Introducing NDcoin

A Cryptocoin-Based Concept for Incentivized, Distributed Nondeterministic Computation

Michael P. Frank[1] and David Mondrus[2]

February 1st, 2014

Revision History

- v0.1 (2/1/14) MPF – Initial revision.
- v0.2 (2/2/14) MPF – Added discussion of commitments to separate solving from block creation, and solve-request burn fees to prevent certain kinds of spam/DOS attacks.
- v0.3 (2/6/14) MPF – Added mining fee items, and discussion of recycling instead of burn.
- v0.4 (2/7/14) MPF – Correcting some typos, clarifying some text.
- v0.5 (2/9/14) MPF – Adding material about harnessing SCIP/zk-SNARK for more power.
- v0.6 (2/10/14) MPF – Added a short section about wasted redundancy and job auctions.
- v0.7 (2/13/14) MPF – Minor edits.

1 Abstract

This document proposes a simple idea for a cryptocurrency platform that can also be used as a trustworthy distributed marketplace for robustly carrying out high-performance computing tasks, such as, in particular, for solving instances of difficult (*e.g.*, NP-complete, or apparently exponentially hard) problem classes that are in the complexity class NP (nondeterministic polynomial time) in massively-parallel fashion, or, more generally, for harnessing the network to efficiently contract out the execution of any desired nondeterministic (ND) algorithms – including, in an extended version, algorithms that do not essentially rely on nondeterminism. Other techniques such as homomorphic encryption and code obfuscation can later be layered on top of this capability to maintain privacy of user data and of remotely-executed user computations, respectively.

2 Some Background

The Bitcoin distributed cryptocurrency system was proposed on Oct. 31st, 2008 by Satoshi Nakamoto, as a workable solution to the double-spending problem of decentralized digital cash systems. Today the Bitcoin currency has over $10 billion market capitalization, and the Bitcoin network has a collective computational power (in terms of bit-operations per second) which is estimated to be 8 times greater than that of the Top 500 supercomputers put together. In Bitcoin, that computing power is applied in a competitive way to automatically arbitrate the decision-making authority of the network. But, what if a large cryptocoin network like Bitcoin's could also do useful computational work for other applications?

The original idea of using a cryptocoin-based system as being simultaneously a trusted platform and an automated marketplace for distributed high-performance computing was suggested by David Mondrus, in a discussion between the authors on Feb. 1st, 2014. We were both partly inspired by some related

[1] Department of Electrical and Computer Engineering, FAMU-FSU College of Engineering. mpf@eng.fsu.edu.
[2] CryptoWerks, Inc. david@cryptowerks.com.

earlier generalized blockchain platform projects such as Ethereum, and by Frank's previously-proposed Nomicoin project (a distributed cryptocoin with an automatically reprogrammable rule-base).

One of the authors (Frank) also had previously done research on automatic market-based distributed computing in the Computer & Information Science and Engineering Department at the University of Florida from 2000-2004 in the O.C.E.A.N. (Open Computation Exchange and Arbitration Network) project. However, that research was conducted before the advent of Bitcoin, so did not yet benefit from the enhancements made possible by its innovative new concepts.

3 Outline of the NDcoin Concept

NDcoin is, in essence, a typical cryptocurrency like Bitcoin (or any of the numerous altcoins), but with certain special extra features. In addition to supporting ordinary currency-sending transactions, NDcoin supports a special extra type of transaction order – namely, a request from a user to execute a given nondeterministic algorithm, specified in a nondeterministic Turing-complete language designed for this purpose. Initially, the user code to be executed will be specified in plaintext; however, later on in the project's development, we intend to explore the use of some form of fully homomorphic encryption and indistinguishability obfuscation for increased privacy of user data and computations.

The user requesting a given nondeterministic computation (call him the "customer") will, as part of his request, specify a bounty (denominated in its native NDcoin currency) to be paid out automatically, upon the successful (accepting) execution of the nondeterministic algorithm before a certain deadline, to whichever node first generates (and broadcasts to the network, for inclusion in the blockchain) a correct and efficiently-checkable proof of successful execution of the nondeterministic algorithm. In the case of nondeterministic computations having short witness strings, this proof can simply be that string.

For longer computations, a technique such as that used in Ben-Sasson *et al.*'s SCIP (Succinct Computational Integrity and Privacy) a.k.a. zk-SNARK system can be used to provide a short, easily-verified probabilistically-checkable proof of correct execution, under certain cryptographic assumptions.

If there are multiple distinct successful executions (*i.e.*, multiple solutions to the ND problem), bounties can be paid for as many unique solutions as the customer has specified are payable.

Until the bounty is paid out for a verified-successful computation, when the deadline expires, the bounty funds are considered to be "on hold," that is, they cannot be otherwise spent by the user. Thus, the cryptocoin network plays the role of a trusted escrow service, which ensures that payment will be tendered if, and only if, computational services are correctly rendered.

It is interesting to note that, since any standard proof-of-work function is, at essence, simply a typical example of a nondeterministic algorithm, NDcoin, if broadly adopted, can even be used for the purpose of contracting out mining (*i.e.*, block-nonce-finding) work for any of the existing cryptocurrencies.

Now, let's delve a little more deeply into the relationship between problem solving in NDcoin, and ordinary transactions and ordinary mining of new NDcoin blocks. Unlike ordinary (Bitcoin-like) "coin-send" transactions, which are already complete when initially ordered, and which only require checking a signature to verify, an NDcoin "solve" transaction is initially *incomplete* as-ordered (when initially broadcast to the network); it must be subsequently annotated with a corresponding solution (acceptable witness string) before its bounty can be paid out. Verifying the solve transaction requires executing the nondeterministic algorithm with the given witness string to confirm that it is acceptable, or (in case direct witness verification would be too slow), by verifying one of SCIP's *succinct certificates*. Incorporation of the solved transaction into the blockchain (before competing solvers do) implies (by consensus convention) that the reward amount is thereby (at the time the deadline expires) transferred from the customer's account to the solver's account. (Of course, part of the verification process for the solve order also involves confirming that the customer's bounty had not already been spent.)

NDcoins will be initially created and distributed in the usual way, by allowing mining nodes to solve a generic proof-of-work problem that is tied to the creation (and commitment to the blockchain) of new blocks. Our (the authors') personal opinion is that adoption of NDcoin will be best if there is no pre-mining, since that is most fair. (That is, the date of launch of the network will be publicly announced well in advance.) Funds supporting the project should be raised in some other way.

Solvers can double as mining nodes, and it is recommended that there be a reasonable balance of computing power dedicated to general problem solving (distribute compute services) vs. mining. This is because nodes need to ensure that their solutions will be successfully incorporated into newly-mined blocks. Solutions cannot be broadcast in plaintext except as part of a new block, since then another node could easily steal the solution and claim the associated bounty. However, a node without mining capabilities can safely broadcast a secure *commitment* to a solution (a kind of cryptographic lockbox) to timestamp it on the blockchain and thereby stake the solver's claim on the bounty, and only later broadcast the actual solution details to unlock the commitment and redeem the claim, and receive the bounty payout at a later time. In such scenarios, bounties should be paid out first to the earliest-posted commitments that were unlocked before the deadline.

Of course, if there is no generic ND solving work that is in demand from customers at a given point in time, or if the available problems are too difficult to solve quickly, blocks may be generated that have no new ND solutions in them at all. (The same is true in Bitcoin where blocks may contain no transactions.)

It is anticipated that NDcoins will be bought and sold on exchanges, just like other cryptocoins.

In general, it is up to individual mining/solver nodes to decide whether or not they wish to attempt to solve a given problem. This decision can be based on the bounty size, problem complexity, and other factors. Thus, the bounty sizes have to be large enough to incentivize mining nodes to do the extra work required to solve problems. The market rates for solving problems of various complexity can also be tracked and published, so that customers will know what bid prices (bounty amounts) to place on their solve orders. The larger the bounty, the more nodes will likely attempt the problem, and the faster it will be solved. This relationship can be tracked and published as well.

The language for describing nondeterministic algorithms can/should be in some simple, straightforward virtual machine model, which can be JIT compiled to efficient code on native platforms. The virtual machine model should be simple enough to be able to be checked thoroughly for security, and should run in a sandbox environment, to protect the host from attacks by malicious user code. When techniques like SCIP/zk-SNARK are used, a machine model suited for creating succinct certificates should be used instead.

To deal with the possibility of infinite loops and memory leaks in the user code, problem instances (ND solve orders) can be annotated with a fixed numerical upper bound on the number of steps and memory required to execute the algorithm. A solver (or verifier) can automatically halt if the bounds are exceeded during execution. Solvers do not have to accept solve orders whose space or time bounds are beyond what they can, or wish to, deal with. Verifiers also do not have to verify solved problems with high verification complexity if that would exceed their capacity – however, it is recommended that nodes do still accept solutions that they don’t wish to verify themselves which have already been verified by other nodes. There is a small risk that an attacker may attempt to claim the bounty for generating blocks containing (faked) “solutions” to problems that require too many resources to actually verify, such that no (or only a few) other full nodes will even attempt to verify them. For this reason, there should perhaps be a global limit on the advertised problem complexity that the network should be able to verify in a trustworthy fashion. (Say, complexity bounds that can be met by at least a majority of verifier nodes.) However, if SCIP/zk-SNARK are used, the associated risks should be greatly diminished, since it should be cheap for all nodes to verify the succinct certificates produced in that system.

4 Advantages of NDcoin

The key advantage of NDcoin over other distributed computing networks is that the cryptocoin network plays the role of a trusted escrow service. Solver nodes can be guaranteed to receive payment for their revealed solutions (as long as they are among the first *N* solutions posted that were revealed before the deadline). And, customers are guaranteed not to have to pay the bounty if no solutions are found. Also, a request fee paid by coin burn or recycling (mentioned later) helps ensure that the network will not get spammed with excessively many impossible (intentionally fraudulent) problem-solve requests.

In NDcoin, as opposed to previous distributed computing services, the architecture of the system provides a cryptographically-strong guarantee that “payment is tendered for services rendered” – i.e., no party to the transaction is able to cheat. All this is accomplished with no trusted centralized parties, except for possibly a party that does certain preprocessing required for SCIP, although this function can possibly be performed by the customer, or else decentralized (done redundantly by each solver node), using shared random-number generation techniques.

Because NDcoin’s security model is essentially fraud-proof, it requires no trust, and so new entities do not have to establish reputations prior to participating in the system. Thus, the barriers to entry for new entities to join and participate in the system are very low. So, the network should be able to grow very quickly and organically. For comparison, the Bitcoin network’s aggregate hashing power has grown at an average rate of 100x per year for the past four years (2010-2013).

5 Data Formats (summary)

In this section we summarize the content of various special transactions/messages that are unique to the NDcoin protocol. This protocol has not yet been extended to support SCIP/zk-SNARK, to more efficiently handle arbitrary nondeterministic computations, but such an extension should be straightforward.

5.1. Format of Solve Request Transactions

Messages that constitute transaction orders that comprise requests to solve specific nondeterministic problem instances will include the following information, at least:

1. The maximum length, in bits, of any witness bit-strings for this specific problem instance; this is used to control the path taken by the nondeterministic algorithm. The typical naïve operation of the solver is to just randomly generate and test different witness candidates until an acceptable solution is found. (If a particular computation path happens to halt before all bits of a given witness string have been used, the remaining bits can be ignored. If it runs off the end of the witness string without halting, it can be terminated.)

2. The specification of the nondeterministic witness-verification algorithm, encoded in a simple, Turing-complete virtual-machine language (but with fixed time and space bounds on any given run). The language should include operations to utilize bits from the witness bit-string in the course of the computation. It should also include an "accept" operation to assert that the witness string is acceptable (i.e., that it is a valid solution to the problem). It can also include operations to return longer results. (Theoretically, a multi-bit result could be found via a sequence of computations with single-bit outputs, but that would be less efficient.)

3. The numerical limits on the number of virtual-machine operations, or amount of memory required by the verification algorithm. If, at any point, these bounds are exceeded, the witness string will be rejected as not acceptable.

4. Burn transaction effectively paying all NDcoin holders (via deflation) a fee for incorporating the incomplete solve request transaction in the blockchain. This is included an anti-spam measure. (A burn is preferred to a miners' fee here to prevent miners from launching spam from within their own blocks.) Another, more sustainable option would be a "recycling" transaction that returns burned coins to a "smelting pool" from which new mining rewards will be generated (as the smelting pool grows, the block reward size is increased accordingly).

5. The size of the bounty to be paid by the customer to the solver who provides the first claim to an acceptable solution. The bounty gets paid when the deadline expires (see item 7 below).

6. The number *N* of different acceptable solutions for which the bounty is actually payable. *E.g.*, N=1 if only the first solution published (or first commitment published and opened before the deadline) is payable; *N*=100 if the customer will pay that same bounty amount for each of up to the first 100 different solutions (solutions with different witness strings) posted.

7. A deadline date/time by which solutions must be incorporated (in plaintext) into the blockchain in order to receive payment.

8. Public key of customer's payment account. The NDcoin balance of that account must be large enough to pay the request burn/recycling fee, as well as all of the remaining possible bounties, or else the transaction will be ignored by the network. The burn fee is debited right away when the solve-request transaction is committed, and the *N* bounties are all kept "on hold" until the deadline expires, at which point they are paid out, in the order of which solutions (or later-opened solution commitments/lockboxes) were posted first.

9. A digital signature for all of the information in the transaction request, signed by the customer's public key (which is also included, in plaintext).

Solve-request transactions are broadcast to the network, and will be included in the blockchain as usual; however, solvers who want to get an early start on the problem don't need to wait for confirmations.

5.2. Format of Solution Transactions

A plaintext solved version of a given solve order includes the above information items 1-9 from the solve request (the inclusion can be by reference, via a hash pointer), plus also:

10. The problem solution, in the form of an actual witness bitstring that is verified to be acceptable by the given ND algorithm, and the associated output of the computation. (In the case of computations using SCIP/zk-SNARK, the solution can include its succinct certificate in place of the full witness string.)

11. Public address of the solver's account which is to receive the transaction bounty.

12. Miners' fee to be paid, out of the bounty, to miners who post this transaction.

All of the items 1-12 in the solution transaction then get hashed together with all of the other transactions in the given block (using the ordinary Merkle tree approach) to determine the block header, which is then also subject to an ordinary proof-of-work (nonce finding) computation before the block is published to the network. Competing nodes can copy the solution after the block is broadcast, but they will not be likely to find a new block before the solver's block gets accepted by the majority of the network. To obtain even stronger assurances that the solution will not be stolen, the solver can optionally instead use a two-stage process to timestamp the solution prior to publicly revealing its contents (see next two sections).

5.3. Format of Solution Lockbox Transactions

A problem solution may be time-stamped on the blockchain prior to actually revealing the solution publicly by posting a version of the transaction that includes, instead of items 10-12 above, the following information instead:

10. A secure commitment consisting of a cryptographic hash (say using SHA-256) of the witness string or succinct certificate. (If the length of this is shorter than 256 bits, it can be padded out with random data to a full 256 bits, to put a lower bound on the difficulty of hash preimage finding.)

11. Public address of the solver's account, which will automatically receive the transaction bounty when the problem deadline expires, if the commitment has been unlocked by then (and the solution is valid, and it is one of the first *N* solutions posted via plaintext or lockbox).

12. A transaction paying an immediate miners' fee, from an account owned by the solver node, to miners who commit this transaction (incorporate it into a block on the main chain). (This prevents the blockchain from getting spammed with empty lockboxes.)

5.4. Format of Lockbox Opening Transactions

When a solver is satisfied that his solution lockbox has been securely incorporated into the blockchain, he can reveal its contents publicly by posting a transaction with a hash reference to the earlier lockbox transaction as specified above, plus.

13. The actual witness string verified to be acceptable by the given ND algorithm, or the succinct certificate of the verification computation.

14. Any additional random data that the witness string or succinct certificate had been padded with before hashing to ensure the security of the commitment (in case the witness string was short).

6 Potential Flaws, Vulnerabilities, Attacks, and Weaknesses

This section discusses some potential security weaknesses and how we are resolving them.

6.1 DOS attack

Without the transaction burn fee (item 4 in sec. 5.1), a type of Denial-Of-Service (DOS) attack would be for the attacker to submit a lot of high-bounty problems that are unsolvable to the network (this would be a form of spam). Competing nodes could waste a lot of time and resources trying to solve these problems to claim the bounty, with no chance of success.

One potential countermeasure would be a machine learning system that learns to identify problems that are likely to be unsolvable. However, there may be ways to foil such a countermeasure.

A simpler countermeasure, which is what we are proposing, would be a request-posting fee (or burn transaction) that is required to be included as part of the solve request. In this scenario, when the request is posted in its initial, incomplete form, it gets incorporated in the blockchain immediately (rather than waiting for a solution first). The fee or burn gets debited right away; the bounty is not paid until a solution is posted.

In our solution, the reason why we propose to use a burn instead of an ordinary mining fee is to prevent spammers from colluding with mining pools to submit impossible solve requests to the network in their own blocks at no cost. The burn transactions could create deflation, but this can be countered easily by simply returning the burned coins to a "coin recycling pool" or “smelt pool” from which future mining rewards can be drawn, so that in the long run, the supply of NDcoins (ignoring lost coins) can still approach a constant as it does in Bitcoin. In such a scenario, we would call the burn transaction a "recycle" transaction instead.

6.2 High-complexity jobs

Earlier we mentioned that some jobs may require a greater amount of nondeterministic space and time complexity than some nodes are willing or able to commit towards solving, or even verifying them. This raises certain security hazards, for example:

If only a small number of nodes have sufficient resources to (or choose to) verify a given solution, the network will not have as strong an assurance that the solution is actually correct. The customer could end up being cheated if an incorrect solution ends up getting accepted by the network because the only nodes that have enough capacity to actually verify it are (for example) in collusion with each other, and most other nodes are just accepting the committed solution without actually verifying it. Customers will need to be aware that high-complexity jobs will be, for this reason, less secure. To facilitate this, the relationship between job complexity and the share of hashing power available for verification purposes should be well publicized.

If SCIP/zk-SNARK can be used, then the above problem becomes much less acute, because re-verifying that a verification computation was successful can be done easily by all nodes. In this scenario, only the node that first publishes a given solution needs to run the full computation to verify it.

6.3 Wasted computational effort

For computations that do not have many nondeterministic solutions, there is implicitly waste involved if many nodes attempt to solve the problem, since only the first submitted solution will be rewarded. To reduce these inefficiencies, it may be preferable if jobs are assigned in a multi-stage process, in which first, nodes bid for the job to do the computational work, and then the bidder that best fulfills the customer’s needs (*e.g.*, in terms of having the lowest bid, or highest advertised performance) is either automatically selected by the network itself, or by the customer machine; and then the computation is performed, and payment automatically tendered (if the computation was completed on time and the terms of the contract are otherwise fulfilled). In general, the network could provide a complete distributed market service. Protocol extensions to support such features are not yet elaborated in the present paper, but they would be straightforward to develop.

7 Conclusion

NDcoin, once developed and released into the wild, could become an extremely effective and popular system for harnessing the vast computational resources of a cryptocurrency network to perform arbitrary useful computations, at least, for problem classes having feasible deterministic or nondeterministic algorithms. Problem instances are competitively solved and thoroughly verified by the network. This idea could have numerous practical applications, since the set of important computational problems that have a nondeterministic aspect is a very broad one, and SCIP/zk-SNARK allows provable fast verification of even deterministic computations; so essentially, at this point, only quantum computation is excluded by the present framework (as described).

Note however, that in this particular document, we are envisioning computations as being space- and time-bounded. This rather limits the type of distributed applications we can address, to just offline computations (as opposed to online, *i.e.*, stream-based processing). In future work, the concept of this system may be able to expanded to also permit non-time-bounded computations, such as online services, and even entire distributed operating system, which would then need to be paid for on a recurring (rather than one-time) basis. For now, how best to accomplish such enhancements needs further thought.