algorithm, called pow, is the proof of work-finding procedure. It takes as input a chain and attempts to extend it via solving a proof of work. This algorithm is parameterized by two hash functions $H(\cdot), G(\cdot)$. Moreover, the algorithm calls the target calculation function $D(\cdot)$ in order to determine the value T that will be used for the proof of work. The procedure, given a chain C and a value x to be inserted in the chain, hashes these values to obtain h and initializes a counter ctr. Subsequently, it increments ctr and checks to see whether $H(ctr, h) < T$; in case a suitable ctr is found then the algorithm succeeds in solving the POW and extends chain C by one block.

Bitcoin Cash backbone protocol. The core of the Bitcoin Cash backbone protocol with variable difficulty is similar to that in [5],

with several important distinctions. First is the procedure to follow when the parties become active. Parties check the ready flag they possess, which is false if and only if they have been inactive in the previous round. In case the ready flag is false, they diffuse a special message '**Join**' to request the most recent version of the blockchain(s). Similarly, parties that receive the special request message in their Receive() tape broadcast their chains. As before parties, run "indefinitely" (our security analysis will apply when the total running time is polynomial in κ). The input contribution function $I(\cdot)$ and the chain reading function $R(\cdot)$ are applied to the values stored in the chain. Parties check their communication tape Receive() to see whether any necessary update of their local chain is due; then they attempt to extend it via the POW algorithm pow. The function $I(\cdot)$ determines the input to be added in the chain given the party's state st, the current chain C, the contents of the party's input tape Input() and communication tape Receive(). The input tape contains two types of symbols, Read and (Insert, *value*); other inputs are ignored. In case the local chain C is extended the new chain is diffused to the other parties. Finally, in case a Read symbol is present in the communication tape, the protocol applies function $R(\cdot)$ to its current chain and writes the result onto the output tape Output(). The pseudocode of the backbone protocol is presented in Algorithm 1.

Algorithm 1 The Bitcoin Cash backbone protocol in the dynamic setting at round "round" on local state (st, C) parameterized by the *input contribution function* $I(\cdot)$ and the *chain reading function* $R(\cdot)$. The ready flag is **false** if and only if the party was inactive in the previous round.

1: **if** ready = **true then**
2: Diffuse('ready')
3: $\widetilde{C} \leftarrow$ maxvalid(C all chains C' found in Receive())
4: $\langle st, x\rangle \leftarrow I(st, \widetilde{C}, \text{round}, \text{Input}(), \text{Receive}())$
5: $C_{\text{new}} \leftarrow \text{pow}(\text{round}, x, \widetilde{C})$
6: **if** $(C \neq C_{\text{new}}) \vee$ ('**Join**' $\in$ Receive()) **then**
7: $C \leftarrow C_{\text{new}}$
8: Diffuse(C) ▷ chain is diffused when it is updated or when someone wants to join.
9: **end if**
10: **if** Input() contains Read **then**
11: **write** $R(\mathbf{x}_C)$ to Output()
12: Diffuse(RoundComplete)
13: **end if**
14: **else**
15: ready $\leftarrow$ **true**
16: Diffuse(Join, RoundComplete)
17: **end if**

4.2 ASERT: Full Analysis

Table 1 summarizes the parameters that will be used in our analysis, some of which have already been introduced. Note that our security parameter is κ, and $\varphi = \Theta(m) = \text{polylog}(\kappa)$. Moreover, we consider the fluctuation of the total number of parties (cf. Definition 2.1 and Fact 1).

Table 1: *Summary of parameters (ASERT).*

- δ: Advantage of honest parties, $\forall r(t_r/h_r < 1-\delta)$.
- $\gamma, \sigma, \Gamma, \Sigma$: Determine how the number of parties fluctuates across rounds in a period (cf. Definition 2.1 and Fact 1).
- f: Probability that at least one honest party succeeds generating a PoW in a round assuming h_0 parties and target T_0 (the protocol's initialization parameters).
- m: Smoothing factor (cf. Definition 3.1).
- τ: Parameter that regulates the target that the adversary could query the PoW with.
- ϵ: Quality of concentration of random variables (cf. Definition 4.4).
- κ: The length of the hash function output.
- φ: Related to the properties of the protocol.
- L: The total number of rounds in the execution of the protocol.

During a round r of an execution E, the honest parties might be split and work on different chains, and thus might query the random oracle on different targets. Denote by $T_r^{\min}$ and $T_r^{\max}$ the minimum and maximum of these targets, respectively.

Next, we extend the definition of "goodness" from Section 3.2 to apply rounds and chains, in addition to recalculation points, and following [5], we define a property called *accuracy*, which we will then show most executions satisfy, and which will help achieve the desired application's properties.

Definition 4.1 (Goodness). A target-recalculation point r is *good* if the target T for the next block satisfies $f/2(2-\delta)\Gamma^3 \leq ph_rT \leq 2\Gamma^3 f$. Round r is *good* if $f/2\gamma(2-\delta)\Gamma^3 \leq ph_rT_r^{\min}$ and $ph_rT_r^{\max} \leq 2\gamma\Gamma^3 f$. A chain is *good* if all its target-recalculation points are good.

Definition 4.2 (Accuracy). A block created at round u is *accurate* if it has a timestamp v such that $|u-v| \leq \ell + 2\Delta$. A chain is *accurate* if all its blocks are accurate. A chain is *stale*, if for some $u \geq \ell + 2\Delta$, it does not contain an honest block with timestamp $v \geq u - \ell - 2\Delta$.

For a given round r, we let $\mathcal{S}_r$ denote the set of chains that belong, or could potentially belong to an honest party. Being explicit about this set of chains will help in the formulation of a number of predicates (see below). Specifically, $\mathcal{S}_r$ includes[4]:

- Chain C that belongs to an honest party;
- chain C with $\text{diff}(C) > \text{diff}(C')$ for some chain C' of an honest party; and
- chain C with $\text{diff}(C) = \text{diff}(C')$ for some chain C' of an honest party and head(C) was computed no later than head(C').

Random variables. We are interested in estimating the difficulty acquired by honest parties during a sequence of rounds. For a given round r, the following real-valued random variables are defined in [6]:

- D_r: Sum of the difficulties of all blocks computed by honest parties.
- Y_r: Maximum difficulty among all blocks computed by honest parties.

[4]Note that these chains should exist and be valid at round r.

– Q_r: Equal to Y_r when $D_u = 0$ for all $r < u < r + \Delta$ and 0 otherwise.

A round r such that $D_r > 0$ is called *successful* and one where $Q_r > 0$ *isolated-successful*.

Regarding the adversary, in order to overcome the fact that he can query the oracle for arbitrarily low targets and thus obtain blocks of arbitrarily high difficulty, we would like to upper-bound the difficulty he can acquire during a set J of queries. This is achieved by associating a set of consecutive adversarial queries J with the target of its first query. We denote this target $T(J)$, and say that $T(J)$ is *associated* with J. We then define $A(J)$ and $B(J)$ to be equal to the sum of the difficulties of all blocks computed by the adversary during queries in J for target at least $T(J)/\tau$ and $T(J)$, respectively–i.e., queries in J for targets less than $T(J)/\tau$ (resp. $T(J)$) do not contribute to $A(J)$ (resp. $B(J)$).

For simplicity, we write $h(S) = \sum_{r\in S} h_r$ for a set of rounds S and queries J (similarly, $t(S)$, $D(S)$, $Y(S)$, $Q(S)$, $A(J)$ and $B(J)$).

We also define the random variable $\mathcal{E}$ taking values on our probability space and with a distribution induced by the random coins of all entities (adversary, environment, parties) and the random oracle. Let $\mathcal{E}_{r-1}$ fix the execution just before round r. In particular, a value E_{r-1} of $\mathcal{E}_{r-1}$ determines the adversarial strategy and so determines the targets against which every party will query the oracle at round r and the number of parties h_r and t_r, but it does not determine D_r or Q_r. For an adversarial query j we will use, slightly overloading notation, $E^{(J)}_{j-1}$ to denote the execution just before this query.

The following fact is a consequence of Definition 2.1 (respecting environments):

Fact 1. *Let S be a set of at most Σ consecutive rounds in a $(\langle\gamma,\sigma\rangle, \langle\Gamma,\Sigma\rangle)$-respecting environment and $U \subseteq S$.*

(a) If $U \preccurlyeq S$ and $|U| < \sigma$,

$$\frac{h_S}{\Gamma} \le \frac{h(S)}{|S|} \le \Gamma \cdot h_S \text{ and } \frac{h_U}{\gamma} \le \frac{h(U)}{|U|} \le \gamma \cdot h_U,$$

where $h_S \in \{h_r : r \in S\}$ and $h_U \in \{h_r : r \in U\}$.

(b)

$$h(S) \le (1 + \frac{\Gamma|S\backslash U|}{|U|})h(U) \text{ and } |S| \sum_{r\in S}(ph_r)^2 \le \Gamma(\sum_{r\in S} ph_r)^2.$$

In order to obtain meaningful concentration of random variables, we have to consider a sufficiently long sequence with a number of rounds at least

$$\ell = \frac{4(2-\delta)(1+3\epsilon)}{\epsilon^2 f[1-2\gamma\Gamma^3 f]^{\Delta+1}} \cdot \max\{\Delta,\tau\} \cdot \gamma\Gamma^4 \cdot \varphi. \tag{3}$$

We will assume that ℓ is appropriately small compared to the length m of a sliding interval/window. Specifically,

$$2\ell + 6\Delta \le \frac{\epsilon m}{2\gamma\Gamma^3 f}. \tag{C1}$$

In addition, we would like the advantage δ of the honest parties over adversarial parties to be large enough to absorb error factors. Thus, we require the following inequalities:

$$[1-2\gamma\Gamma^3 f]^{\Delta} \ge 1-\epsilon \text{ and } \epsilon \le \delta/8 \le 1/8. \tag{C2}$$

Next, we show a chain-growth lemma referring to accumulated difficulty (cf. [6]), as opposed to number of blocks in the original formulations [4, 8].

Lemma 4.3 (Chain Growth). *Suppose that at round u of an execution E an honest party diffuses a chain of difficulty d. Then, by round v, every honest party has received a chain of difficulty at least $d + Q(S)$, where $S = \{r : u + \Delta \le r \le v - \Delta\}$.*

Typical executions. The notion of *typical executions* was introduced in the analysis framework we are following [4] in order to capture situations where an execution E's progress does not deviate too much from its expected (desired) progress. Since executions consist of rounds, and within rounds parties perform Bernoulli trials, we can calculate the expected progress when given the corresponding probabilities. On this basis, if the difference between the real execution and its expectation is reasonable, E is declared "typical." Note that besides expectation, the variance should also be taken into consideration. We will later show (applying Theorem A.3—martingale inequality) that either the variance is too high with respect to a set of rounds, or the parties have made progress during these rounds as expected.

In addition to the behavior of the random variables described above, bad events may occur related to the underlying hash function $H(\cdot)$, which is modeled as a random oracle and used to obtain PoWs. The bad events are *insertion* (of a block in between two consecutive blocks), *copy* (same block exists in two different position of the blockchain), and *prediction* (a block extends one with an earlier creation time). Refer to [6] for a precise definition. A typical execution will rule out these bad events as well.

We are now ready to specify what is a typical execution in our setting (compare with [6]'s).

Definition 4.4 (Typical execution). An execution E is *typical* if the following hold:

(a) For any set S of at least ℓ consecutive good rounds,

$$(1-\epsilon)[1-2\gamma\Gamma^3 f]^{\Delta} ph(S) < Q(S) \le D(S) < (1+\epsilon)ph(S).$$

(b) For any set J indexing a set of consecutive adversarial queries and $\alpha(J) = 2(\frac{1}{\epsilon}+\frac{1}{3})\varphi/T(J)$,

$$A(J) < p|J| + \max\{\epsilon p|J|, \tau\alpha(J)\} \text{ and } B(J) < p|J| + \max\{\epsilon p|J|, \alpha(J)\}.$$

(c) No insertions, no copies, and no predictions occurred in E.

The next proposition is a simple application of Definition 4.4 and the honest-majority assumption.

Proposition 4.5. *Let E be a typical execution in a $(\langle\gamma,\sigma\rangle, \langle\Gamma,\Sigma\rangle)$-respecting environment. Let $S = \{r \mid (u \le r \le v) \wedge (v - u \ge \ell)\}$ be a set of consecutive good rounds and J the set of adversarial queries in $U = \{r \mid u - \Delta \le r \le v + \Delta\}$. Then the following inequalities hold:*

(a) $(1+\epsilon)p|J| \le Q(S) \le D(U) < (1+5\epsilon)Q(S)$.

(b) If w is a good round such that $|w - r| \le \Sigma$ for any $r \in S$, then $Q(S) > (1+\epsilon)[1-2\gamma\Gamma^3 f]^{\Delta}|S|ph_w/\Gamma$. If, in addition, $T(J) \ge T_w^{\min}$, then $A(J) < (1-\delta+3\epsilon)Q(S)$.

We are now able to show that almost all Bitcoin Cash backbone protocol executions polynomially bounded (in κ) are typical (the proof is presented in [7]) . Formally:

Theorem 4.6. *Assuming the Bitcoin Cash backbone protocol runs for L rounds, the event "E is not typical" is bounded by* $\mathrm{poly}(L) \cdot e^{-\Omega(\mathrm{polylog}(\kappa))}$.

Accuracy and goodness. Next, we consider accuracy and goodness over the space of typical executions in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, as well as implications between the two. We assume that all the requirements for the initialization parameters h_0 and T_0 are satisfied.

Lemma 4.7. *Let E be a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment. If E_{r-1} is good, then there are no stale chains in $\mathcal{S}_r$.*

Corollary 4.8. *Let E be a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment. If E_{r-1} is good, then all chains in $\mathcal{S}_r$ are accurate.*

Theorem 4.9. *A typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment is accurate and good.*

Blockchain and ledger properties. We now show that the Bitcoin Cash backbone protocol satisfies the two properties common prefix and chain quality (Section 4.1) for a suitable respecting environment. First, a preliminary lemma:

Lemma 4.10. *For any round r of a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment and any two chains C and C' in $\mathcal{S}_r$, the timestamp of head($C \cap C'$) is at least $r - 2\ell - 4\Delta$.*

Theorem 4.11 (Common-Prefix). *For a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, the common-prefix property holds for parameter ϵm.*

Theorem 4.12 (Chain-Quality). *For a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, the chain-quality property holds with parameters $\ell + 2\Delta$ and $\mu = \delta - 3\epsilon$.*

We conclude by showing that a typical execution of the Bitcoin Cash backbone protocol in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, satisfies the two properties of a robust transaction ledger presented in Section 2. They are the direct consequence of the blockchain properties shown above, following the approach in [5, 6].

Theorem 4.13 (Consistency). *For a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, Consistency is satisfied by setting the settled transactions to be those reported more than ϵm blocks deep.*

Theorem 4.14 (Liveness). *For a typical execution in a $(\langle\gamma, \ell+2\Delta\rangle, \langle\Gamma, 4(m/f)\log\Gamma\rangle)$-respecting environment, Liveness is satisfied for depth ϵm with wait-time $(4\Gamma^4+1)\epsilon m/f$.*

5 THE SMA TARGET RECALCULATION FUNCTION

For completeness, here we present a brief description and analysis of Bitcoin Cash's previous target recalculation function. Even though this function has now been deprecated, it provides some insights and elements of comparison against ASERT based on actual party fluctuation. The full analysis of SMA is presented in the full version of this paper [7].

Recall that the target calculation function $D(\cdot)$ aims at maintaining the block production rate constant. The probability $f(T, n)$ with which n parties produce a new block with target T is approximated by $f(T, n) \approx qTn/2^\kappa$. As in the case of Bitcoin, to achieve the above goal Bitcoin Cash tries to keep $qTn/2^\kappa$ close to f. To that end, the SMA function watches an *epoch* of m previous blocks, and based on their difficulty as well as on how fast these blocks were generated, it computes the next target. More specifically, say the last m blocks of a chain C with targets $\{T_i\}_{i\in[m]}$ were produced in Λ rounds. For every block in the epoch and n participants, it holds that $f(T_i, n) \approx qT_in/2^\kappa$. For m consecutive blocks, the average block generating rate $f^* = qn\sum_{i\in[m]} T_i/(m\cdot 2^\kappa)$, and the entire generating time $\Lambda = m/f^* = m^2\cdot 2^\kappa/(qn\sum_{i\in[m]} T_i)$.

Consider the case where a number of parties attempt to produce m blocks of target $\{T_i\}_{i\in[m]}$ in Λ rounds (in expectation). Such number of players can be estimated as $n(T_1, \ldots, T_m, \Lambda) = m^2\cdot 2^\kappa/(q\Lambda\sum_{i\in[m]} T_i)$; then the next target T' is set so that $n(T_1, \ldots, T_m, \Lambda)$ players would need $1/f$ rounds in expectation to produce the next block of target T'. Therefore, it makes sense to set

$$T' = \frac{\Lambda}{m^2/f}\cdot\sum_{i\in[m]} T_i$$

because if the number of players is indeed $n(T_1, \ldots, T_m, \Lambda)$ and remains unchanged, it will take them $1/f$ rounds in expectation to produce next block. If the estimate of the initial number of parties is n_0, we will assume T_0 is appropriately set so that $f \approx qT_0n_0/2^\kappa$ and then $T' = n_0T_0/n(T_1, \ldots, T_m, \Lambda)$.

Based on the above, we can now give a formal definition of the SMA function. In the definition, parameter τ serves as the "dampening filter" to deal with the case $n_0T_0/n(T^{\text{avg}}, \Lambda) \notin [T/\tau, \tau T]$. This mechanism is necessary given an efficient attack presented by Bahack [1] for Bitcoin, which also applies to Bitcoin Cash.

Definition 5.1 (SMA). For fixed constants $\kappa, \tau, m, n_0, T_0$, the target calculation function $D_{\text{SMA}} : \mathbb{Z}^* \to \mathbb{R}$ is defined as

$$D_{\text{SMA}}(\varepsilon) = T_0 \text{ and } D_{\text{SMA}}(r_1, \ldots, r_v) = \begin{cases} \frac{1}{\tau}\cdot T^{\text{avg}}, & \text{if } \frac{n_0\cdot T_0}{n(T^{\text{avg}}, \Lambda)} < \frac{1}{\tau}\cdot T^{\text{avg}} \\ \tau\cdot T^{\text{avg}}, & \text{if } \frac{n_0\cdot T_0}{n(T^{\text{avg}}, \Lambda)} > \tau\cdot T^{\text{avg}} \\ \frac{n_0}{(T^{\text{avg}}, \Lambda)}\cdot T_0, & \text{otherwise} \end{cases}$$

where $n(T^{\text{avg}}, \Lambda) = \frac{m\cdot 2^\kappa}{q\Lambda T^{\text{avg}}}$, with $\Lambda = r_{v-1} - r_{v-m}$ and $T^{\text{avg}} = \frac{1}{m}\cdot\sum_{i=1}^{m} D(r_1, ..., r_{v-i})$.

Remark 1. *At the start of the protocol execution—the first m blocks—parties cannot acquire enough previous blocks for target recalculation. While several alternate approaches exist (e.g., maintain a static target or switch to other target recalculation function), we will assume a "safe start," i.e., there exist blocks with an ideal target value prior to the first round.*

5.1 SMA: Analysis Overview

The difference between SMA and Bitcoin's target recalculation is that the target variation (targets that the adversary can query during a sliding window) is no longer bounded by the dampening filter (τ). This is because in SMA the target is recalculated for *every block*. τ's restriction is for a single calculation; hence, we introduce a new parameter λ to denote the target variation over a long period of time.

To highlight some aspects of the proofs of security of the Bitcoin Cash backbone protocol using SMA, we show that while the targets in an epoch keep varying, we can still always find an appropriate block such that the difficulty accumulated during the epoch (when associated with this block) is well bounded. I.e., even the adversary and an honest party join force, they cannot accelerate this process too much; and even when honest parties work independently, they can still efficiently produce enough blocks.

Moreover, we would like to show that a good relation between the accumulated difficulty in an epoch and its duration always holds, and therefore the target for the next block is good and the block generating rate is close to f. Since the SMA function employs the average targets in the epoch, the previous argument (which only takes into account single target value) collapses. In order to absorb the new errors, We propose new conditions such as the following:

$$2\ell + 6\Delta \leq \frac{\epsilon m}{2(1+\delta)\Gamma^2 f}, \tag{C3}$$

$$\frac{4\lambda\Gamma}{(\lambda\Gamma+1)^2} \cdot [1-(1+\delta)\gamma\Gamma f]^{\Delta} > 1-\epsilon \text{ and } \epsilon \leq \delta/8 \leq 1/8 \tag{C4}$$

where $4\lambda\Gamma/(\lambda\Gamma+1)^2$ (cf. Theorem A.5) helps to "revert" the discrepancy between the accumulated difficulty and the next target.

We conclude that the Bitcoin Cash backbone protocol with the SMA function satisfies Consistency and Liveness using the following parameters:

Theorem 5.2 (Consistency). *For a typical execution in a* $(\langle\gamma/(2-\delta), \ell+2\Delta\rangle, \langle\Gamma/(2-\delta), 2(1+\delta)\Gamma^2 m/f\rangle)$*-respecting environment, Consistency is satisfied by setting the settled transactions to be those reported more than* $\epsilon\gamma^2 m/2\Gamma$ *blocks deep.*

Theorem 5.3 (Liveness). *For a typical execution in a* $(\langle\gamma/(2-\delta), \ell+2\Delta\rangle, \langle\Gamma/(2-\delta), 2(1+\delta)\Gamma^2 m/f\rangle)$*-respecting environment, Liveness is satisfied for depth* $\epsilon\gamma^2 m/2\Gamma$ *with wait-time* $(\gamma^2+1)\epsilon m/f$.

6 COMPARISON WITH THE BITCOIN CASH NETWORK

Our analysis shows that for a sufficiently long execution in a somewhat idealized network environment and without severe external disturbances, Bitcoin Cash would achieve the desired security properties (with overwhelming probability). Comparison of our analysis with the existing Bitcoin Cash network data reveals that the ASERT function performs better than the SMA function in an environment where party fluctuation is large. Further, both SMA and ASERT suffer from undersized parameters. In this section we expand on the above comparisons.

6.1 The Bitcoin Cash Network

Party fluctuation. To perform our comparison, we need to determine the fluctuation (ratio) of the number of parties in the actual Bitcoin Cash network. We can extract it from the public statistical data–the *hashrate*, which is the number of hash queries that all miners in the network perform per second. In our model, the queries that every party can make in one round is bounded. Thus, for the purpose of our comparison we assume the hashrate to be identical to the number of parties, and its fluctuation ratio also reflects the the number of parties'. To be more precise, the available hashrate does not reveal the entire Bitcoin Cash network, but instead the hashing power invested in some well-known mining pools. Nonetheless, since most of the computational power is concentrated in these mining pools, this hashrate can very closely approximate the entire network's.

In addition, instead of the the exact fluctuation ratio with respect to a short period of time (e.g., 10 minutes), we adopt the ratio based on the *daily average hashrate*, which shows the average queries per second in one day; this measure would correspond to parameter Γ in our analysis. Since our analysis should be applied to a relatively long execution, this measure replacement seems reasonable.[5]

Figure 2 shows the *daily average hashrate* in one month period (Jul 18 2020 – Aug 17 2020)[6], which we adopt as the representative case of the Bitcoin Cash network execution under the SMA function. The maximum value in this period is 3.1783EHash/s (on Aug 07 2020) and the minimum is 2.3865EHash/s (on Aug 06 2020), so it suffices to set $\Gamma = 1.398$ as the party fluctuation ratio. Regarding the ratio during a small period of time, note that as we are discussing the average hashrate, $\gamma = 1.057$, which satisfies $\gamma^{\lfloor\Sigma/\sigma\rfloor} > \Gamma$, would be a suitable value.

We note that the assumed behavior may not hold when some type of external disturbance occurs, such as the halving of the block reward or sharp gains and declines in BCH's monetary value, which might cause a large number of miners abruptly joining or dropping out of a mining pool in a short period of time. As it turns out, these events do not happen very frequently (e.g., block reward halves about every 4 years), so we are comfortable extrapolating the execution of the Bitcoin Cash network from what Figure 2 depicts.

Figure 3 shows the daily average hashrate in one month period (Dec 1 2020 – Dec 30 2020), after the Nov. 2020 network update when the ASERT function was adopted. The maximum value in this period is 1.9451EHash/s (on Dec 21 2020) and the minimum is 1.0349EHash/s (on Dec 24 2020). Therefore, Γ should be to 1.88. This fluctuation rate is relatively high, and we speculate was caused by two reasons: (1) the network could not return to a quiet state soon after the update happens, and (2) the price of Bitcoin Cash experienced substantial ups and downs during this period. We thus set the value of $\gamma = 1.099$.

Network delay. Another important aspect to validate our analysis in a bounded-delay network environment is the actual message delay in the network. It is shown in [2] that the delay in the Bitcoin network mainly stems from its multi-hop broadcast ("diffuse" in our model terminology) and block propagation mechanism (Bitcoin Cash employs a similar mechanism to Bitcoin), which we now expand on. As a mining node would typically maintain a limited connections with other nodes, when the node wants to broadcast one block after receiving and verifying it, the node will first send an *inv* message containing the block hash to all its neighbors. If the neighbors do not have the block locally, they will issue a *getdata* message to the sender of *inv*. Then the actual block transfer begins. Thus, at each hop during the broadcast the message incurs a propagation delay consisting of the transmission time and the local

[5] We note that if we consider the fluctuation rate over 10-minute intervals, the overall party fluctuation will become extremely pronounced (parameter Γ will exceed 4), making our analysis inapplicable.

[6] Source: https://bitinfocharts.com/comparison/hashrate-bch.html

Figure 2: *The daily average hashrate of Bitcoin Cash from July 18 to August 16, 2020.*

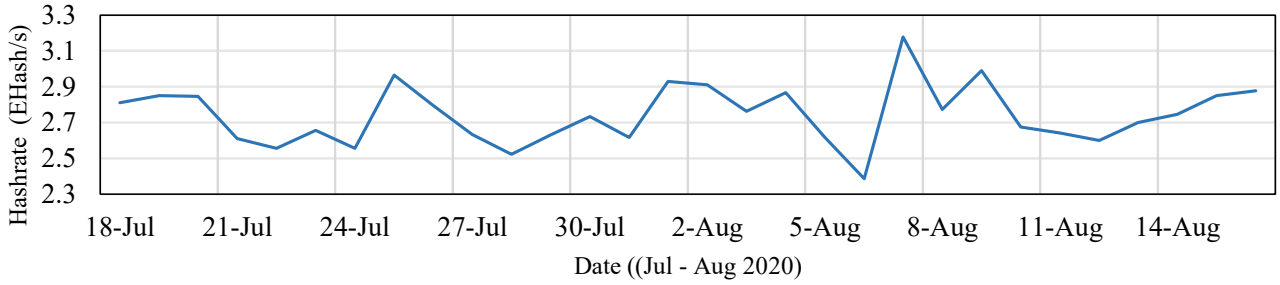
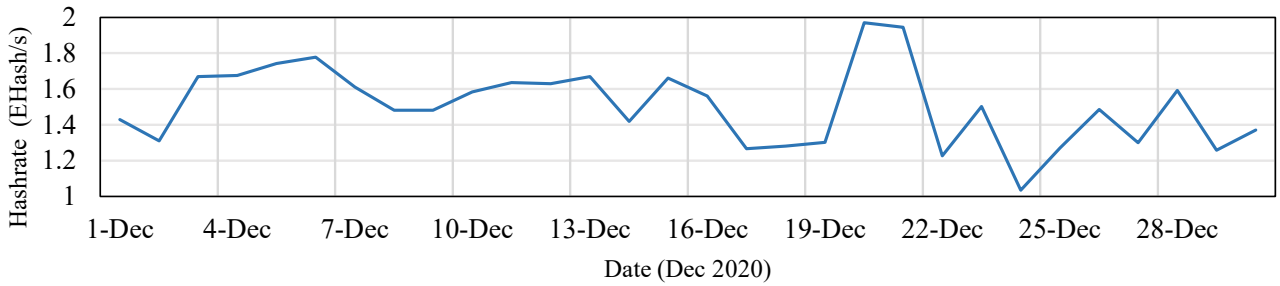


Figure 3: *The daily average hashrate of Bitcoin Cash from Dec 1 to Dec 30, 2020.*



verification of the block. Decker and Wattenhofer [2] evaluated the distribution of the block propagation time since the first block announcement, which shows a median time of 6.5 seconds as well as a mean time of 12.6 seconds. For more recent data, Bitcoin Monitoring [7] carried out by the German Federal Ministry of Education and Research shows that 90% of the block propagation time is below 6 seconds. Thus, assuming a round duration of 6 seconds, it is reasonable to let the delay bound (Δ) in our analysis approximately equal to 1 round.

Target variation. Our formulation (of SMA function) bounds the target variation by a fixed parameter λ and the parties' fluctuation ratio. This is a reasonable idea, since the number of parties, accumulated difficulty and number of blocks (chain growth) are correlated. In the real network, however, the hash power invested in minor blockchains varies wildly, and as such it is unlikely that the number of parties would stay high for a long period of time, and thus target values would seldom vary more pronouncedly than the number of parties. Compared with other parameters, the criteria for selecting λ does not seem as strict. As we can cover more executions if we set a larger value for λ, we choose $\lambda = 1.201$ when applying our analysis. As a result, when compared with previous work [5, 6], our analysis can tolerate situations where targets vary more wildly (i.e., they exceed the $\lambda\Gamma$ bound).

[7] Source: https://dsn.kastel.kit.edu/bitcoin/

6.2 Conclusions

ASERT function. Based on the considerations above, for the purpose of our comparison, we parameterize the real-world Bitcoin Cash network with network bounded-delay $\Delta = 1$ (round), honest advantage $\delta = 0.99$, quality of concentration $\epsilon = 0.123$, long term party fluctuation ratio $\Gamma = 1.88$, short term party fluctuation ratio $\gamma = 1.099$ and ideal block generating rate $f = 0.01$.

On one hand, the resulting probabilities with respect to the goodness parameters are very tight, when considered in the real network parameters (i.e., $m = 288$ and $\Gamma = 1.88$). To be precise, in a sliding window, the probability of the block generation rate exceeding the goodness bound is below 10^{-12}, and the probability of not returning to Cold is less than 10^{-9}. Therefore, it is safe to conclude that the ASERT function achieves a relatively stable block generation rate.

On the other hand, we cannot directly plug in in these parameter values to satisfy Condition (C2), since the party fluctuation is too pronounced, thus making it hard for the concentration parameter ϵ to absorb the errors. Condition (C2), however, can be satisfied in the following two scenarios: (1) party fluctuation ratio in a relatively quiet environment (e.g., $\Gamma = 1.398$ as shown in Fig 2). Then all the requirements are satisfied and so does our analysis. (2) Balancing the block generation rate bound and the sliding window length we

consider (cf. Section 3.2), then all the requirements are satisfied if the upper bound for goodness is $(1+\Gamma^3)f$.

Finally, we observe that the choice of the smoothing factor (m) is too low. While the current value $m = 288$ can satisfy the basic requirements in Condition (C1), it is better to let $m = 432$ to increase the value of ℓ, thus making it closer to the real execution. However, we note that such value of m still fails to give a tight typical execution probability. This is because in order to satisfy the (desired) blockchain properties, the length of a sliding window should be much larger (of the order of years!) so that the martingale probability in Theorem 4.6 can be negligible.

SMA function. We consider the same network delay, honest advantage and quality of concentration as those used for the ASERT evaluation. Afterwards, we parameterize the SMA function with long-term party fluctuation ratio $\Gamma = 1.398$, short-term party fluctuation ratio $\gamma = 1.057$ and target fluctuation parameter $\lambda = 1.201$. We obtain that Condition (C4) is satisfied under these parameters.

Regarding the protocol parameters m and τ in use, $m = 144$ meets the lowest criteria to support our analysis, $\tau = 2$ may fail in some situations, which can be avoided by these two parameters. More specifically, with regard to the epoch length m, from Condition (C3) we get an upper bound for ℓ of about 640 seconds, which would not be desirable as it exceeds the expected block production interval (600s). We would like that the actual value of ℓ amounts at least to a few blocks' total expected generation time. Thus,setting $m = 432$ (almost 3 days) would yield an ideal epoch length. Regarding the dampening filter τ, it follows from our analysis that it should hold that $\tau \geq 2(1+\delta)\Gamma^2 \approx 7.8$. Thus, $\tau = 8$ would be a suitable value.

ASERT vis-à-vis SMA. The above evaluation of the SMA function reveals two shortcomings. Under some circumstances, letting $\lambda = 1.201$ may fail to bound the target fluctuation during a sliding window. Moreover, we can see that parameters in Condition (C4) cannot be increased, i.e., according to our analysis the protocol cannot be secure under a respecting environment with party fluctuation ratio $\Gamma > 1.4$; if we consider a larger value for λ, then the party fluctuation it can tolerate is even smaller.

Compared with the ASERT function (notably, ASERT can work well when $\Gamma = 1.88$), SMA performs considerably worse when party fluctuation rate is relatively high. Thus, we conclude that ASERT is a better choice as a target recalculation function for blockchains with lesser hashing power.

REFERENCES

[1] Lear Bahack. 2013. Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft). Cryptology ePrint Archive, Report 2013/868. (2013). https://eprint.iacr.org/2013/868.

[2] C. Decker and R. Wattenhofer. 2013. Information propagation in the Bitcoin network. In *IEEE P2P 2013 Proceedings*. 1–10.

[3] Cynthia Dwork and Moni Naor. 1992. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology — CRYPTO' 92*, Ernest F. Brickell (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 139–147.

[4] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 281–310.

[5] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In *Advances in Cryptology – CRYPTO 2017*, Jonathan Katz and Hovav Shacham (Eds.). Springer International Publishing, Cham, 291–323.

[6] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2020. Full Analysis of Nakamoto Consensus in Bounded-Delay Networks. Cryptology ePrint Archive, Report 2020/277. (2020). https://eprint.iacr.org/2020/277.

[7] Juan A. Garay and Yu Shen. 2021. On Bitcoin Cash's Target Recalculation Functions. Cryptology ePrint Archive, Report 2021/143. (2021). https://ia.cr/2021/143.

[8] Aggelos Kiayias and Giorgos Panagiotakos. 2015. Speed-Security Tradeoffs in Blockchain Protocols. Cryptology ePrint Archive, Report 2015/1019. (2015). https://eprint.iacr.org/2015/1019.

[9] Colin McDiarmid. 1998. Probabilistic Methods for Algorithmic Discrete Mathematicss, chapter Concentration, pages 195–248. Springer Berlin Heidelberg, Berlin, Heidelberg. (1998).

[10] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.

[11] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. (2009). http://www.bitcoin.org/bitcoin.pdf.

[12] Satoshi Nakamoto. 2009. Bitcoin open source implementation of p2p currency. (Feb. 2009). http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source.

[13] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Advances in Cryptology – EUROCRYPT 2017*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer International Publishing, Cham, 643–673.

[14] G. S. WATSON. 1955. Serial correlation in regression analysis I. *Biometrika* 42, 3-4 (1955), 327–341.

[15] Sam M. Werner, Dragos I. Ilie, Iain Stewart, and William J. Knottenbelt. 2020. Unstable Throughput: When the Difficulty Algorithm Breaks. (2020). arXiv:arXiv:2006.03044

A SOME MATHEMATICAL FACTS

Theorem A.1 (Chernoff bound). *Let $X_i, \ldots, X_T$ be independent random variables with $\mathrm{E}[X_i] = p_i$ and $X_i \in [0,1]$. Let $X = \sum_{i=1}^{T}$ and $\mu = \sum_{i=1}^{T} p_i = \mathrm{E}[X]$. Then, for all $\Lambda > 0$,*

$$\Pr[X \geq (1+\Lambda)\mu] \leq \exp(-\frac{\Lambda^2}{2+\Lambda}\mu),$$
$$\Pr[X \leq (1-\Lambda)\mu] \leq \exp(-\frac{\Lambda^2}{2+\Lambda}\mu).$$

Definition A.2. [10, Chapter 12] A sequence of random variables $X_0, X_1, \ldots$ is a martingale with respect to sequence $Y_0, Y_1, \ldots$, if, for all $n \geq 0$, (1)X_n is a function of $Y_0, \ldots, Y_n$, (2)$\mathbb{E}[|X_n|] < \infty$, and (3) $\mathbb{E}[X_{n+1}|Y_0, \ldots, Y_n] = X_n$.

Theorem A.3. [9, Theorem 3.15] *Let $X_0, X_1, \ldots$ be a martingale with respect to the sequence $Y_0, Y_1, \ldots$. For $n \geq 0$, let $V = \sum_{i=1}^{n} \mathrm{var}(X_i - X_{i-1}|Y_0, \ldots, Y_{i-1})$ and $b = \max_{1 \leq i \leq n} \sup(X_i - X_{i-1}|Y_0, \ldots, Y_{i-1})$, where* sup *is taken over all possible assignments to $Y_0, \ldots, Y_{i-1}$. Then, for any $t, v \geq 0$,*

$$\Pr[(X_n \geq X_0 + t) \wedge (V \leq v)] \leq \exp\Big\{-\frac{t^2}{2v + 2bt/3}\Big\}.$$

Fact 2. *Suppose that $x_1, x_2, \ldots, x_n$ are positive real numbers. Then,*

$$\frac{x_1 + x_2 + \cdots + x_n}{n} \geq \frac{n}{1/x_1 + 1/x_2 + \cdots + 1/x_n}.$$

Theorem A.4 (Cassel's inequality). [14] *Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ be two positive n-tuples with $0 < m \leq \frac{a_k}{b_k} \leq M < \infty$ for each $k \in \{1, \ldots, n\}$ and constants m, M. Then,*

$$(\sum_{k=1}^{n} w_k a_k^2)(\sum_{k=1}^{n} w_k b_k^2) \leq \frac{(M+m)^2}{4mM} \cdot (\sum_{k=1}^{n} w_k a_k b_k)^2.$$

Theorem A.5. *Suppose that $x_1, x_2, \ldots, x_n$ are positive real numbers. If $\max_{i \in [n]} x_i < \gamma \cdot \min_{i \in [n]} x_i$, then*

$$\frac{x_1 + x_2 + \cdots + x_n}{n} \leq \frac{(\gamma+1)^2}{4\gamma} \cdot \frac{n}{1/x_1 + 1/x_2 + \cdots + 1/x_n}.$$