Elastic Project: The Decentralized Supercomputer

The Community, Revision 04

ABSTRACT

Elastic Coin provides the infrastructure for a decentralized and distributed computation of arbitrary tasks over the internet. In this context, Elastic Coin is built on-top of a crypto currency and provides a market-based mechanism to buy and sell computational resources. Buyers, those who need computational resources, model their problem using Elastic Coin's software development kit and broadcast it, along with a certain amount of ELC coins, to the network. The so-called miners are then motivated to offer their computational resources in exchange for a portion of those ELC coins. The size of this portion depends on the amount of work a miner has contributed in relation to the rest of the network. Using ELC as the driving force, Elastic Coin offers potential buyers a large parallel computation cluster composed of many CPUs, GPUs, FPGAs and other devices supplied by the miners. All at a fair and market-driven price.

1. INTRODUCTION

The number of internet users has increased more than tenfold from 1999 to 2016. With an increasing number of users, the number of devices—each equipped with a processor—that is connected to the Internet increases as well. As most of these users are within the scope of non-techincal and normal user activity, it can be assumed that, at any given moment, most of these devices are idle. It is not a new idea to bundle these idle processors and distribute the computational resources among various problems that require large computational power [1, 2]. In fact, this movement has enabled previously infeasible research to be accomplished. Furthermore, with the growing adoption of crypto currencies such as Bitcoin [8], where solving puzzles that are intentionally designed to be resource-intensive is rewarded with a certain number of coins, *supercomputing* becomes more and more relevant to the regular user.

Because each of these devices is owned by a different entity with different motivations and goals, there is no a-priori motivation for cooperation. Current approaches such as [1, 2] rely on the enthusiasm that users may develop for one or more interesting projects and so voluntarily share their computational resources with them. However, this scheme does not work well (efficently) in general. More precisely, there must be some kind of motivation for the ordinary user to contribute their computational resources to a global supercomputer that solves arbitrary tasks that do not necessarily pursue their own particular interests.

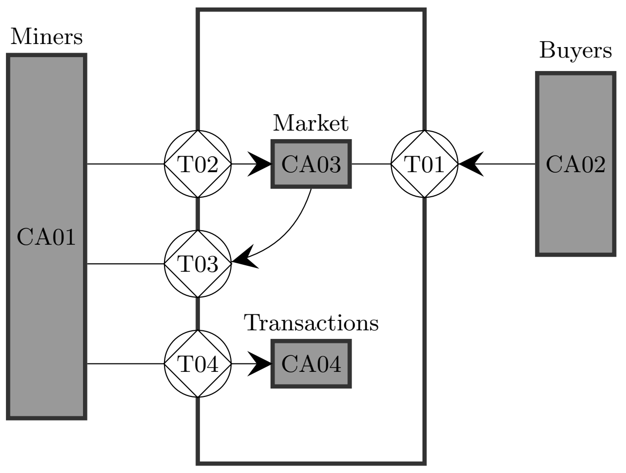
In this paper, we suggest a system that is built around a crypto currency termed *Elastic Coin* and addresses these difficulties. More precisely, Elastic Coin constitutes a market that matches buyers and sellers of computational resources according to economic criteria. Buyers, in this context, are those who demand computational resources in order to solve some arbitrary and computationally intensive task. They submit their task to the network and attach a certain amount of ELC—that is the currency of Elastic Coin—to it. Then, sellers, in the remainder of the paper termed *miners*, are then motivated to offer their computational resources for solving these tasks in exchange for a portion of the attached ELC. The size of this portion depends on the amount of work a miner has contributed in relation to the rest of the network. All this happens "behind the scenes" in a process termed *mining* without the requirement of manual intervention.

The remainder of this paper is structured as follows: Section 2 presents the general idea of a decentralized and market-based infrastructure for decentralized and distributed computation of arbitrary tasks over the internet. Section 3 and Section 4 discuss how these ideas can be embedded into the context of crypto currencies where security and reliability require special attention. Section 5 concludes the paper.

2. DESIGN

2.1 The Big Picture

The general flow is sketched in Figure 2.1. Elastic is a network which targets scientists and others who are in need of large amounts of computational power. These entities, we call them *buyers*, program their desired arbitrary task $T(x_1, \dots, x_k)$—mostly this will be a program or algorithm that pursues a certain goal such as mining blocks for a different coin, finding prime numbers, or anything else—and publish it to the network. In this context they pay three amounts: transaction fee F which goes to the miner who finds a block (see below), a reward fee $R = x \cdot p$ which is composed of the number of units of computation x purchased and a price per unit of computation p and a bounty



TID	Transaction Description
T01	Submits work to market to be solved by the network
T02	Proof-of-work issued periodically by miners
T03	Rewarding the miners proportionally to the work done
T04	Regular transaction (such as sending ELC)

Figure 1: Global ATD

fee B which is rewarded to the miners once they find a relevant solution. In this context we have to differ between rewards and bounties. Rewards are paid out to the miners periodically when they provide so-called proof-of-works and thereby show that they have invested computational resources. Bounties are paid out to those who provide an input $x_1, \ldots, x_k$ to T so that its output $[y_1, \ldots, y_l] = T(xi, \ldots, x_k)$ is in some way relevant to the buyer (not to the network). The relevance can be described by a hook function, a function that is provided by the buyer and takes $[y_1, \ldots, y_l]$ as an input and returns true if the output is relevant, and false in all other cases. Tasks are then worked off by the miners. In this case, the tasks with the highest per unit computation price p are worked off first. Miners work on the task by periodically providing proof-of-works and so proving the amount of computational power they invest. For every proof-of-work they are credited p coint. Once all R coins are given away to the miners(that is after x proof of works), the task is removed and no longer worked on. If a miner finds an input $x_1, \ldots, x_k$ to T so that its output $[y_1, \ldots, y_l] = T(x_1, \ldots, x_k)$ is in some way relevant to the buyer, he gets credited the bounty B. If this happens before all R coins are distributed, the task remains active. If all coins R are distributed before the bounty B has been given away, it gets returned to the buyer. Blocks themselves are generated using proof-of-stake. This is neccessary to disconnect the block generation from the arbitrary tasks that are provided externally. The reasons are covered in the remainder of this paper.

2.2 Proof-of-Work

A proof-of-work function is a *challenge* which is fairly difficult to calculate but easy for others to verify. This ensures that everyone can quickly verify that a particular entity has invested a certain (arbitrarily large) amount of work into obtaining the proof-of-work. One of the first use-cases for proof-of-work was preventing email spam [3]. The central idea was to use *Hashcash* and require a certain amount of proof-of-work by hashing the email's content for every mail that is sent. While the cost is reasonably low on a per-email basis it grows significantly for a larger amount of emails. Mass spam mailers have a hard time providing the amount of proof-of-work required due to limited computational resources. The idea behind proof-of-work, particularily the idea behind Hashcash, has been adopted by Bitcoin: in order to generate a block that is accepted by the rest of the network, miners must complete a proof-of-work which covers all of the data in the block. The difficulty of the proof-of-work is dynamically adjusted to the entire network's calculation power so that—on average—it takes around 10 minutes until one of the network participants finds a block. This process is termed *mining.*

2.3 Mining and the Faster Algorithm Attack

The security of Bitcoin relies on the distributed consensus achieved by a process termed *mining.* In the original Bitcoin scheme, *miners* essentially are users who spend a large amount of computational resources to solve a certain type of cryptographic puzzle [8]. The solutions to these cryptographic puzzles (this in fact is the proof-of-work, cf. Section 2.2) allow for the generation of so-called *blocks.* Users, in this context, are encouraged to mine blocks by rewarding each block with a payment of currently 25 BTC. Blocks are ordered in a linked list and both verify the correctness of bitcoin transactions and form a consensus in terms of a history of transactions that all participants agree upon. In Bitcoin this history is termed *Blockchain* and aims for preventing fraudulent behavior such as double spending a coin multiple times [6] or charging back already done transactions. More precisely, this history is defined to be the *longest chain* of blocks from the genesis block (that is, the first block) to the current block. Of course, it may happen here that multiple side chains emerge from one block of which the longest eventually survives reverting the transactions included in the shorter. In order to be on the safe sides in terms of random reorganizations, users are suggested to wait until the transaction has been added to a block and z blocks have been linked to it [8]. That means, the transaction has received $z + 1$ confirmations. The only way for an attacker to remove one of his transactions from the longest chain in order to take back money he recently spent is to create an alternative longest chain himself that does not contain the particular transaction. In this context the attacker would start calculating an alternative chain from a point where he still owned the spent coins (in this case more than $z+1$ behind the current block). In order to be successful, he must (on average) calculate blocks faster than the rest of the network working on the original longest chain. More formally, the probability of an attacker catching up is analogous to the Gambler's Ruin problem [5, 8]. That is, let q be the probability the attacker finds the next block and p denote the probability that the rest of the network finds the next block.

$$q_z = \begin{cases} 1 & \text{for } p < q \\ (\frac{q}{p})^z & \text{for } p \geq q \end{cases}$$

denotes the probability that an attacker can catch up building an alternative chain from z blocks behind. As the probability of finding a block increases linearly with the calculation power invested in solving the cryptographic puzzles, it can be assumed that an attacker can successfully perform this attack (in literature referred to as the 51 % attack [4, 9]) once he controls more than 50 % of the entire network's calculation power. Putting aside large mining pools, it is not economical from an attacker's point of view to acquire more than 50 % of the networks calculation power. In this paper we suggest a crypto currency-driven market for computational resources. As described in Section 1 the key idea is to working towards solving arbitrary computationally expensive tasks and in exchange get rewarded with a certain amount of ELC. This scheme is very similar to the Bitcoin mining process. An obvious and appealing idea is to utilize the immense amount of work put into solving these arbitrary tasks and use it to form the consensus, i.e., ensure the security of the blockchain, itself. In this particular case, however, the 51 % attack could be pulled off by an attacker with considerably less resources than required in the case of Bitcoin. This attack has been identified and thoroughly discussed by members of the Elastic Coin community and called *The Faster Algorithm Attack* (FAA). Without loss of generality, imagine an attacker hand crafts an algorithm with a complexity of $\mathcal{O}(P)$ with $P \in \Omega(N)$ bounded below by N asymptotically. Furthermore, assume he knows an alternative algorithm which produces the same results but can be run in $\mathcal{O}(Z)$ with $Z \in \mathrm{o}(N)$ being dominated by N asymptotically. When such an algorithm is submitted to the Elastic Coin network to be solved by the miners, the attacker needs significantly less resources to have the equivalent of 50 % of the networks calculation power. More precisely, he needs only $O(\frac{P}{Z})$ of the network's calculation power. To give a concrete example: imagine an attacker submits an algorithm with a complexity of $\mathcal{O}(2^N)$ while at the same time he knows an algorithm producing the same output with a complexity of $O(1)$. He then only needs $O(\frac{1}{2^N})$ of the entire network's calculation power to successfully perform the 51 % attack.

The only incentive to perform this attack is to pull of the 51 % attack. We eliminate this incentive by entirely disconnecting the calculation of arbitrary tasks from the security of the blockchain. In this context we suggest a pure proof-of-stake scheme [7] for the block generation. The proof-of-work scheme is used only for the measurement of who has contributed how much work the payments of the proportional rewards. Performing the FAA as it is described above would correspond to an attacker which can solve *his own* work quicker than everyone else and so get paid his own money (subtracted by a small fee to prevent DDOS in this context). This is not dangerous from the network's point of view. In order to maintain a certain degree of incentive to generate proof-of-stake blocks instead of just focusing on the calculation of proof-of-works we have decided to credit the proof-of-stake block with all transaction fees that are included in that particular block.

2.4 Reliably Measuring Proof-of-Work For Arbitrary Functions

As already pointed out in Section 2.1 our goal is to reward *miners* for contributing their computational resources for calculating arbitrary tasks that have been provided by other Elastic network users. In this context it is wished for that the users get rewarded a certain amount of Elastic Coins which is proportional to the amount of resources they have contributed in comparison to the rest of the network. For this to work, it is required to have a reliable and tamper-proof way to measure (and verify) the work that has been put into calculating such task. Additionally, as it can be assumed that the termination of a piece of arbitrary code can neither be predicted nor guaranteed, we need a fine-granulated proof-of-work feedback to avoid a too high variance. That is, miners shall be allowed to periodically submit proof-of-works *while* the arbtrary task is being executed. One can think of this as sub-dividing the large arbitrary task into many sub-tasks of fixed size where the output of subtask S_k costitutes the input to the subtask S_{k+1}.

Before we take a look at the desired fine granularity by dividing the task into subtasks, let us focus on the calculation of the whole arbitrary task (or function) first. As pointed out in Section 2.2, the general idea is to allow a computationally weak entity (in our case this relates to every non-miner in the network) to outsource the computation of an arbitrary task $T(x_1, \ldots, x_k)$ to a worker with significantly higher amounts of computational power (in our case, that is the miners). The workers (here, miners) then return the output of the arbitrary task, e.g., $[y_1, \ldots, y_l] = T(x_1, \ldots, x_k)$ along with a proof-of-work that the computation of T was carried out correctly. If this proof-of-work matches certain criteria (such as a certain difficulty) it is then accepted by the network and the worker is rewarded. As we do not necessarily know how long the calculation of T actually takes, it is required that the verification of this proof requires significantly less computational effort or time than computing $T(x_1, \ldots, x_k)$ itself—everything else would expose the Elastic network to a variety of DOS attacks.

As, in our scheme, users are rewarded for contributing computational power, this proof-of-work must be secure. Otherwise, it can be assumed that dishonest users will find ways to return plausible results without performing any actual work. Also we require mid-execution feedbacks, that allow for a more fine-granulated payout of rewards yet. That is, we also want to payout *weaker* miners that not necessarily manage to calculate the arbitrary task until the end (or termination) before others have already consumed all R reward coins. To mitigate this problem we need a way for everyone to constantly and periodically provide proof-of-works for sub-portions of the arbitrary task. We suggest to solve the problem as follows:

Assume we have a task T that takes an arbitrary input $x_1, \ldots, x_k$. Furthermore, we assume are using the full Python language, so, due to the halting problem we cannot determine whether the program will terminate (and if yes, when) or not. Using that task T we want to generate *proof-of-works* on a regular basis, which, if being solved, result in a payment of p coins to the miner. A requirement is that all such *proof-of-works* are on average equally difficult to be created. To achieve that, we introduce a buffer B with length L, that is filled as follows: After each executed statement, the "number" of the currently executed statement as well as the content of all variables that were "touched" (even if they have been assigned with the same value) are collected in a temporary binary string Z. The buffer B gets then appended with T. Filling the buffer basically relates to executing the program. In fact, the buffer is assumed to be filled as a fixed rate as statements which change more variables will also take proportionately longer to execute. Statements like memory operations may cause some extra padding to the buffer as they, more likely, take longer to be executed (so they should fill the buffer more). Padding is a process whose parameters must be evaluated and seeded empirically.

Once the buffer is full, the binary concatenation of buffer B and the initial state at the beginning of a sub-block T_x of task T is fed into a *modified* SHA256 function. A sub-block T_x in this context is the fragment of code between an empty and an entirely filled Buffer. The *initial state* corresponds to the set of variables on the stack (including the function arguments). In fact, for practical reasons, SHA256 works sequentially so this process can be performed meanwhile executing T_x itself. The length of the Buffer implicitly defines the duration of the so-called sub-blocks T_x. If the SHA256 hash meets a target value the miner has found a proof-of-work that is entitled to get a reward. In this context the proof-of-work, as it is submitted to the Elastic network, is composed by the index x of sub-block $T - x$ and the initial state at beginning of a sub-block T_x.

Let us assume that each such sub-block T_x of task T has runtime $\Theta(N)$ with N being the number of operations in it. With the buffer-related overhead which itself is in $\Theta(N)$ an honest *miner* performs work in the order of $\Theta(2 \cdot N) \in O(N)$. This implies that in practive the network is calculating an arbitrary task T with half of the network's theoretically possible computational power. This is a trade-off that we have to accept in favor of a decentralized and reward-based cloud computing engine.

Allowing arbitrary solutions to sub-blocks T_x could allow the execution on the FAA attack. A possible attacker could identify a certain sub-block T_x of which he knows a faster way obtain the buffer which is fed into the SHA256 function. This could allow him for being credited coins without performing any meaningful work. In order to mitigate this problem we do not allow arbitrary values for x. Instead, we suggest to set x pseudo-randomly. This avoids the focus of malicious miners on particularily simple sub-blocks T_x. This alone, however, may lead to the problem, that people start caching their proof-of-work solutions (that match the required target of course) to all possible sub-blocks T_x and run into a race condition when a particular x is set for which many miners have cached proof-of-work solutions waiting in the queue.

To avoid caching and precomputation, we introduce a so-called *state* S with a length of 32 bit. The state is set globally and deterministically for the entire network and is influenced by the blocks that are confirmed. At this point, a fairly short block generation period is advised to have the state changed on a frequent basis. Above, we were talking about a *modified* SHA256 function. The SHA256 function is modified in a way, that the initial state of the SHA256 function are the first 32 bits of the fractional parts of the square roots of the first 8 primes XOR'ed with the 32 bit long state. This inherently reduces the problem caching/precalculation problem to an acceptable level[1] as all cached proof-of-work solutions to an arbitrary T_x are void the moment the state changes, i.e., a new proof-of-stake block is generated. In this context, the cryptographically secure nature of state S can be used directly to influence value x.

[1]The block generation time and the frequency at which proof-of-work solutions are transmitted are parameters which influency this degree

EK: Here we have to think about miners who will only focus on the currently active x instead of the whole program.

Acknowledgement

We would like to express my sincere gratitude to everyone who has contributed to this paper with his encouragement and consistent involvement.

3. REFERENCES

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[3] A. Back et al. Hashcash-a denial of service counter-measure, 2002.

[4] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[5] W. Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.

[6] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[7] S. King and S. Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[8] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[9] M. Vasek, M. Thornton, and T. Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *Financial cryptography and data security*, pages 57–71. Springer, 2014.